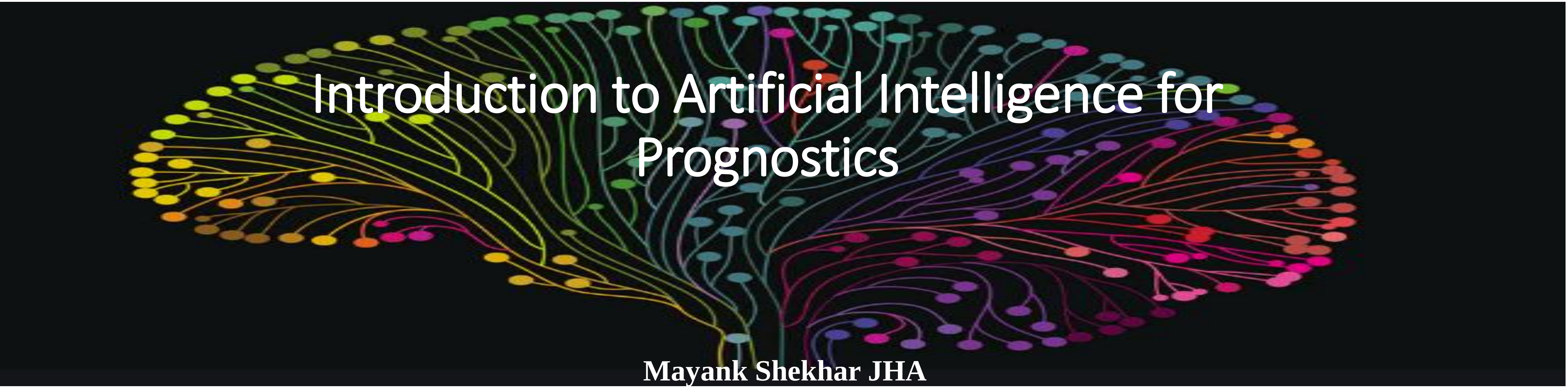


Introduction to Artificial Intelligence for Prognostics

Mayank Shekhar JHA
Associate Professor
(Maitre de Conférences)
Automatic Control, Reliability of Systems
mayank-shekhar.jha{at}univ-lorraine.fr

Research
Centre de Recherche en Automatique de Nancy
(CRAN) UMR 7039,
Faculté des Sciences et Technologies
Boulevard des Aiguillettes - BP 70239 - Bât.
1er cycle 54506 Vandoeuvre-lès-Nancy
Cedex France

Teaching
Polytech Nancy (ESSTIN),
2 Rue Jean Lamour 54509
Vandoeuvre-lès-Nancy,
Cedex France

Email: mayank-shekhar.jha [at] univ-lorraine.fr

Contents

JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

Introduction and Few Reminders

Artificial Intelligence Domains

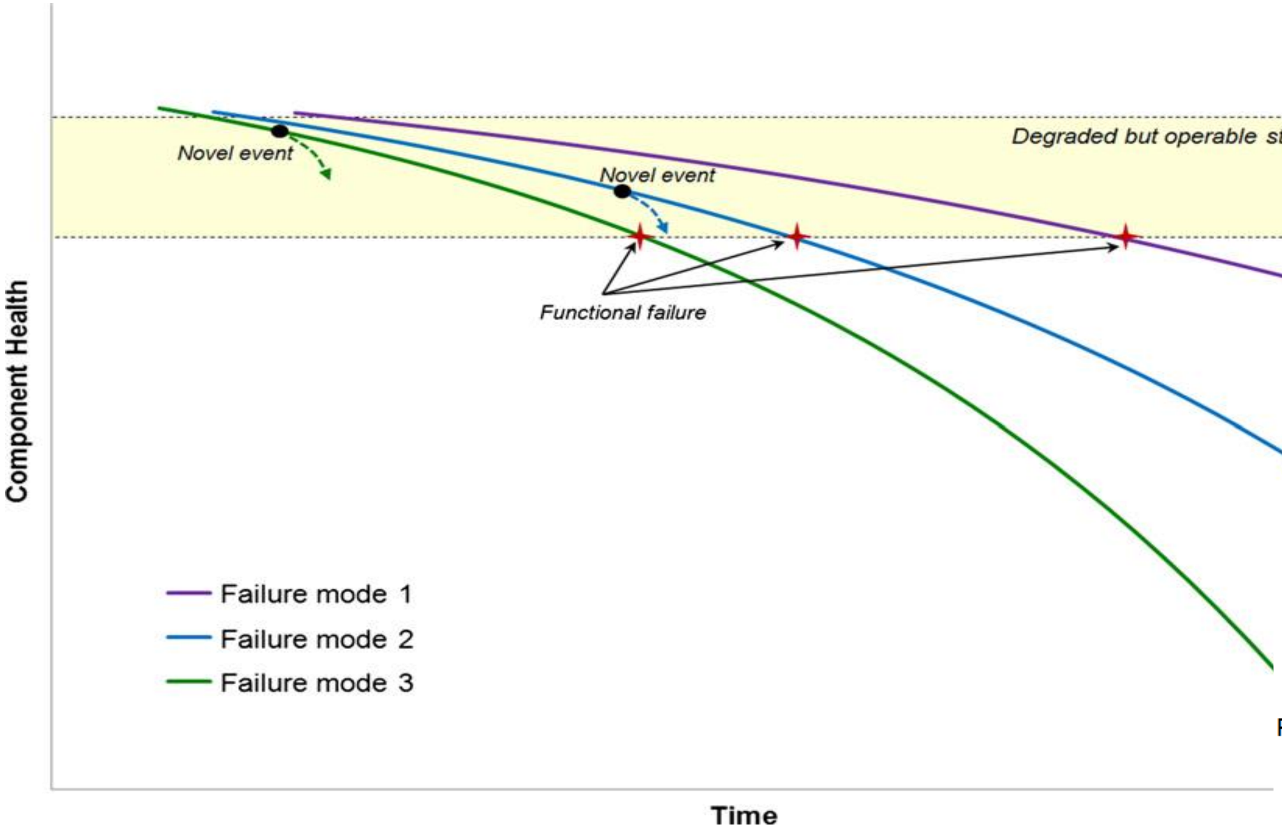
Types of Learning

Linear and Logistic Regression

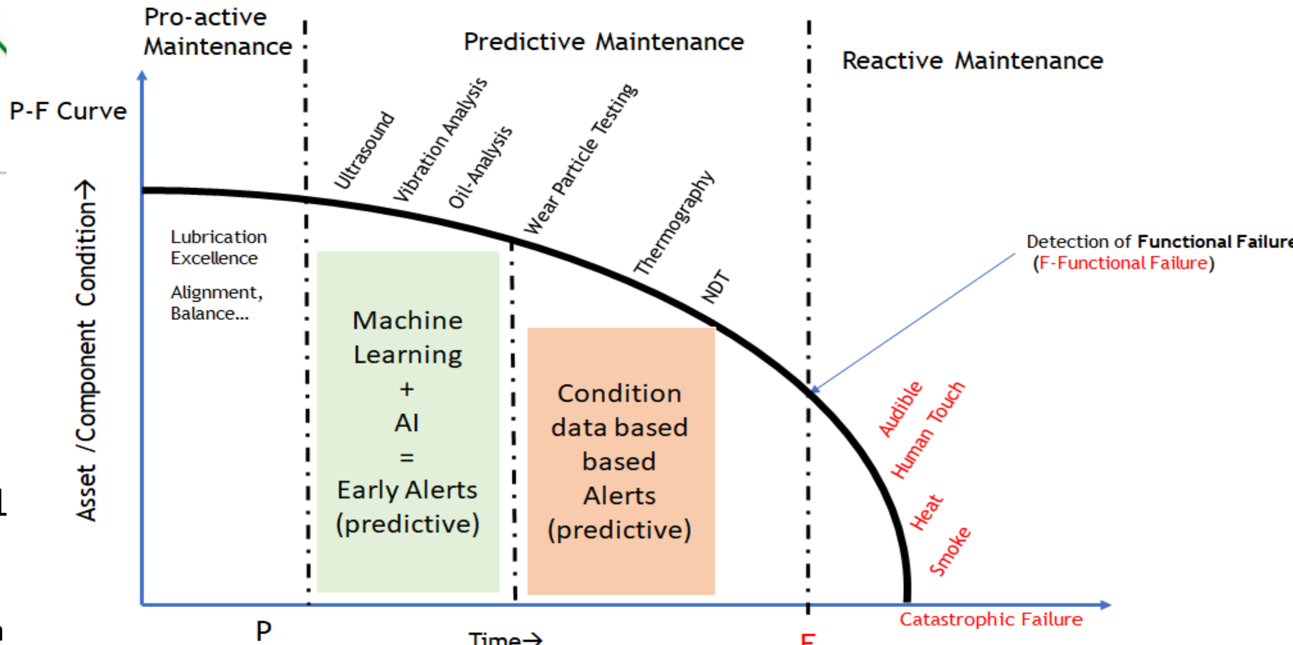
JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

Motivation:

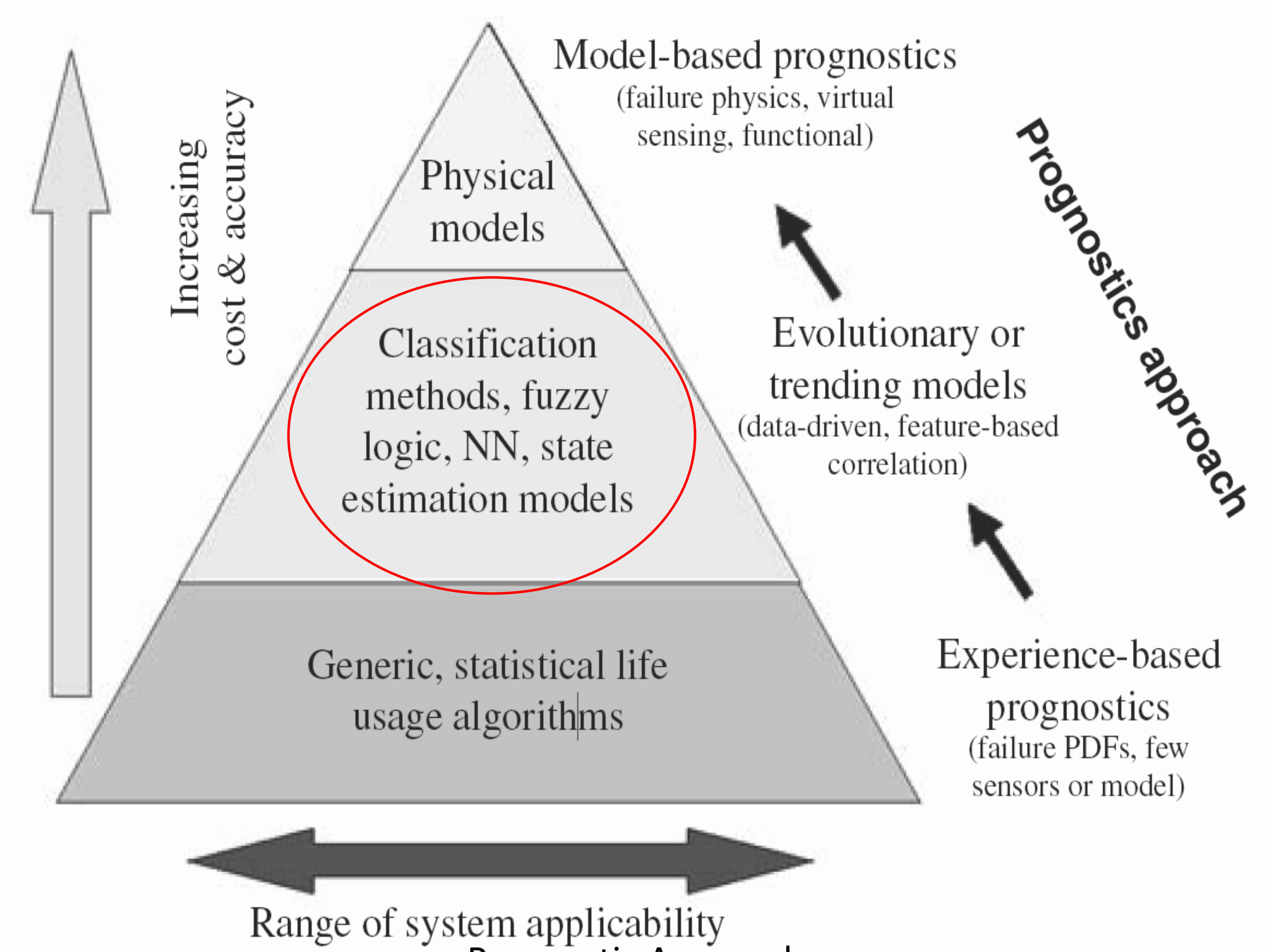
Prognostics (ISO13381-1,2004) : "*the estimation of time to failure and risk for one or more existing and future failure modes*".



Component health degradation curve (Sikorska, Hodkiewicz et al. 2011



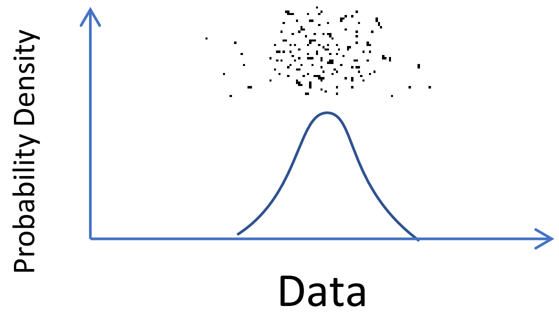
RUL prediction Methods

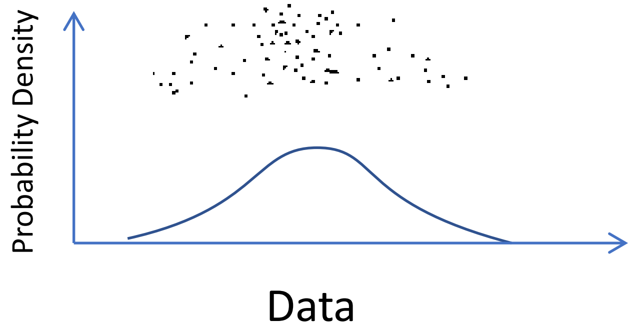


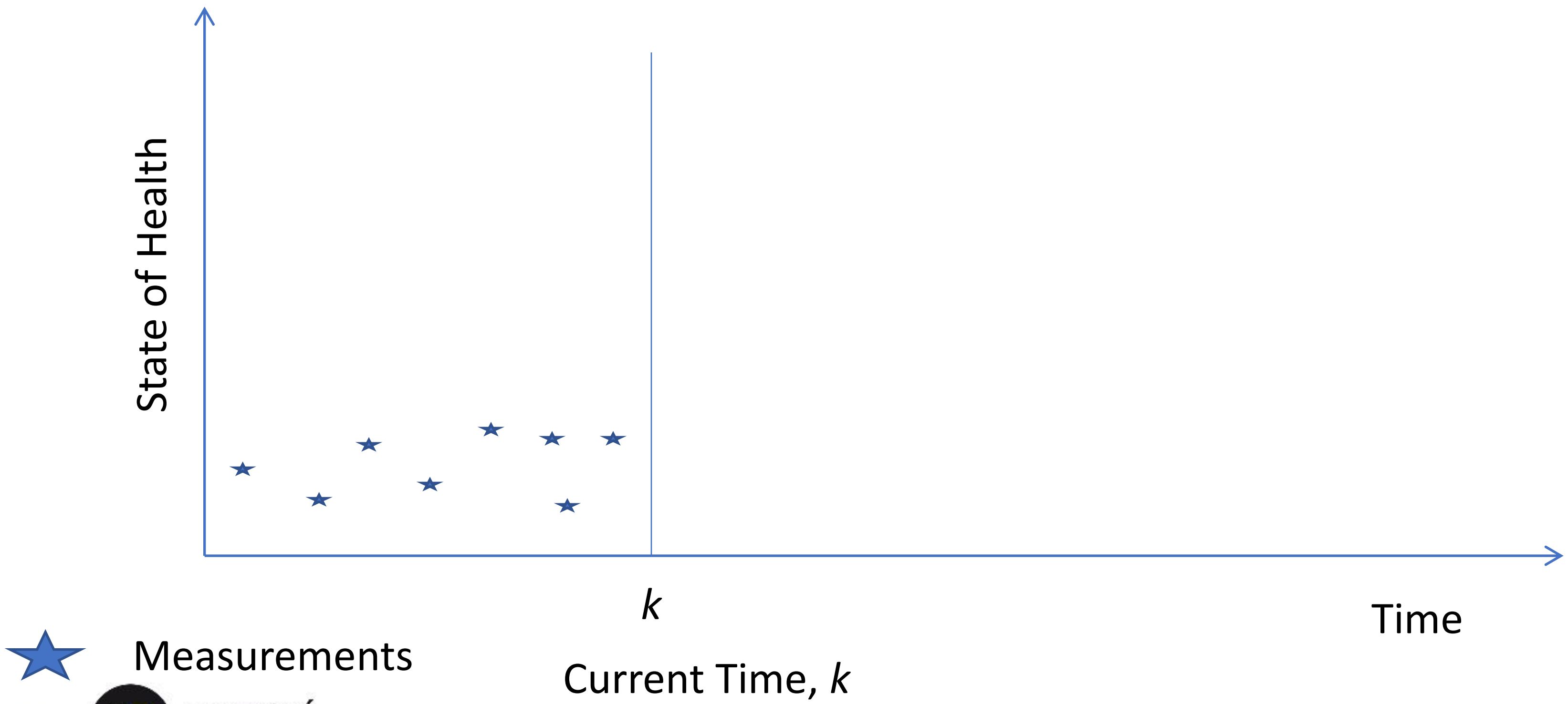
Prognostic Approaches

(Vachtsevanos, Lewis et al. 2007, **ISO13381-1 2004, Liao 2005, Jardine et al. 2006, Lee et al. 2006)**

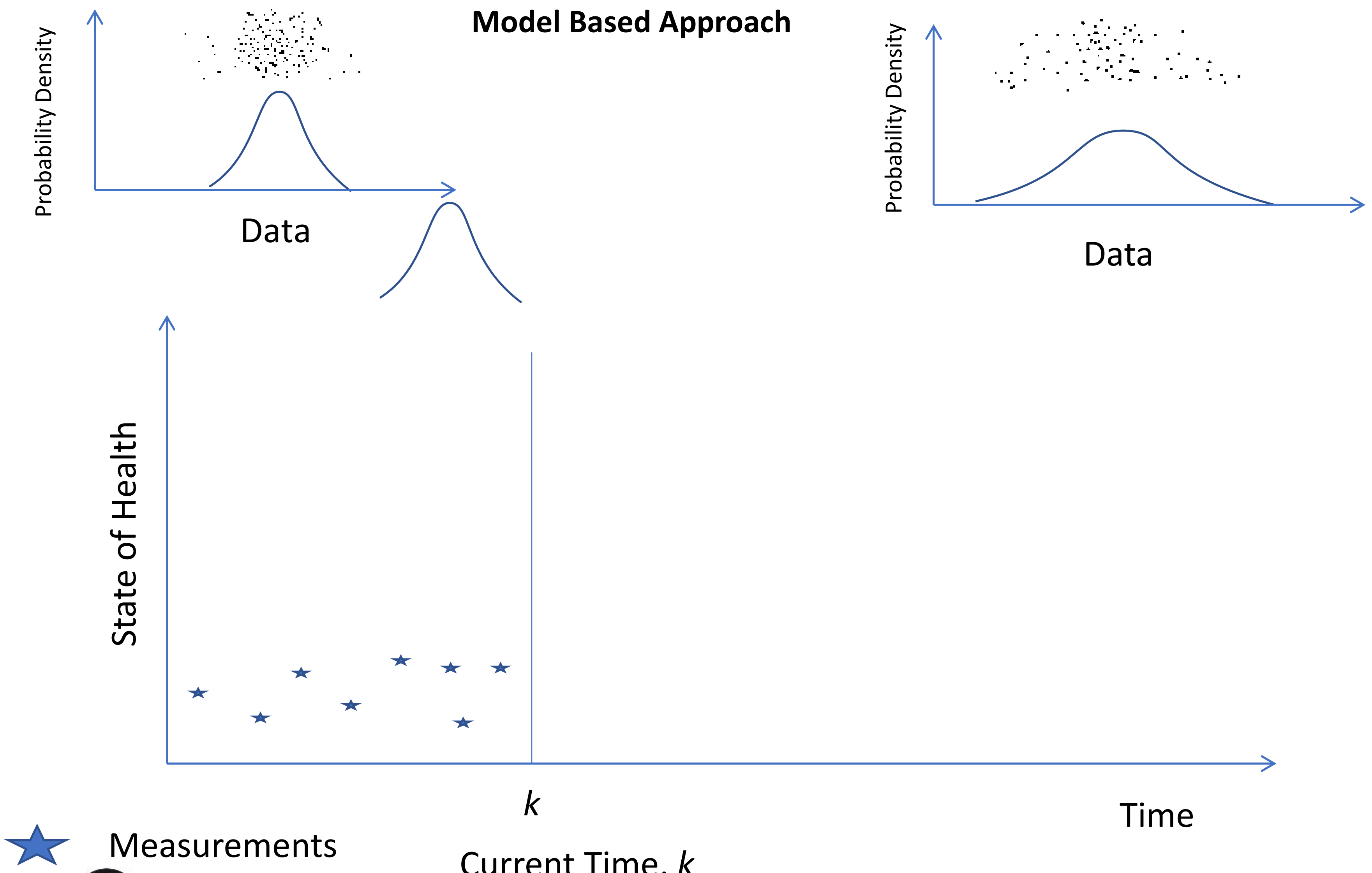
RUL Prediction Illustration



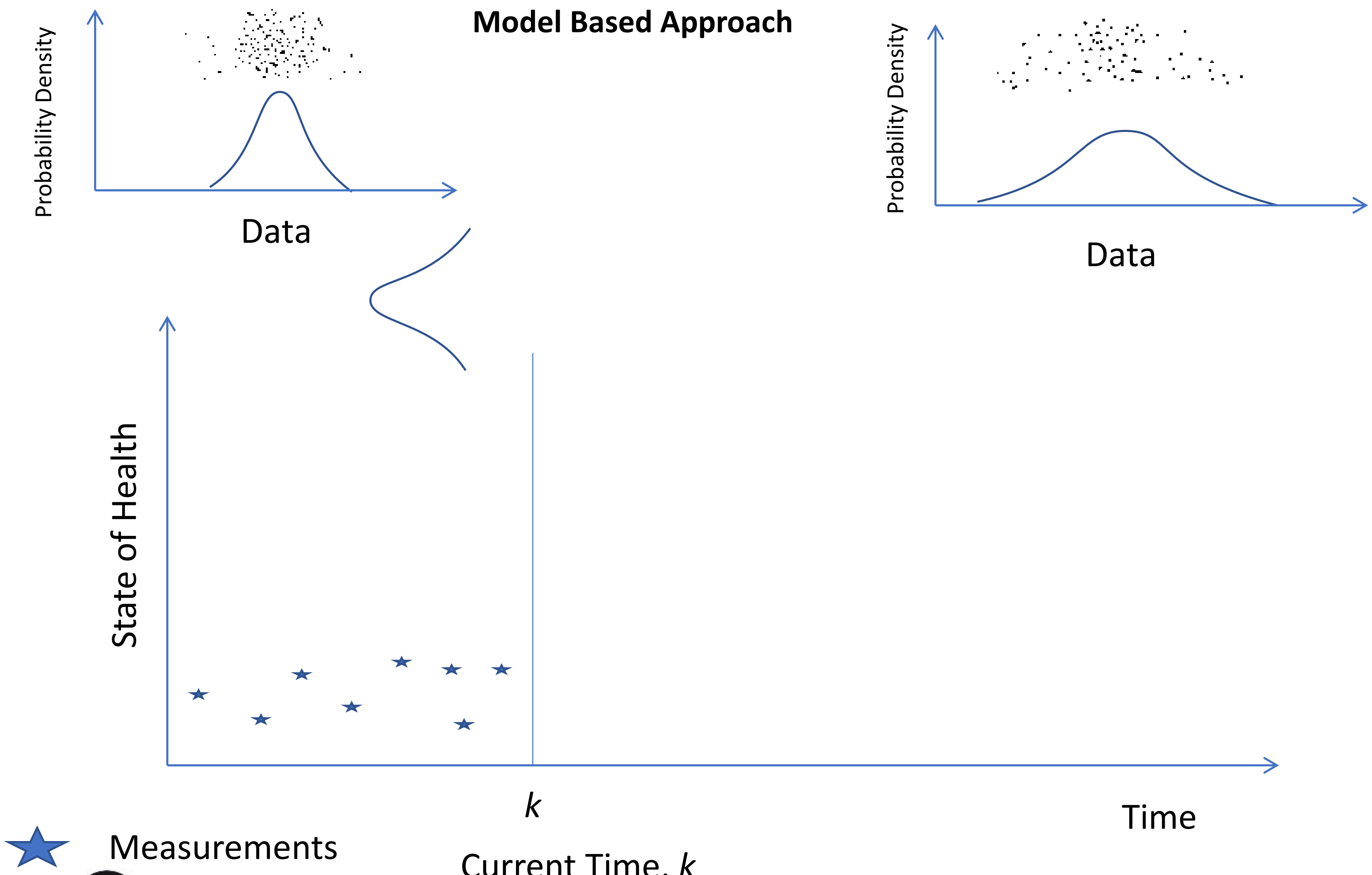




RUL Prediction Illustration

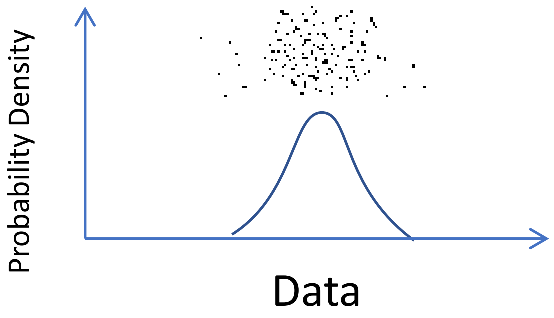


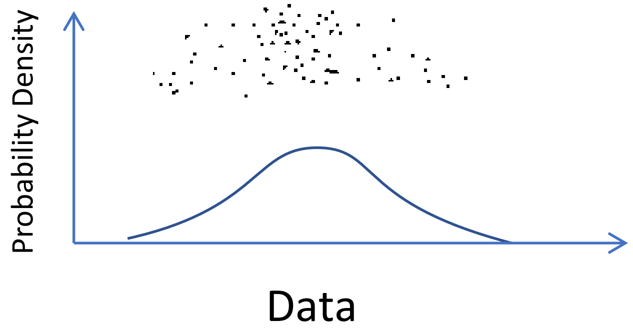
RUL Prediction Illustration

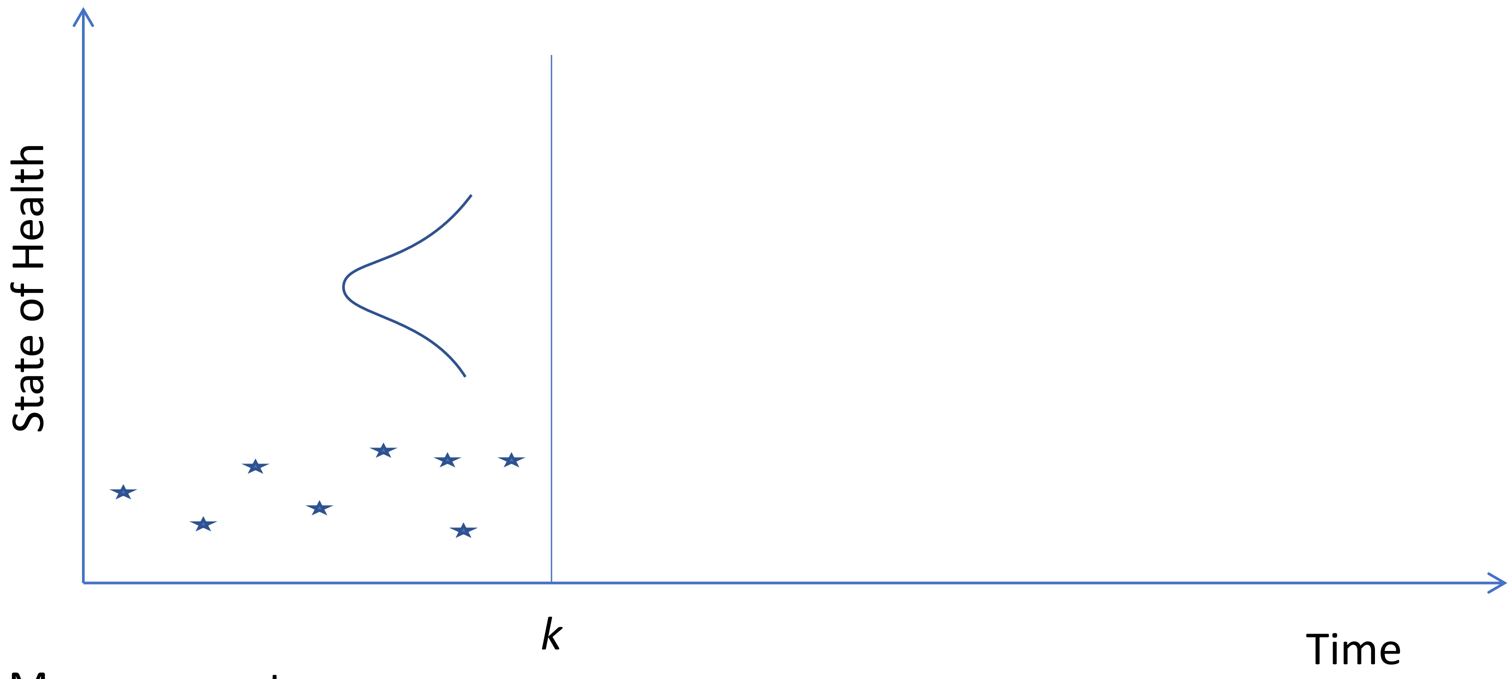


RUL Prediction Illustration

Model Based Approach



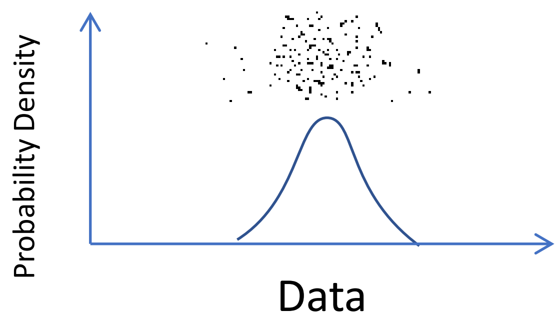




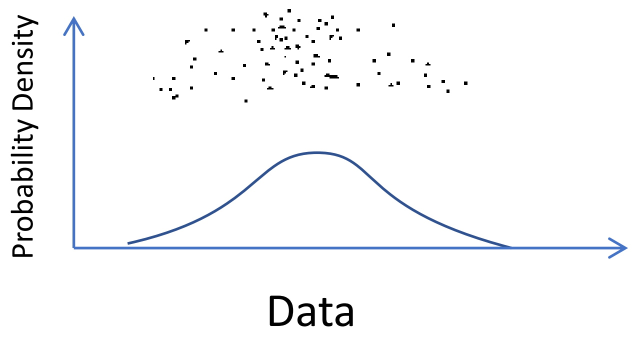
★ Measurements

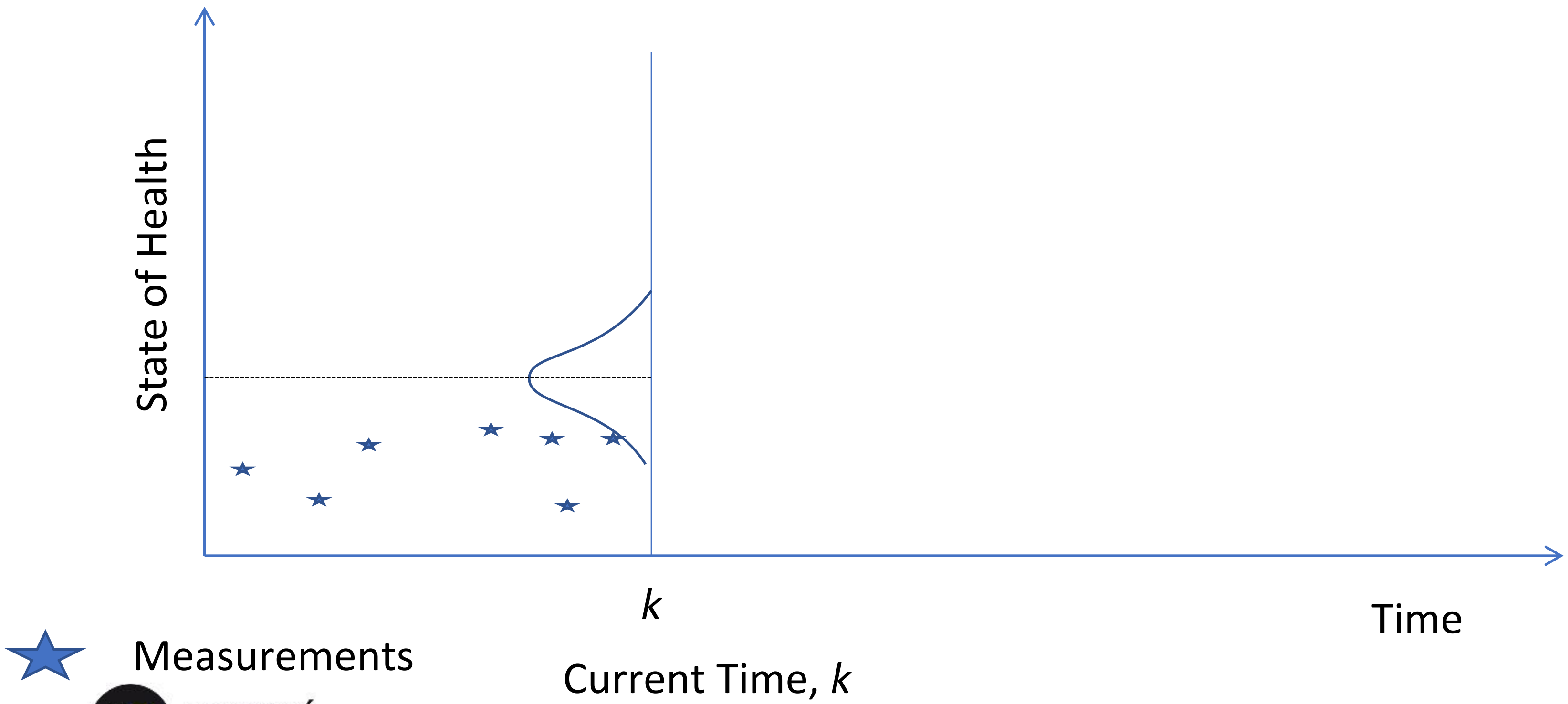
Current Time, *k*

RUL Prediction Illustration



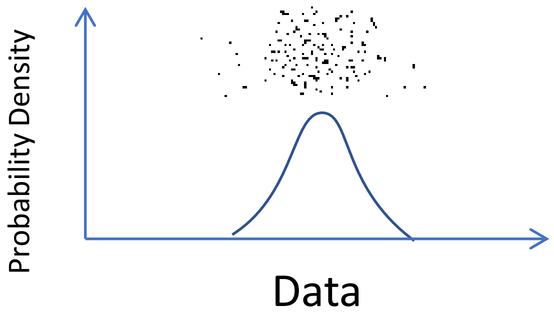
Model Based Approach

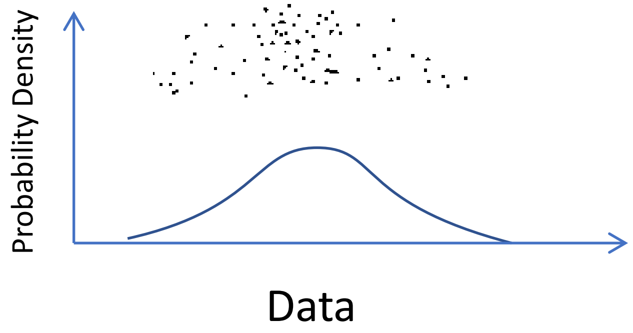


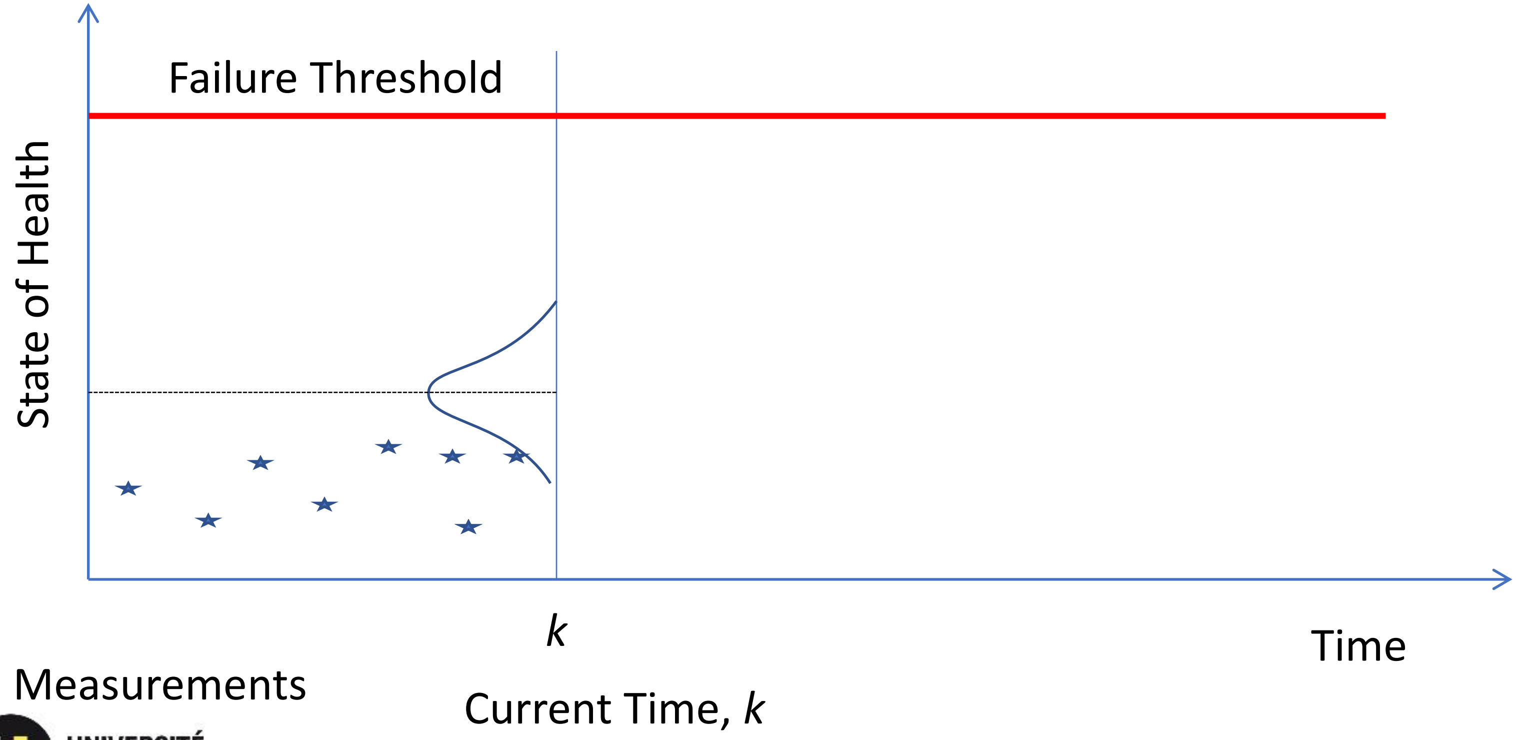


RUL Prediction Illustration

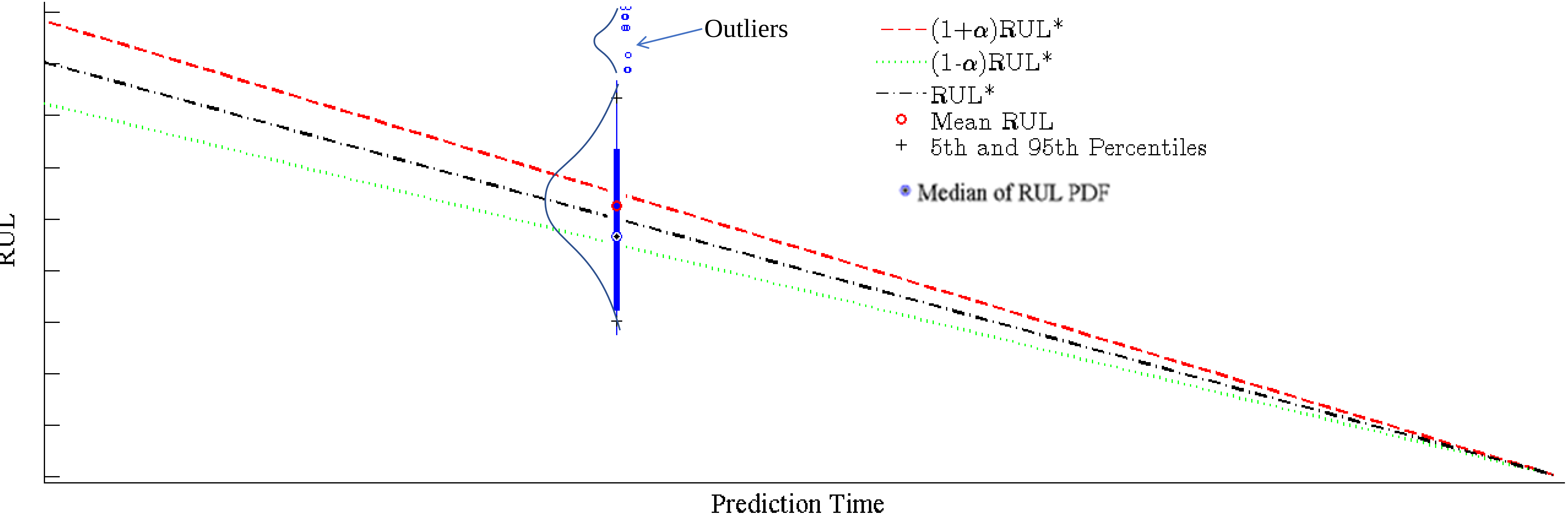
Model Based Approach





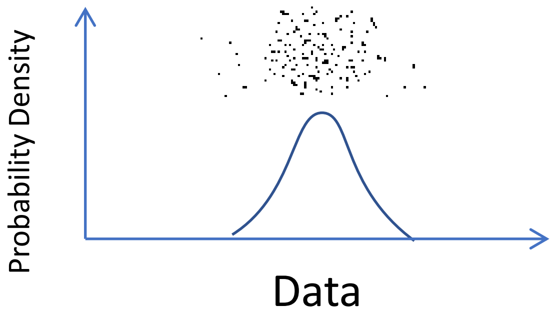


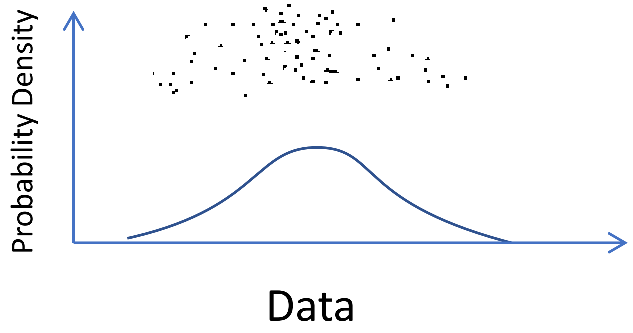
RUL Prediction Illustration

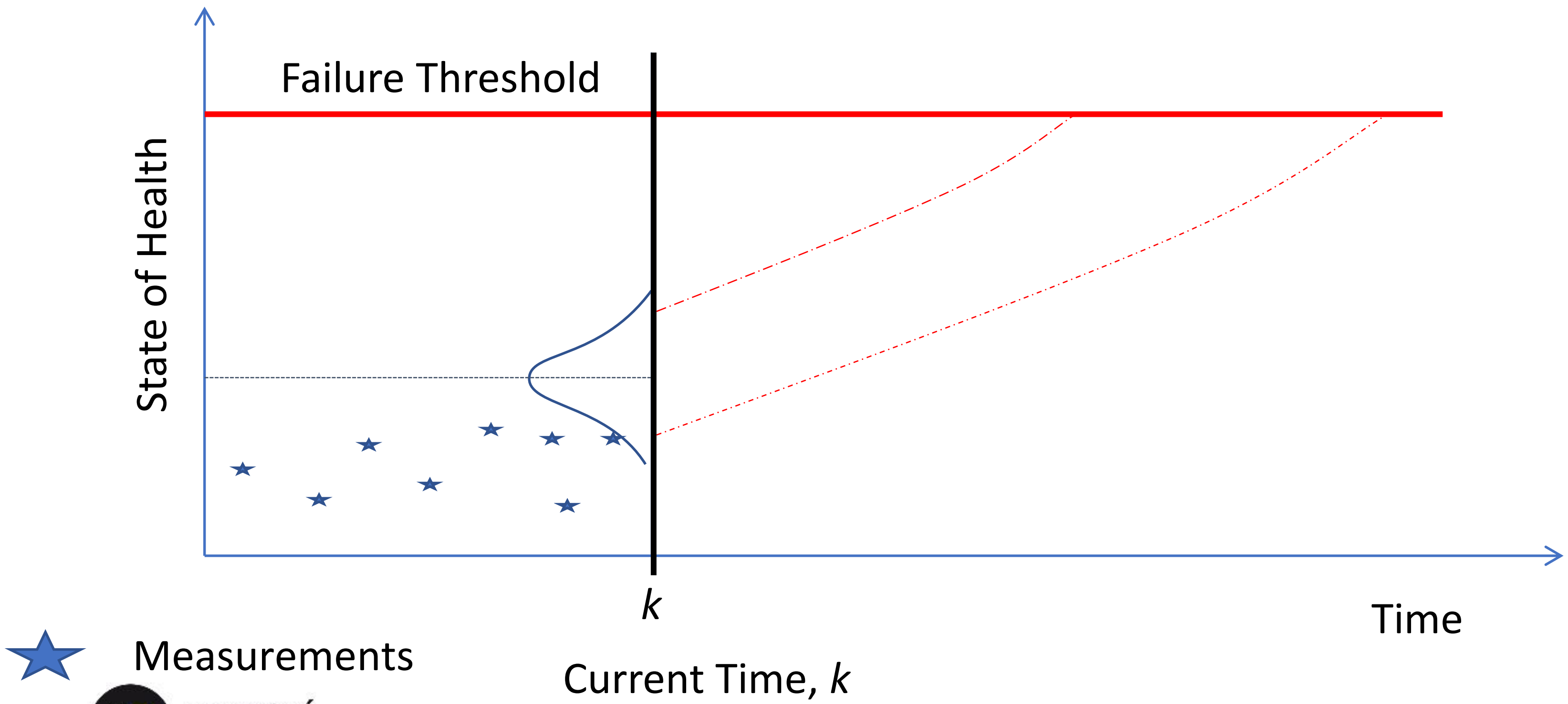


RUL Prediction Illustration

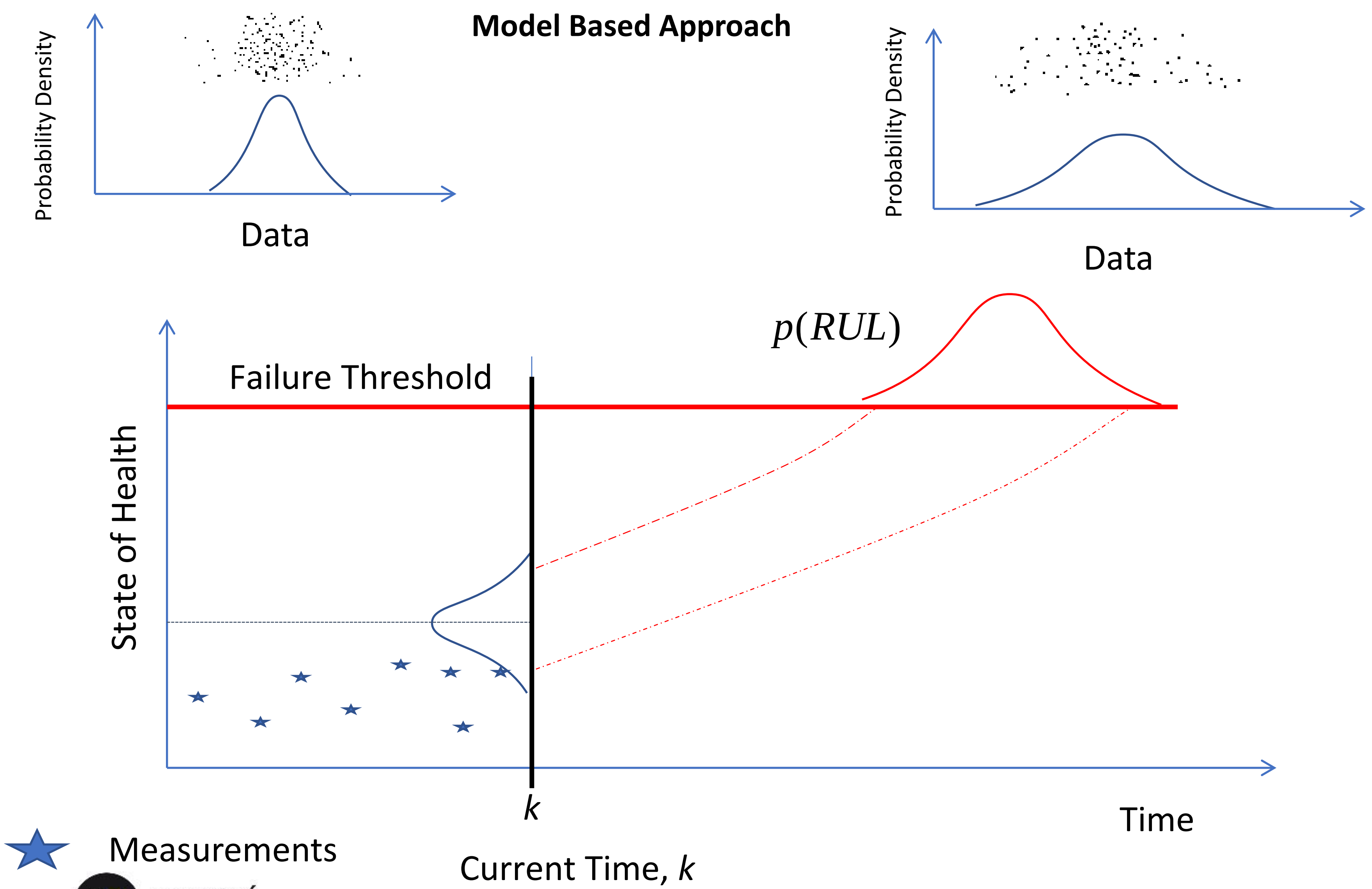
Model Based Approach



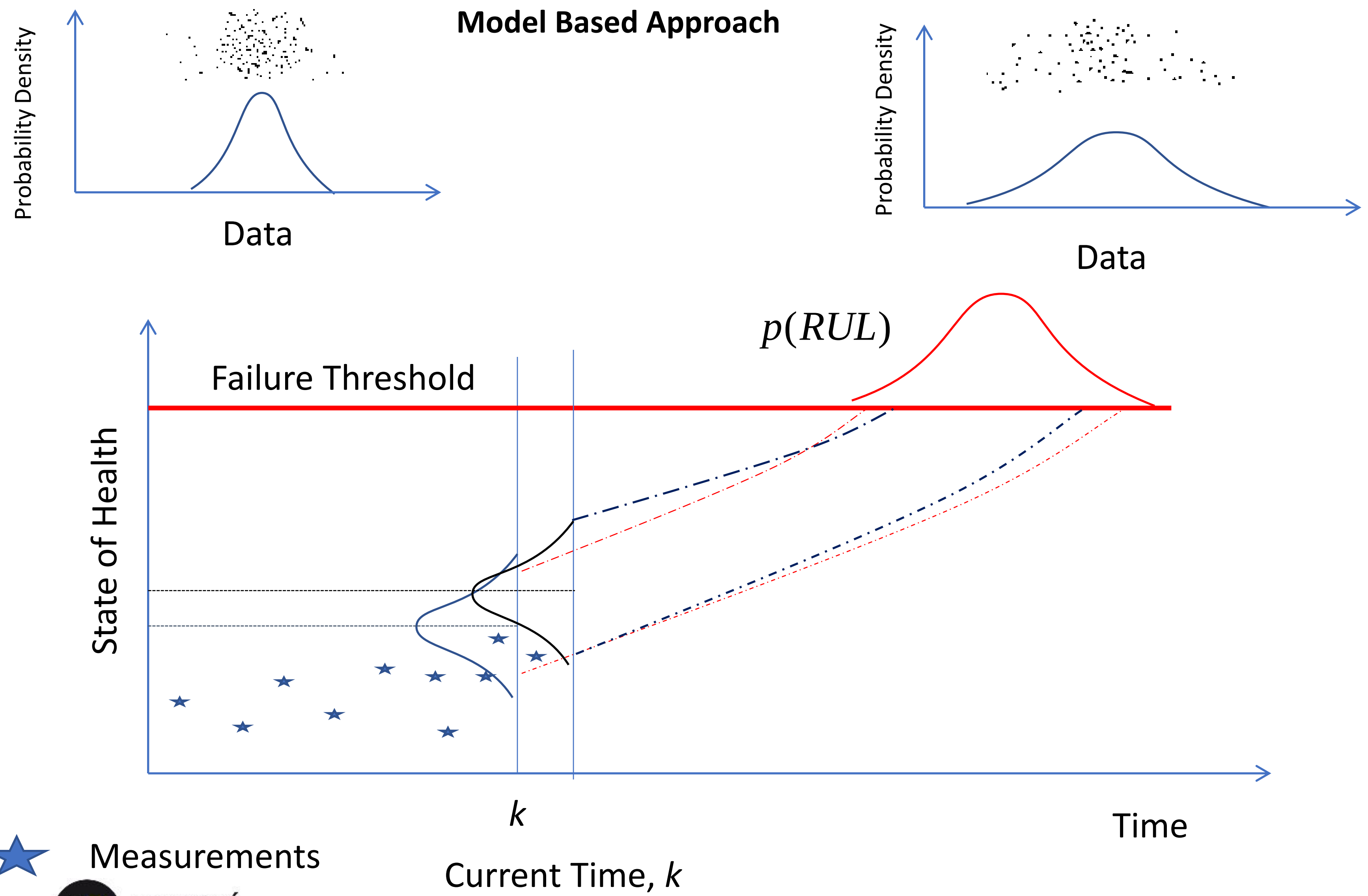




RUL Prediction Illustration

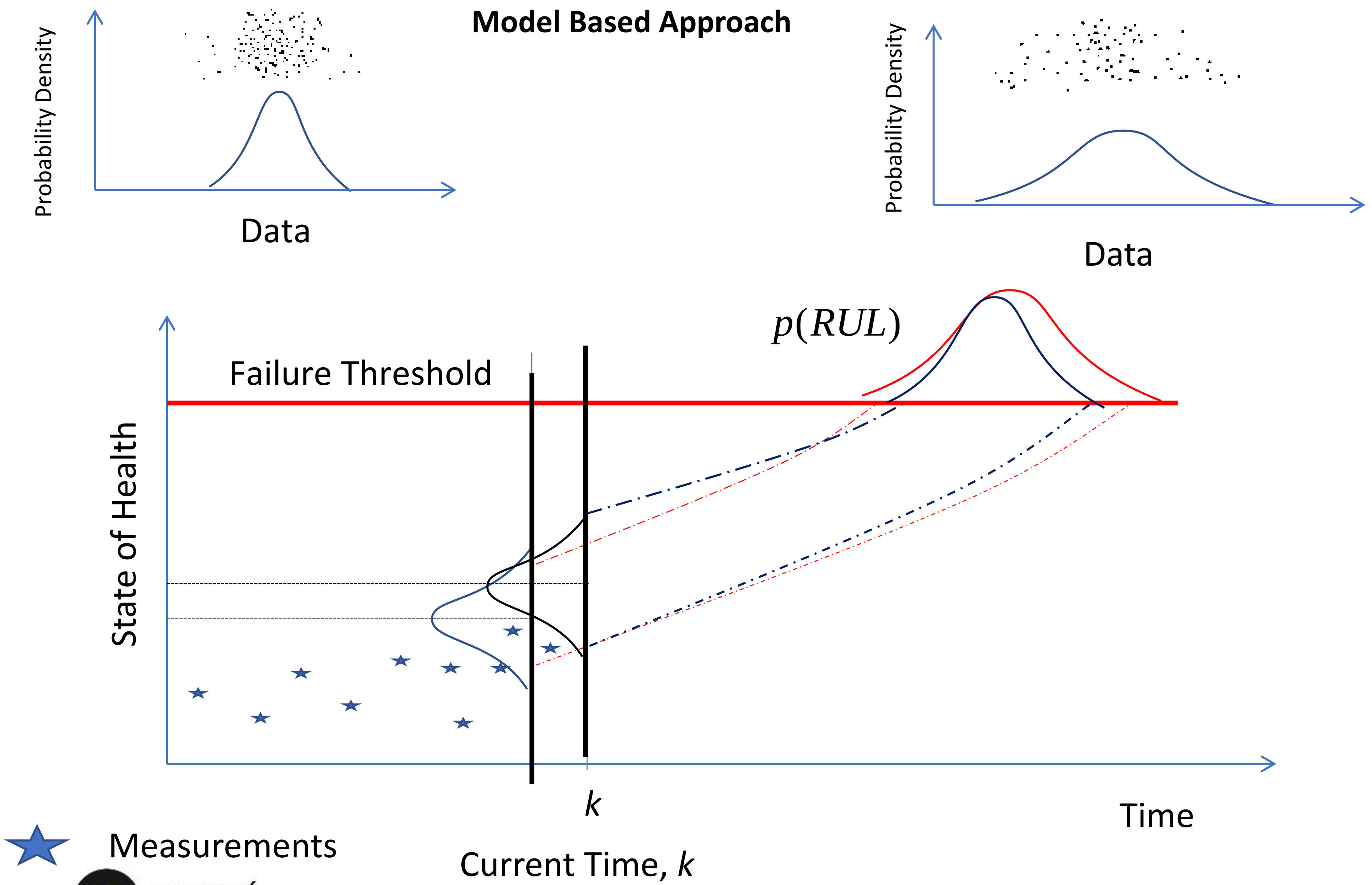


RUL Prediction Illustration



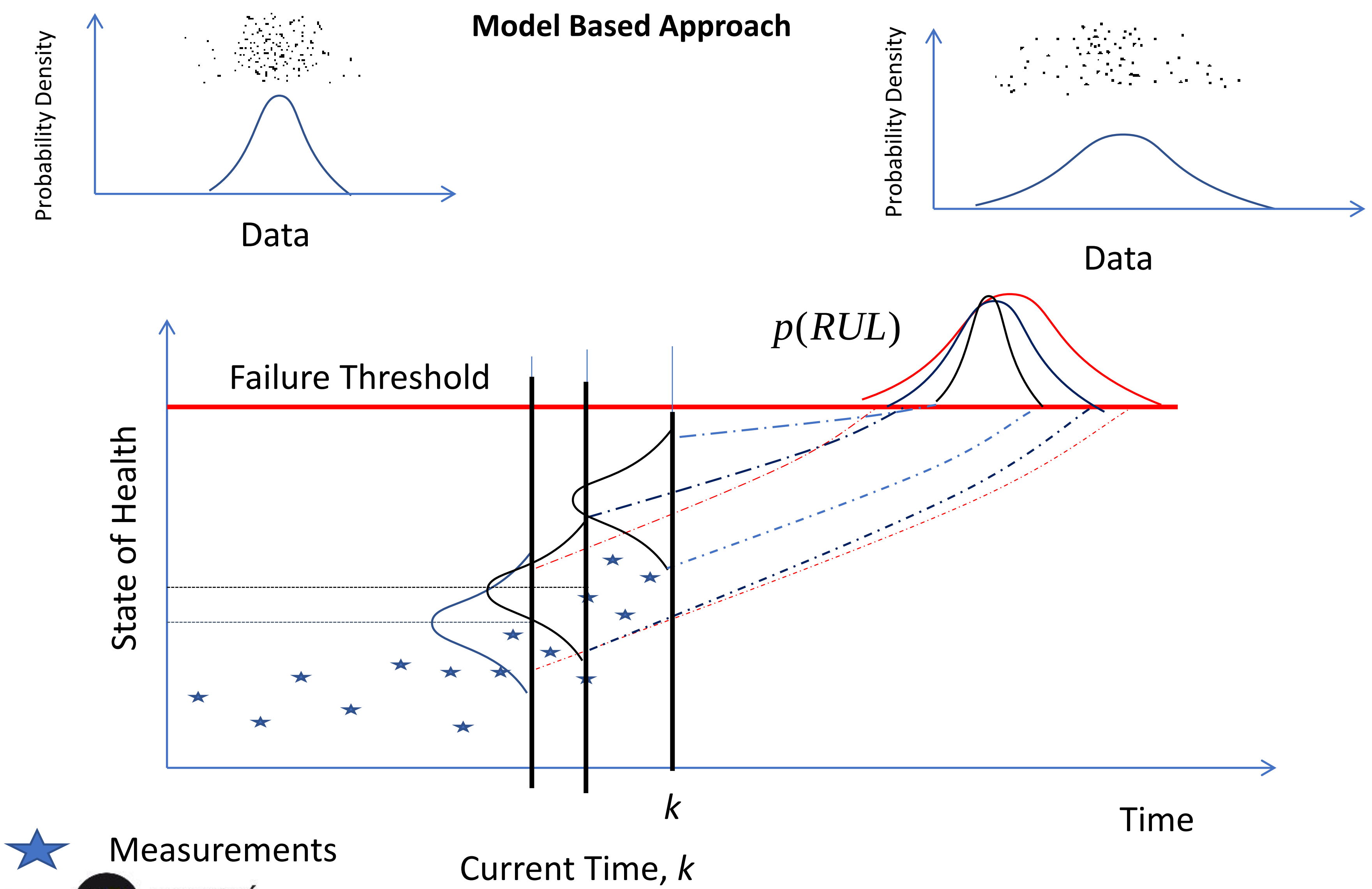
RUL Prediction Illustration

Model Based Approach

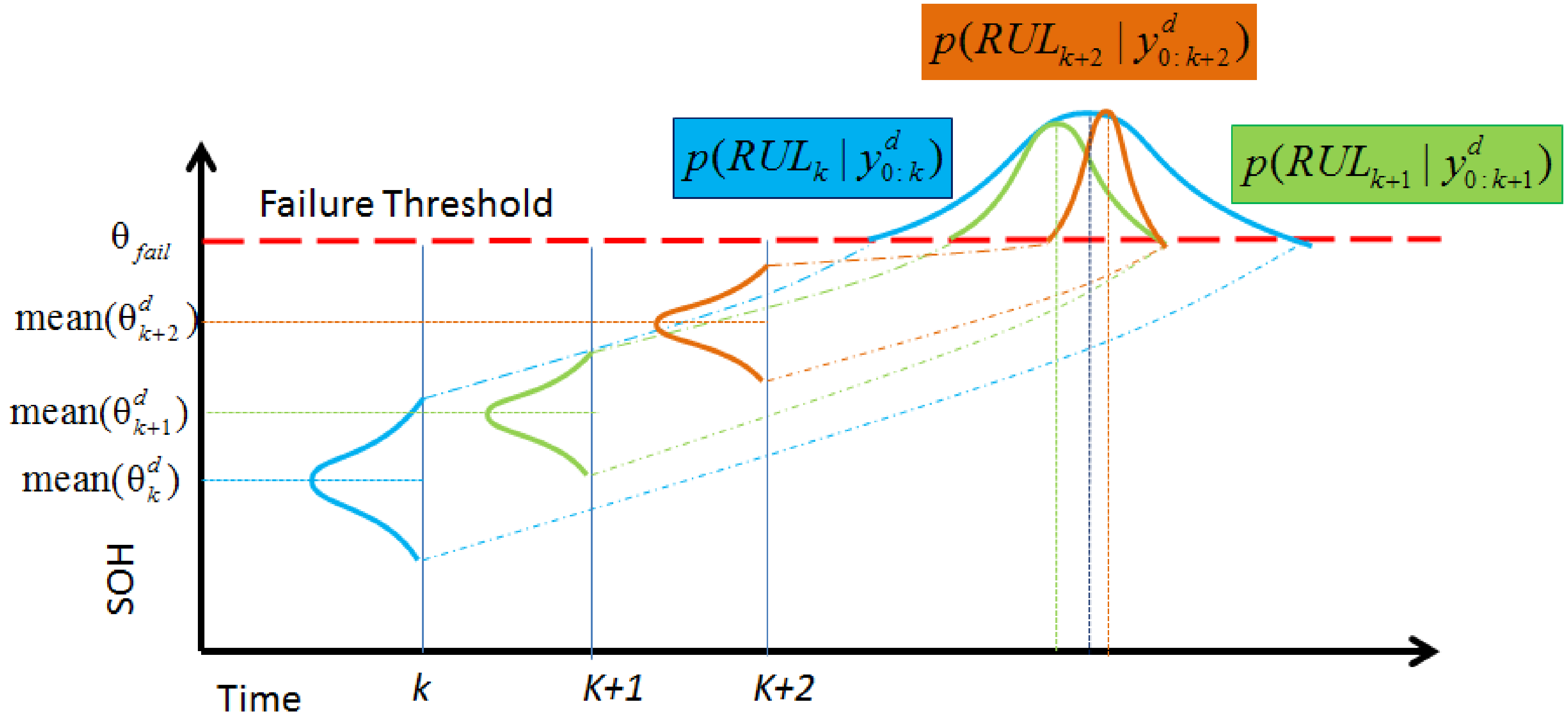


RUL Prediction Illustration

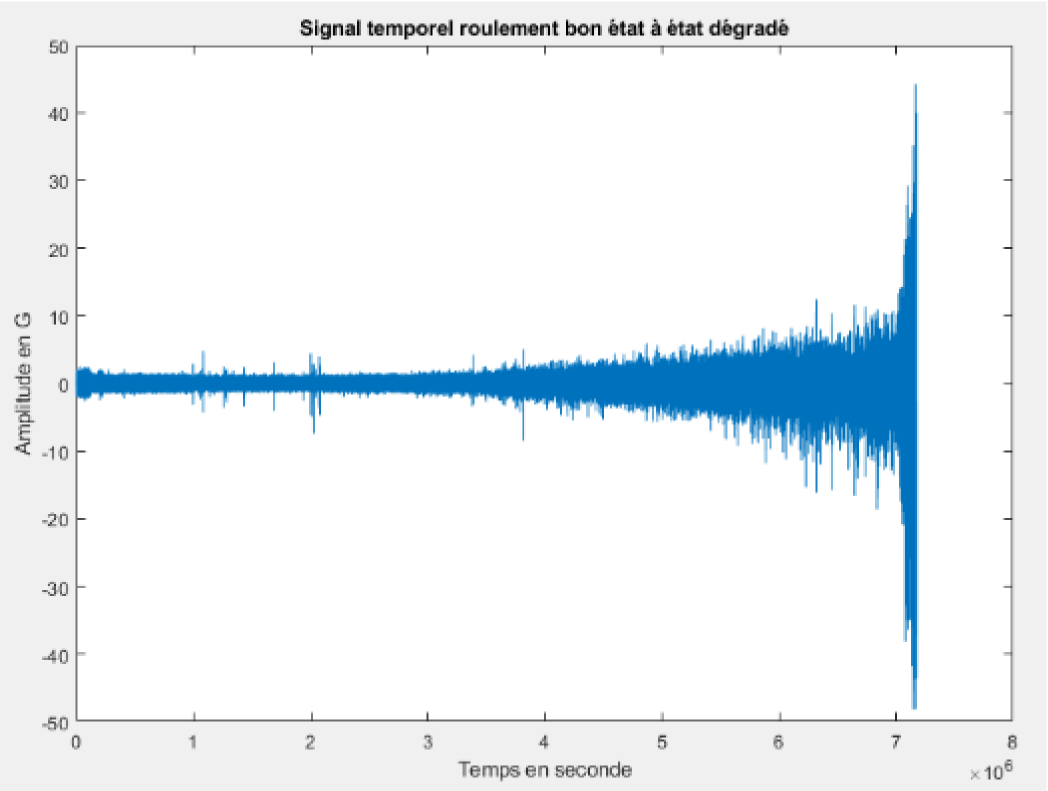
Model Based Approach



RUL Prediction Illustration

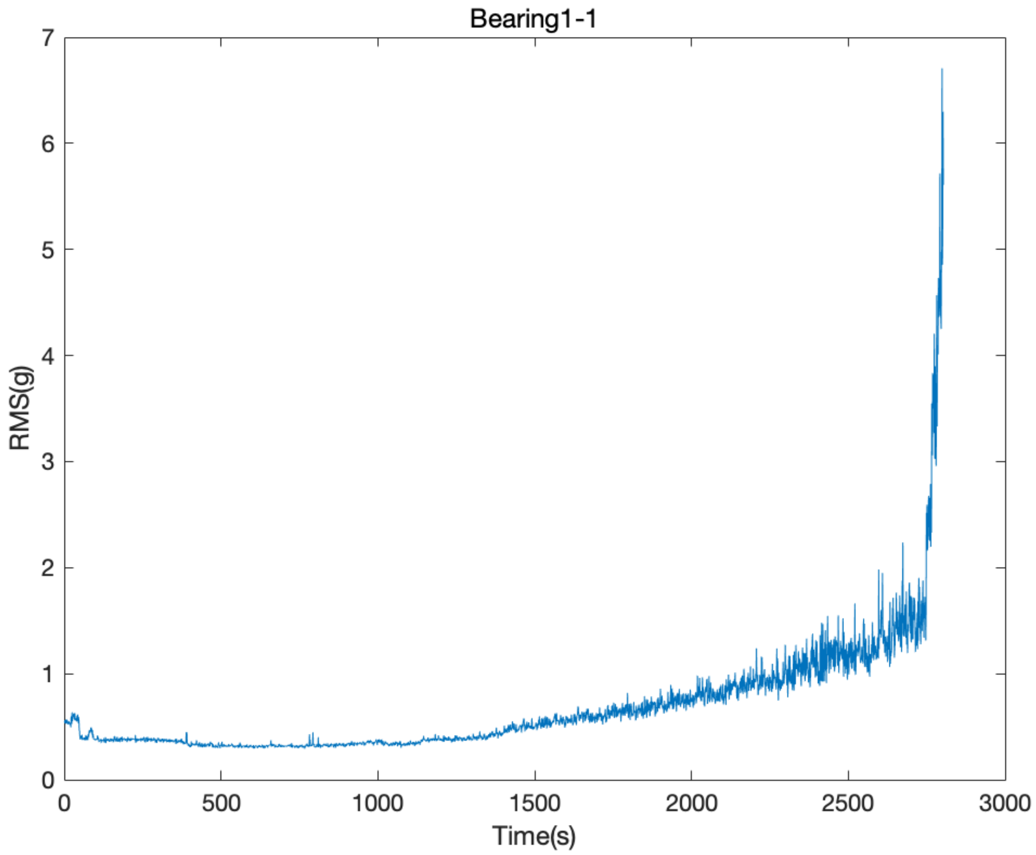


Motivation: Degradation Models (Roller Bearing)



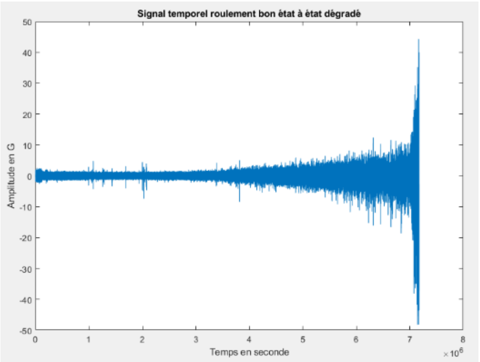
Raw Data

Feature Extraction



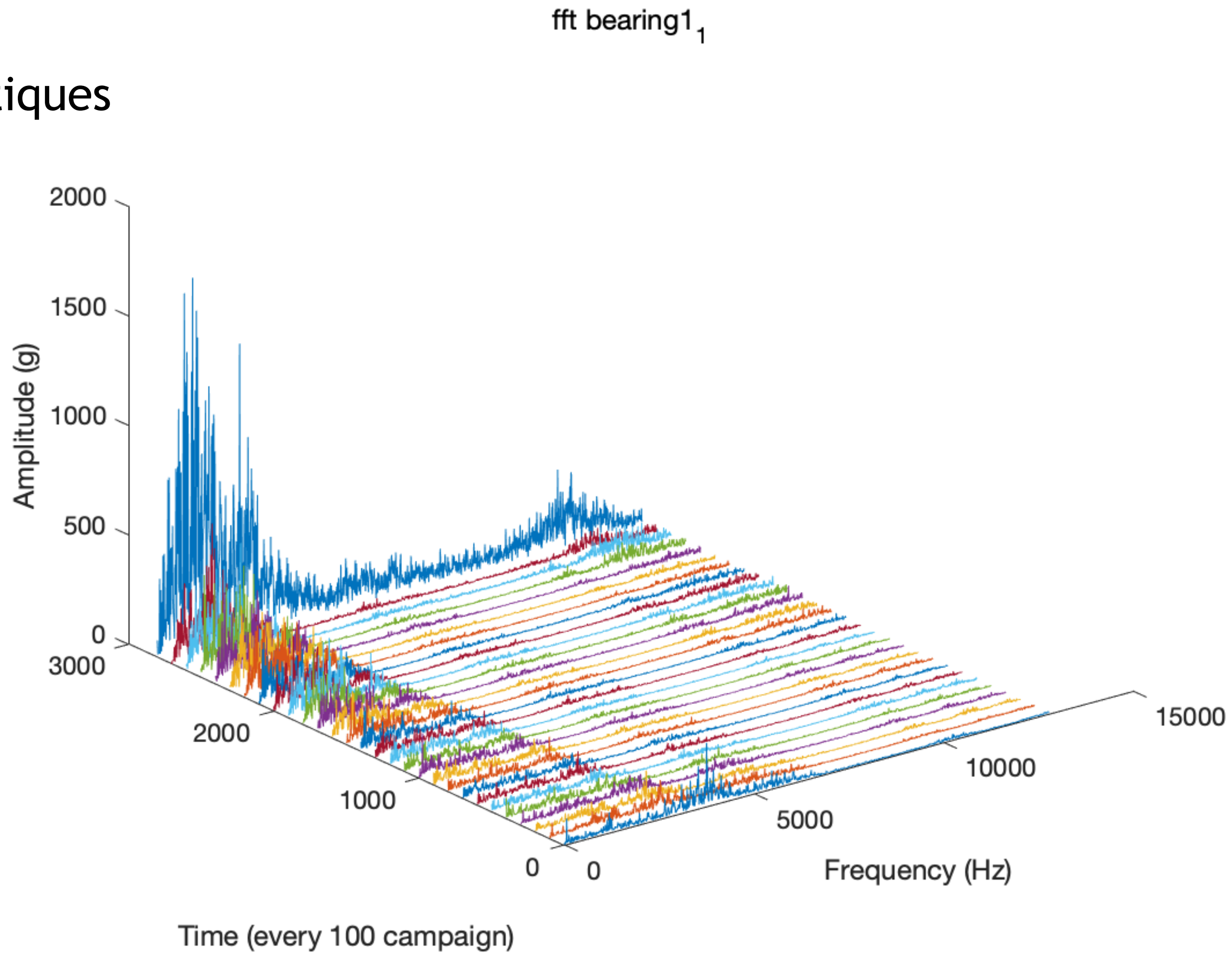
RMS, Time Domain

Motivation: Degradation Models (Roller Bearing)



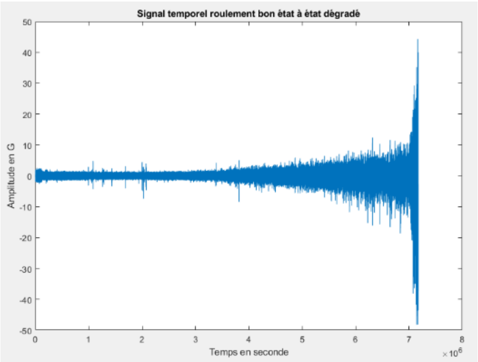
Raw Data

Extraction des caractéristiques
2



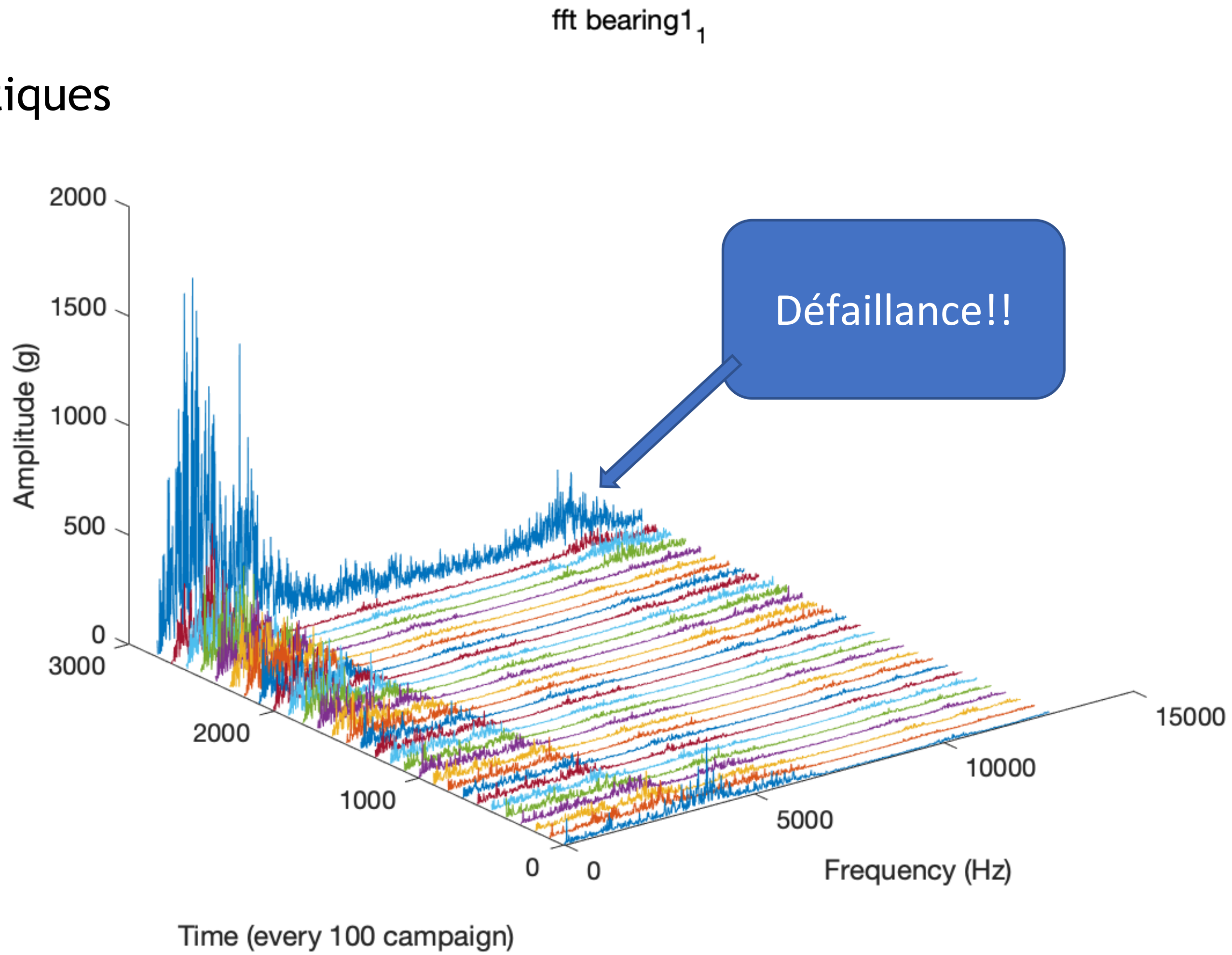
Frequency Domain (FFT)

Motivation: Degradation Models (Roller Bearing)



Raw Data

Extraction des caractéristiques
2

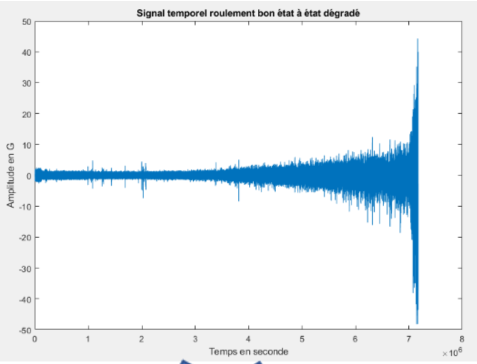


Frequency Domain (FFT)

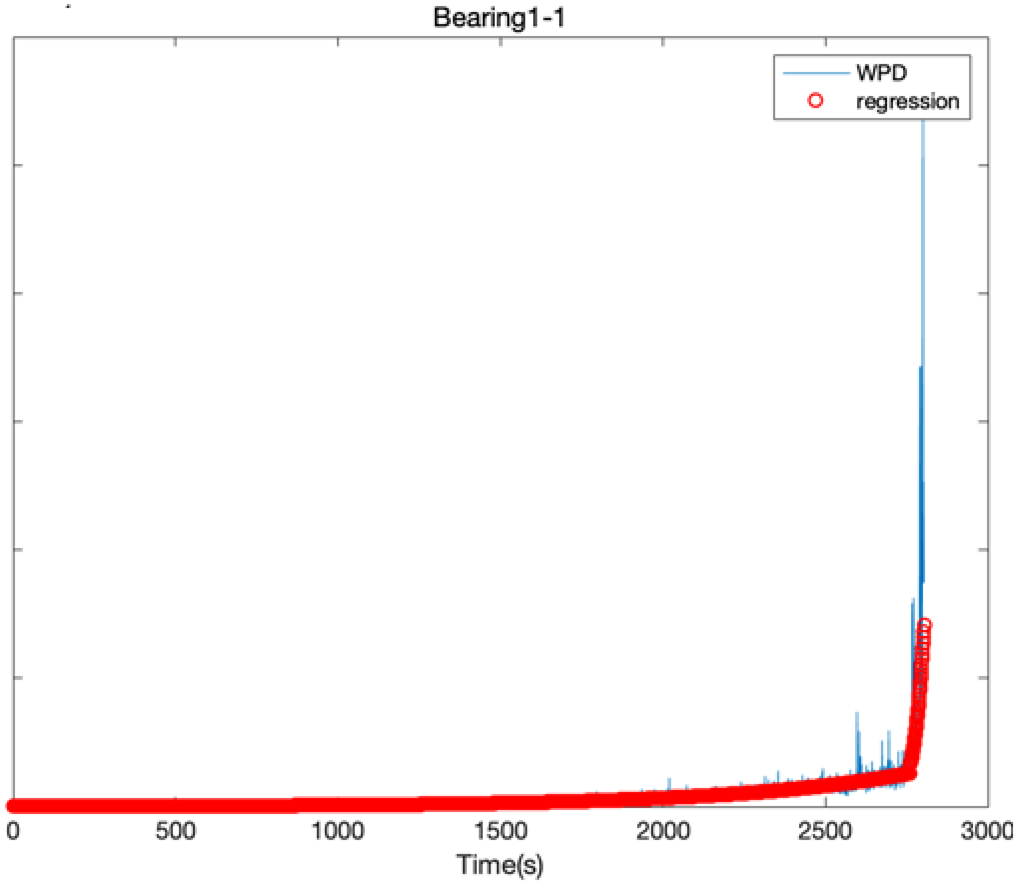
Motivation: Degradation Models (Roller Bearing)

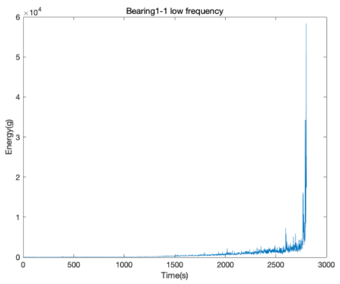
Quelques résultats

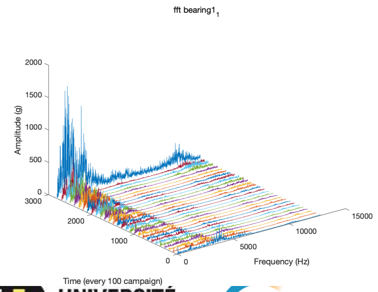
Données brutes

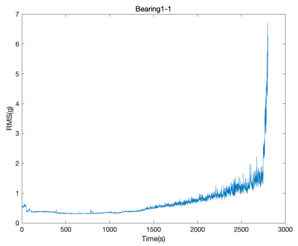


Régression et prédiction

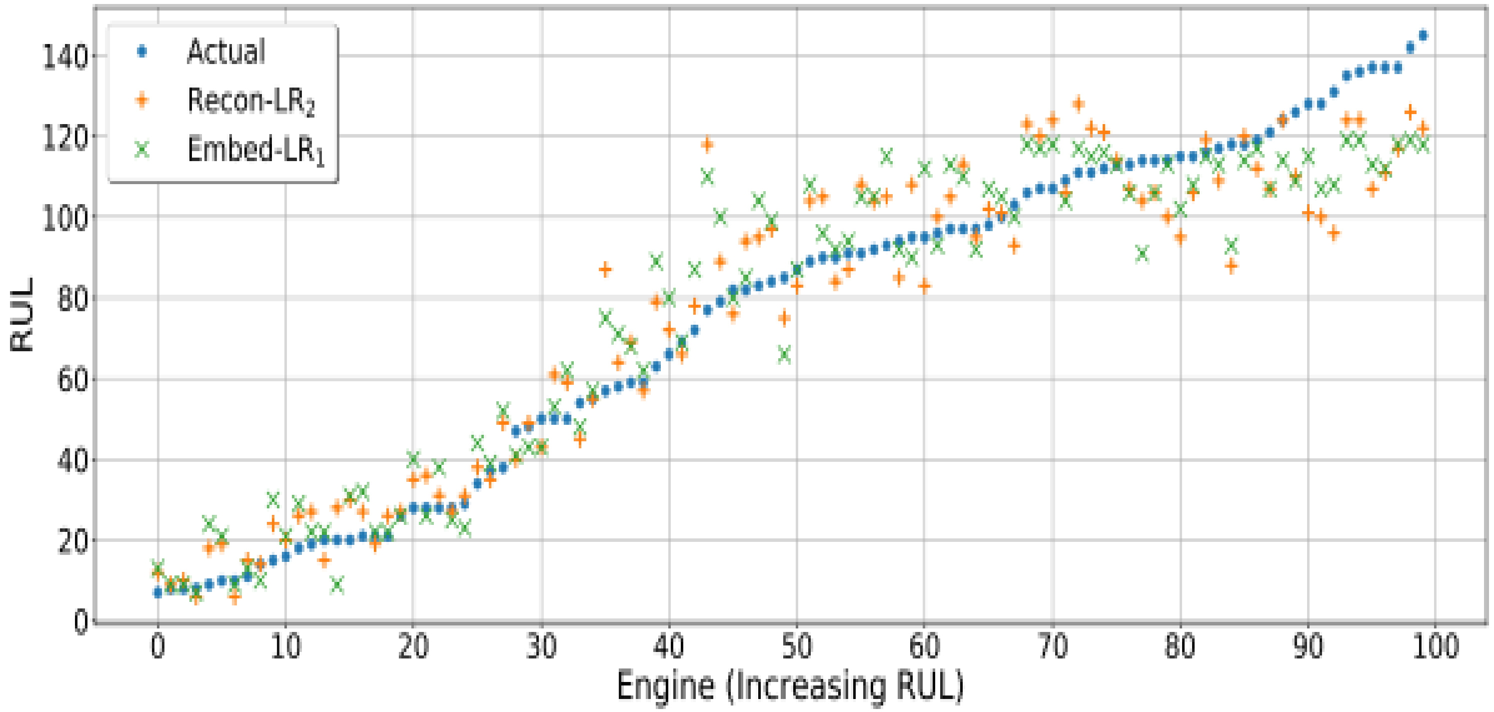






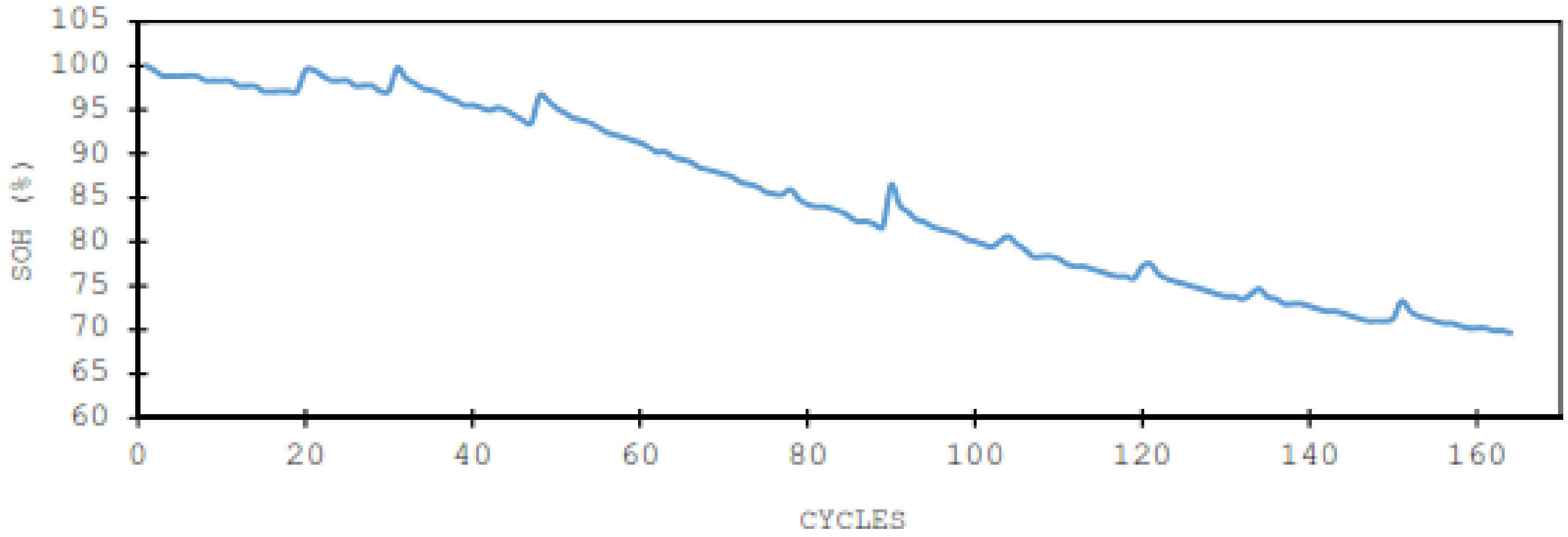


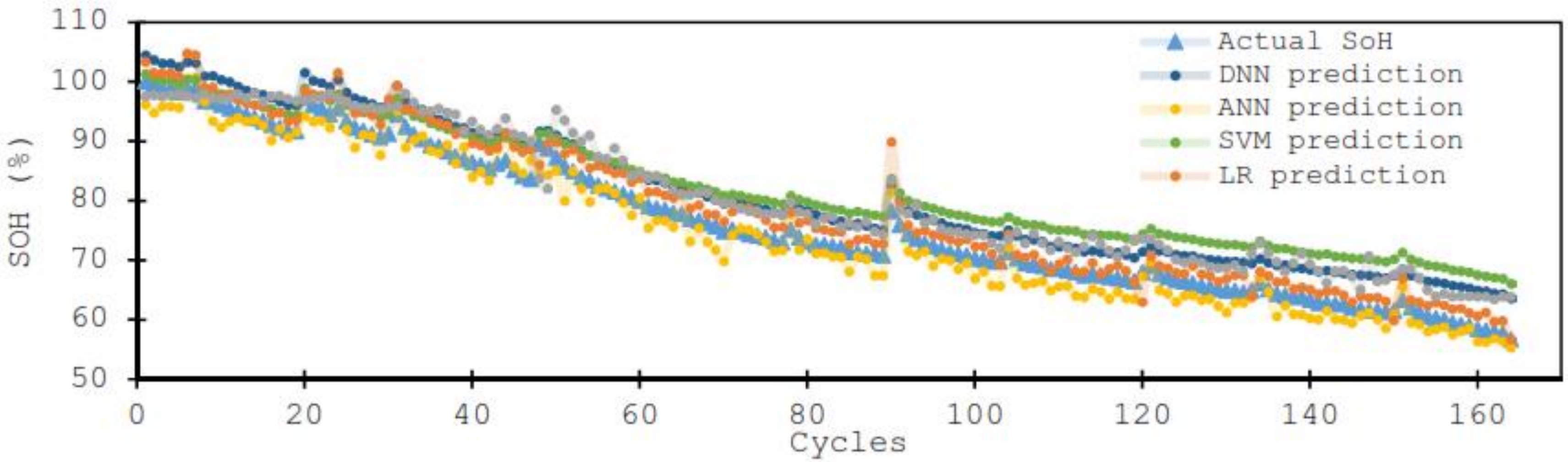
Motivation for Predictive Maintenance: Engine RUL Prediction based on Data



Gugulothu et al.

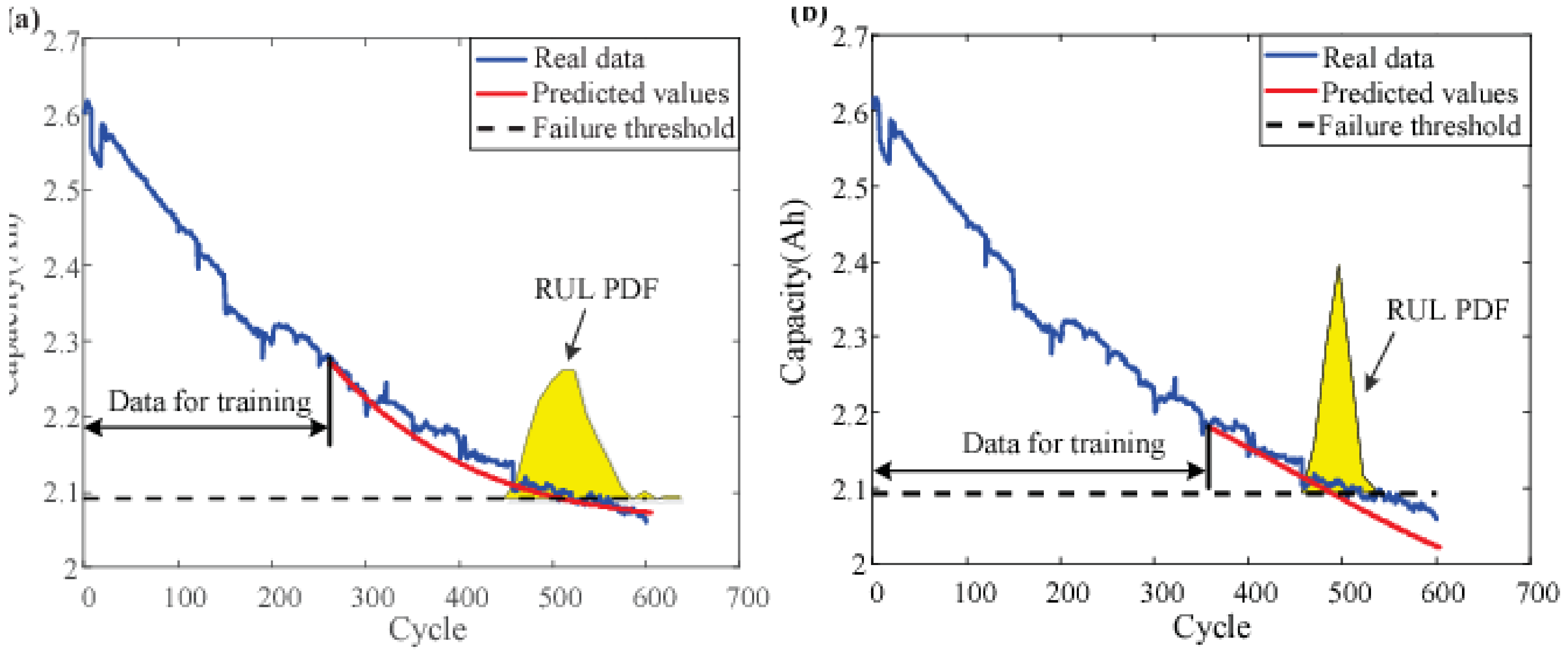
Motivation for Predictive Maintenance: Battery SOH Prediction using Data





Khumprom et al.

Lithium Ion Battery RUL prediction



Zhang et al.

Artificial Intelligence (AI) Domains

AI

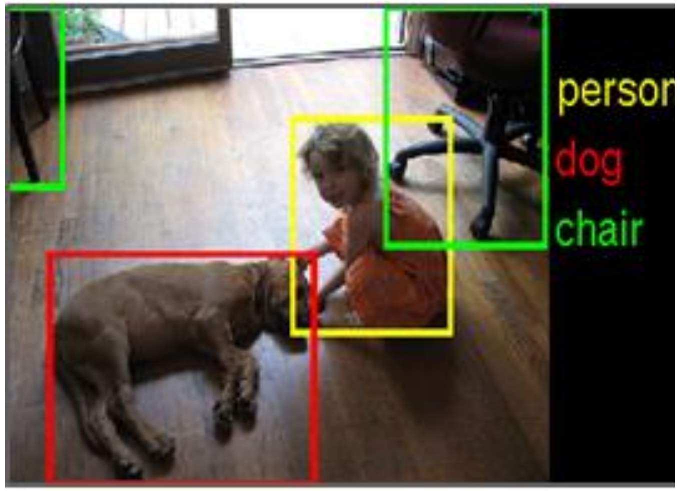
Artificial Intelligence (as of today): Detection and Exploitation of useful patterns and trends in data

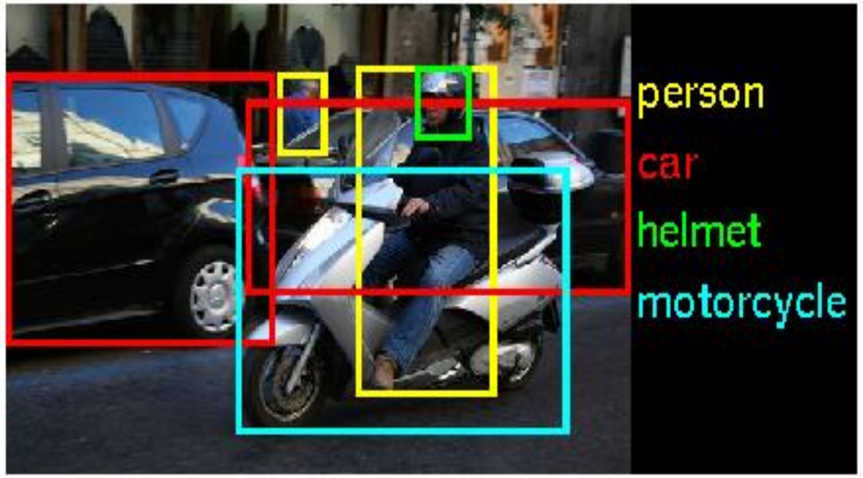
→Decisions

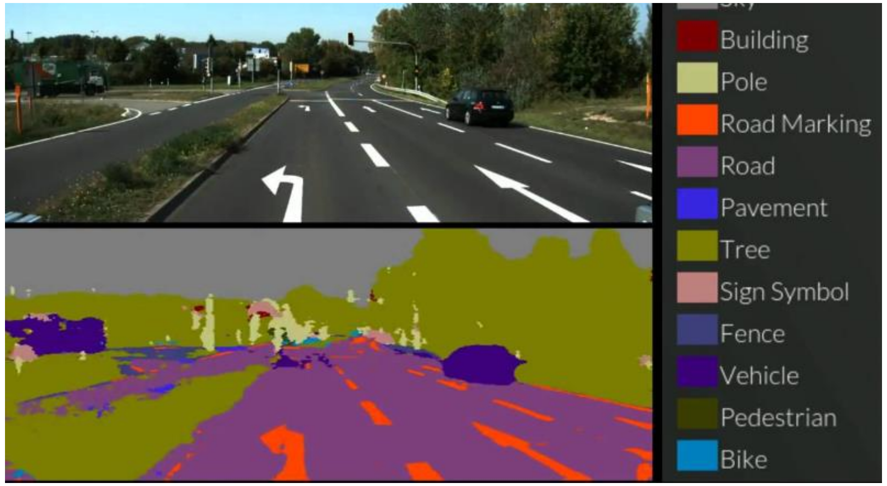
→ Predictions

→Automated Actions

Major Domains C1: Computer vision & Self Driving Cars



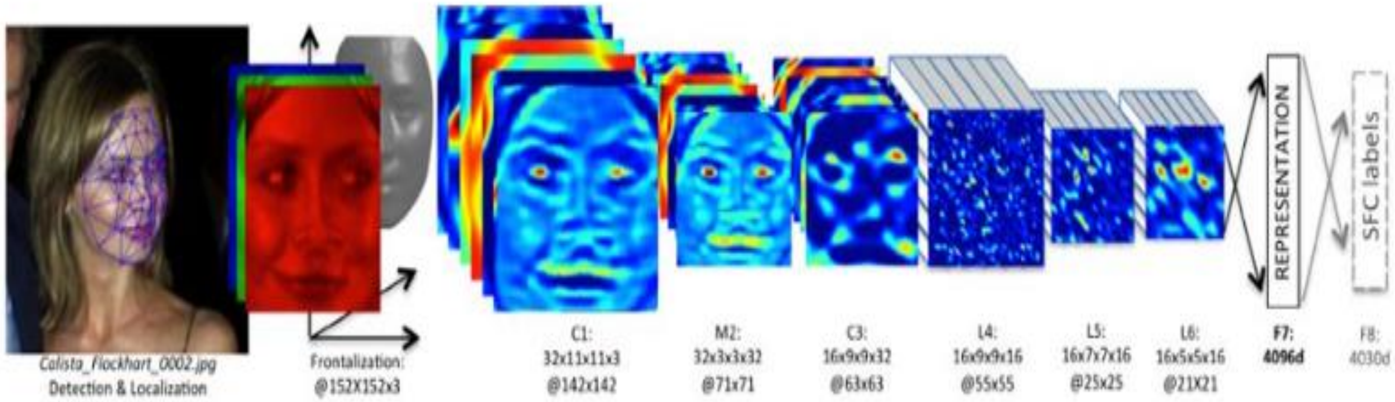




Source : Nvidia, L. Fridman et al.

Domains of AI

C2. Image processing: Shape & Object Detection



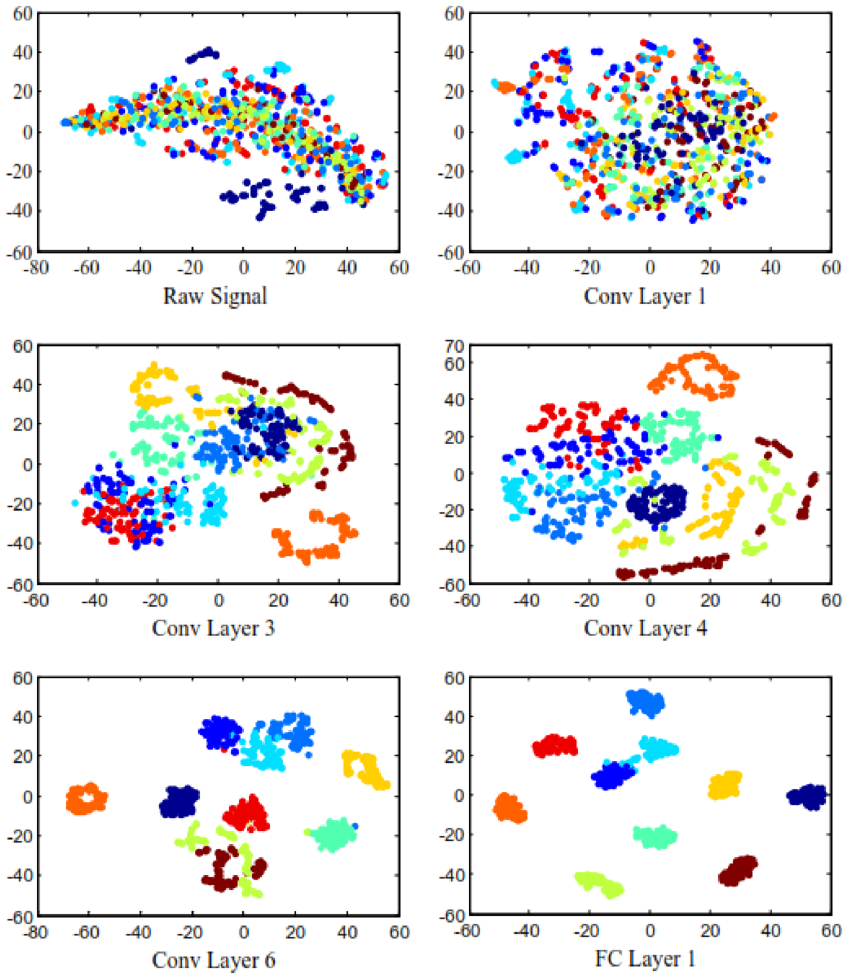
Face detection and Recognition.

Detection  Recognition "XYZ"

[DeepFace: Closing the Gap to Human-Level Performance in Face Verification]
Taigman, Yang, Ranzato, & Wolf (Facebook, Tel Aviv), CVPR 2014

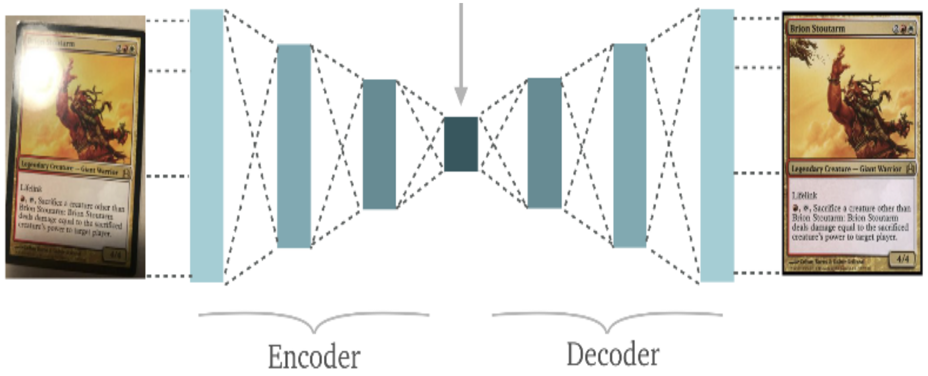
Predictive Maintenance
Fault Detection (Roller Bearing)

Zhang et al.

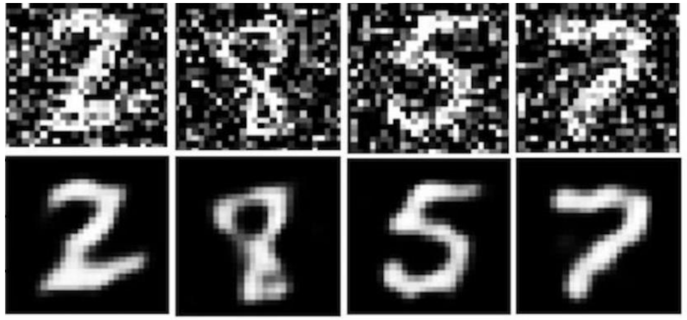


Domains of AI

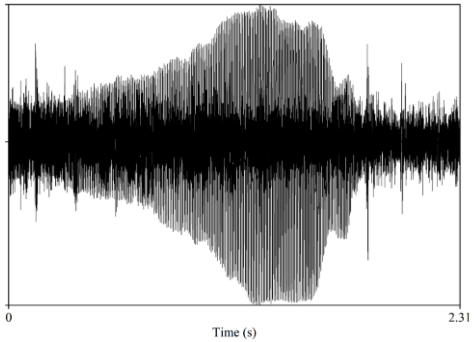
C3. Filtering and Denoising : Auto encoders



Link: Denoising autoencoder for Image classification



End to End learning: Fault detection and Prediction:
Unknown Model, Environment. (JHA et al. 2017)



Inputs

Learning in Black Box

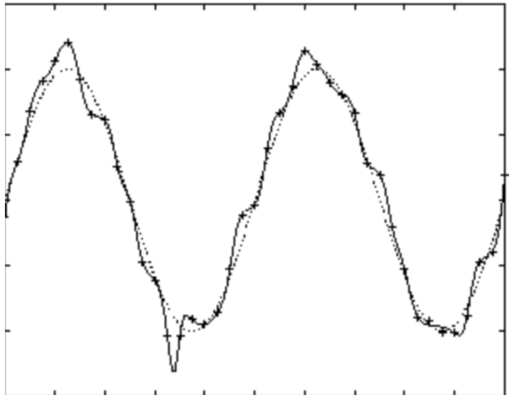
Black Box
- Feature extraction,
- selection,
- Unsupervised Learning

Decisions

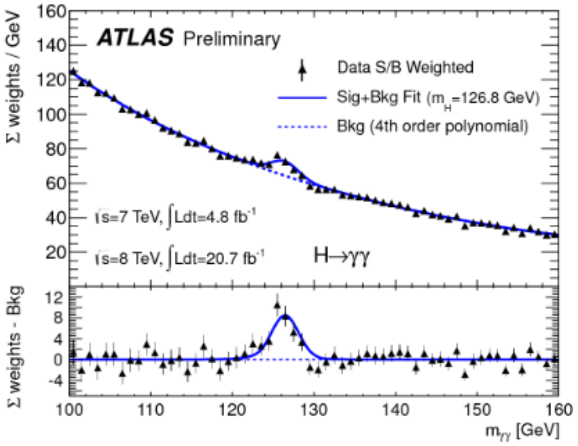
Denoising Autoencoders: MATLAB Central

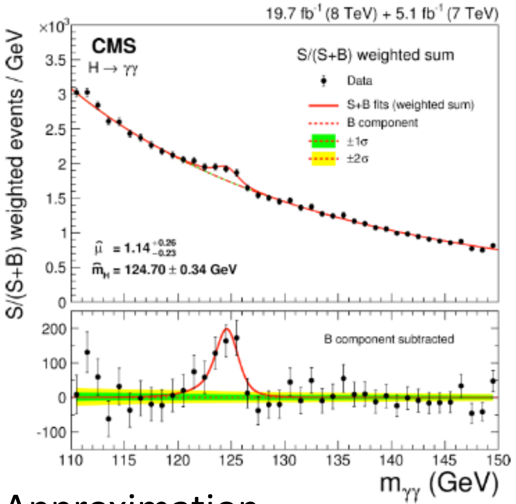
Domains of AI

C3. Particle Physics, Intelligent control (adaptive) of systems, Robotics: Function Approximation



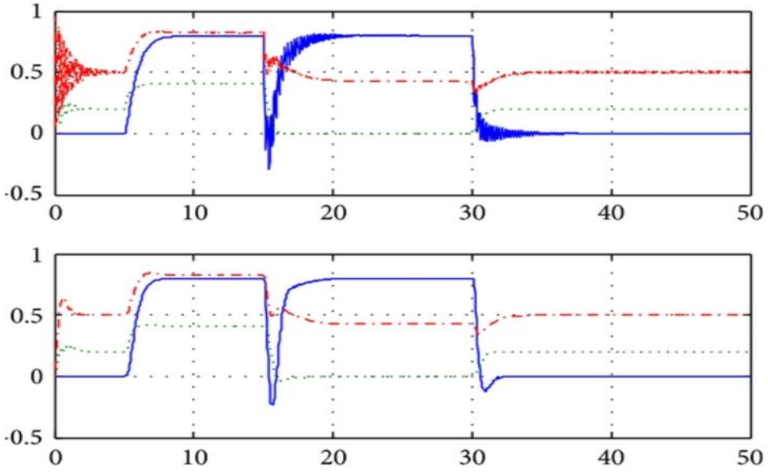
- Universal function approximators
- Efficient approximation of unknown dynamics .

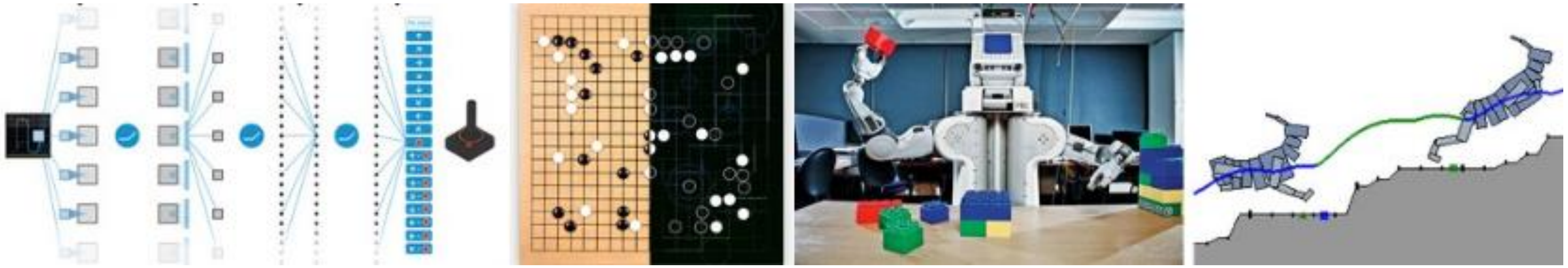




Deep learning enabled function Approximation in LHC physics.

Sirunyan et al. 2019, Physical Review letters





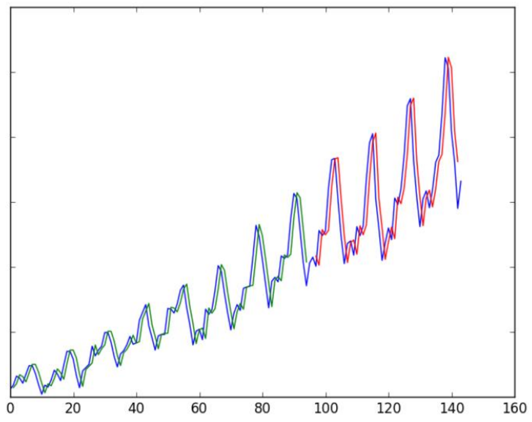
Deep learning **based function approximation of Extremely large state space** (World)

Human Level control through Deep Learning

Mnih et al. 2015, *Nature*

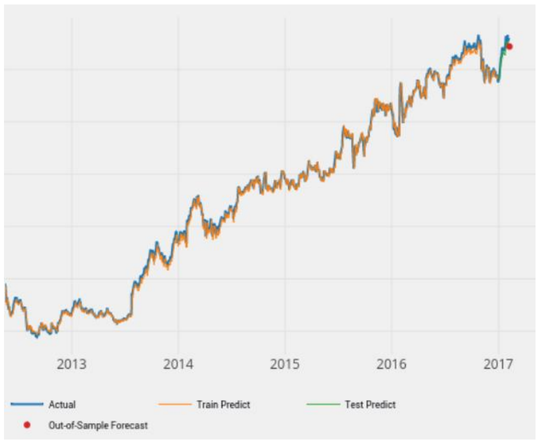
Domains of AI

R1: Time Series Forecasting, Trend Prediction, Event Prediction

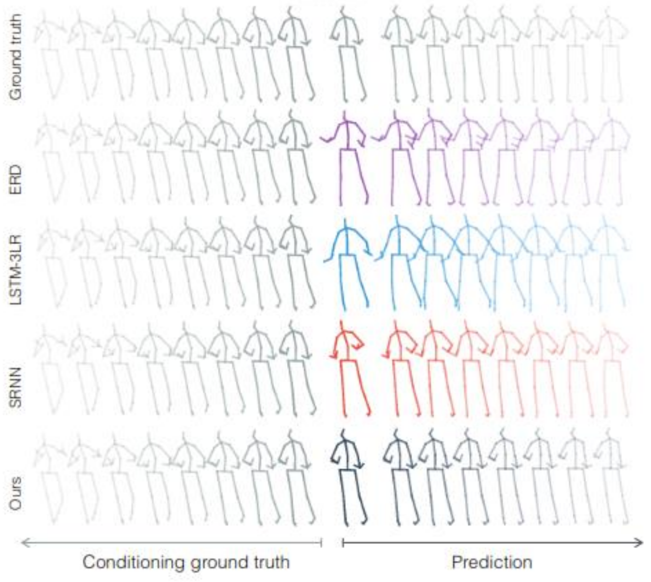


Long terms traffic Speed prediction

Ma, Xiaolei, et al

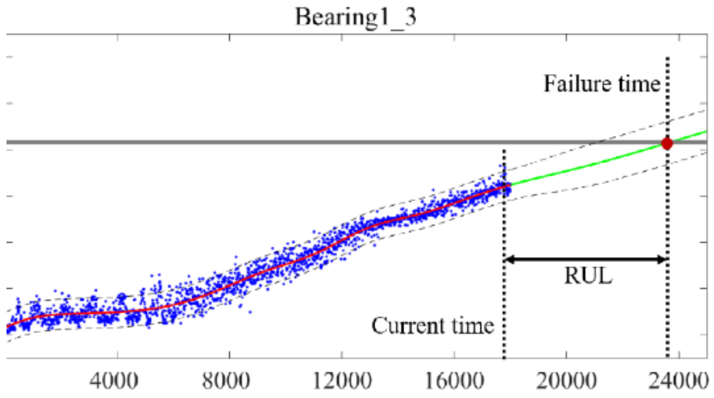


Financial market prediction (Dixon et al.)

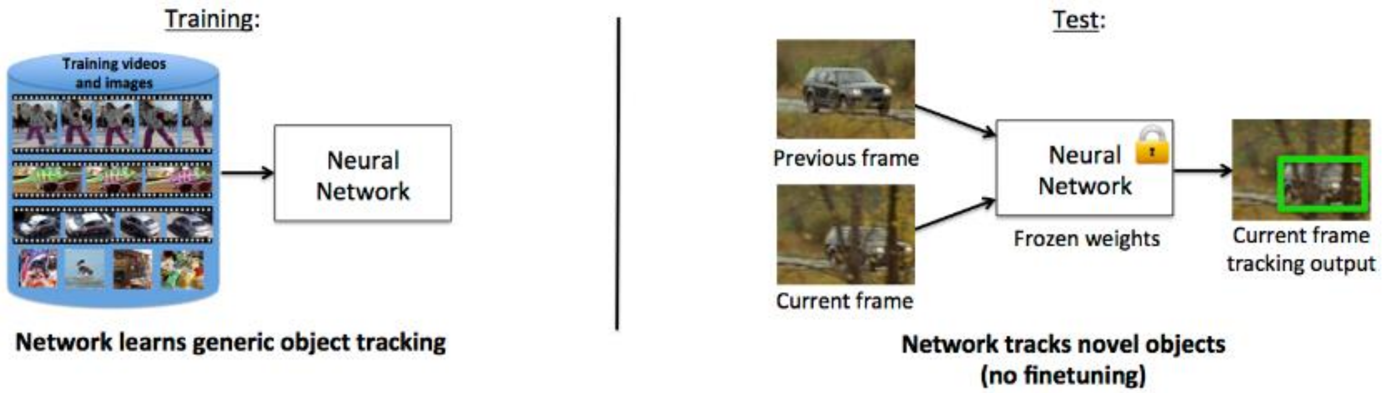


Human Motion Prediction

Martinez et al., 2016



Component Failure Prediction (Yoo et al., 2018)

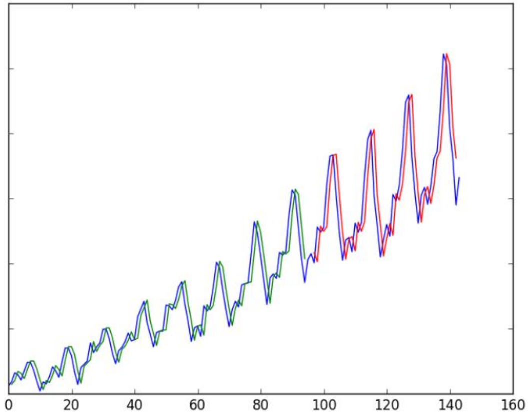


Video Frame tracking and Prediction

Held et al.

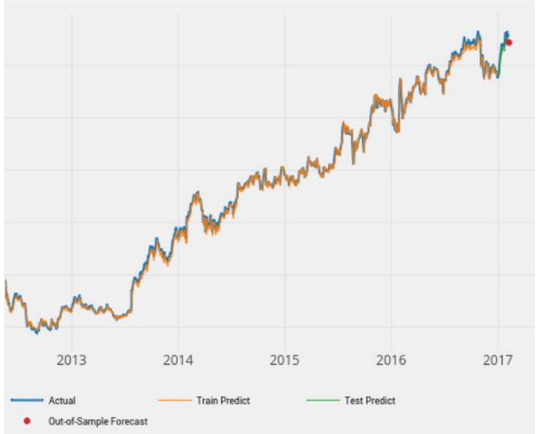
Domains of AI

R1: Predictive Maintenance : Bearing RUL Prediction.



Long terms traffic Speed prediction

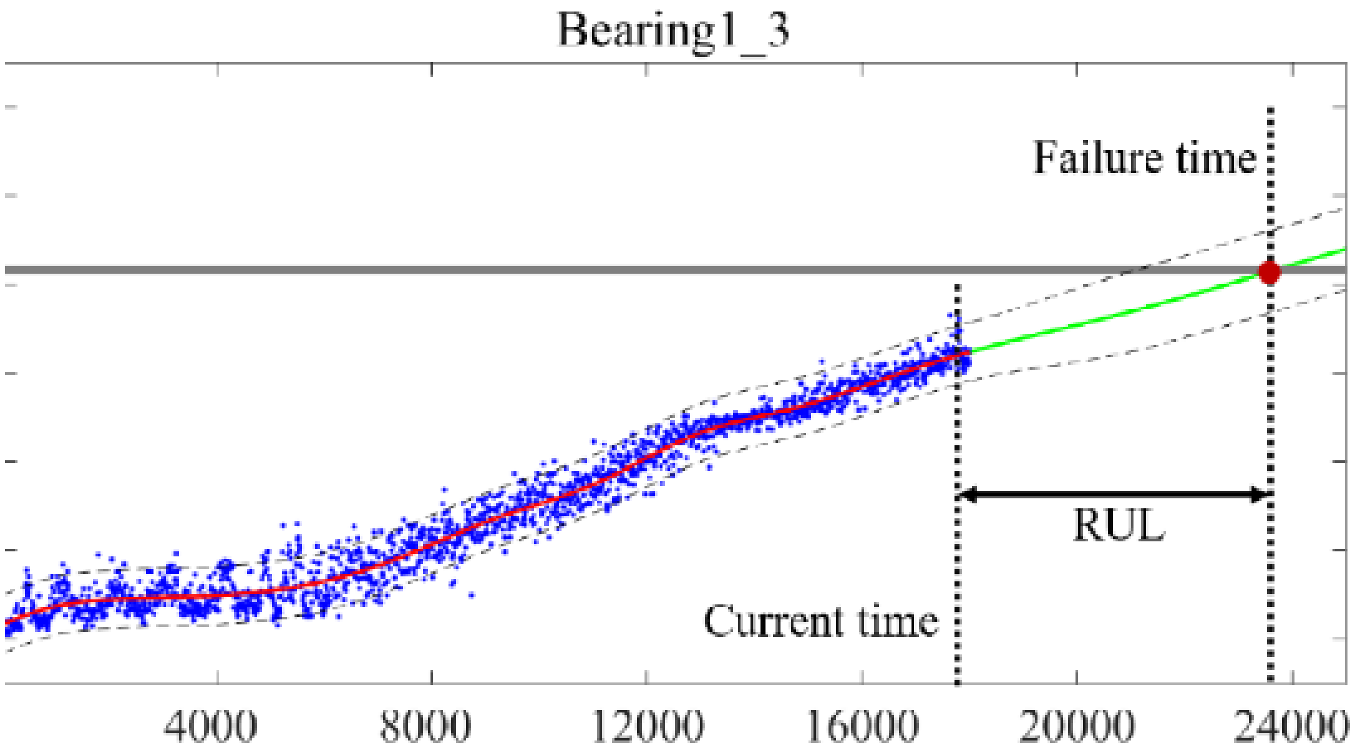
Ma, Xiaolei, et al



Financial market prediction (Dixon et al.)

Video Frame tracking and Prediction

Held et al.

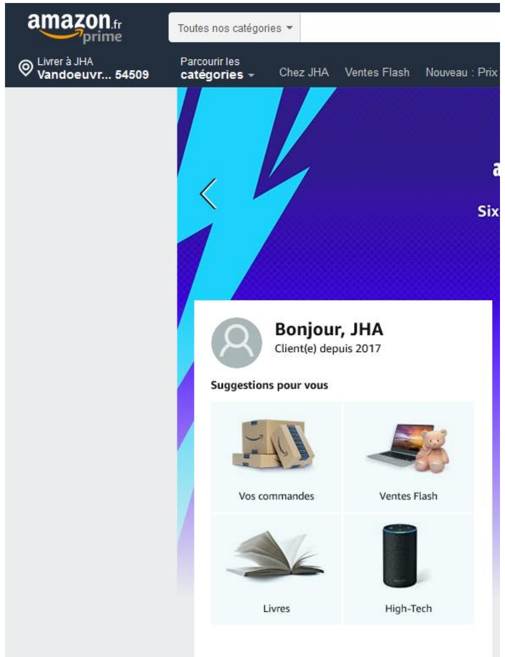


Component Failure Prediction (Yoo et al., 2018)

Domains of AI

R2 : Recommendation Systems

- Candidate Profiling,
- Scoring , similarity measures,
- Prediction



- google Translate
- voice recognition
- text prediction
- voice to text and vice versa
- echo cancellation

Google home Mini
Alexa

Sequence prediction often involves forecasting the next value in a real valued sequence.

Domains of AI

Reinforcement Learning: Towards human level :

control ((Finding the optimal way of doing a given task)

prediction

Adaptation (Robots That Can Adapt like Animals, *Nature*)

Built
new
moves

Email: *mayank-shekhar.jha [at] univ-lorraine.fr*

Types of Learning

Learning : Supervised vs Unsupervised

Machine Learning: Study of algorithms that improve their **performance,** for a given **task**, with more **experience**.

Training data: $\{y,x\}=(y,x)_1,\ (y,x)_2, \ldots .. \ (y,x)_N$

Function space: F(x,w)
and constraints on function

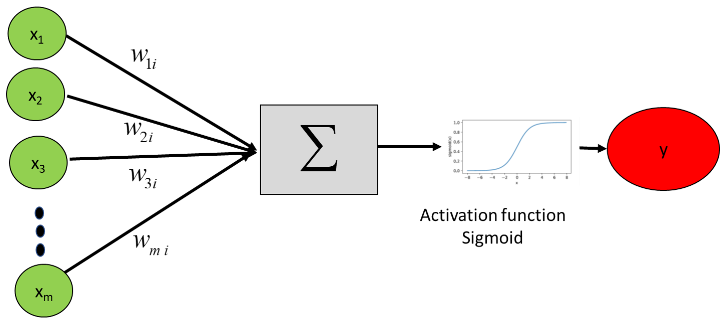
F

$f\,(x, \mathbf{w}^*)$ $\quad$ $C(\mathbf{w})$

Teach a machine to learn the **mapping y = f(x,w*)**

Optimal parameters or "BEST "parameters

Learning : Supervised



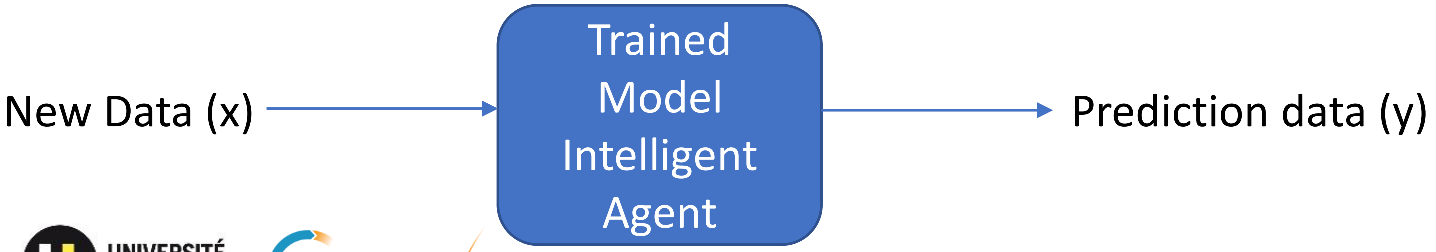
Training data: $\{y,x\}=(y,x)_1,\ (y,x)_2,\ldots..\ (y,x)_N$

Function space: F(x,w)
and constraints on function

Teach a machine to learn the **mapping y = f(x,w*)**

Supervised learning:

- Training of intelligent agent under 'supervision'.
- Model known, environment known.
- Data sources, labels known!
- An algorithm is employed to learn the mapping function from the input variable (x) to the output variable (y) and optimal function parameters: that is $y=f\,(x,\,\mathbf{w}^*)$
- Objective: Mapping function estimated accurately→ Agent Intelligent! WHY??



POLYTECH NANCY
UNIVERSITÉ DE LORRAINE
CRAN

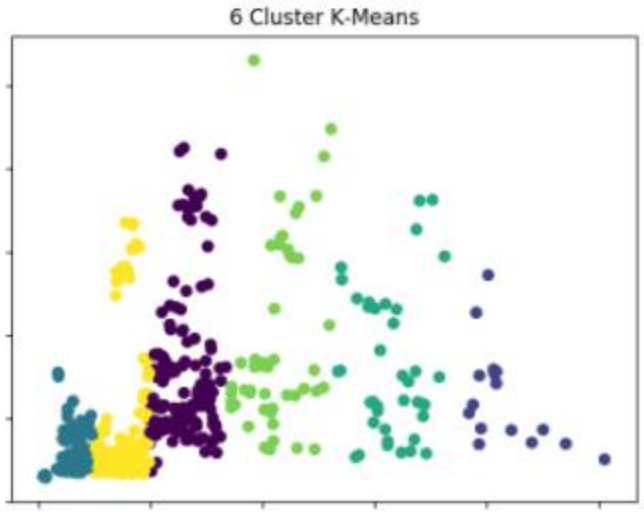
Learning : Unsupervised

Unsupervised learning = Available input data (X) and NO output .

- LEARNING DONE IN AUTONOMOUS WAY.

- The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

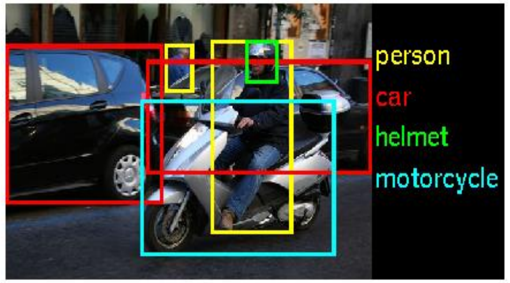
Example: K-mean clustering (using distance measures , similarity index, other ranking algos)

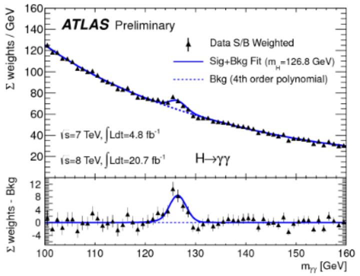


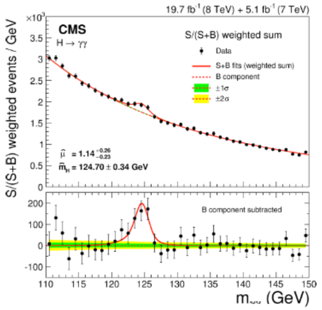
There is no correct answer and there is no teacher.

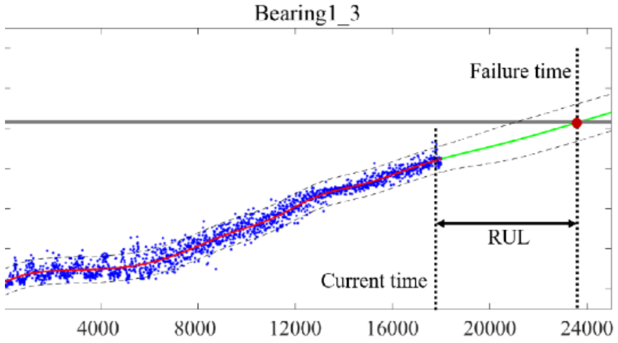
Algorithms are left to their own to discover and present the interesting structure in the data.

Remark: Most learning (in practice) : **supervised.**



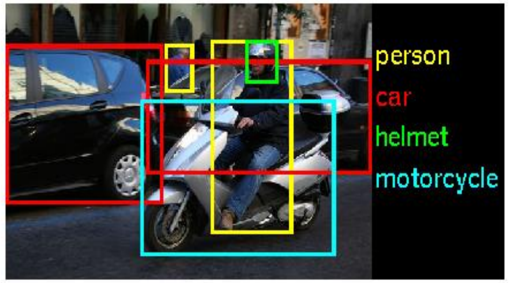


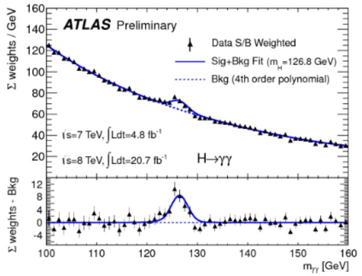


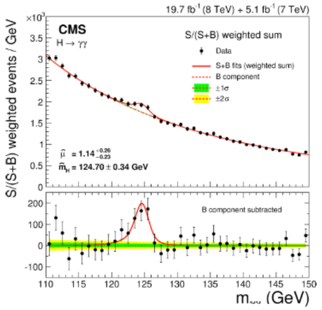


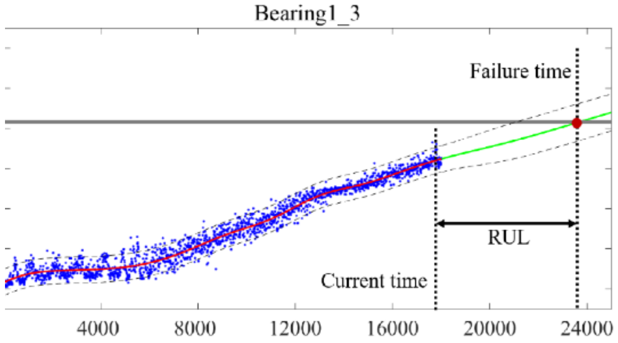
Remark: Most learning (in practice) : **supervised.**

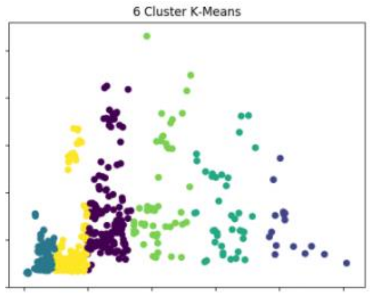
Remark: Most learning (in research) : **Unsupervised, RL**

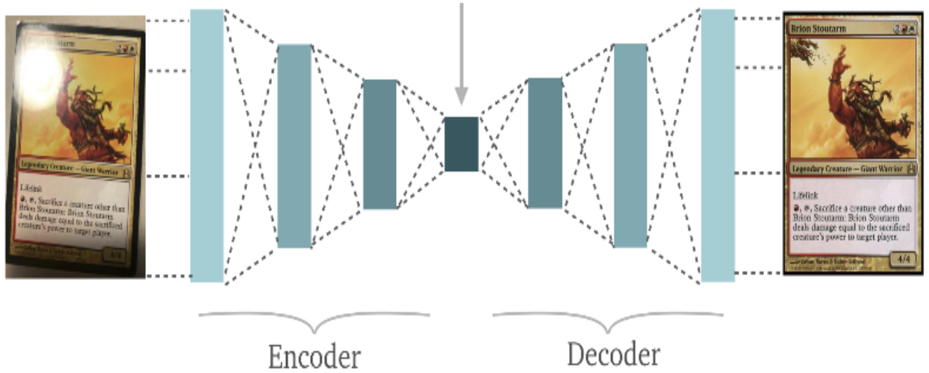


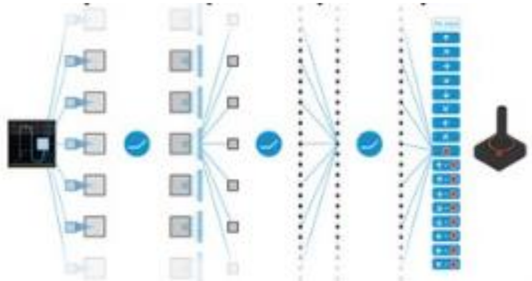


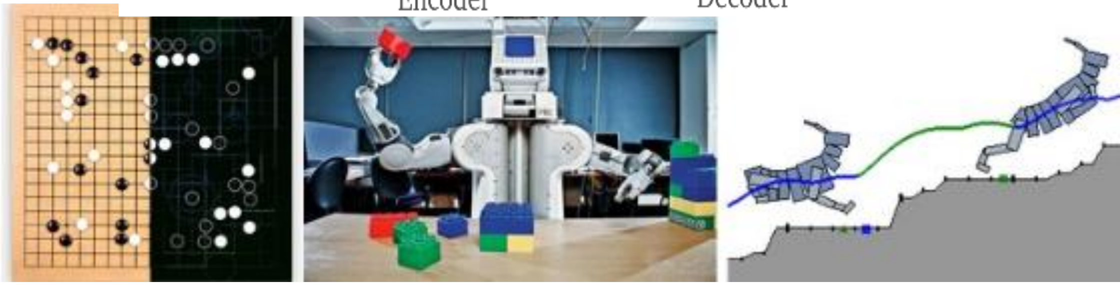




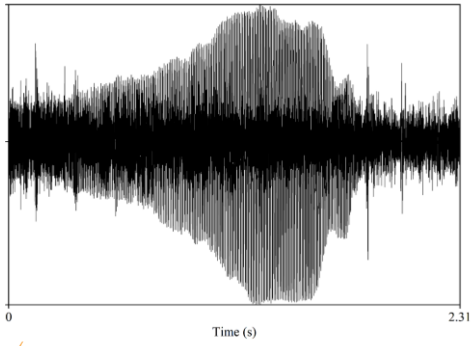








End to End Learning in Black Box

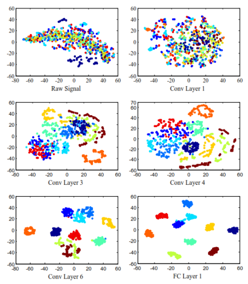


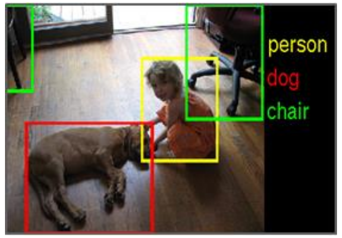
Black Box

- Feature extraction,
- selection,
- Unsupervised Learning

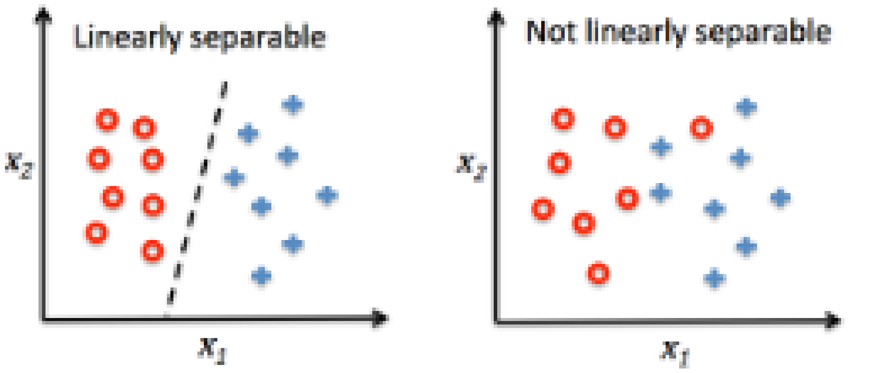
Decisions

Basic Processes: Classification and Regression



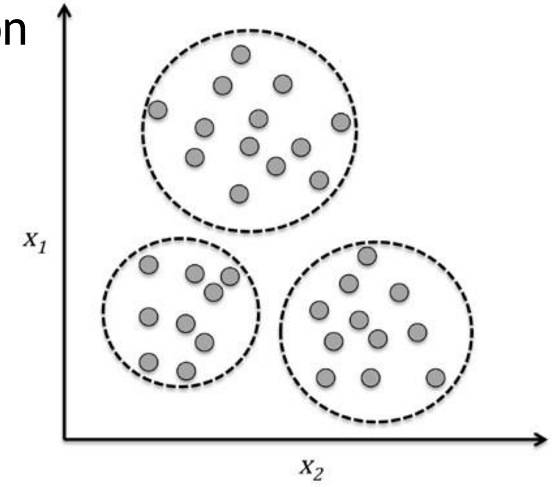


Classification : Prediction of Categorical variables (Labels)



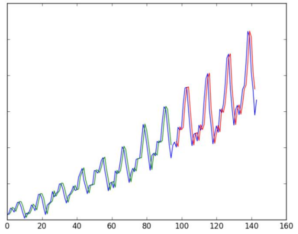
Multi-class Classification

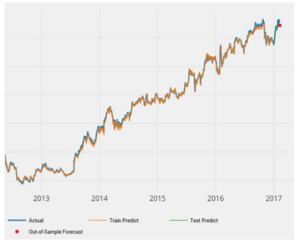
- Inter class: Maximum separation
- Inter class: Minimal variance

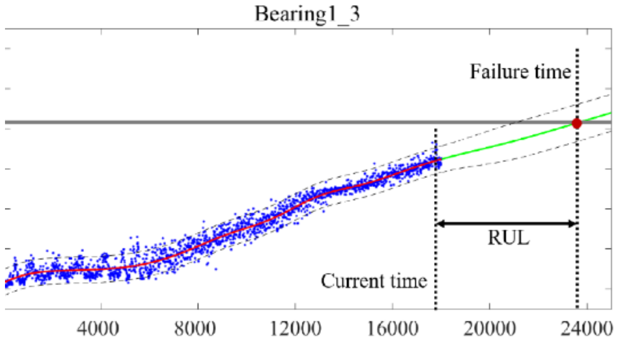


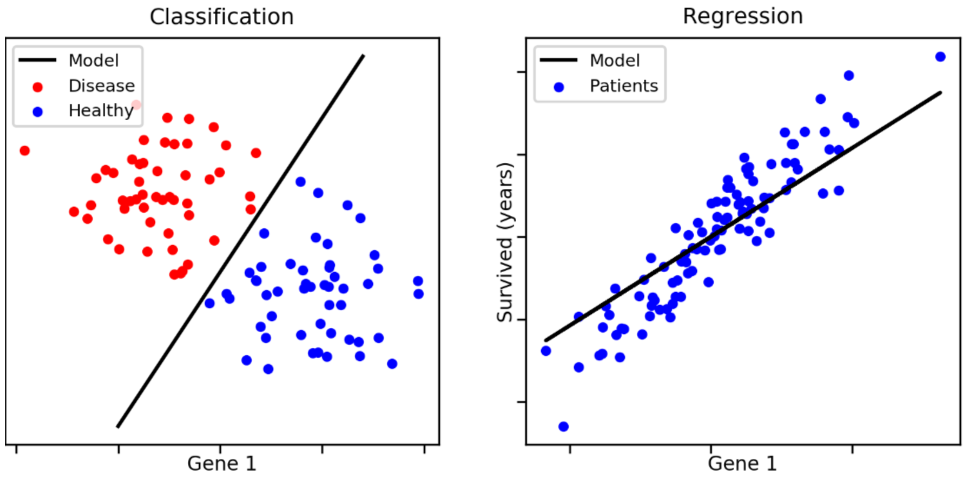
Basic Processes: Regression

Regression: Prediction of numerical or continuous output variables

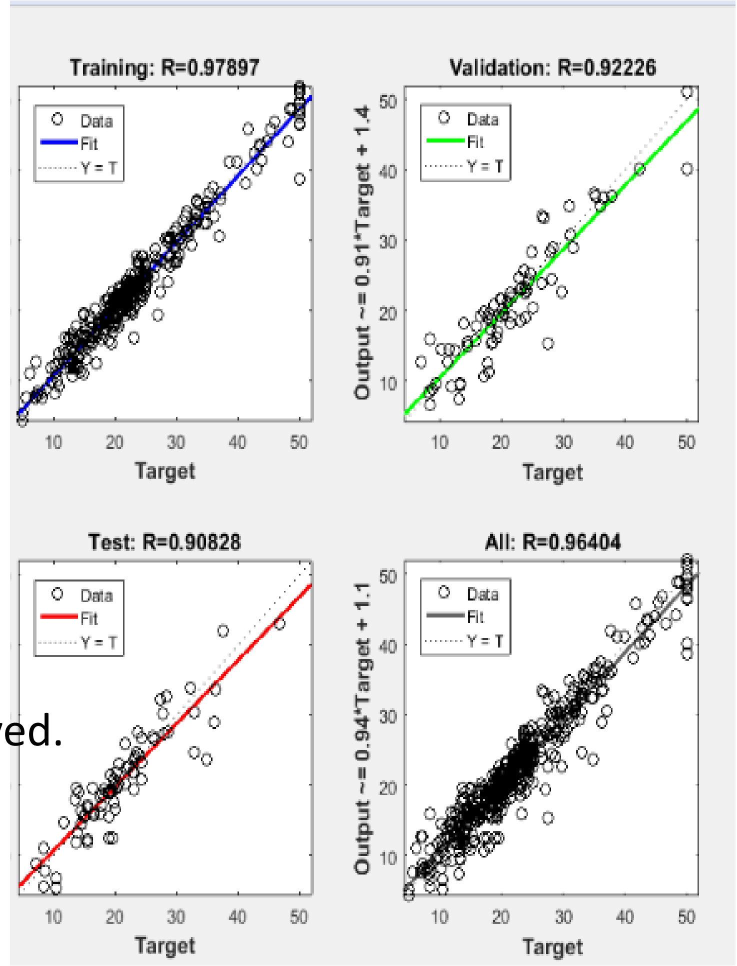








Source: Personal tutorials, also see: Park et al. 2015, Nature genomics



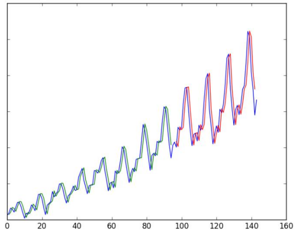
- Forecasting of object based upon the past dynamics (behavior), historical trends observed.
- Sequence to sequence Model → next sequence prediction, long time prediction.

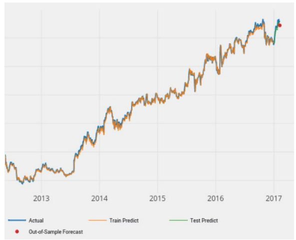
POLYTECH NANCY
UNIVERSITÉ DE LORRAINE
CRAN

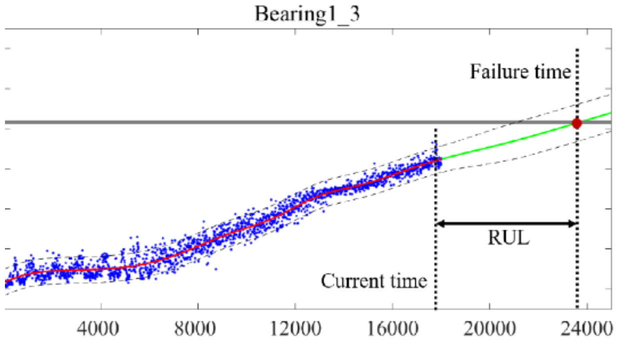
Basic Processes: Regression

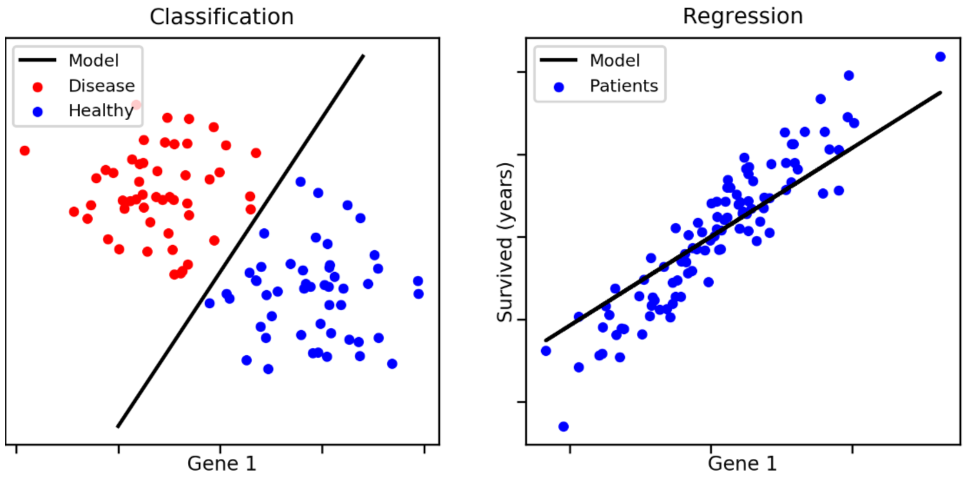
Regression: Prediction of numerical or continuous output variables

$$y = w_1x_1 + w_2x_2 + w_3x_3 ..... + b$$
$$= \sum_{i=1}^{m} w_i x_i + b$$

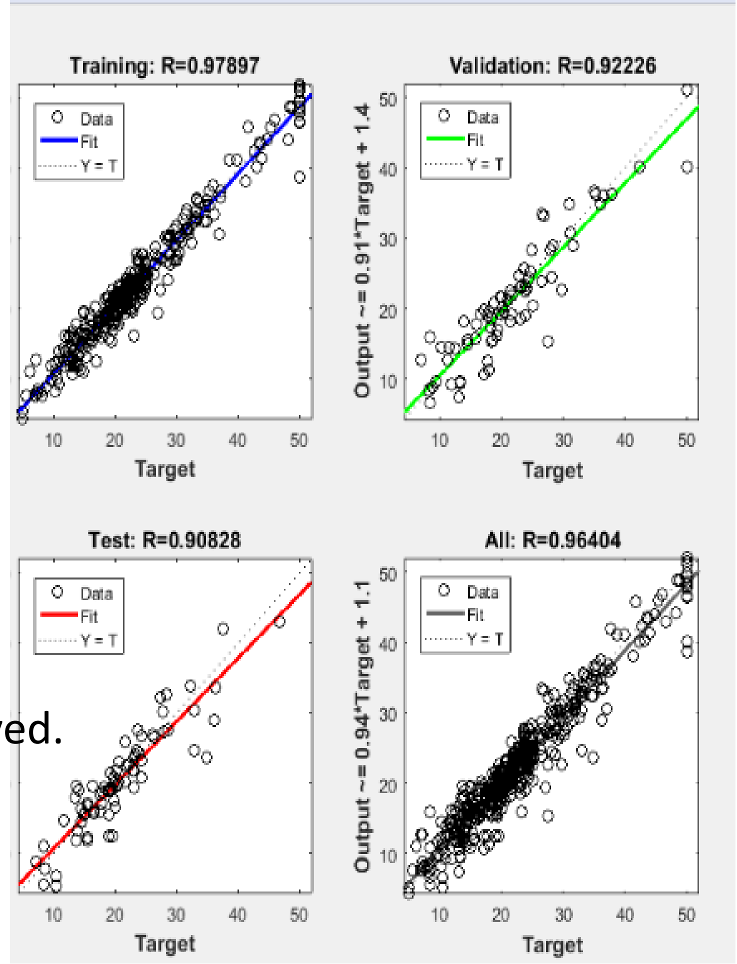








Source: Personal tutorials, also see: Park et al. 2015, Nature genomics



NETFLIX

- Forecasting of object based upon the past dynamics (behavior), historical trends observed.
- Sequence to sequence Model → next sequence prediction, long time prediction.

POLYTECH NANCY   UNIVERSITÉ DE LORRAINE   CRAN

Ordinary Least Square (OLS) based regression

$(x_i, y_i); \quad i = 1,2,3 \dots n$

- Error term $e_i = y_i - (\widehat{c + mx_i})$

- Objective : Minimize the sum of square of errors

$$e_1 + e_2 + e_3 \dots e_n$$

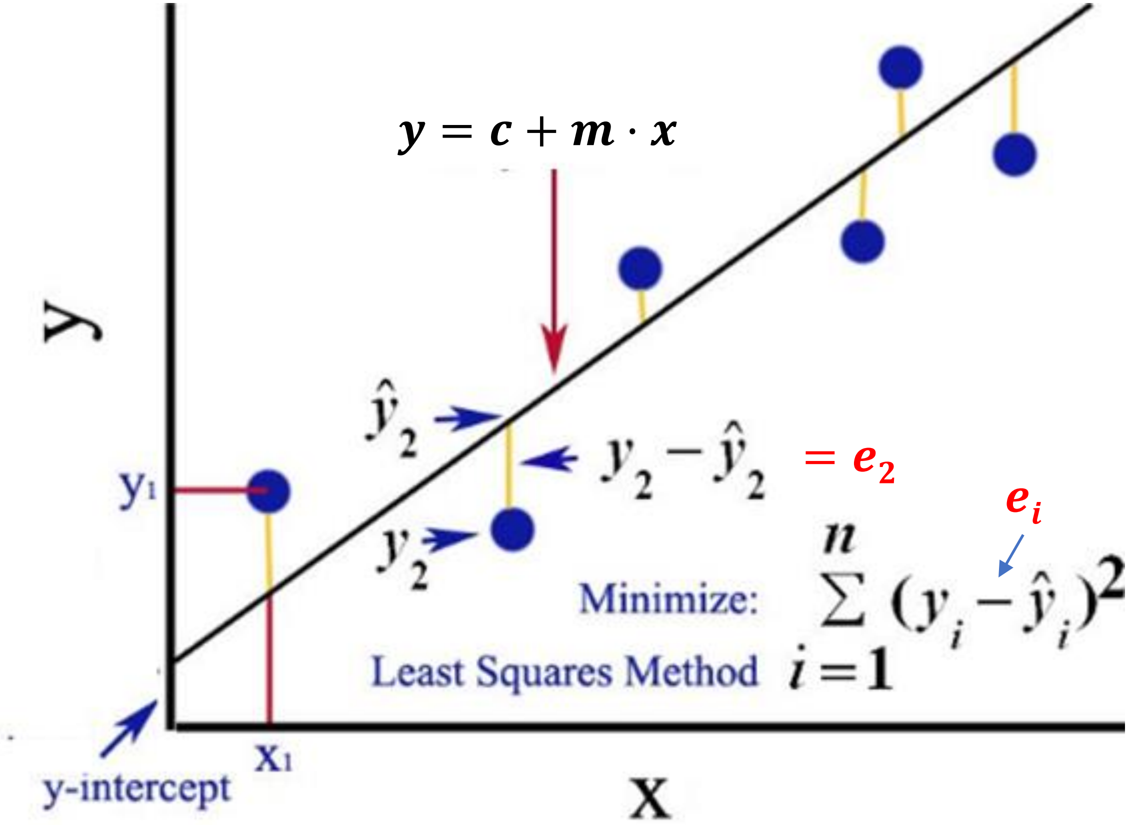
$$\sum_{i=1}^{n} {e_i}^2 = \sum_{i=1}^{n} \left(y_i - (\widehat{c + mx_i})\right)^2$$

$$\hat{m} = \frac{\sum_{i=1}^{n}(x_i - \bar{x}) \times (Y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

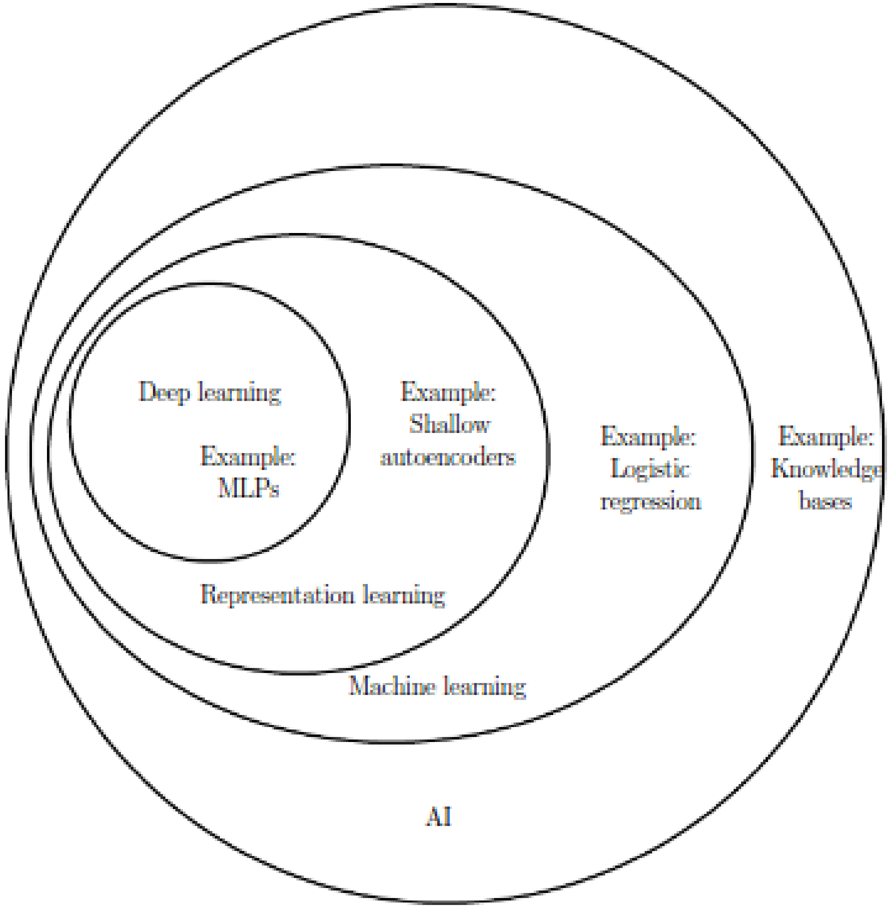
$$\hat{c} = \bar{y} - \hat{m}\bar{x}$$

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}$$



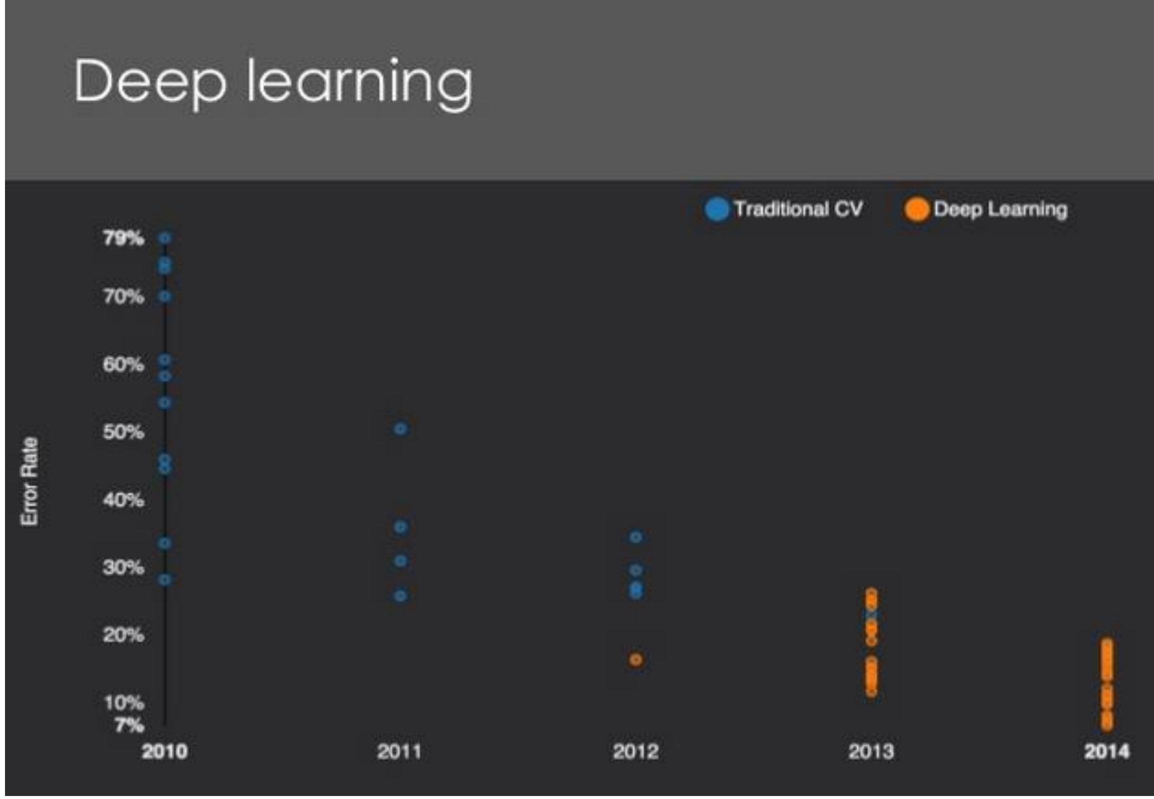
Relation AI, ML and DL



Source: Deep Learning

Machine Learning techniques for AI

Naïve Bayes,
Kernel Density Estimation
Rule Based,
Decision Trees,
Random Forests
Genetic Algorithms



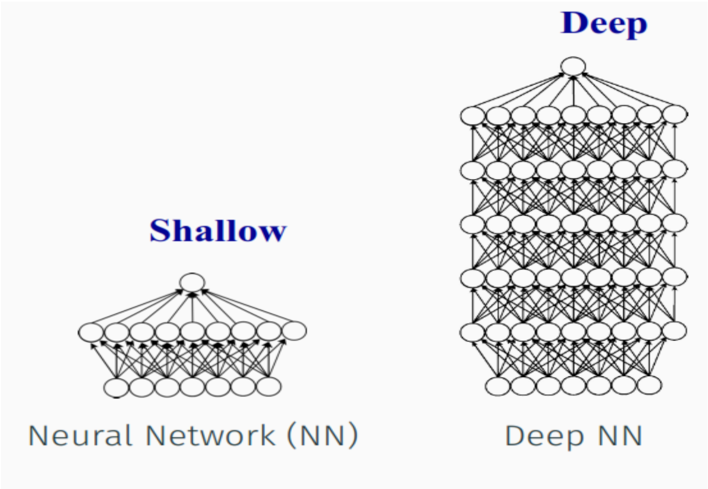
Support vector machines (1990-2007): very promising, better than NNs….till 1998.

Neural networks (NNs) (1960-1986, 1986-1998, 1998-2007)

Deep Neural Networks (1998,DNNs) : CNNs revolutionized NN based works,

Enter 2007,

- Availability of data & data acquisition methods,
- GPU based distributed calculations
- Huge community of developers
- Surge in DNN



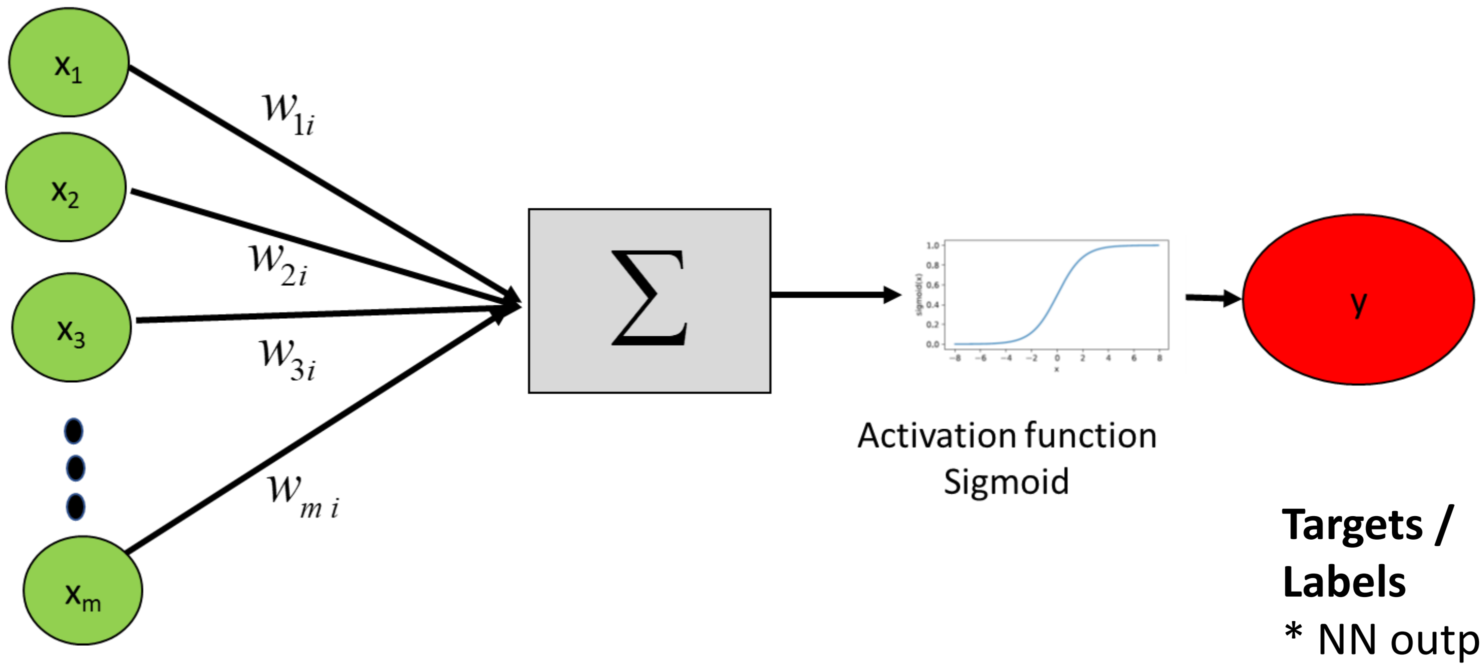
POLYTECH NANCY UNIVERSITÉ DE LORRAINE CRAN

Learning Using Deep Neural networks : Supervised Learning

In this lecture, we look at **Neural Networks** and

Mechanism of **Supervised type learning** .

Input Features/
Features
Input data/
Inputs



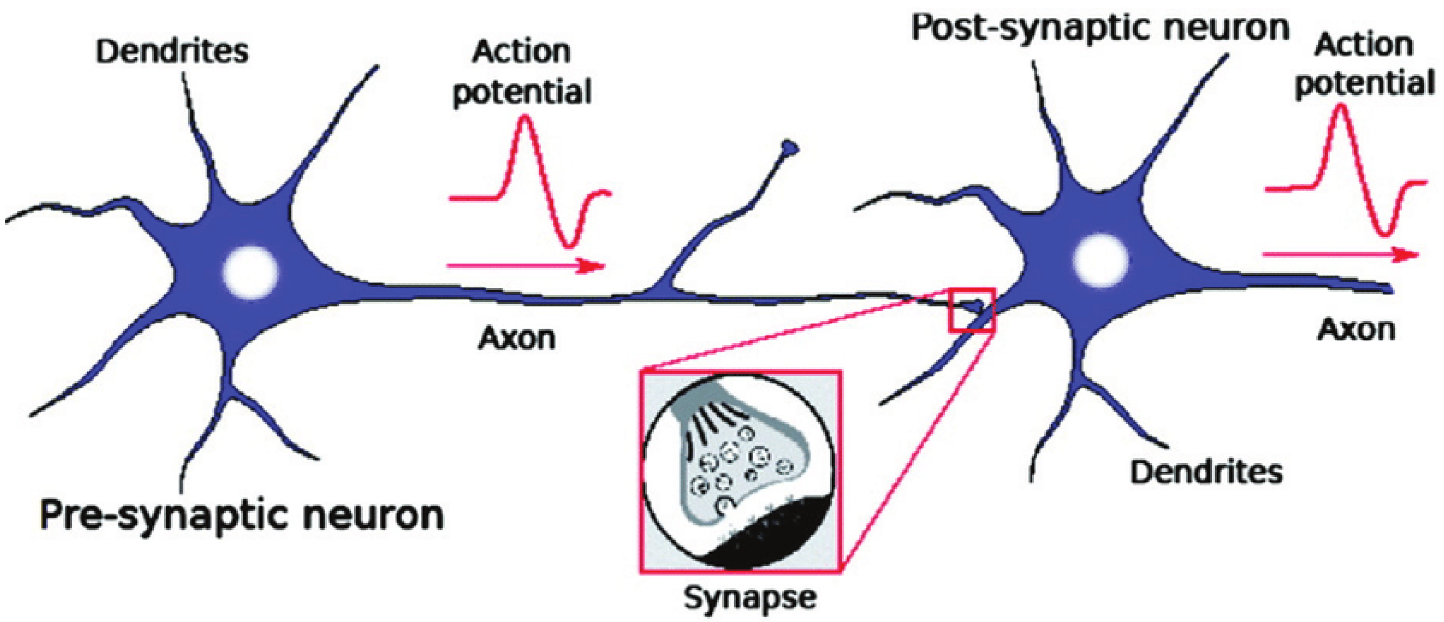
Targets /
Labels

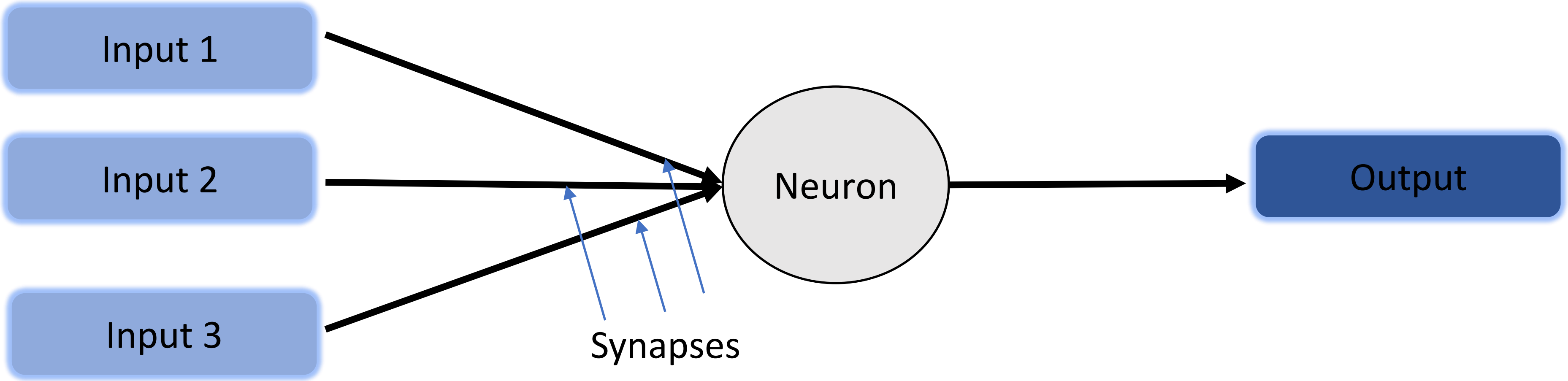
* NN output may or may not be equal to the target. Why?

The Neuron

The Neuron

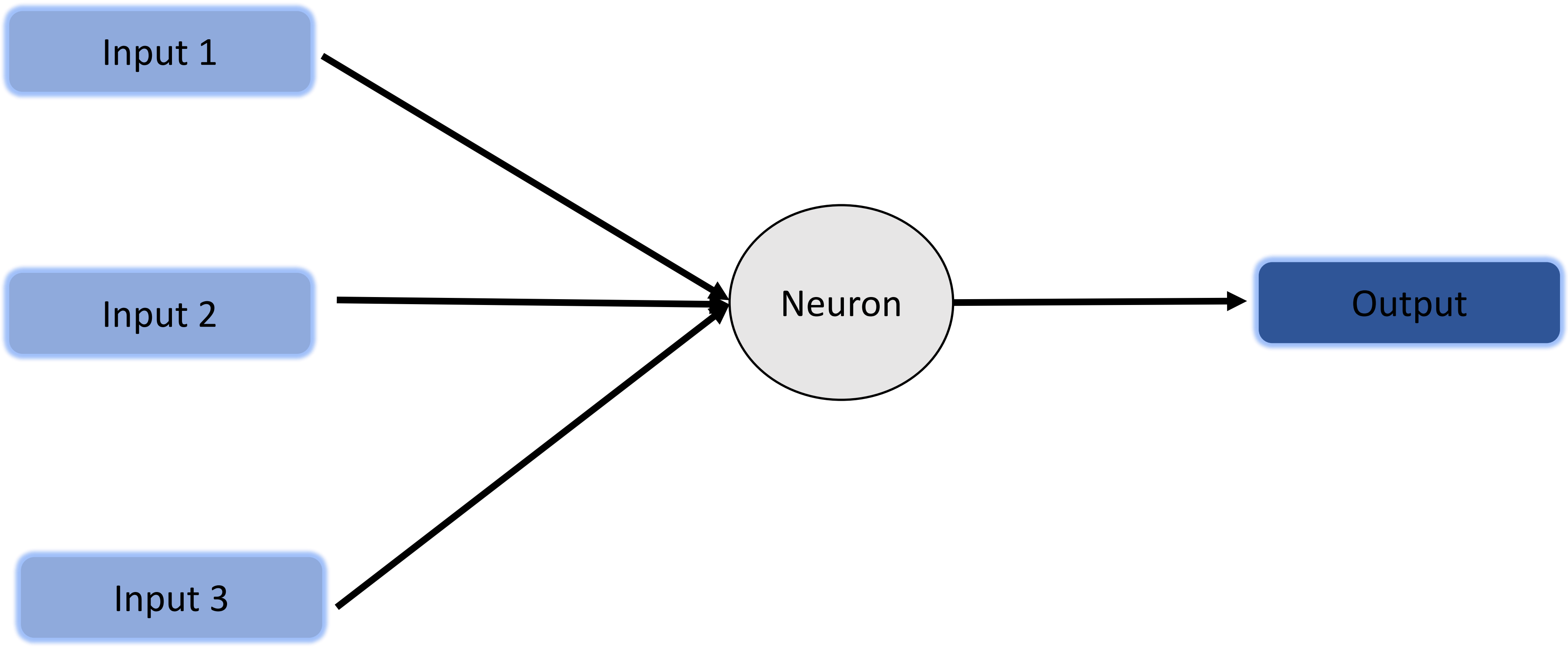
- A neuron only fires if its input signal exceeds a certain amount (the threshold) in a short time period.



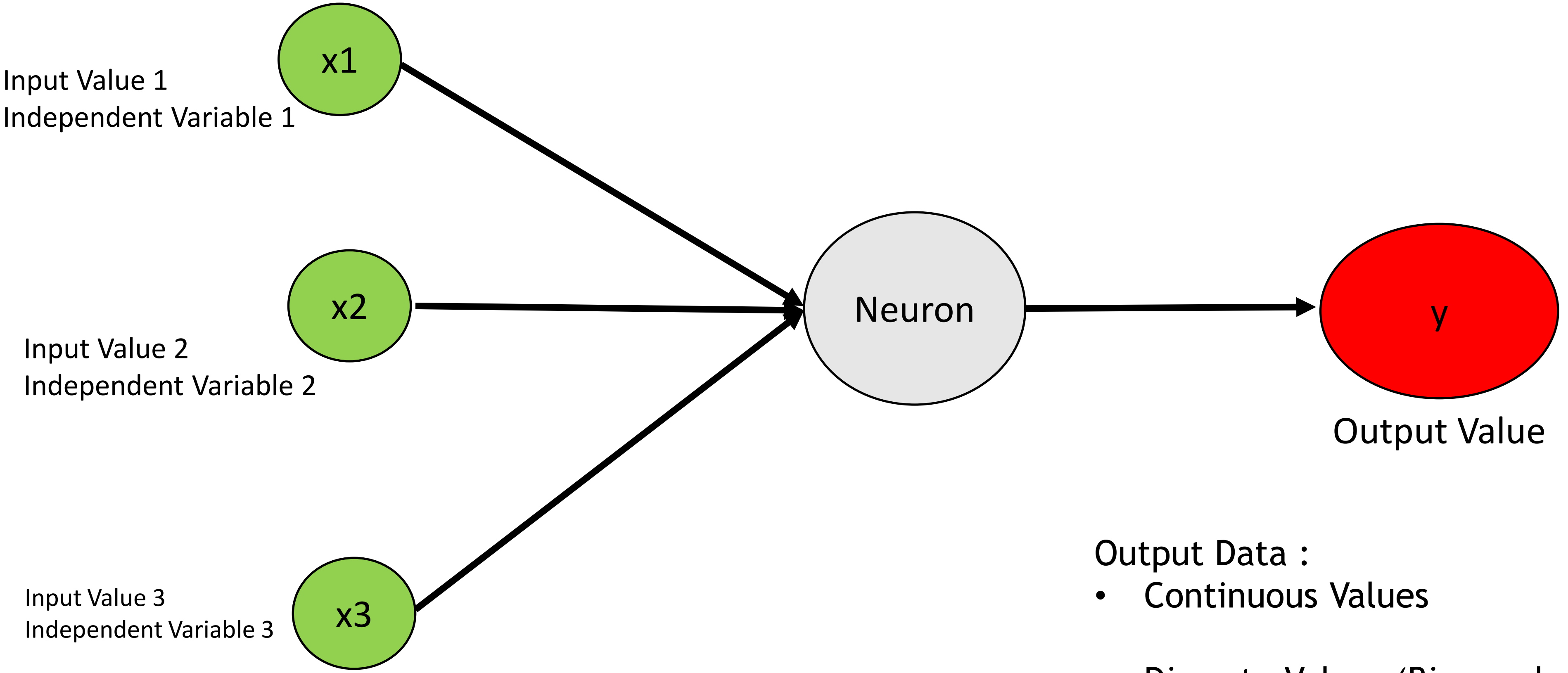


Huang, Anping, et al. 2017.

The Neuron



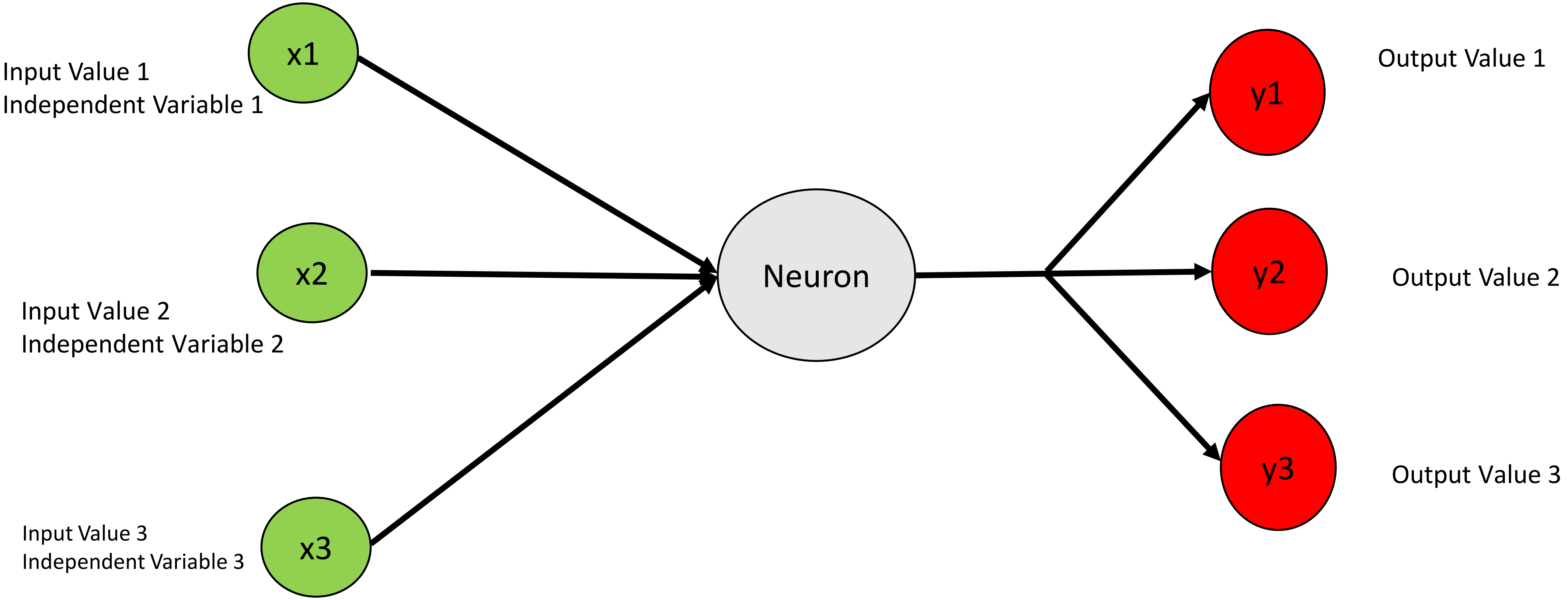
The Neuron



Output Data :
- Continuous Values
- Discrete Values (Binary classes → Yes/No..)
- Categorical Variables (very small, small, large, very Large)

- Standardization of input data (Same Scale)
- Data preprocessing

The Neuron



- Standardization of input data (Same Scale)
- Data preprocessing

Single Observation

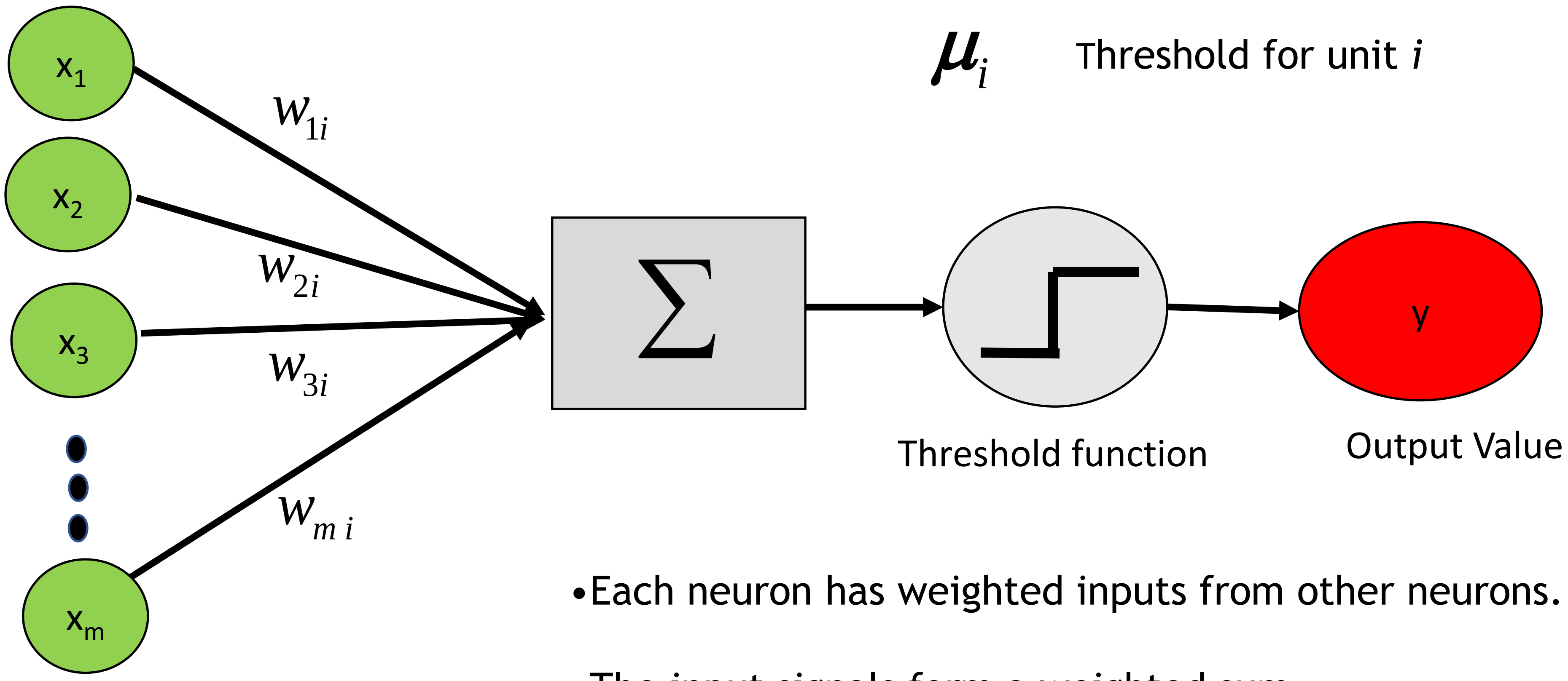
Same Observation
(Input data , Output Lable

Single Observation

The Neuron: Basic Perceptron

$w_{m\,i}$ Synaptic weight from unit *m* to unit *i*

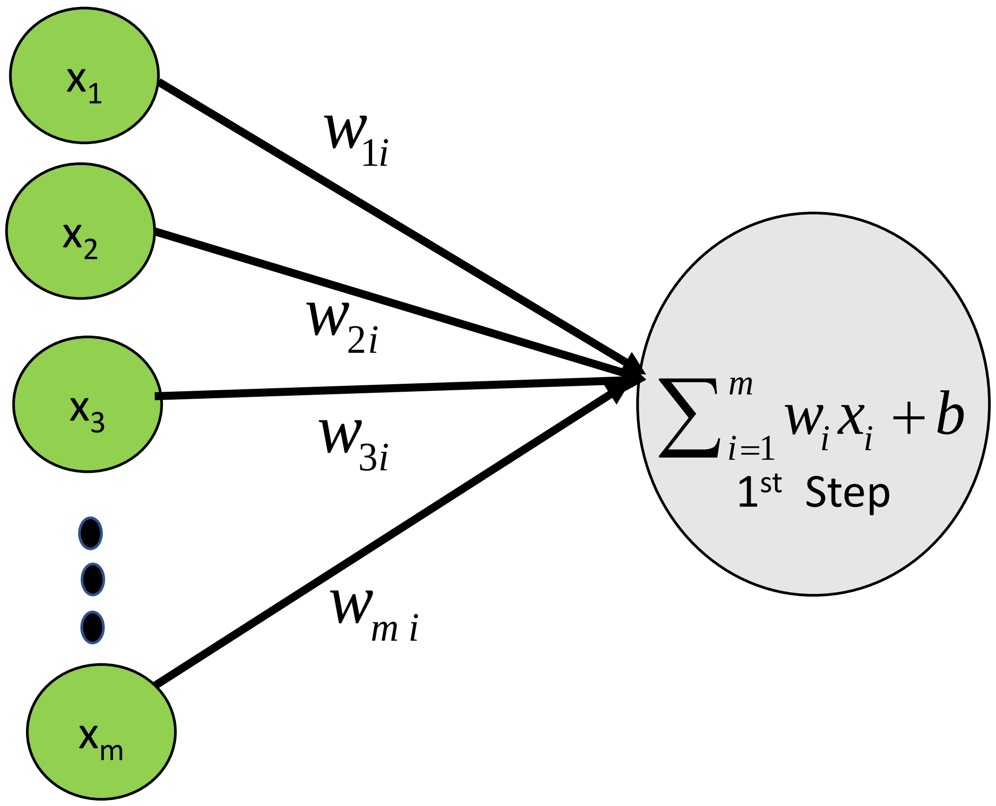
μ_i Threshold for unit *i*



- Each neuron has weighted inputs from other neurons.
- The input signals form a weighted sum.
- If the activation level exceeds the threshold, the neuron "fires".
- Each neuron has a threshold value.

Artificial Neural Networks (ANNs)

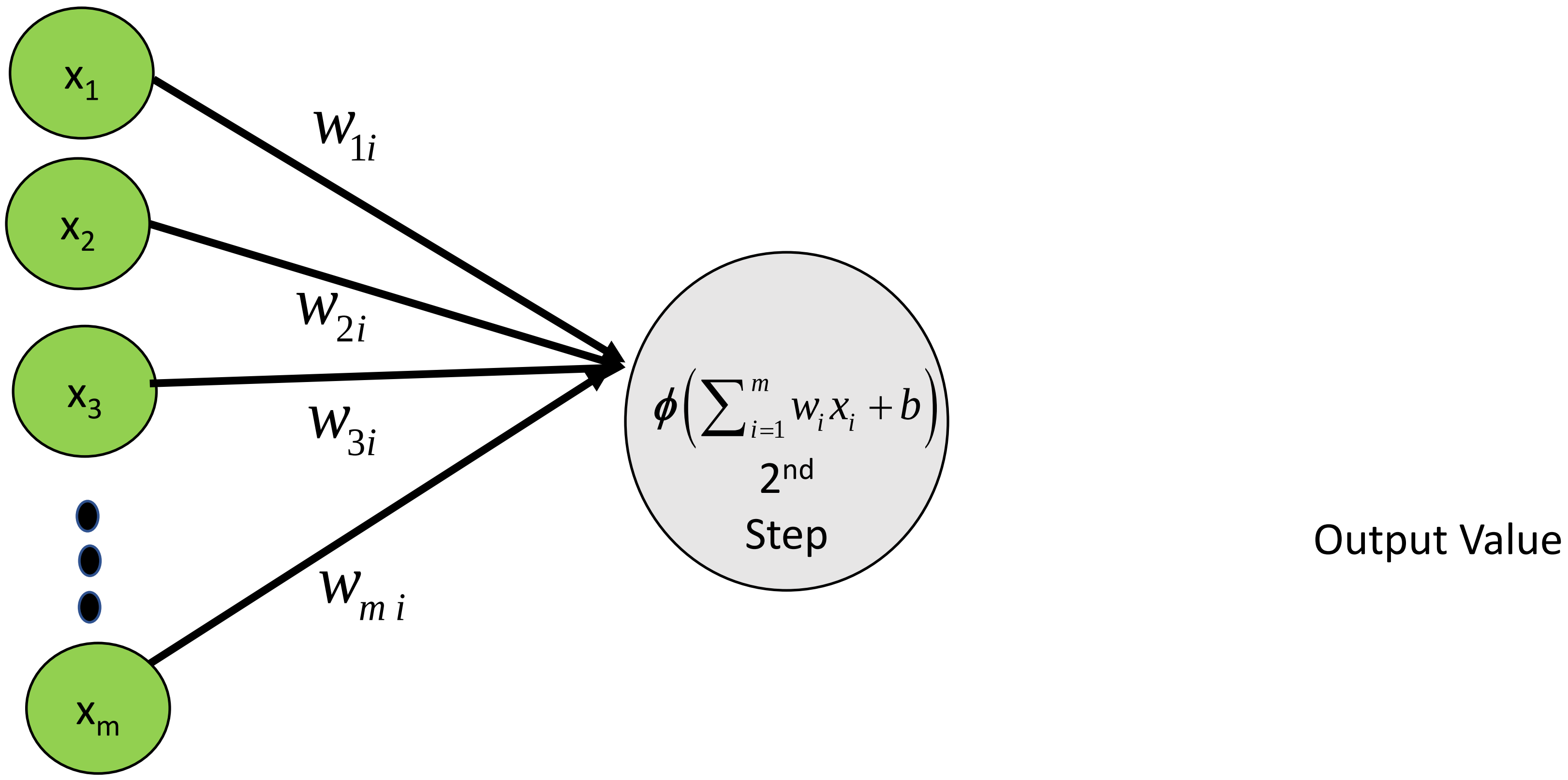
$$y = w_1x_1 + w_2x_2 + w_3x_3 ...... + b$$
$$= \sum_{i=1}^{m} w_i x_i + b$$



Output Value

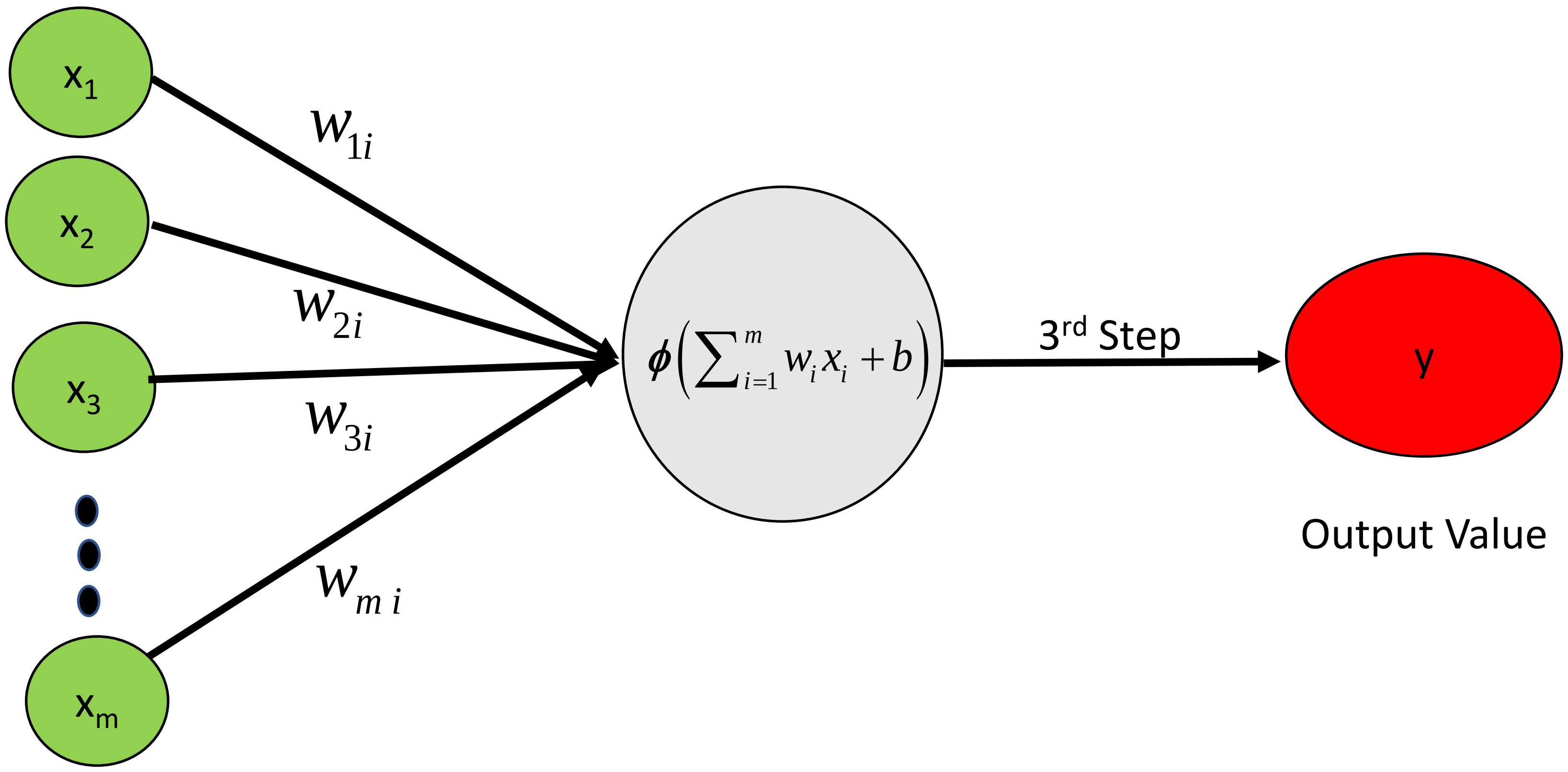
- Each hidden or output neuron has weighted input connections from each of the units in the preceding layer.
- The unit performs a weighted sum of its inputs, and subtracts its threshold value, to give its activation level

Artificial Neural Networks (ANNs)

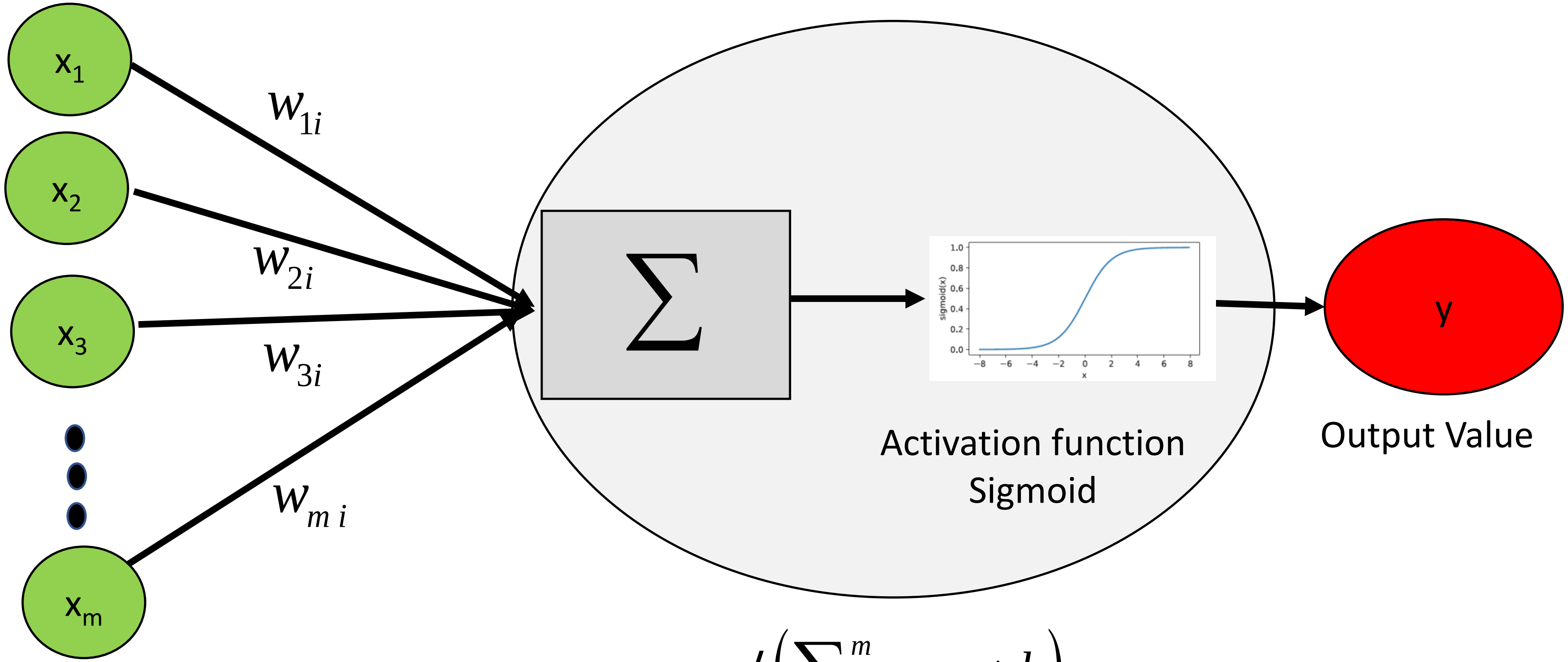


- Activation level is passed through an activation function $\phi(x)$ to determine output

Artificial Neural Networks (ANNs)

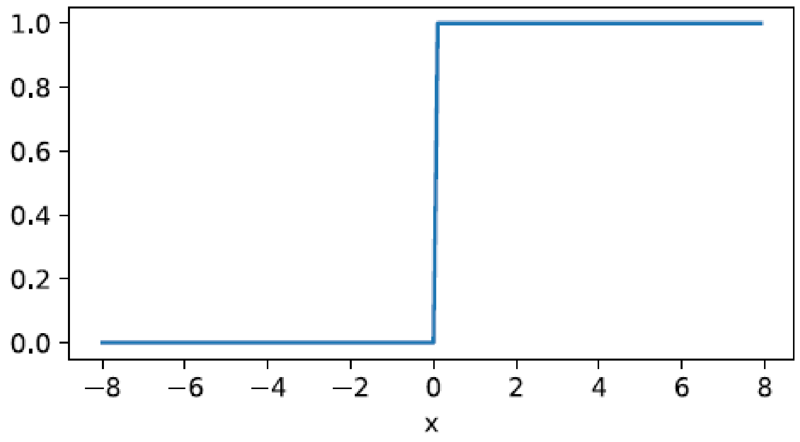


The Artificial Neural Network (ANN)

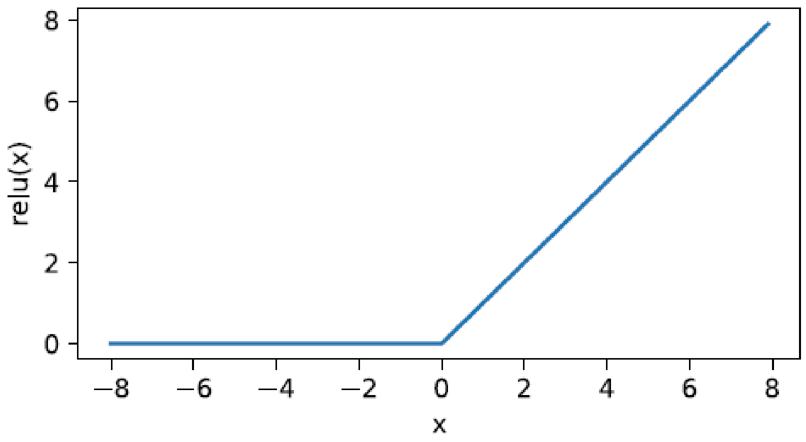


$$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$$

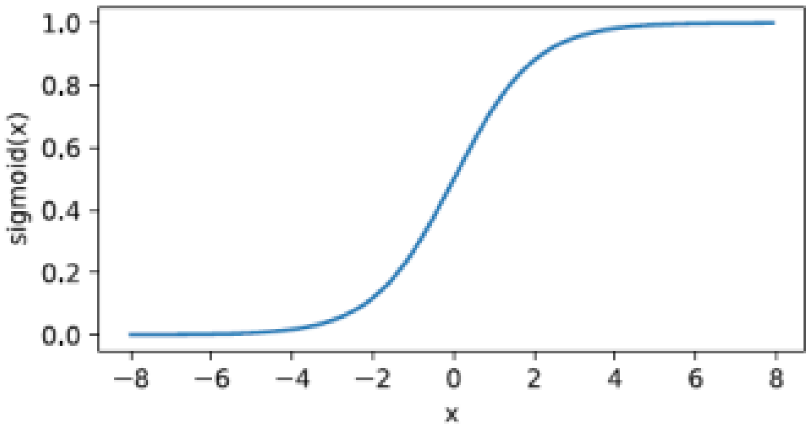
Activation functions (discussed later)



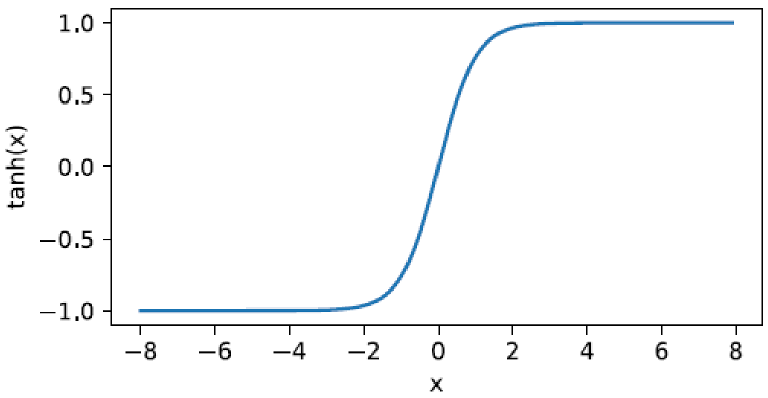
Threshold function (binary step function)



ReLu (Rectified Linear Unit)

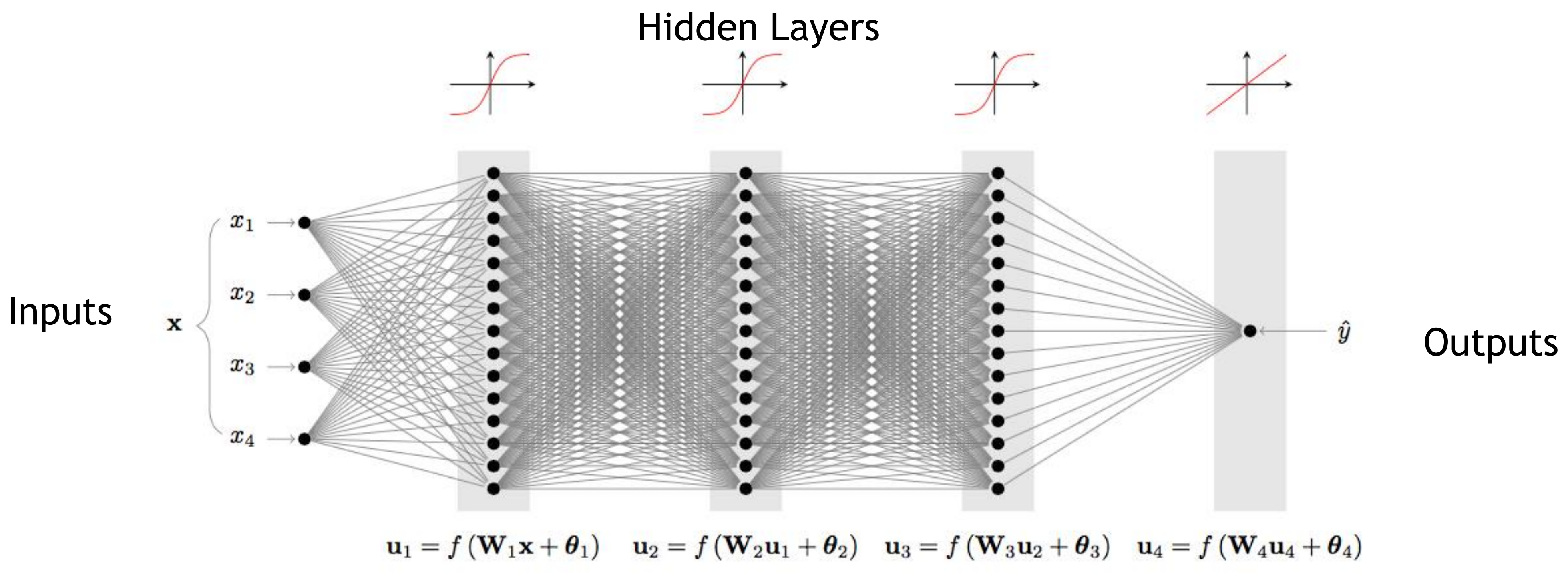


Sigmoid function



TanH / Hyperbolic Tangent

Multi Layered (Deep) Feed Foreword Neural Networks

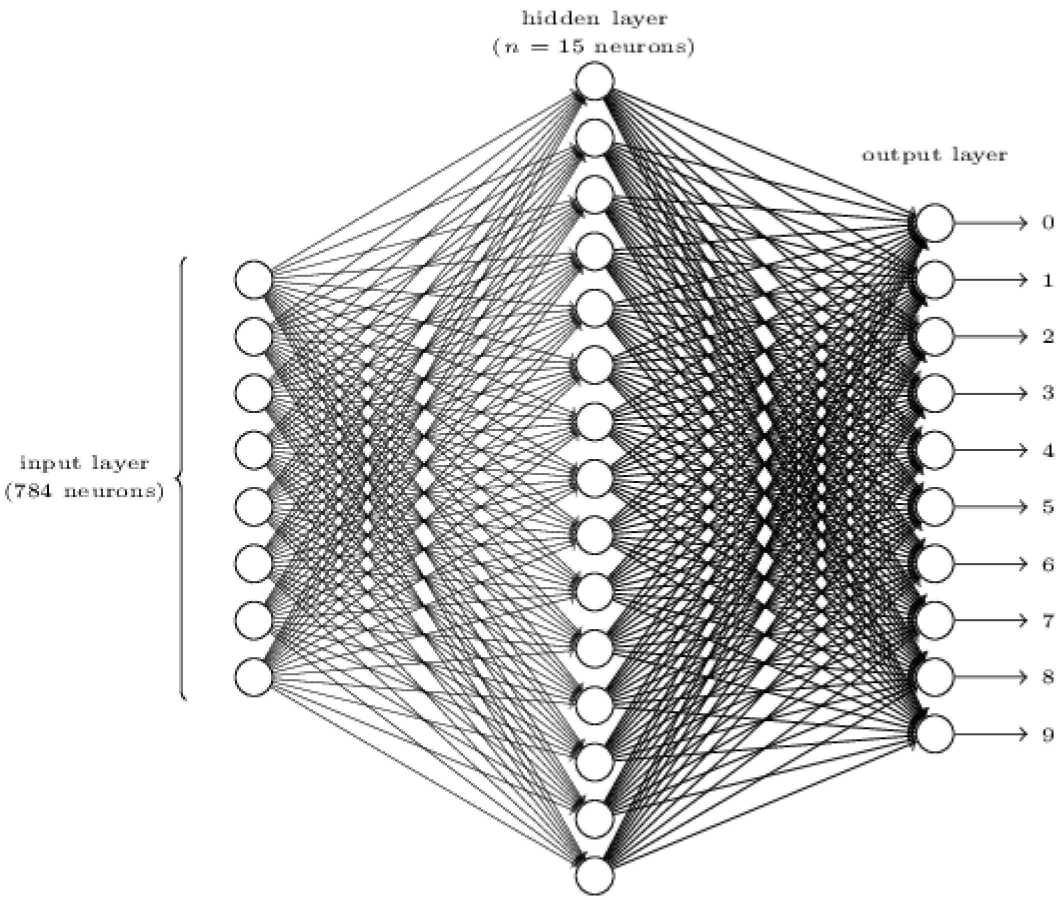


- These are fully connected layers, **but need not be.**

Multi Layered (Deep) Feed Foreword Neural Networks

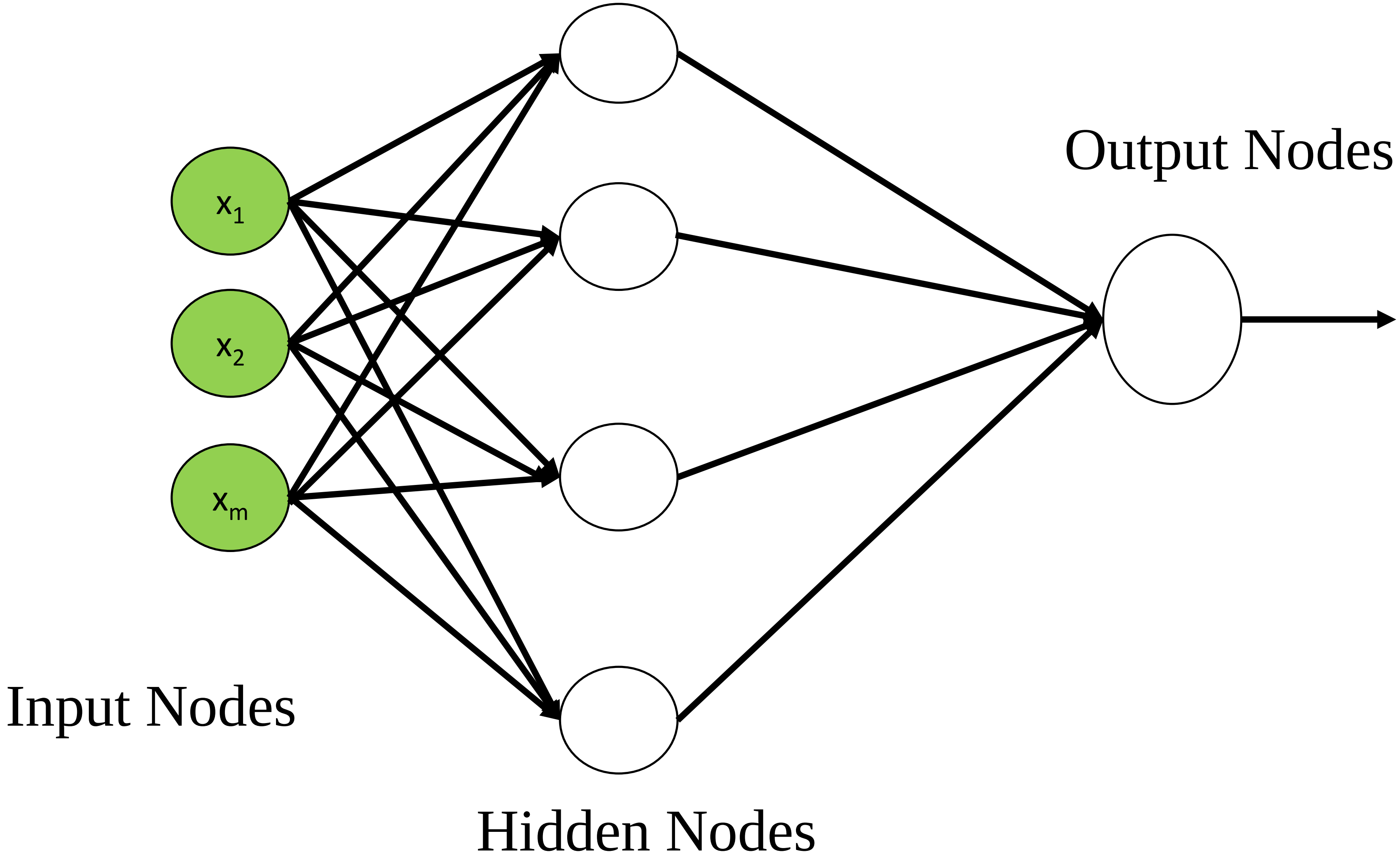
Hidden Layers

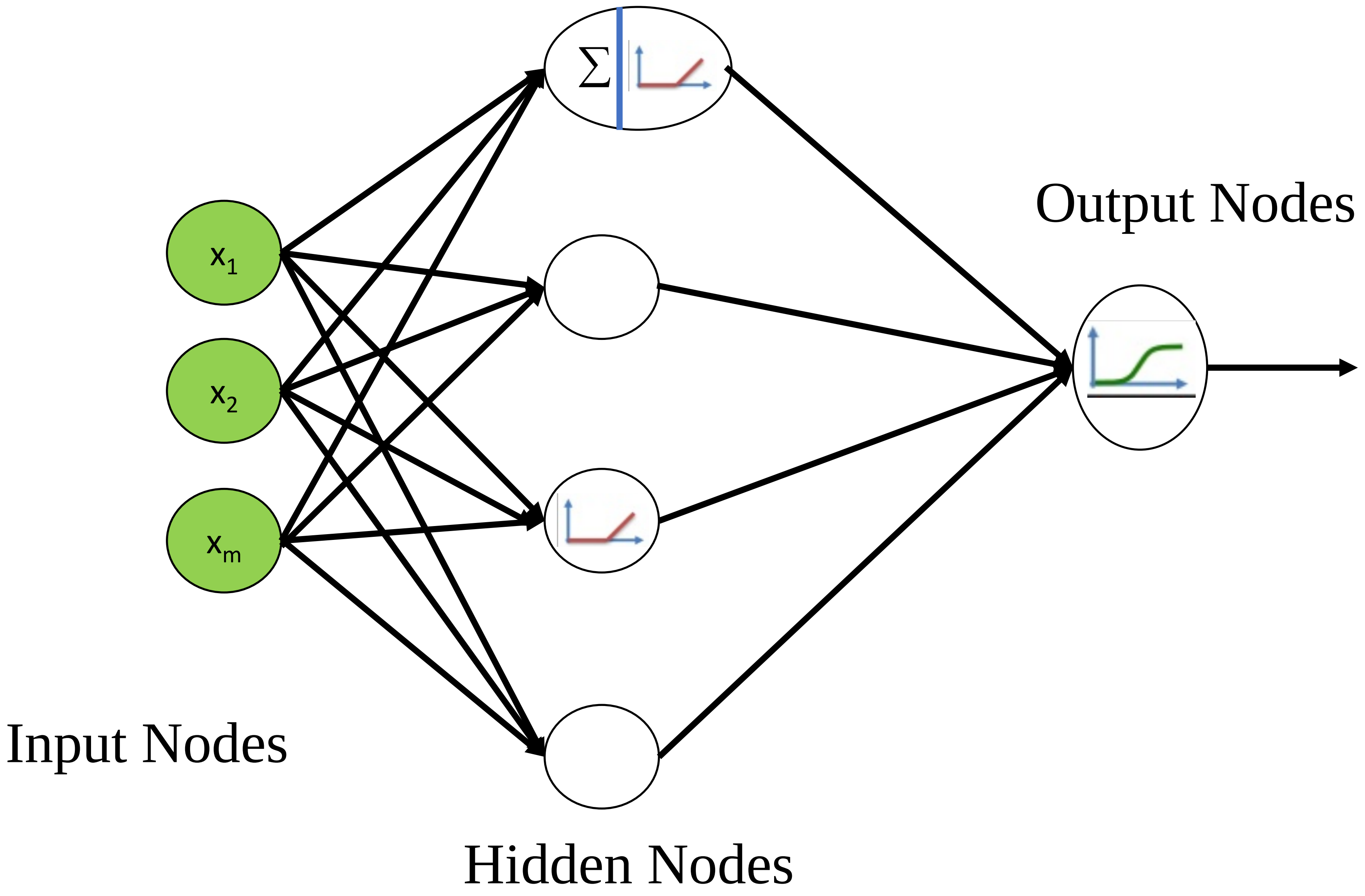
Inputs



Outputs

- Outputs can be multiple (multiple targets). See softmax activation later.

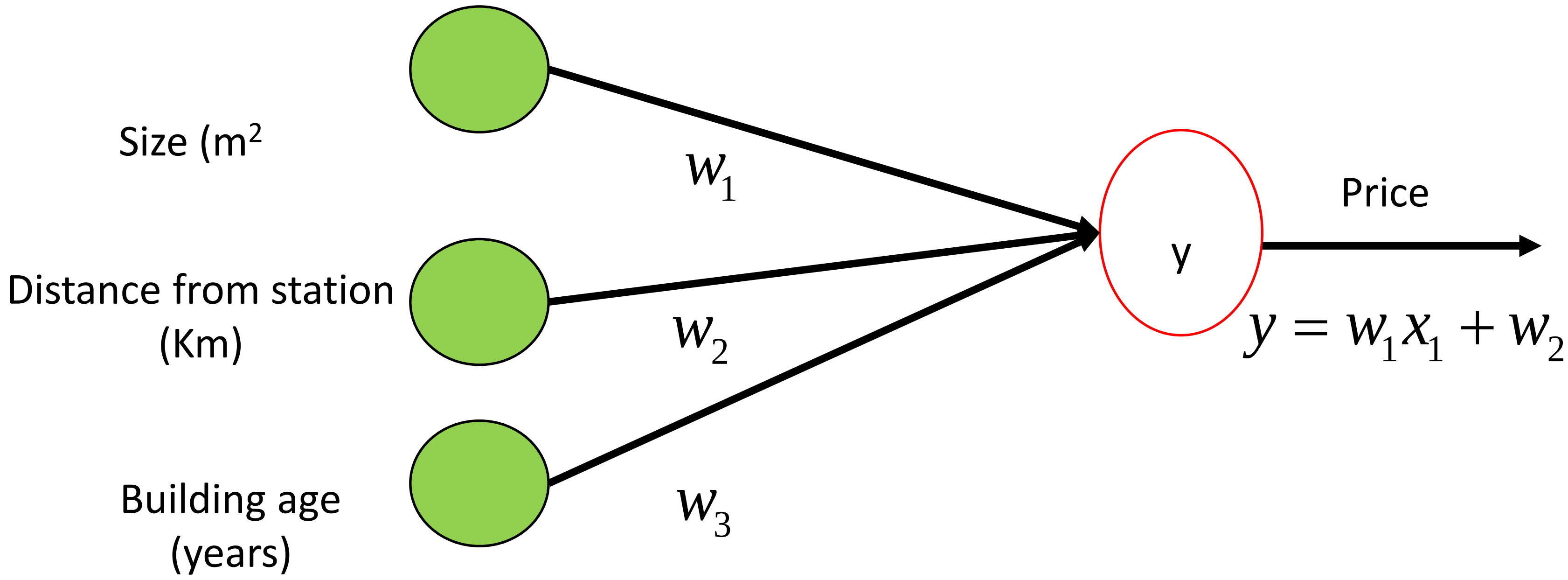

Output Nodes
x_1
x_2
x_m
Input Nodes
Hidden Nodes



Σ
x1
x2
xm
Output Nodes
Input Nodes
Hidden Nodes

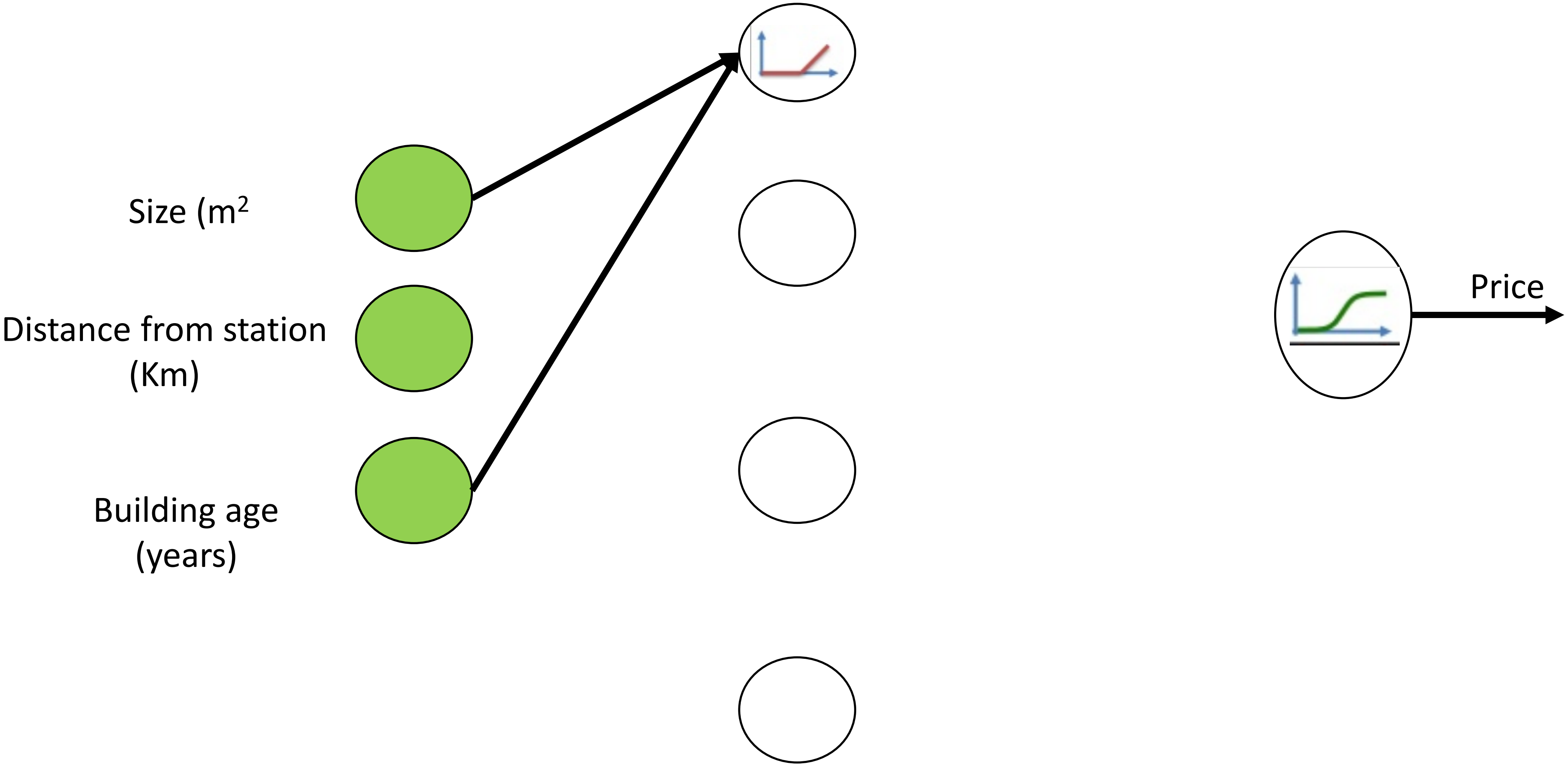
Basic functioning of NNs

How do NNs work : Illustrative Example Apartment Price

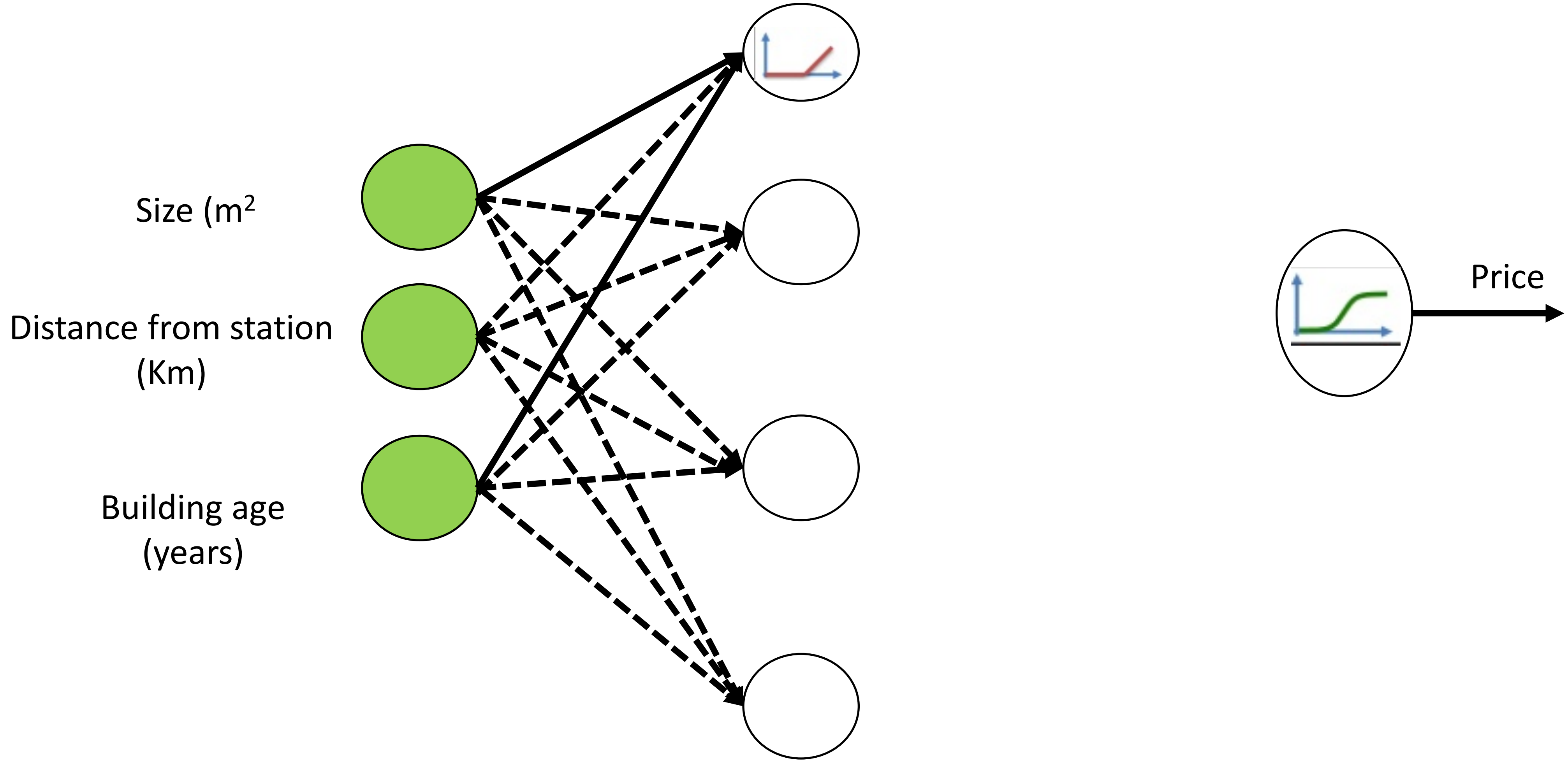


$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

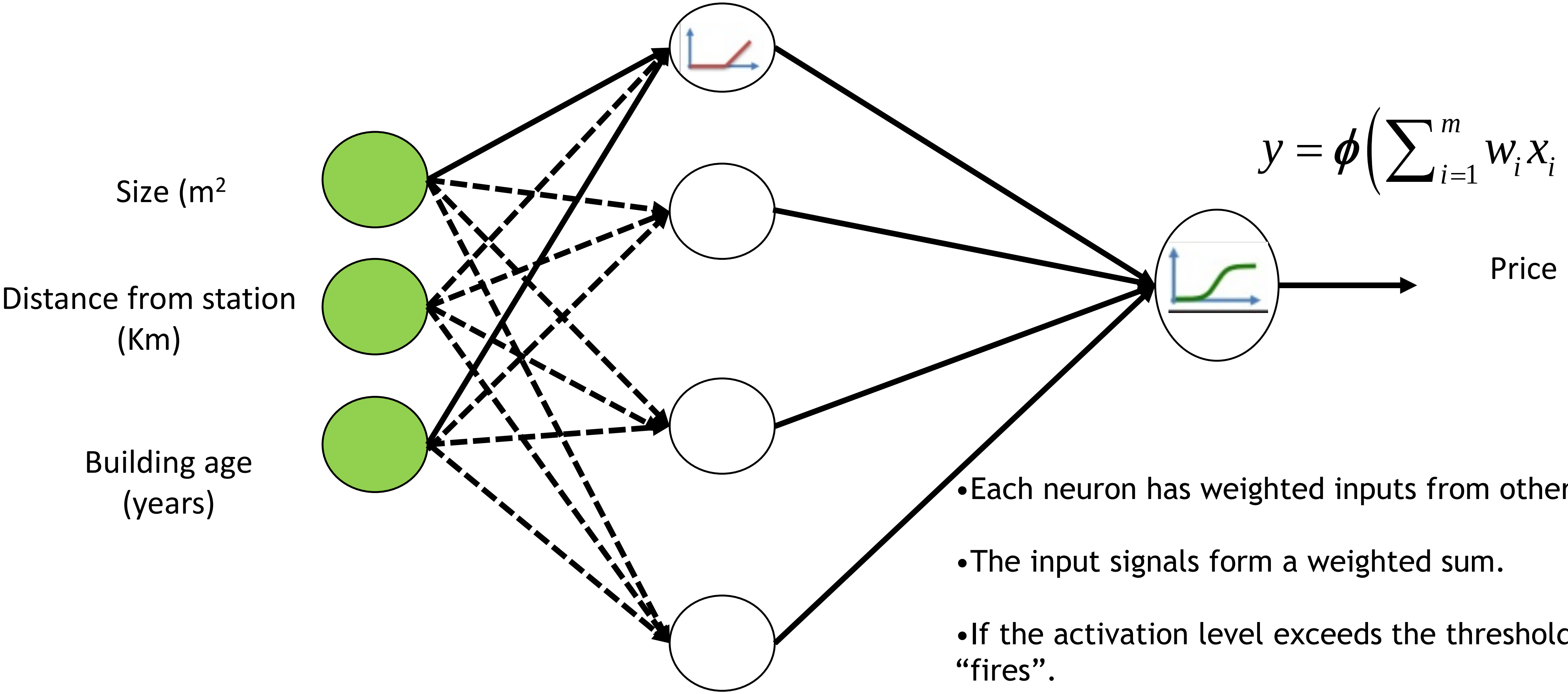
How do NNs work : Illustrative Example Apartment Price



How do NNs work : Illustrative Example Apartment Price



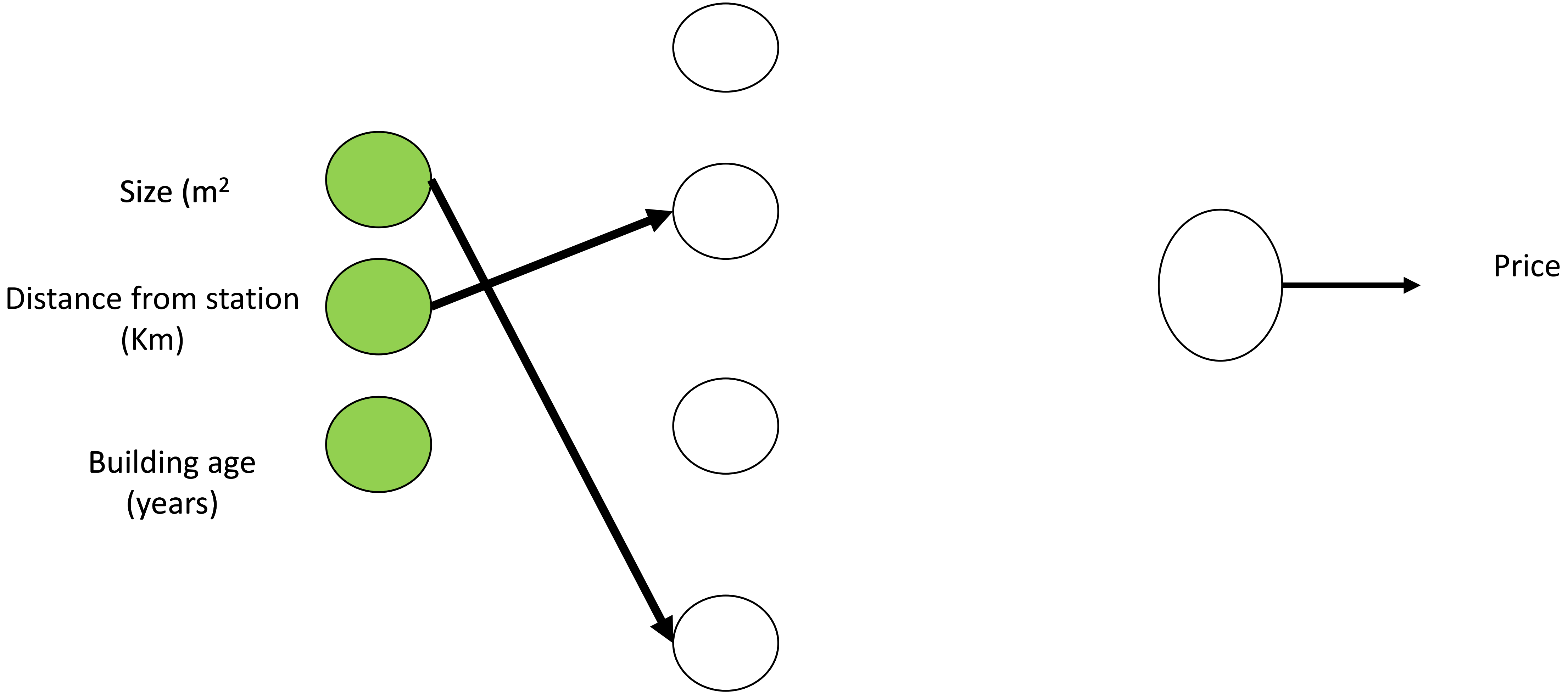
How do NNs work : Illustrative Example Apartment Price



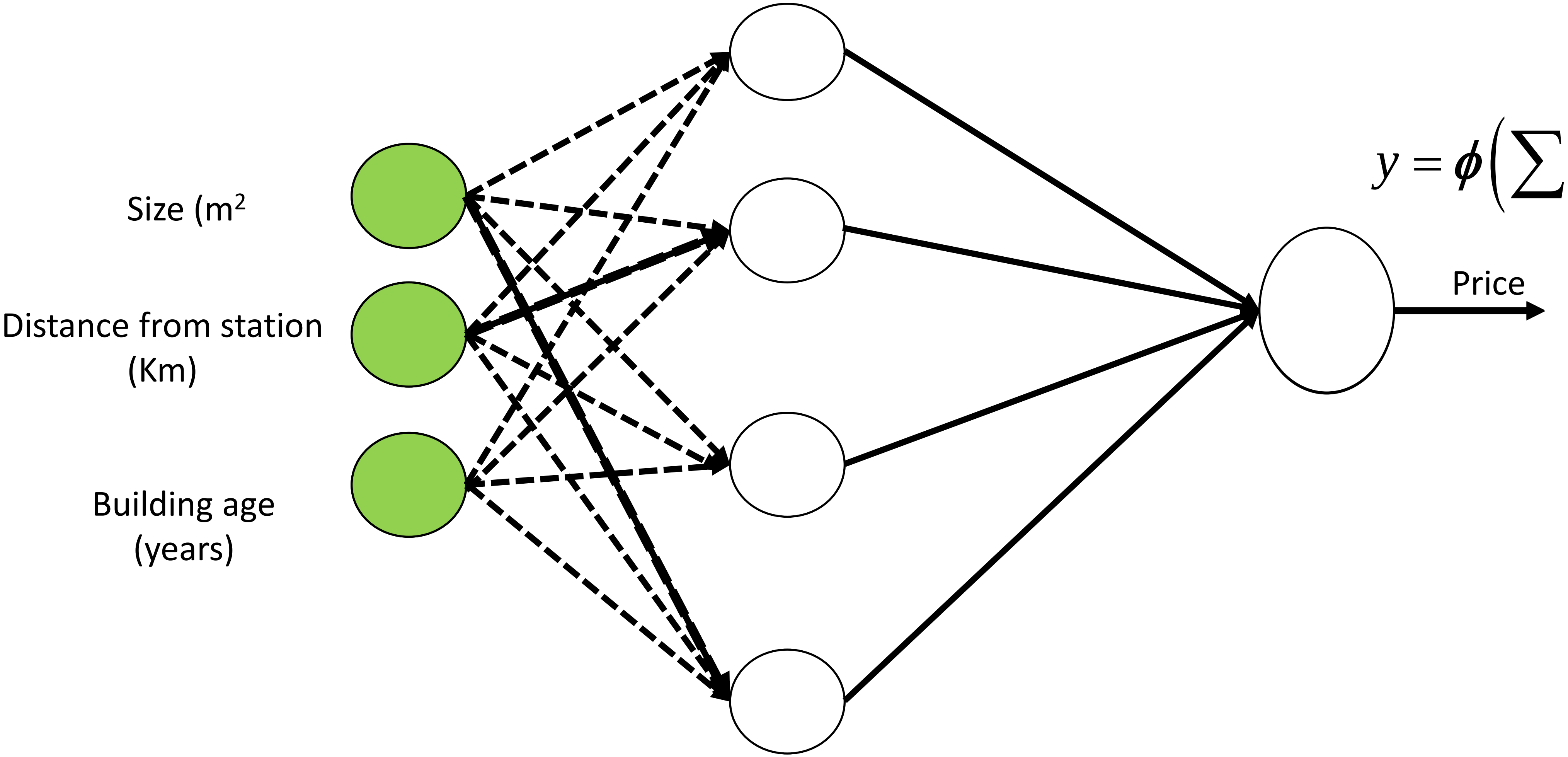
$$y = \phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$$

- Each neuron has weighted inputs from other neurons.
- The input signals form a weighted sum.
- If the activation level exceeds the threshold, the neuron "fires".
- Each neuron has a threshold value.

How do NNs work : Illustrative Example Apartment Price

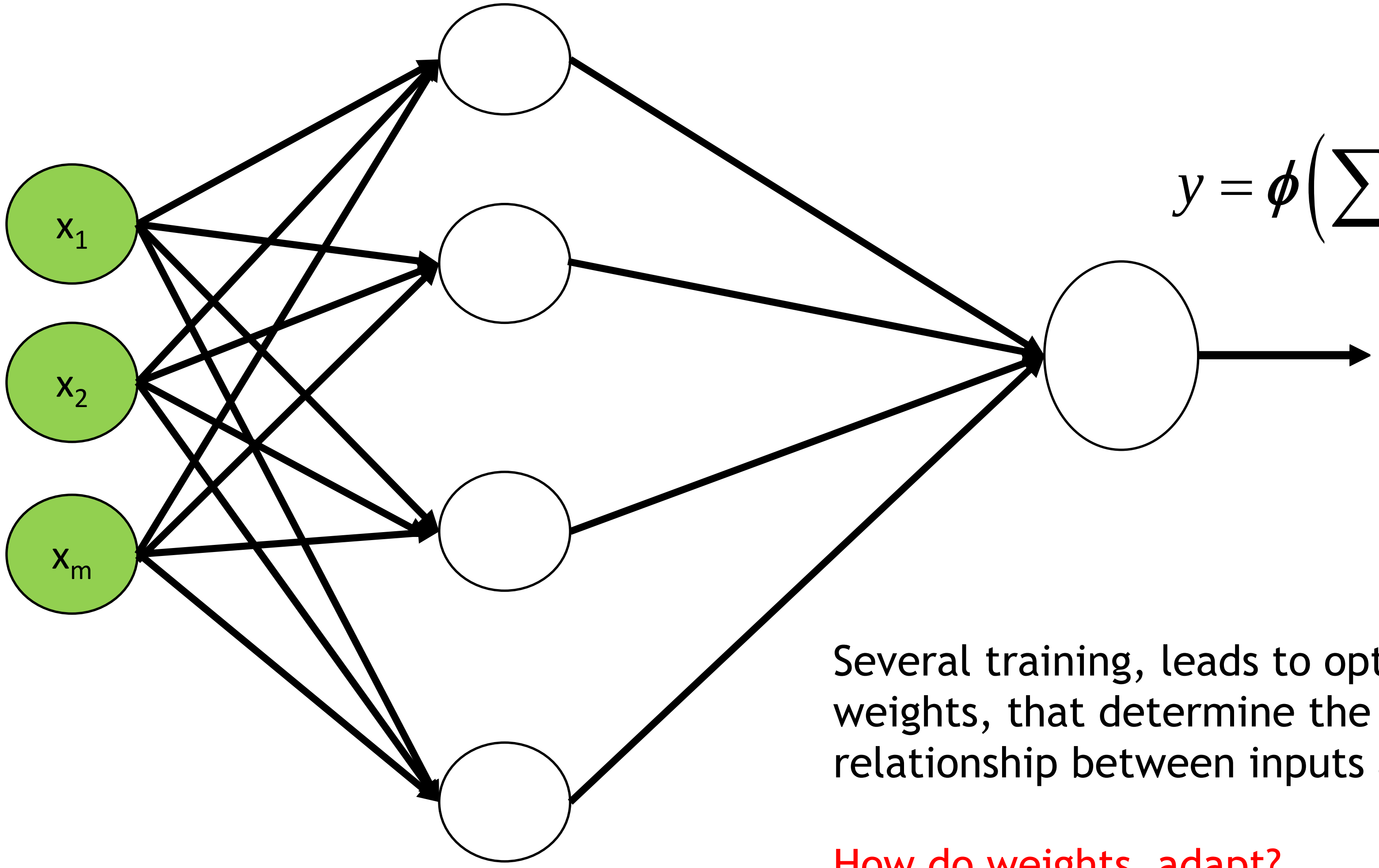


How do NNs work : Illustrative Example Apartment Price



$$y = \phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$$

How do NNs work : Illustrative Example Apartment Price



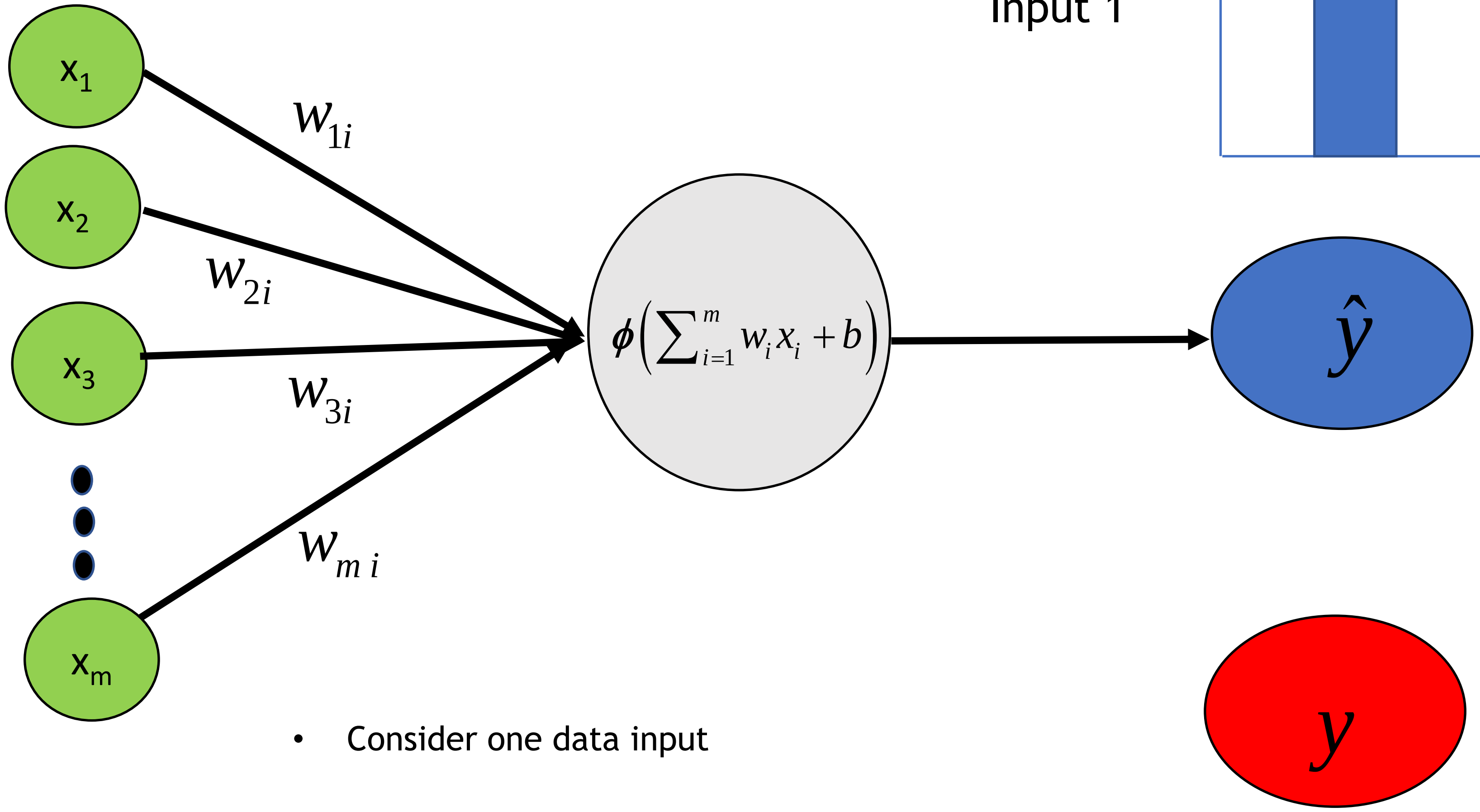
$$y = \phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$$

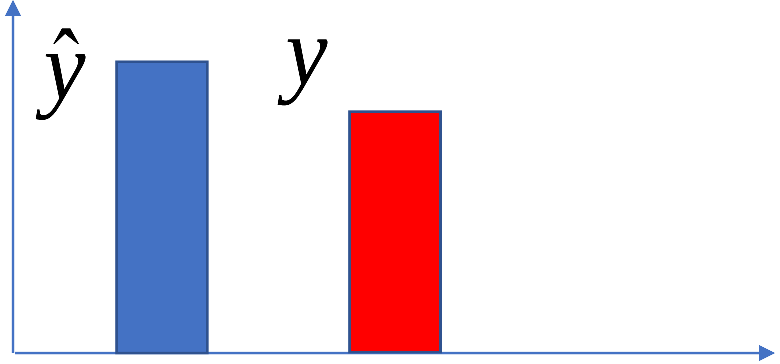
Several training, leads to optimal sets of weights, that determine the non-linear relationship between inputs and targets.

How do weights adapt?
Or,
How do NNs learn?

Learning in NNs

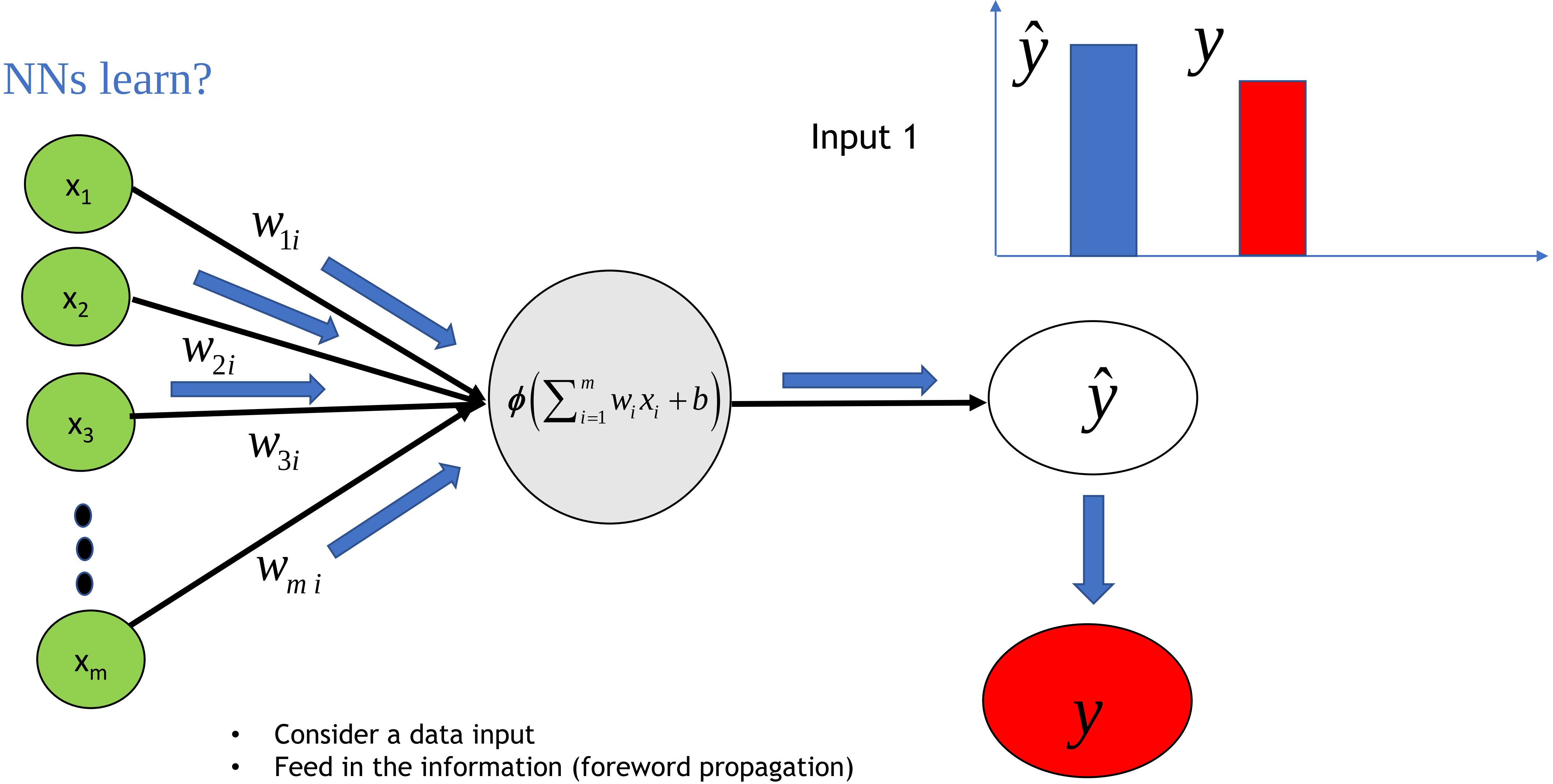
How do NNs learn?





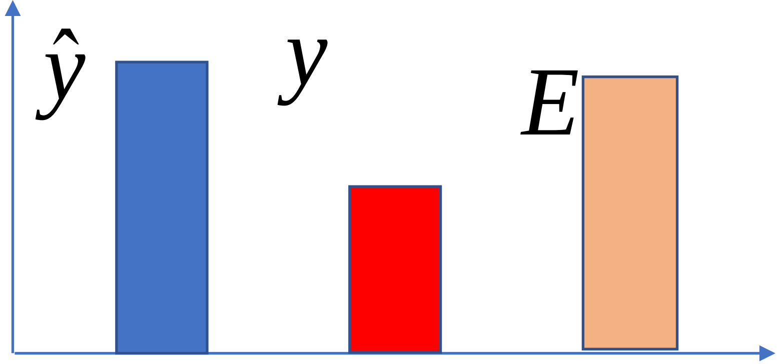
- Consider one data input

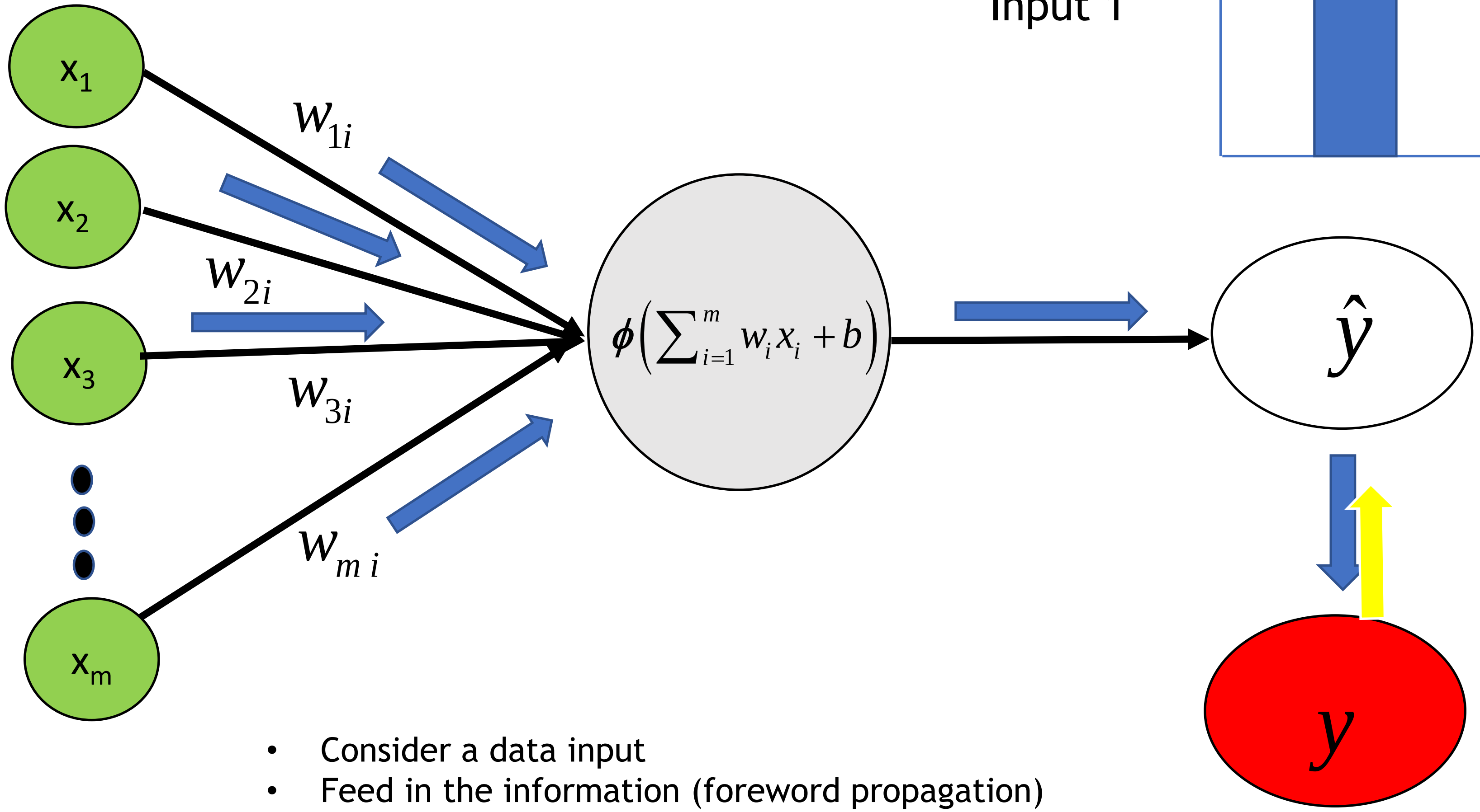
How do NNs learn?



- Consider a data input
- Feed in the information (foreword propagation)

Input 1

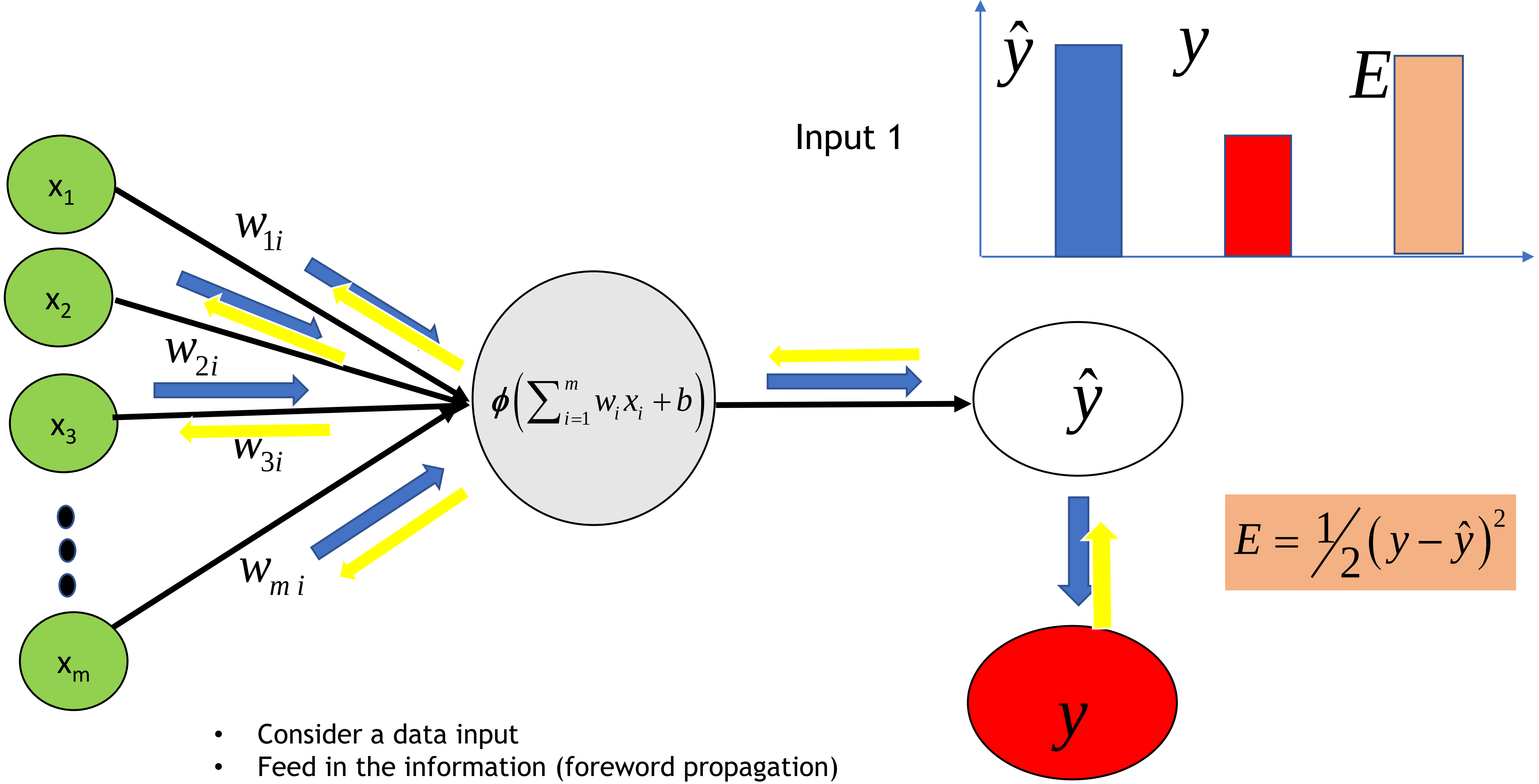




$$E = \frac{1}{2}(y - \hat{y})^2$$

- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the individual loss wrt actual value.

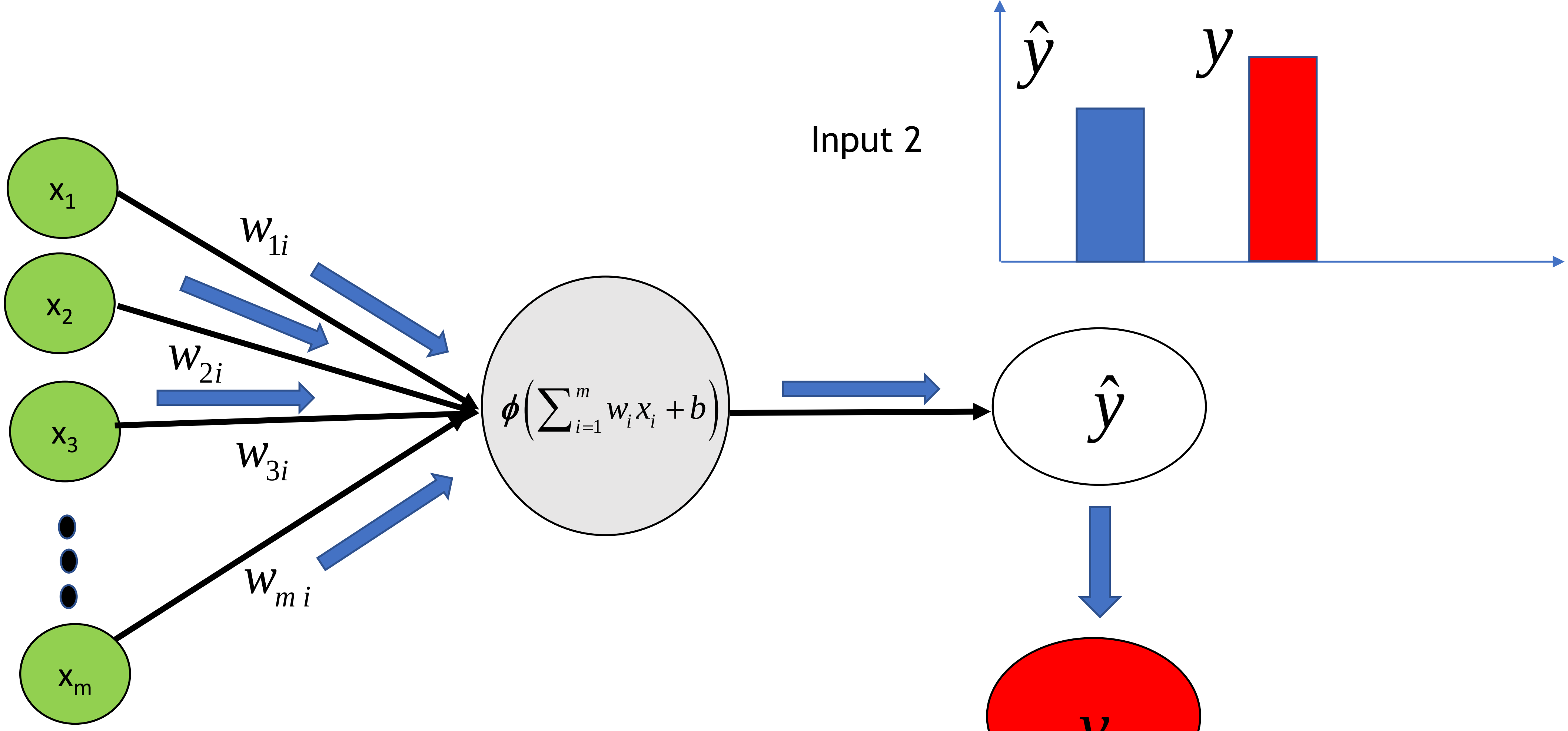
Note: Objective to minimise the cost function. Find optimal weights.



- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

Note: Objective to minimise the cost function. Find optimal weights.

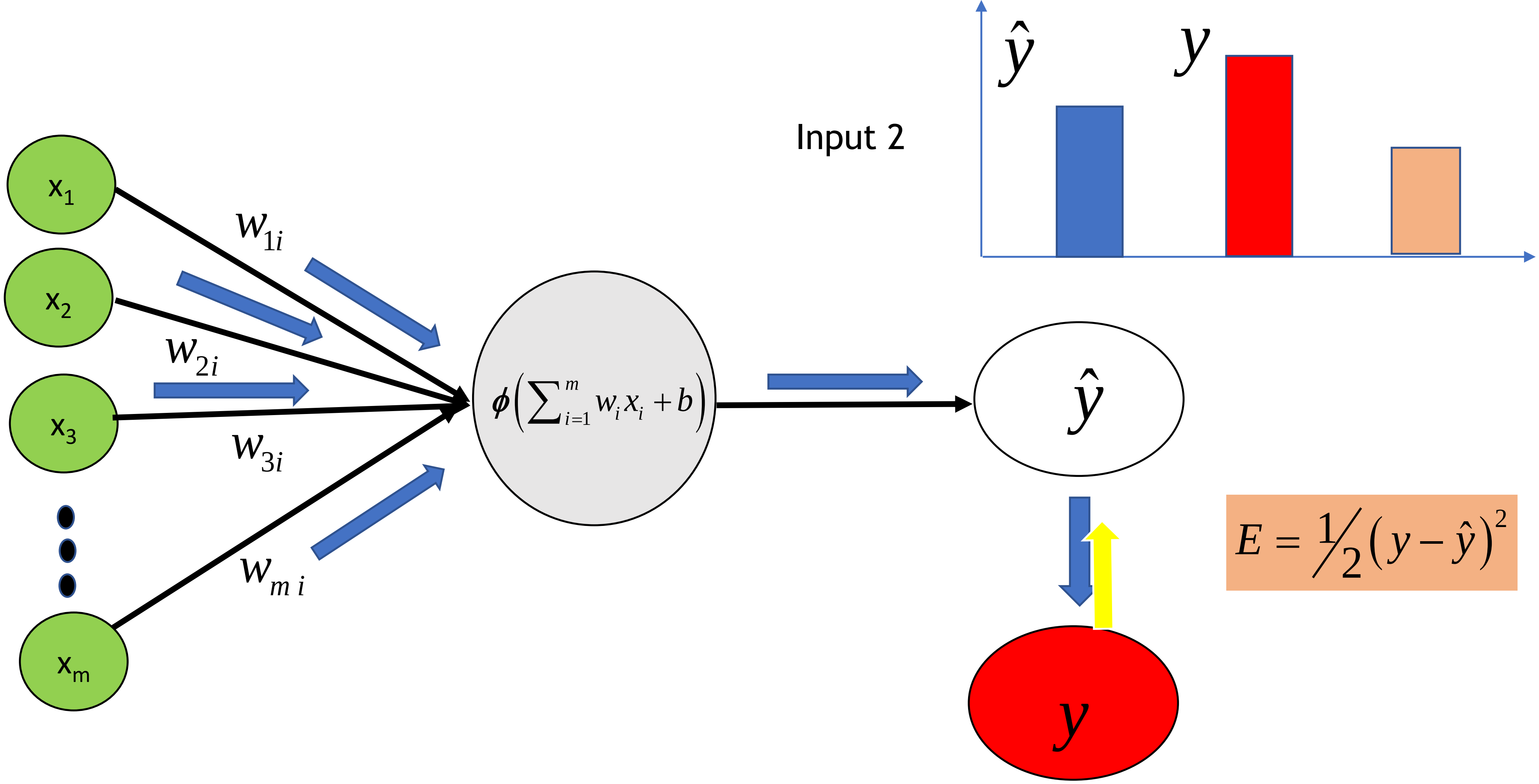
- Information can be fed back, to adjust the weights.



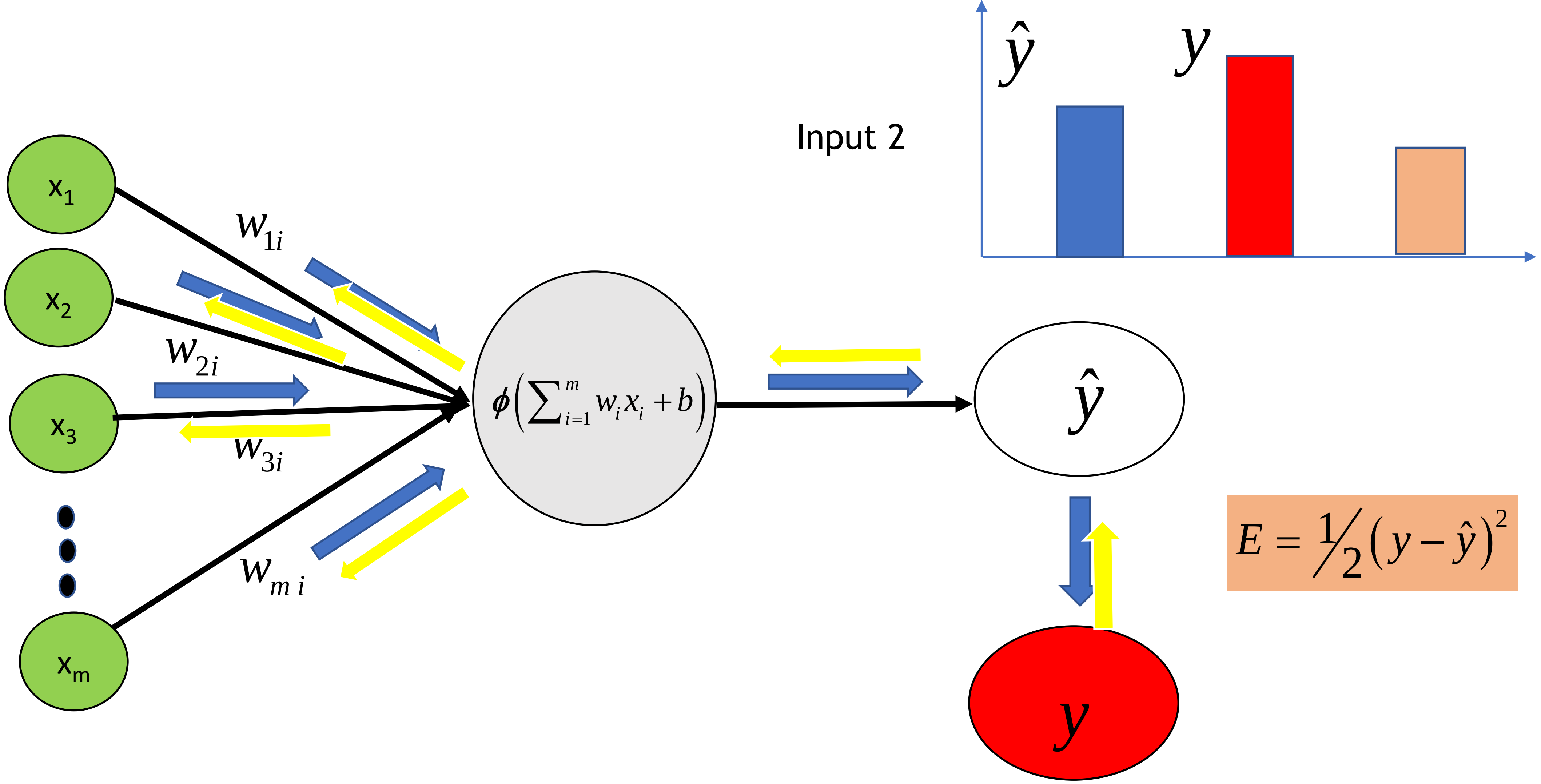
- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

Note: Objective to minimise the cost function. Find optimal weights.

- Information can be fed back, to adjust the weights.
- Repeated with other data inputs.

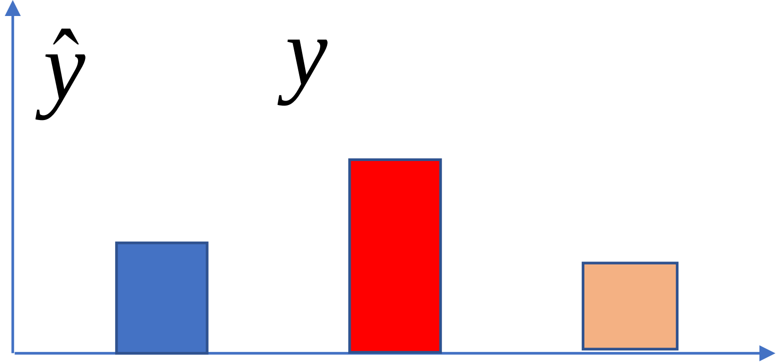


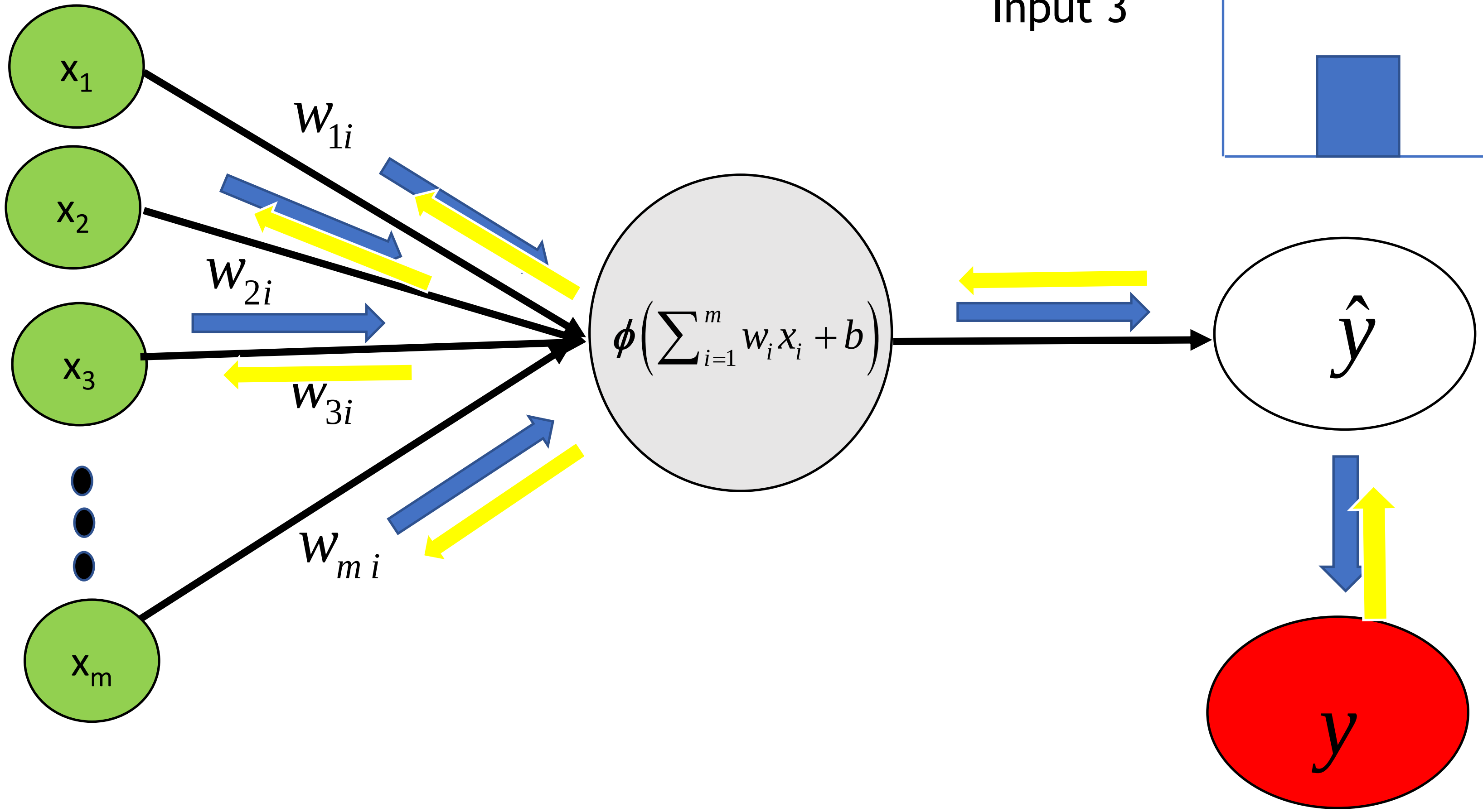
x_1
x_2
x_3
x_m
w_{1i}
w_{2i}
w_{3i}
$w_{m\,i}$
$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$
$\hat{y}$
y
Input 2
$\hat{y}$
y
$E = \frac{1}{2}(y-\hat{y})^2$



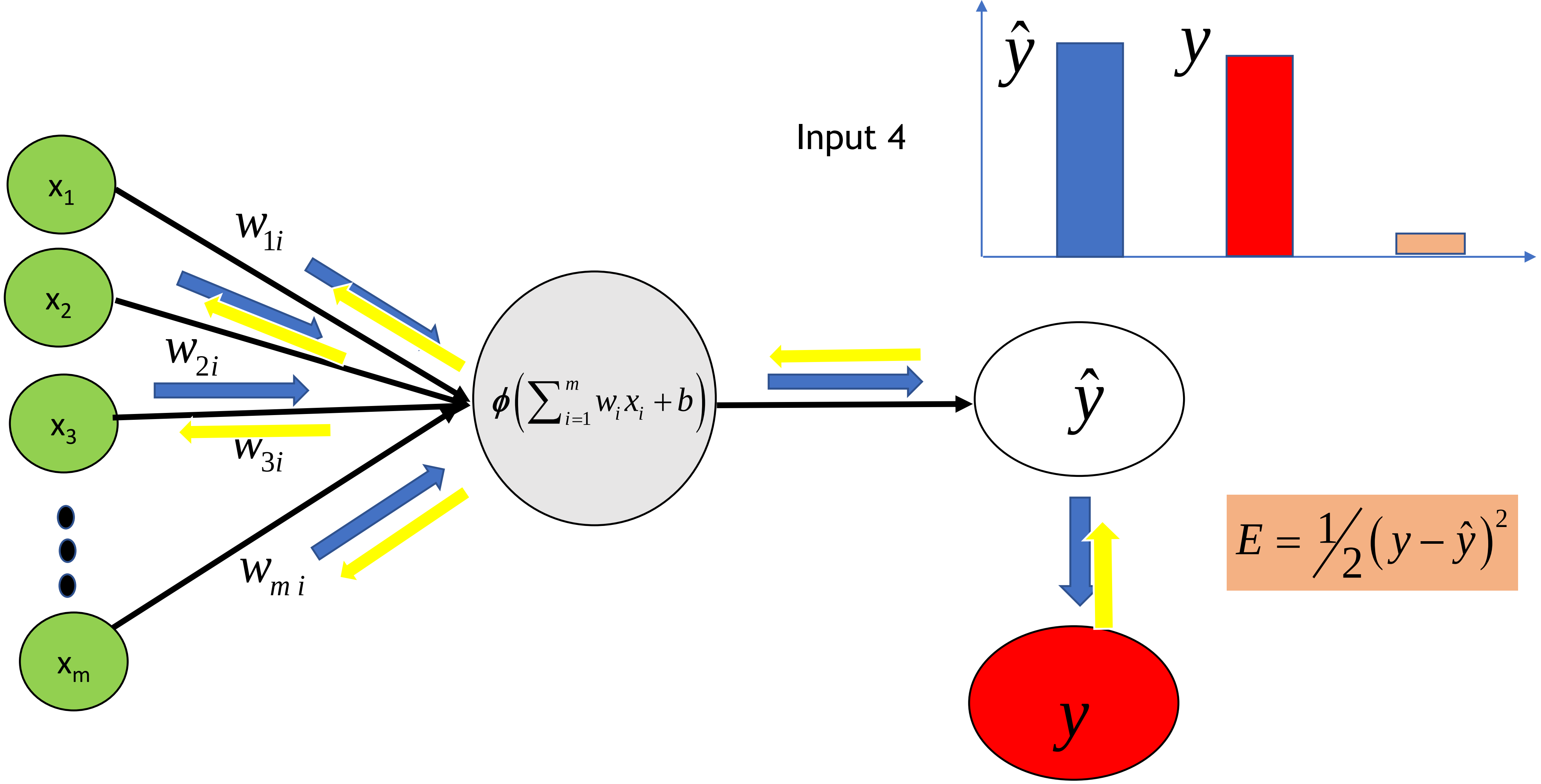
Input 2
$\hat{y}$
y
x_1
x_2
x_3
x_m
w_{1i}
w_{2i}
w_{3i}
$w_{m\,i}$
$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$
$\hat{y}$
y
$E = \frac{1}{2}(y-\hat{y})^2$

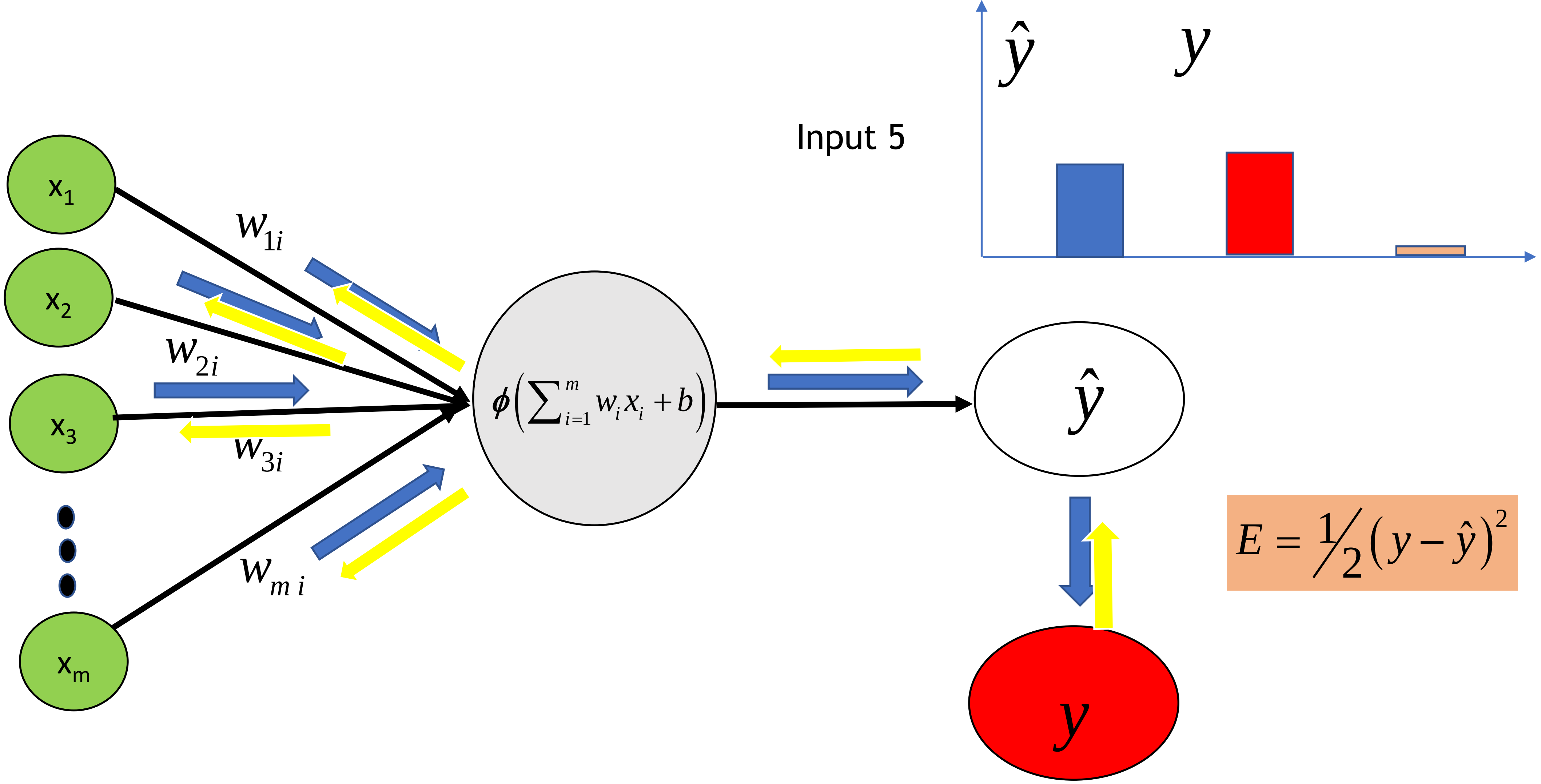
Input 3





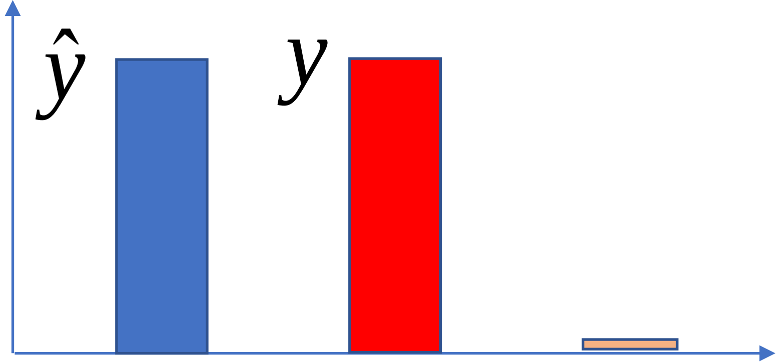
$$E = \tfrac{1}{2}(y-\hat{y})^2$$

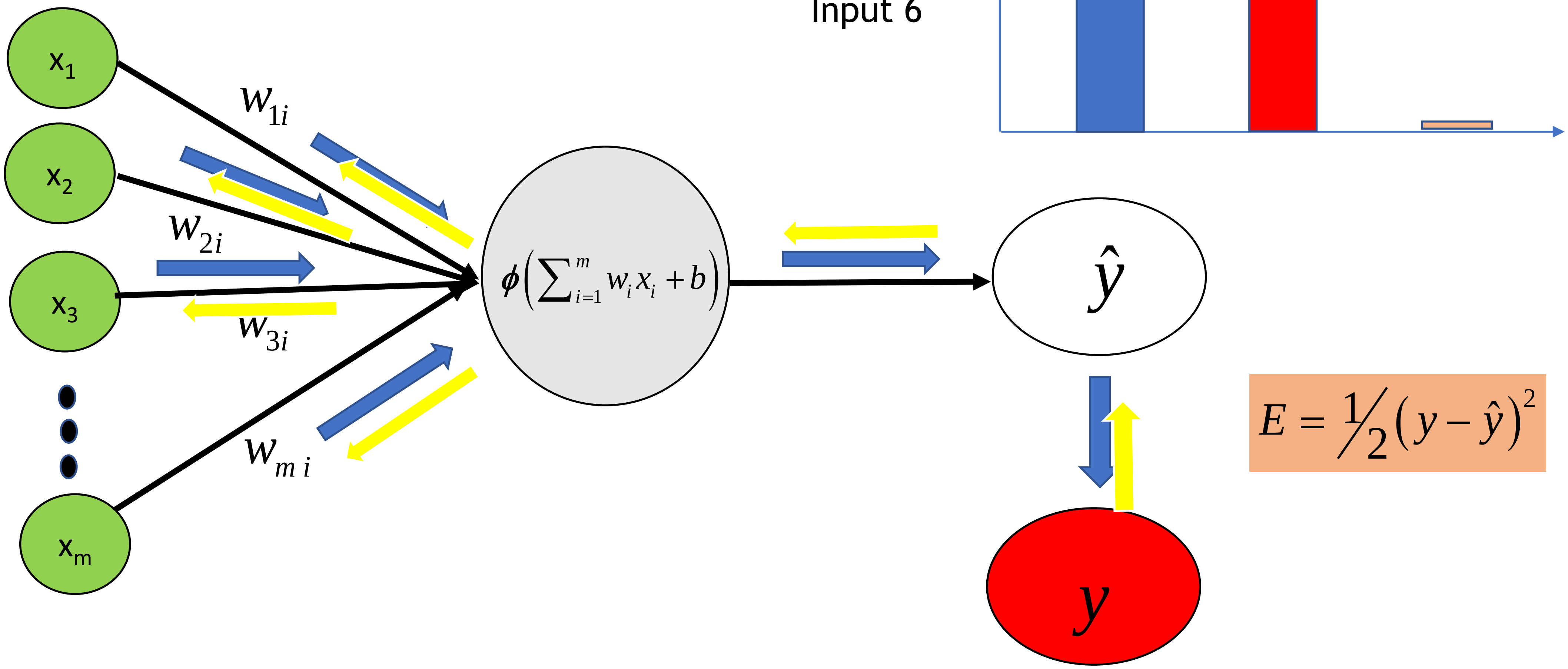

$\hat{y}$
y
Input 4
x_1
x_2
x_3
x_m
w_{1i}
w_{2i}
w_{3i}
$w_{m\,i}$
$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$
$\hat{y}$
y
$E = \frac{1}{2}(y-\hat{y})^2$

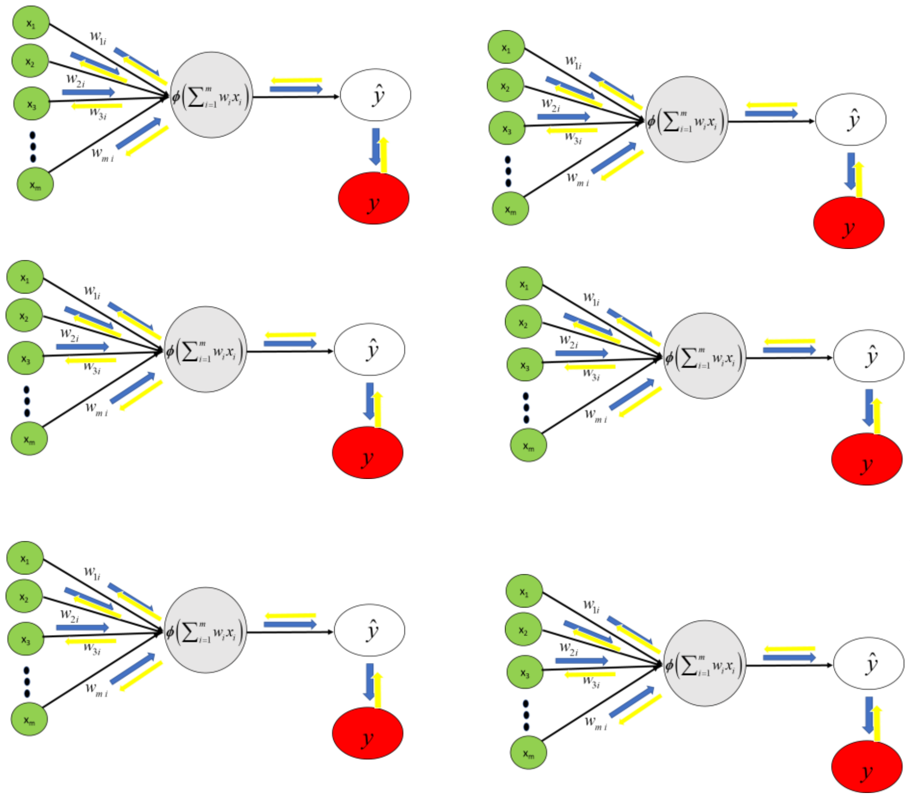


Input 5
$\hat{y}$
y
x_1
x_2
x_3
x_m
w_{1i}
w_{2i}
w_{3i}
$w_{m\,i}$
$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$
$\hat{y}$
y
$E = \frac{1}{2}(y-\hat{y})^2$

Input 6


$\hat{y}$
y


x_1
x_2
x_3
x_m
w_{1i}
w_{2i}
w_{3i}
$w_{m\,i}$
$\phi\left(\sum_{i=1}^{m} w_i x_i + b\right)$
$\hat{y}$
y
$E = \frac{1}{2}(y - \hat{y})^2$

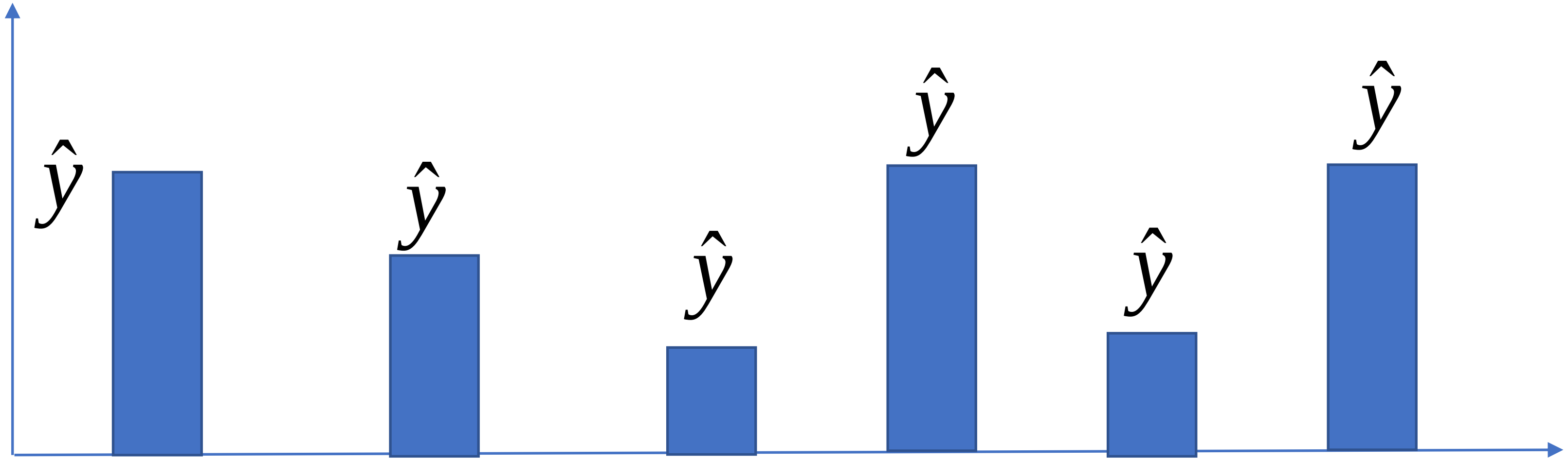


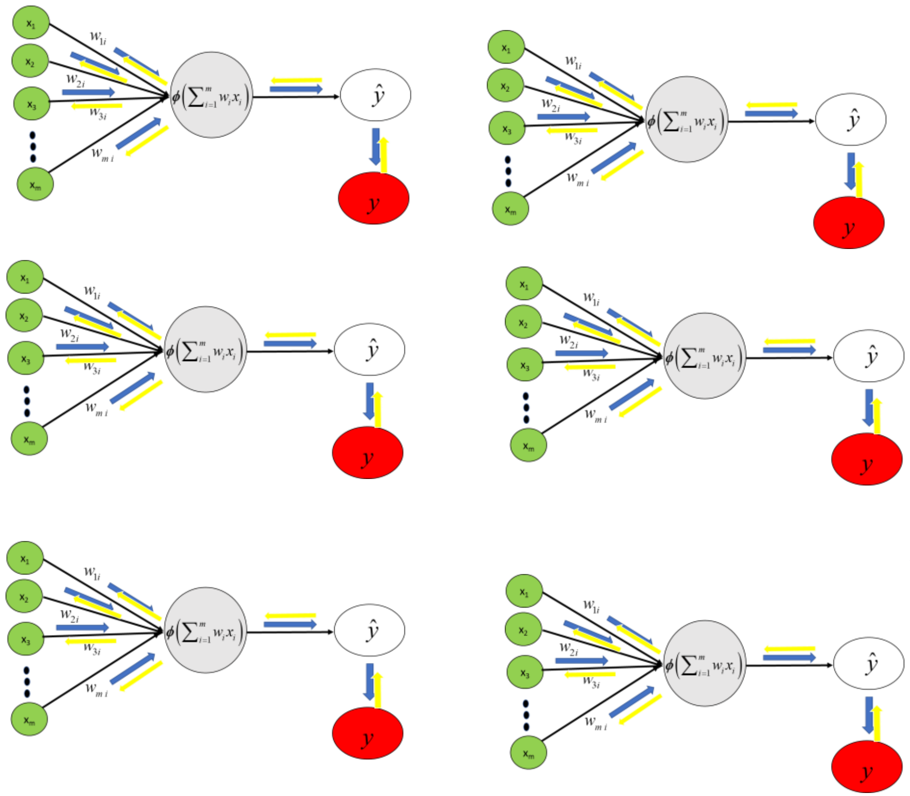
Batch update (One iteration)

- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

Note: Objective to minimise the cost function. Find optimal weights.

- Information can be fed back, to adjust the weights.
- Repeated with other data inputs.



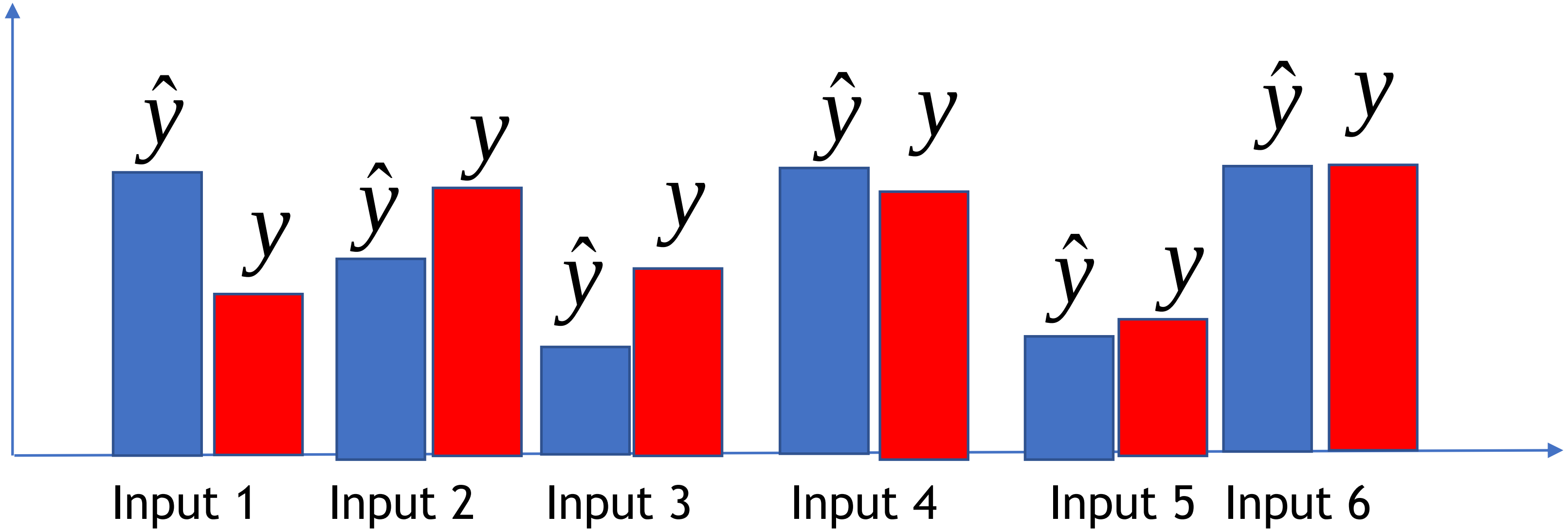


Batch update (One iteration)

- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

Note: Objective to minimise the cost function. Find optimal weights.

- Information can be fed back, to adjust the weights.



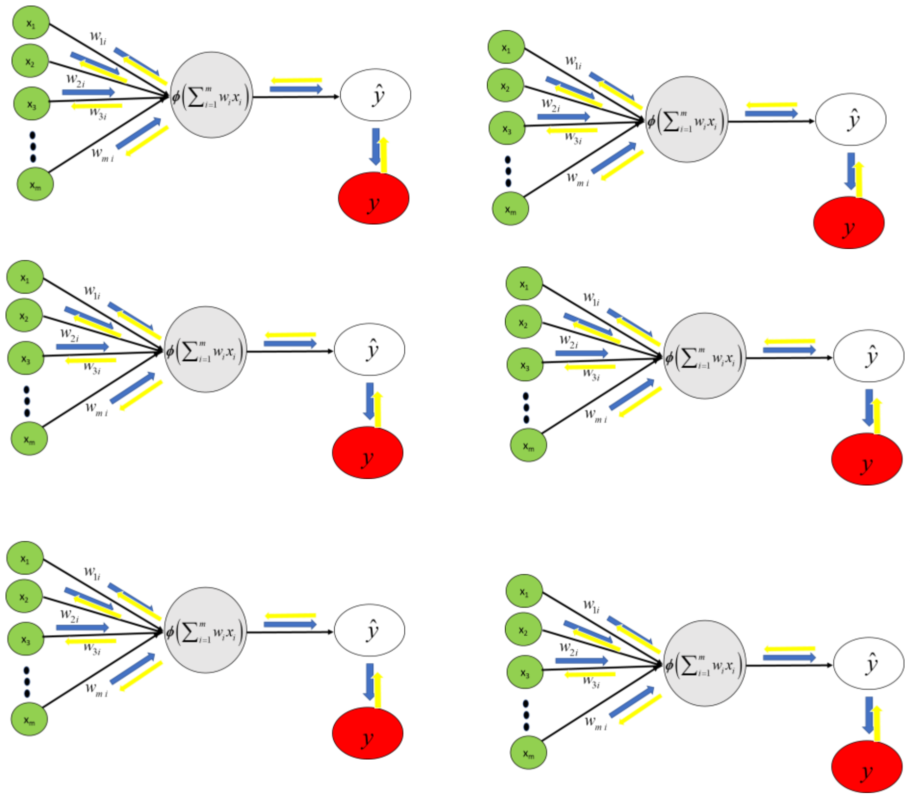
Rationale: The global error is backward propagated to network nodes, weights are modified proportional to their contribution.

Batch update (One iteration)

- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

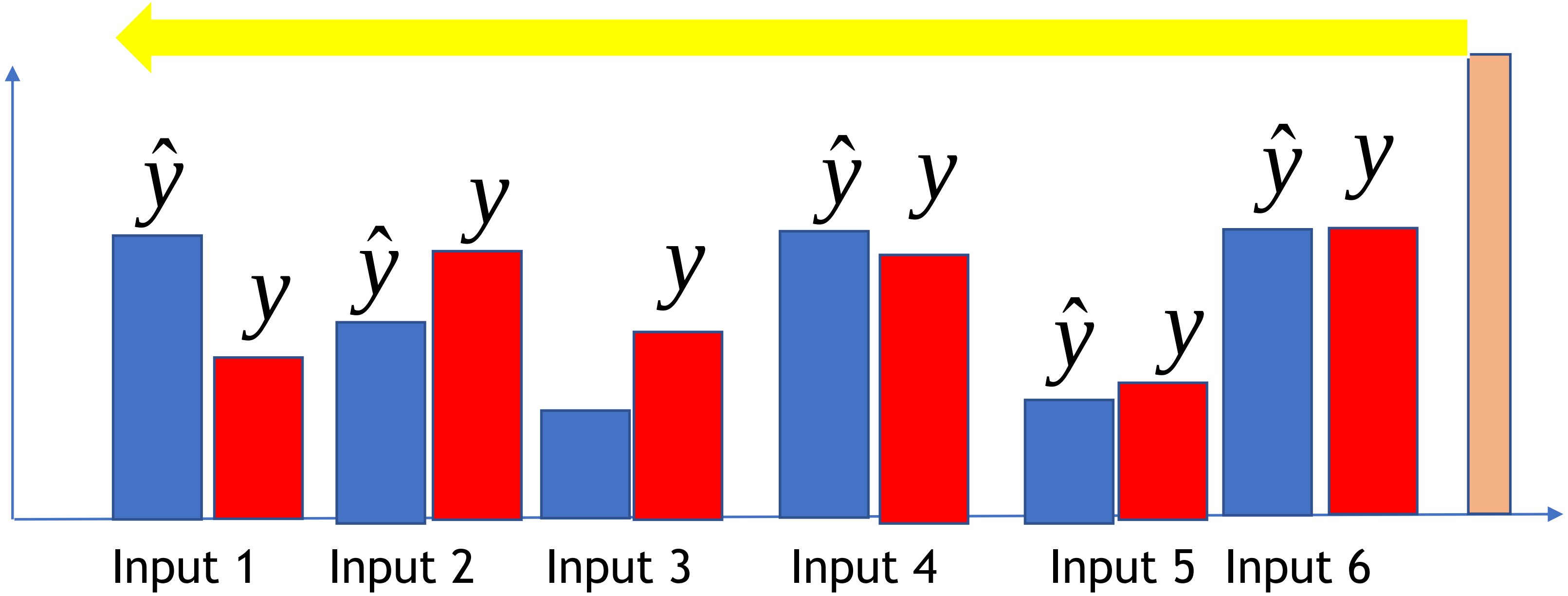
Note: Objective to minimise the cost function. Find optimal weights.

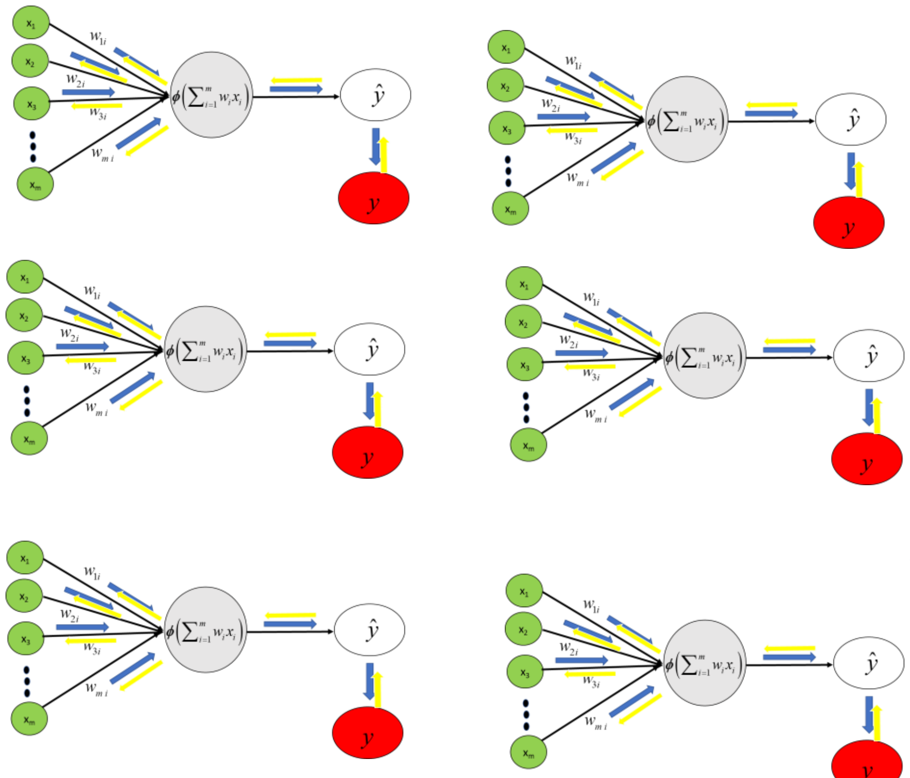
- Information can be fed back, to adjust the weights.



$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$

Update all the weights





Batch update (One iteration)

- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

Note: Objective to minimise the cost function. Find optimal weights.

- Information **will be** fed back, to adjust the weights.
- Repeated with other data inputs.

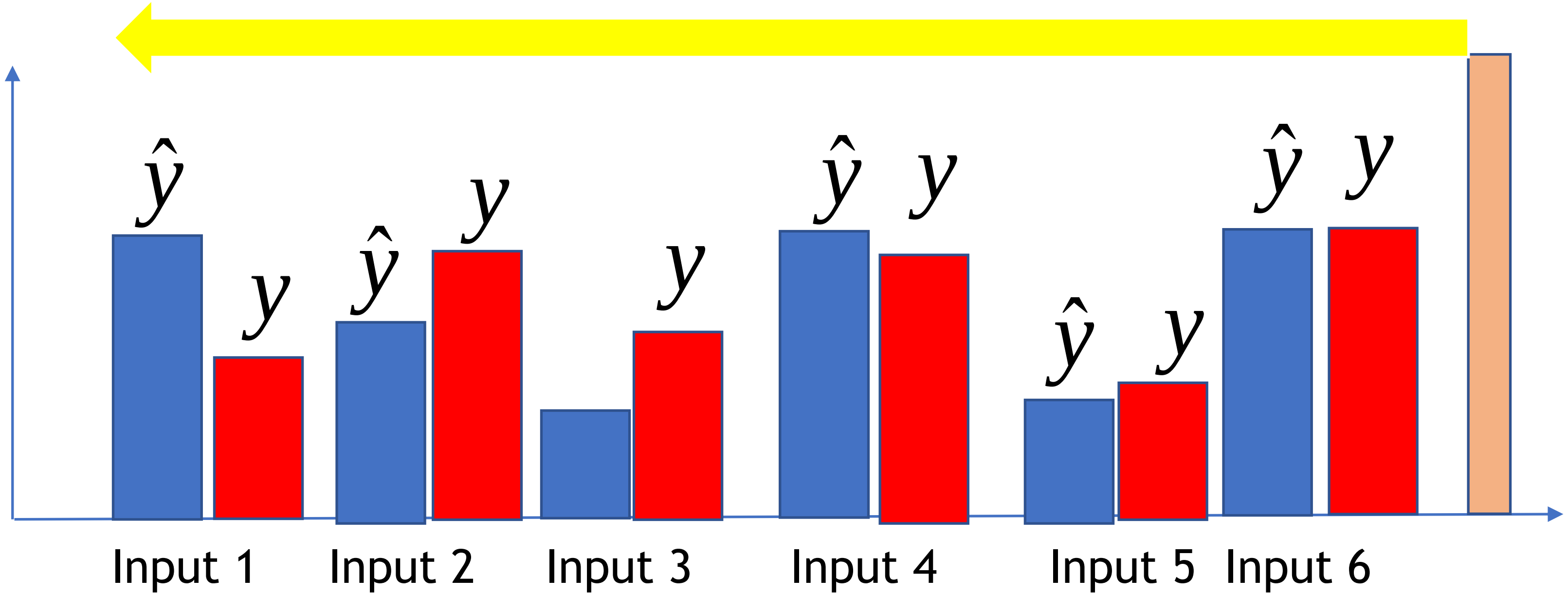
.

.

- Total loss → cost function
- The weights adjusted 'at the same time' using total loss.

$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$

Update all the weights



Batch update (One iteration)

- Consider a data input
- Feed in the information (foreword propagation)
- Calculate the loss with respect to its actual value.

Note: Objective to minimise the cost function. Find optimal weights.

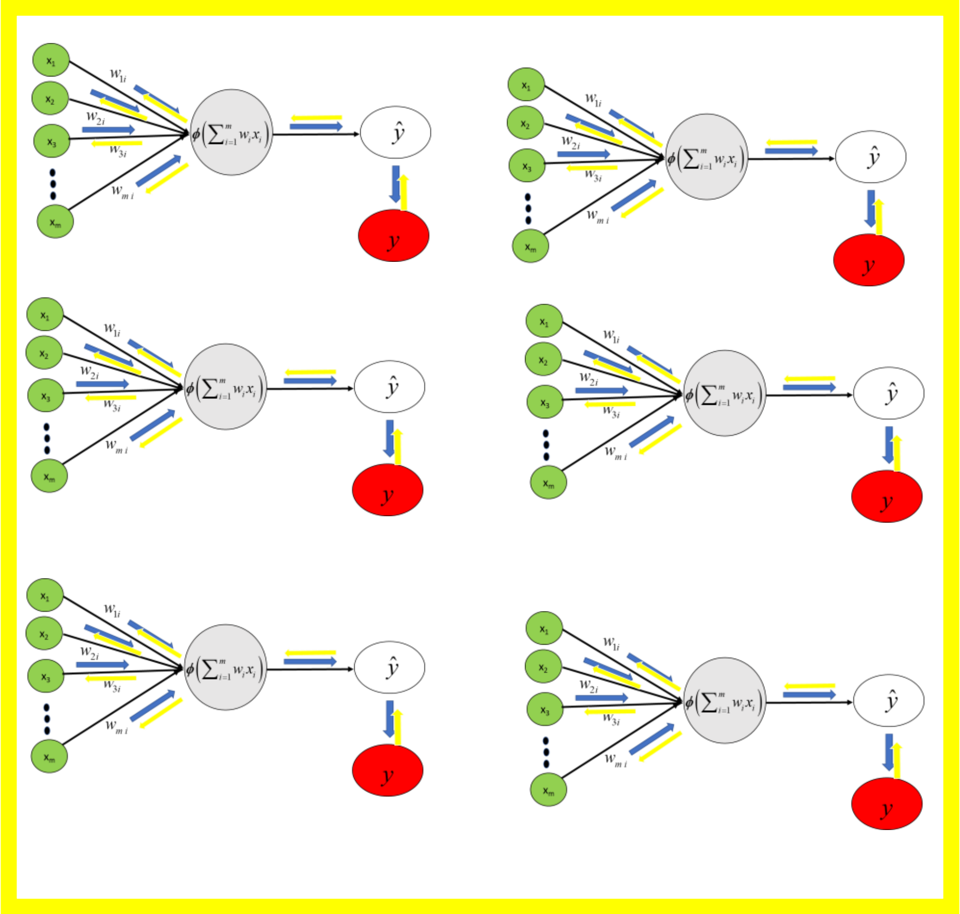
- Information **will be** fed back, to adjust the weights.
- Repeated with other data inputs.

.

.

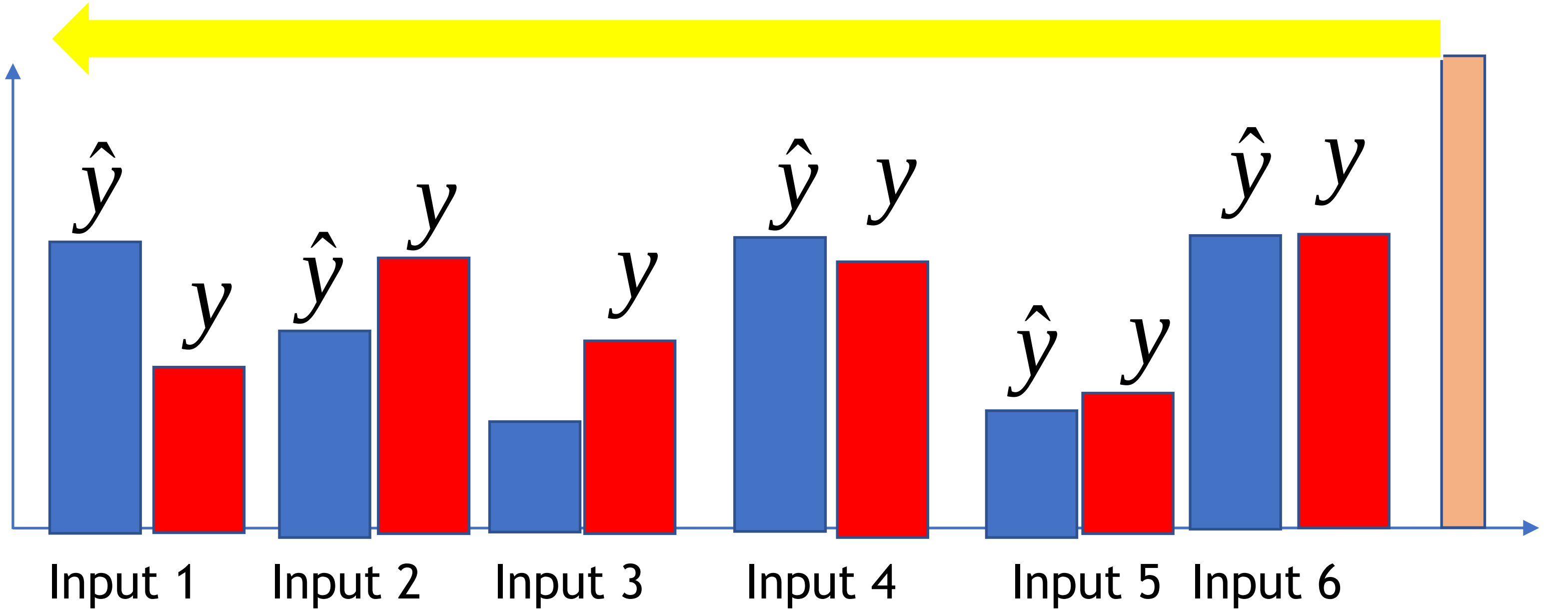
- Total loss → cost function
- The weights adjusted ‘at the same time’ using total loss.

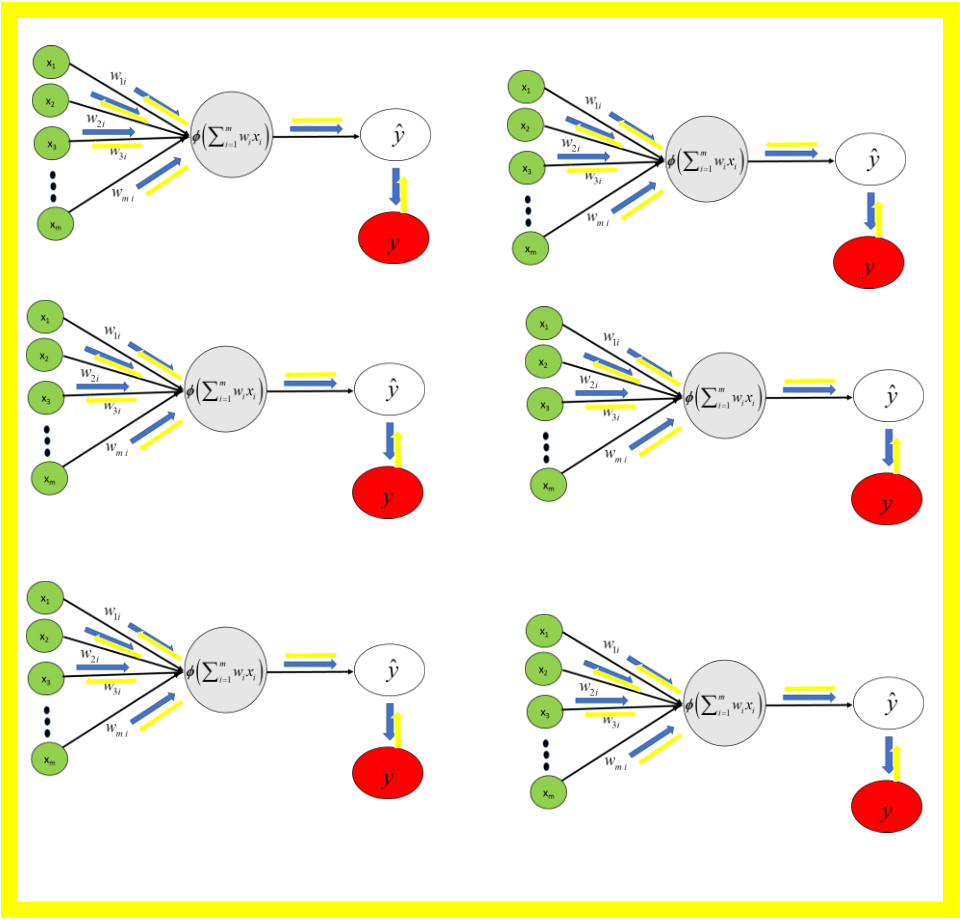
$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$



One epoch = training done on entire data set once.

Update all the weights

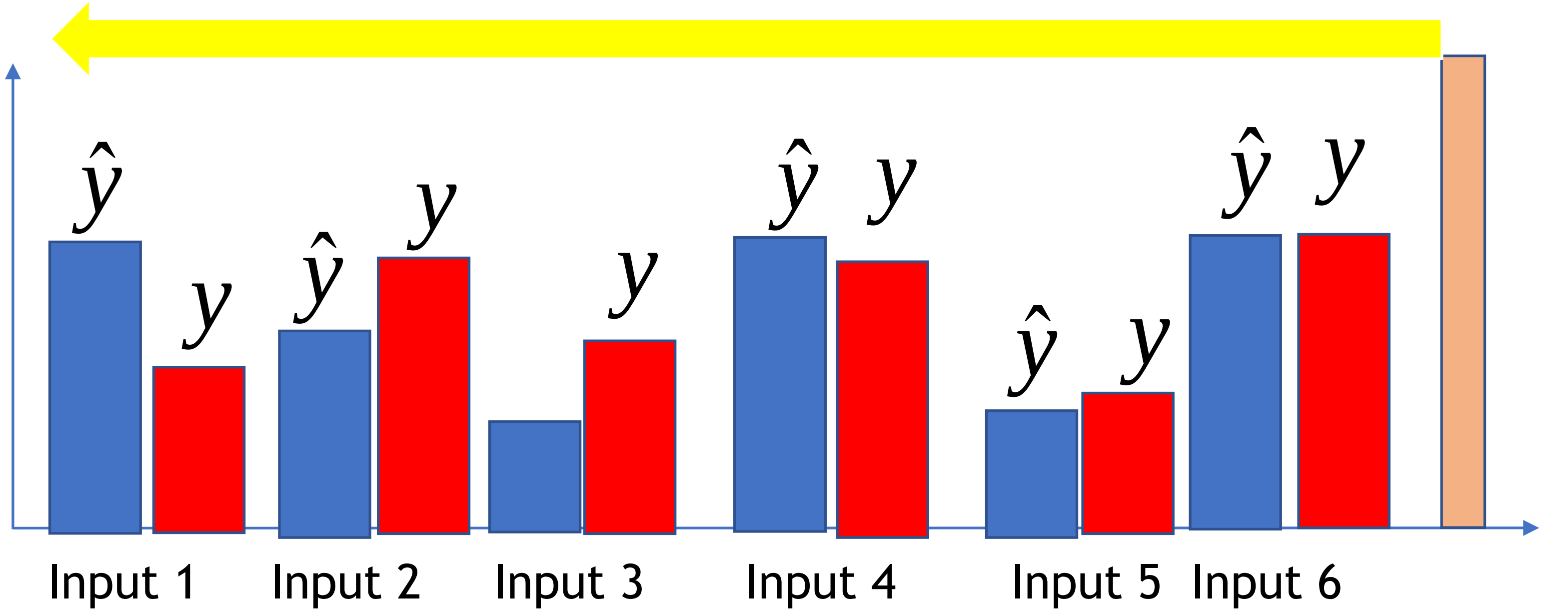




Objective : To minimize this loss,

$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$

Update all the weights

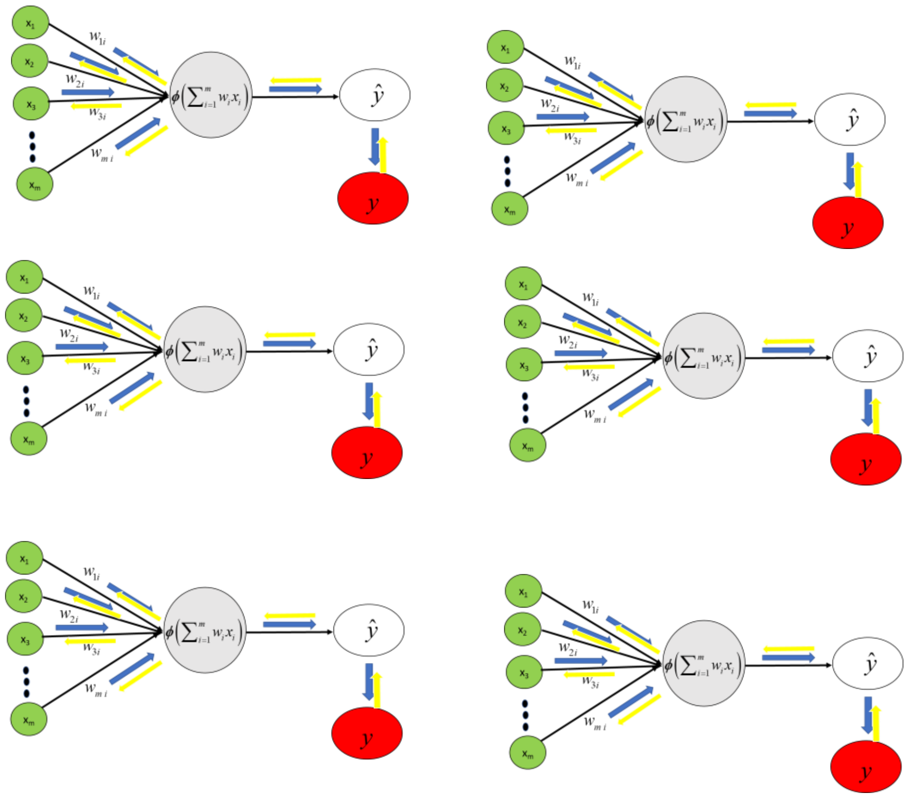


One epoch = training done on entire data set once.

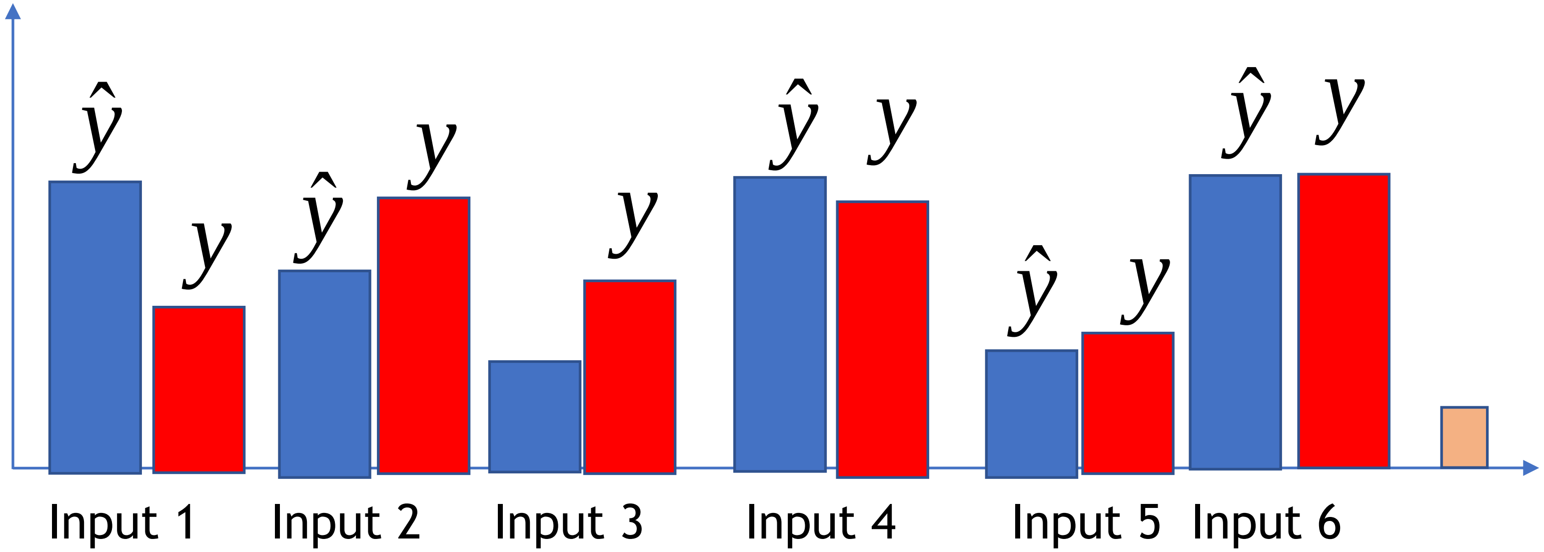
Objective : To minimize this loss, find optimal sets of weights.

How to minimise the loss and update the weights??

$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$



Update all the weights



One epoch = training done on entire data set once.

Gradient Descent

Gradient Descent

$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$

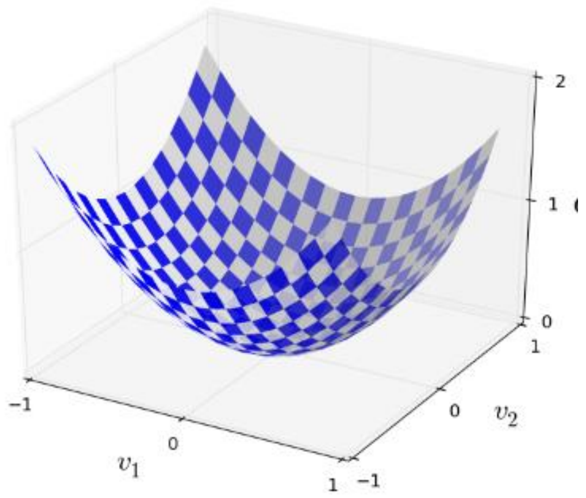
- GD: iterative method of finding minimum of any given function. Why iterative method preferred?
- NNs involve non-linear functions, close solutions of min of loss functions not available.
- Objective: To minimize the loss function **(cost function)** or mean error between neural network output and actual values (chosen by user, Example: mean square error) .

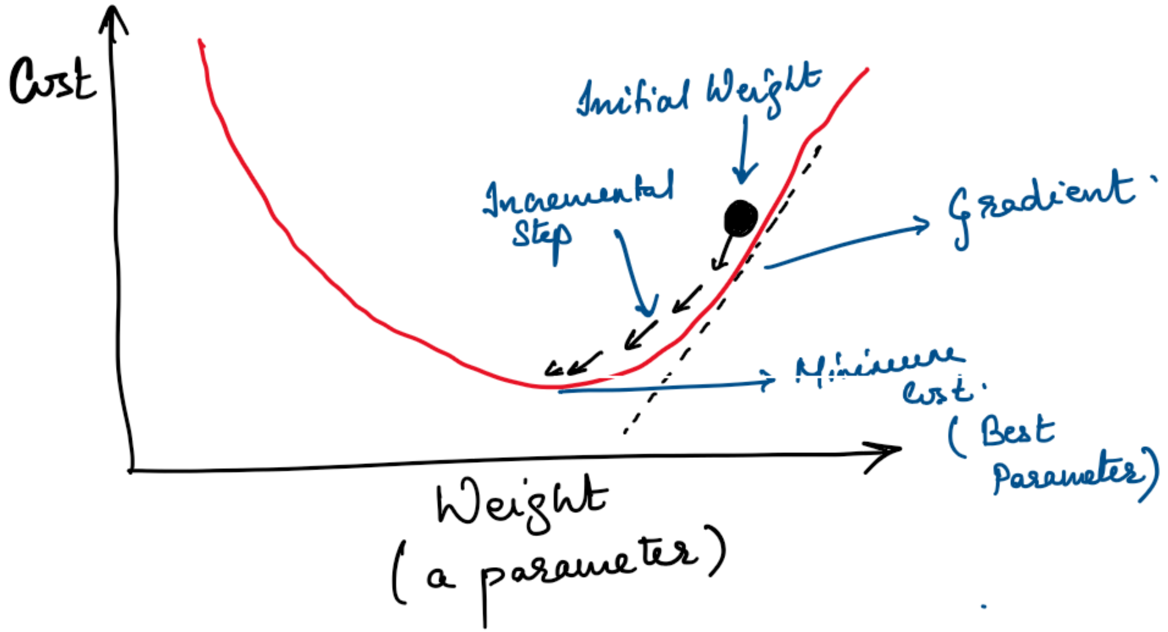
Gradient Descent

$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$

- GD: iterative method of finding minimum of any given function. Why iterative method preferred?
- NNs involve non-linear functions, close solutions of min of loss functions not available.
- Objective: To minimize the loss function or mean error between neural network output and actual values (chosen by user, Example: mean square error) .

- Intuition behind GD: Climbing down the hill to find its bottom or minimum value given by best parameters.





Gradient Descent

- GD: iterative method of finding minimum of any given function. Why iterative method preferred?
- NNs involve non-linear functions, close solutions of min of loss functions not available.
- Objective: To minimize the loss function or mean error between neural network output and actual values (chosen by user, Example: mean square error) .

$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$

- Intuition behind GD: Climbing down the hill to find its bottom or minimum value given by best parameters.

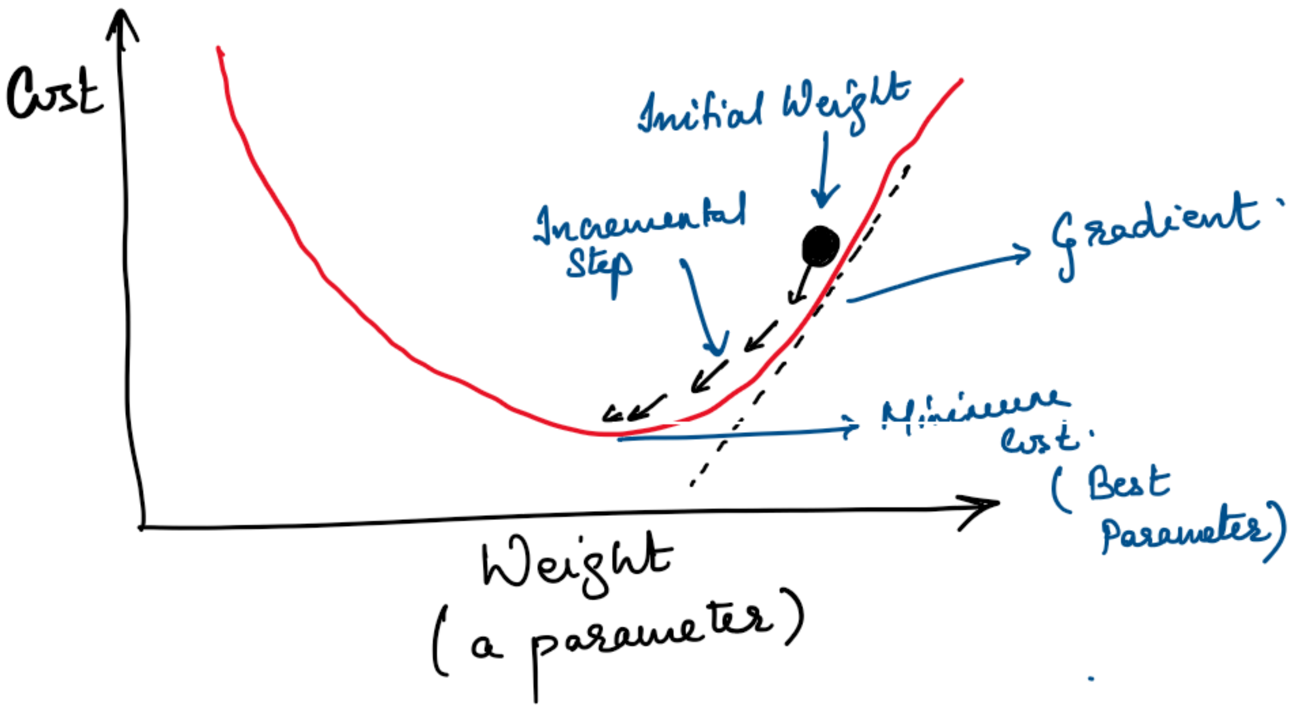
Basic steps:

Given the loss function $J(w,b)$

- Compute the slope (gradient) that is the first-order derivative of the function at the current point.
- Move-in the opposite direction of the slope increase from the current point by the computed amount.

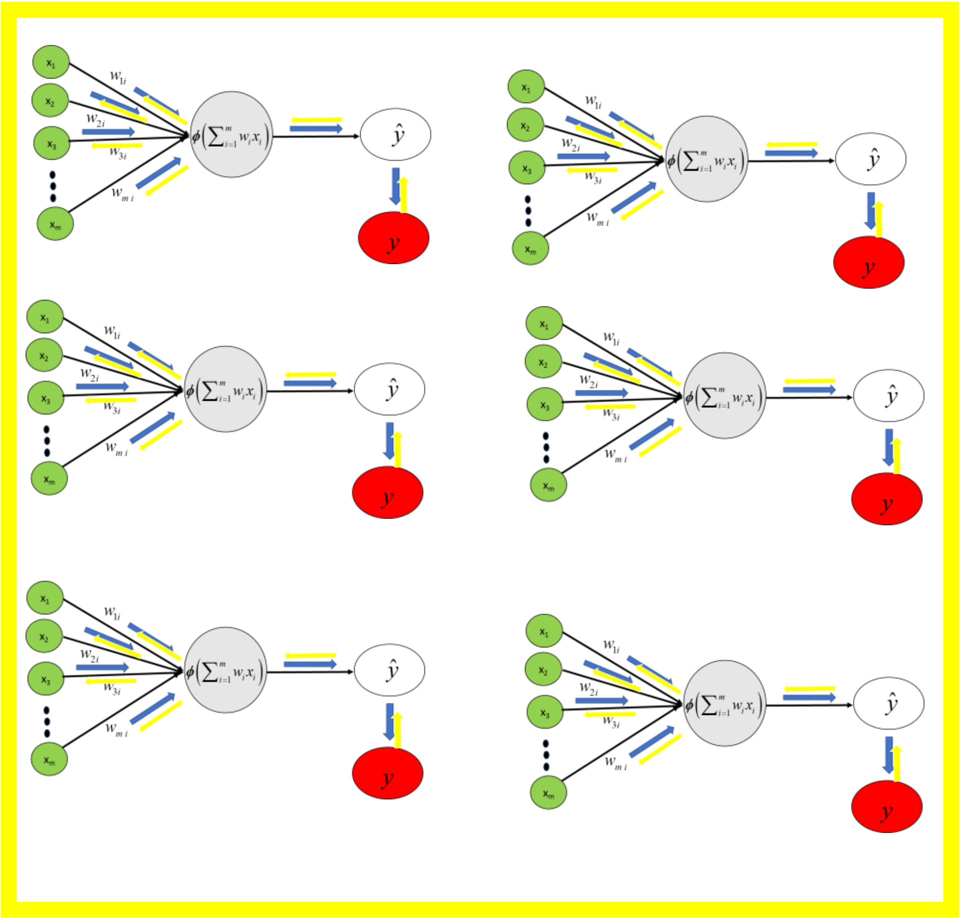
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial\,(J(\mathbf{w},b))}{\partial\,\mathbf{w}}$$

$$b \leftarrow b - \alpha \frac{\partial\,(J(w,b))}{\partial\,b}$$



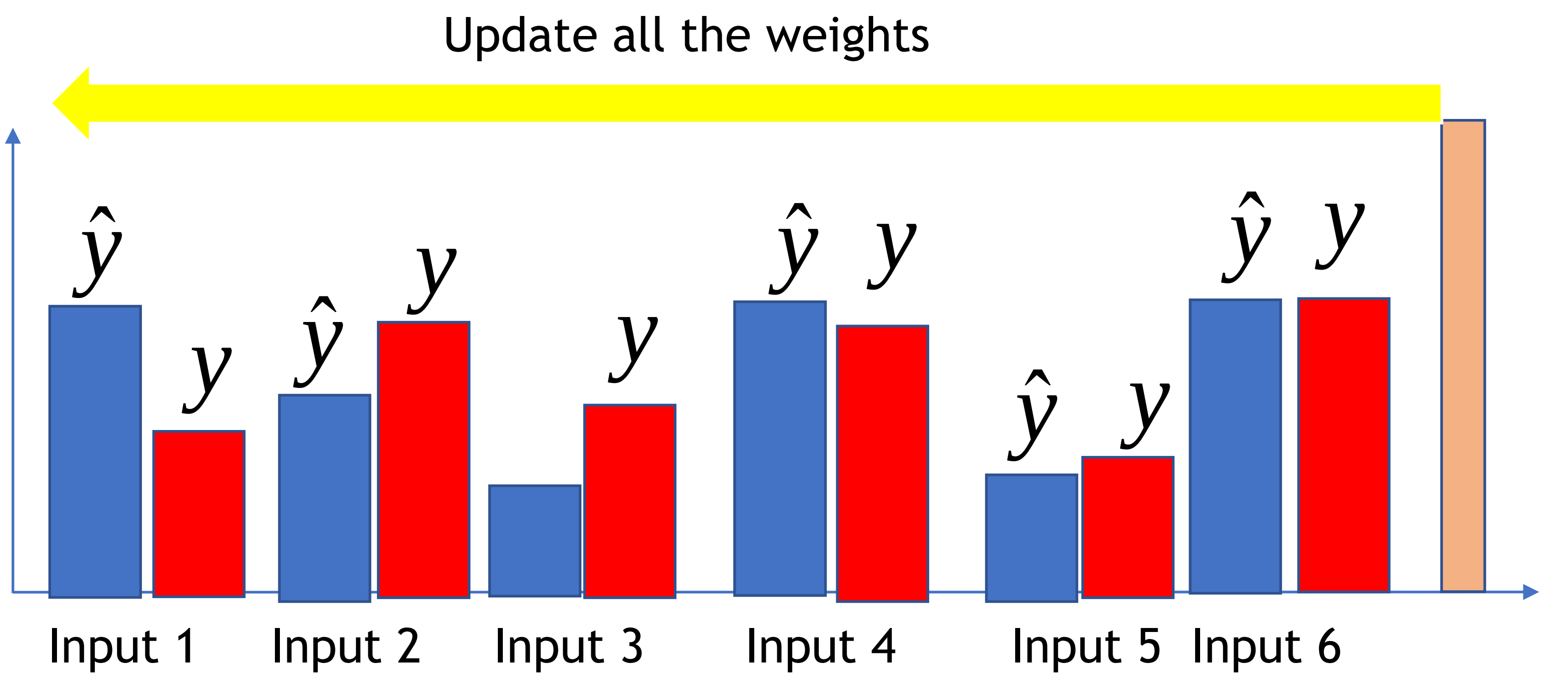
Batch Gradient Descent

Saw earlier: Weights were updated using total loss of a data batch ➔ Batch GD.



One epoch = training done on entire data set once.

$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$

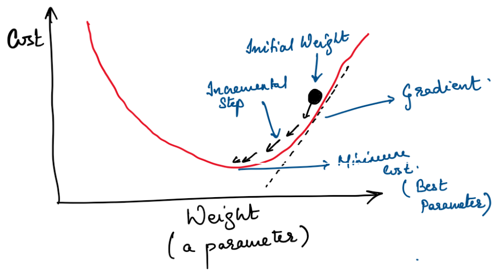


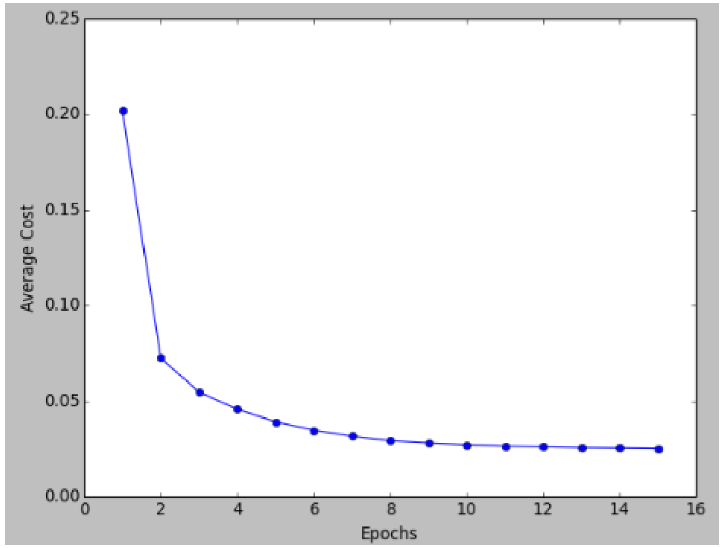
Gradient Descent

$$E_{tot} = \sum \frac{1}{2}(y - \hat{y})^2$$

- GD: iterative method of finding minimum of any given function. Why iterative method preferred?
s
- NNs involve non-linear functions, close solutions of min of loss functions not available.
- Objective: To minimize the loss function or mean error between neural network output and actual values (chosen by user, Example: mean square error) .

- Intuition behind GD: Climbing down the hill to find its bottom or minimum value given by best parameters.





Basic steps:

Given the loss function $J(w,b)$

- Compute the slope (gradient) that is the first-order derivative of the function at the current point.
- Move-in the opposite direction of the slope increase from the current point by the computed amount.

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial\,(J(\mathbf{w},b))}{\partial\,\mathbf{w}}$$

$$b \leftarrow b - \alpha \frac{\partial\,(J(w,b))}{\partial\,b}$$

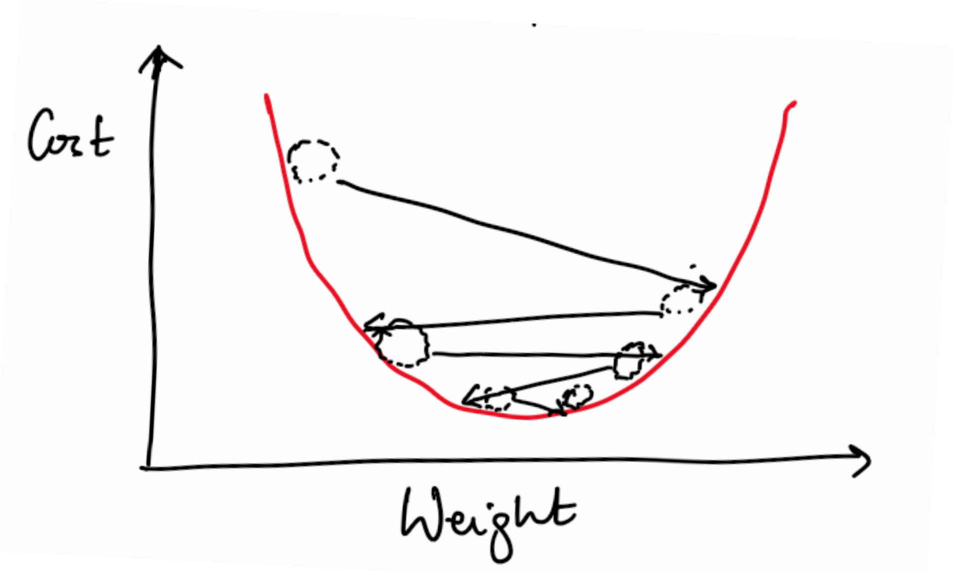
α learning rate.
What happens when learning rate is very low?
What happens when learning rate is very high?

Gradient Descent

- When learning rate too low ➔ slow convergence.
- When learning rate too high ➔ minima will be overshot ➔ slow or no convergence.

- Learning rate is a ***Hyperparameter***.
- **It must be fine tuned. Neither too high, nor too low. We see hyperparameter tuning later.**

- GD works well when the total loss function is a convex function.



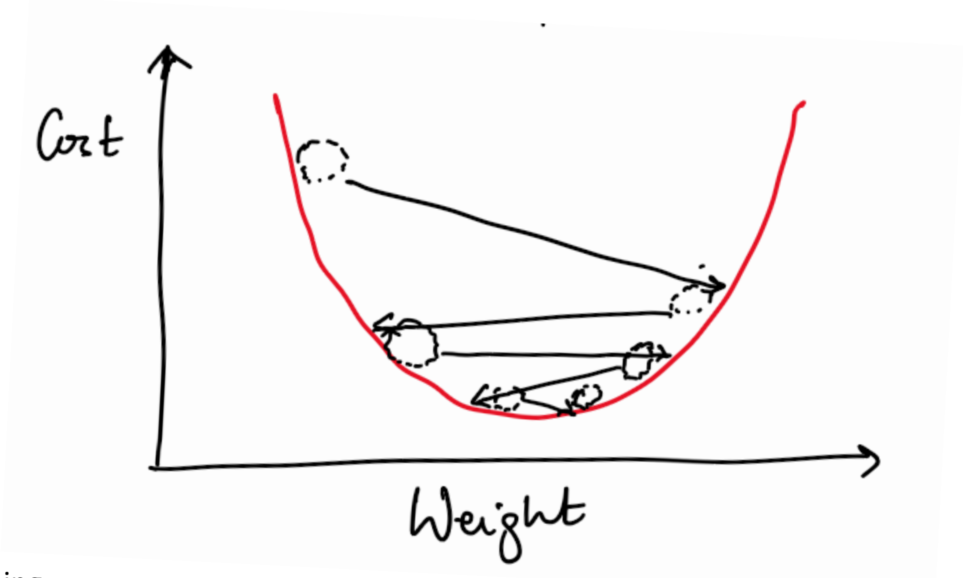
Gradient Descent

- When learning rate too low ➔ slow convergence.
- When learning rate too high ➔ minima will be overshot ➔ slow or no convergence.

- Learning rate is a ***Hyperparameter***.
- **It must be fine tuned. Neither too high, nor too low. We see hyperparameter tuning later.**

- GD works well when the total loss function is a convex function.

- What happens when function is non-convex?

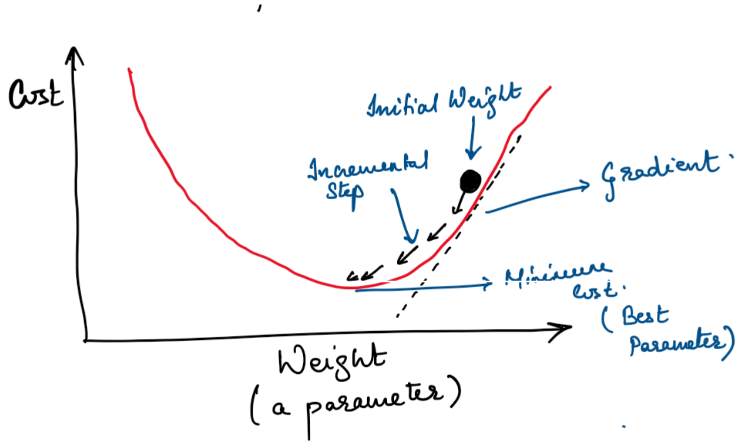


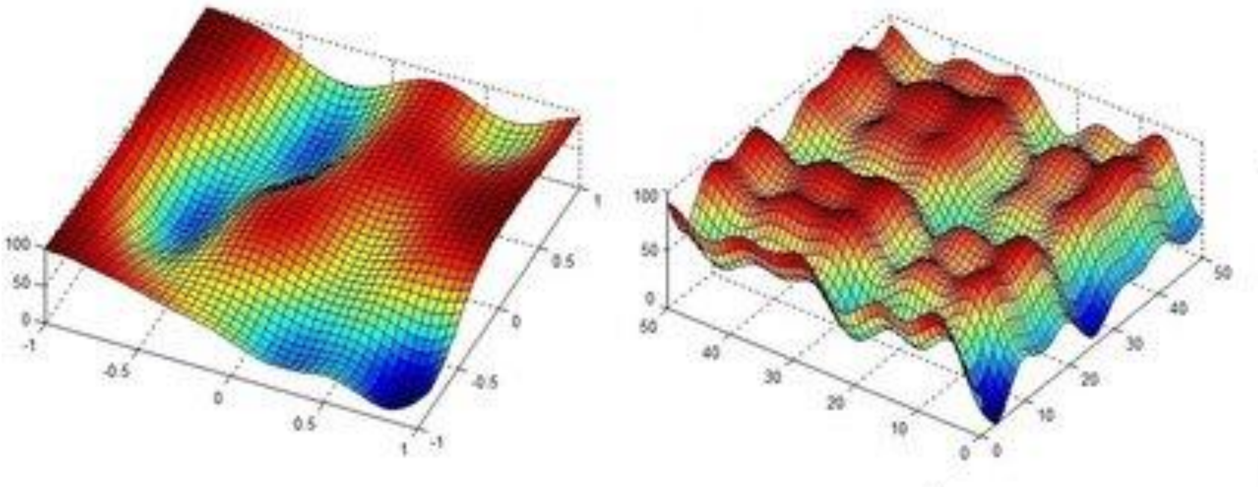
Gradient Descent

- GD works well when the total loss function is a convex function.

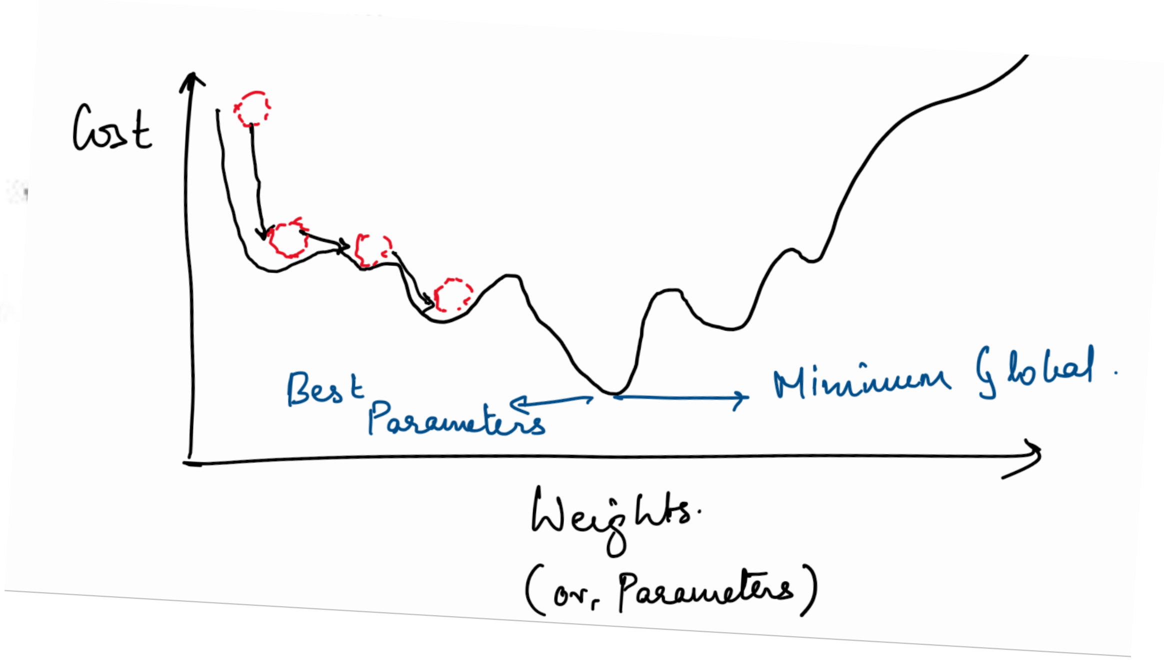
- What happens when function is non-convex?

Usually, the case, when millions of data are considered for training, with millions of parameters (weights in many layers of NNs).



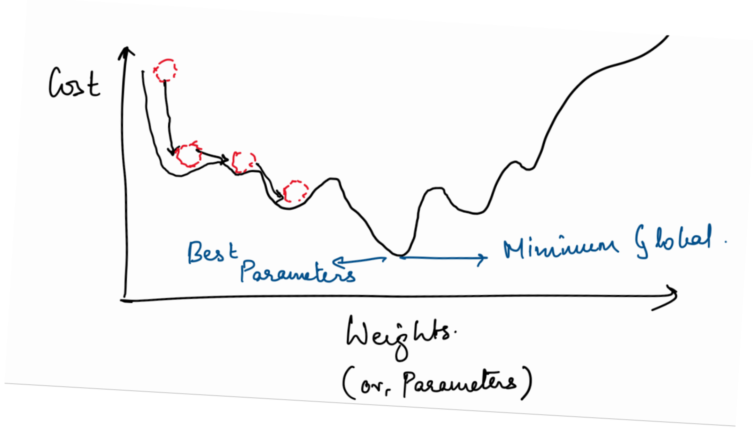


a b



Source: Taig et al.

Stochastic Gradient Descent

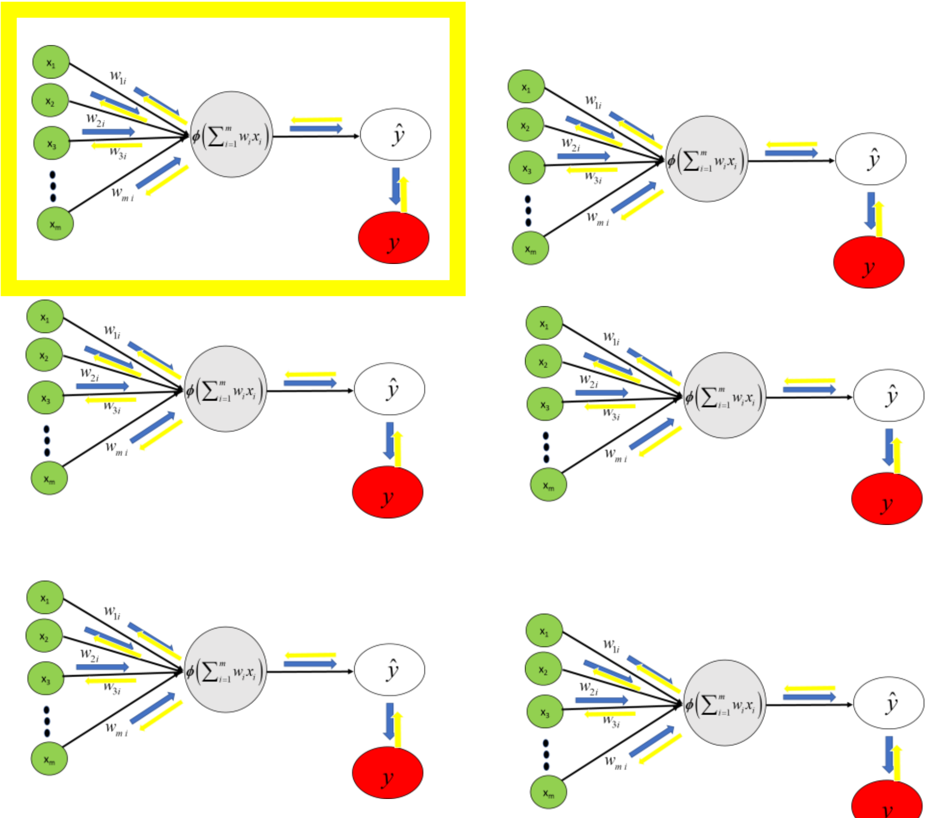


- GD : Consider a batch (set) of training data samples:
  - calculate loss
  - update weights based on total loss.

- Curse of dimensionality: Need more data for training, updating for whole set → extremely slow updates.

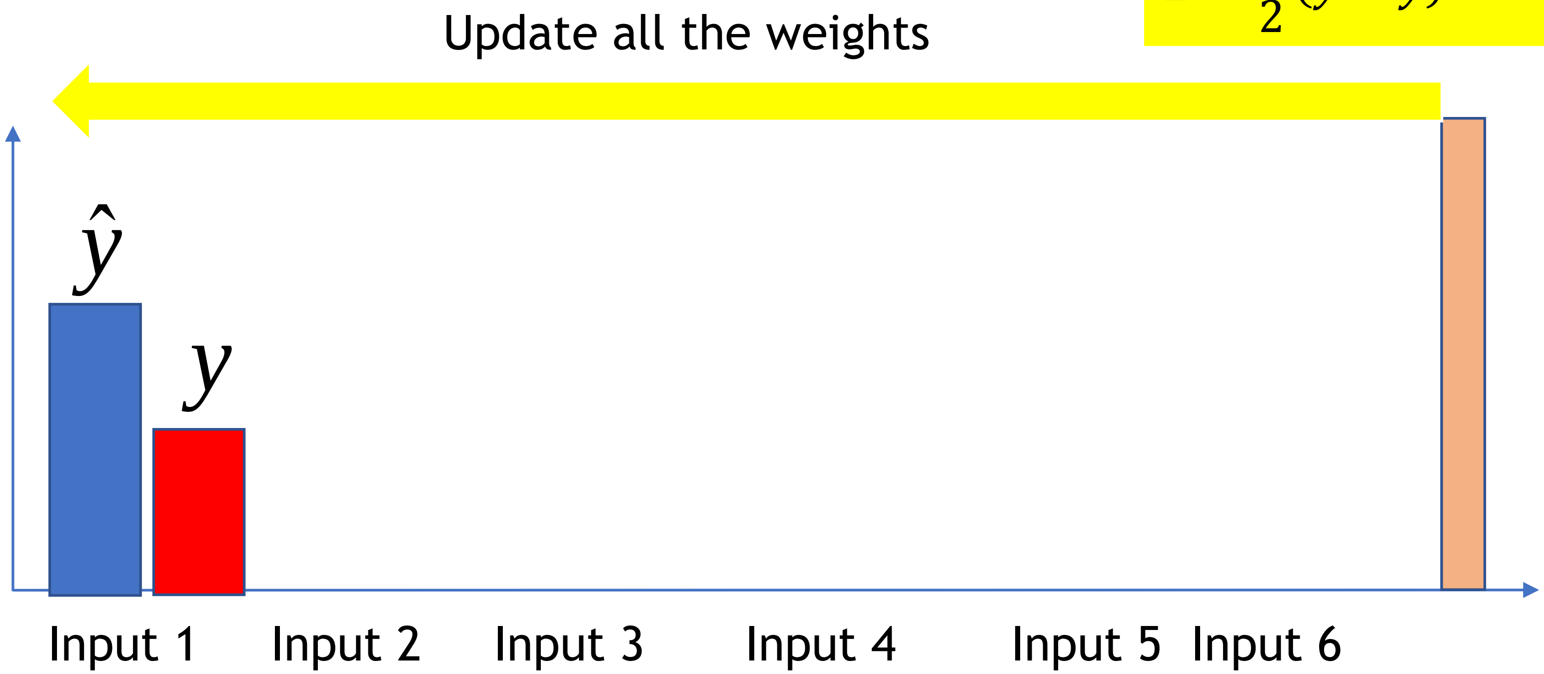
- To avoid getting stuck in local minima, a certain "jittering" or noise /exploration is needed.

Stochastic Gradient Descent



- Stochastic GD (SGD) : Updating weights after **each** training data sample.
- "Jittering" Provided by SGD : presence of diverse and many data inputs and update done for each data inputs until convergence.
- Probability to get unstuck from local minima and converge towards global minima.

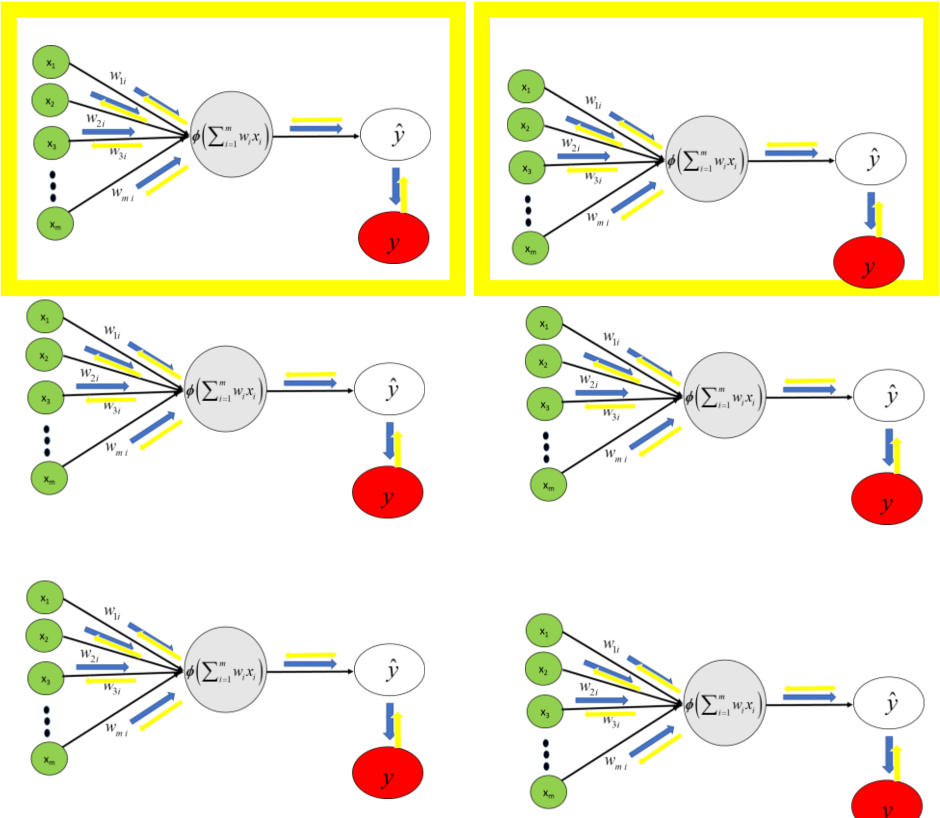
$$E = \frac{1}{2}(y - \hat{y})^2$$



One epoch = training done on entire data set once.

JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

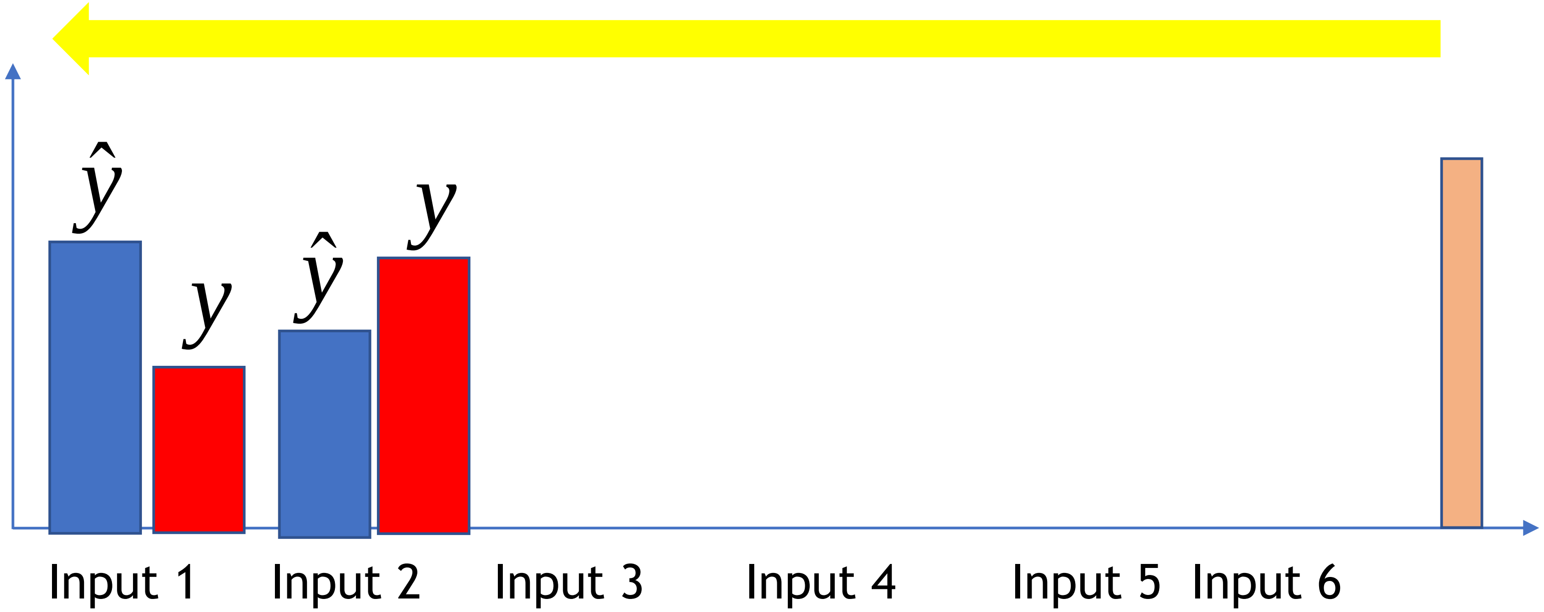
Stochastic Gradient Descent



- Stochastic GD (SGD) : Updating weights after **each** training data sample.
- "Jittering" Provided by SGD : presence of diverse and many data inputs and update done for each data inputs until convergence.
- Probability to get unstuck from local minima and converge towards global minima.

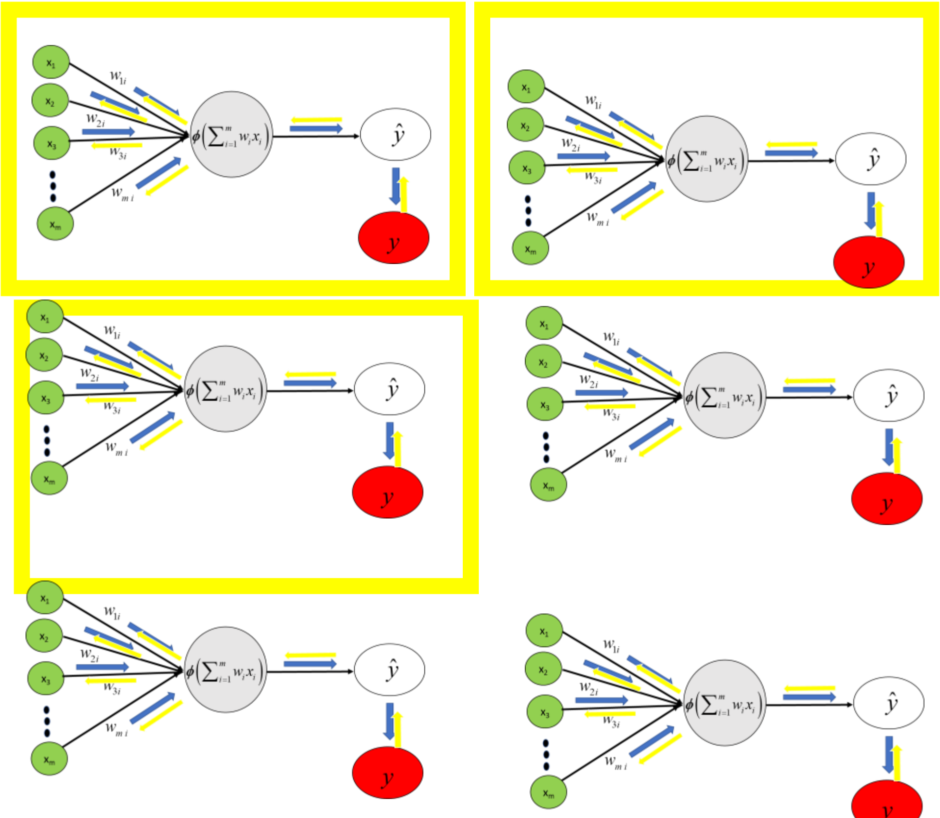
$$E = \frac{1}{2}(y - \hat{y})^2$$

Update all the weights



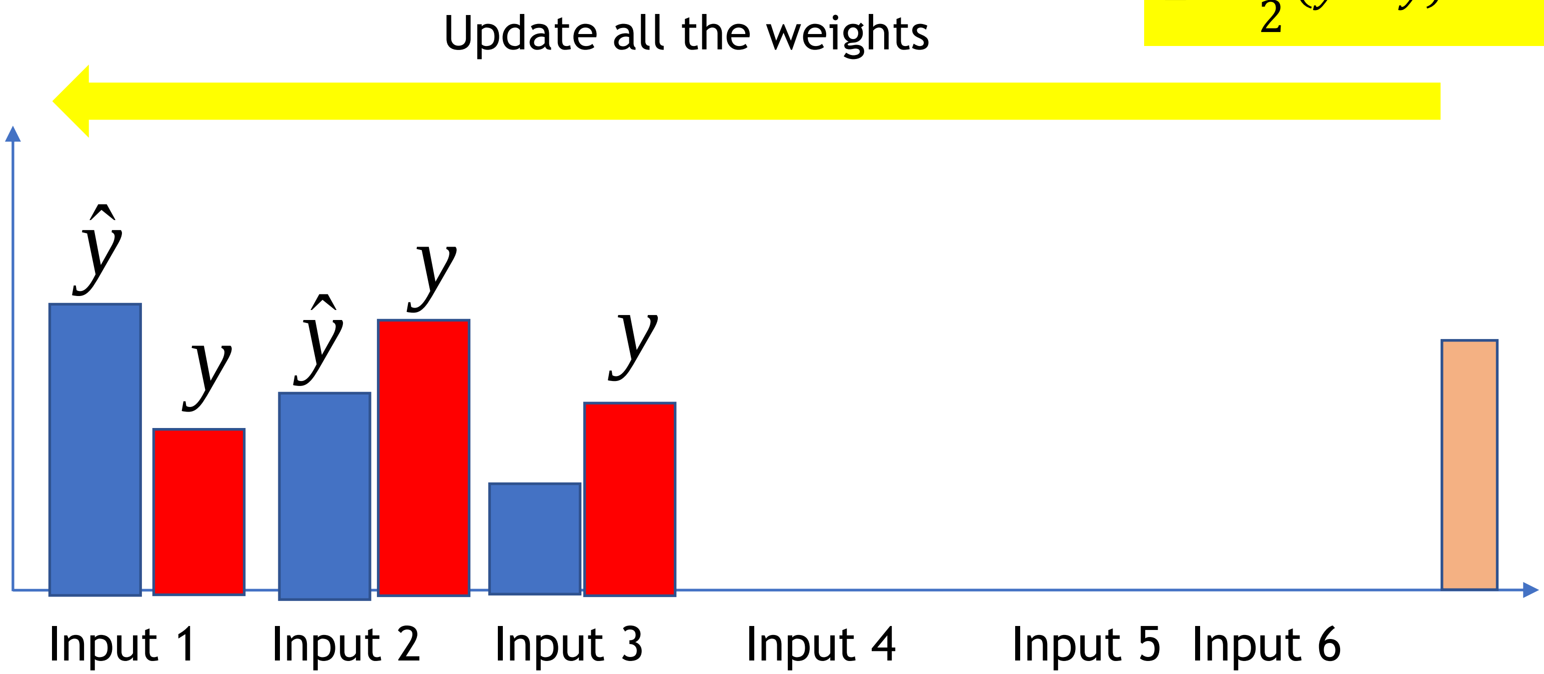
One epoch = training done on entire data set once.

Stochastic Gradient Descent



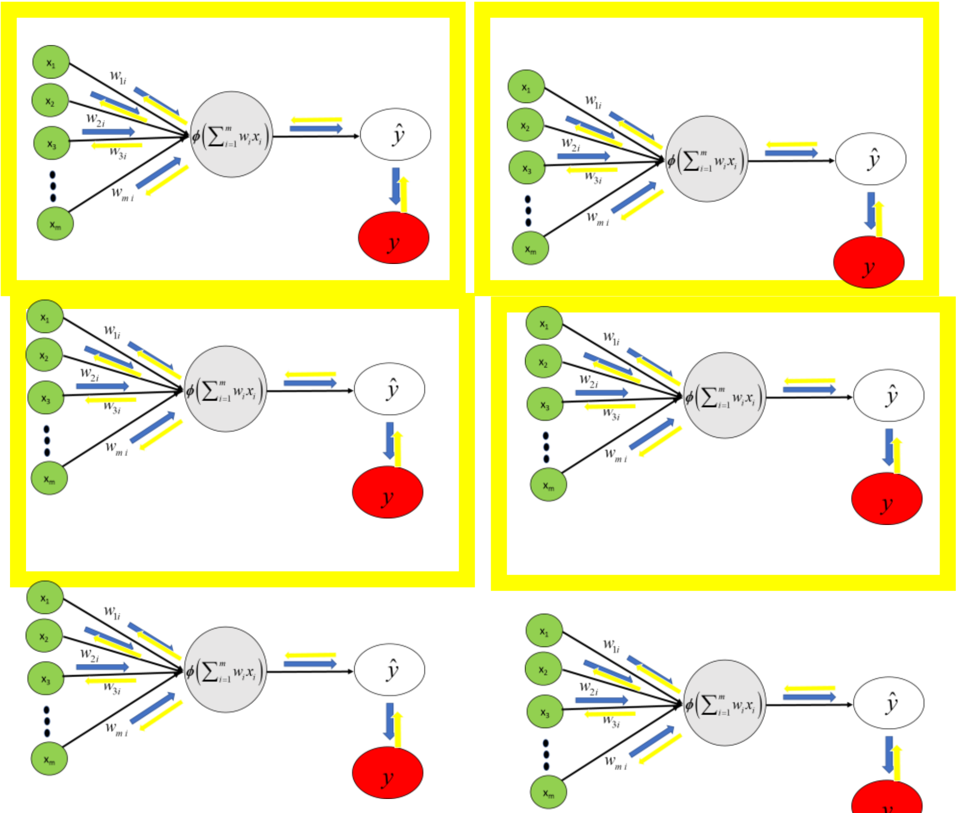
- Stochastic GD (SGD) : Updating weights after **each** training data sample.
- "Jittering" Provided by SGD : presence of diverse and many data inputs and update done for each data inputs until convergence.
- Probability to get unstuck from local minima and converge towards global minima.

$$E = \frac{1}{2}(y - \hat{y})^2$$



One epoch = training done on entire data set once.

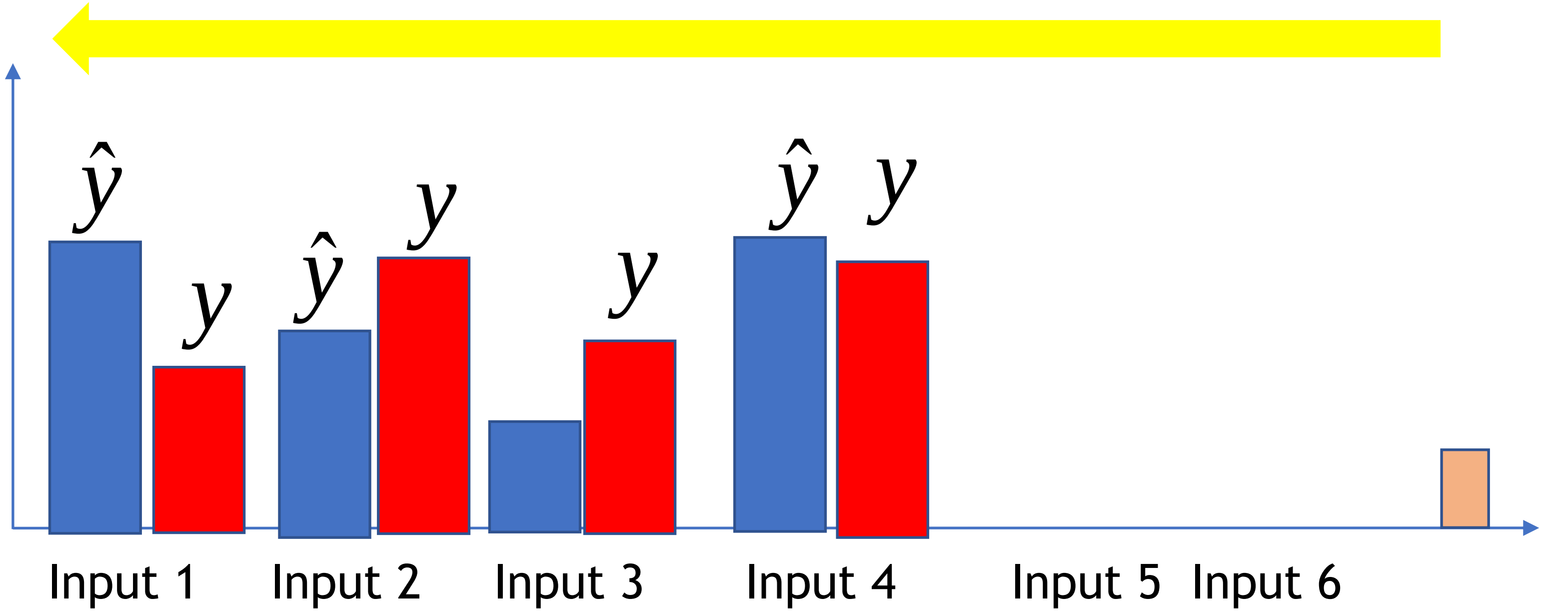
Stochastic Gradient Descent



- Stochastic GD (SGD) : Updating weights after **each** training data sample.
- "Jittering" Provided by SGD : presence of diverse and many data inputs and update done for each data inputs until convergence.
- Probability to get unstuck from local minima and converge towards global minima.

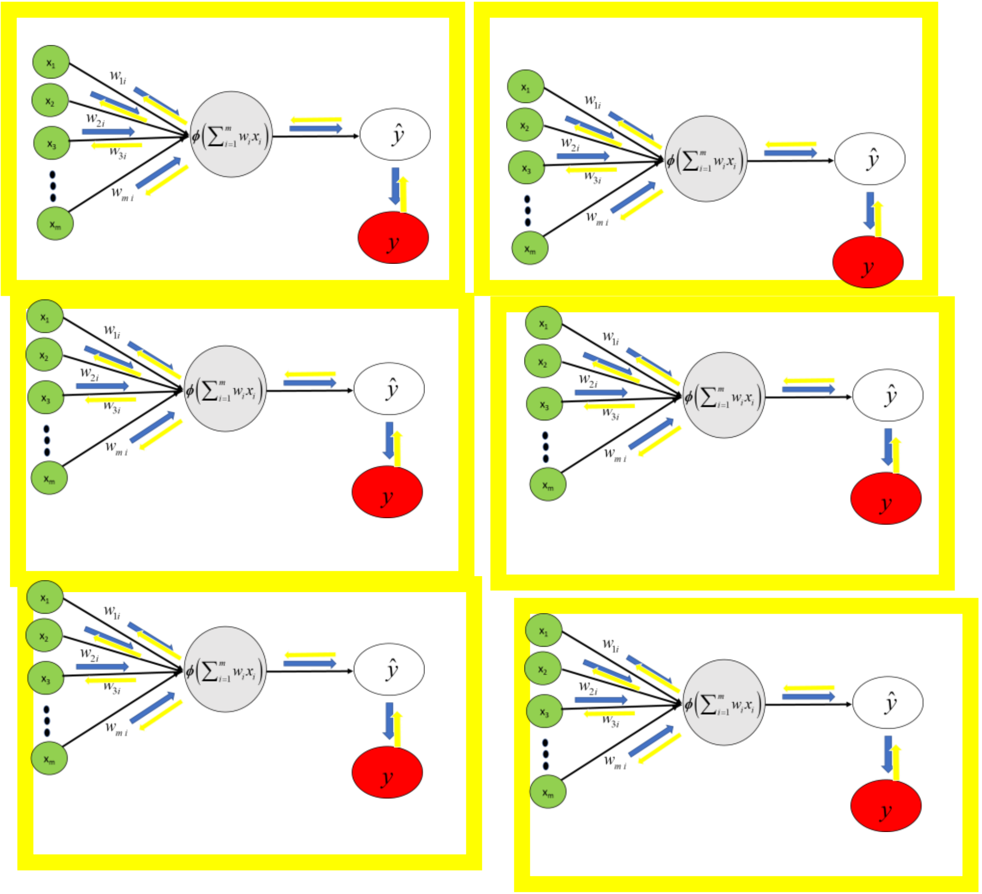
$$E = \frac{1}{2}(y - \hat{y})^2$$

Update all the weights



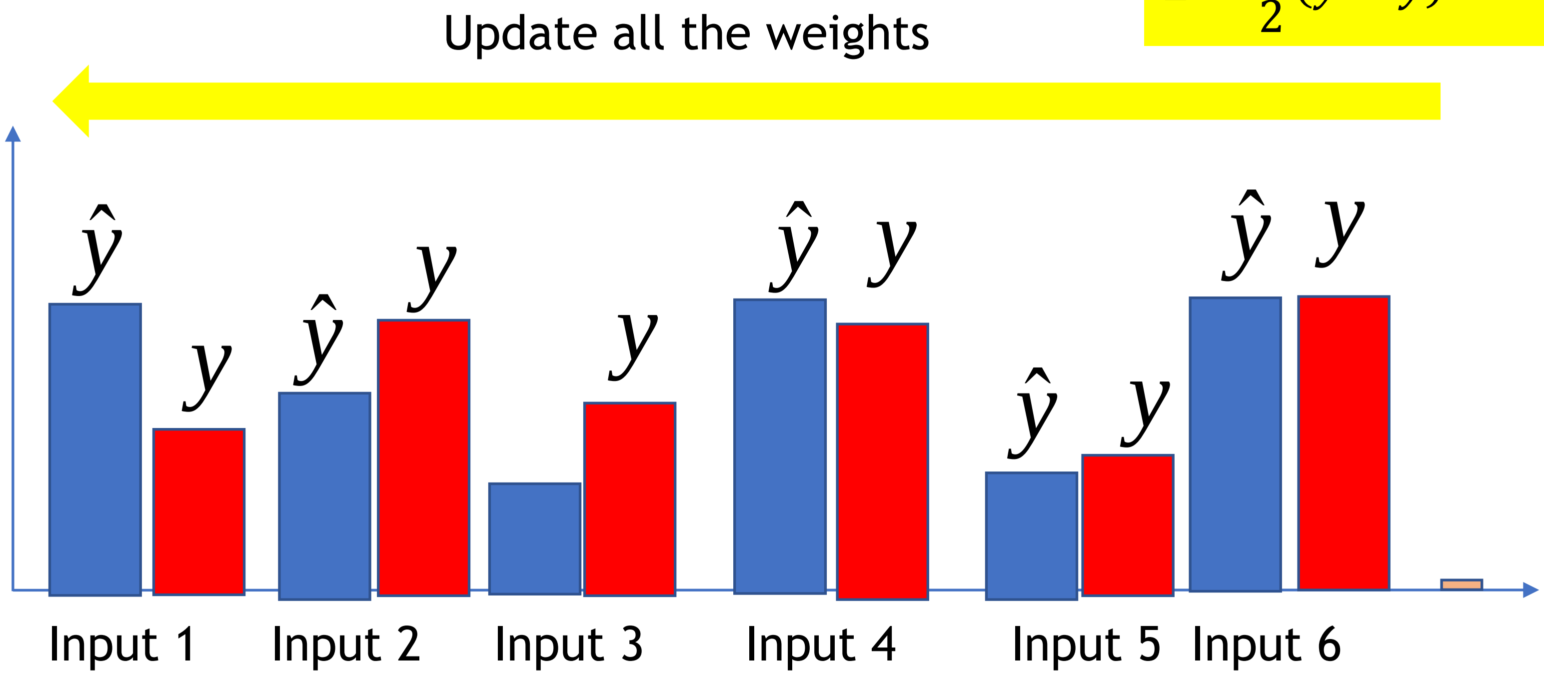
One epoch = training done on entire data set once.

Stochastic Gradient Descent

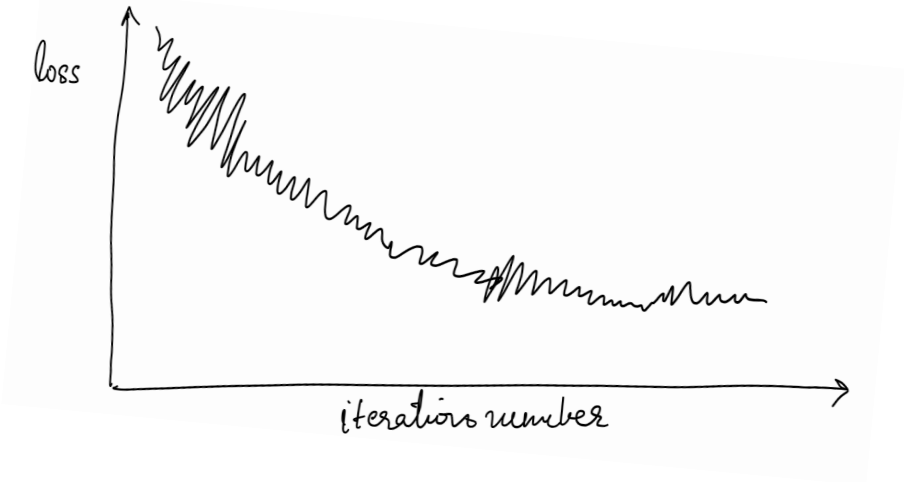


- Stochastic GD (SGD) : Updating weights after **each** training data sample.
- "Jittering" Provided by SGD : presence of diverse and many data inputs and update done for each data inputs until convergence.
- Probability to get unstuck from local minima and converge towards global minima.
- Iterate until convergence detected.

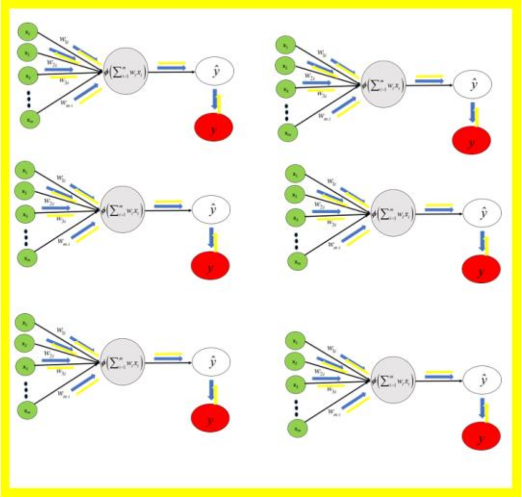
$$E = \frac{1}{2}(y - \hat{y})^2$$



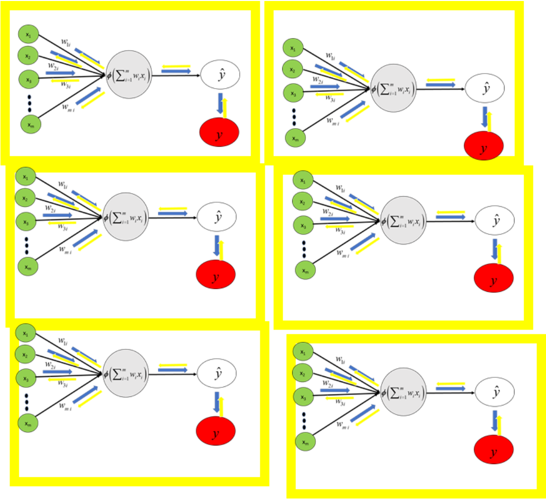
One epoch = training done on entire data set once.



- Stochastic GD (SGD) : Updating weights after **each** training data sample.
- “Jittering” Provided by SGD : presence of diverse and many data inputs and update done for each data inputs until convergence.
- Probability to get unstuck from local minima and converge towards global minima.
- Iterate until convergence detected.



Batch GD



Stochastic GD

Batch GD : stores all data loss, updates after all data loss taken into account.

SGD : updates after each data sample.

- less time consuming
  - NN updated after each data,
  - memory not allocated to all data at once.
  - but, cannot vectorize the computations. (as only one data input treated once).

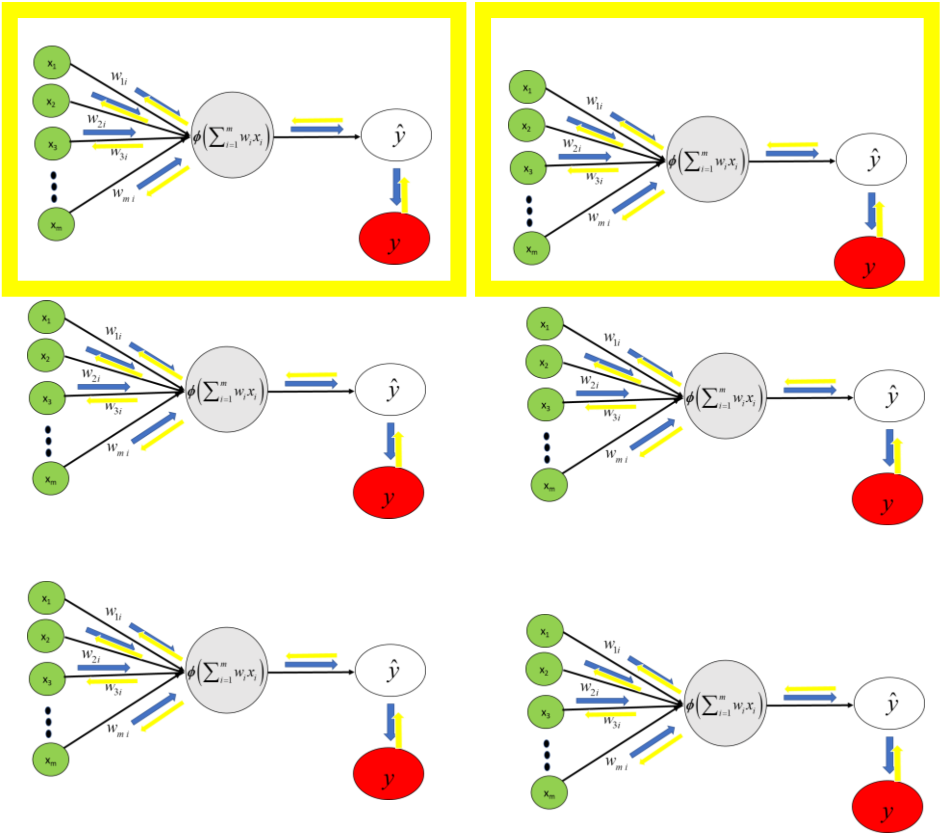
What happens when millions of data samples? but limited memory resources?

Mini batch GD

- Blends advantages of both GD and SGD.
- Mini-batches of fixed size are created.

In **one epoch:**

1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5.

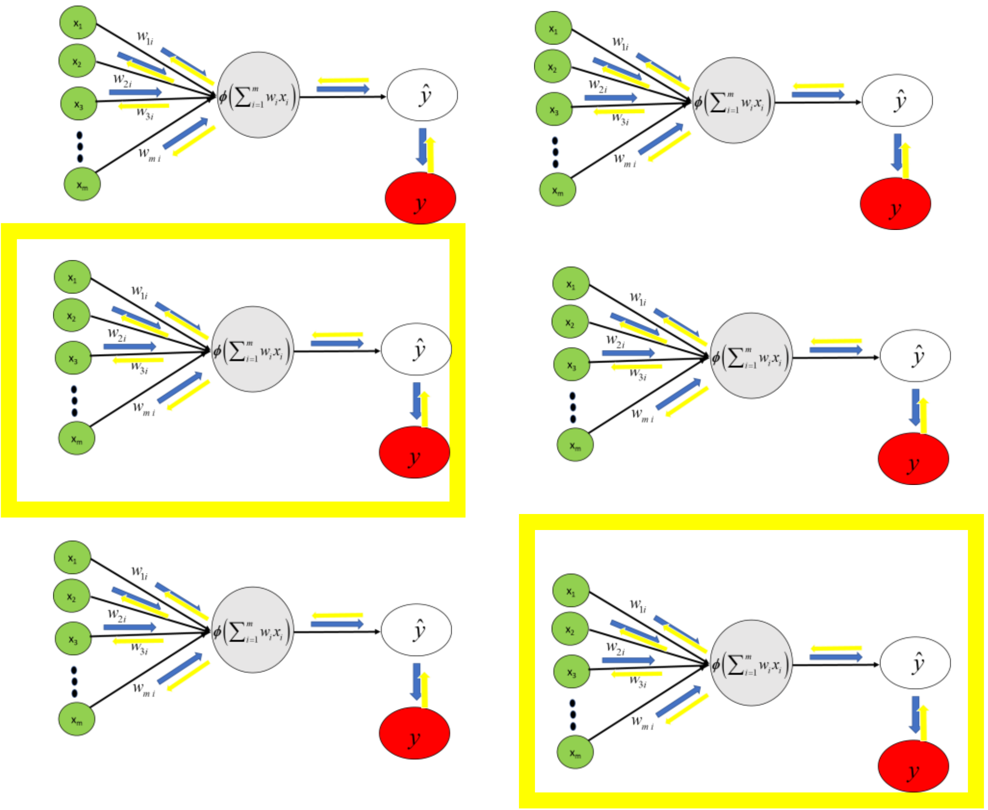


Mini batch GD

- Blends advantages of both GD and SGD.
- Mini-batches of fixed size are created.

In **one epoch:**

1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5. **Repeat steps 1-4 for all the mini-batches**

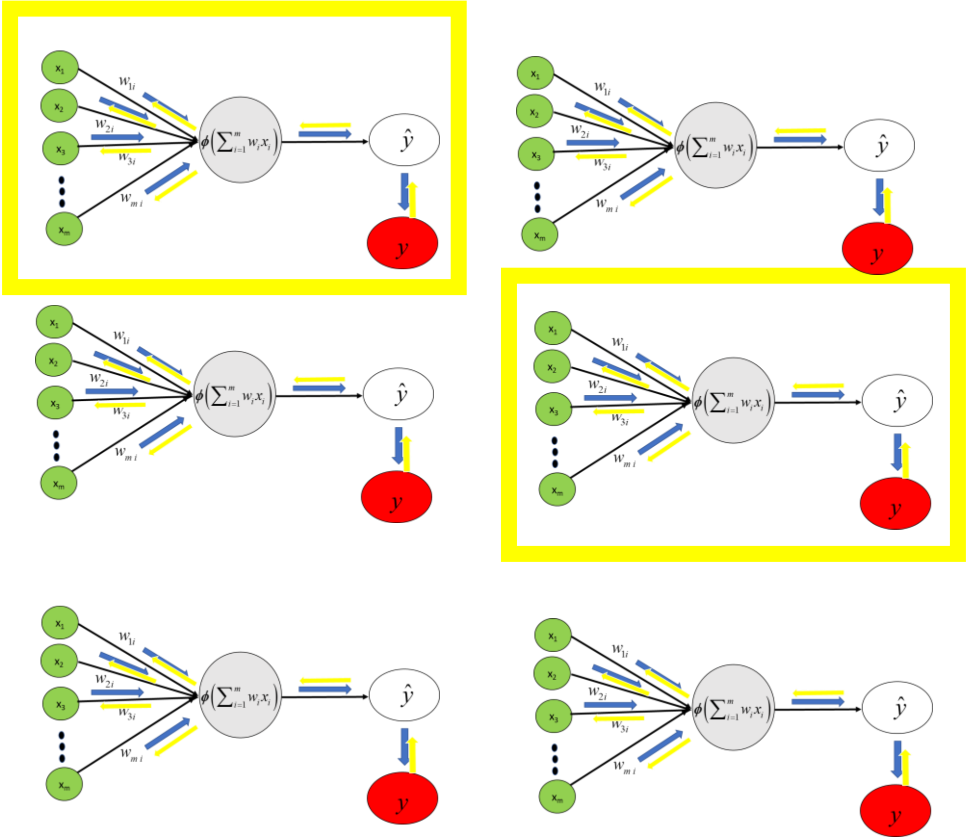


Mini batch GD

- Blends advantages of both GD and SGD.
- Mini-batches of fixed size are created.

In **one epoch:**

1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for all the mini-batches.

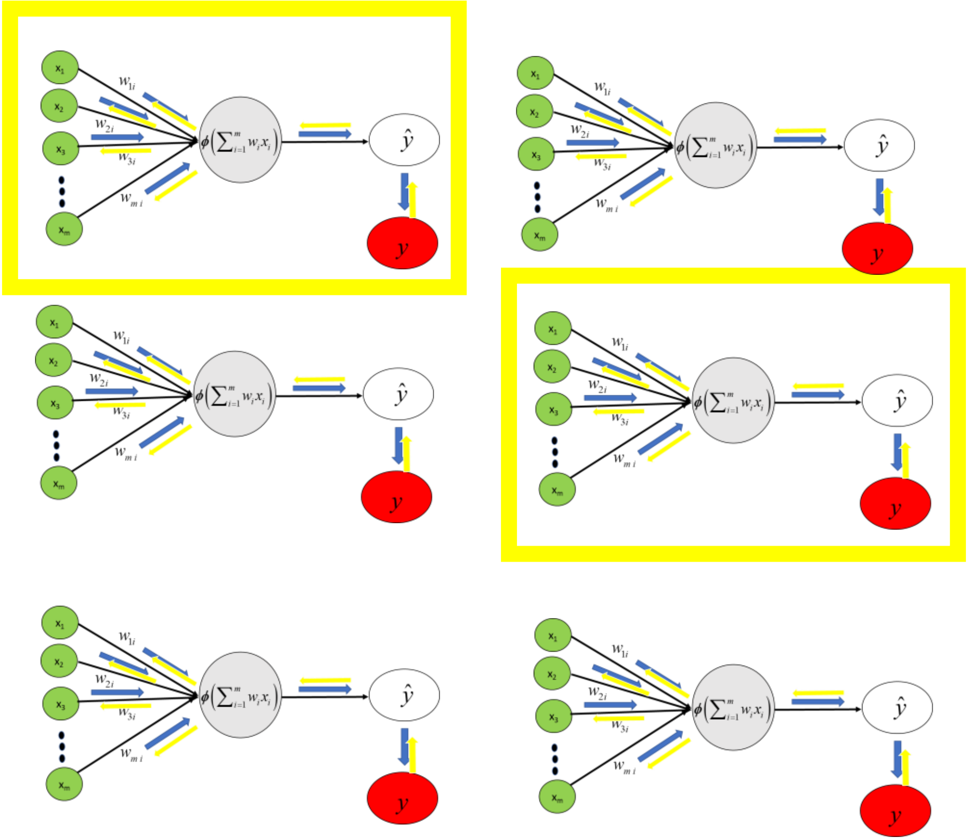


Mini batch GD

- Blends advantages of both GD and SGD.
- Mini-batches of fixed size are created.

In **one epoch:**

1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for the mini-batches we created.



Great!! We now know how NNs update weights …..using:

batch-GD, SGD or mini batch SGD….but…

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial\,(J(\mathbf{w},b))}{\partial\,\mathbf{w}}$$

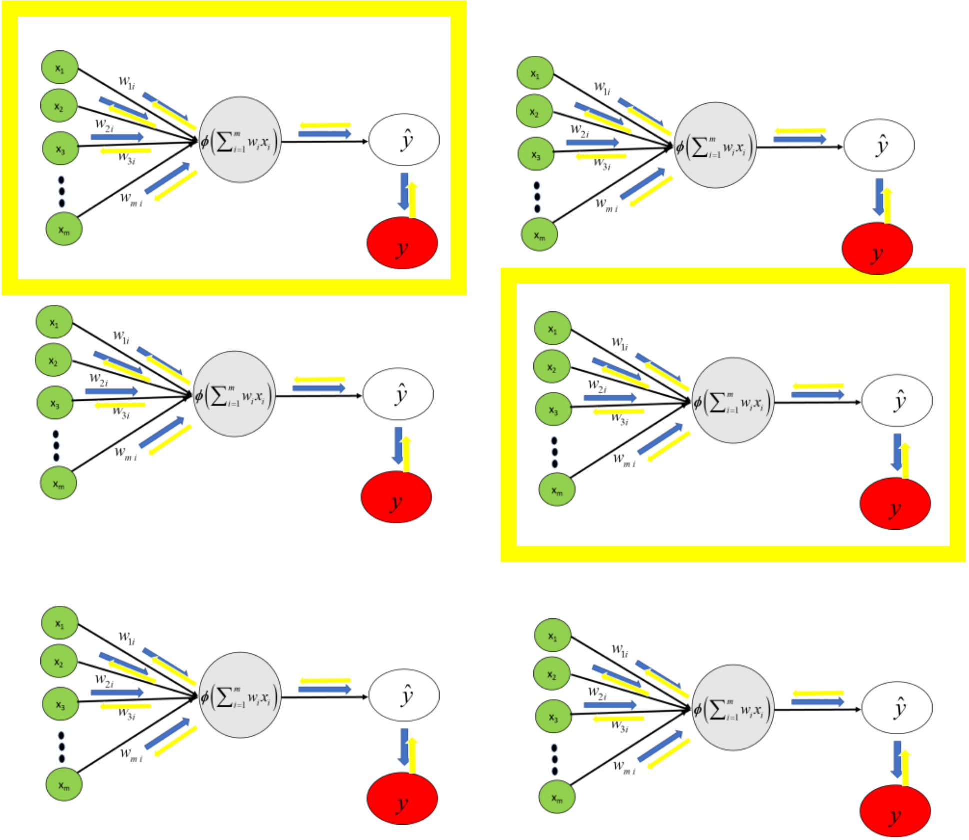
$$b \leftarrow b - \alpha \frac{\partial\,(J(w,b))}{\partial\,b}$$

Mini batch GD

- Blends advantages of both GD and SGD.
- Mini-batches of fixed size are created.

In **one epoch:**

1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for the mini-batches we created.



Great!! We now know how NNs update weights …..using: batch-GD, SGD or mini batch SGD….but…

how to calculate the gradient of the cost function!!

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial\,(J(\mathbf{w},b))}{\partial\,\mathbf{w}}$$

$$b \leftarrow b - \alpha \frac{\partial\,(J(w,b))}{\partial\,b}$$

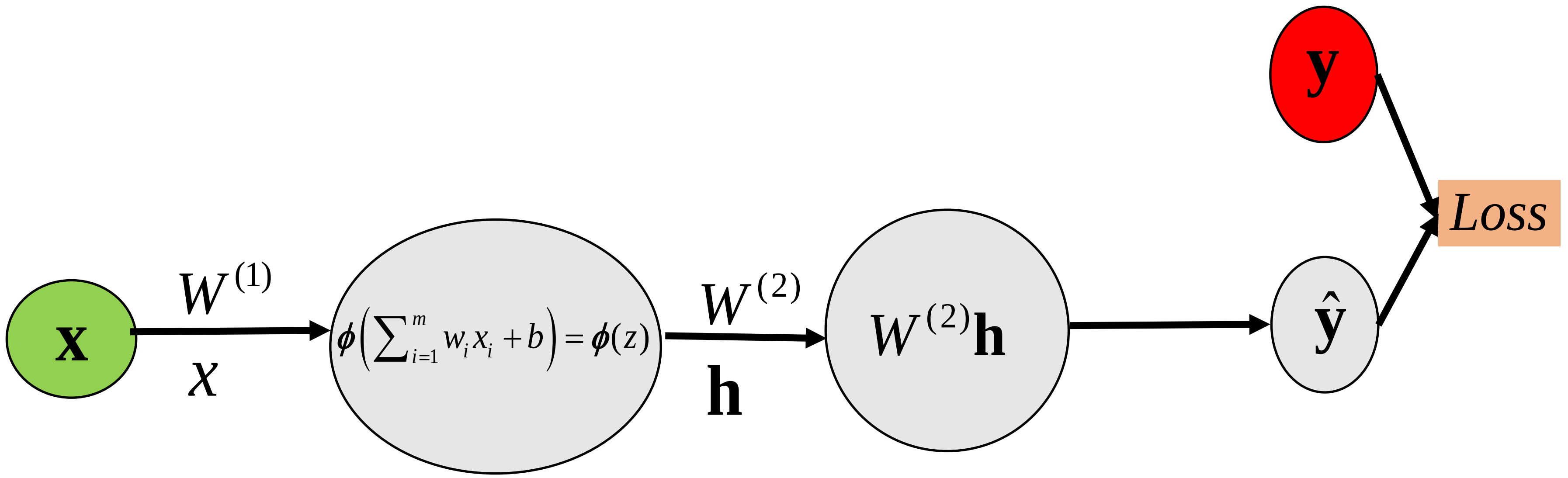
Backpropagation (Backprop)

Back propagation

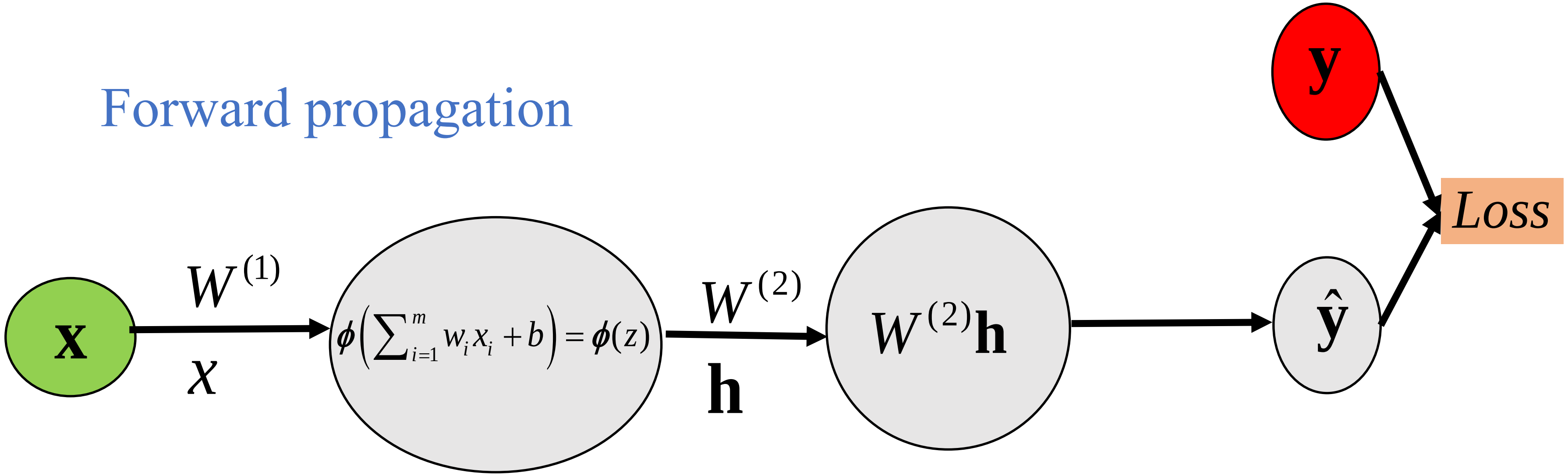
- Intuition: the global error is backward propagated to network nodes, weights are modified proportional to their contribution
- Objective: Calculate rate of change of Error with respect to each weights, to correct the weights.

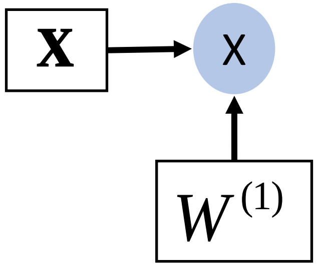
- Backpropagation rediscovered in 1986, efficient way of propagating backwards the error gradient and updating the weights.

but first, Forward Propagation : Illustration using **2 Hidden layer Deep NN**.

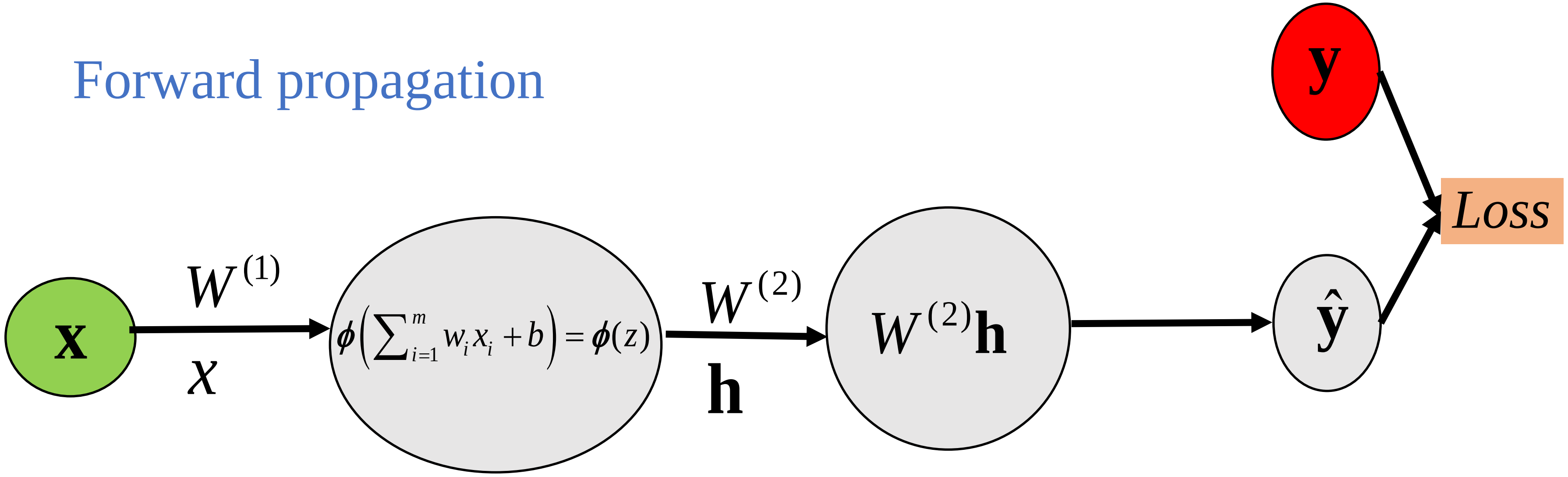


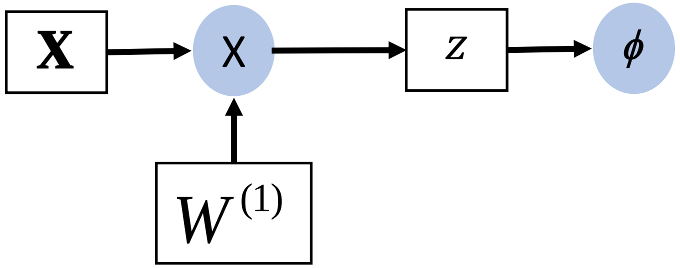
Forward propagation





Forward propagation





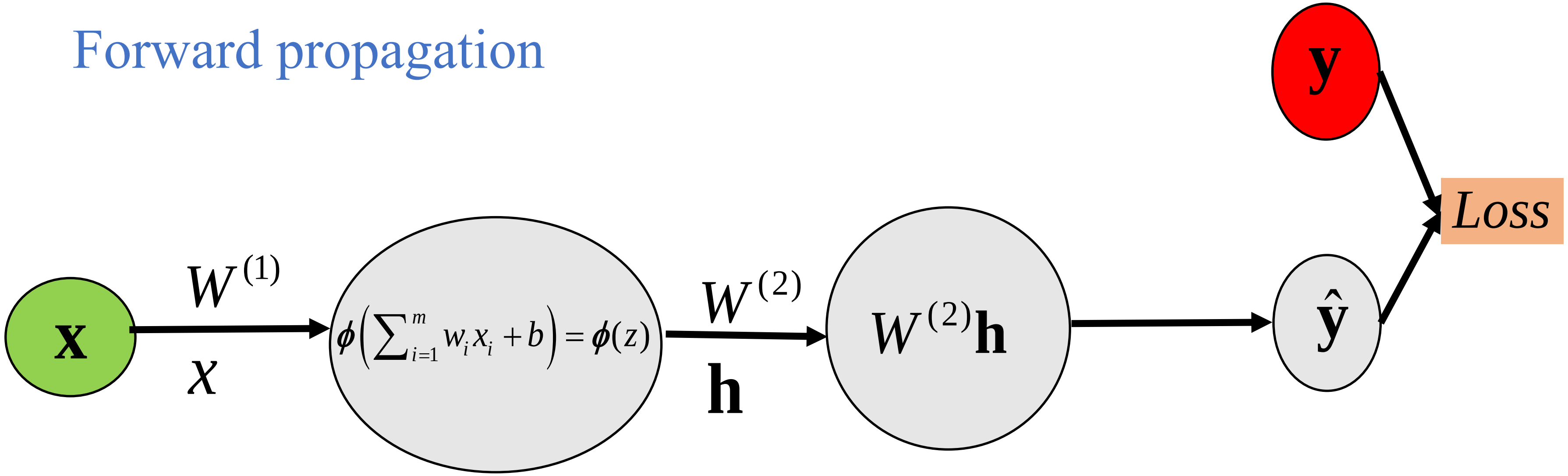
$$z = W^{(1)} x$$

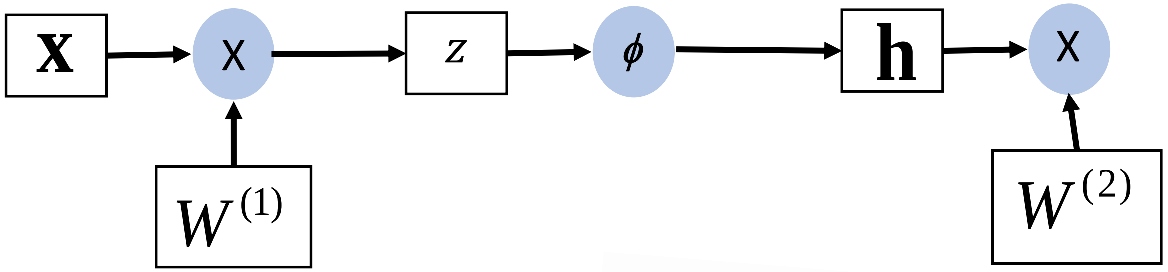
$x \in \mathbb{R}^d$

$W^{(1)} \in \mathbb{R}^{h \times d}$

$z \in \mathbb{R}^h$

Forward propagation





$z = W^{(1)} x$

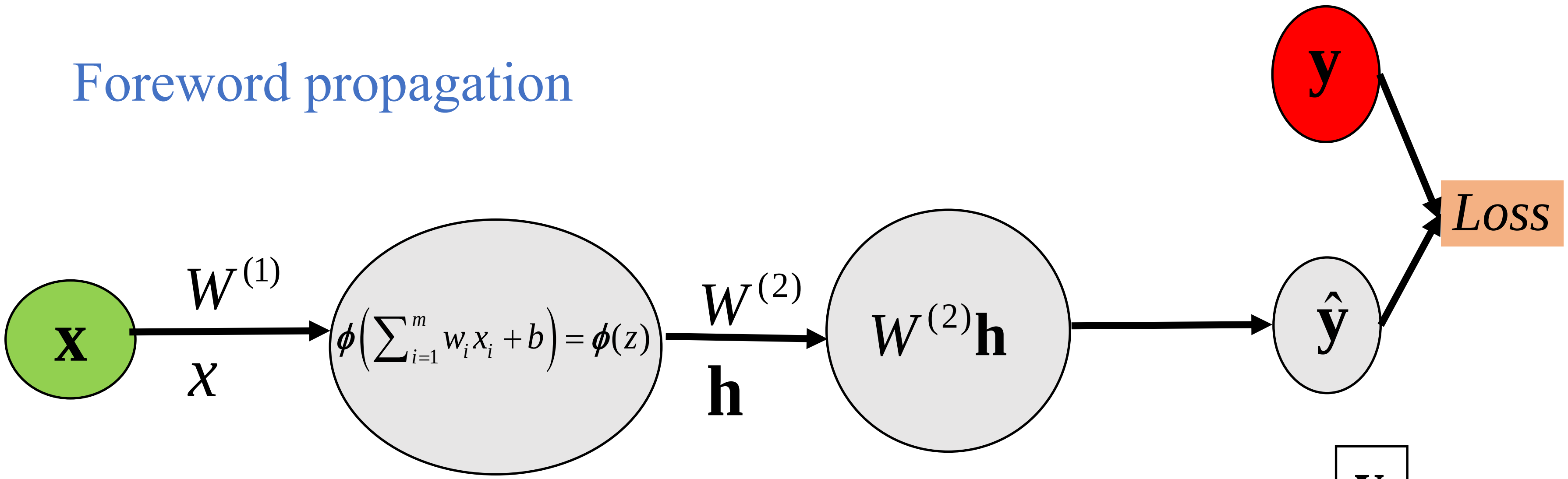
$h = \phi(z)$

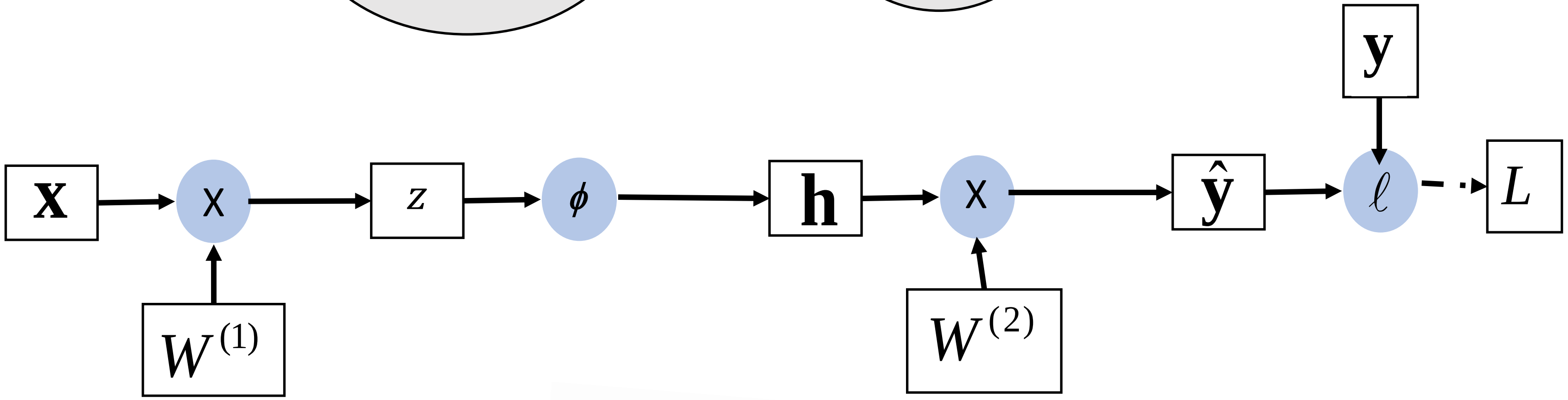
$x \in \mathbb{R}^d$

$W^{(1)} \in \mathbb{R}^{h \times d}$

$z \in \mathbb{R}^h$

Foreword propagation





$z = W^{(1)} x$

$x \in \mathbb{R}^d$
$W^{(1)} \in \mathbb{R}^{h \times d}$
$z \in \mathbb{R}^h$

$h = \phi(z)$

$\hat{y} = W^{(2)} h$

$W \in \mathbb{R}^{q \times h}$

$L = \ell(\hat{y}, y)$

$J = L$

Back propagation

- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

$$Y = f(x)$$
$$Z = g(Y) = g \circ f(x)$$
Then, $\frac{\partial Z}{\partial x} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial x}\right)$

Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial W^{(2)}}$$

Back propagation

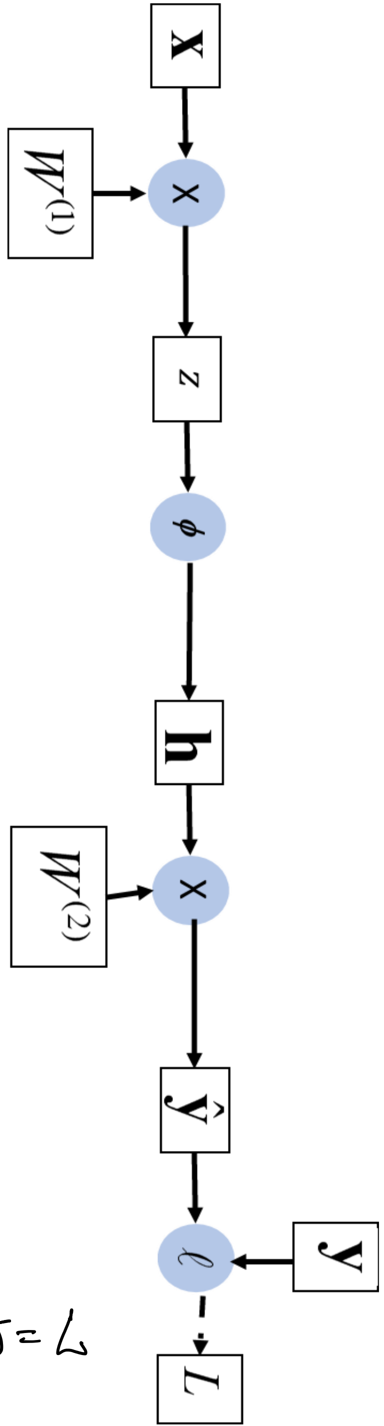
- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

$$Y = f(x)$$
$$Z = g(Y) = g \circ f(x)$$
Then, $\frac{\partial Z}{\partial x} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial x}\right)$

Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \quad \frac{\partial J}{\partial W^{(2)}}$$



$\frac{\partial J}{\partial L} = 1$ as $J = L$

Back propagation

- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

$$Y = f(x)$$
$$Z = g(Y) = g \circ f(x)$$
Then, $\frac{\partial Z}{\partial x} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial x}\right)$

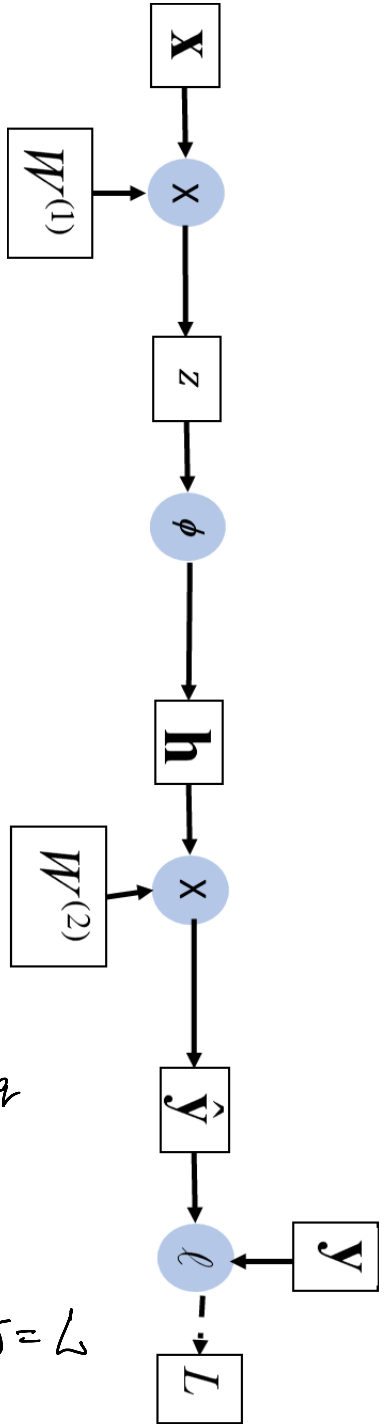
Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial W^{(2)}}$$

Gradient of objective function wrt output layer variables $\hat{y}$

$$\frac{\partial J}{\partial \hat{y}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \hat{y}}\right) = \frac{\partial L}{\partial \hat{y}} \in \mathbb{R}^q$$

$$\frac{\partial J}{\partial L} = 1 \quad \text{as } J = L$$



Back propagation

- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

$$Y = f(x)$$
$$Z = g(Y) = g\circ f(x)$$
Then, $\frac{\partial Z}{\partial x} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial x}\right)$

Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \quad \frac{\partial J}{\partial W^{(2)}}$$

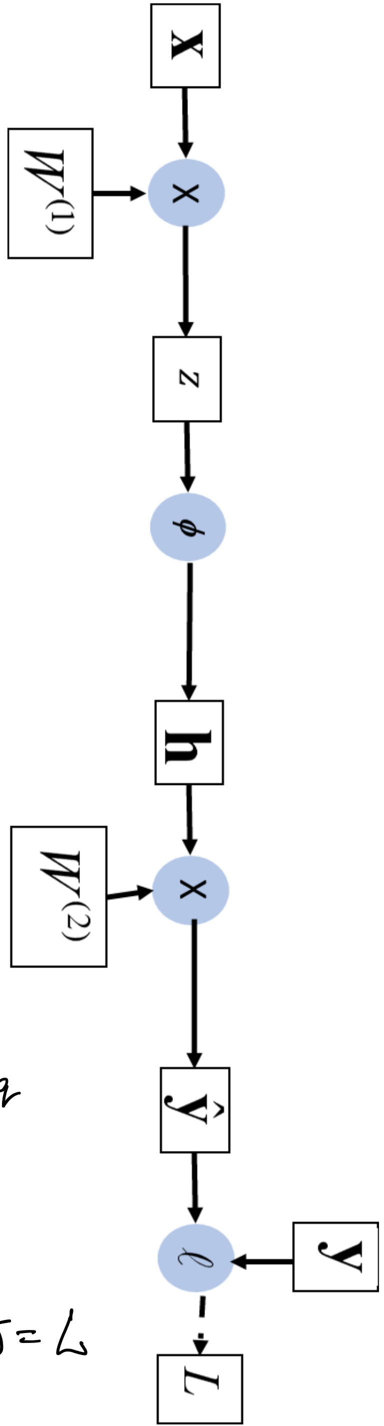
Gradient wrt $W^{(2)}$

$$\frac{\partial J}{\partial W^{(2)}} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial W^{(2)}}\right)$$

Gradient of objective function wrt output layer variables $\hat{y}$

$$\frac{\partial J}{\partial \hat{y}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \hat{y}}\right) = \frac{\partial L}{\partial \hat{y}} \in \mathbb{R}^q$$

$$\frac{\partial J}{\partial L} = 1 \quad \text{as} \quad J = L$$



Back propagation

- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

$$Y = f(x)$$
$$Z = g(Y) = g \circ f(x)$$
Then, $\frac{\partial Z}{\partial x} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial x}\right)$

Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial W^{(2)}}$$

$$\frac{\partial J}{\partial h} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial h}\right) = W^{(2)^T} \frac{\partial J}{\partial \hat{y}}$$

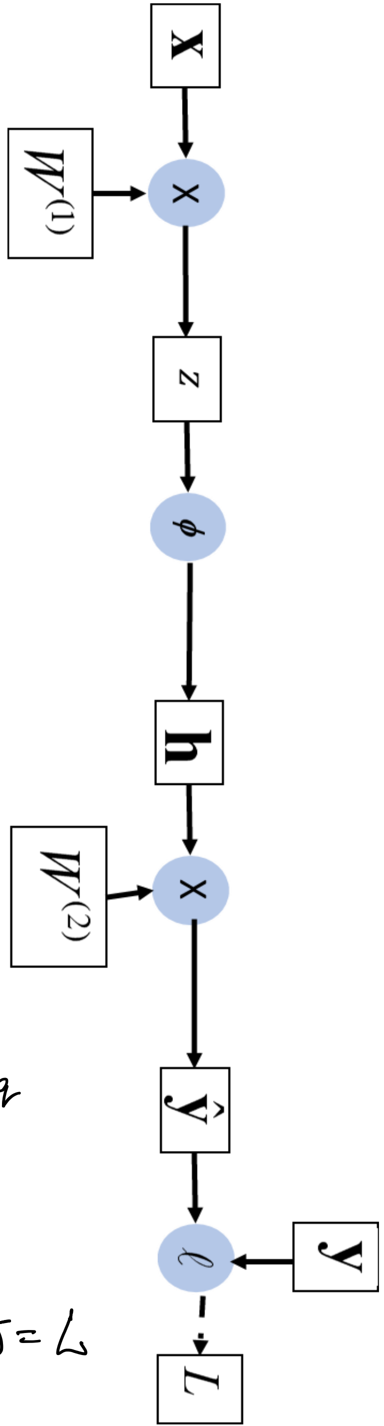
Gradient wrt $W^{(2)}$

$$\frac{\partial J}{\partial W^{(2)}} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial W^{(2)}}\right)$$

Gradient of objective function wrt output layer variables $\hat{y}$

$$\frac{\partial J}{\partial \hat{y}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \hat{y}}\right) = \frac{\partial L}{\partial \hat{y}} \in \mathbb{R}^q$$

$$\frac{\partial J}{\partial L} = 1 \quad \text{as } J = L$$



Back propagation

- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

$$Y = f(x)$$
$$Z = g(Y) = g \circ f(x)$$
Then, $\frac{\partial Z}{\partial x} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial x}\right)$

$$\frac{\partial J}{\partial z} = \text{prod}\left(\frac{\partial J}{\partial h}, \frac{\partial h}{\partial z}\right) = \frac{\partial J}{\partial h} \odot \phi'(z)$$

$$\frac{\partial J}{\partial h} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial h}\right) = W^{(2)T} \frac{\partial J}{\partial \hat{y}}$$

Gradient wrt $W^{(2)}$

$$\frac{\partial J}{\partial W^{(2)}} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial W^{(2)}}\right)$$

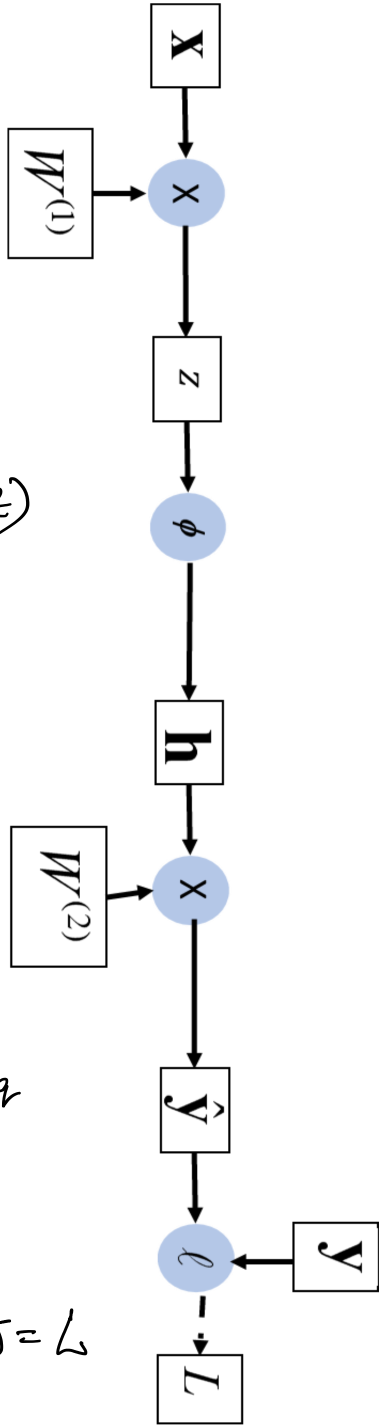
Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial W^{(2)}}$$

Gradient of objective function wrt output layer variables $\hat{y}$

$$\frac{\partial J}{\partial \hat{y}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \hat{y}}\right) = \frac{\partial L}{\partial \hat{y}} \in \mathbb{R}^q$$

$$\frac{\partial J}{\partial L} = 1 \quad \text{as} \quad J = L$$



Back propagation

- Calculate the gradient with respect to all parameters.
- Intermediate values and gradients are calculated.

- Reminder: Chain rule

$$Y = f(x)$$
$$Z = g(Y) = g \circ f(x)$$

Then, $\frac{\partial Z}{\partial x} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial x}\right)$

Objective of Backprop:

$$\frac{\partial J}{\partial W^{(1)}}, \frac{\partial J}{\partial W^{(2)}}$$

$$\frac{\partial J}{\partial W^{(1)}} = \text{prod}\left(\frac{\partial J}{\partial z}, \frac{\partial z}{\partial W^{(1)}}\right) = \frac{\partial J}{\partial z} x^T$$

$$\frac{\partial J}{\partial z} = \text{prod}\left(\frac{\partial J}{\partial h}, \frac{\partial h}{\partial z}\right) = \frac{\partial J}{\partial h} \odot \phi'(z)$$

$$\frac{\partial J}{\partial h} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial h}\right) = W^{(2)T} \frac{\partial J}{\partial \hat{y}}$$

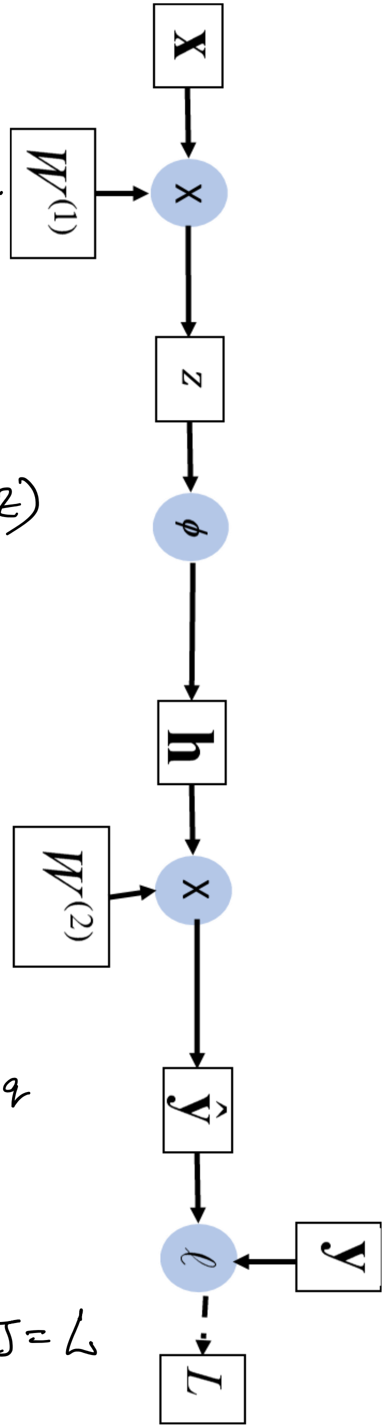
Gradient wrt $W^{(2)}$

$$\frac{\partial J}{\partial W^{(2)}} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial W^{(2)}}\right)$$

Gradient of objective function wrt output layer variables $\hat{y}$

$$\frac{\partial J}{\partial \hat{y}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \hat{y}}\right) = \frac{\partial L}{\partial \hat{y}} \in \mathbb{R}^q$$

$$\frac{\partial J}{\partial L} = 1 \quad \text{as } J = L$$



Deep NN Training Algorithm

On-Line algorithm:

1. Initialize weights

Deep NN Training Algorithm

On-Line algorithm:

1. Initialize weights

2. Present the data input and targets for the deep NN

Forward propagation: Traverse the computational graph in the direction of dependencies and compute all the variables on its path.

Deep NN Training Algorithm

On-Line algorithm:

1. Initialize weights

2. Present the data input and targets for the deep NN

Forward propagation: Traverse the computational graph in the direction of dependencies and compute all the variables on its path.

3. Compute Deep NN output

4. Back propagation of errors

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

where, $\Delta w_{ij} = -\alpha \frac{\partial J}{\partial w_{ij}}$

5.Update all the weights using Gradient descent:

Deep NN Training Algorithm

On-Line algorithm:

1. Initialize weights
2. Present the data input and targets for the deep NN

Forward propagation: Traverse the compute graph in the direction of dependencies and compute all the variables on its path.

3. Compute Deep NN output
4. Back propagation of errors
5. Update all the weights using Gradient descent:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

where, $\Delta w_{ij} = -\alpha \frac{\partial J}{\partial w_{ij}}$

6. Repeat the steps from 2 , until acceptable error levels observed .

Remarks:

- intermediate values must be stored until backpropagation
- backpropagation requires significantly more memory than plain inference.
- Gradients as tensors variables must be stored to invoke the chain rule.
- Minibatches → GD on several data inputs together ➔ more intermediate activations need to be stored.

Deep NN Training Algorithm

On-Line algorithm:

1. **Initialize weights.** **How? what is the best way?**

2. Present the data input and targets for the deep NN

Forward propagation: Traverse the compute graph in the direction of dependencies and compute all the variables on its path.

3. Compute Deep NN output

4. Back propagation of errors

5.Update all the weights using Gradient descent:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\text{where, } \Delta w_{ij} = -\alpha \frac{\partial J}{\partial w_{ij}}$$

6. Repeat the steps from 2 , until acceptable error levels observed.

How to access? What is the best model? When is training over?

Remarks:

- intermediate values must be stored until backpropagation
- backpropagation requires significantly more memory than plain inference.
- Gradients as tensors variables must be stored to invoke the chain rule.
- Minibatches → GD on several data inputs together ➔ more intermediate activations need to be stored.

Summary

- Forward propagation sequentially calculates and stores intermediate variables within the compute graph defined by the neural network. It proceeds from input to output layer.

- Back propagation sequentially calculates and stores the gradients of intermediate variables and parameters within the neural network in the reversed order.

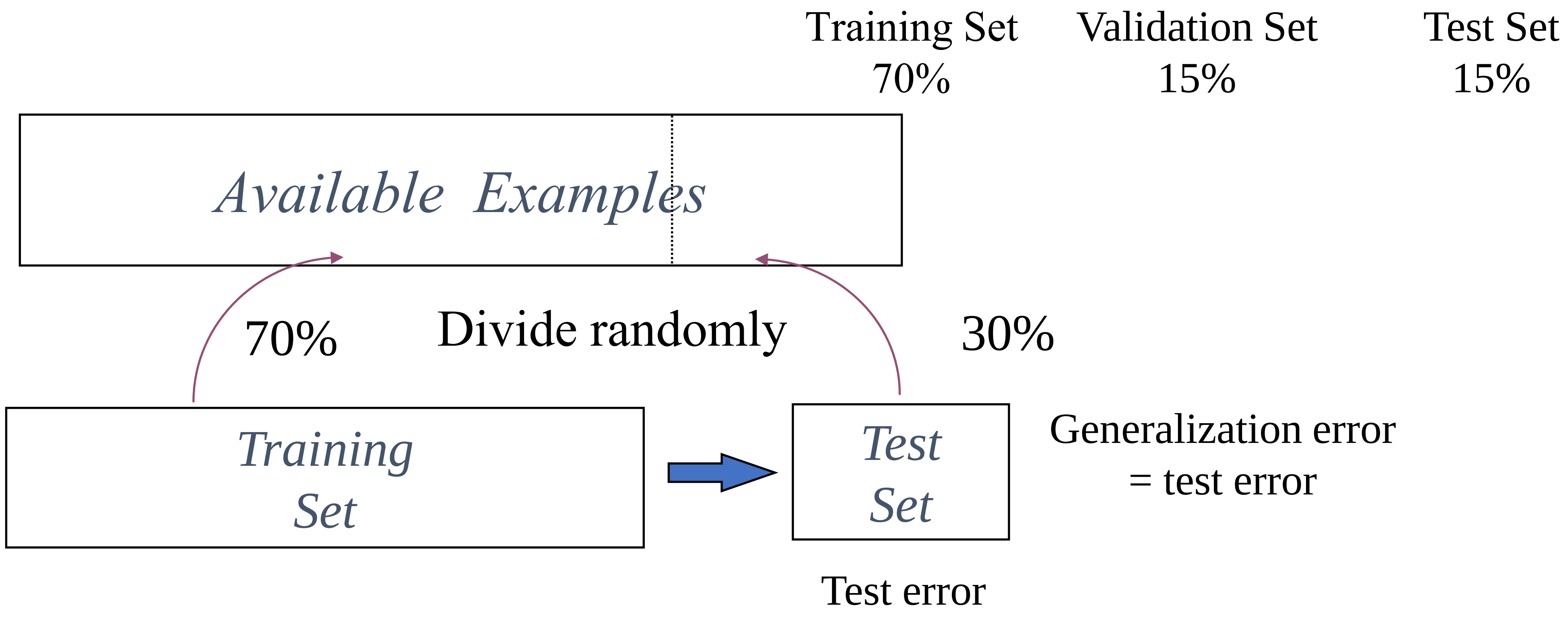
- When training deep learning models, forward propagation and back propagation are interdependent.

- Training requires significantly more memory and storage.

Training data, Test Data and Validation data

Rich and large data sets: Different data sets for training, parameter tuning and testing of the model.

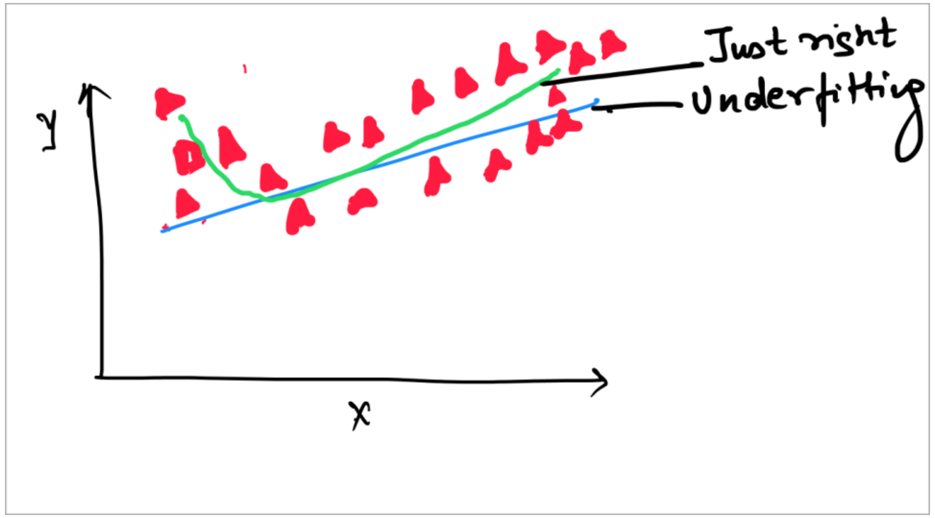
When amount of data is large

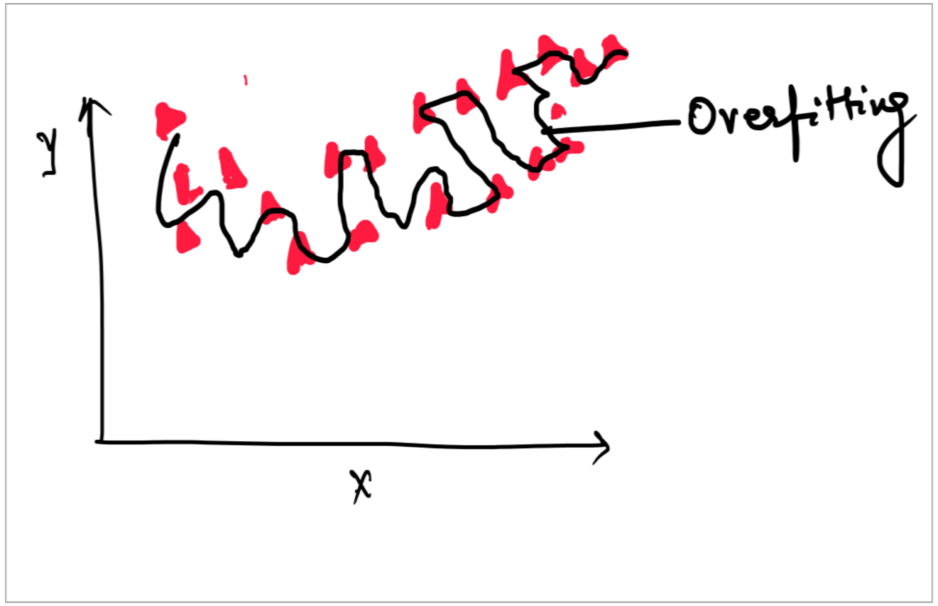


Generalization: Underfitting and Overfitting

- Under fitting: model is unable to reduce training errors.

- Overfitting: model test error is significantly higher than training error.



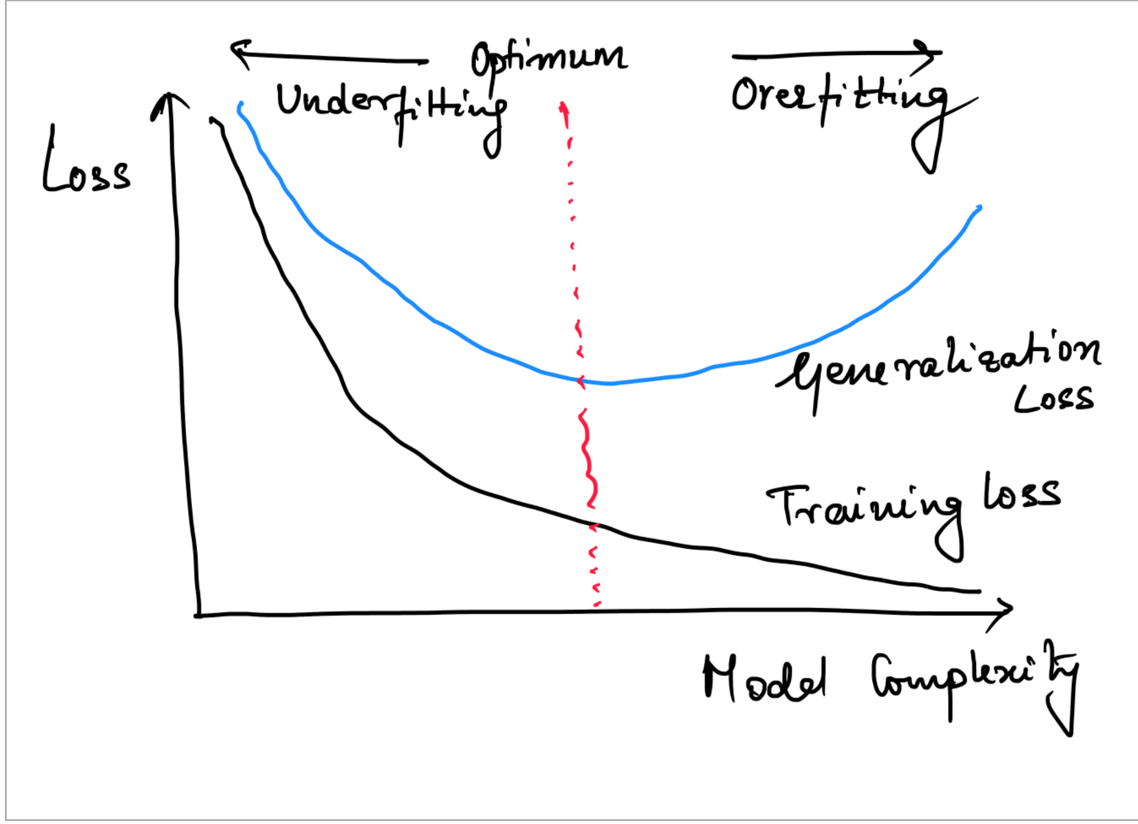


Generalization: Underfitting and Overfitting

- Under fitting: model is unable to reduce training errors.

- Overfitting: model test error is significantly higher than training error.

How does it depend on Model complexity?



Underfitting and Overfitting

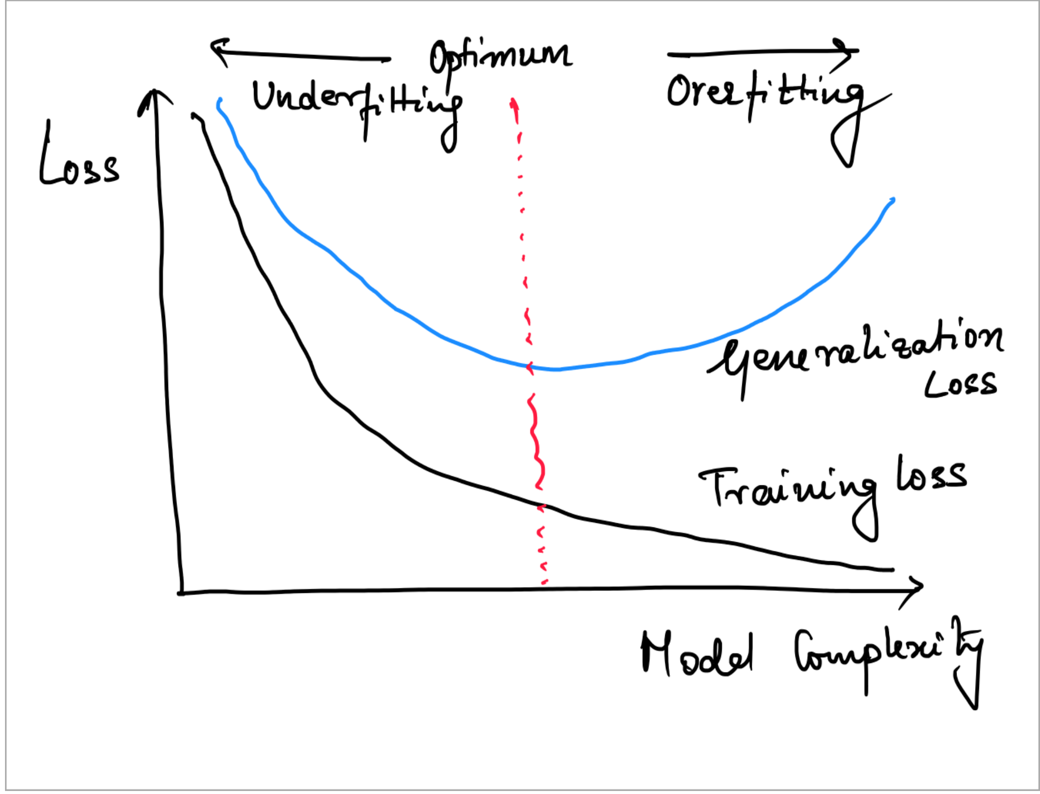
- Under fitting: model is unable to reduce training errors.

- Overfitting: model test error is significantly higher than training error.

How does it depend on Model complexity?

What is model complexity?

- number of hyper-parameters (tunable parameters)
- number of layers, hidden nodes in each layer
- number of weights, range of values taken by weights
- Minibatch size



Generalization : Preventing over-fitting (over-training)

Goal: To achieve good generalization accuracy on new examples/cases

How to ensure that a network has been well trained??

1. Rich and large data sets: Different data sets for training, parameter tuning and testing of the model.

- Monitor error on the test set as network trains.
- Stop network training just prior to over-fit error occurring - *early stopping* or tuning

2. Number of effective weights is reduced : Number of weights and value range.

Generalization : Preventing over-fitting (over-training)

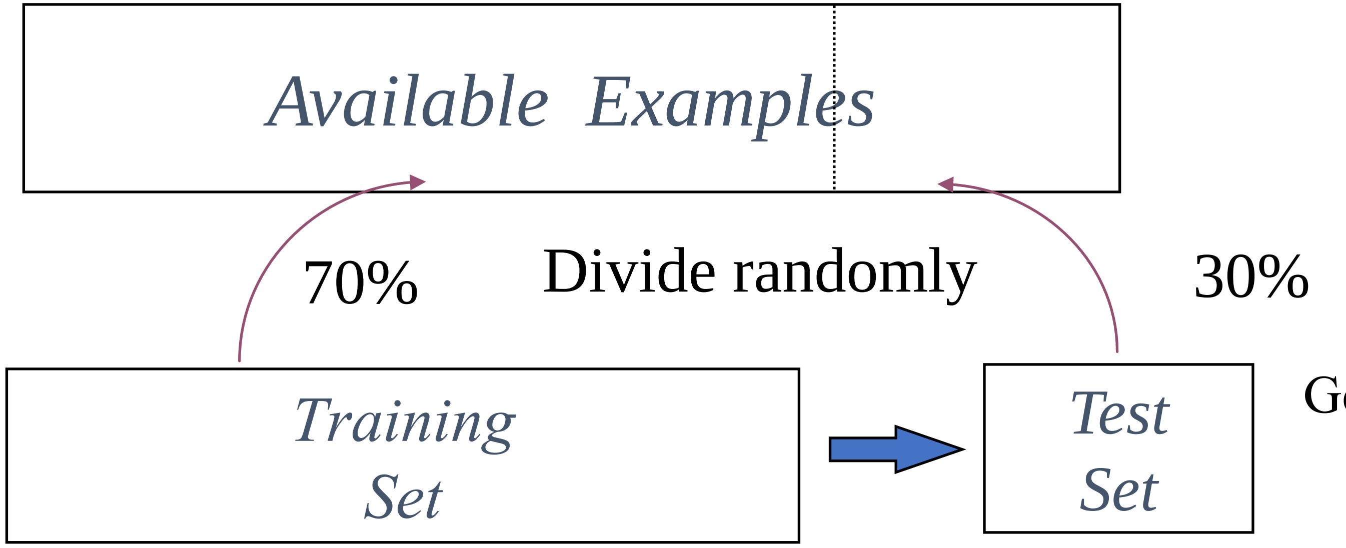
Goal: To achieve good generalization accuracy on new examples/cases

How to ensure that a network has been well trained??

1. **Rich and large data sets: Different data sets for training, parameter tuning and testing of the model.**

Training Set	Validation Set	Test Set
70%	15%	15%

When amount of data is large



Generalization error = test error

Test error

Generalization : Preventing over-fitting (over-training)

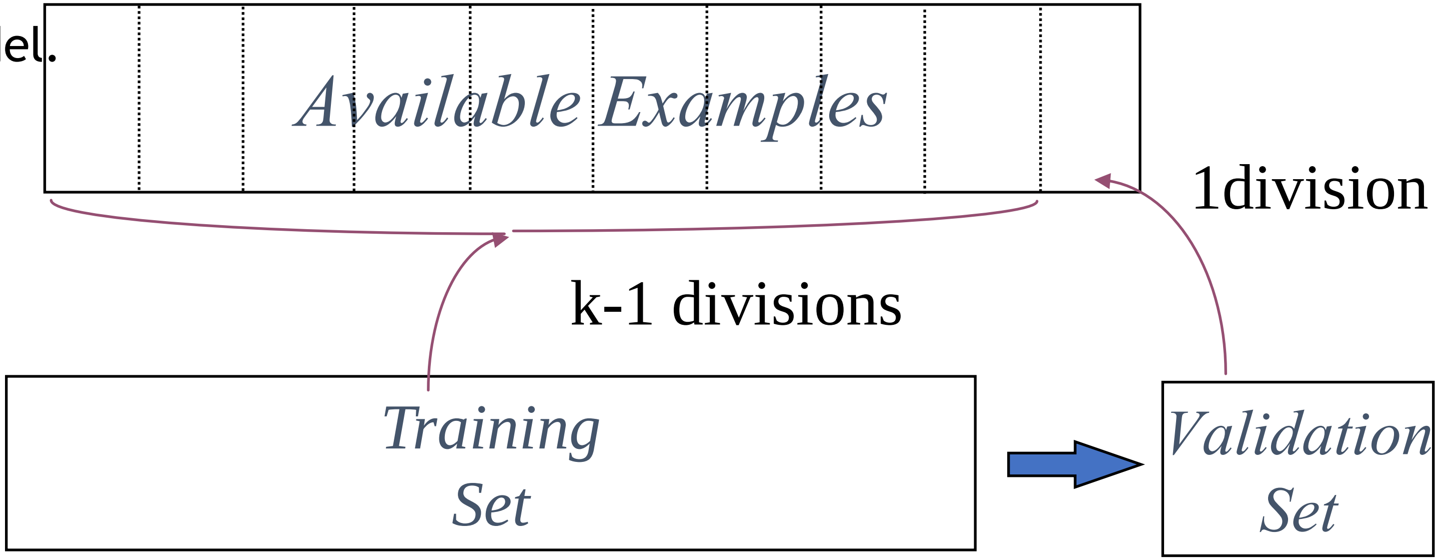
Goal: To achieve good generalization accuracy on new examples/cases

How to ensure that a network has been well trained??

1.Rich and large data sets: Different data sets for training, parameter tuning and testing of the model.

When amount of data is small: Cross-Validation (K-fold)

- original training data set is split into K noncoincident sub-data sets
- use the K -1 sub-data set to train the model.
- validate the model using a sub-data set
- Repeat model training and validation process *k* times.



Generalization : Preventing over-fitting (over-training)

2. How to control number of effective weights?

- Manually or automatically select optimum number of hidden nodes and connections.
  - Not scalable, often needs expert opinion.

- Regularization methods
  - Adjust the bp error function to penalize the growth of unnecessary weights
  - Keep the weight vector small magnitude ➔add its value as a penalty to the problem of minimizing the loss.

Generalization : Preventing over-fitting (over-training)

2. How to control number of effective weights?

- Manually or automatically select optimum number of hidden nodes and connections.
  - Not scalable, often needs expert opinion.

- Regularization methods
  - Adjust the bp error function to penalize the growth of unnecessary weights
  - Keep the weight vector small magnitude ➔add its value as a penalty to the problem of minimizing the loss.
    - Weight vector becomes too large, ➔ the learning algorithm prioritizes minimizing ***w*** over minimizing the training error.

$$l(w, b) + \frac{\lambda}{2} \|w\|^2$$

Generalization : Preventing over-fitting (over-training)

2. How to control number of effective weights?

- Manually or automatically select optimum number of hidden nodes and connections.
  - Not scalable, often needs expert opinion.

- Regularization methods
  - Adjust the bp error function to penalize the growth of unnecessary weights
  - Keep the weight vector small magnitude ➔add its value as a penalty to the problem of minimizing the loss.
    - Weight vector becomes too large, ➔ the learning algorithm prioritizes minimizing w over minimizing the training error.

$$l(w, b) + \frac{\lambda}{2} \|w\|^2$$

  - Squared Norm Regularization:

$$\|x\|^2 = \sum_{i=1}^{n} x_i^2$$

  - Gradient Descent update becomes :

$$w \leftarrow w(1-\alpha\lambda)w - \alpha \frac{\partial J}{\partial w}$$

Weights decay by an amount proportional to its magnitude

λ weight-cost parameter

another *Hyperparameter*

Training

1. Network Design (Architecture of NN networks.) #layers, #hidden nodes, activation functions, model ..
2. Initialize model parameters.
3. Choose Loss function
4. Training and Backpropagation : Mini batch, batch, or stochastic GD.
5. Monitor the loss function and error .

When no overfitting observed (epochs of training)

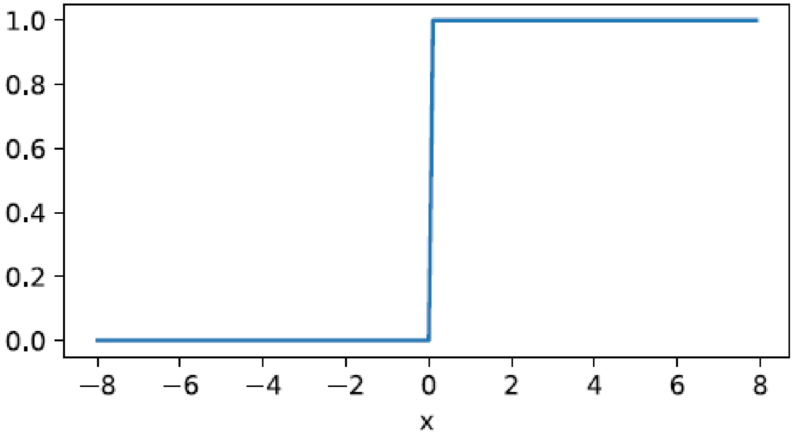
- Stop if the error fails to improve (has reached a minimum)
- Stop if the rate of improvement drops below a certain level
- Stop if the error reaches an acceptable level
- Stop when a certain number of epochs have passed

When overfitting observed: fine tune the NN network

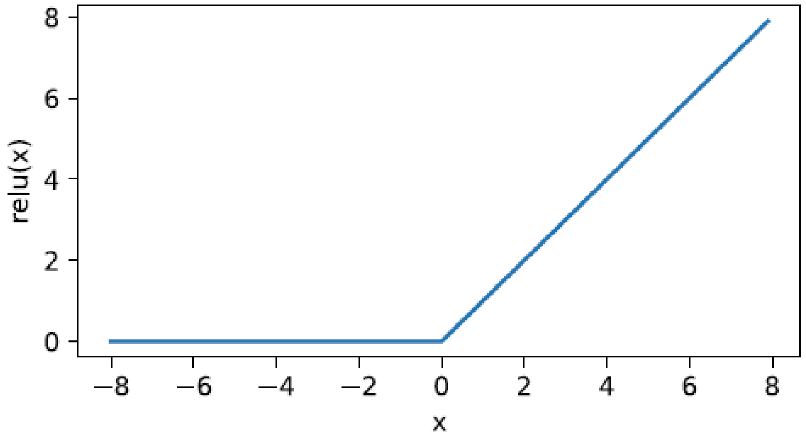
(initialize parameters, prune or regularize the weights, …)

Types of Activation functions

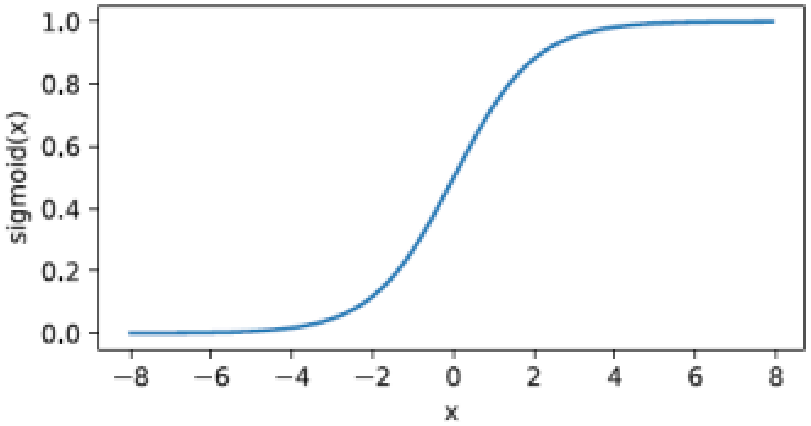
Activation functions



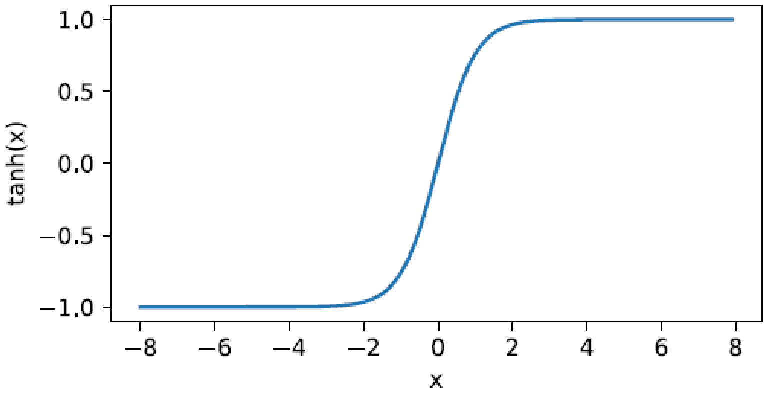
Threshold function (binary step function)



ReLu (Rectified Linear Unit)

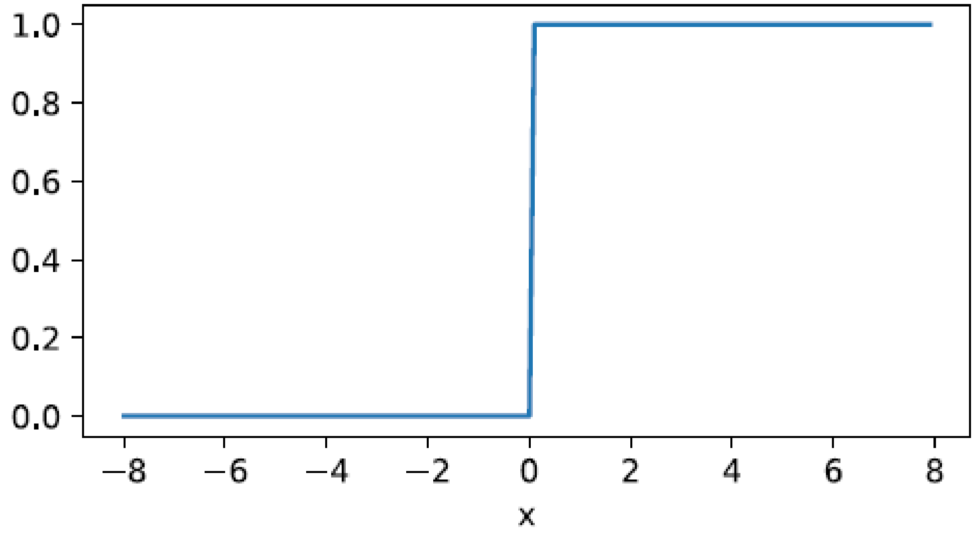


Sigmoid function



TanH / Hyperbolic Tangent

Activation Functions



$$\phi(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

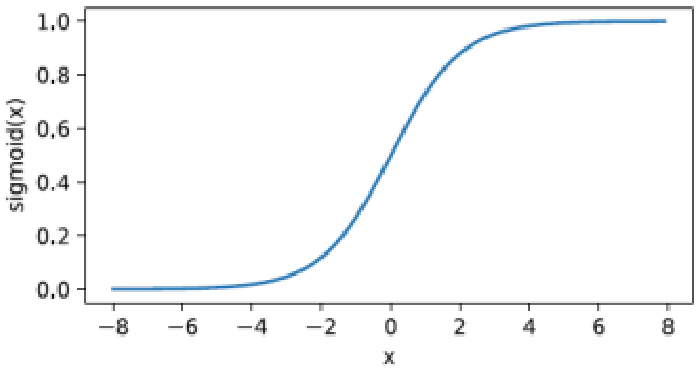
Threshold function (binary step function)

- If the input value is above or below a certain threshold,

the neuron is activated and sends the same signal to the next layer.

- Good for Binary outputs → 2 class classifications.
- Does NOT allow multi value outputs → does not support classification of input into multiple categories.

Activation Functions : Non-linear functions (why linear functions not preferred?



$$\phi(x) = \frac{1}{1+e^{-x}}$$

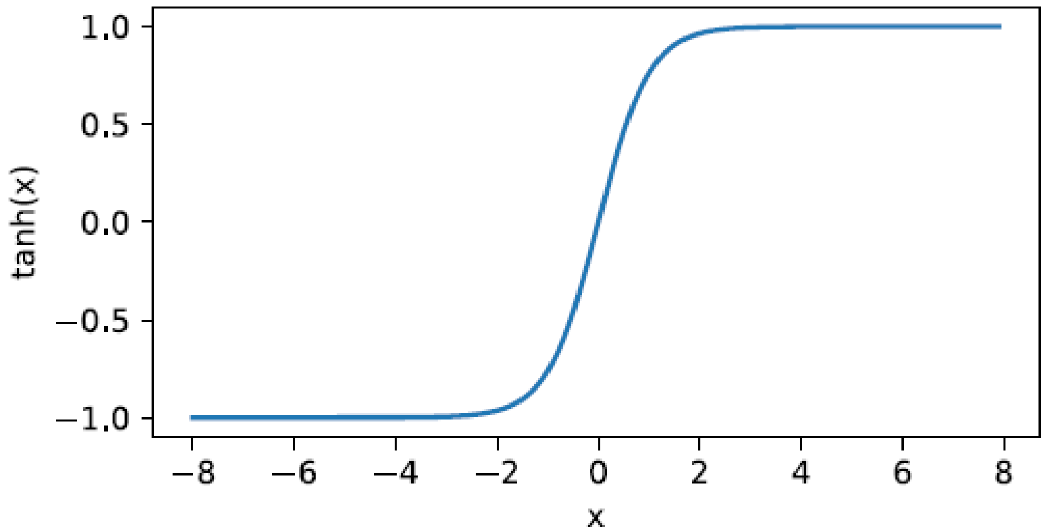
Sigmoid function

- Smooth gradient, preventing "jumps" in output values.
- Output values bound between 0 and 1, normalizing the output of each neuron.
- Clear predictions—For X above 2 or below -2, tends to bring the Y value (the prediction) to the edge of the curve, very close to 1 or 0. This enables clear predictions.
- The Sigmoid function used for **binary classification** in logistic regression model.
- While creating artificial neurons sigmoid function used as the **activation function**.

Disadvantages

- Vanishing gradient—for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem.
- This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.
- Computationally expensive
- Not Zero centered !!

Activation Functions



$$\phi(x) = \left\{ \frac{1 - e^{-2x}}{1 + e^{-2x}} \right.$$

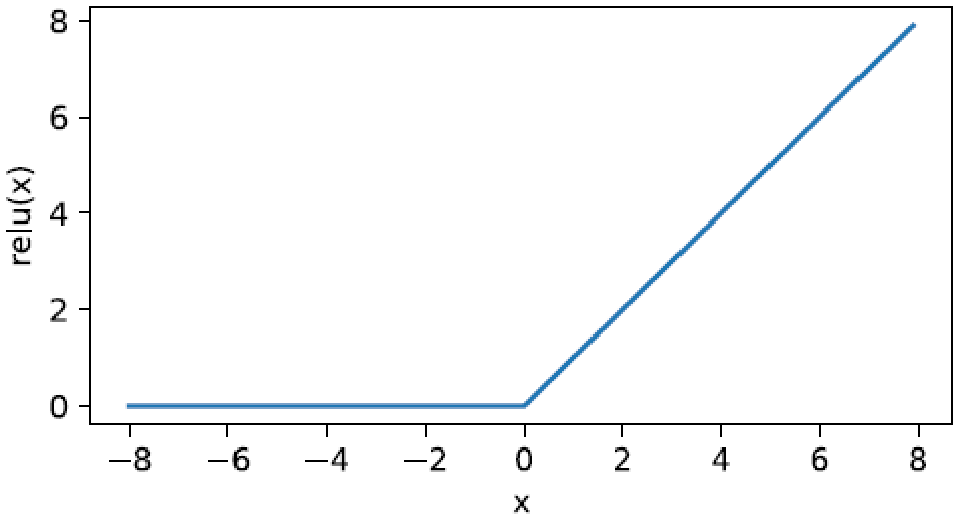
TanH / Hyperbolic Tangent

Zero centred ➔making it easier to model inputs that have strongly negative, neutral, and strongly positive values.

All advantages of Sigmoid function preserved.

Computationally expensive.

Activation Functions



$$\phi(x) = \max(x, 0)$$

ReLu (Rectified Linear Unit)

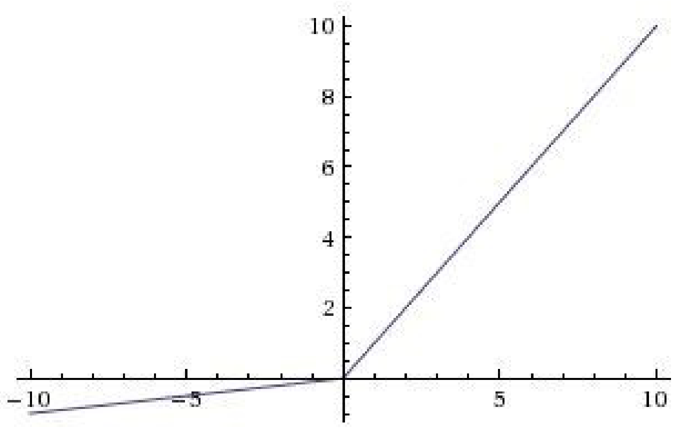
- **Computationally efficient**—allows the network to converge very quickly
- **Non-linear—**although it looks like a linear function, ReLU has a derivative function and allows for backpropagation.
- Avoids vanishing or exploding gradient problems unless…

Disadvantages:

The Dying ReLU problem—when inputs approach zero, or negative,

the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.

Activation Functions



$$\phi(x) = \max(x, 0)$$

Leaky ReLu

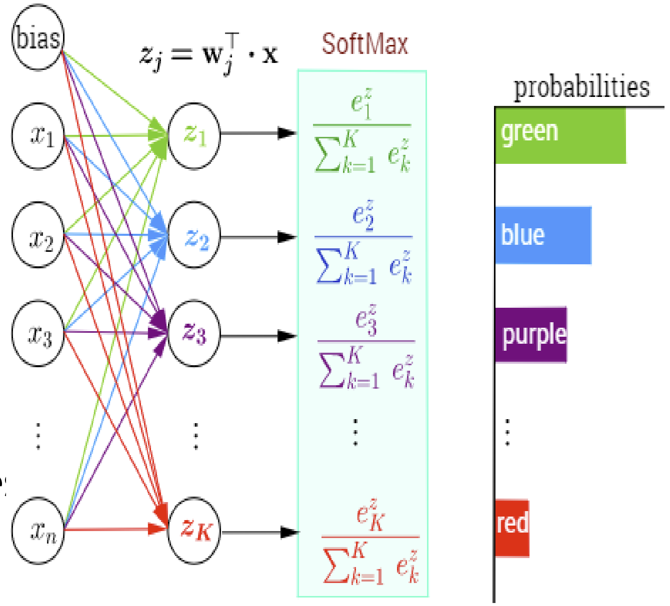
- **Computationally efficient**—allows the network to converge very quickly (faster than Sigmoid/tanh)
- **Does not Saturate/**
- **Does not "die"**

[Mass et al., 2013] [He et al., 2015]

Activation function

Softmax function

$$\phi(x_i) = \frac{\exp(x_i)}{\sum_{j-0}^{j=k} \exp(x_j)} \quad for\ i = 1, 2, 3 \ldots k$$



- Calculates the probabilities distribution of the event over 'n' different events.

- In general, calculates the probabilities of each target class over all possible target classe

- Later the calculated probabilities will be helpful for determining the target class for the given inputs.

- The range will **0 to 1**, and the sum of all the probabilities will be **equal to one.**

Remark: Useful for output neurons—typically Softmax is used only for the output layer, for neural networks that need to classify inputs into multiple categories.

- Very often used for multi-class classification.

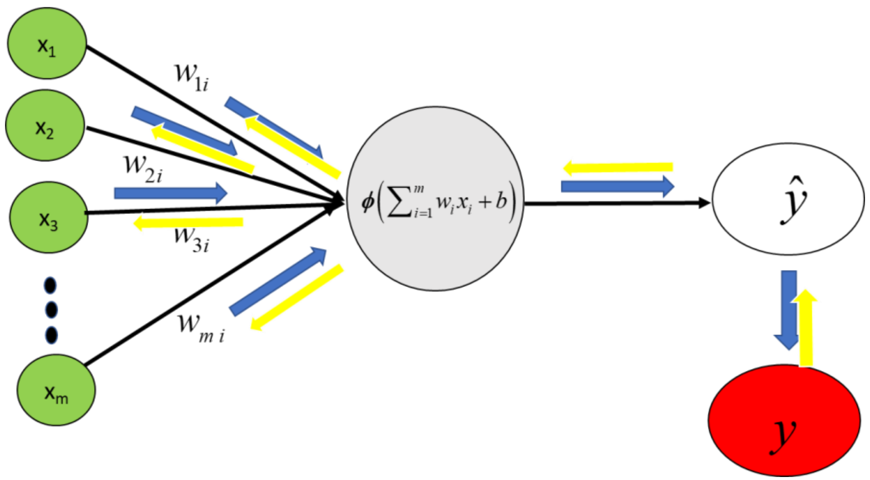
Loss functions

Common Loss functions

Regression

Mean Square Error (MSE) Loss: measured as the average of squared difference between predictions and actual observations.

$$L_{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2$$



Also known as: L2 loss, Quadratic loss, MSE loss, ..

Remarks:

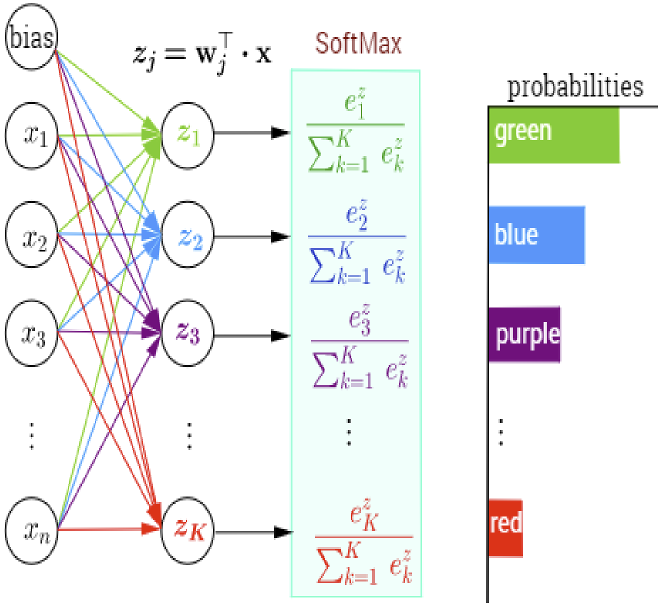
- Predicted values that are far from actual values are penalized heavily.
- Squaring : positivity, quadratic function→ nice properties helpful in finding gradients.

Common Loss functions

Classification (recall: binary classification and multi class classification

Softmax function)

- Often, for classification: outputs are probabilities of belonging to each class.
- Thus, loss must be calculated based on assessment of probabilities.



$$\phi(x_i) = \frac{\exp(x_i)}{\sum_{j-0}^{j=k} \exp(x_j)} \quad for\ i = 1, 2, 3 \ldots k$$

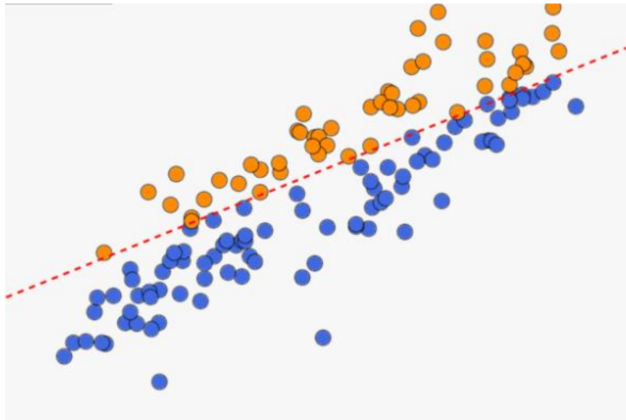
Common Loss functions

Classification Loss (recall: binary classification and multi class classification

Softmax function)

Cross Entropy Loss (log loss, logistic loss, logarithmic loss, negative log loss..)

(Binary Class , or 2 classes)



$$L_{CE} = -\left(y\log(\hat{p}) + (1-y)\log(1-\hat{p}) \right)$$

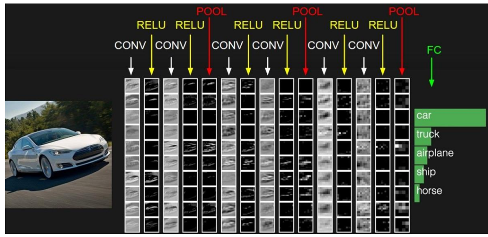
- Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.

- Cross-entropy loss increases as the predicted probability diverges from the actual label.

- Notice that when actual label is 1 (y = 1), second half of function disappears whereas in case actual label is 0 (y = 0) first half is dropped off.

- A perfect model would have a log loss of 0.

Common Loss functions



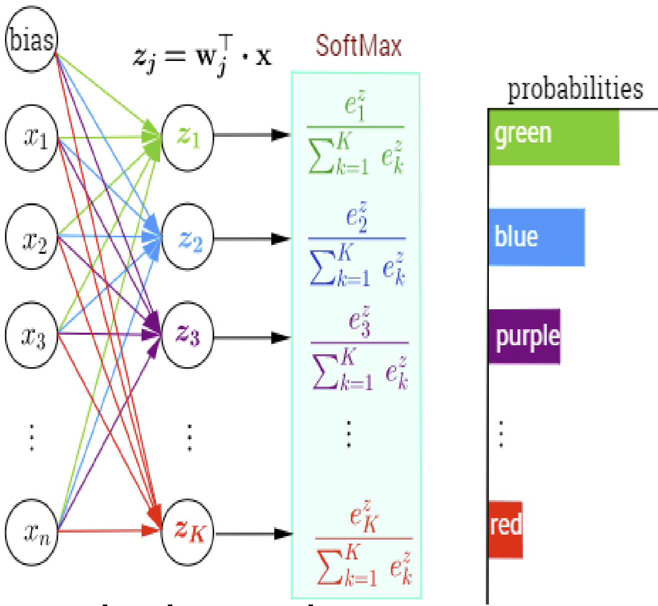
Classification Loss (multi class classification, Softmax function)

Cross Entropy Loss

(Multi Class)

$$L_{CE} = -\sum_{c=1}^{M} y_{i,c} \log(p_{i,c})$$

M : Number of classes
$y_{i,c}$ *:* true probability of belonging to that class
$p_{i,c}$: predicted probability of belonging to that class.



- Cross-entropy can be calculated for multiple-class classification.
- The classes have been **one hot encoded**, meaning that there is a binary feature for each class value.
- The predictions must have predicted probabilities for each of the classes (Example: Softmax).
- The cross-entropy is **then summed across each binary feature** and averaged across all examples in the dataset.

$$\phi(x_i) = \frac{\exp(x_i)}{\sum_{j-0}^{j=k} \exp(x_j)} \quad for\ i = 1, 2, 3....k$$

Suggestion: [Read this thread of discussion on forum on using Cross entropy in practice](#).

Loss functions: Best practices

Regression Problem

- A problem where you predict a real-value quantity.
- **Output Layer Configuration**: One node with a linear activation unit.
- **Loss Function**: Mean Squared Error (MSE).

Binary Classification Problem

- A problem where you classify an example as belonging to one of two classes.
- The problem is framed as predicting the likelihood of an example belonging to class one, e.g. the class that you assign the integer value 1, whereas the other class is assigned the value 0.
- **Output Layer Configuration**: One node with a sigmoid activation unit.
- **Loss Function**: Cross-Entropy

Multi-Class Classification Problem

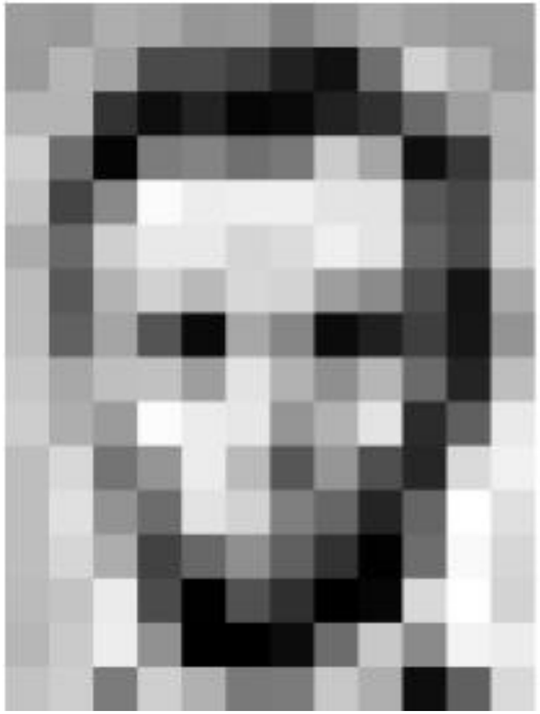
- A problem where you classify an example as belonging to one of more than two classes.
- The problem is framed as predicting the likelihood of an example belonging to each class.
- **Output Layer Configuration**: One node for each class using the softmax activation function.
- **Loss Function**: Cross-Entropy.

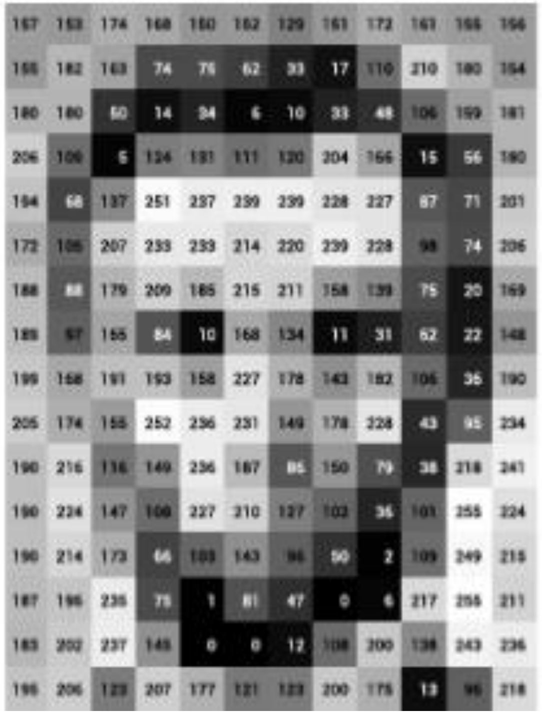
Summary

- Simple NN functioning, analogy with linear regressions
- Feed foreword Deep NN functioning
- Weight updates through backprop and gradient descent (batch, mini batch and stochastic GD)
- Generalization :Training /validation/test set
- Generalization and Training issues: overfitting, underfitting, finding the right tradeoff.
- Weights initializations: Exploding and Vanishing gradients, Xavier initilisations.
- Note on Activation functions.

Convolutional Neural Networks

Images are just numbers for computer!





157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

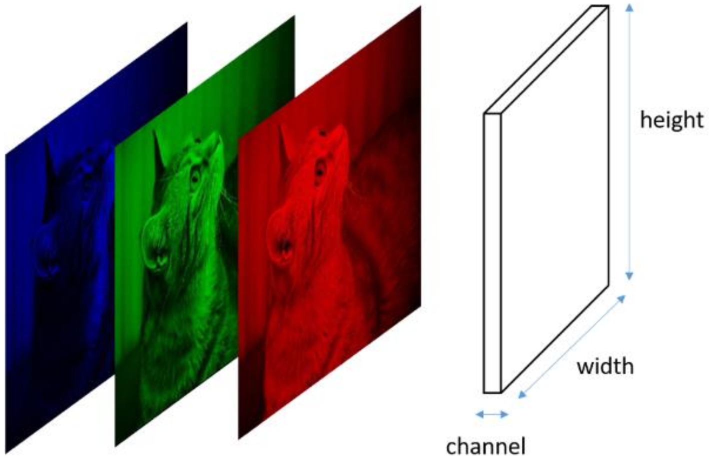
Source: OpenFrames

Images are matrix of numbers.

Gray Scale Images ➔ One channel Grey Scale ➔ 2D matrix of numbers (pixel values).
Each pixel =[0,255],
No of pixels proportional to image size → No of rows and columns.

Color images

Source: Broher



Source: Medium

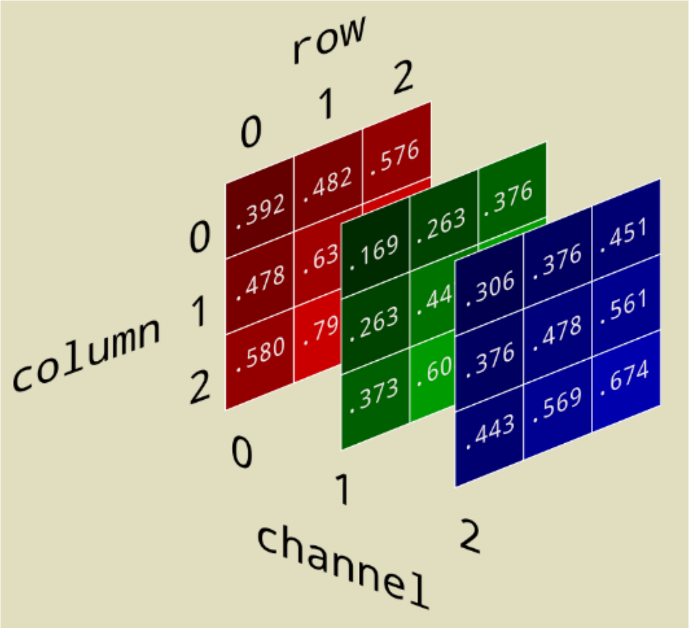
Colored Images: 3 channels of colors: 3D array

RGB channels ➔ 3D Arrays

Red: 2-D matrix

Green: 2-D matrix

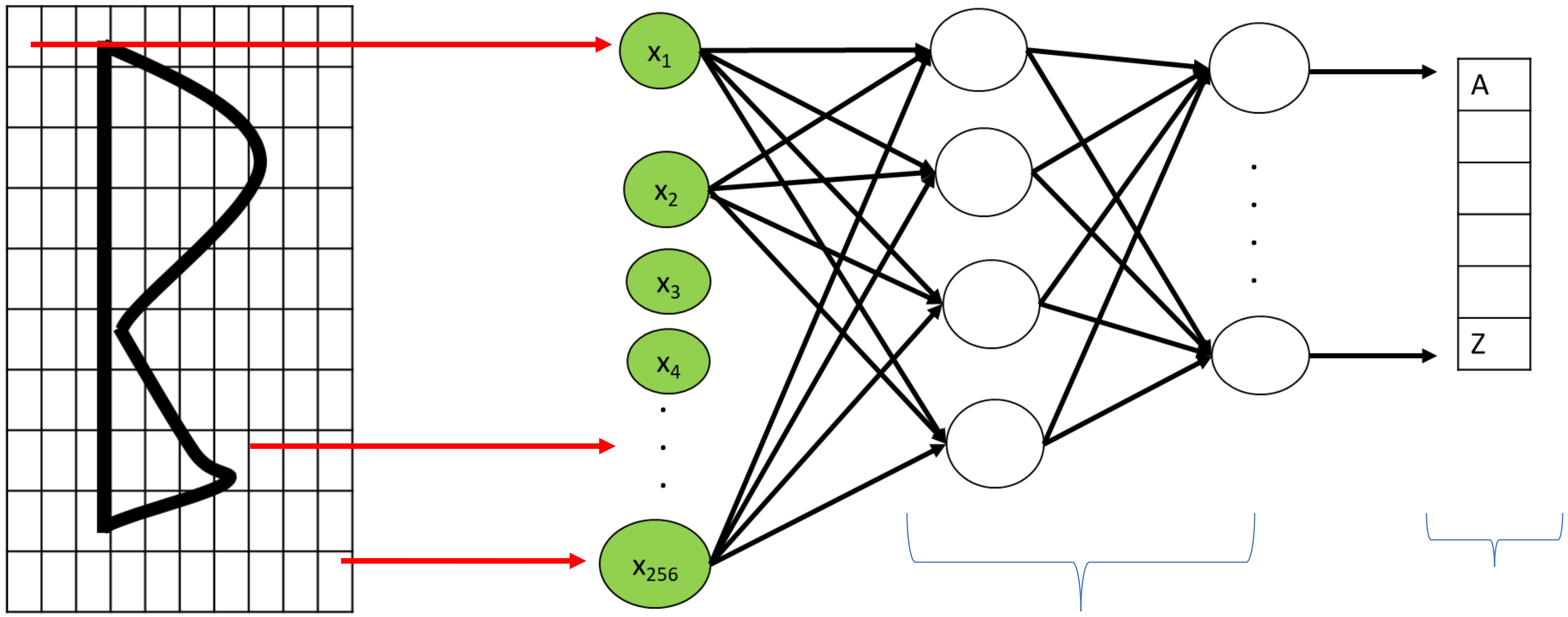
Blue: 2-D matrix



Drawbacks

Drawbacks

Explosion of training parameters



16 X 16 Image GreyScale

Deep Hidden layers

Multi class output

Colored Images: 3 channels of colors: 3D array

RGB channels ➔ 3D Arrays

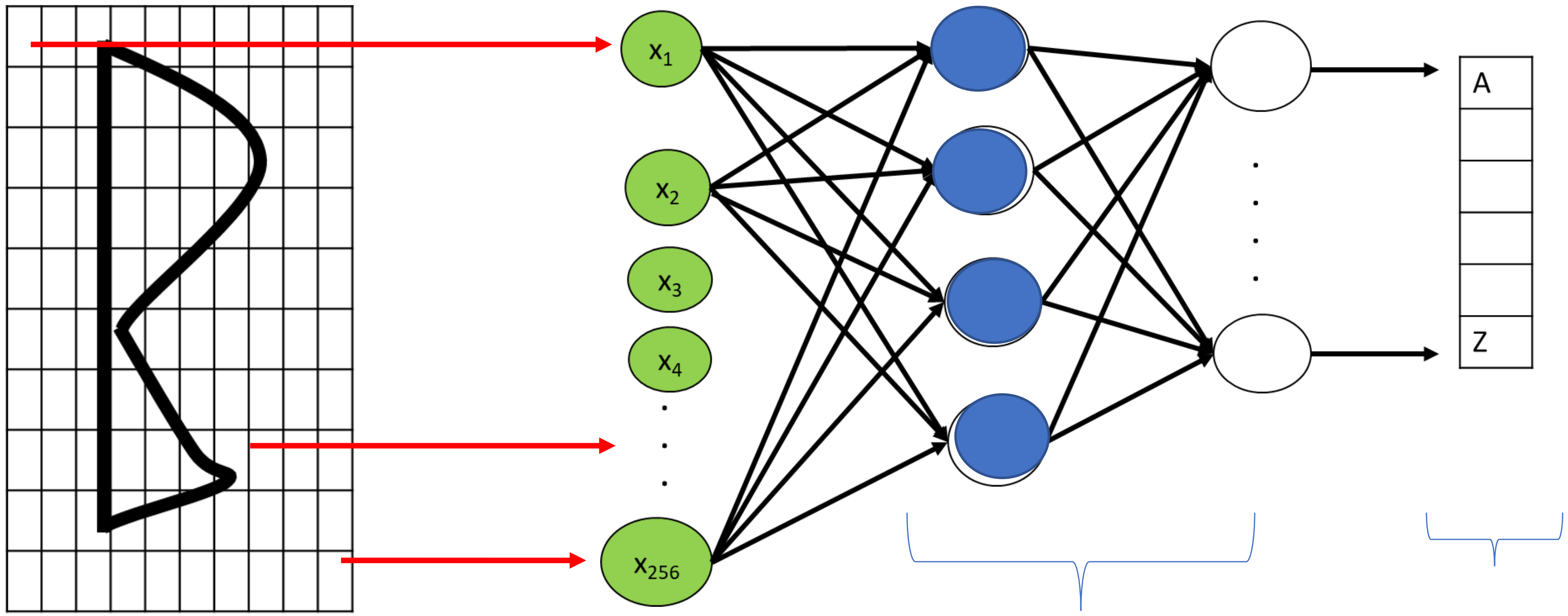
Red: 2-D matrix

Green: 2-D matrix

Blue: 2-D matrix

Drawbacks

Explosion of training parameters



16 X 16 Image GreyScale

Deep Hidden layers

Multi class output

Colored Images: 3 channels of colors: 3D array

RGB channels → 3D Arrays

Red: 2-D matrix

Green: 2-D matrix

Blue: 2-D matrix

Simple calculation for 1 layer , 100 hidden units:

256 inputs → 256 weights

100 hidden units → 256 x 100=25600 input weights

Bias → 100 bias

26 Outputs (A-Z)→ 26 X 100 output weights

Biases → 26

Total: 25600 + 100+ 2600 + 26 = 28326

That is just with one layer !!

Drawbacks: Trainable parameter explosion

- Most images → high resolution (1MB or more) → several thousands of pixels ➔ several thousands inputs.

- Several hundreds of hidden layers with several hundreds of units.

- Total parameters to train → Extremely large → Computation intractable !!

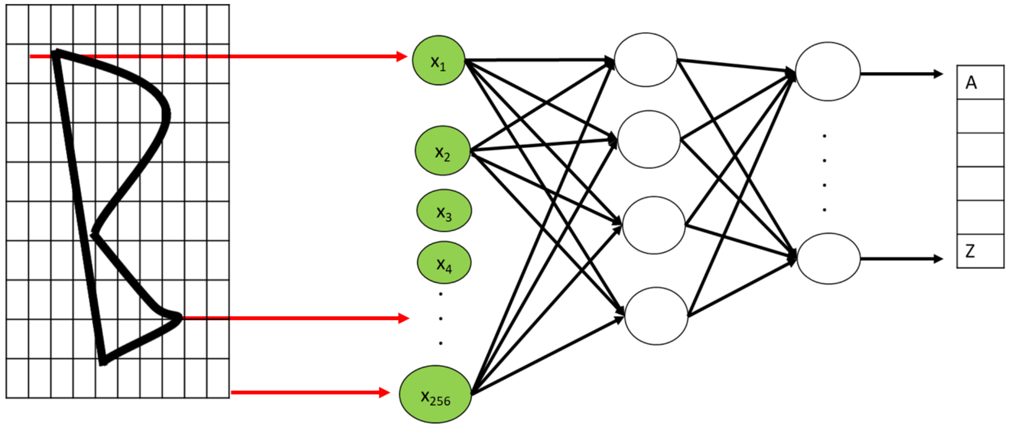
- Strong regularization needed → difficult and little reproducibility.

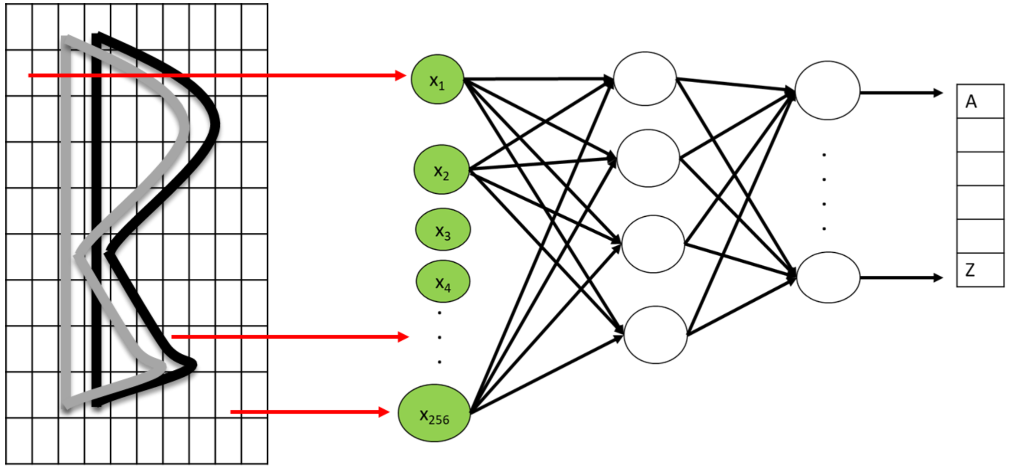
Drawbacks: Variance to distortions

- The orientation / location of object within an image should have little influence over it getting detected.

  This is not true with previous NNs (MLPs, ANNs).

- Variance to scaling, shifting and other distortions, influence of surroundings (global context).
- The topology of the data is ignored.
- Inherent distributions are not learnt well.





- Must avoid parameter explosion in face of large inputs.
- Identification of object should be invariant to scaling, shifting and different orientation.

- The object should be identifiable in any location / orientation ➔ placement of object in an image should not influence the outcome, only local information about the object should be sufficient.

Convolutional neural networks

JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

Motivation

Convolutional layer: Motivation

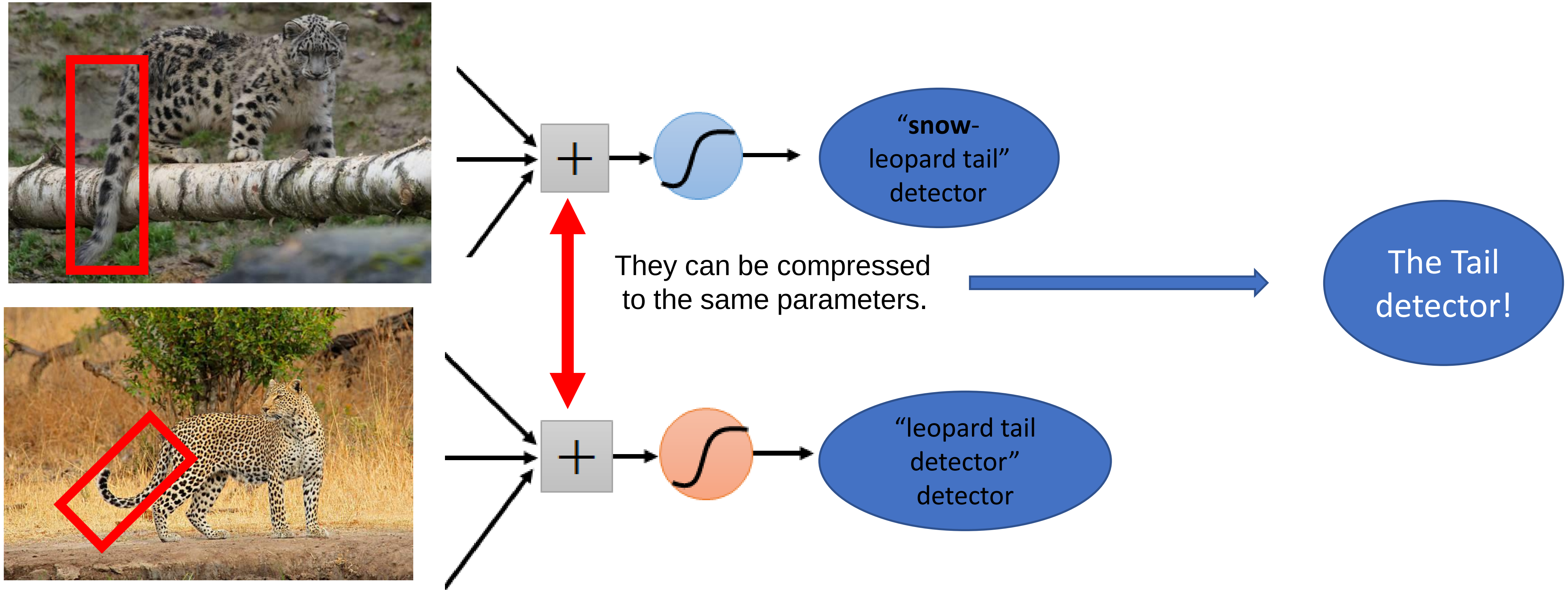
Proposed by Yann LeCun and Yoshua Bengio in 1995.

- Convolutional Neural Networks are a special kind of multi-layer neural networks.
- Inspired by neuro-biology: brain's mechanism of understanding different attributes of an object
  - Attributes : shape, size, orientation and color.

Convolutional layer: Motivation

Proposed by Yann LeCun and Yoshua Bengio in 1995.

- Convolutional Neural Networks are a special kind of multi-layer neural networks.
- Inspired by neuro-biology: brain's mechanism of understanding different attributes of an object
  - Attributes : shape, size, orientation and color.



Convolutional layer: Motivation

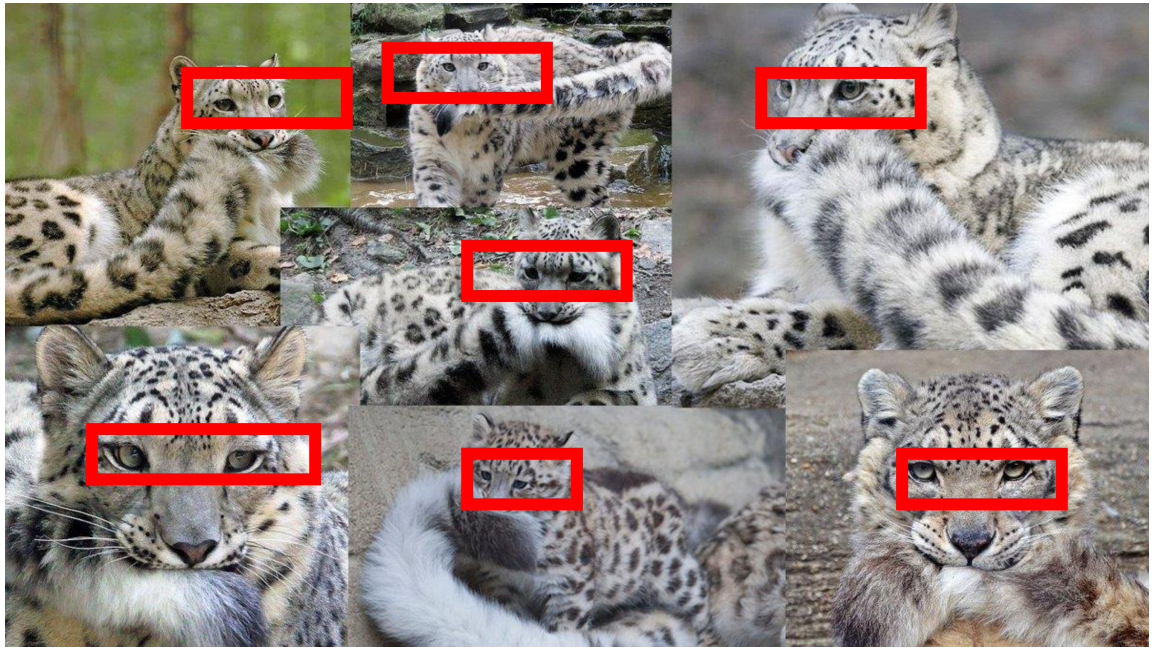
Proposed by Yann LeCun and Yoshua Bengio in 1995.

- Convolutional Neural Networks are a special kind of multi-layer neural networks.
- Inspired by neuro-biology: brain's mechanism of understanding different attributes of an object
  - Attributes : shape, size, orientation and color.

Intuition:

- Understanding the inherent data distribution.
- Using local information to extract topological properties from image.
- Implicitly extract relevant features.

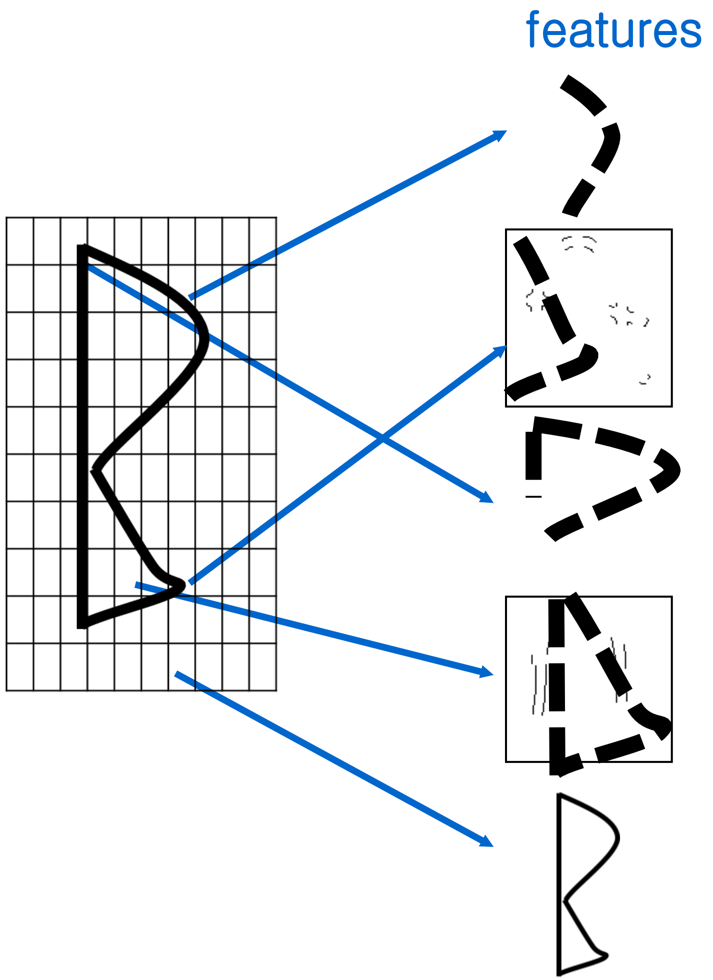
Understanding the new data using learnt attributes.



Convolutional layer: Motivation

Proposed by Yann LeCun and Yoshua Bengio in 1995.

- Convolutional Neural Networks are a special kind of multi-layer neural networks.
- Inspired by neuro-biology: brain's mechanism of understanding different attributes of an object
  - Attributes : shape, size, orientation and color.



Intuition:

- Understanding the inherent data distribution.
- Using local information to extract topological properties from image.
- Implicitly extract relevant features.

Understanding the new data using learnt attributes.

Example:

A door is always rectangular in shape,

A ship has a characteristic shape,

a car of any brand shall have a typical shape....

Intuition

Intuition

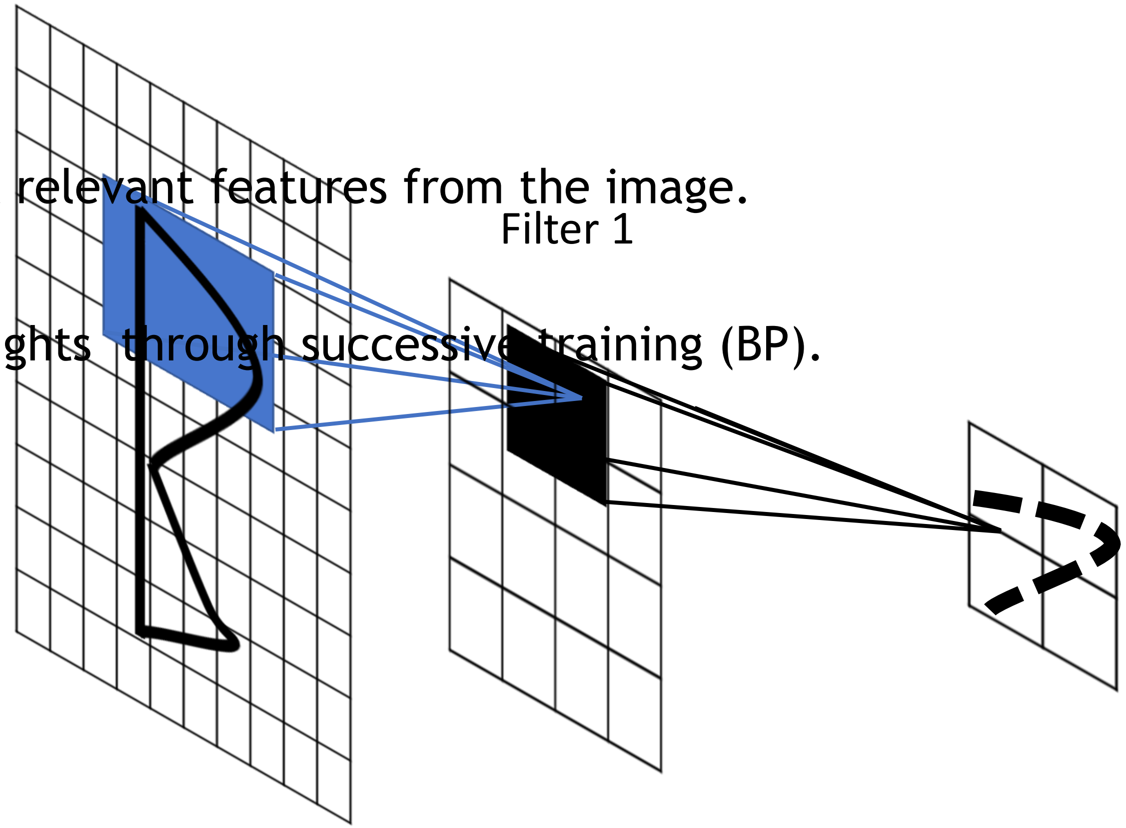
Intuition:

Use an image kernel to extract relevant features from the image.

Image kernel = image matrix.

Learn an appropriate filter weights through successive training (BP).

Shape 1 / feature 1



Intuition

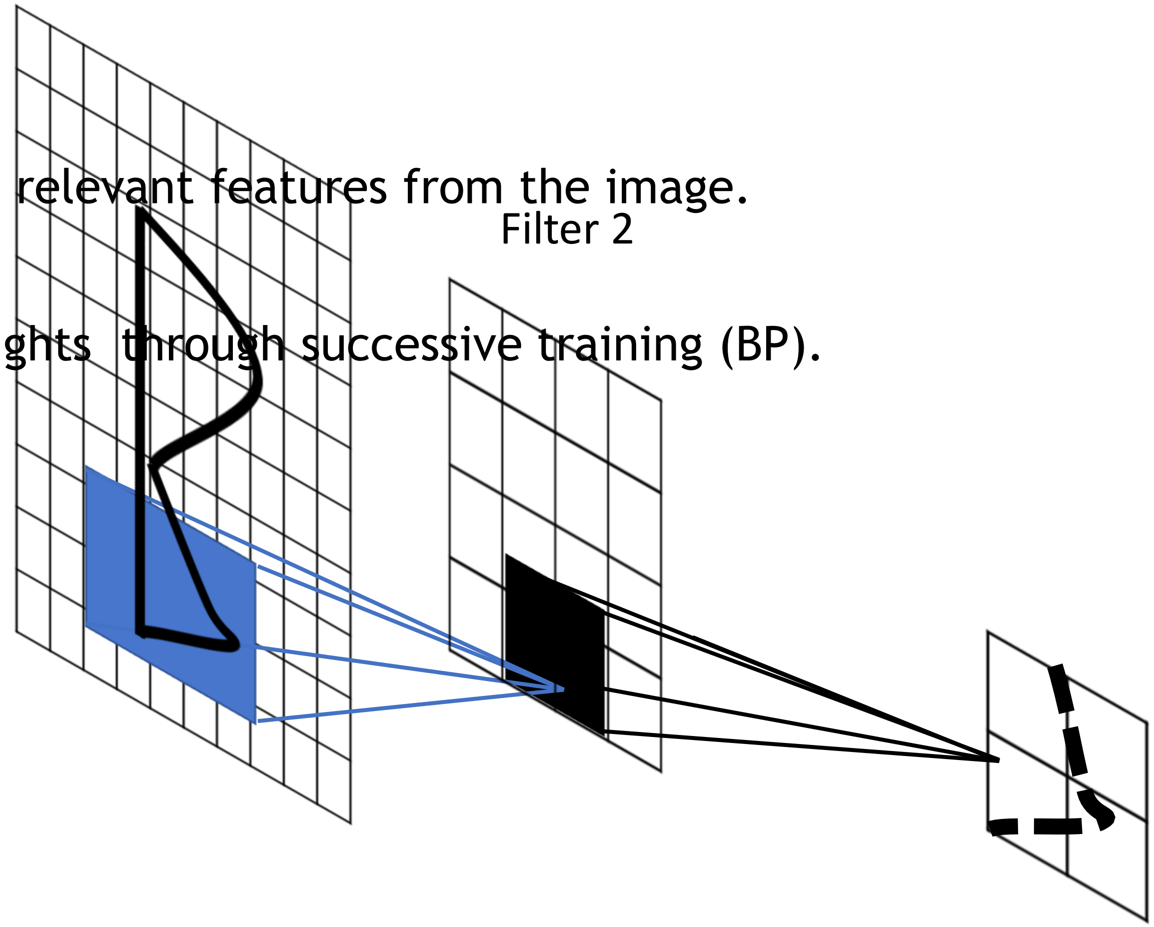
Intuition:

Use an image kernel to extract relevant features from the image.

Image kernel = image matrix.

Learn an appropriate filter weights through successive training (BP).

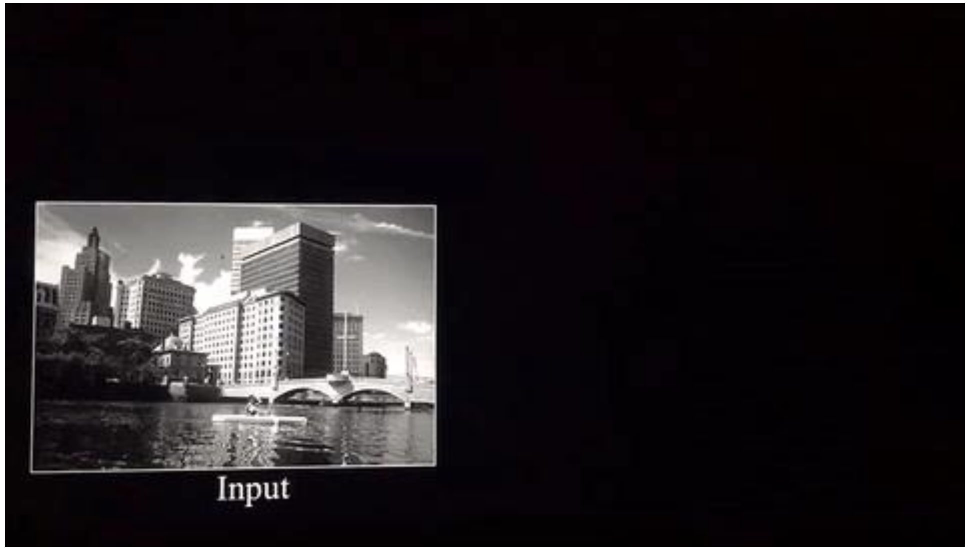
Shape 2 / feature 2



Intuition

Source :Blog AndrewSzot

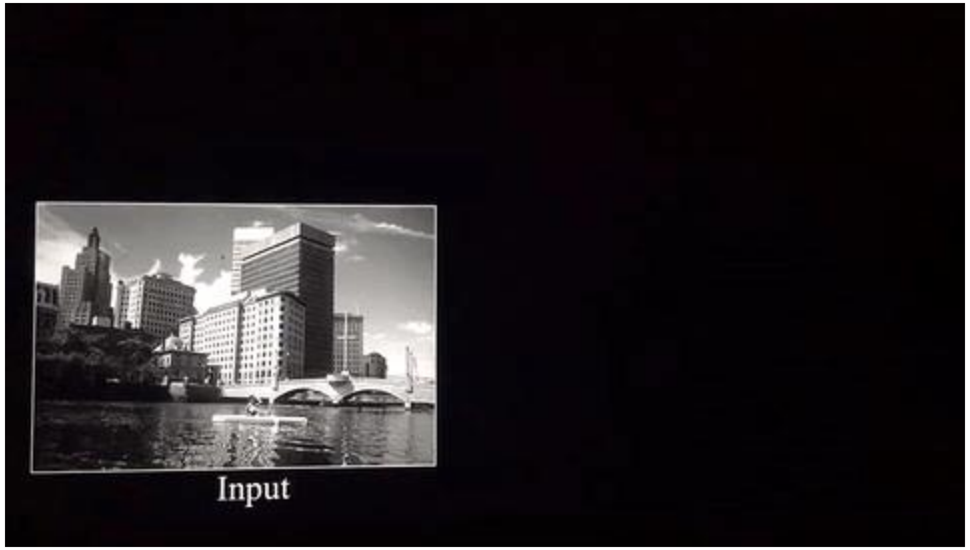
Intuition



Source :Blog AndrewSzot

Intuition

Source :Blog AndrewSzot



Intuition:

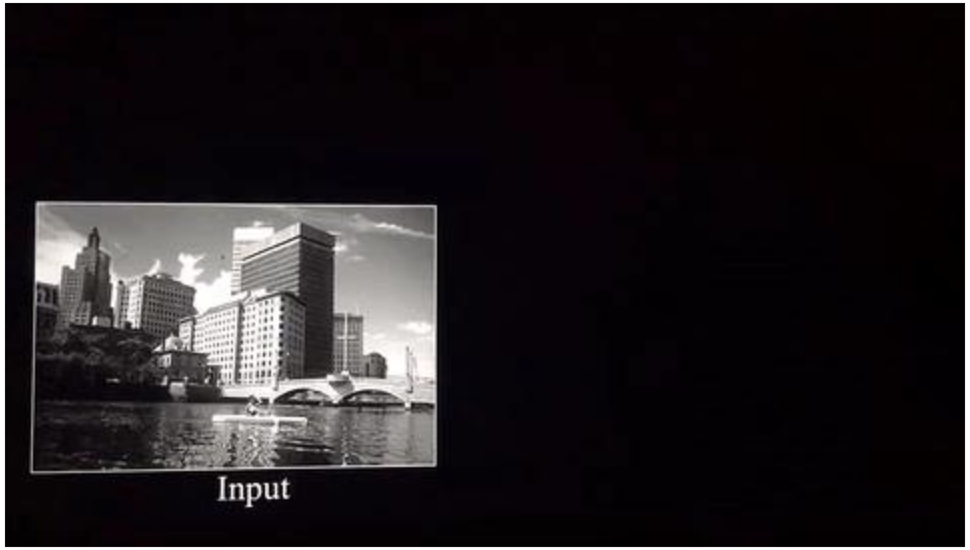
Edge Detection: image kernel for edge detection.

Multiple image kernels to extract different features.

Why not multiple kernels to extract set of features expected from object / required for the objective.

Intuition

Source :Blog AndrewSzot



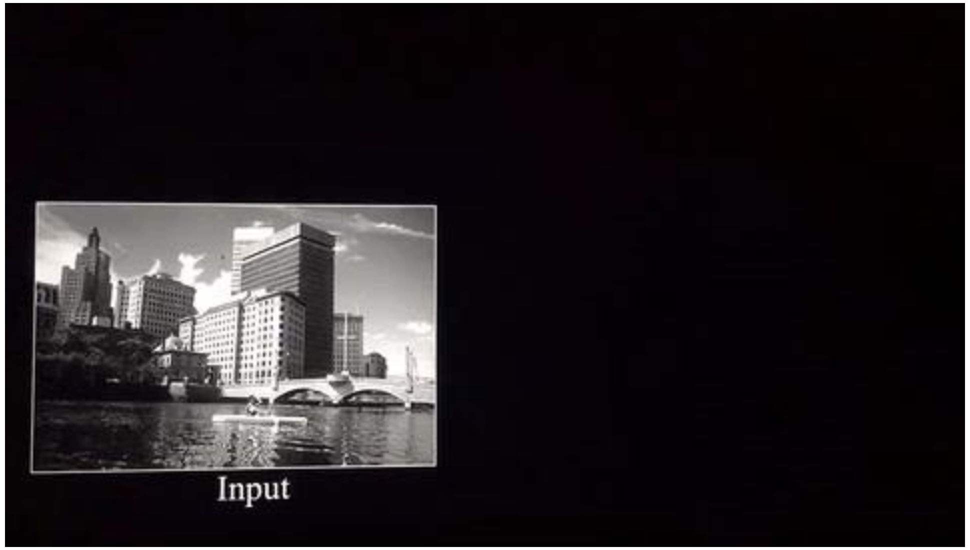
Intuition:

How to construct these filters? Edge detection is straight foreword. Not obvious in general.

Essence of CNN :

- learn the values (weights) of these filters (BP).
- stack multiple layers of feature detectors (kernels) on top of each other for abstracted levels of feature detection.

Intuition



Source :Blog AndrewSzot

Intuition:

How to construct these filters? Edge detection is straight foreword. Not obvious in general.

Essence of CNN :

- learn the values (weights) of these filters (BP).
- stack multiple layers of feature detectors (kernels) on top of each other for abstracted levels of feature detection.
- extract relevant features: Convolution operation . What and How?

Convolution Operator

Convolution Operator

$$x(t) * y(t) = \int_{-\infty}^{+\infty} x(\tau)y(t-\tau)d\tau = \int_{-\infty}^{\infty} y(\tau)x(t-\tau)d\tau$$

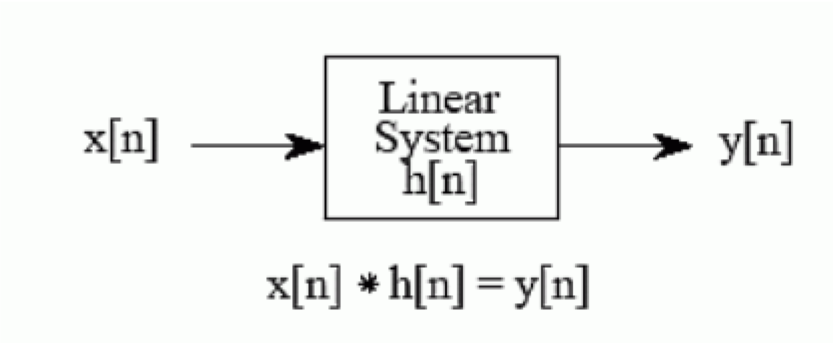
Reminders:

- origins in Signal Processing.
- convolution of two signals produces a third signal
- In signal processing, input signal convolution with impulse response of the system → output response

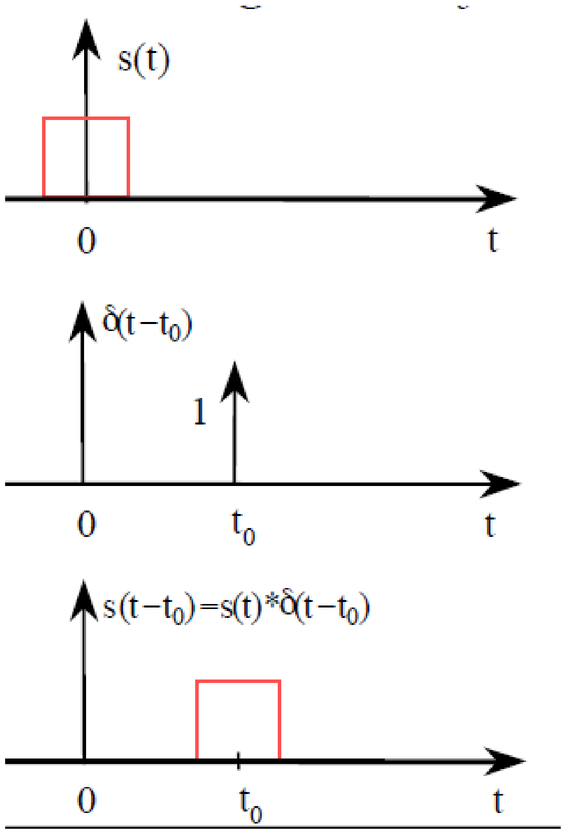
Convolution Operator

$$x(t) * y(t) = \int_{-\infty}^{+\infty} x(\tau)y(t-\tau)d\tau = \int_{-\infty}^{\infty} y(\tau)x(t-\tau)d\tau$$

$$s(t) * \delta(t-t_0) = s(t-t_0)$$



Source: DSP Guide book



Source: DSP Course by Prof. Garnier, Polytech Nancy

Reminders:

- origins in Signal Processing.
- convolution of two signals produces a third signal
- input signal * impulse response of the system ➔ output response.

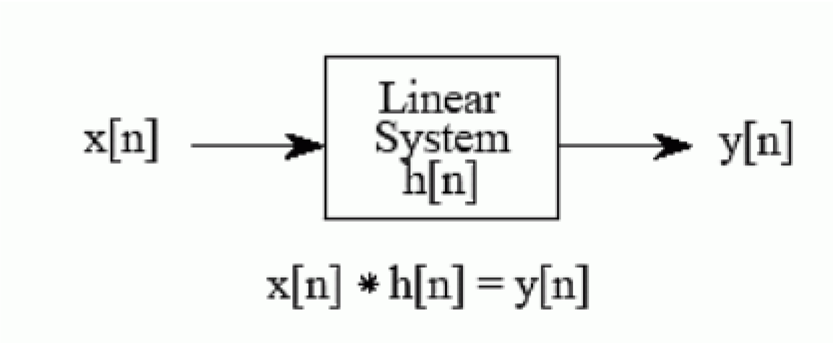
Convolution of a signal by Dirac impulse positioned at t_0 ➔ signal shift to t_0.

Convolution Operator

$$x(t) * y(t) = \int_{-\infty}^{+\infty} x(\tau)y(t-\tau)d\tau = \int_{-\infty}^{\infty} y(\tau)x(t-\tau)d\tau$$

$$s(t) * \delta(t-t_0) = s(t-t_0)$$

$$s(t) * \delta_{Te}(t) = \sum_{k=-\infty}^{+\infty} s(t-kTe)$$

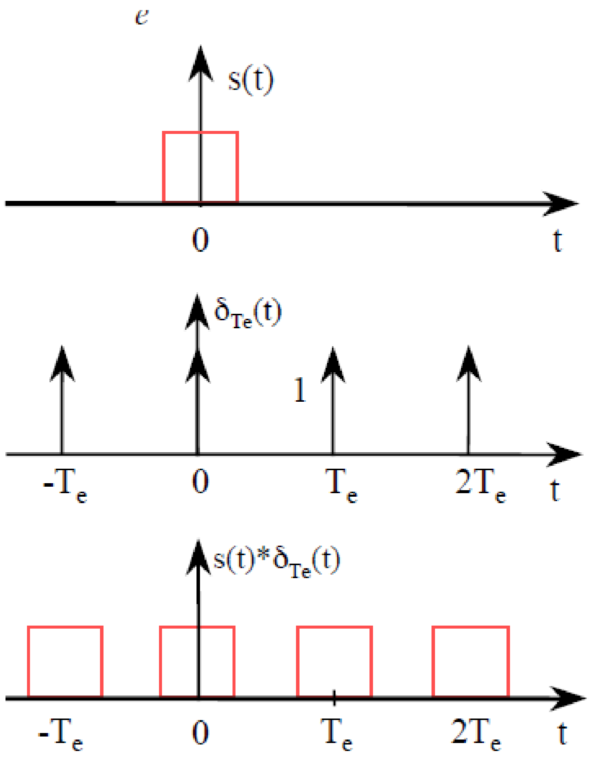


Source: DSP Guide book

Reminders:

- origins in Signal Processing.
- convolution of two signals produces a third signal
- input signal * impulse response of the system ➔ output response.

Convolution of a signal by Dirac train → periodic signal with period T_e



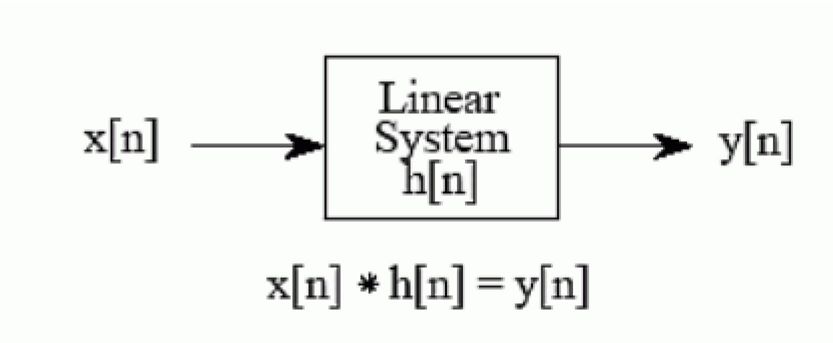
Source: DSP Course by Prof. Garnier, Polytech Nancy

Convolution Operator

$$x(t) * y(t) = \int_{-\infty}^{+\infty} x(\tau)y(t-\tau)d\tau = \int_{-\infty}^{\infty} y(\tau)x(t-\tau)d\tau$$

$$s(t) * \delta(t-t_0) = s(t-t_0)$$

$$s(t) * \delta_{Te}(t) = \sum_{k=-\infty}^{+\infty} s(t-kTe)$$



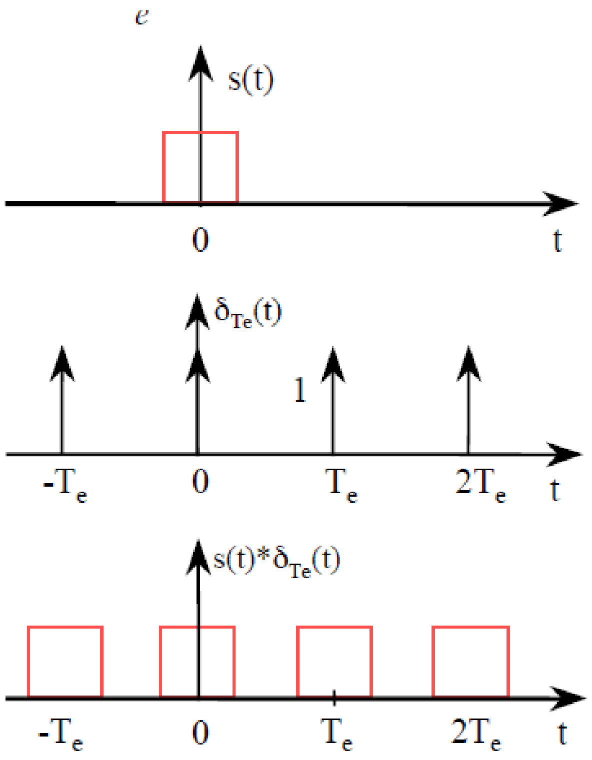
Source: DSP Guide book

Reminders:

- origins in Signal Processing.
- convolution of two signals produces a third signal
- input signal * impulse response of the system ➔ output response.

Convolution of a signal by Dirac train → periodic signal with period T_e

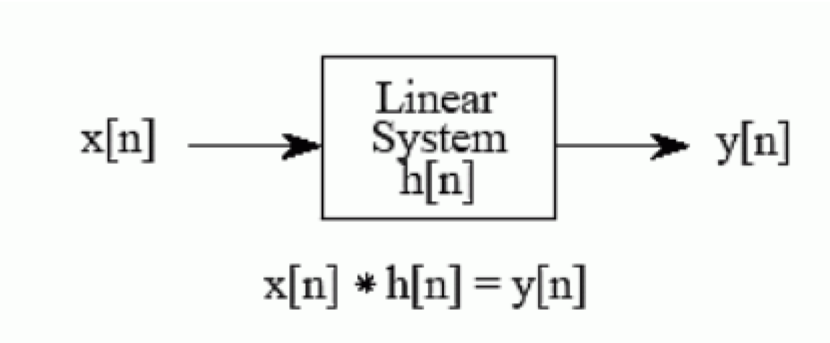
- Convolution operation constructs a system response signal.
- Convolution operation fundamental in assessing the similarity between two signals.



Source: DSP Course by Prof. Garnier, Polytech Nancy

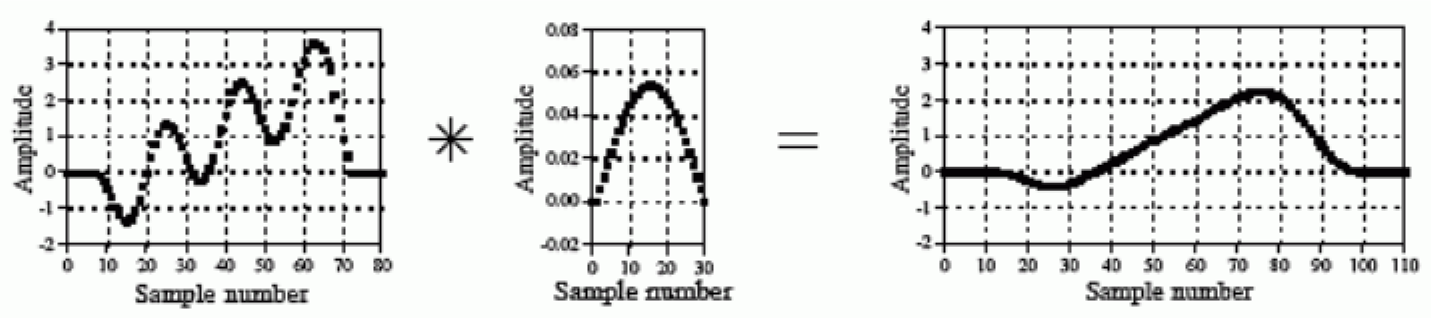
Convolution Operator

$$x(t) * y(t) = \int_{-\infty}^{+\infty} x(\tau)y(t-\tau)d\tau = \int_{-\infty}^{\infty} y(\tau)x(t-\tau)d\tau$$



Source: DSP Guide book

$$S(t) = (x * w) \times (t) = \sum_{a=-\infty}^{\infty} w(a)x(t-a)$$



Reminders:

Low pass filtering:

Input: three cycles of sine wave plus a slow increasing ramp.

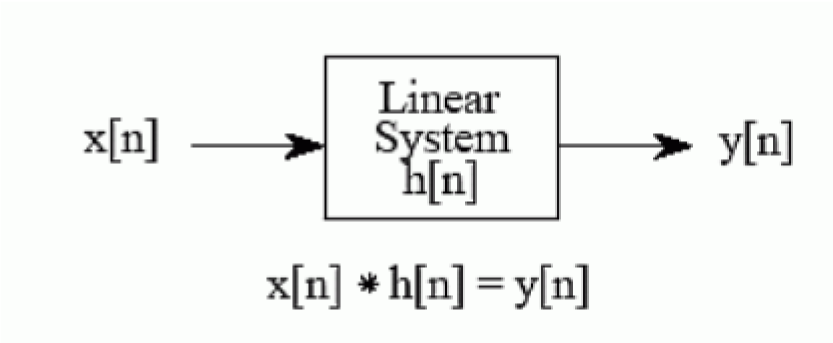
Low pass filter impulse response (or Convolution kernel / filter kernel)

Output = slow component ramp.

Convolution operation → extracts the weighted feature.

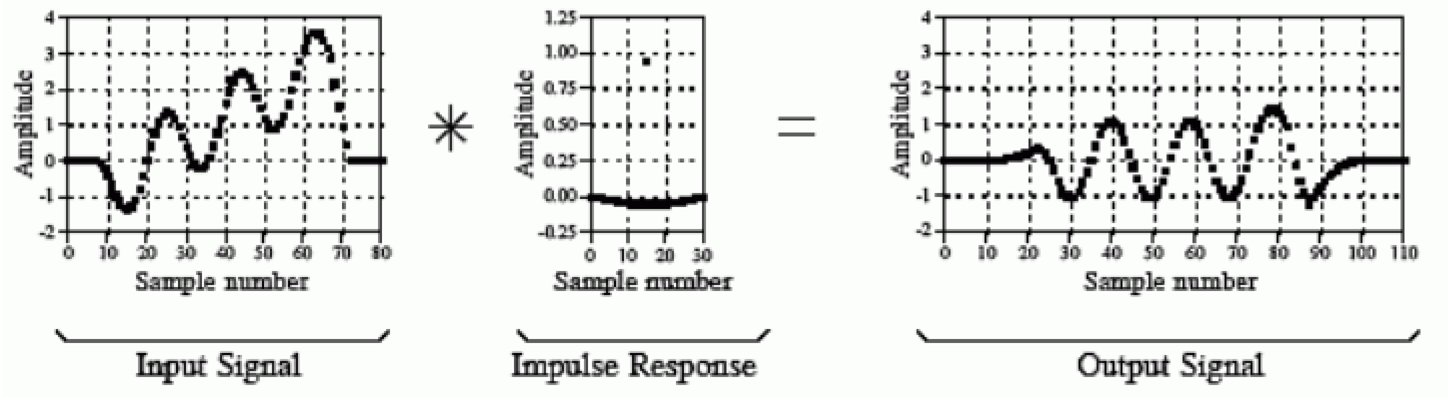
Convolution Operator

$$x(t) * y(t) = \int_{-\infty}^{+\infty} x(\tau)y(t-\tau)d\tau = \int_{-\infty}^{\infty} y(\tau)x(t-\tau)d\tau$$



Source: [DSP Guide book]

$$S(t) = (x * w) \times (t) = \sum_{a=-\infty}^{\infty} w(a)x(t-a)$$



Reminders:

High pass filtering:

Input: three cycles of sine wave plus a slow increasing ramp.

High pass filter impulse response (or Convolution kernel / filter kernel)

Output = Fast component ramp.

Convolution operation → extracts the weighted feature.

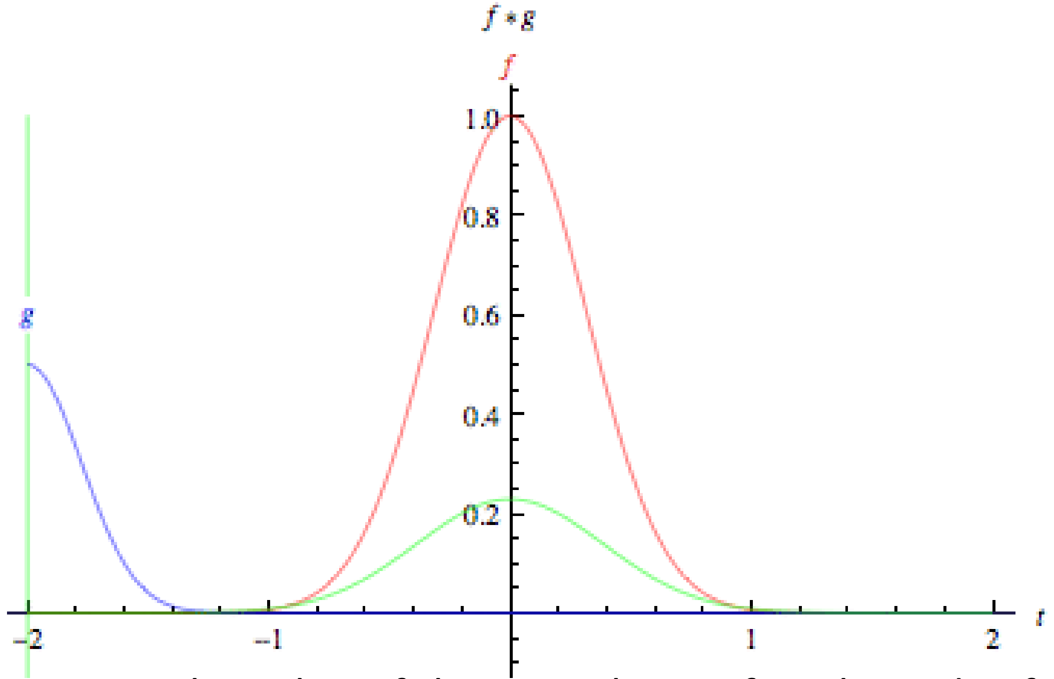
Convolution Operator

$$x(t) * y(t) = \int_{-\infty}^{+\infty} x(\tau)y(t-\tau)d\tau = \int_{-\infty}^{\infty} y(\tau)x(t-\tau)d\tau$$

$$s(t) = (x * w) \times (t) = \int_{-\infty}^{\infty} w(a)x(t-a)da$$

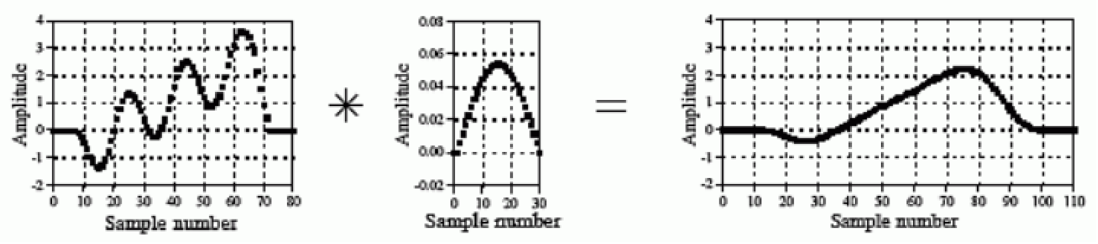
$$S(t) = (x * w) \times (t) = \sum_{a=-\infty}^{\infty} w(a)x(t-a)$$

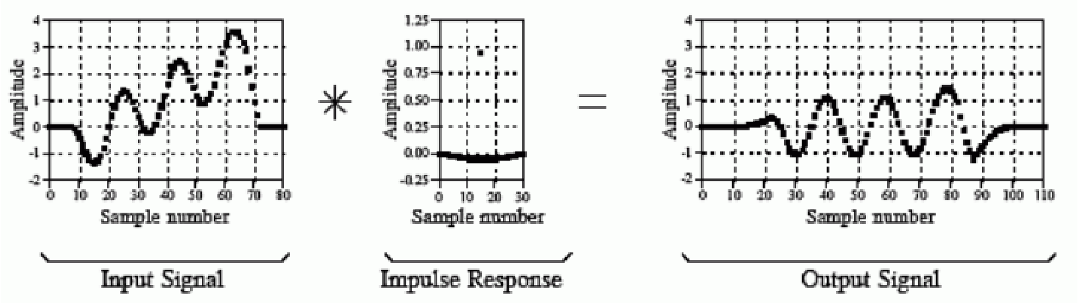
Thus, convolution measures the overlap between any two functions.



Green curve is the value of the convolution $f*g$, the red is f, the blue g and the shaded area is the product f(a)g(t−a) where t is the x-axis.

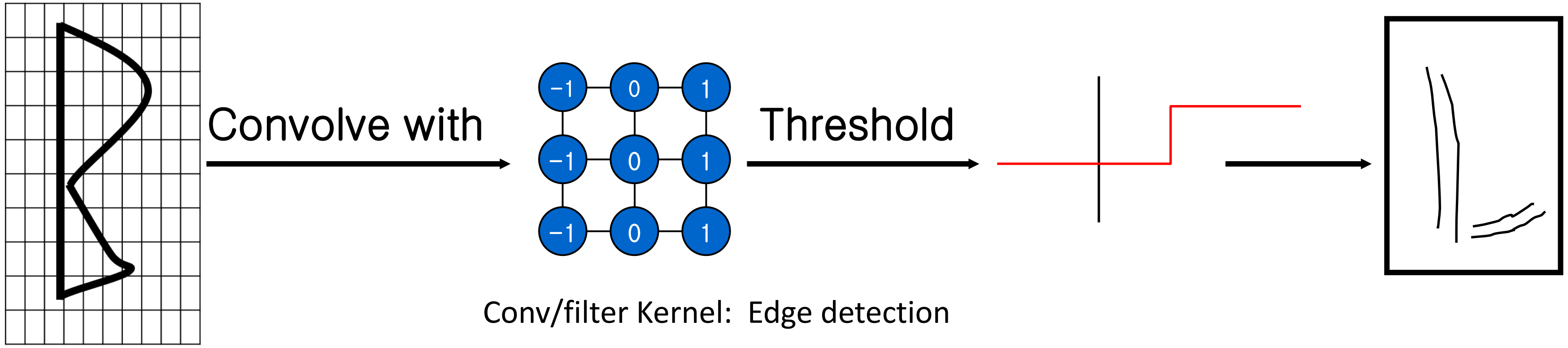
Source :Blog AndrewSzot





Convolution

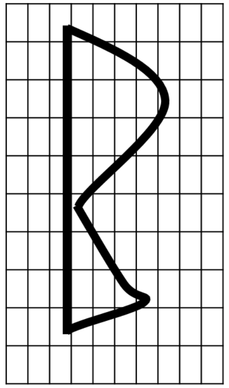
Convolution: CNN context.



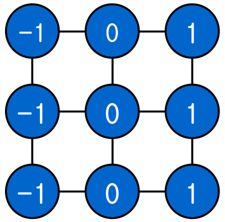
Conv/filter Kernel: Edge detection

Back to CNNs:

Convolution Operator : CNN context.



Convolve with



Conv/filter Kernel: Edge detection

Back to CNNs:

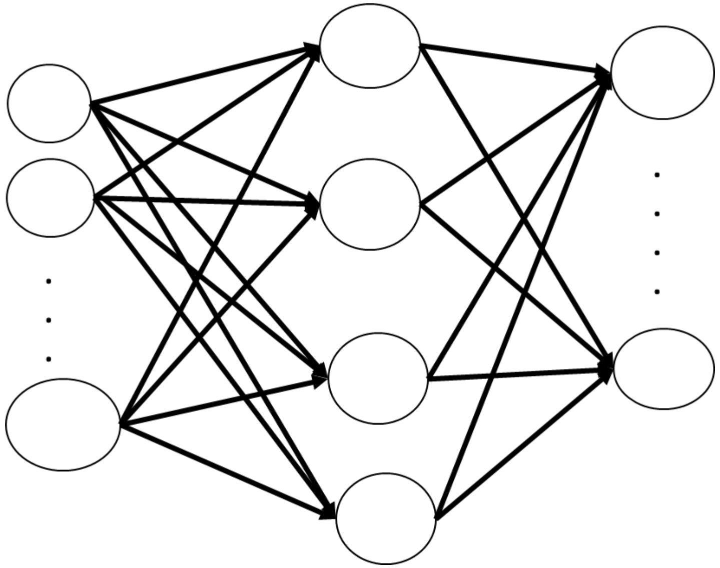
Images can be represented as 2D array.

Consider (i,j) → any position in an image.

Consider Hidden layers as 2-D array ,

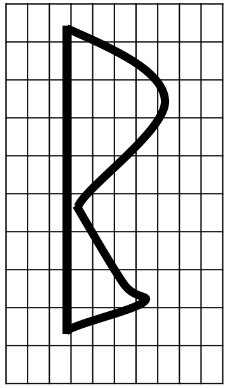
Then, dense layers → 4D tensors (Weights in a hidden layer X no of layers)

Weights matrices become weight tensors

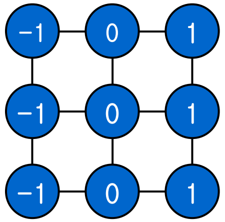


$$h[i,j] = \sum_{a,b} W(i,j,a,b) \cdot x(i+a, j+b)$$

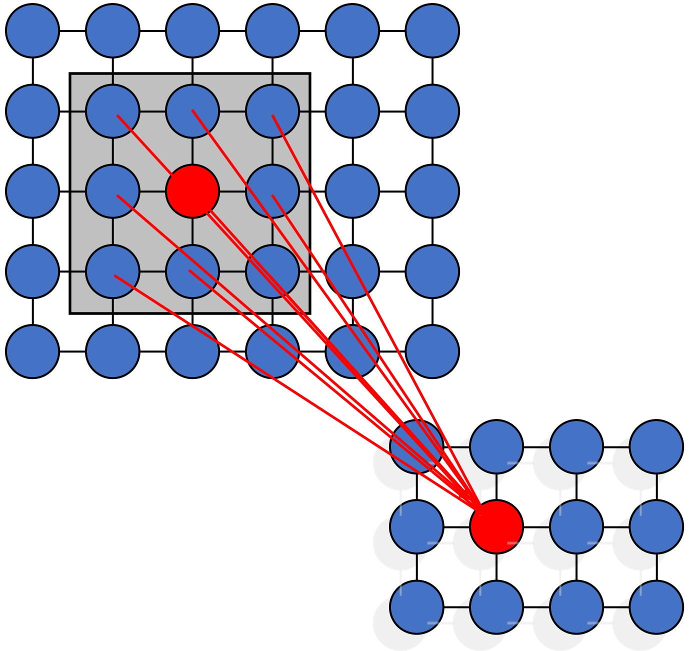
Convolution Operator : CNN context.



Convolve with



Conv/filter Kernel: Edge detection

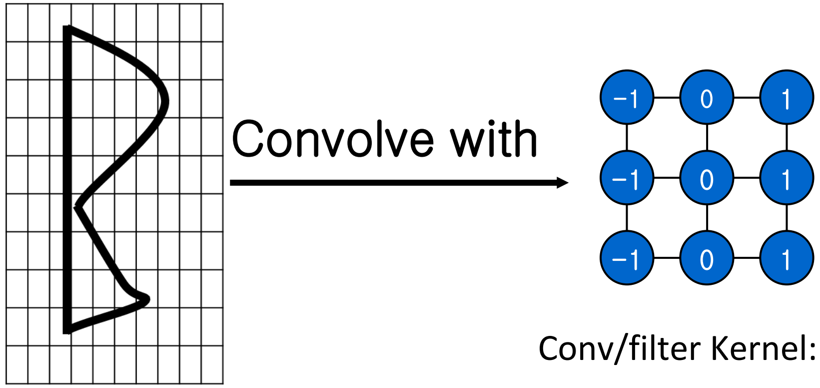


For any location, (i,j), consider an activation value in hidden layer $h[i,j]$

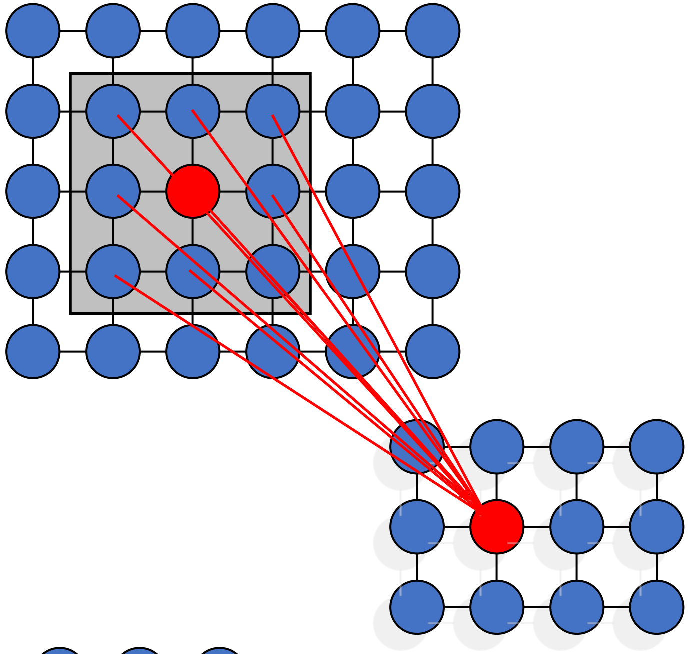
$h[i,j]$ is computed by summing over pixels in x and centered around (i,j).

$$h[i,j] = \sum_{a,b} W(i,j,a,b) \cdot x(i+a, j+b)$$

Convolution Operator : CNN context



Conv/filter Kernel: Edge detection

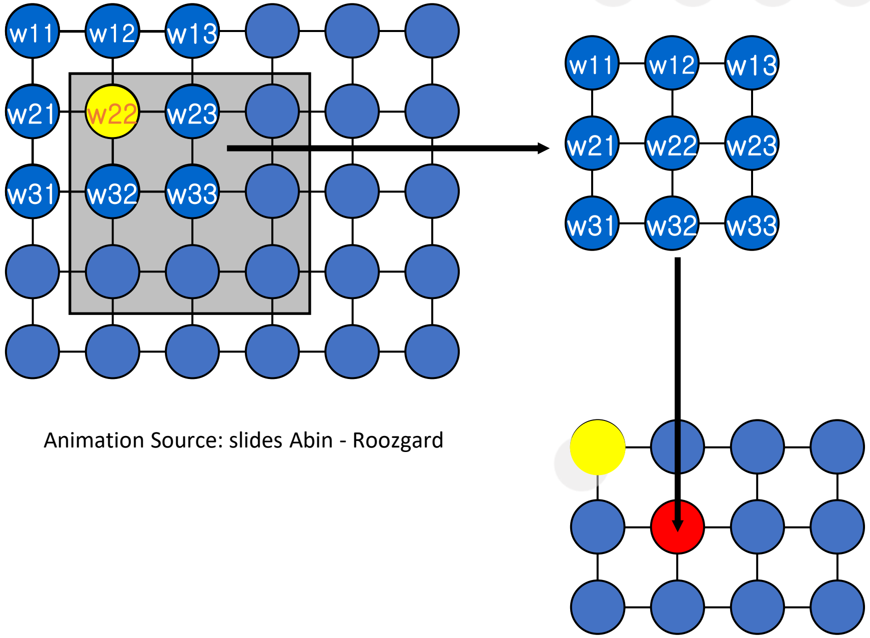


For any location, (i,j), consider an activation value in hidden layer $h[i,j]$

$h[i,j]$ is computed by summing over pixels in x and centered around (i,j).

Run the image kernel (filter kernel, convolution) over entire a and b.

$$h[i,j] = \sum_{a,b} W(i,j,a,b) \cdot x(i+a, j+b)$$



Animation Source: slides Abin - Roozgard

Convolution Operator

Invoke *Translation invariance:*

Now, activation *h should only change with* shift in inputs *x* .

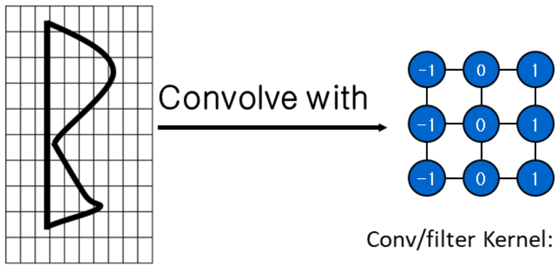
Or, filter kernel (weights) should be same for all *(i,j)*(pixel positions)

➔This means same feature is searched over whole image.

➔In this way all neurons detect the same feature at different positions in the input image.

$$W[i,j,a,b] = V[a,b]$$

$$h[i,j] = \sum_{a,b} V[a,b] \cdot x[i+a, j+b]$$



Conv/filter Kernel: Edge detection

Source :Blog AndrewSzot

Convolution Operator

Invoke *Locality:*

The feature should be recognized using local aspects, look in proximity and not very far.

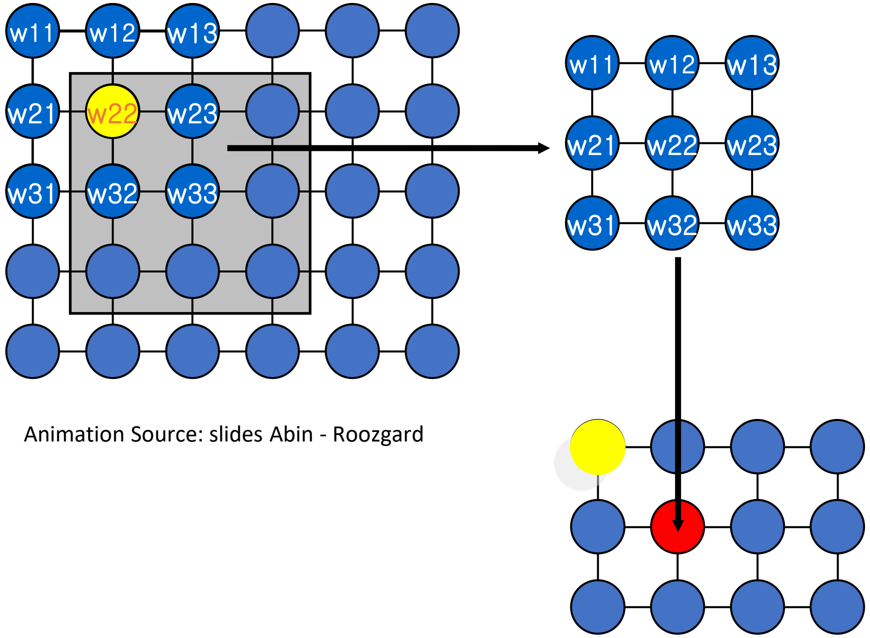
i.e. constrain the size of the kernel filter.

$$W[i,j,a,b] = V[a,b]$$

$$h[i,j] = \sum_{a,b} V[a,b].\cdot x[i+a, j+b]$$

$$for\ |a|,|b| > \Delta$$
$$put\ V[a,b] = 0$$

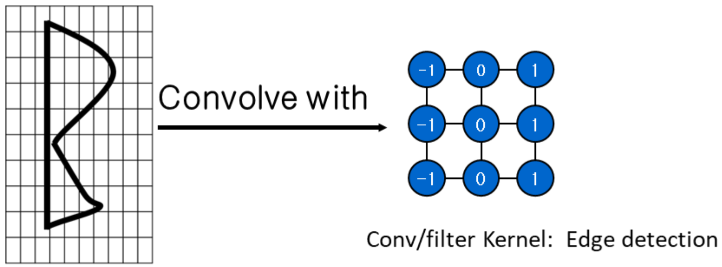
$$h[i,j] = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} V[a,b] \cdot x[i+a, j+b]$$



Animation Source: slides Abin - Roozgard

Source :Blog AndrewSzot

Example:



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

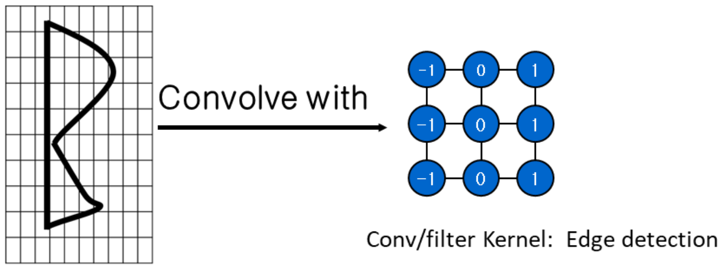
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

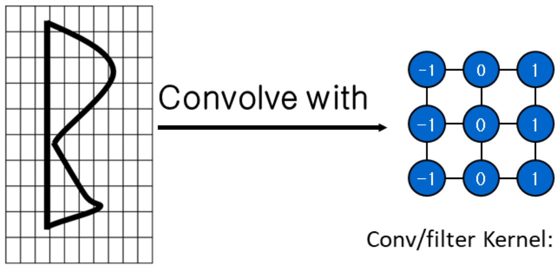
convolve (slide) over all spatial locations

3			

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:



Conv/filter Kernel: Edge detection

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

3	-1		

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:

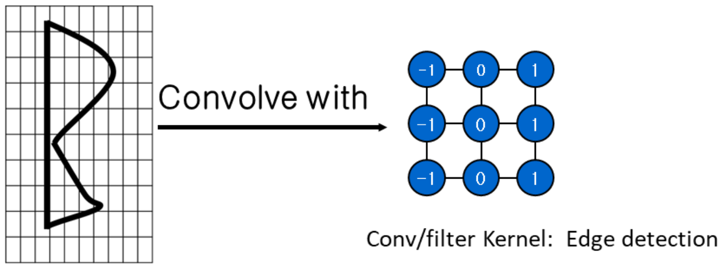
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



convolve (slide) over all spatial locations

3	-1	-3	

⋮ ⋮

Each filter detects a small feature (3 x 3).

POLYTECH NANCY UNIVERSITÉ DE LORRAINE CRAN

Example:

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

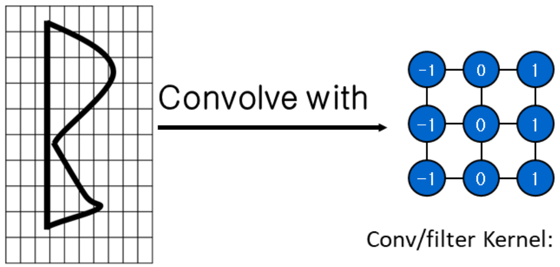
6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

3	-1	-3	-1



Conv/filter Kernel: Edge detection

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:

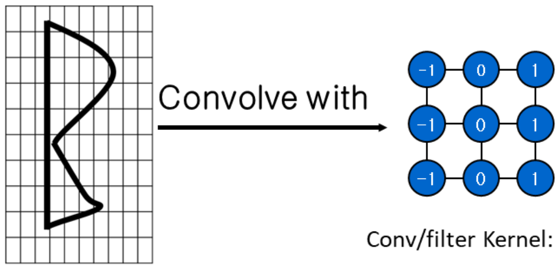
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



Conv/filter Kernel: Edge detection

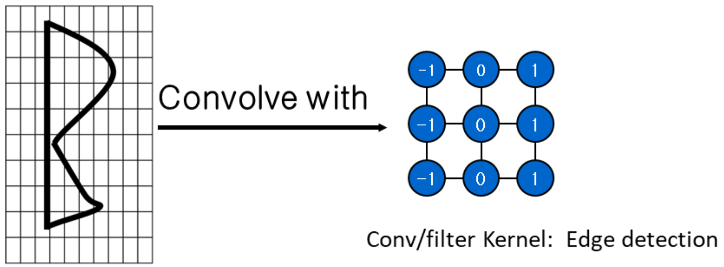
convolve (slide) over all spatial locations

3	-1	-3	-1
-3			

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:



stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

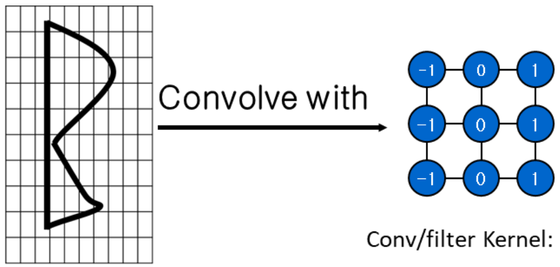
3	-1	-3	-1
-3	1	0	

⋮ ⋮

Each filter detects a small feature (3 x 3).

POLYTECH NANCY UNIVERSITÉ DE LORRAINE CRAN

Example:



Conv/filter Kernel: Edge detection

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

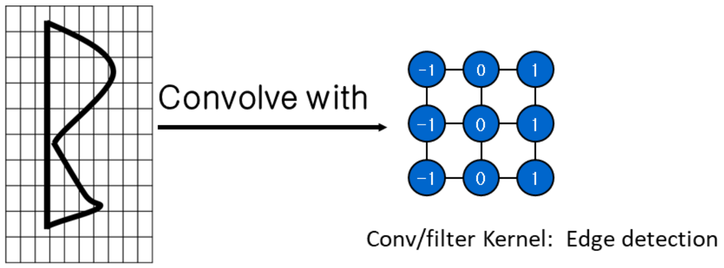
3	-1	-3	-1
-3	1	0	

⋮ ⋮

Each filter detects a small feature (3 x 3).

POLYTECH NANCY UNIVERSITÉ DE LORRAINE CRAN

Example:



stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	1

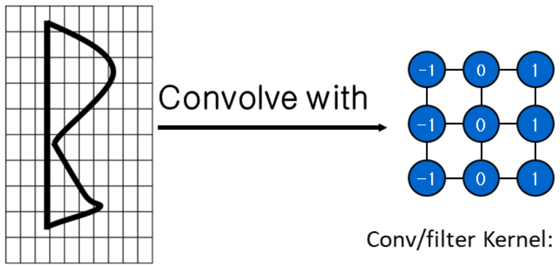
Feature Map

⋮ ⋮

Each filter detects a small feature (3 x 3).

POLYTECH NANCY UNIVERSITÉ DE LORRAINE CRAN

Example:



Conv/filter Kernel: Edge detection

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	1

Feature Map

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

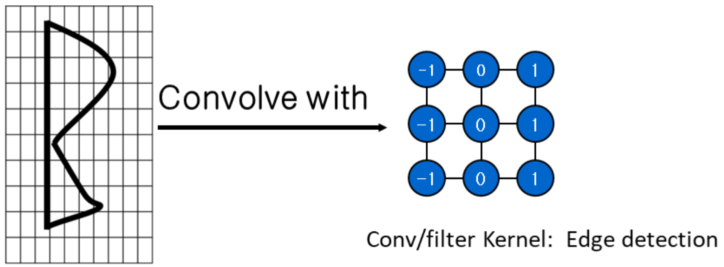
-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	4	3

Feature Maps

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	4	3

Feature Maps

⋮ ⋮

2 images of 4 x 4 matrix is produced.
This procedure is repeated for each filter

Example: Edge detection

1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1

H x W image

1	-1

Edge detector kernel

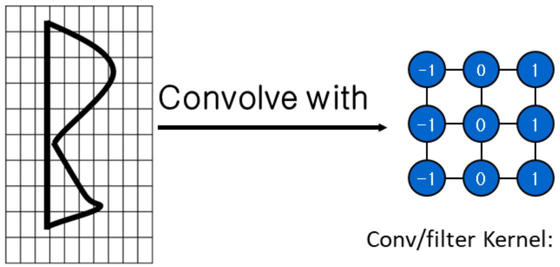
0	1	0	0	0	-1	0
0	1	0	0	0	-1	0
0	1	0	0	0	-1	0
0	1	0	0	0	-1	0
0	1	0	0	0	-1	0
0	1	0	0	0	-1	0

Feature Map

Kernel: if
horizontally elements are same , output is 0.
Else, non-zero.

Detected:
1 for edge from white to **black**
-1 for edge from **black** to white

Difficult to handcraft such filters.
Thus, filter kernel weights must be learnt !!



Conv/filter Kernel: Edge detection

Remarks:

Output shape determined by shape of input and convolutional kernel window.

Small convolution with filter kernels → "smaller" outputs (feature maps).

$$n_h \left\{ \underbrace{\begin{array}{|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 \\ \hline \end{array}}_{n_w} \right. \qquad k_h \left\{ \underbrace{\begin{array}{|c|c|c|} \hline 1 & -1 & -1 \\ \hline -1 & 1 & -1 \\ \hline -1 & -1 & 1 \\ \hline \end{array}}_{k_w} \right. \Longrightarrow \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array}$$

$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

Padding and Strides

Padding

- Multiple layers of convolution may reduce the information available at boundary.
- Padding prevents this problem.
- Adding zeros around the edges such that multiple convolution operation does not lead to information loss.
- Pixels added around edges.
- These pixels are zero in value.

0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0
0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0

p_h

p_w

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

Padding: In practice

- $p_h = k_h - 1$,
- $p_w = k_w - 1,$
- Kernel dimensions : k_w, k_h are chosen odd numbers (Ex: 1,3,5,7..)
- Padding dimensions are even. $p \quad = k \quad - 1$,
- then, each side padded with $p/2$ *zeros*
- *or,* padding dimensions = *(k-1)/2*

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0
0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0

p_h

p_w

Strides

Stride=1

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=1

Output =5 x5

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Number of rows and columns per slide → stride.

- Useful in reducing information (resolution) drastically.

Convolution : Strides

Input: 7x7 (spatially)

Stride=1

Filter kernel: 3X3

Stride=1

Output =5 x5

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

Stride=1

Input: 7x7 (spatially)

Filter kernel: 3X3

Stride=1

Output =5 x5

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

Input: 7x7 (spatially)

Stride=1

Filter kernel: 3X3

Stride=1

Output =5 x5

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

Stride=1

Input: 7x7 (spatially)

Filter kernel: 3X3

Stride=1

Output =5 x5

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

Input: 7x7 (spatially)

Stride=1

Filter kernel: 3X3

Stride=1

Output =5 x5

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Feature Map

Convolution : Strides

Input: 7x7 (spatially)

Stride=2

Filter kernel: 3X3

Stride=2

Output =3 x3

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

Input: 7x7 (spatially)

Stride=2

Filter kernel: 3X3

Stride=2

Output =3 x3

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

Input : 7x7 (spatially)

Stride=2

Filter kernel: 3X3

Stride=2

Output =3 x3

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

Input: 7x7 (spatially)
Stride=2

Filter kernel: 3X3

Stride=2

Output =3 x3

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=2

Output =3 x3

Stride=2

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

Stride=2

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=2

Output =3 x3

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

Stride=2

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=2

Output =3 x3 Feature map matrix

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

Input: 7x7 (spatially)

Stride=2

Filter kernel: 3X3

Stride=2

Output =3 x3 Feature map matrix

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

In general, with stride =*s*

output size: $$\left(\frac{n_h - \kappa_h + p_h}{s} + 1\right) \times \left(\frac{n_w - k_w + p_w}{s} + 1\right)$$

Convolution : Strides

Input: 7X7 (spatially)

Stride=3

Filter kernel: 3X3

Stride=3 ?? (stride increased)

Cannot apply 3x3 filter kernel on 7X7 input → Does not fit.

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

Stride=3

Input: 7X7 (spatially)

Filter kernel: 3X3

Stride=3 ?? (stride increased)

Cannot apply 3x3 filter kernel on 7X7 input → Does not fit.

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Convolution : Strides

In general, with stride $=s$

output size: $$\left(\frac{n_h - \kappa_h + p_h}{s} + 1\right) \times \left(\frac{n_w - k_w + p_w}{s} + 1\right)$$

Stride=3

Input: 7X7 (spatially)

Filter kernel: 3X3

Stride=3 ?? (stride increased)

Cannot apply 3x3 filter kernel on 7X7 input → Does not fit.

1	0	0	0	0	1	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	1	0	0
0	1	0	0	1	0	0
0	0	1	0	1	0	0
0	0	1	0	1	0	0

Apply padding

0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	0
0	0	1	0	0	1	0	0	0
0	0	0	1	1	0	0	0	0
0	1	0	0	0	1	0	0	0
0	0	1	0	0	1	0	0	0
0	0	0	1	0	1	0	0	0
0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=3 ?? (stride increased)

Output= 3 X 3

In general, with stride =*s*

output size: $$\left(\frac{n_h - \kappa_h + p_h}{s} + 1\right) \times \left(\frac{n_w - k_w + p_w}{s} + 1\right)$$

Apply Padding: 1 pixel border on each side (p_h=2 , p_w=2)

Kernel =3X3, Stride =1 ,

output =7 X 7 !!

In practice:

Stride =1,

kernel dim: F X F where F is an odd number (Ex: 1,3,5,7..)

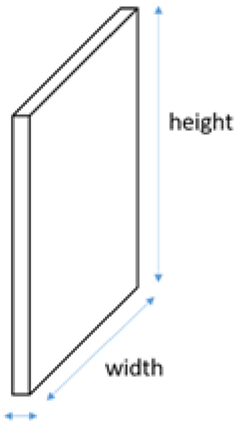
Padding on each side = (F-1)/2

Multi input and output channels

Multi input channels

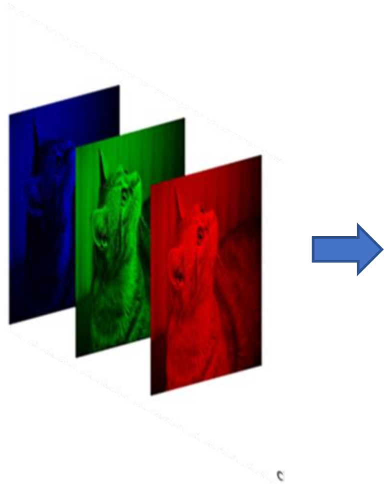
so far, greyscale images→ One channel.

Most images are colorful → 3 channels RGB



→Input as multi-dimensional array : 3 X *h* X w

→construct a convolution kernel with the same number of input channels as the input data (3 here)

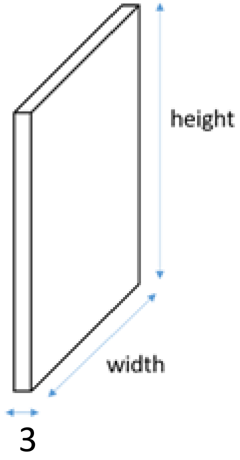


1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Multi input channels

so far, greyscale images→ One channel.

Most images are colorful → 3 channels RGB



➔Input as multi-dimensional array : 3 X *h* X w

➔construct a convolution kernel with the same number of input channels as the input data (3 here)

➔Assign a 2-D kernel to each channel → concatenation gives 3D conv kernel.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Multi input channels

Convolution with 3 input channels:

- slide the 2D filter kernel on 2D input , for each channel.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

*

1	-1	-1
-1	1	-1
-1	-1	1

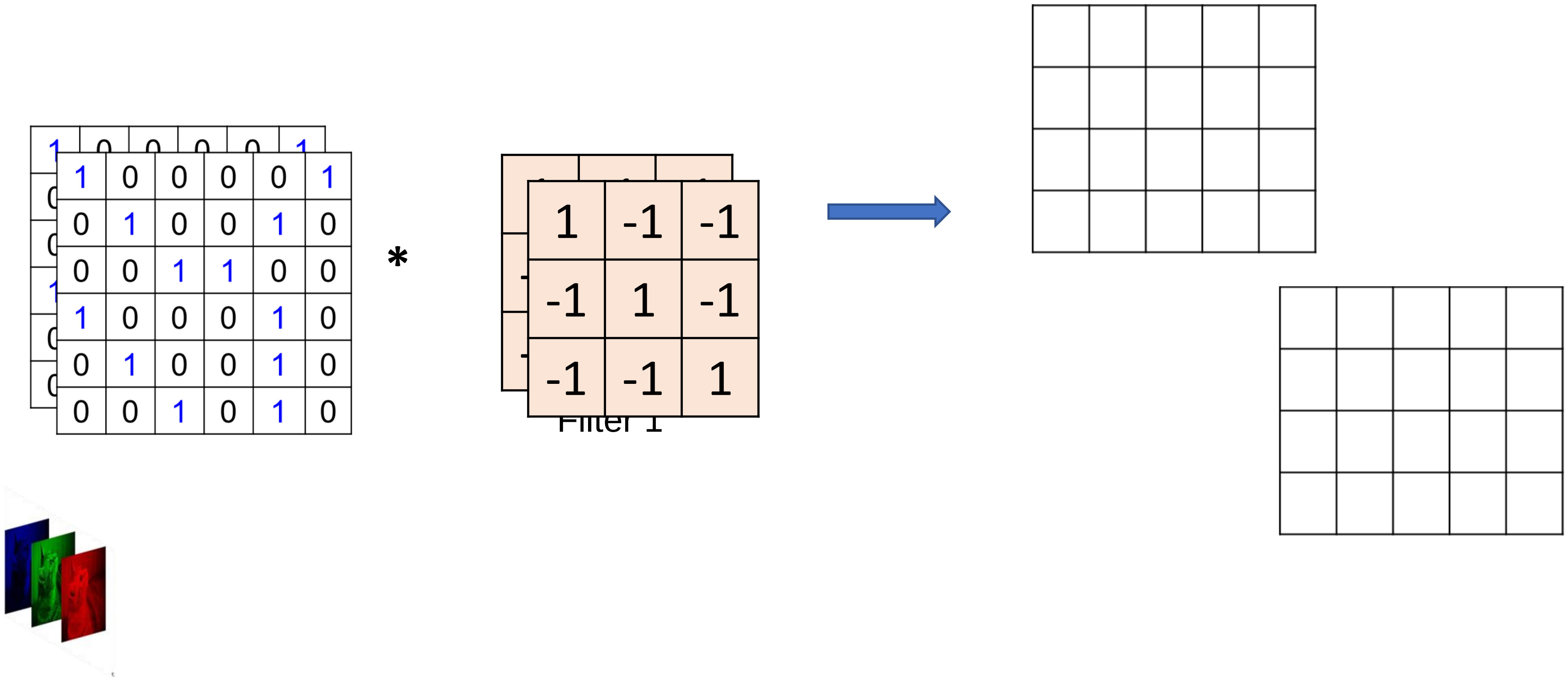
Filter 1

→

Multi input channels

Convolution with 3 input channels:

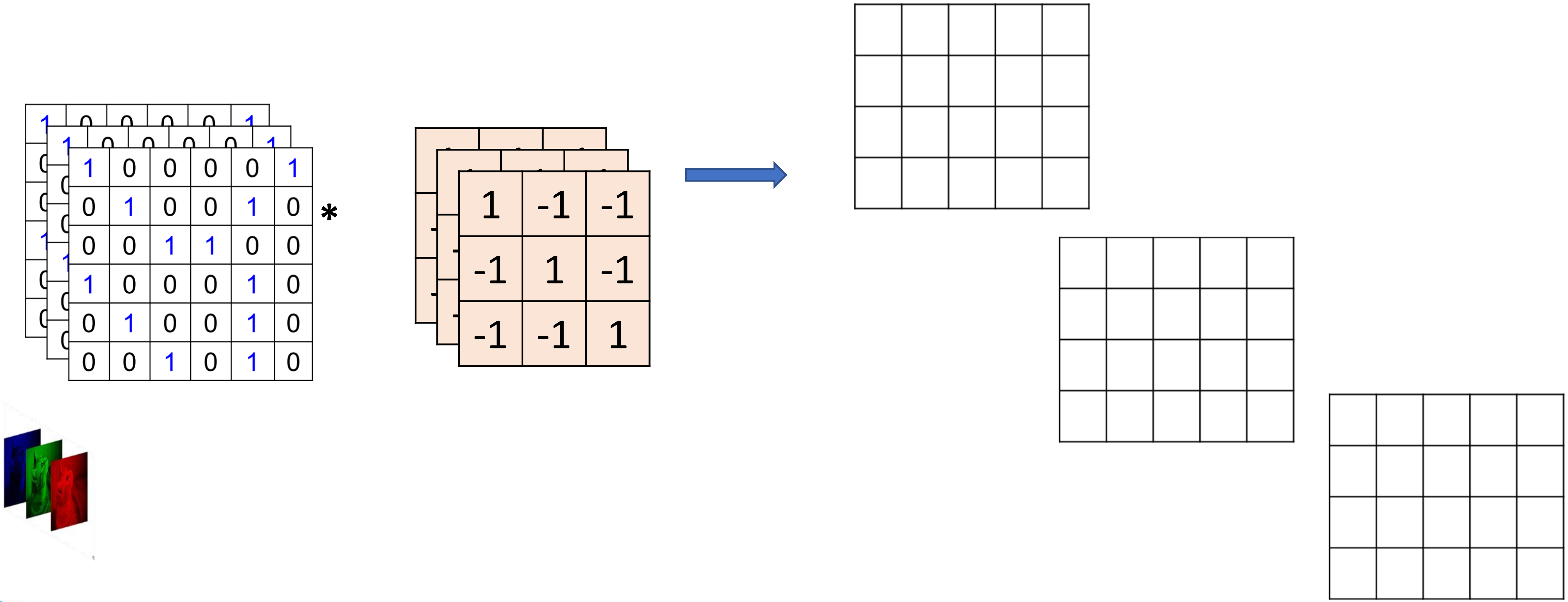
- slide the 2D filter kernel on 2D input , for each channel.



Multi input channels

Convolution with 3 input channels:

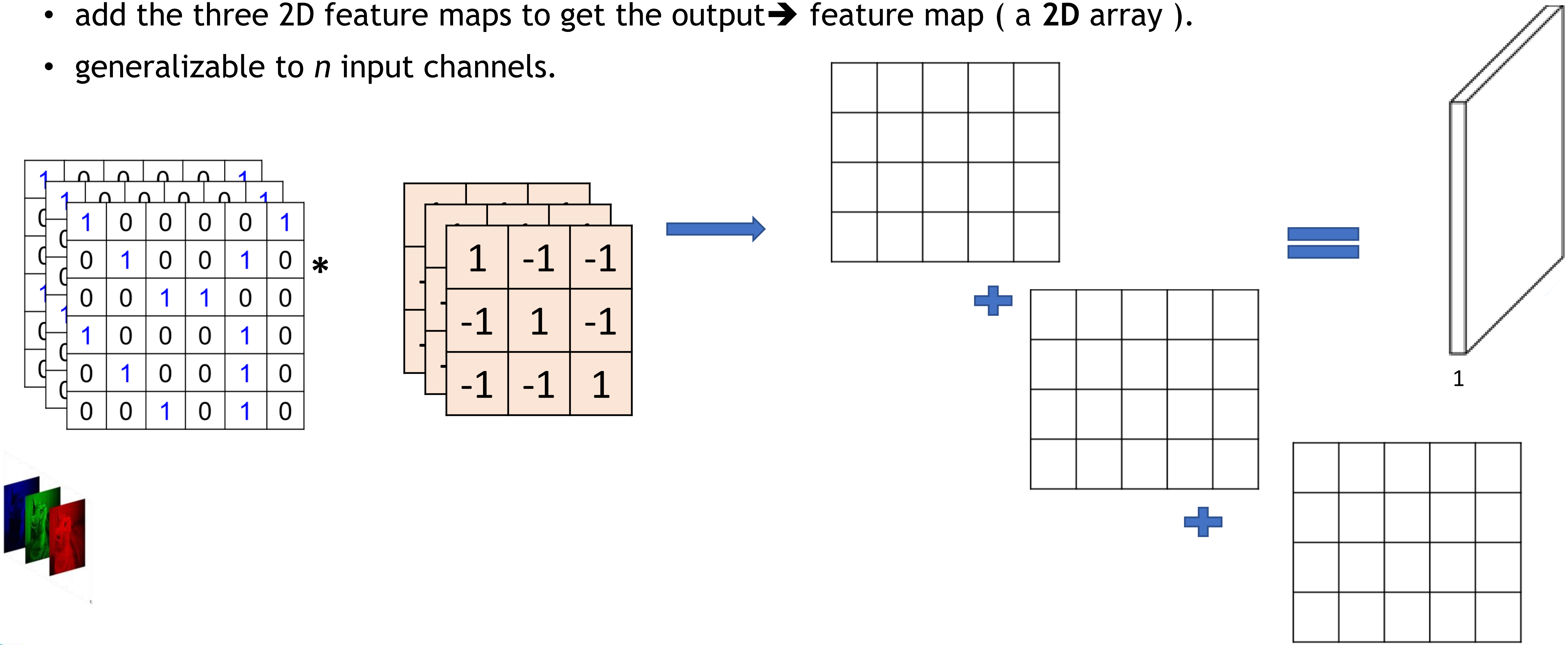
- slide the 2D filter kernel on 2D input , for each channel.



Multi input channels

Convolution with 3 input channels:

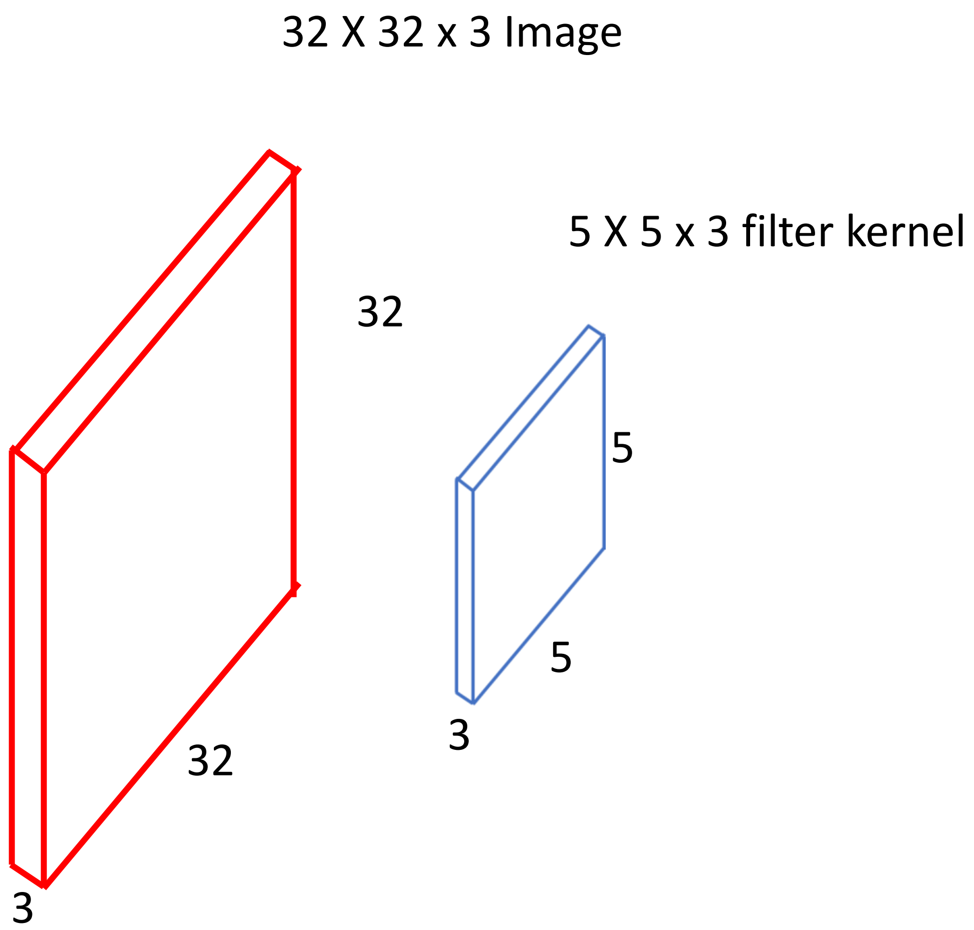
- slide the 2D filter kernel on 2D input , for each channel.
- add the three 2D feature maps to get the output➔ feature map (a **2D** array).
- generalizable to *n* input channels.



Multi input channels: Summary

Convolution of image (3 channels)

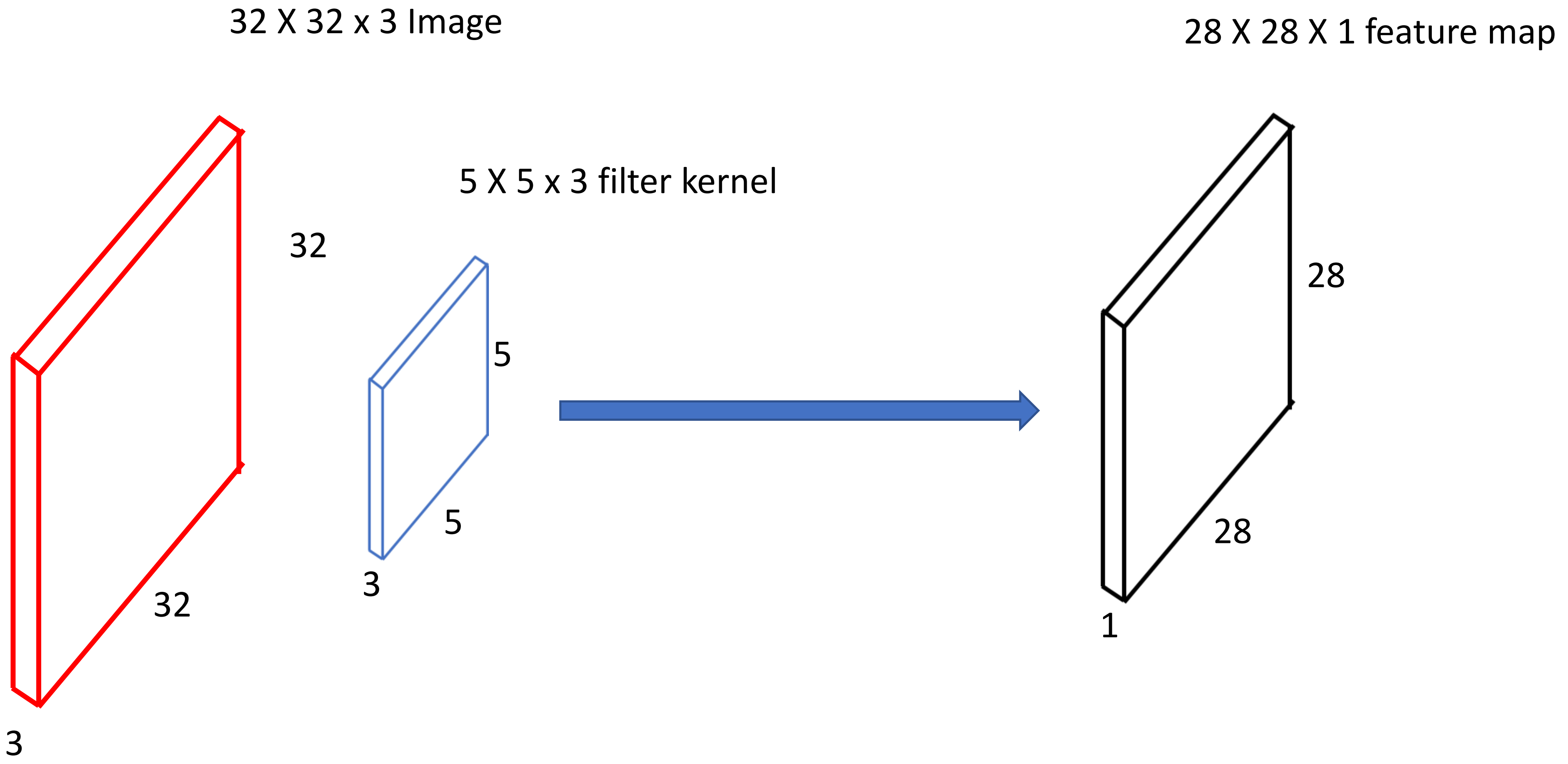
with 3 channel filter ➔ 1-D feature map.



Multi input channels: Summary

Convolution of image (3 channels)

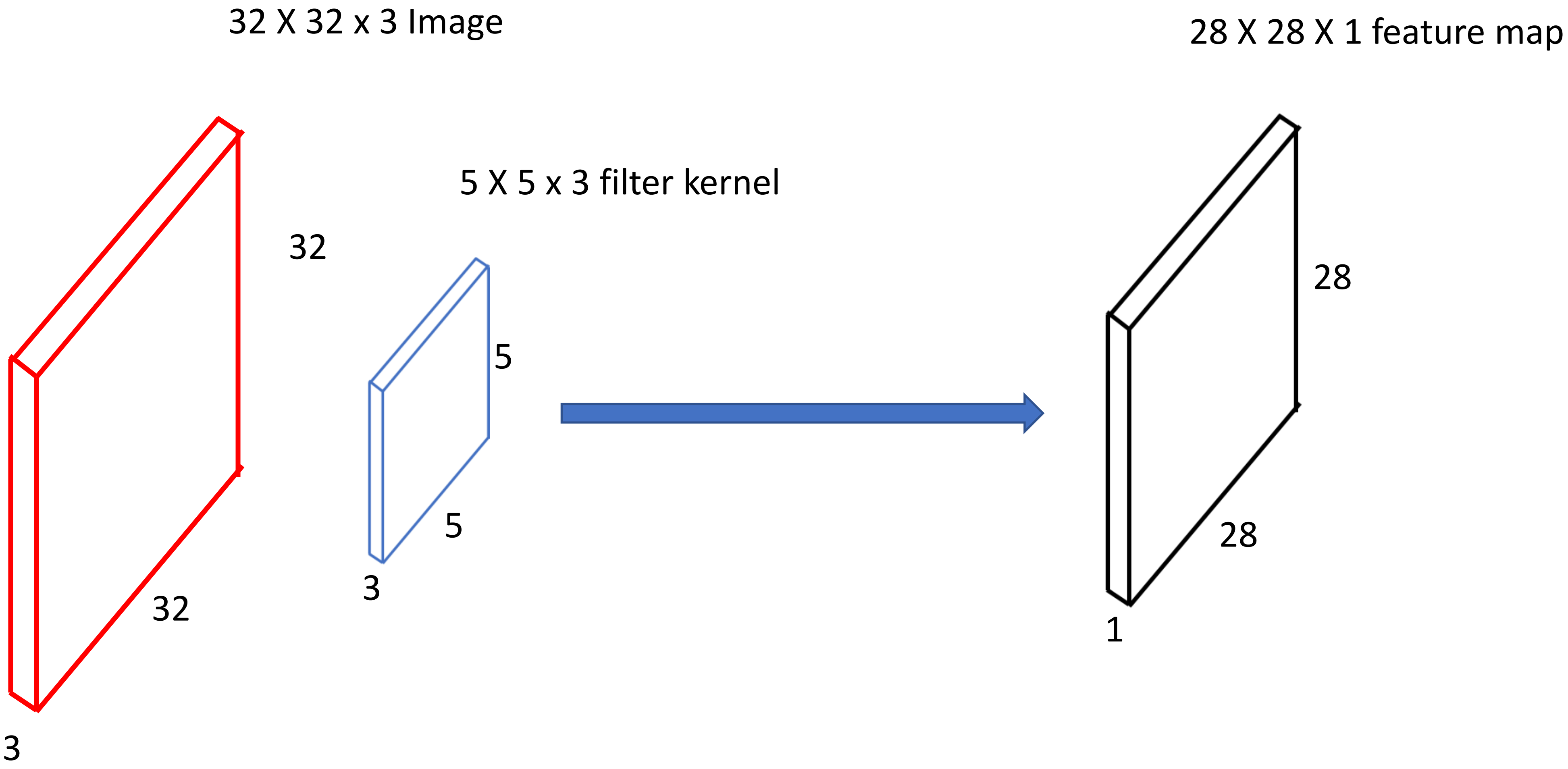
with 3 channel filter ➔ 1-D feature map.



Multi input channels: Summary

Convolution of image (3 channels)

with 3 channel filter ➔ 1-D feature map.



Multi outputs

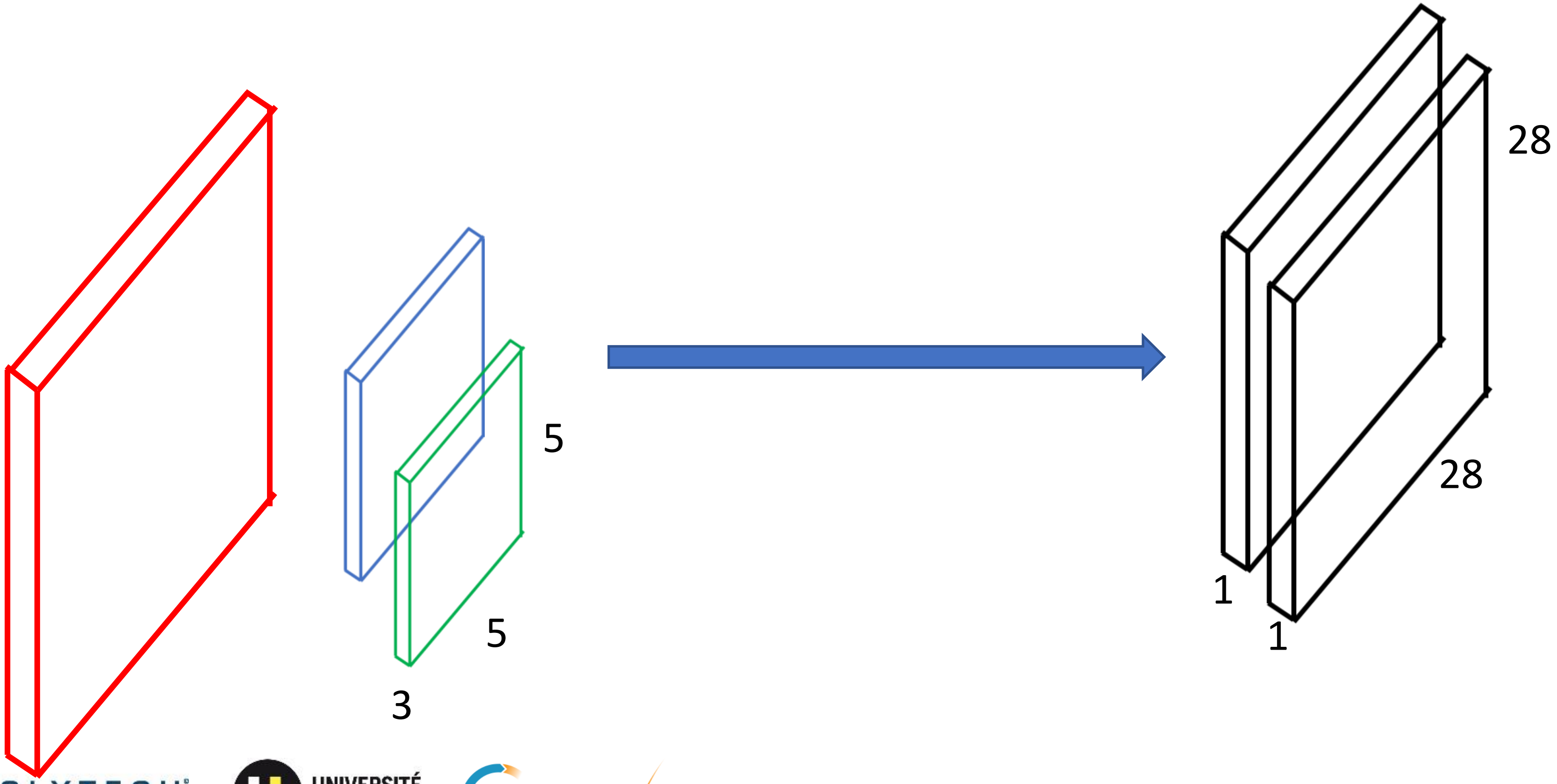
When more than one feature is to be extracted → multiple filters are used.

Output for each filter is desired. Output has multiple channels.

Convolution is performed with each filter kernel , for each output channel.

Output is concatenated along number of filter (output channel) dimension.

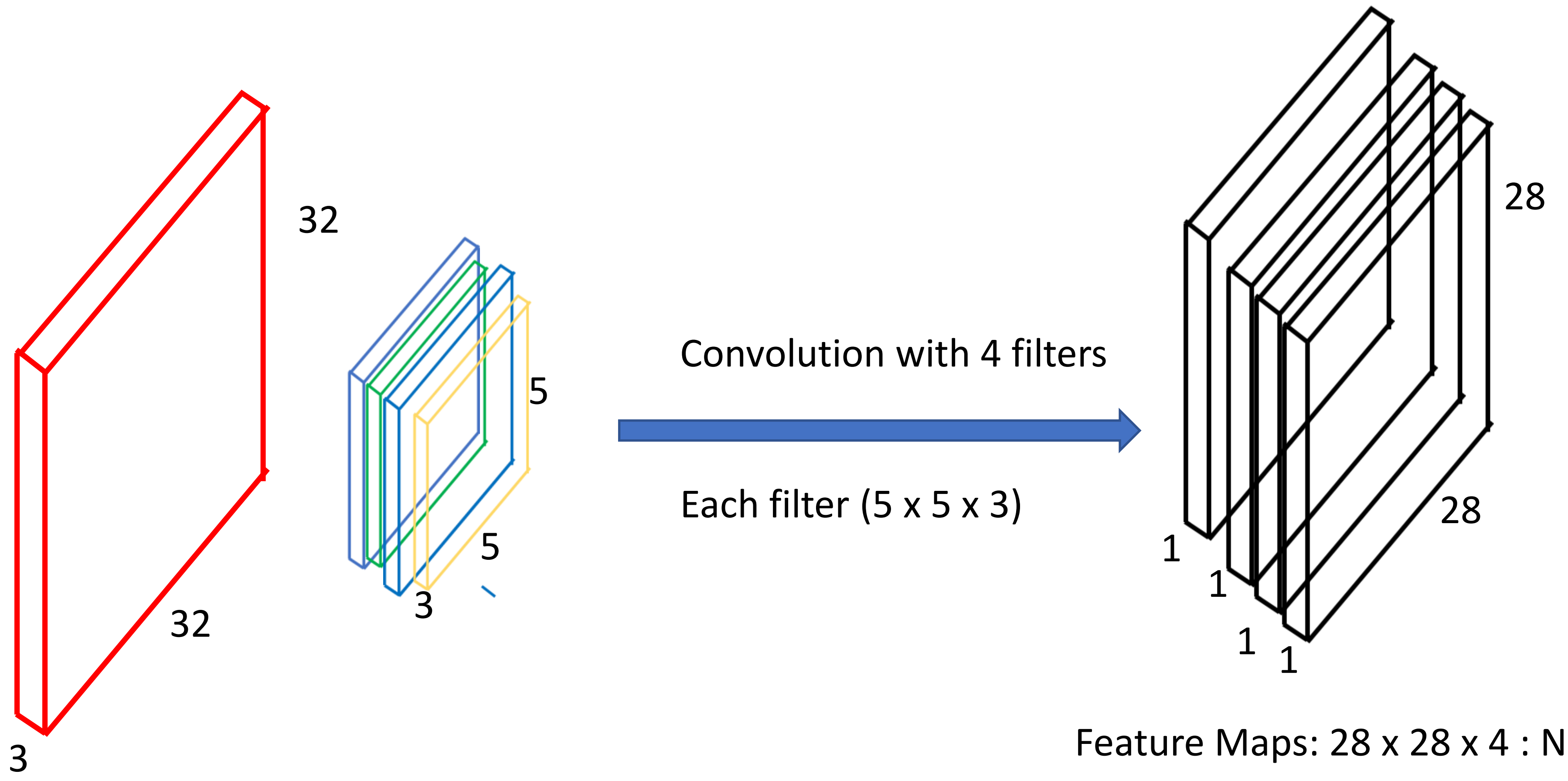
2 different filters ➔ convolution with each filter kernel and concatenated along output channel dimension.



Multi outputs

When more than one feature is to be extracted → multiple filters are used.

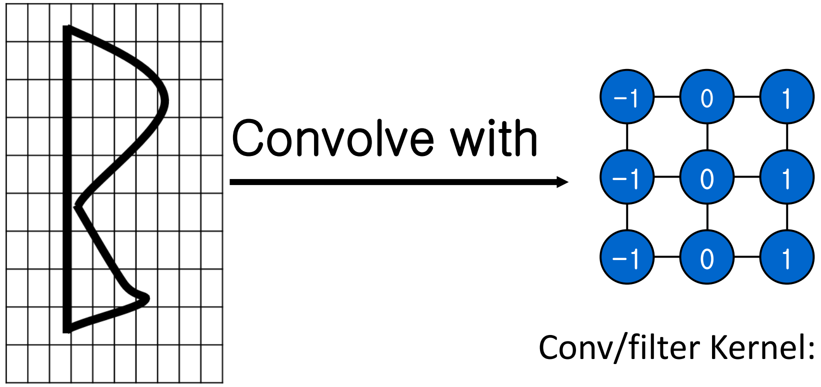
Output for each filter is desired. Output has multiple channels.



Feature Maps: 28 x 28 x 4 : New image!

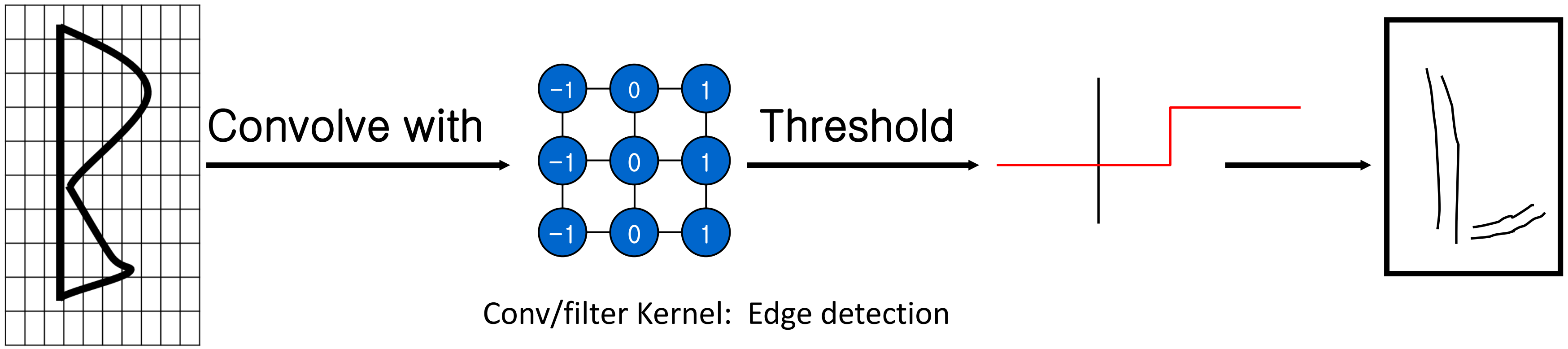
Summary

- So far: Just Convolutions!



Conv/filter Kernel: Edge detection

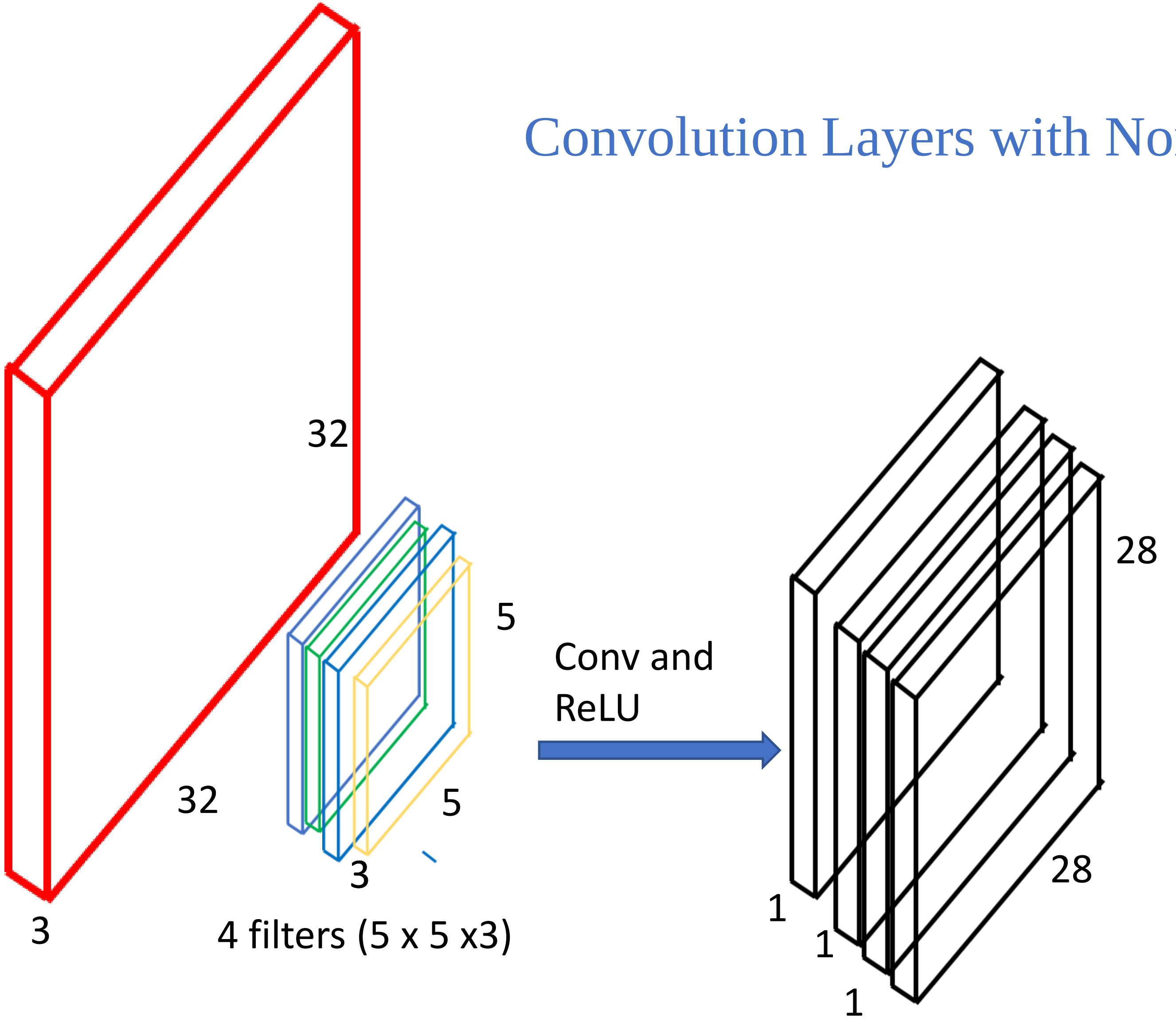
Summary



- Apply Non-linearity (as seen earlier) : features pass thru activation functions → activation maps (terminology is loose , feature maps/activation maps both are used often to mean the same)

- In practice: ReLu is mostly preferred (fast convergence, no zero-gradient problem..

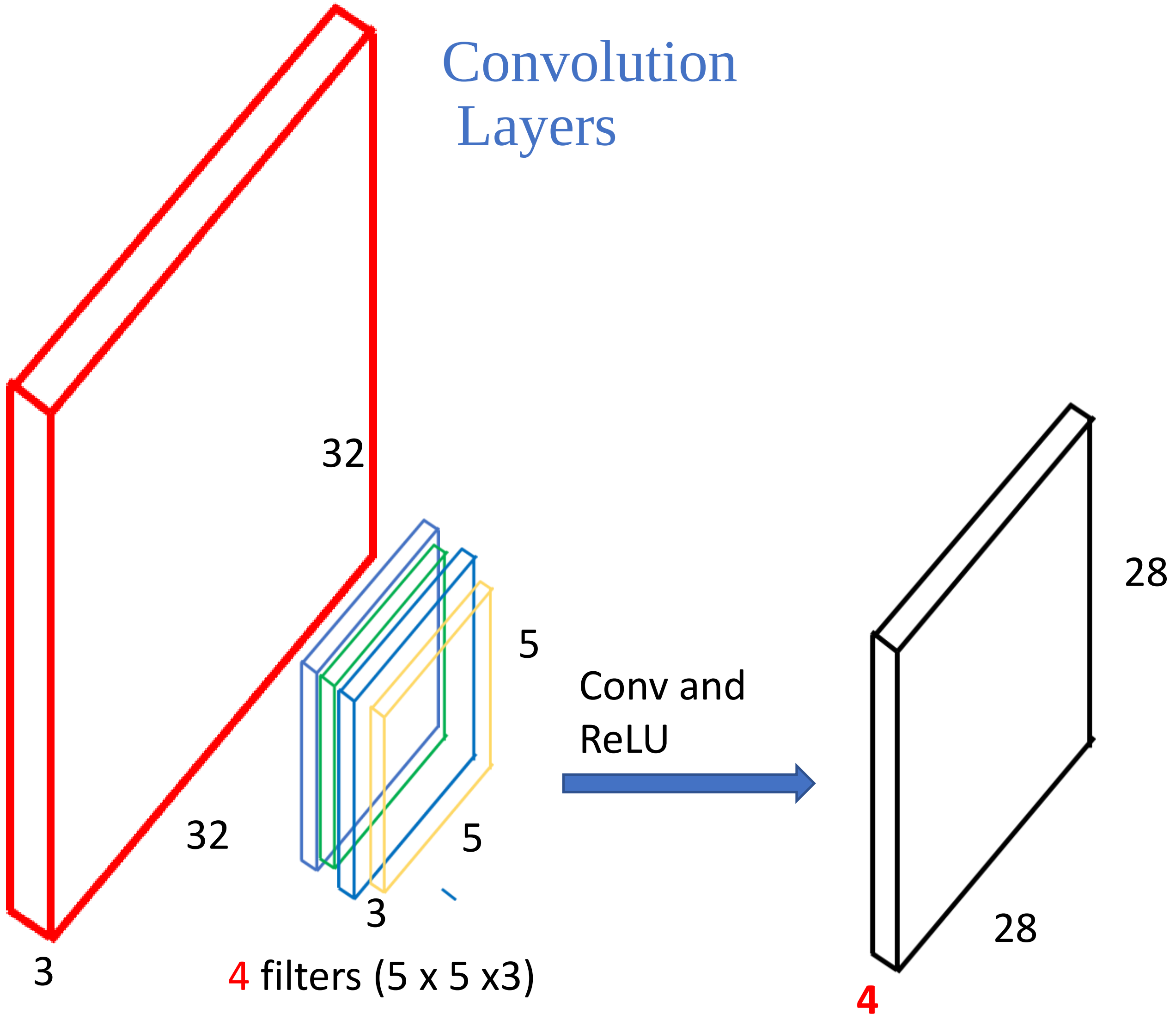
Convolution Layers with Non linearity Activation



Feature Maps / Activation maps
: 28 x 28 x 4

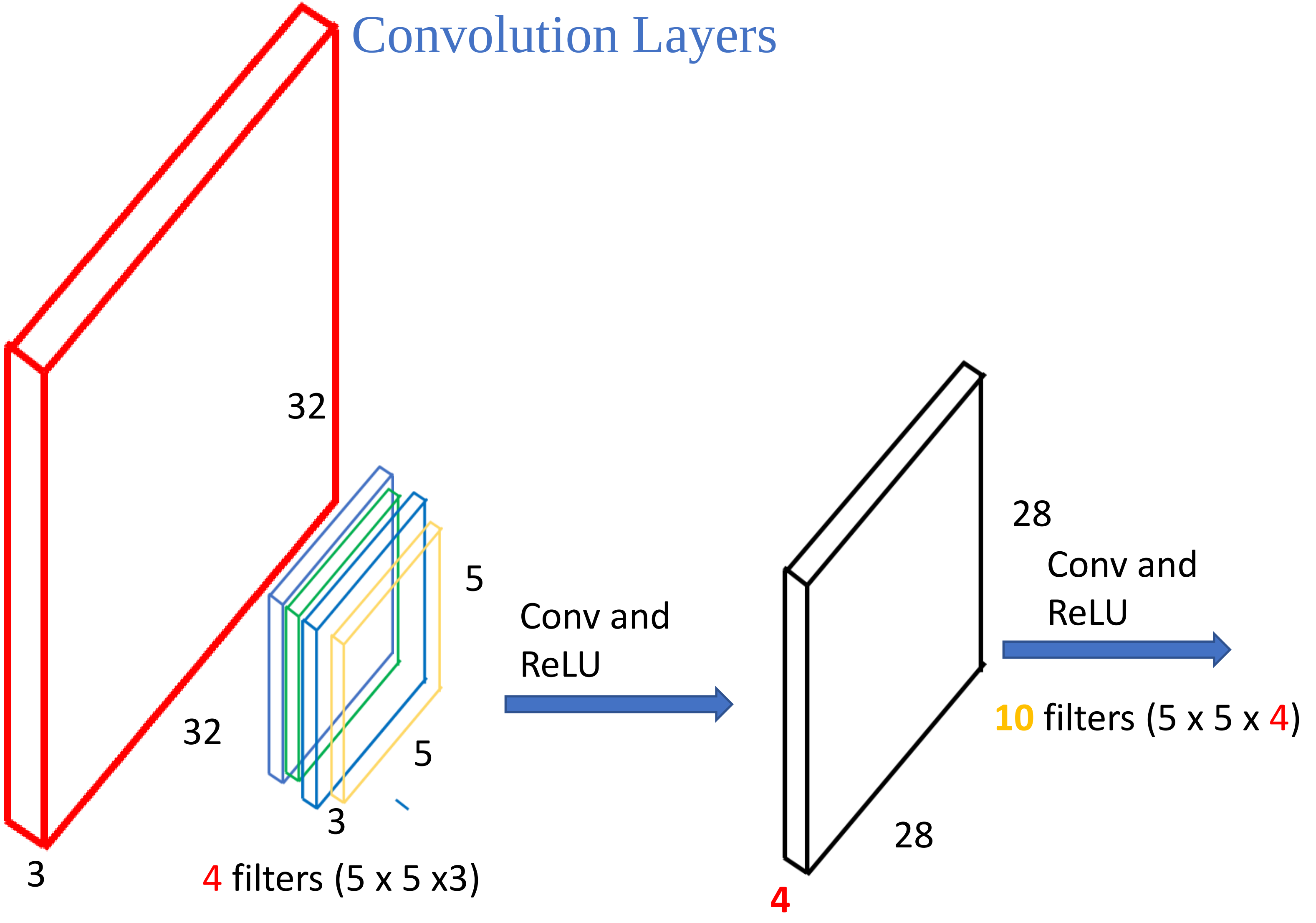
Feature Maps: 28 x 28 x 4 : New image!

Convolution Layers



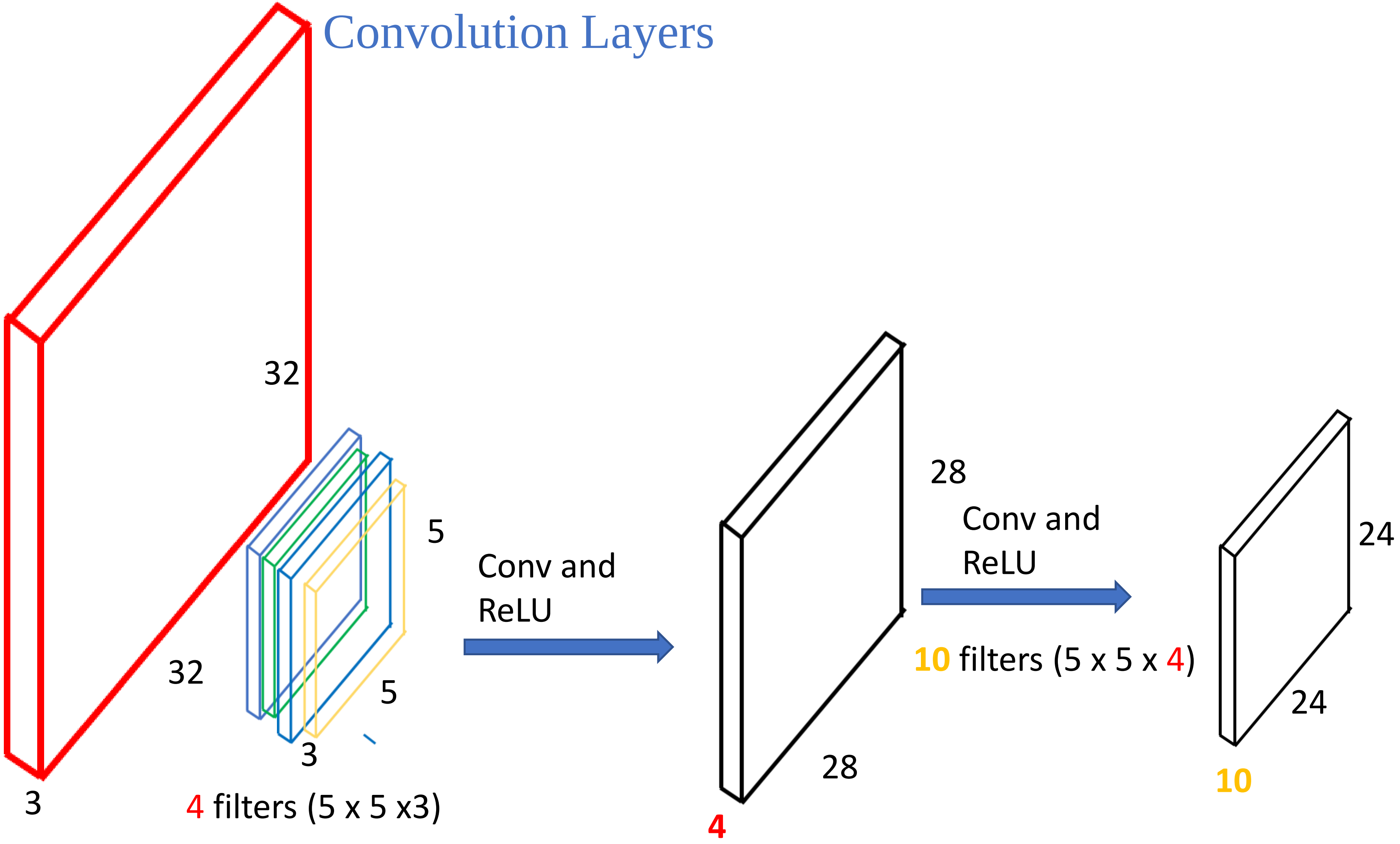
Feature Maps / Activation maps
: 28 x 28 x 4

Convolution Layers



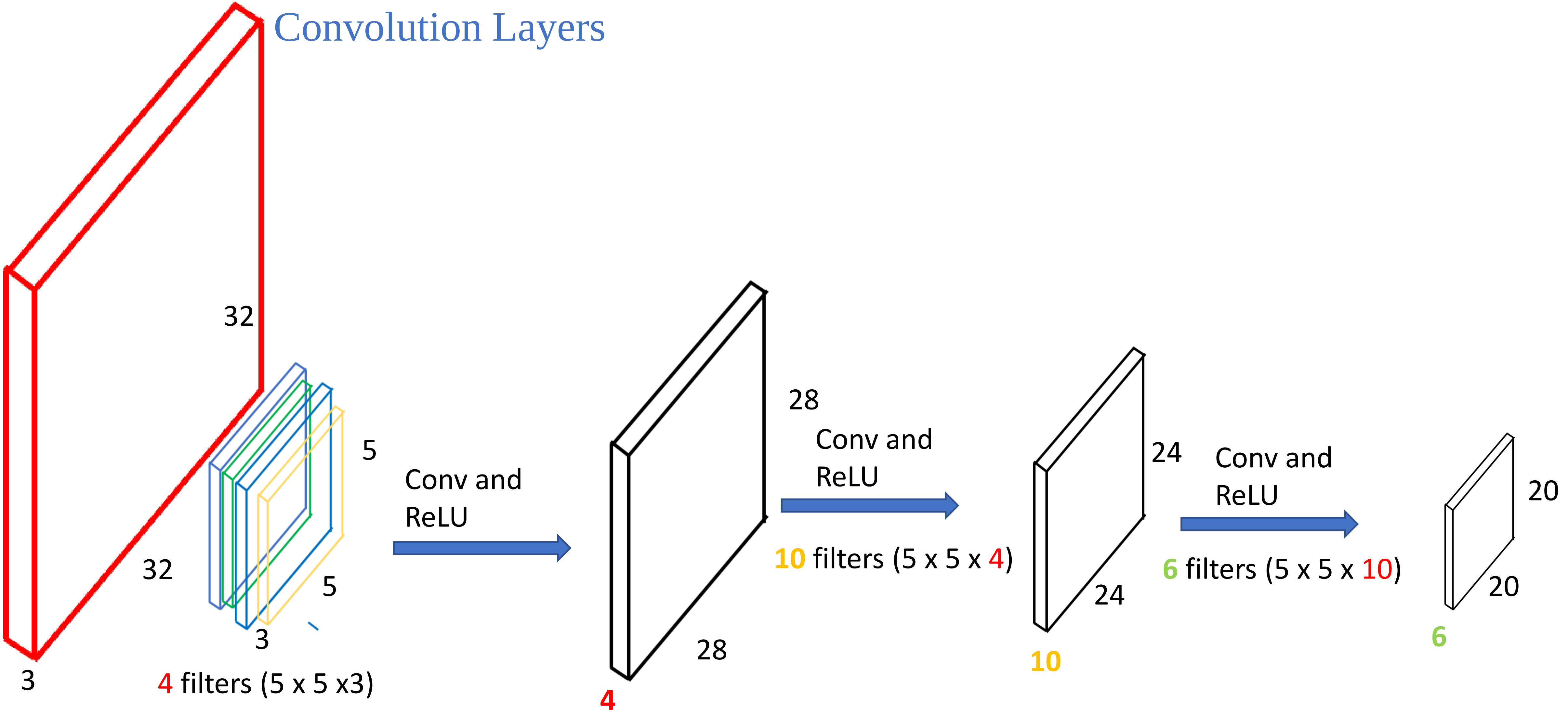
Feature Maps / Activation maps
: 28 x 28 x 4

Convolution Layers



Feature Maps / Activation maps
: 28 x 28 x 4

Convolution Layers



Feature Maps / Activation maps
: 28 x 28 x 4

Convolutional neural network (CovNets) CNNs

Convolution Layers

Remarks: Observe the reduction in size.

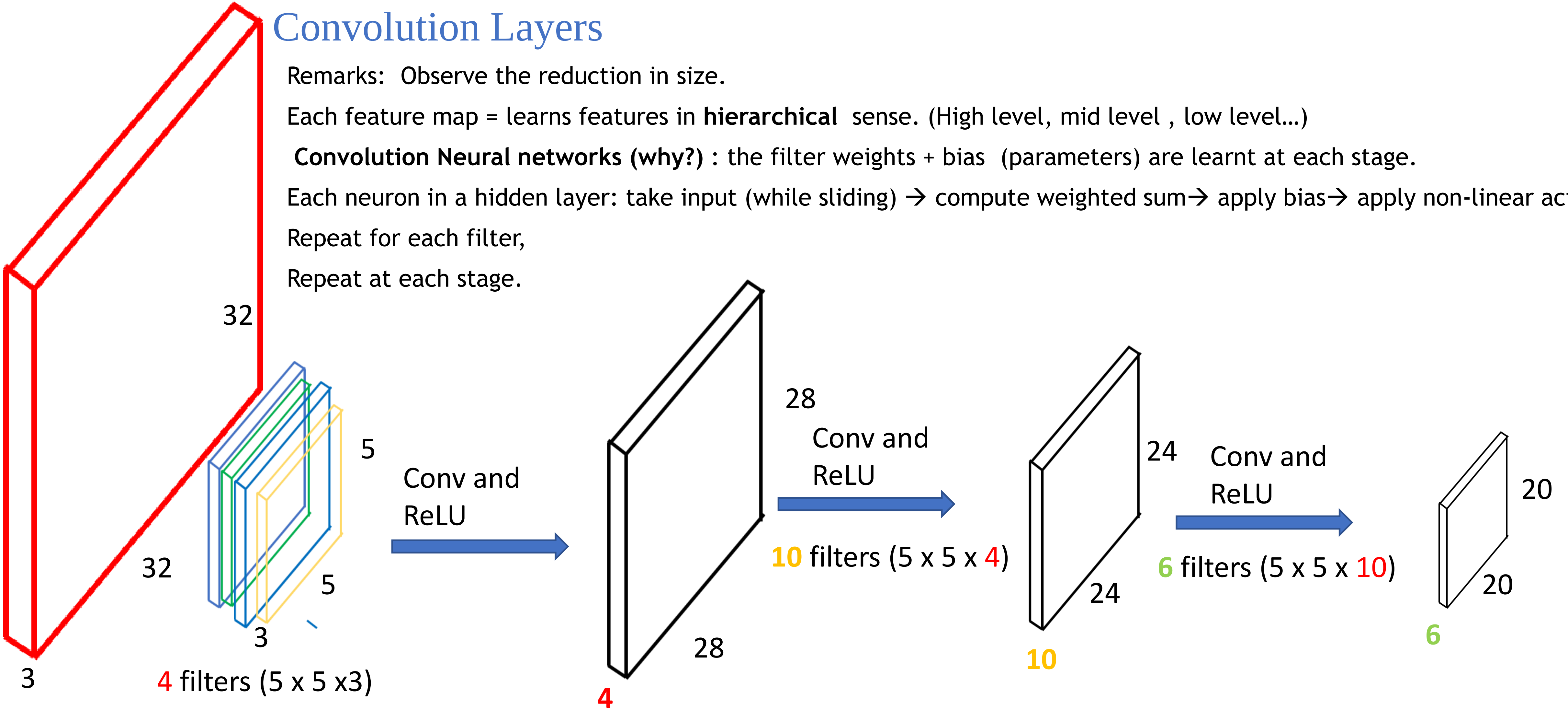
Each feature map = learns features in **hierarchical** sense. (High level, mid level , low level…)

Convolution Neural networks (why?) : the filter weights + bias (parameters) are learnt at each stage.

Each neuron in a hidden layer: take input (while sliding) → compute weighted sum→ apply bias→ apply non-linear activation

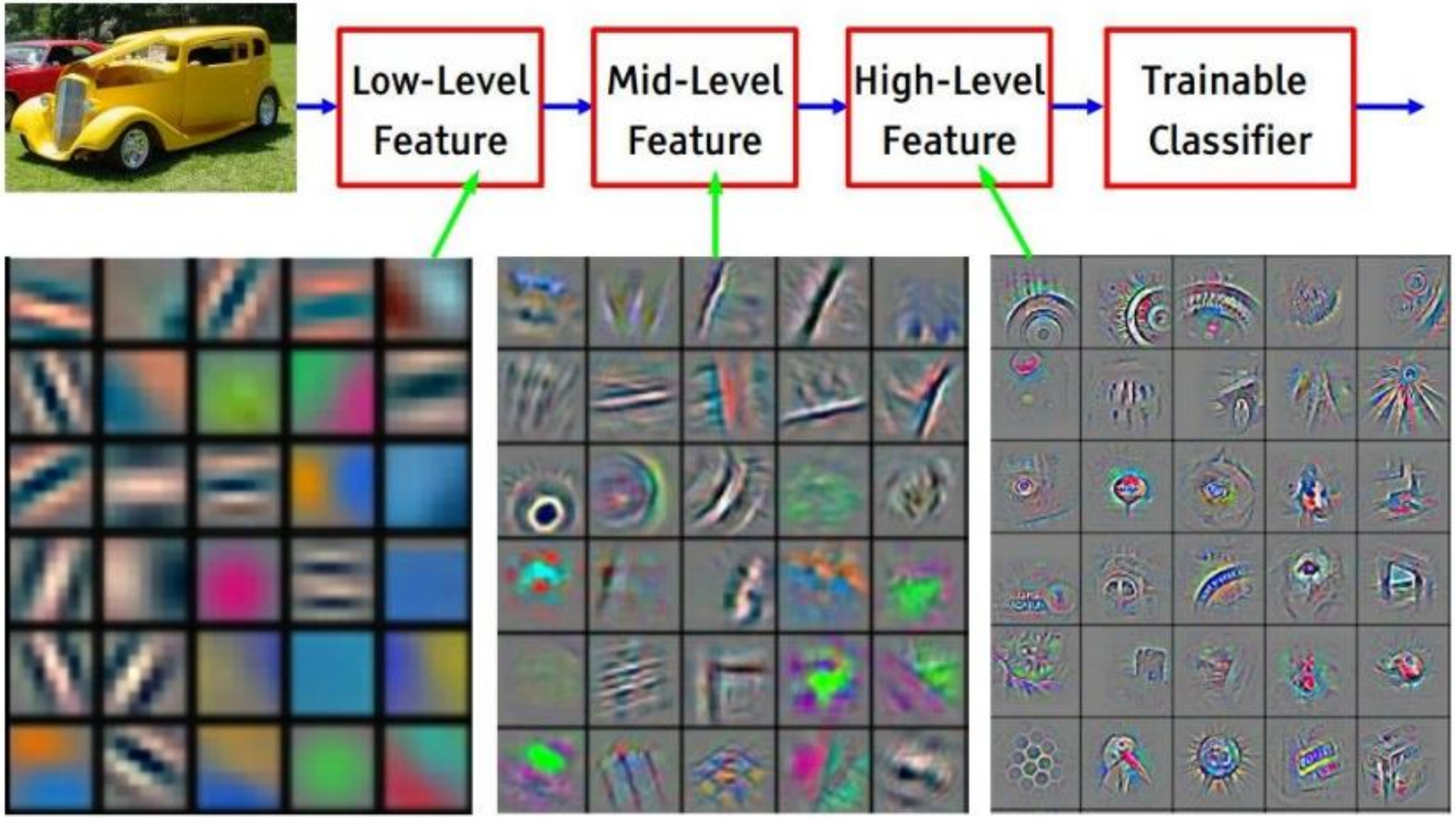
Repeat for each filter,

Repeat at each stage.



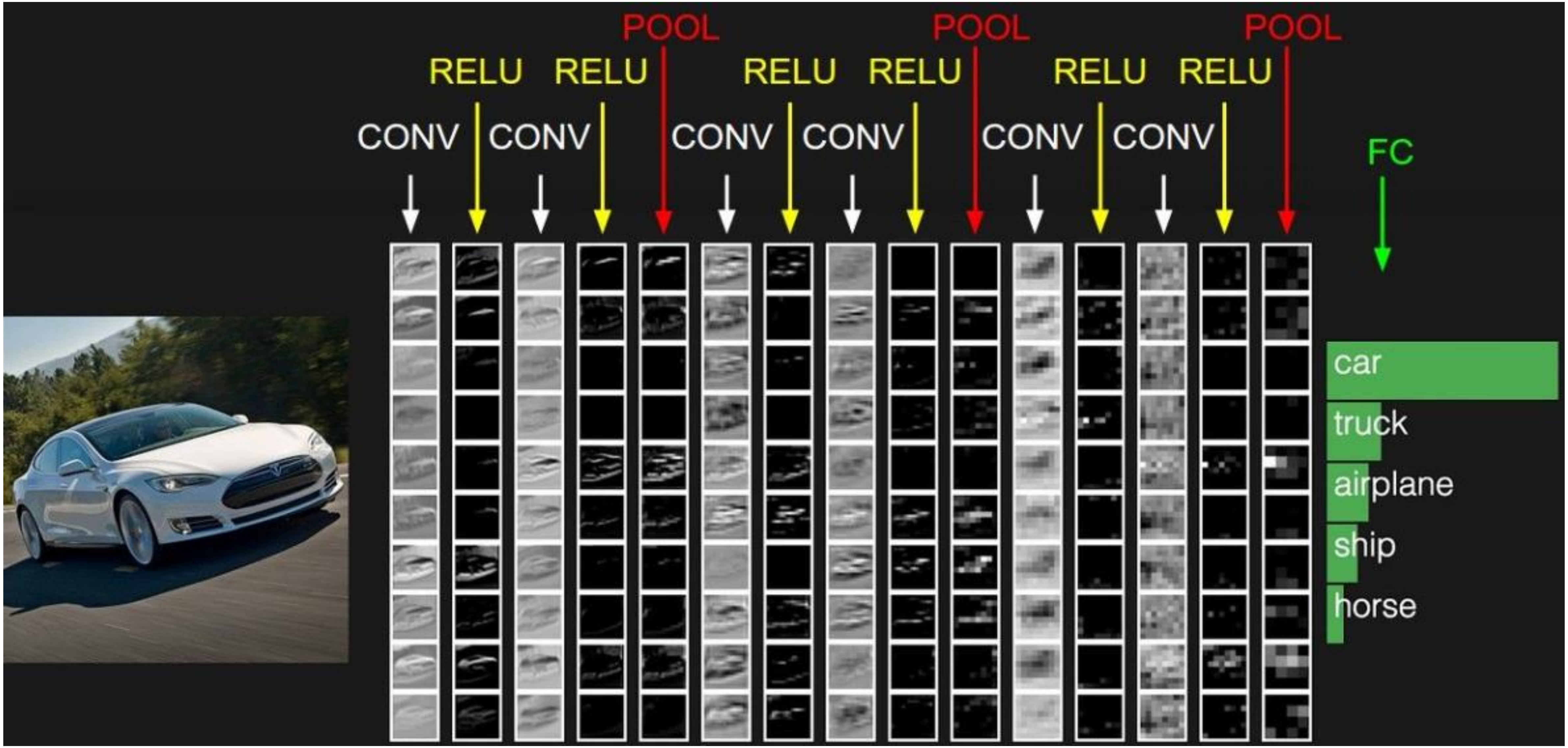
$$\sum_{i=1}^{5}\sum_{j=1}^{5} w_{i,j}x_{i+p,j+q} + b$$

Feature Maps / Activation maps : 28 x 28 x 4



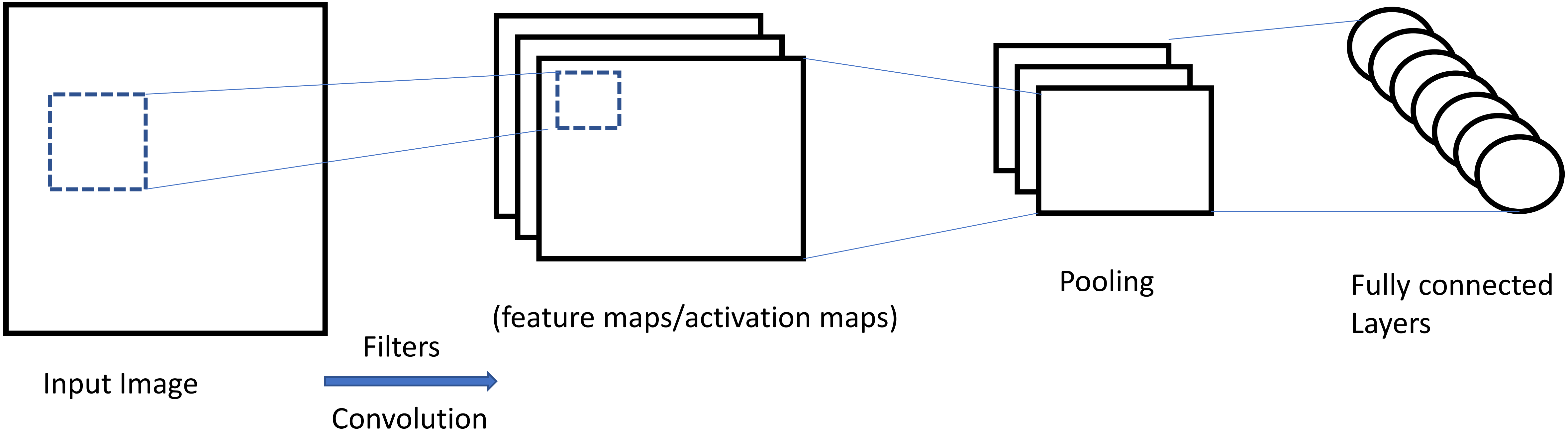
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Source: Dr. Fie Fie Li slides



Source: Prof. Fie Fie Li slides

CNNs for classification



- We discussed convolution operation and feature maps.
- Pooling:

Pooling

Pooling: Motivations

Down sampling:

- We want to reduce the resolution of images.
- The output should not depend on the dimensionality of the original image.

Invariance to translation:

In reality, objects hardly ever occur exactly at the same place.

- Detection should be invariant to translation to some extent.

Example: For instance, image with sharp feature and shifted by one pixel → detection result should not be vastly different from original image.

Pooling layers:

- reduce the sensitivity of Conv layer to location
- reduce the resolution through the processing pipeline.

Pooling layer : Max pooling or Average Pooling

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Max Pooling

Choose the maximum value in pooling window

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Average Pooling

Choose the average value in pooling window

Pooling layer : Max pooling or Average Pooling

0	3	1	6
2	1	0	2
-1	1		1
-1	0	-1	3

Max Pooling

Choose the maximum value in pooling window

3	

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Average Pooling

Choose the average value in pooling window

1.5	

Pooling layer : Max pooling or Average Pooling

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Max Pooling

Choose the maximum value in pooling window

3	6

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Average Pooling

Choose the average value in pooling window

1.5	2.25

Pooling layer : Max pooling or Average Pooling

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Max Pooling

Choose the maximum value in pooling window

3	6
1	

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Average Pooling

Choose the average value in pooling window

1.5	2.25
-0.25	

Pooling layer : Max pooling or Average Pooling

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Max Pooling

Choose the maximum value in pooling window

3	6
1	3

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Average Pooling

Choose the average value in pooling window

1.5	2.25
-0.25	1

Strides and padding also available for pooling.

In practice, pooling window size: 2 x 2 , stride = 2.

Note: Zero padding is NOT common for pooling layers.

Pooling

Pooling layers / Subsampling pixels does not change the object.

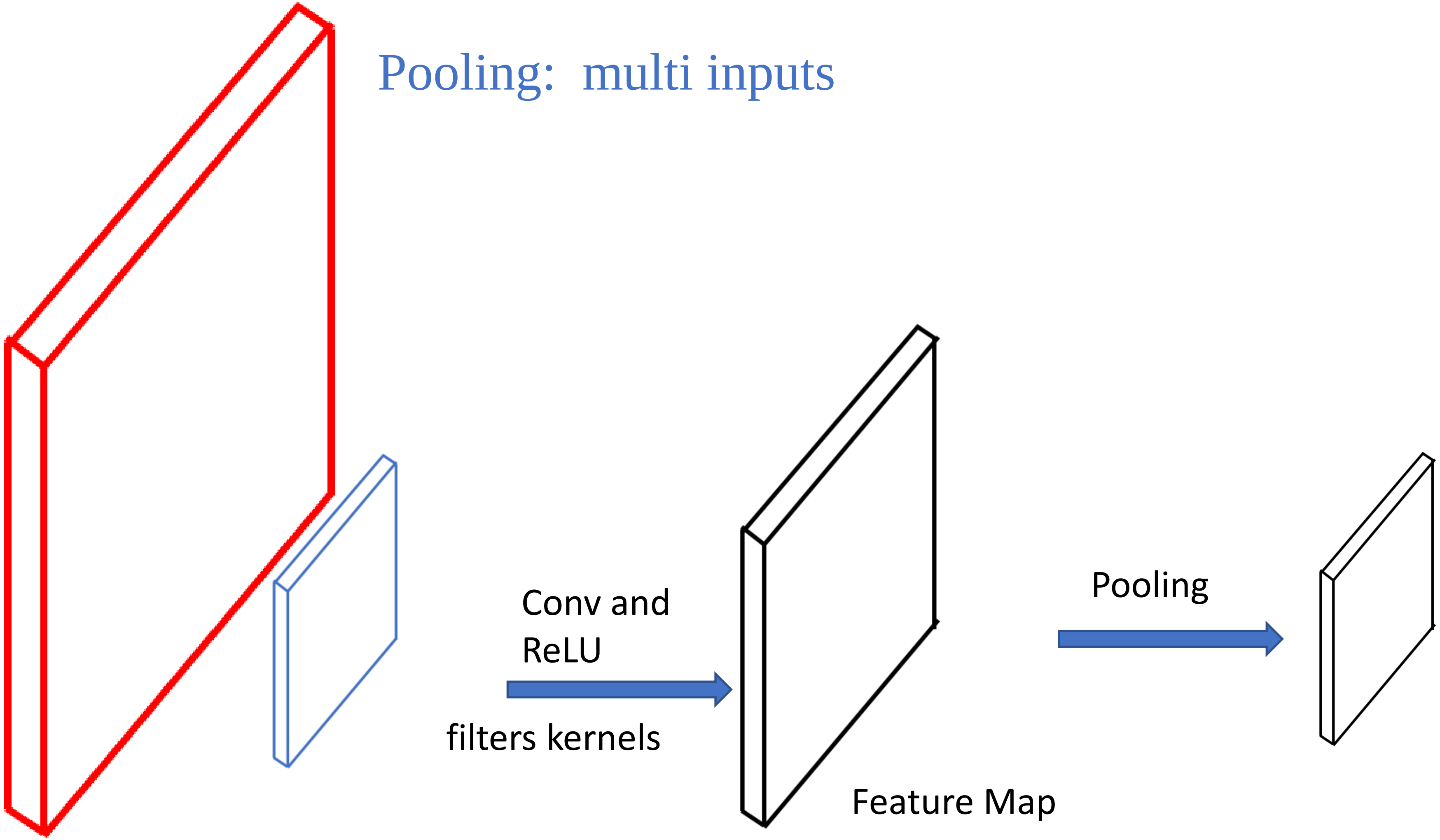
Changes the resolution , fewer parameters to characterize the image.

The subsampling layers reduce the spatial resolution of each feature map

By reducing the spatial resolution of the feature map, a certain degree of shift and distortion invariance is achieved.

Reduces the effect of noises and shift or distortion

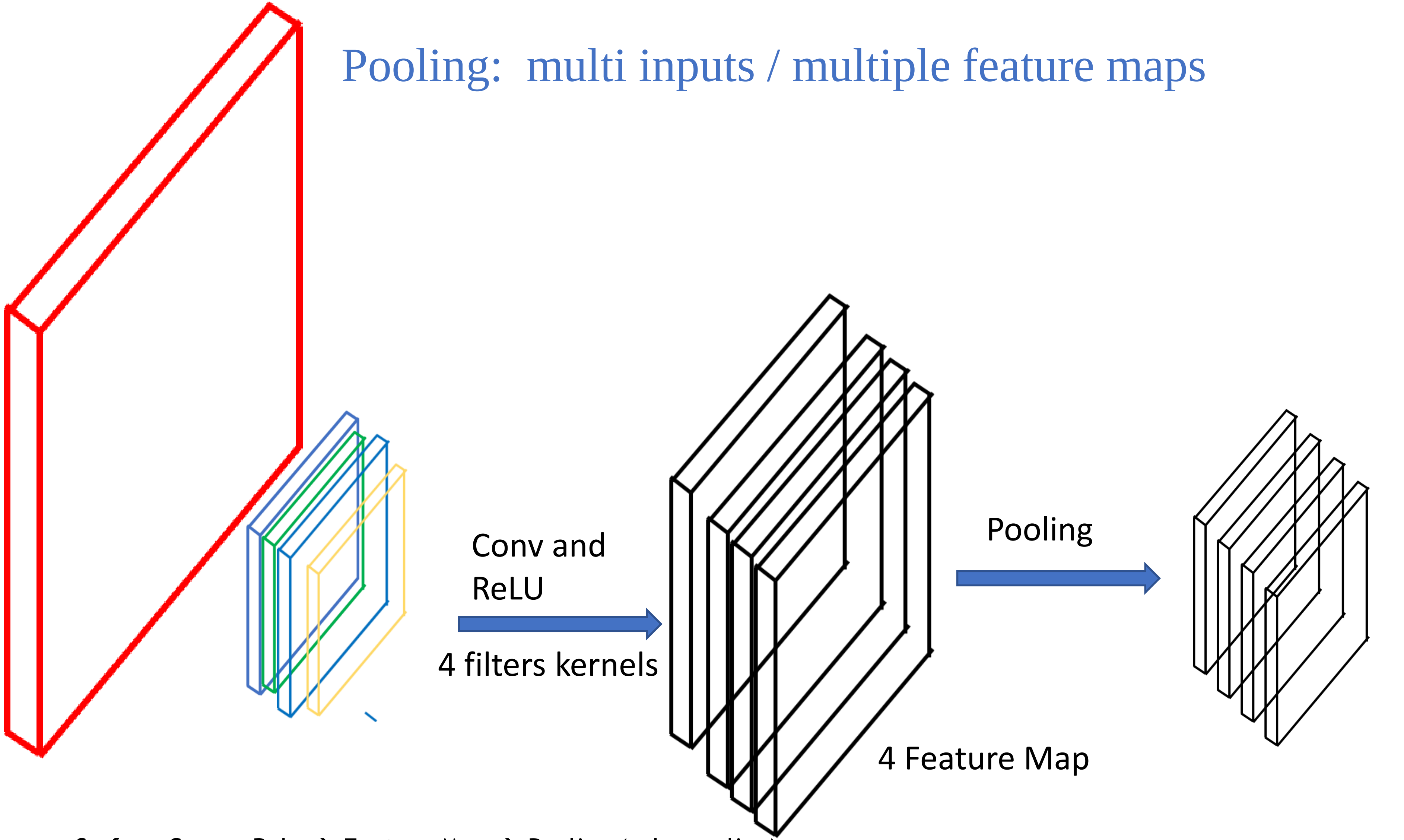
Pooling: multi inputs



So far: Conv + Relu → Feature Map → Pooling (subsampling)

When, multiple filters used:

Pooling: multi inputs / multiple feature maps

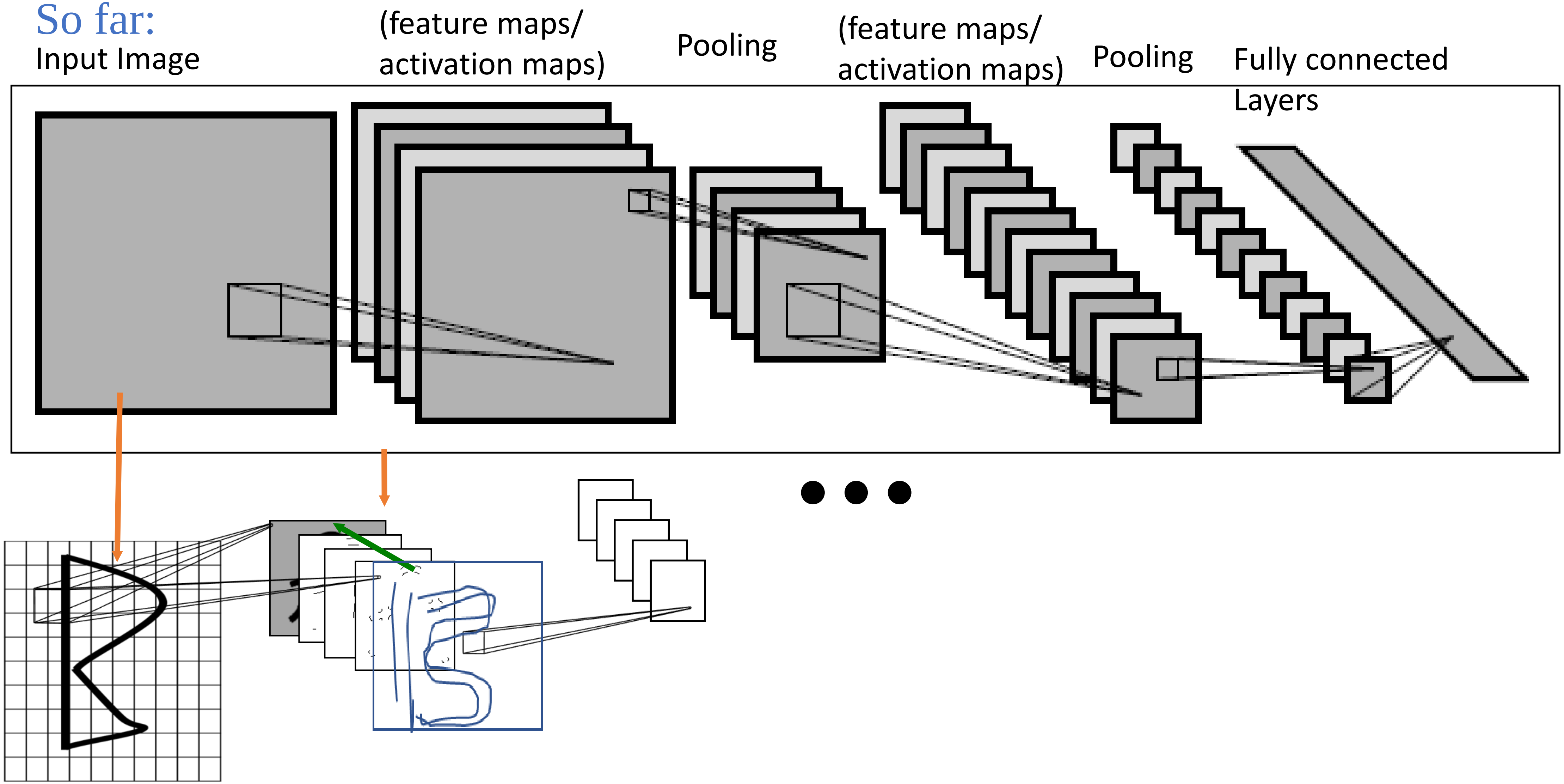


So far: Conv + Relu → Feature Map → Pooling (subsampling)

When, multiple filters used: Pooling done on each input feature map.

New set of images but smaller images.

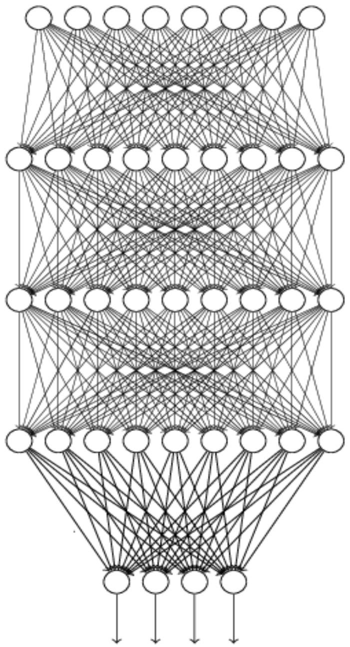
So far:


Input Image
(feature maps/
activation maps)
Pooling
(feature maps/
activation maps)
Pooling
Fully connected
Layers

Convolution → Max Pooling → Convolution → Max Pooling → Flattening

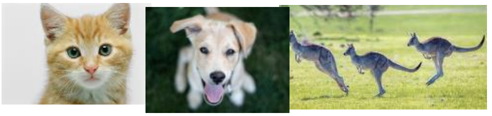
A new image

A new image



Fully Connected Feedforward network

cat , dog , kangaroo……



At each stage, a new image (reduced resolution) is obtained, ready for convolution.

The output structure is “flattened” to create a single long feature vector

The dense layer for the final classification.

Convolution → Max Pooling → Convolution → Max Pooling → Flattening

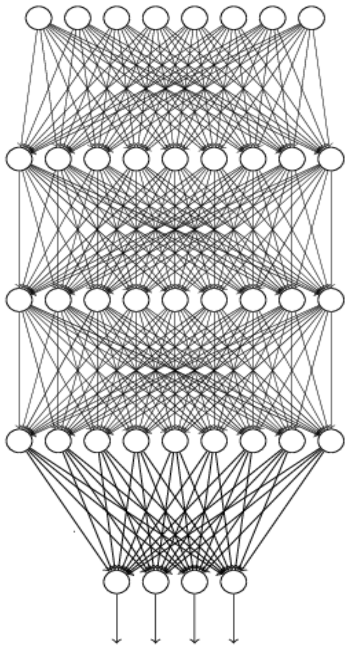
A new image

A new image

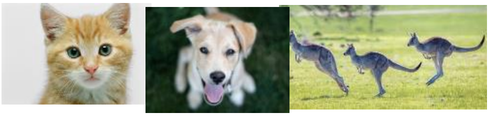
This can repeat many times

Pooling leads to subsampling

Fully Connected Feedforward network



cat , dog , kangaroo……



Flattening

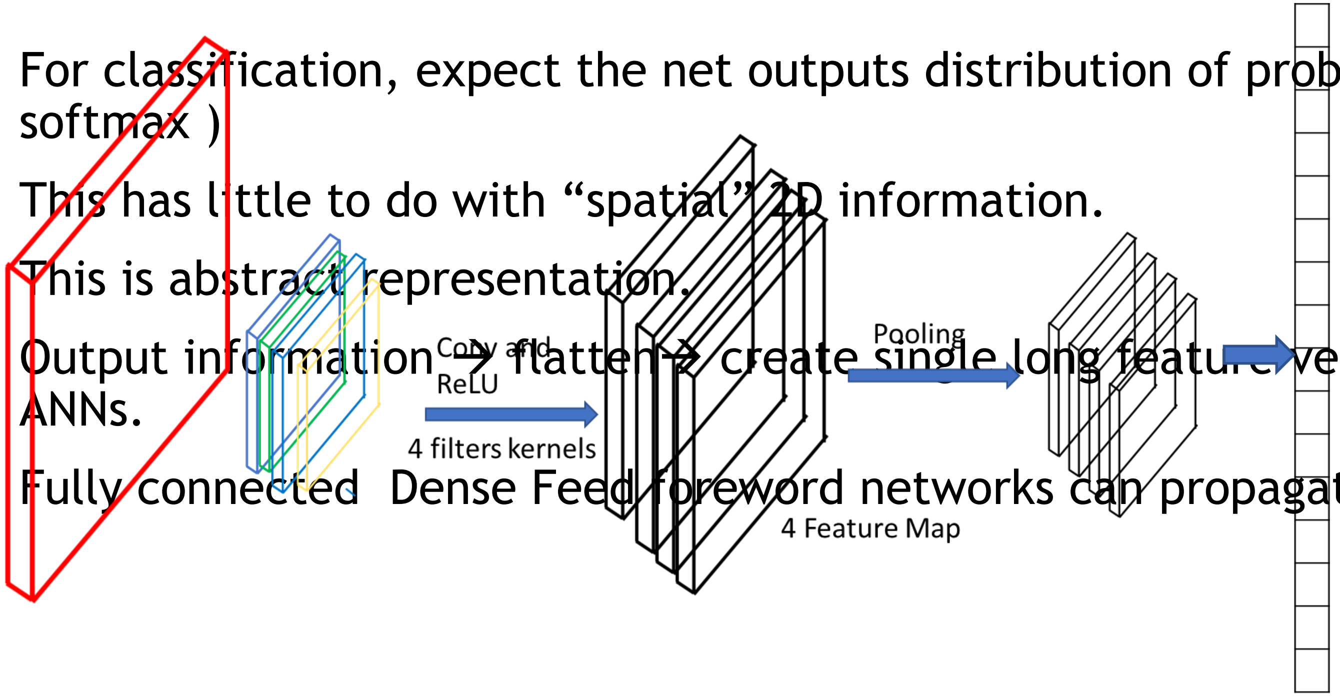
For classification, expect the net outputs distribution of probability of ea
softmax)

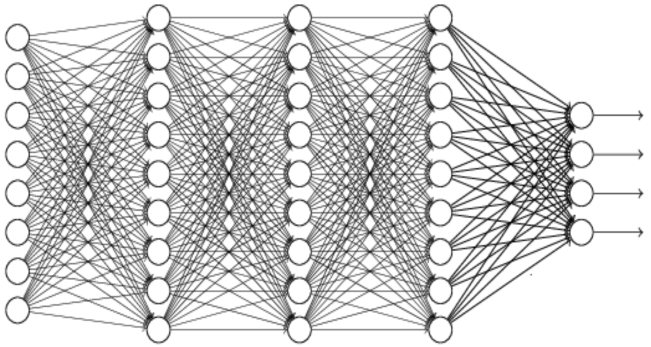
This has little to do with “spatial” 2D information.

This is abstract representation.

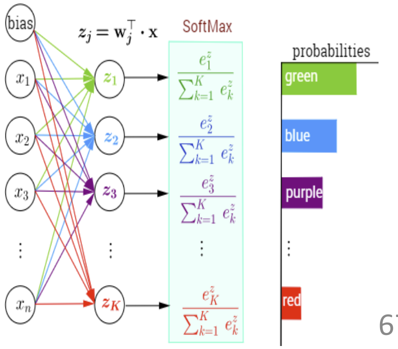
Output information → flatten→ create single long feature vector (like last lecture)→ feed to Dense ANNs.

Fully connected Dense Feed foreword networks can propagate this information.





Fully Connected
Feedforward network



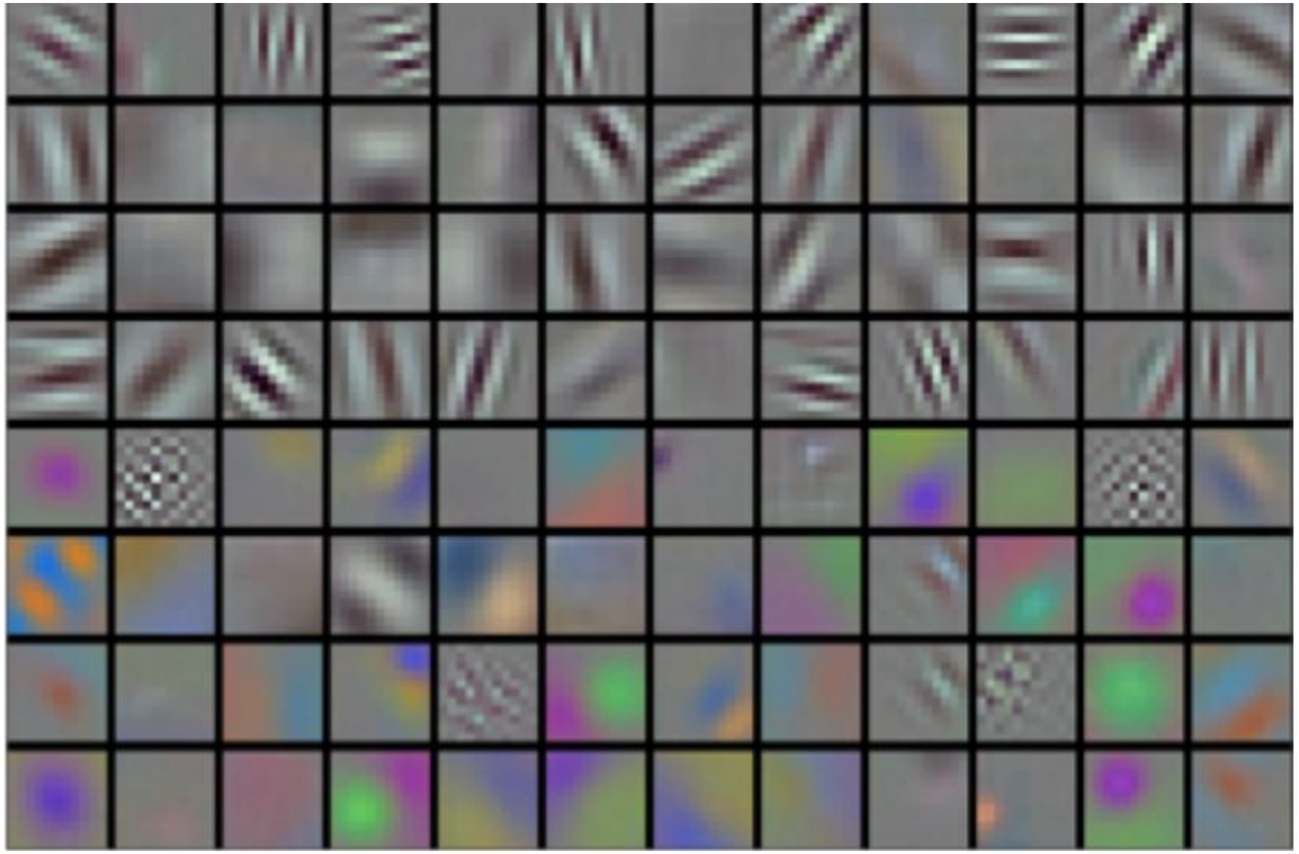
Demo: training on CIFAR-10 dataset

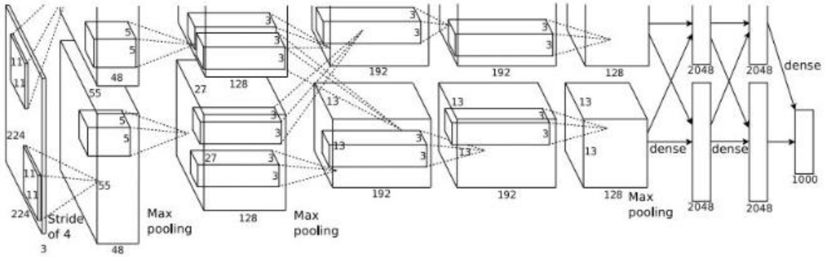
https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.htmlt

AlexNet: obtained above par state of art results on ImageNet challenge,

learnt good low level features,

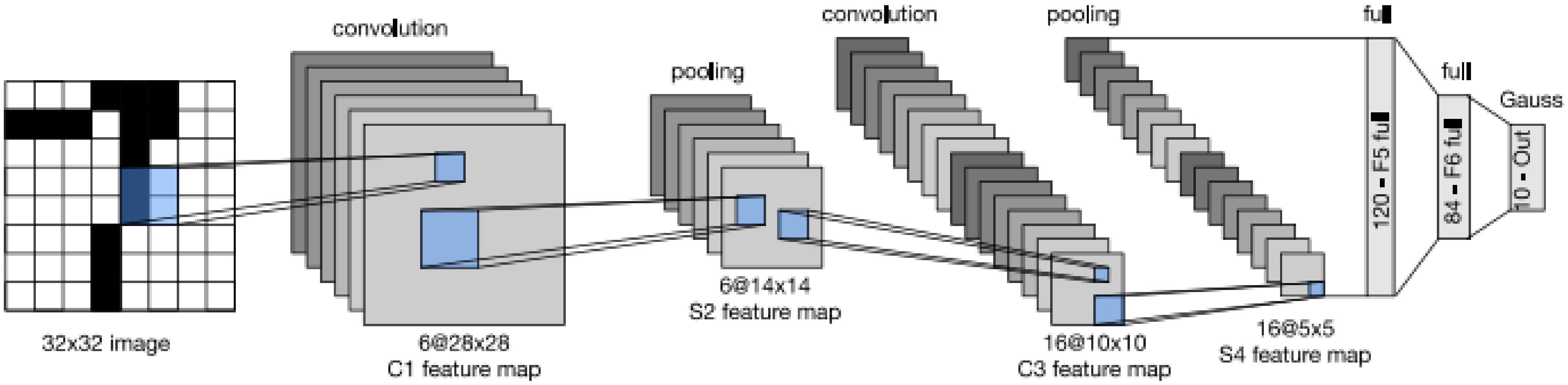
higher level features built upon these.





LeNet-5 (LeCun et al. 1998)

- state-of-the-art performance on hand digit recognition tasks.



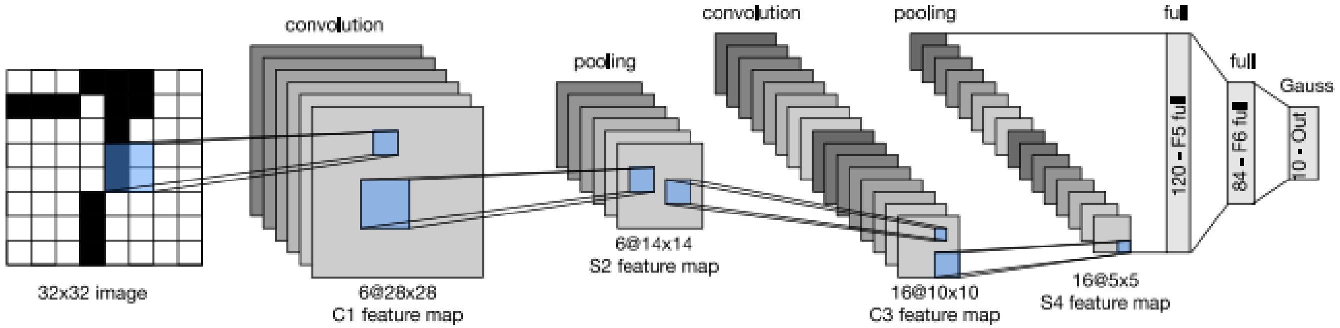
LeNet-5 (LeCun et al. 1998)

Advantages :

- convolution with learnable parameters (sharable parameters) ➔ effective way to extract similar features at multiple locations with few parameters .
- correlation with neighboring pixels (data) considered.
- optical character, fingerprint recognition…

Limitations:

- High computational burden: each pixel as separate input .
- Traditional activations functions: slow learning.



Stagnation of CNN : Early 2000

ML paradigm in 1990-1998:

- Typically such datasets were **hand generated** using very expensive sensors.
- Lacked richness, diversity ➔ insignificant improvement of performance (lack of complex training data, representations etc.
- Till 2012, feature representation had to be thought, or based on intuition.

CNNs:

- Backpropagation → not effective to reach global minima.
- Activation functions: Sigmoid function (variants)
  - vanishing gradient problem (exponential decay)
  - exploding gradient problem.

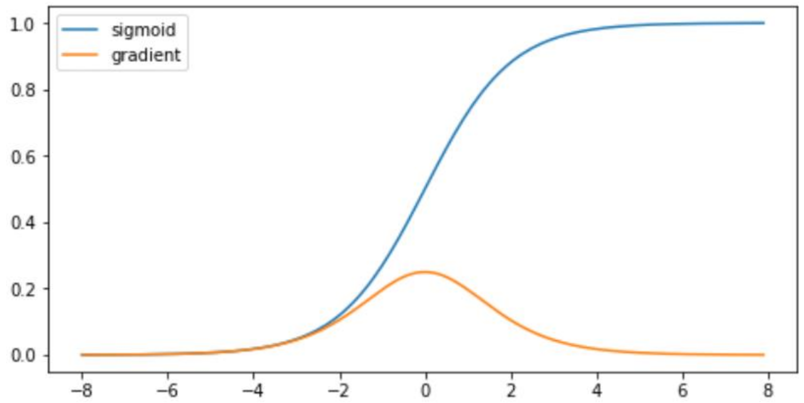


image credits (with permission): https://neurohive.io/

Stagnation of CNN : Early 2000

ML paradigm in 1990-1998:

- Typically such datasets were **hand generated** using very expensive sensors.
- Lacked richness, diversity ➔ insignificant improvement of performance (lack of complex training data, representations etc.
- Till 2012, feature representation had to be thought, or based on intuition.

CNNs:

- Backpropagation → not effective to reach global minima.
- Activation functions: sigmoid function (variants)
  - vanishing gradient problem (exponential decay)
  - exploding gradient problem (no efficient initialization methods)
- Little attention : object detection, classification/prediction of spatio-temporally comp~~lex data~~
- Limited computational resources (no GPUs)

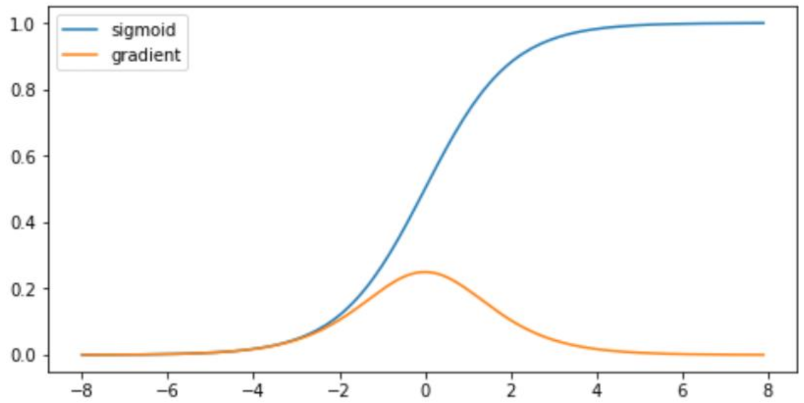
$$\frac{\partial J}{\partial W^{(1)}} = \text{prod}\left(\frac{\partial J}{\partial z}, \frac{\partial z}{\partial W^{(1)}}\right) = \frac{\partial J}{\partial z} x^T$$

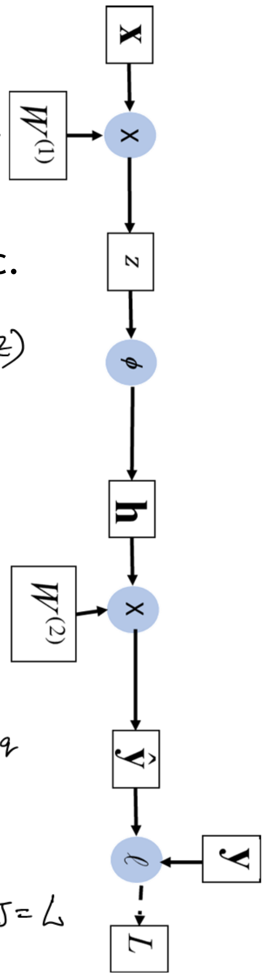
$$\frac{\partial J}{\partial z} = \text{prod}\left(\frac{\partial J}{\partial h}, \frac{\partial h}{\partial z}\right) = \frac{\partial J}{\partial h} \odot \phi'(z)$$

$$\frac{\partial J}{\partial h} = \text{prod}\left(\frac{\partial J}{\partial \hat{y}}, \frac{\partial \hat{y}}{\partial h}\right) = W^{(2)T} \frac{\partial J}{\partial \hat{y}}$$

$$\frac{\partial J}{\partial \hat{y}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \hat{y}}\right) = \frac{\partial L}{\partial \hat{y}} \in \mathbb{R}^q$$

$$\frac{\partial J}{\partial L} = 1 \quad \text{as} \quad J = L$$



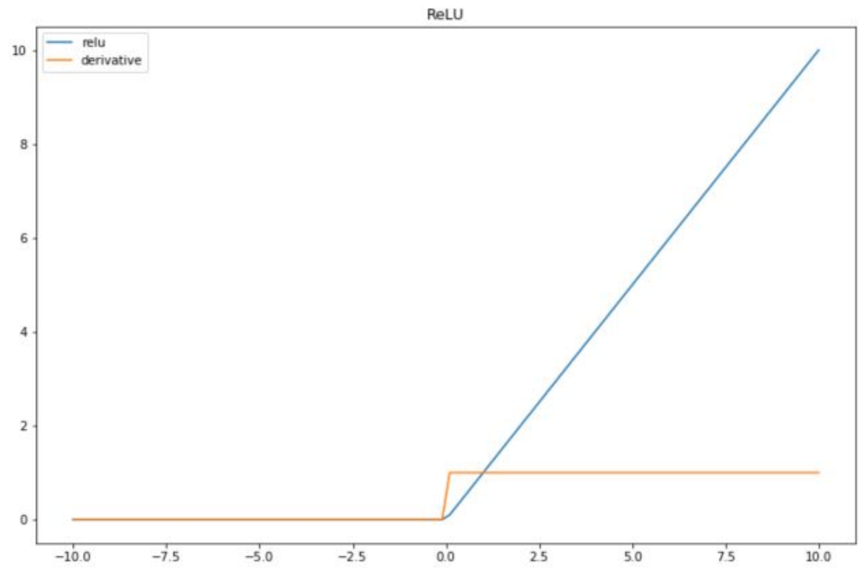


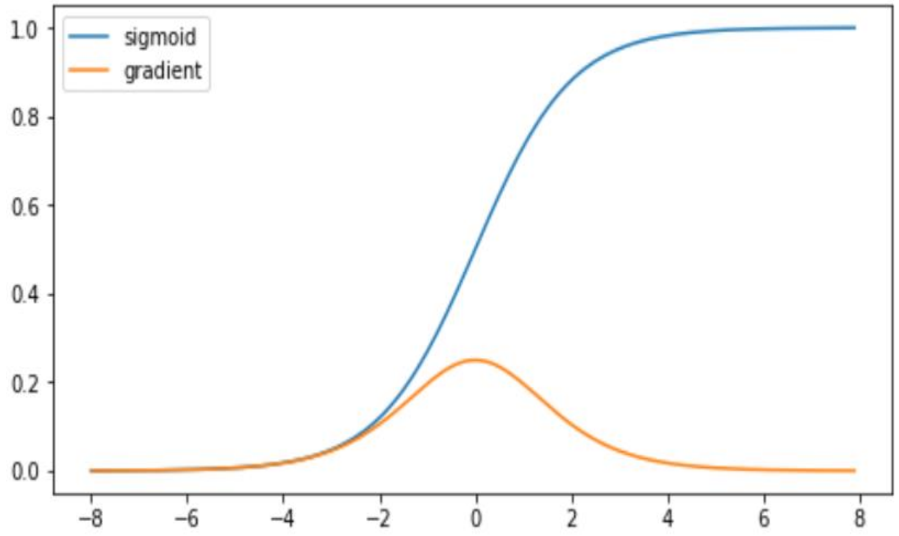
101

Revival of CNNs: 2006-2011

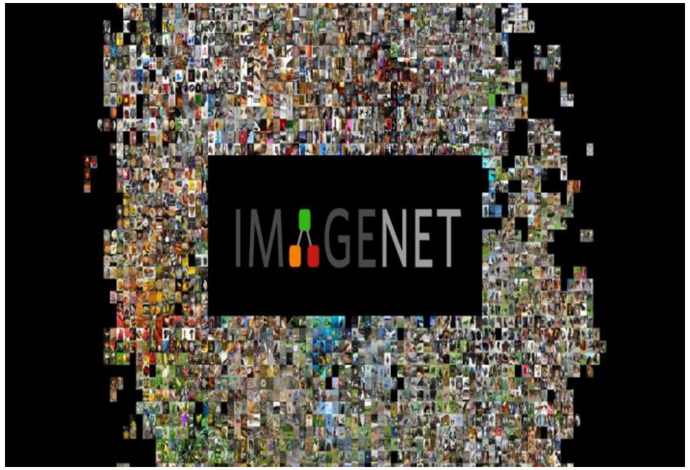
- Efficient initialization techniques:
  - greedy layer-wise pre-training (Hinton et al. 2006)
  - unsupervised/supervised training-based pre-training
  - Xavier initialization (Glorot and Bengio, 2010)
- Use of Non-saturating Activation Functions : ReLu (Glorot and Bengio, 2010)
- Max-pooling > Sub-sampling (Ranzato et al, 2007) → learnt better invariant features.

- Late 2006: GPUs for training CNNs.
- 2007: NVIDIA → CUDA programming → harness parallel processing power of GPUs





Revival of CNNs: 2006-2011



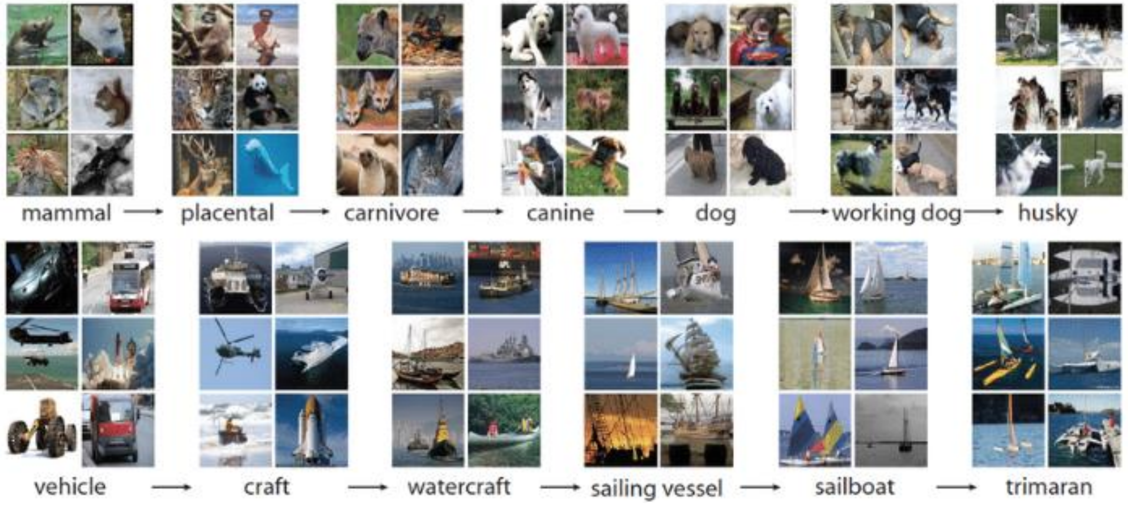
- Efficient initialization techniques:
  - greedy layer-wise pre-training (Hinton et al. 2006)
  - unsupervised/supervised training-based pre-training
  - Xavier initialization (Glorot and Bengio, 2010)
- Use of Non-saturating Activation Functions : ReLu (Glorot and Bengio, 2010)
- Max-pooling > Sub-sampling (Ranzato et al, 2007) → learnt better invariant features.

- Late 2006: GPUs for training CNNs.
- 2007: NVIDIA → CUDA programming → harness parallel processing power of GPUs

- 2010: Dr. Fei-Fei Li group (Stanford) → ImageNet platform

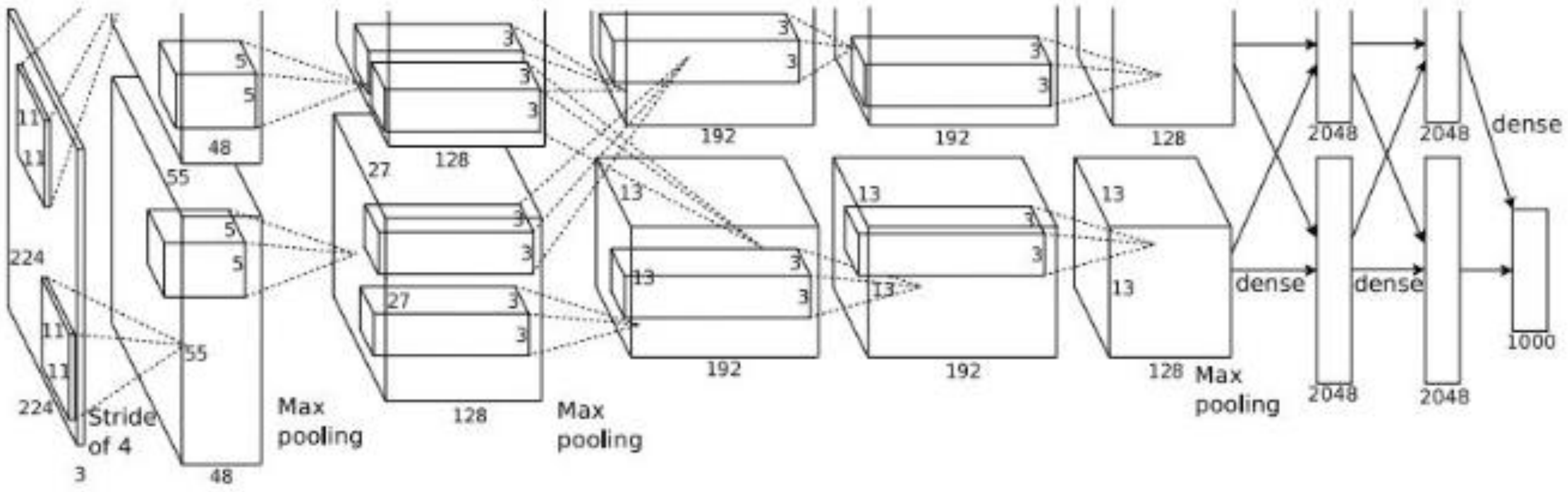
today ImageNet → 15 millions, large number categories and classes (target labels).

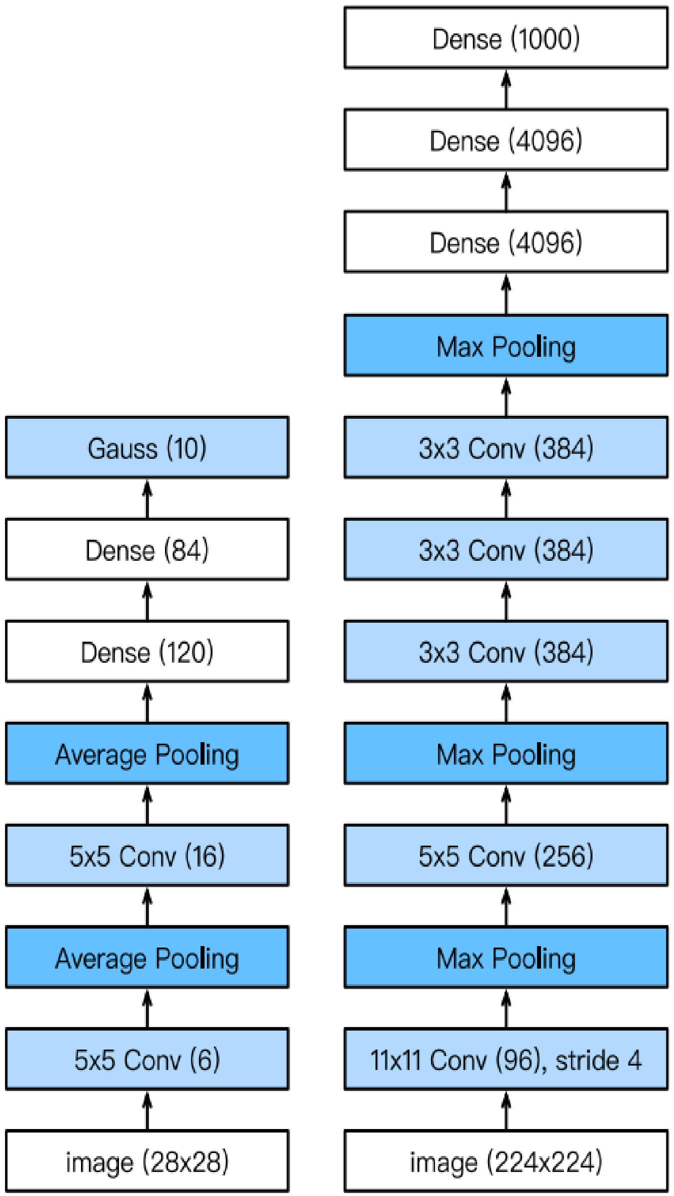
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (2010-2017)



AlexNet (Krizhevsky et al. 2012)

- Considered first “modern deep architecture”
- Deeper than LeNet-5: from 5 to 8 layers

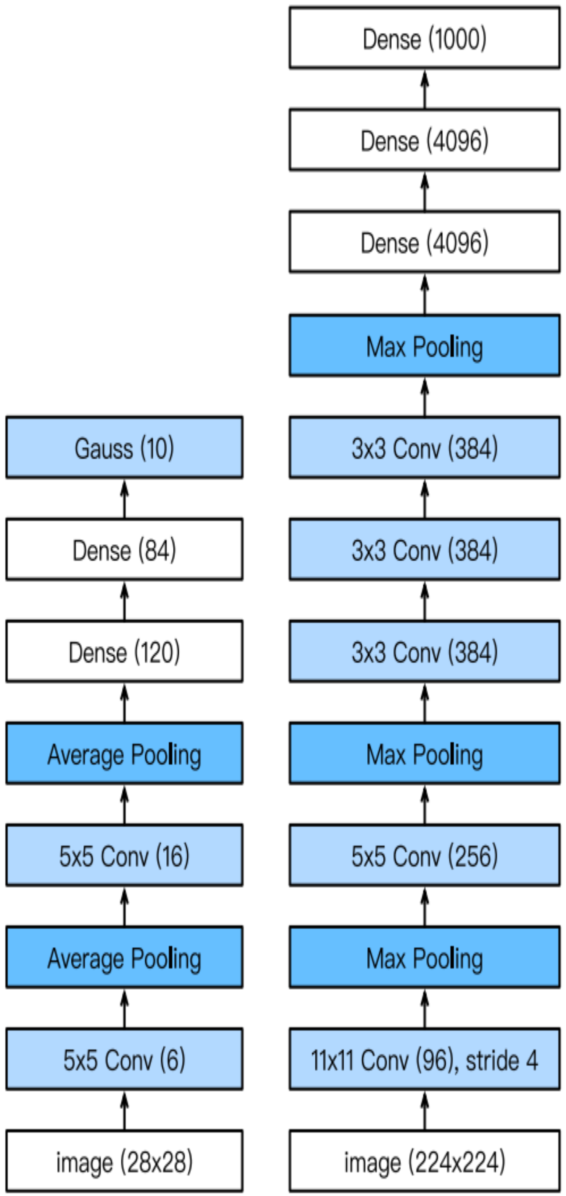




LeNet AlexNet

AlexNet (Krizhevsky et al. 2012)

- Considered first “modern deep architecture”, deeper than LeNet-5: from 5 to 8 layers,
- 60 Million parameters
- Depth increases overfitting: learning algo: skips some transformational units.
- ReLU : improve convergence ➔ reduce vanishing gradient problem.
- Heavy data augmentation for training: flipping, clipping, color change etc.
- Use of multiple GPUs for training : trained in parallel on two NVIDIA GTX 580
- Use of large filter (11X11, 5X5) as initial layers
- Overlapping pooling layers: (0.5% reduction in overfitting).
- Other adjustments:
  - Dropout for regularization
  - SGD Momentum



LeNet AlexNet

AlexNet (Krizhevsky et al. 2012)

- Considered first "modern deep architecture", deeper than LeNet: from 5 to 8 layers,
- 60 Million parameters
- Depth increases overfitting: learning algo: skips some transformational units.
- ReLU : improve convergence ➔ reduce vanishing gradient problem.
- Heavy data augmentation for training: flipping, clipping, color change etc.
- Use of multiple GPUs for training : trained in parallel on two NVIDIA GTX 580
- Use of large filter (11X11, 5X5) as initial layers
- Overlapping pooling layers: (0.5% reduction in overfitting).
- Other adjustments:
  - Dropout for regularization: 0.5
  - SGD Momentum
- Winner of ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) 2012
  - Recognize off-center objects.
- Beginning of Modern era of Deep learning: SOTA
  - Deep Learning to new fields: medical imaging, data extraction, end to end learning…
- Missing ➔ A template for Deep NN design.

Visual Geometry Group or VGG (Simonyan and Zisserman 2015)

19 layers deeper compared to AlexNet

Addition:

- Studied the relation of depth with the representational capacity of the network.
- Replaced: large kernel-sized with small receptive field (multiple 3×3 kernels).
- All hidden layers: ReLu activation.
- Suggested that small size filters can improve the performance of the CNNs

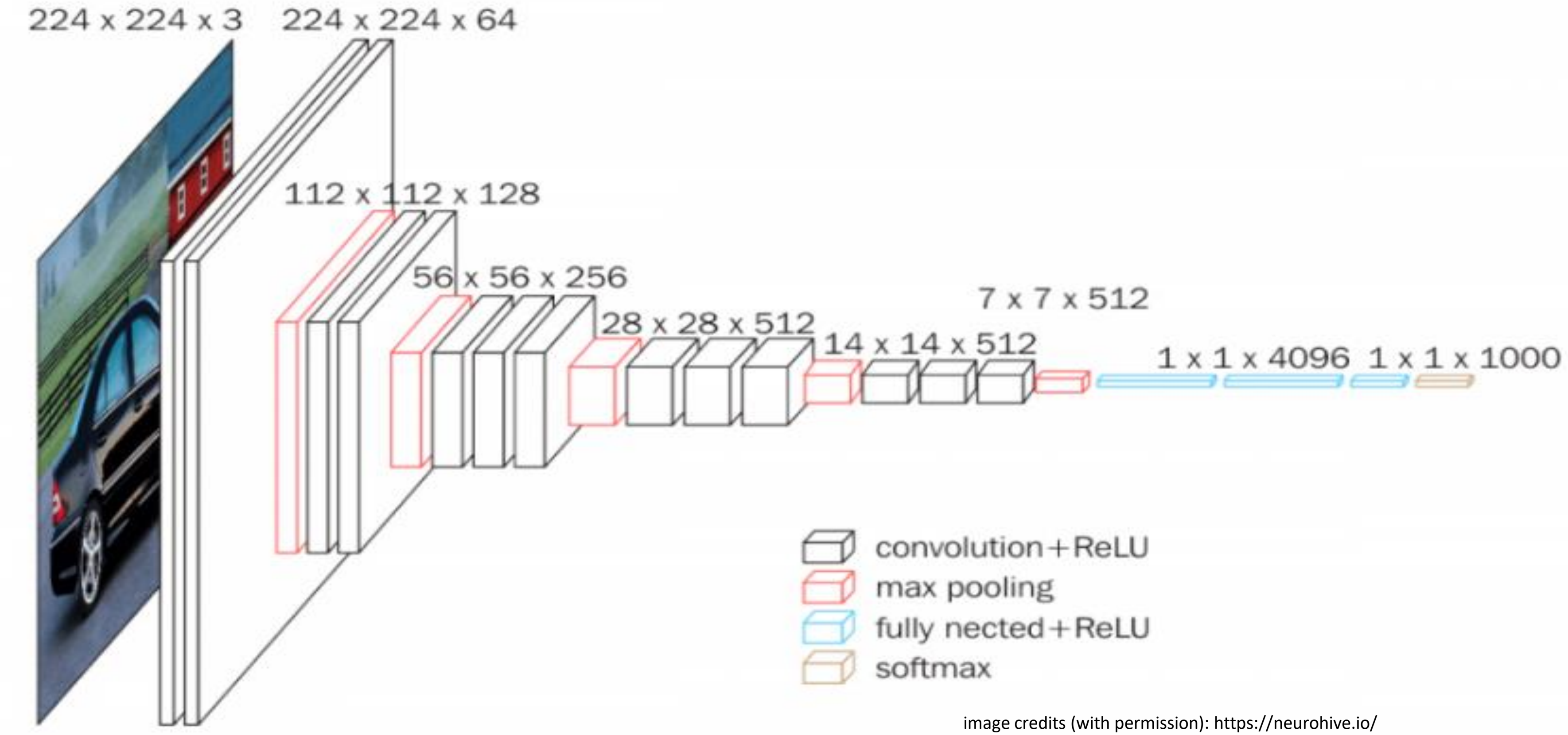


image credits (with permission): https://neurohive.io/

Visual Geometry Group or VGG (Simonyan and Zisserman 2015)

Dataset:

- ImageNet , inputs down-sampled → 256×256

Architecture:

- Image passed through a stack of convolutional (conv.) layers, with filters → with a very small receptive field: 3×3
    - The convolution stride is fixed to 1 pixel
    - the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers.
    - Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling).
    - Max-pooling is performed over a 2×2 pixel window, with stride 2.
    - Complexity regulation: 1X1 convolutions between conv layers (learn linear combination of resultant feature maps)
    - Followed by: Three Fully-Connected (FC) layers : 4096, 4096,1000 (for ILSVRC classification)
    - The final layer is the soft-max layer.
    - The configuration of the fully connected layers is the same in all networks.

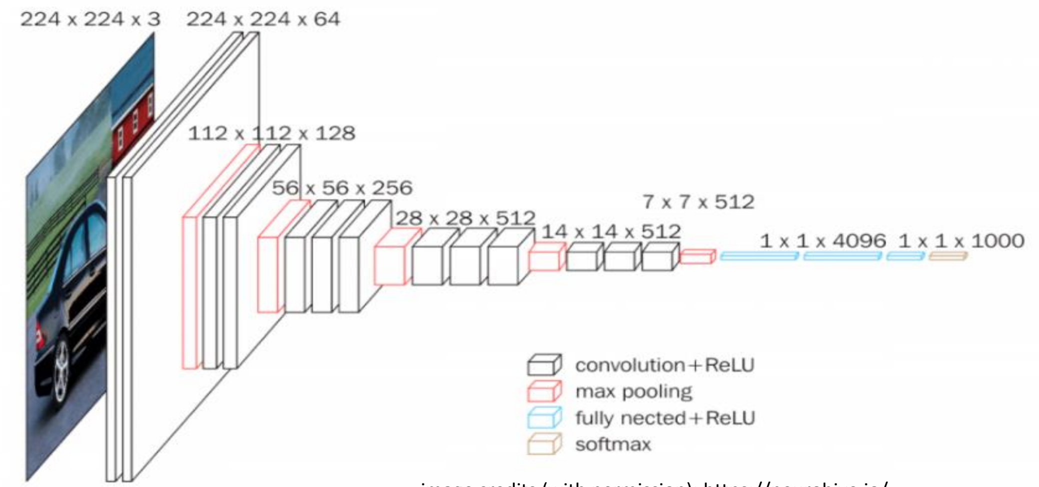


image credits (with permission): https://neurohive.io/

Visual Geometry Group or VGG (Simonyan and Zisserman 2015)

19 layers deeper compared to AlexNet

Addition:

- Studied the relation of depth with the representational capacity of the network.
- Replaced: large kernel-sized with small receptive field (multiple 3×3 kernels).
- All hidden layers: ReLu activation.

Advantages:

- Significantly outperformed previous generation models with respect to classification accuracy.
- Representation depth is beneficial for the classification accuracy.
- Suggested that small size filters can improve the performance of the CNNs.
- Several layers of deep and narrow convolutions (i.e., 3×3) were more effective than fewer layers of wider convolutions.
- 2nd Place 2014-ILSVRC

Set the trend: smaller sized filters.

Limitations:

- Very slow to train (For example: VGG16 was trained for weeks , NVIDIA Titan Black GPU’s)
- Large no pf parameters 138 million parameters
- Heavy architecture → 533MB

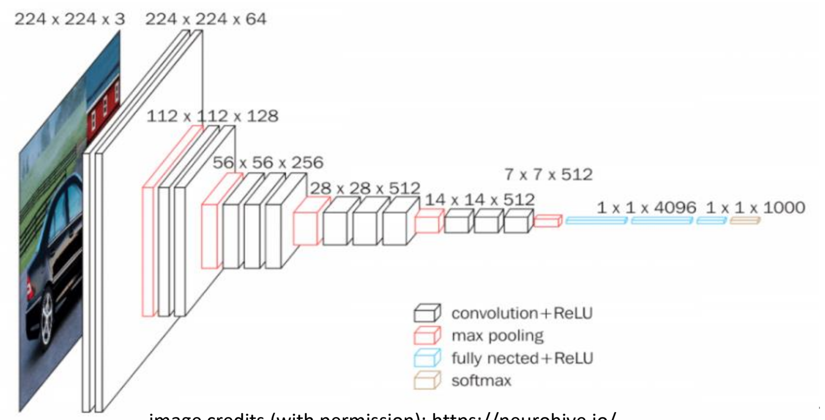
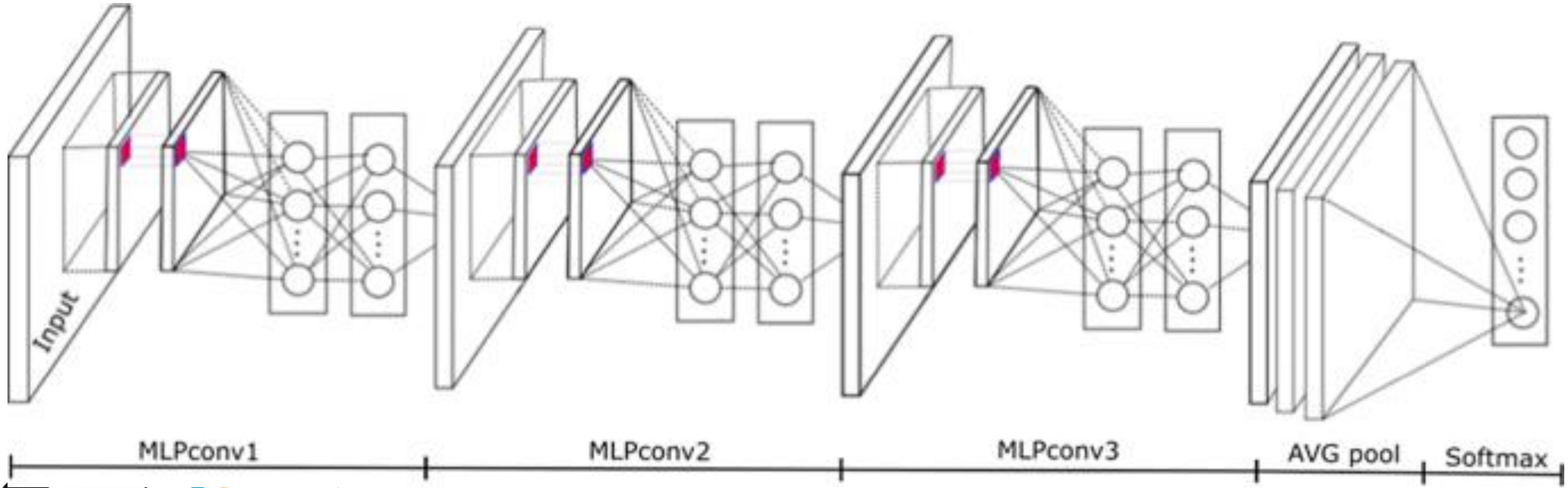


image credits (with permission): https://neurohive.io/

Network in Network (NiN) (Lin et al., 2013)

- Intuition:
  - to use an MLP on the channels for each pixel separately.
  - Apply a fully-connected layer at each pixel location (for each height and width).



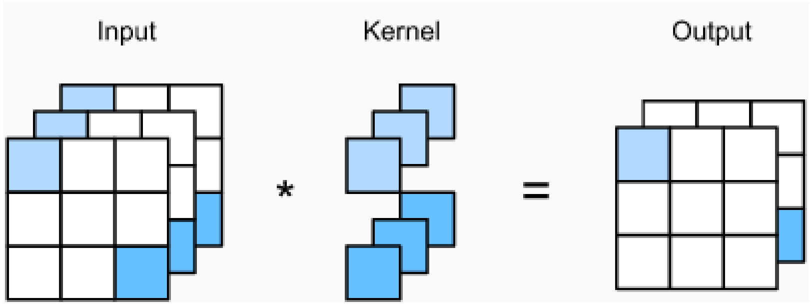
Source: Alaeddine, H., Jihene, M. Deep network in network. *Neural Comput & Applic* (2020).

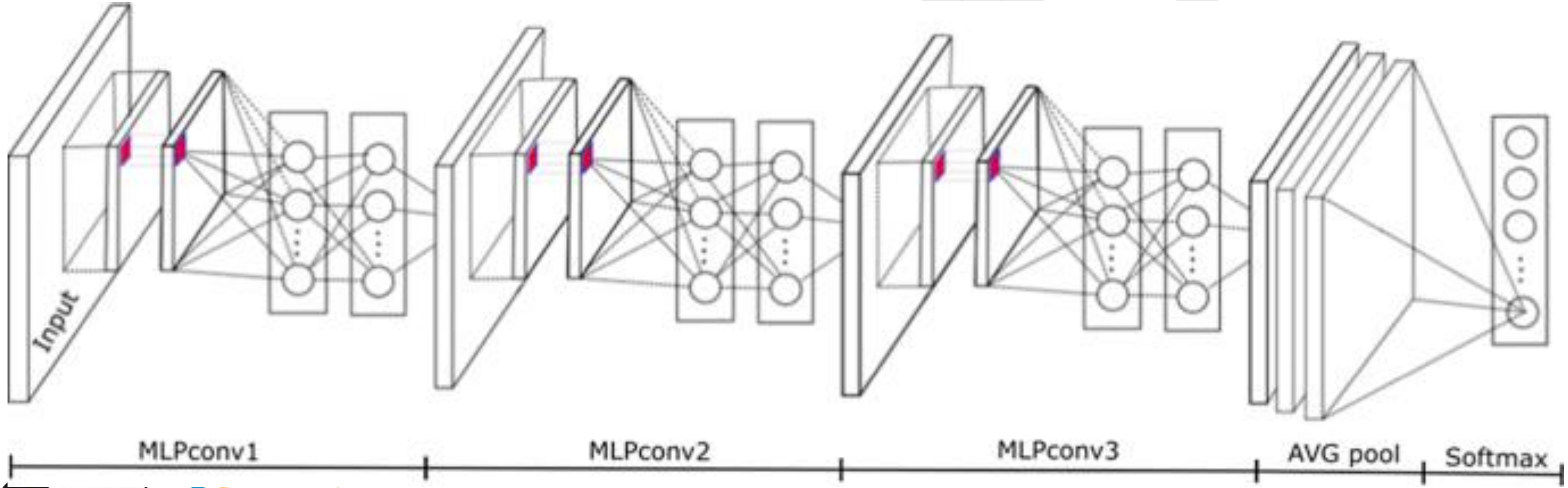
Network in Network (NiN) (Lin et al., 2013)

- Intuition:
    - to use an MLP on the channels for each pixel separately.
    - Apply a fully-connected layer at each pixel location (for each height and width).
    - If we tie the weights across each spatial location becomes ➔ 1X1 convolution layer.

    or

    fully-connected layer acting independently on each pixel location

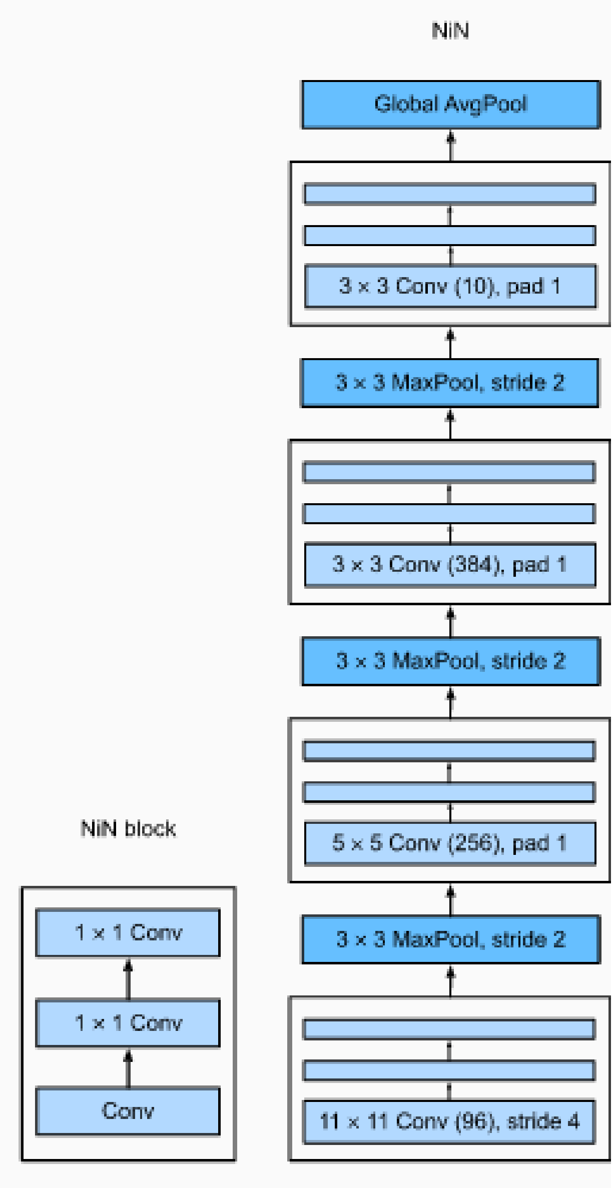




Source: Alaeddine, H., Jihene, M. Deep network in network. *Neural Comput & Applic* (2020).

Network in Network (NiN) (Lin et al. 2013)

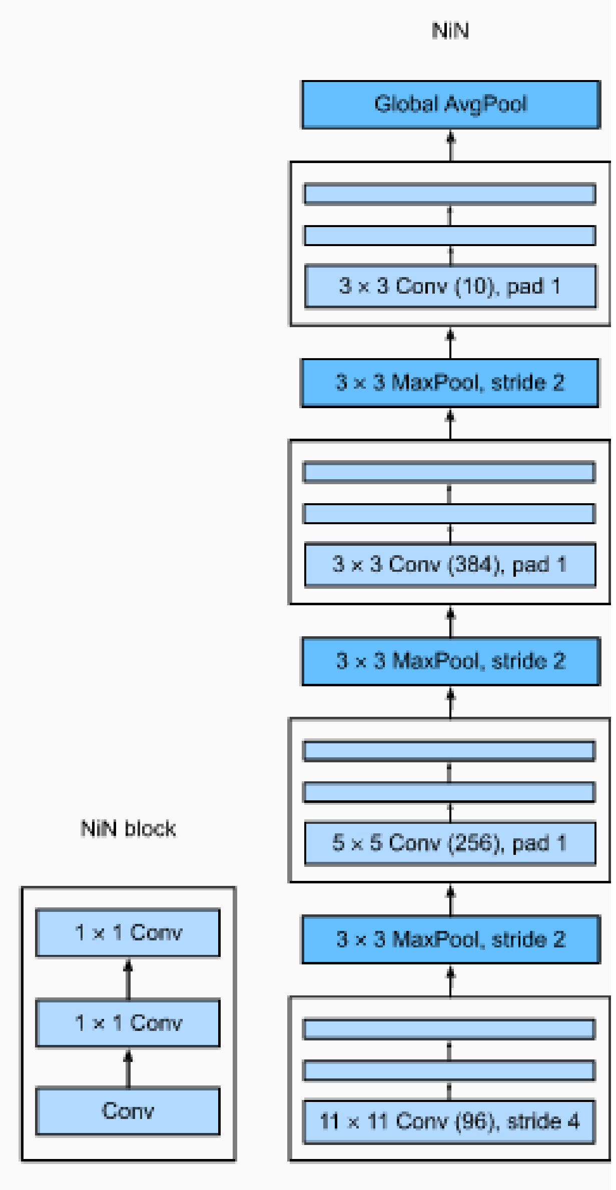
- Architecture:
  - inspired from AlexNet.
  - Convolutional layers: 11×11, 5×5, and 3×3
    - followed by two 1×1 convolutional layers that act as per-pixel fully-connected layers with ReLU activations
    - Each NiN block is followed by a maximum pooling layer (stride 2, window shape of 3×3).
  - The convolution window shape of the first layer is typically set by the user.
  - Output: number of output channels equal to the number of label classes, followed by a *global* average pooling layer.
  - Avoids fully-connected layers totally (against AlexNet, LeNet…)
- Advantages:
  - 1X1 convolutions ➔ allow for more per-pixel nonlinearity within convolutional stack.
  - NiN removes the fully-connected layers and replaces them with global average pooling.
    - Removing fully-connected layers reduces overfitting.
  - NiN has dramatically less parameters.



Source: Dive into Deep learning book

Network in Network (NiN) (Lin et al. 2013)

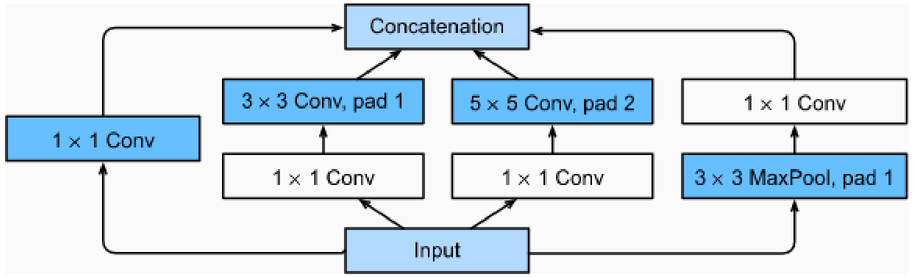
- Architecture:
  - inspired from AlexNet.
  - Convolutional layers: 11×11, 5×5, and 3×3
    - followed by two 1×1 convolutional layers that act as per-pixel fully-connected layers with ReLU activations
    - Each NiN block is followed by a maximum pooling layer (stride 2, window shape of 3×3).
  - The convolution window shape of the first layer is typically set by the user.
  - Output: number of output channels equal to the number of label classes, followed by a *global* average pooling layer.
  - Avoids fully-connected layers totally (against AlexNet, LeNet…)
- Advantages:
  - 1X1 convolutions ➔ allow for more per-pixel nonlinearity within convolutional stack.
  - NiN removes the fully-connected layers and replaces them with global average pooling.
    - Removing fully-connected layers reduces overfitting.
  - NiN has dramatically less parameters.



Source: Dive into Deep learning book

GoogLeNet (Szegedy et al., 2015)

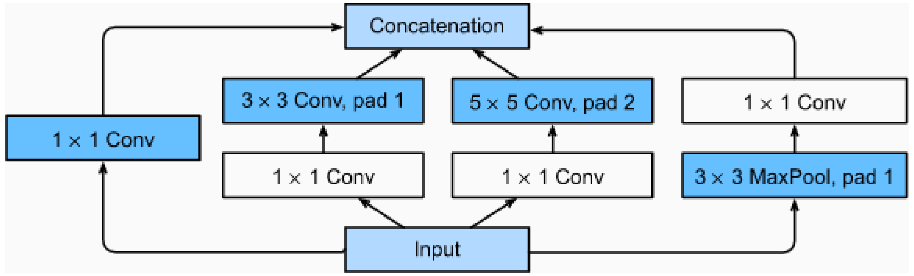
- Winner of 2014 ILSVRC
- One focus: Which sized convolution kernels are best (1X1, 3X3, 11X11 …) ?
- Introduced *Inception* block:
    - incorporates multi-scale convolutional transformations using split, transform and merge idea.
    - encapsulates filters of different sizes (1x1, 3x3, and 5x5)
    - captures spatial information at different scales: fine and coarse grain level.



Source: Dive into Deep learning book

GoogLeNet (Szegedy et al., 2015)

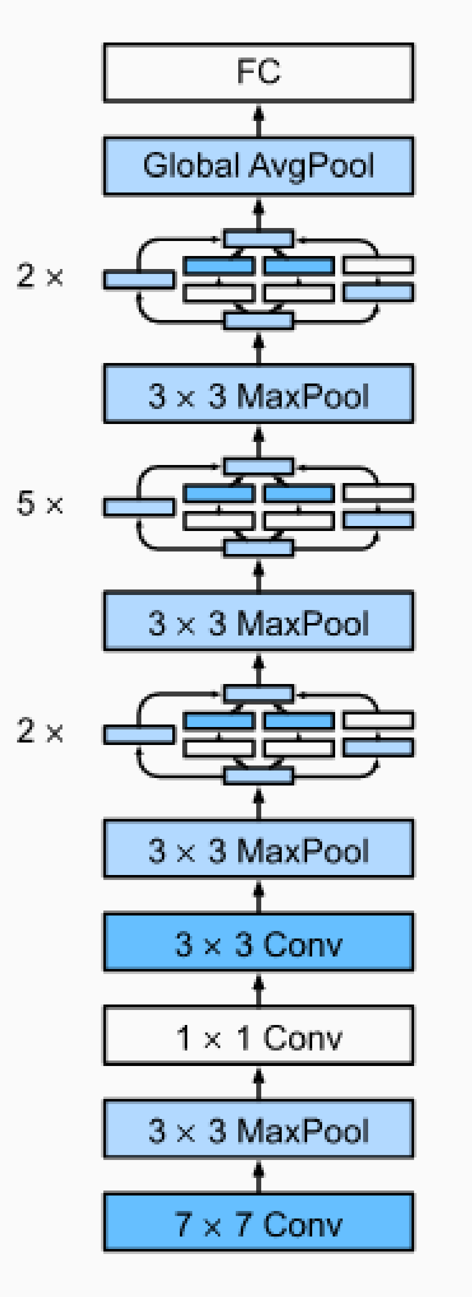
- Winner of 2014 ILSVRC
- One focus: Which sized convolution kernels are best (1X1, 3X3, 11X11 …) ?
- Introduced *Inception* block:
  - incorporates multi-scale convolutional transformations using split, transform and merge idea.
  - encapsulates filters of different sizes (1x1, 3x3, and 5x5)
  - captures spatial information at different scales: fine and coarse grain level.
  - computation regularization ➔ adding a bottleneck layer of 1x1 convolutional filter, before employing large size kernels.



Source: Dive into Deep learning book

GoogLeNet (Szegedy et al., 2015) or Inception V1

- Winner of 2014 ILSVRC
- One of the focus: Which sized convolution kernels are best (1X1, 3X3, 11X11 …) ?
- Introduced *Inception* block:
  - Incorporates multi-scale convolutional transformations using **split, transform and merge idea.**
  - Encapsulates filters of different sizes (1x1, 3x3, and 5x5)
  - Captures spatial information at different scales: fine and coarse grain level.
  - Computation regularization ➔ adding a bottleneck layer of 1x1 convolutional filter, before employing large size kernels.
- Advantages:
  - Density reduced → use of global average pooling at the last layer and NOT instead of using a fully connected layer
  - Significant decrease in parameters: from 138 Million to 4 Million parameters.
  - Other novelties:
    - Batch Normalization
    - RmsProp as optimizer,…

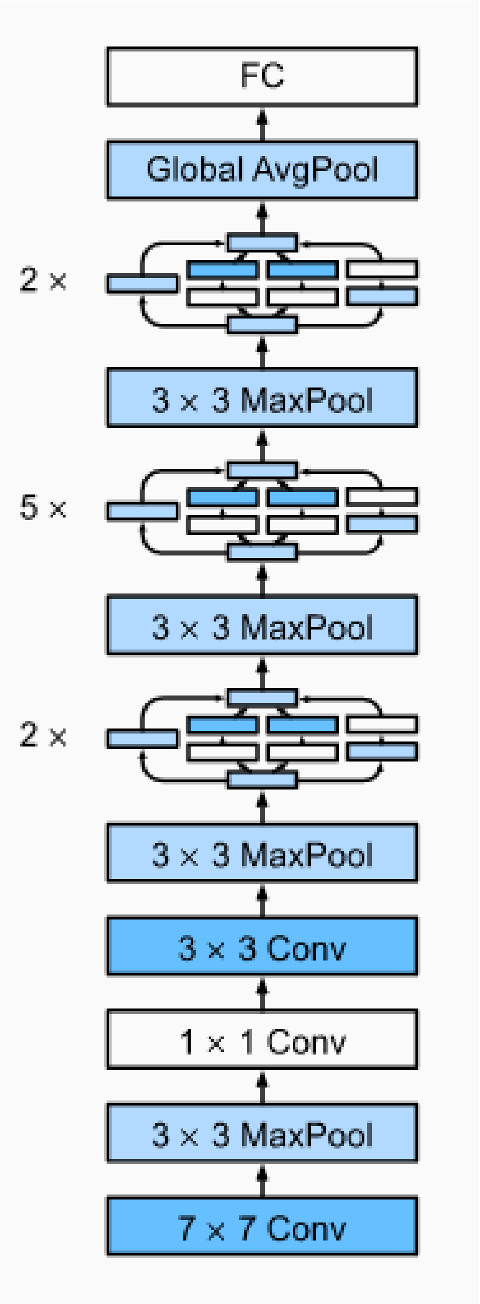


POLYTECH NANCY · UNIVERSITÉ DE LORRAINE · CRAN

Source: Dive into Deep learning book

GoogLeNet (Szegedy et al., 2015) or Inception V1

- Winner of 2014 ILSVRC
- One of the focus: Which sized convolution kernels are best (1X1, 3X3, 11X11 …) ?
- Introduced *Inception* block:
  - Incorporates multi-scale convolutional transformations using **split, transform and merge idea.**
  - Encapsulates filters of different sizes (1x1, 3x3, and 5x5)
  - Captures spatial information at different scales: fine and coarse grain level.
  - Computation regularization ➔ adding a bottleneck layer of 1x1 convolutional filter, before employing large size kernels.
- Advantages:
  - Density reduced → use of global average pooling at the last layer and NOT instead of using a fully connected layer
  - Significant decrease in parameters: from 138 Million to 4 Million parameters.
  - Other novelties:
    - Batch Normalization
    - RmsProp as optimizer,…
- Limitations:
  - heterogeneous topology that needs to be customized from module to module
  - representation bottleneck that drastically reduces the feature space
  - in the next layer and thus sometimes may lead to loss of useful information.
- Variants: Inception V2, Inception V3



Source: Dive into Deep learning book

POLYTECH NANCY UNIVERSITÉ DE LORRAINE CRAN

ResNet (He et al., 2015)

Problem: Deeper networks do not necessarily lead to better accuracy.

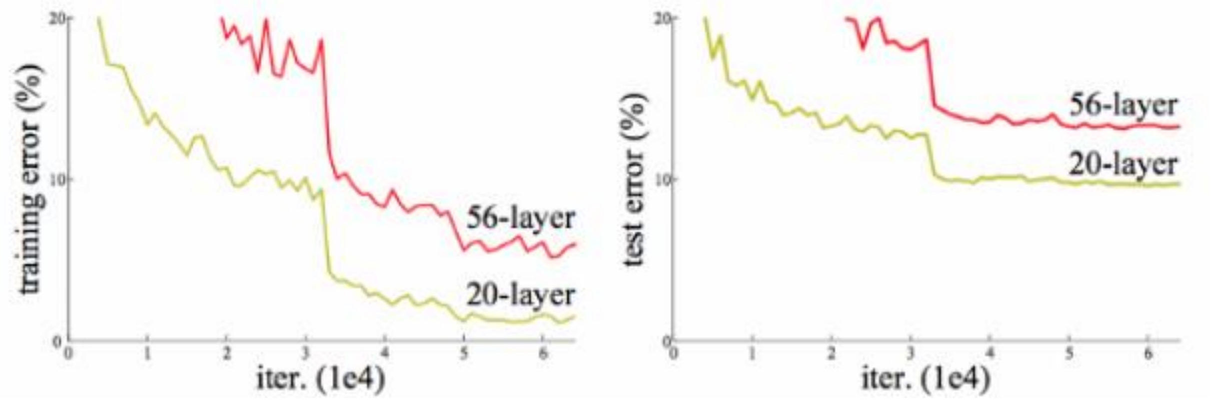
ResNet (He et al. 2015)

Problem: Deeper networks do not necessarily lead to better accuracy. WHY?

Vanishing gradients? (infinitesimally small gradients?)

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

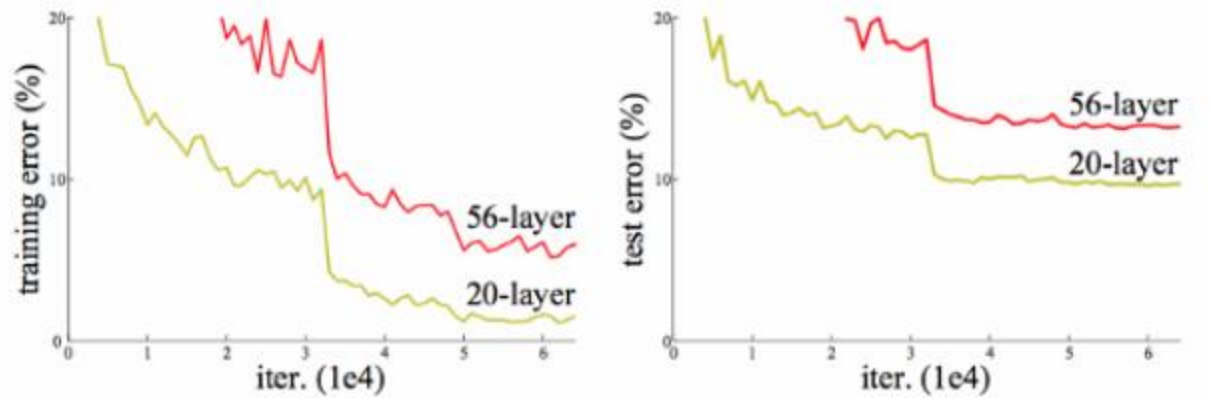


Source: Dive into Deep learning book

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

Overfitting ?



ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

Degradation Problem:

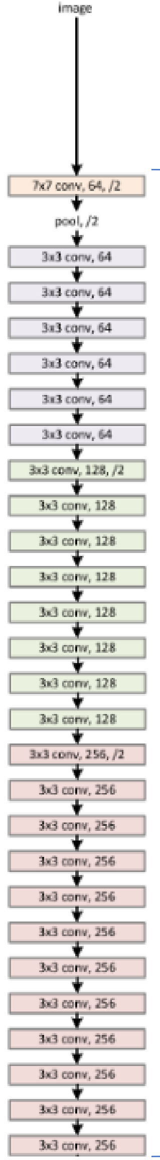
With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.



Learning is done

Image Credits: (He et al. 2015)

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.

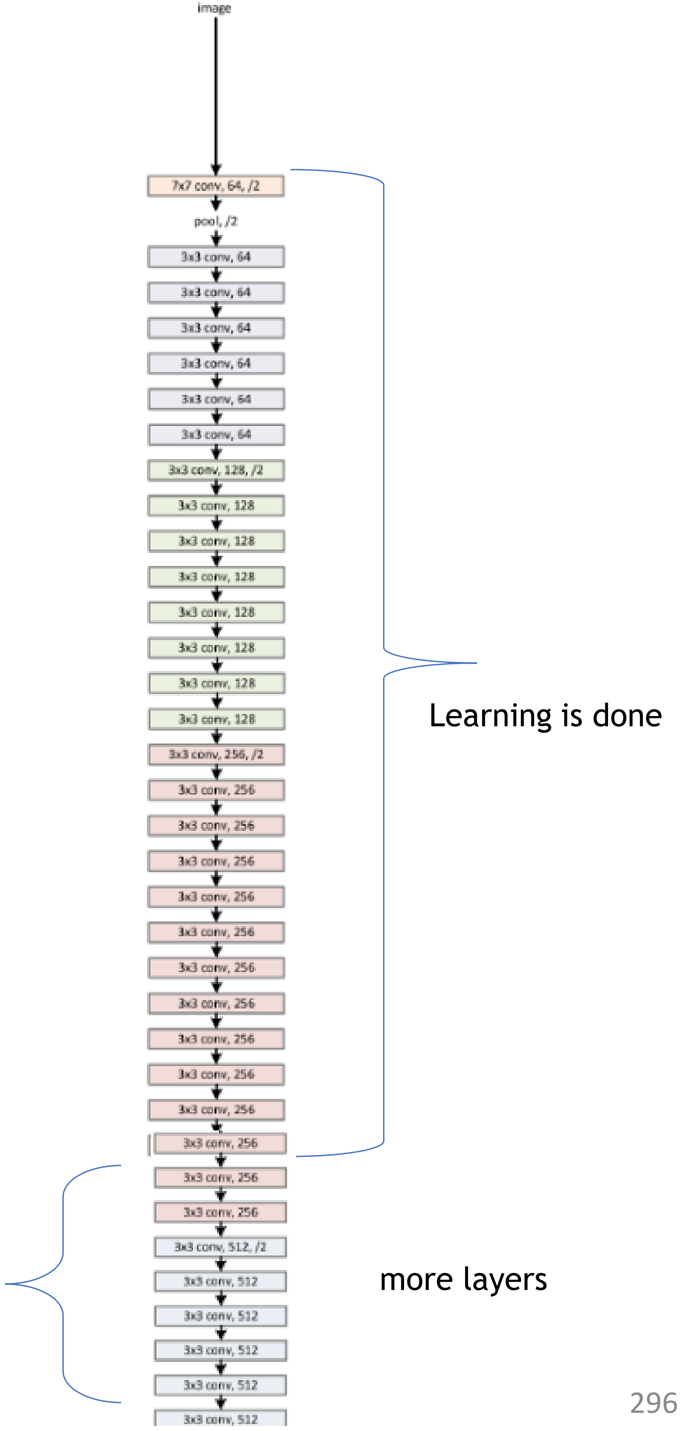


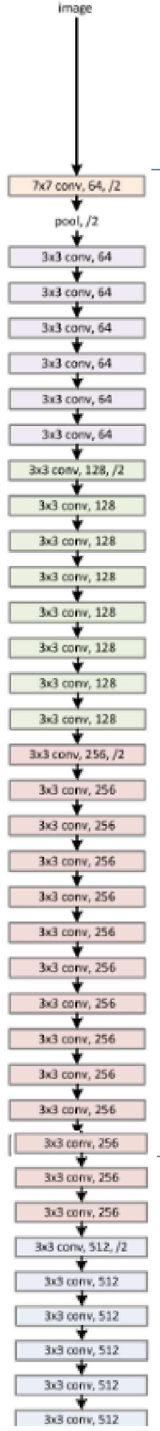
Image Credits: (He et al. 2015)

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.



Learning is done

Should behave as Identity Function (let the input from previous layer flow ahead)

$$f(x) = x$$

more layers

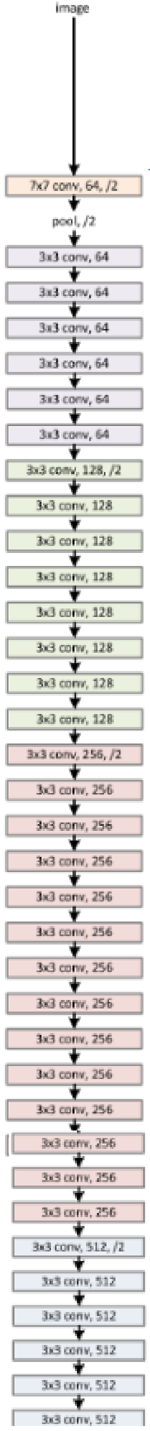
Image Credits: (He et al. 2015)

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.



Learning is done

more layers

~~Should behave as Identity Function (let the input from previous layer flow ahead)~~

$\cancel{f(x) = x}$

Image Credits: (He et al. 2015)

ResNet (He et al. 2015)

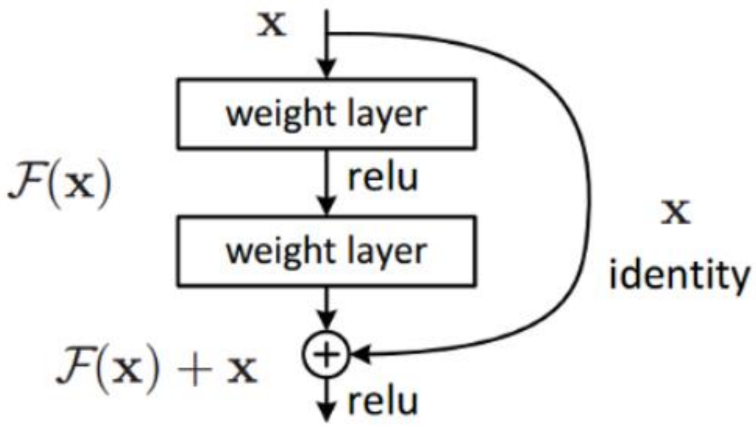
Observation: Training accuracy dropped when the count of layers was increased.

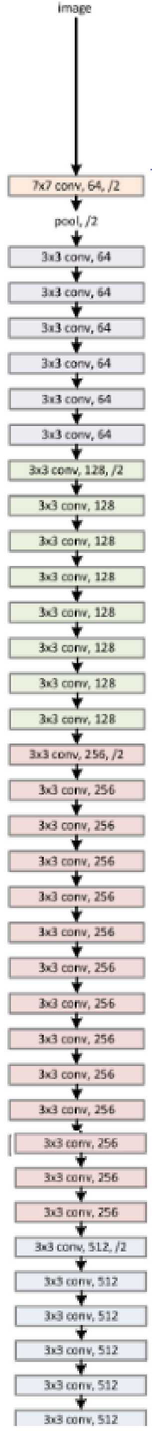
Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.

Intuition :

- Learn Residual mapping





Learning is done

But, more layers present

Image Credits: (He et al. 2015)

ResNet (He et al. 2015)

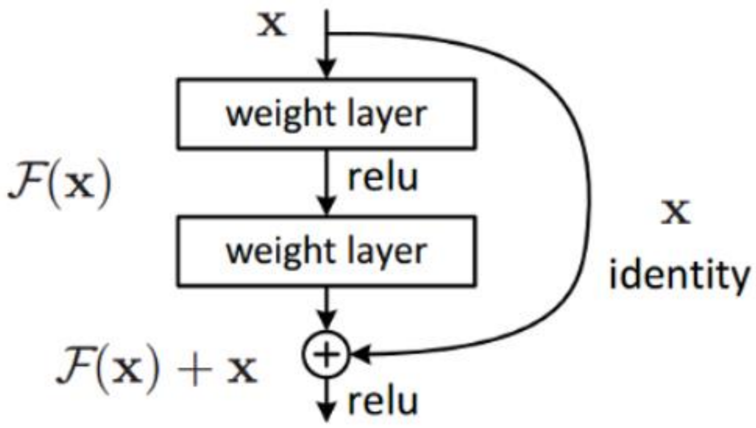
Observation: Training accuracy dropped when the count of layers was increased.

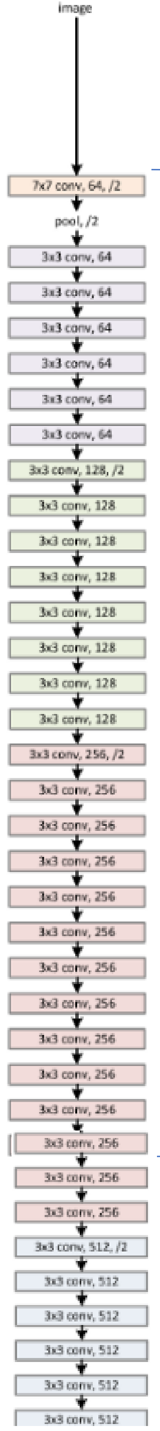
Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.

Intuition :

- Learn Residual mapping
- Use skip connections
- If any layer hurts performance ➔ skip it!
- Easier to learn $F(x) = 0$ so that it behaves as identity function.





Learning is done

But, more layers present

ResNet (He et al. 2015)

Architecture:

- Identity Block: skip connections
- Conv block: restructure incoming data
- 153 layers Deep
- Less computational complexity (but deeper : 20 X AlexNet, 8 X VGG)

Advantage:

- Residual mapping can learn the identity function more easily
- Stacking more layers ➔ equivalent to stacking identity mappings
- Inputs can forward propagate faster through the residual connections across layers.

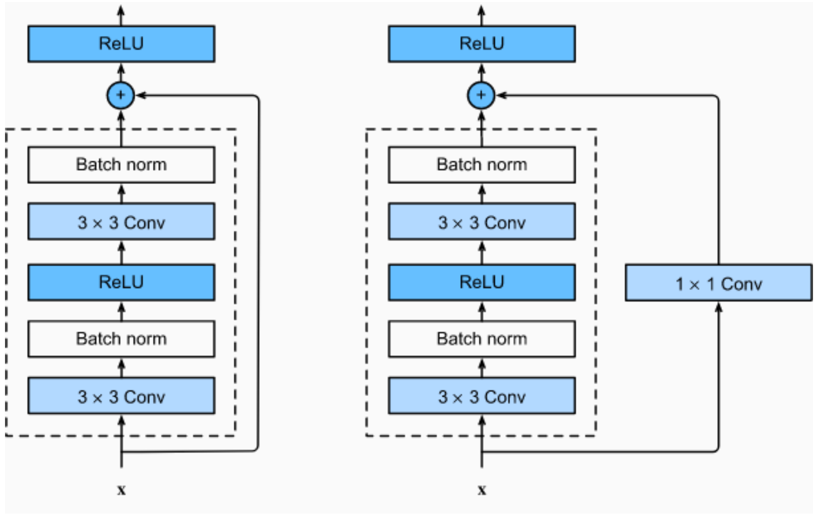


Image Credits: Dive into Deep learning

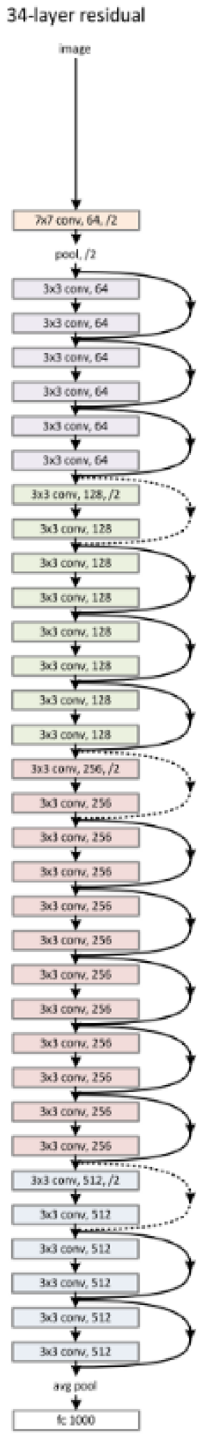
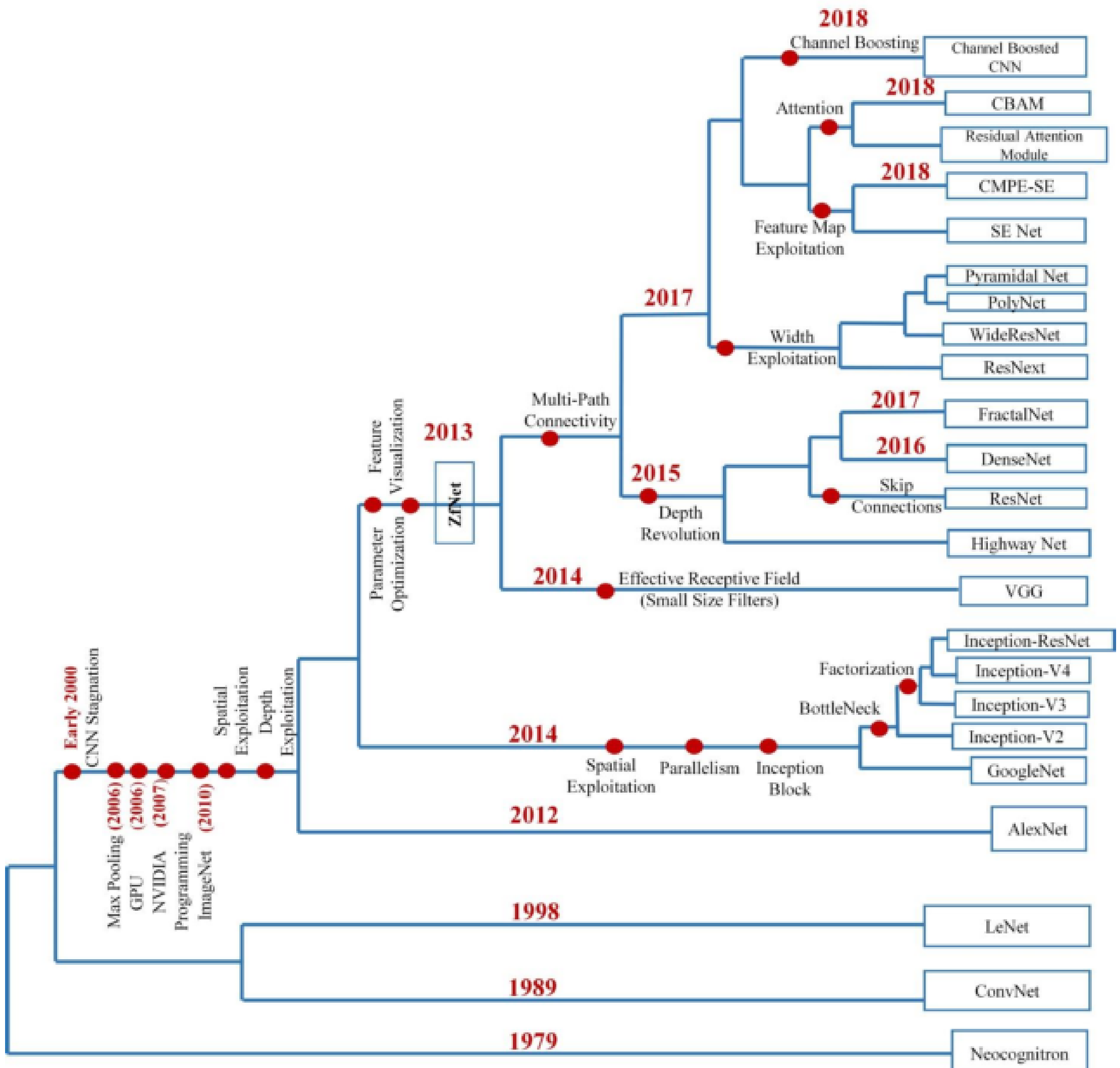


Image Credits: (He et al. 2015)

Where are we?



Khan et.al. 2020

Context: Predictive Maintenance

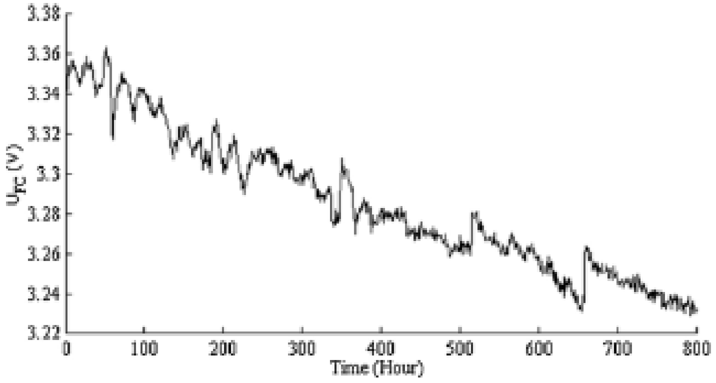
Degradation Models and RNNs

Degradation Models : Sensor signals ➔ time series data ➔ Hidden pattern: ➔ Cyclic

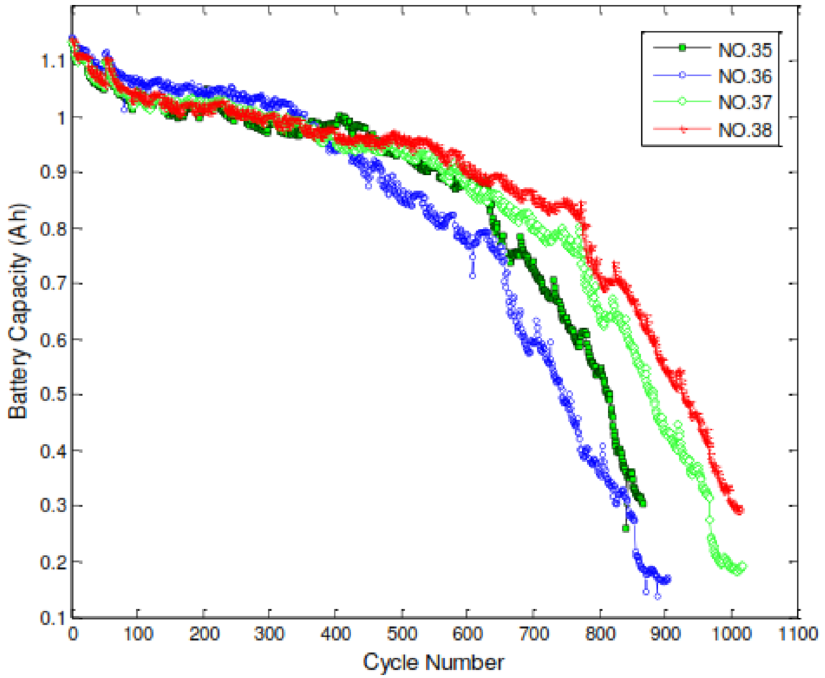
➔ Trend

➔ Seasonal

➔ Trend + Cyclic+seasonal

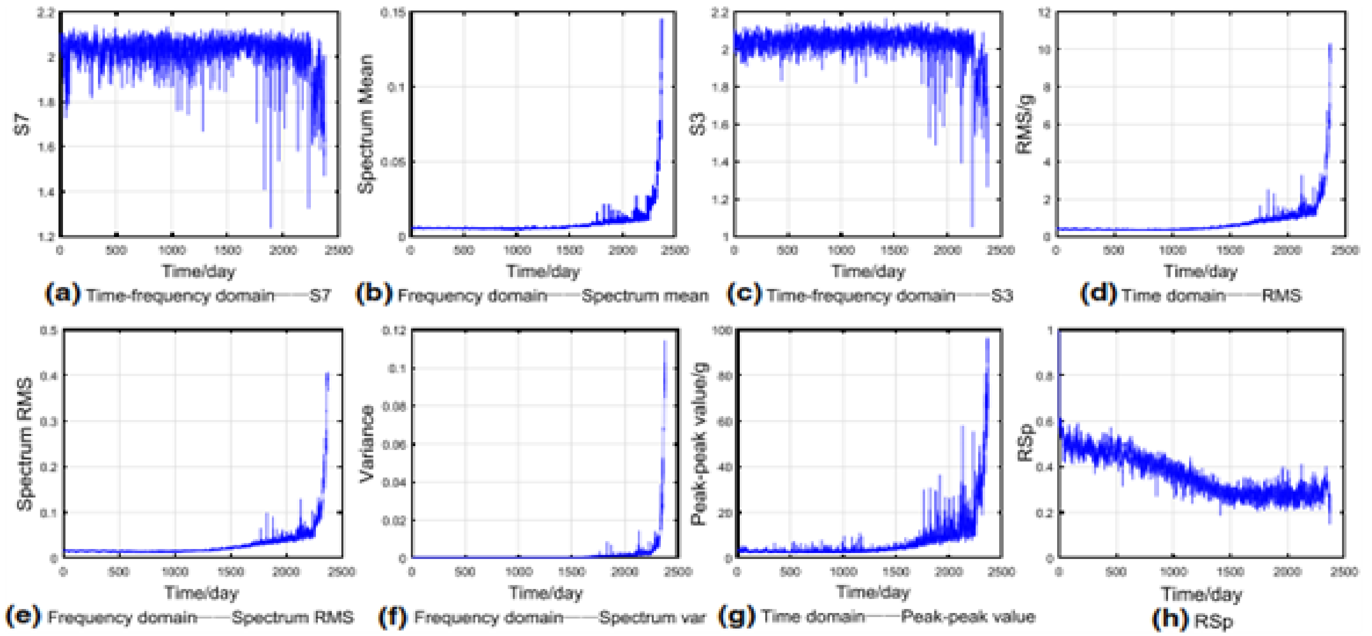


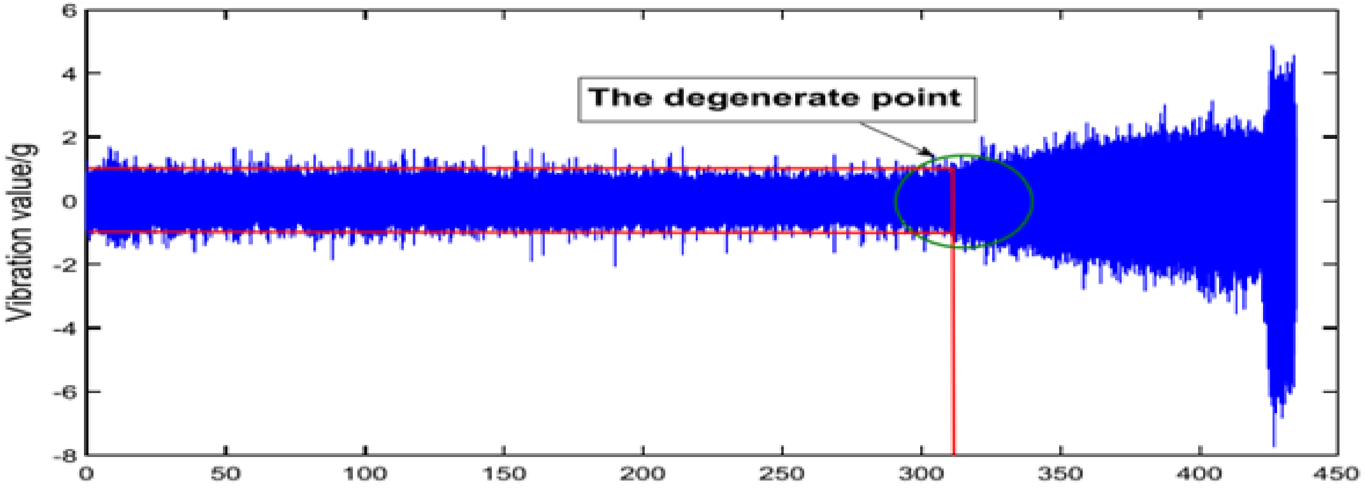
PEM Fuell Cells



Lithium-ion battery degradation,
Center for Advanced Life Cycle Engineering (CALCE)
in University of Maryland (He W., Williard N., Osterman M., & Pecht M., 2011)

Degradation Models and RNNs





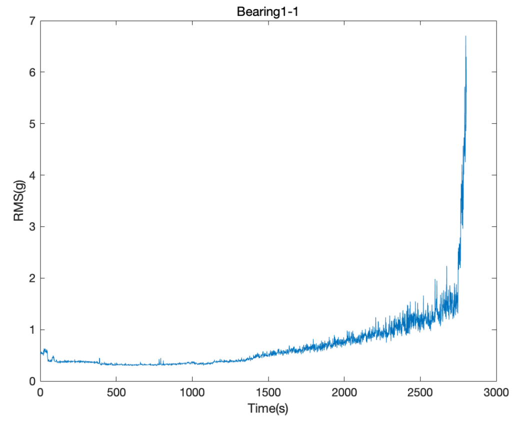
Bearing Degradation Dataset

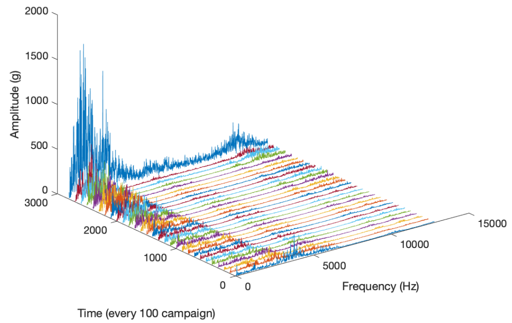
Degradation Data

- Degradation:
  - unknown, non-linear varying dynamics
  - sensor data: non-stationary process → trend, seasonality, cyclic etc.
  - depends on qualitative+ quantitative factors.

- Raw degradation data → Hidden features / representation:
  - Spatially varying
  - Temporally varying
  - Multimodal characteristics

CNNs





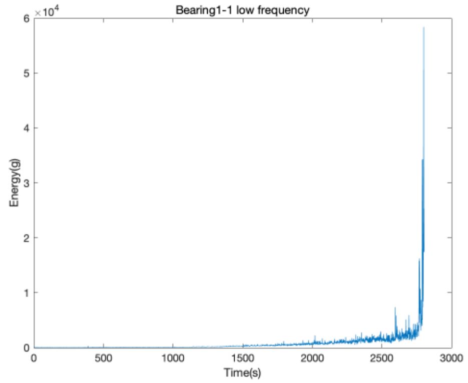
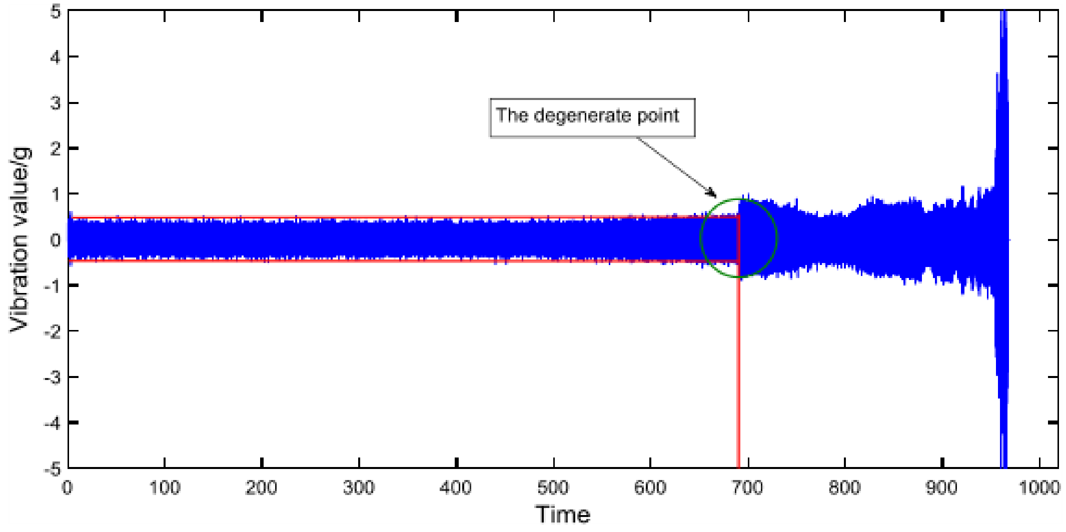


Photo: Report of Jha



Roller bearing degradation (PRONOSTIA platform)

CNNs for Prognostics

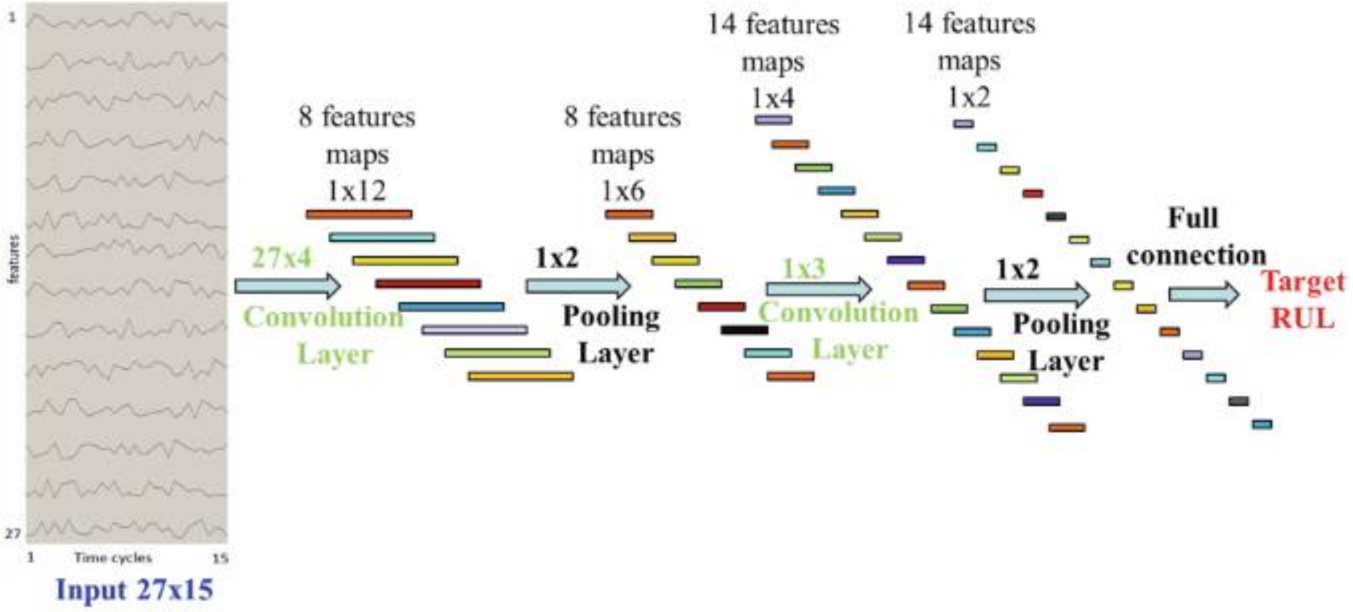
CNN for multi-variate time series signals:

- Sliding windows approach
- Segments of time series multi variate signal

(short pieces of signal)

- **Highlight: Joint feature learning** on each segmented signal
- concatenate MLP at end, for RUL target.

Li et al.	2018	Automatic feature extraction and failure prognostics: C-MAPSS data set [106]
Liu et al.	2017	Automatic feature extraction and fault diagnostics: Electric machine fault simulator
Jing et al.	2017	Automatic feature extraction and fault diagnostics: Gearbox
Babu et al.	2016	Automatic feature extraction and failure prognostics: PHM08 data set [101] C-MAPSS data set [106]



Babu et al.2016

Deep LSTMs for RUL prediction

- Degradation data➔ Time Series sequence ➔ segmented into sliding windows.
- Each sliding window is assigned a target RUL value [Zeng et al, 2017]

$X = [X_1, X_2, \ldots, X_t, \ldots X_{T-1}]$ to estimate RUL_{T-1}

$X = [X_1, X_2, \ldots, X_t, \ldots X_{T-2}]$ to estimate RUL_{T-2}

Deep LSTMs for RUL prediction

- Degradation data➔ Time Series sequence ➔ segmented into sliding windows.
- Each sliding window is assigned a target RUL value [Zeng et al, 2017]

$X = [X_1, X_2, ..., X_t, ... X_{T-1}]$ to estimate RUL_{T-1}

$X = [X_1, X_2, ..., X_t, ... X_{T-2}]$ to estimate RUL_{T-2}

Many variants exist!

$[X_t, X_{t-1}, \ldots, X_{t-d+1}], \epsilon \mathbb{R}^d$

$[\mathrm{RUL}_{t+L}, \mathrm{RUL}_{t+L+1}, \ldots, \mathrm{RUL}_n]$

Training tuples:

$([X_t, X_{t-1}, \ldots, X_{t-d+1}], \mathrm{RUL}_{t+L})$

$$\widehat{\mathrm{RUL}_{t+L}} = \phi(X_t, X_{t-1}, \ldots, X_{t-d+1})$$

Loss Calculation : Error based cost function

$$J = \sum_t \|(RUL_{est}^t - RUL_{calc}^t\|^2$$

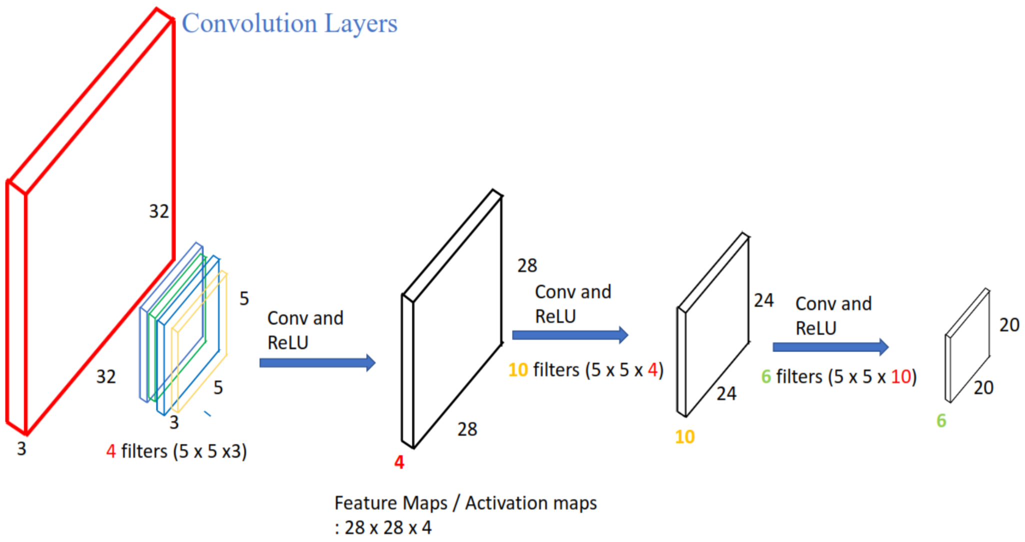
Some issues:

- Independent Windows → to assure assumption of i.i.d
- Dependent windows → claim more realistic.

CNNs for Prognostics

- Traditionally, 2D-3D structured data for face/object recognitio

- Application to PHM: 1D grid structured topology of sequential data.



Jha, course on Deep learning 2020, Polytech Nancy

L. Jing, M. Zhao, P. Li, and X. Xu, "A convolutional neural network based feature learning and fault diagnosis method for the condition monitoring of gearbox," *Measurement*, vol. 111, no. Supplement C, pp. 1 – 10, 2017.

Diagnostics:
- Input: 1D segments of vibration data
- Highlight: Automatic extraction of features
- Train: several layers CNN + Softmax classification

CNNs for Prognostics

- Automatically learn feature representation, hidden multimodal distributions

[Liu et al., 2017] [Jing et al., 2017] [Li et al., 2018]

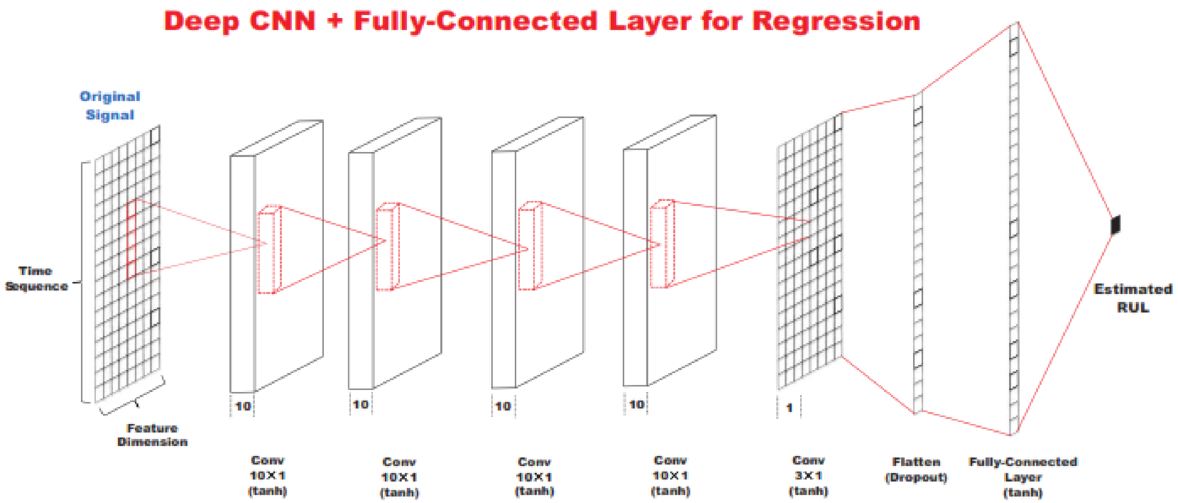
&

- Efficient learning with multi-variate sequential (time series) data.

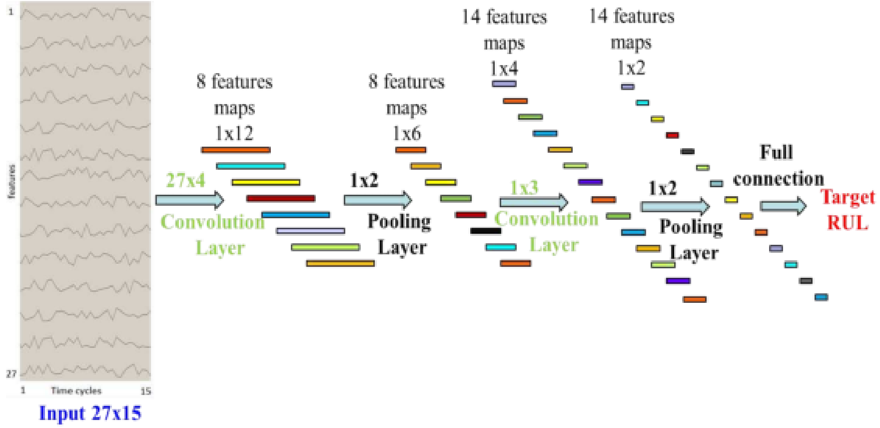
[Babu et al., 2016]

- Hybrid structure

[Babu et al., 2016]



[Liu et al., 2017]



Turbo jet Fan Engine NASA

CMAPSS

NASA/TM—2007-215026

User's Guide for the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS)

Dean K. Frederick
Saratoga Control Systems, Inc., Saratoga Springs, New York

Jonathan A. DeCastro
ASRC Aerospace Corporation, Cleveland, Ohio

Jonathan S. Litt
U.S. Army Research Laboratory, Glenn Research Center, Cleveland, Ohio

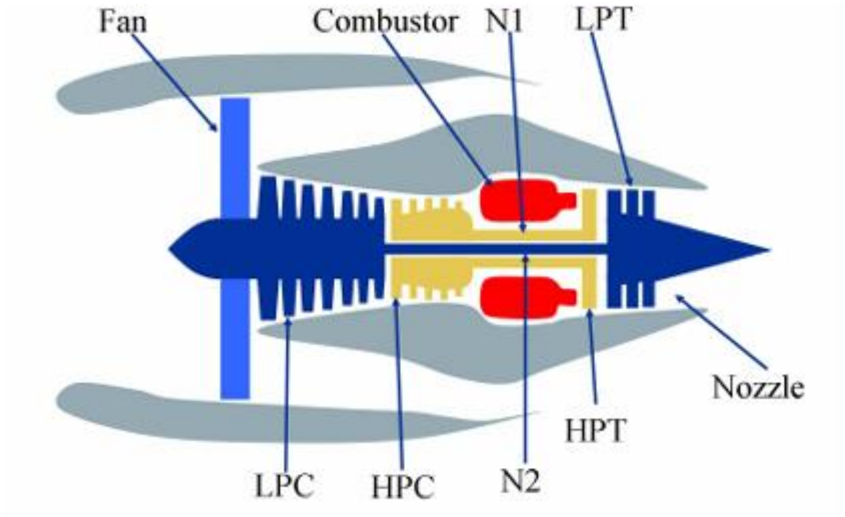


Figure 1.1.—Simplified diagram of the 90K engine.

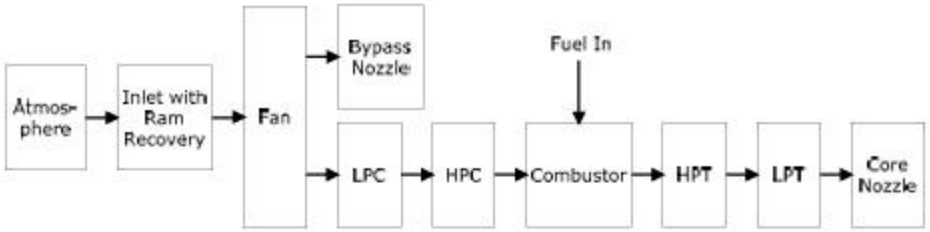


Figure 1.2.—Subroutines of the 90K engine simulation with ducts and bleed omitted.

Sequence Modelling

Recurrent Neural Networks

Long Short Term Memory (LSTMs)

Application: Prognostics and Deep Learning

Sequence Modelling

Motivations

Challenges

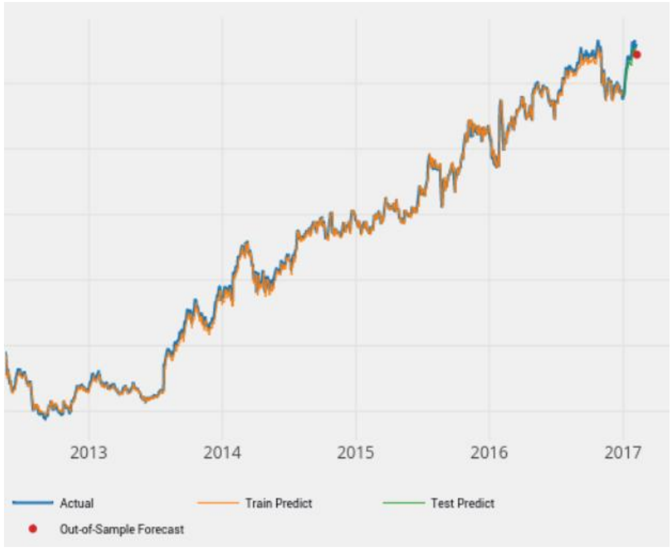
Some ideas

Design

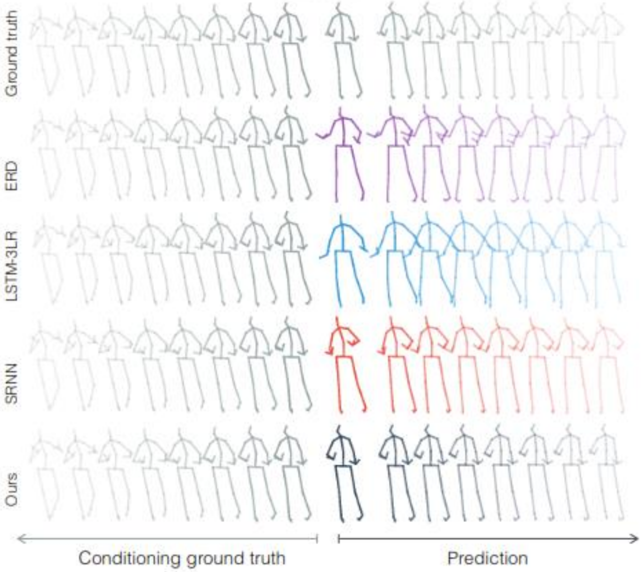
JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

Sequence modelling : Motivations

- Sequential data:
  - time series forecasting,
  - motion prediction (human, self driving cars)
  - sensor data: machine health monitoring/prediction
  - text processing/prediction
  - machine translation



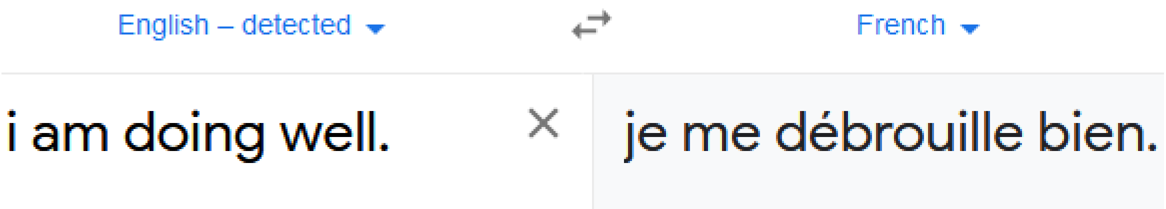
Financial market prediction (Dixon et al.)

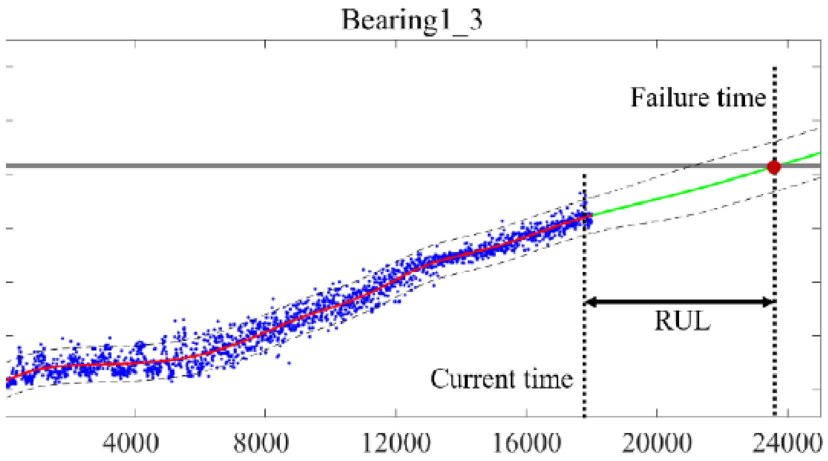


Human Motion Prediction

Martinez et al., 2016

I am from London but I live in Paris and I speak fluent ***English.***





Component Failure Prediction
(Yoo et al., 2018)

Sequence modelling : Motivations > Challenges

- Inputs data
  - Variable lengths
  - spatially + temporally dependent
  - ordered
  - output data different length than input (machine translation)

1. Fixed window
   - cannot model long term dependencies
2. Use whole sequence as counts (I occurs 3 times)
   - no learning of order (what followed by what?)
3. Large window length input
   - each has separate parameter
   - learning will not transfer at other places in the sequence.

I am from London but I live in Paris so I speak fluent ....

One hot coding

[00100101000101011000011....] → ???

I live in Paris so I speak fluent

Sequence modelling : Motivations >Challenges > Some ideas

I am from London but *I live in Paris so I speak fluent* ....

One hot coding

[0010010100010101100001 1....] ➡ ???

I *live* *in* *Paris* *so* *I* *speak* *fluent*

1. Fixed window
   - cannot model long term dependencies
2. Use whole sequence as counts (*I* occurs 3 times)
   - no learning of order (what followed by what?)
3. Large window length input
   - each has separate parameter
   - learning will not transfer at other places in the sequence.

- Feed forward NN, not designed to:
  - handle variable data lengths
  - parameter sharing (correlation, temporal dependency…)
  - track long term dependency + order

- CovNets:
  - can share parameters across time but remain shallow.

Sequence modelling : Motivations >Challenges > Some ideas > Design

- Variable length inputs

- Learn long term dependencies

- Learn the order in data

- Share parameters across sequence

- Make predictions (long term) efficiently.

Recurrent Neural Networks

RNNs: Structure

- Recurrence of states. Ex. a dynamical system
- Recursive computation ➔ Computational graph

$$s_t = f(s_{t-1}, w)$$

$$\cdots \xrightarrow{f} s_{t-1} \xrightarrow{f} s_t \xrightarrow{f} s_{t+1} \xrightarrow{f} s_{t+2}$$

$$s_3 = f(s_2, w)$$

$$= f(f(s_1, w), w)$$

RNNs: Structure

$$s_t = f(s_{t-1}, w)$$

- Recurrence of states. Ex. a dynamical system
- Recursive computation ➔ Computational graph
- When system driven by external input,

RNNs: Structure

- Recurrence of states. Ex. a dynamical system
- Recursive computation ➔ Computational graph
- When system driven by external input,

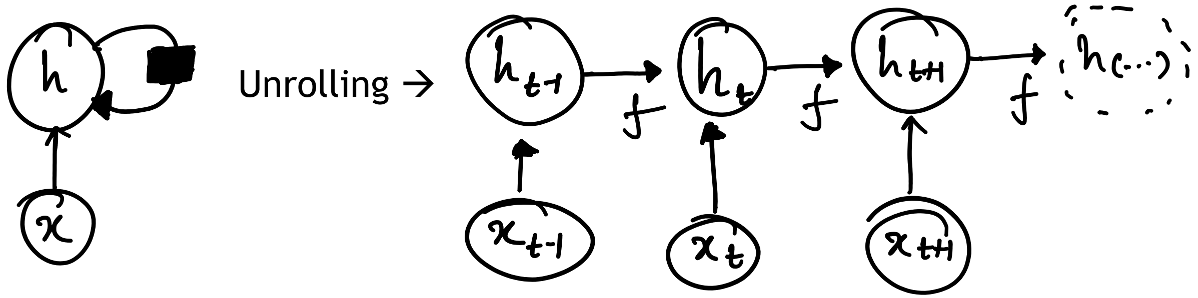
  New state contains information about history.

$$s_t = f(s_{t-1}, w)$$

Rewritten

$$h_t = f(h_{t-1}, x_{t-1}, w)$$

RNNs : Output of node fed back into the hidden nodes (recurrent, cyclic structure)



- Captures dependency in input data.
- Same weights at each time step : some weight sharing.

$$h_t = f(h_{t-1}, x_{t-1})$$
$$= \sigma(W^h h_{t-1} + W^x x_{t-1})$$

RNNs: Structure

- Recurrence of states. Ex. a dynamical system
- Recursive computation ➔ Computational graph
- When system driven by external input,

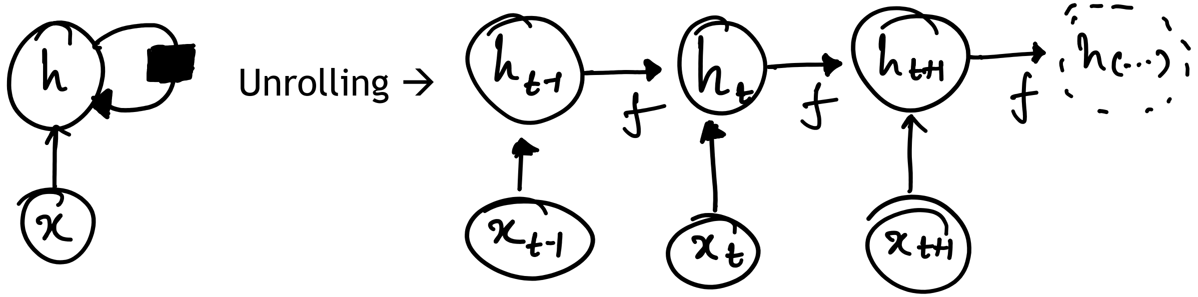
  New state contains information about history.

$$s_t = f(s_{t-1}, w)$$

Rewritten

$$h_t = f(h_{t-1}, x_t, w)$$

RNNs : Output of node fed back into the hidden nodes (recurrent, cyclic structure)

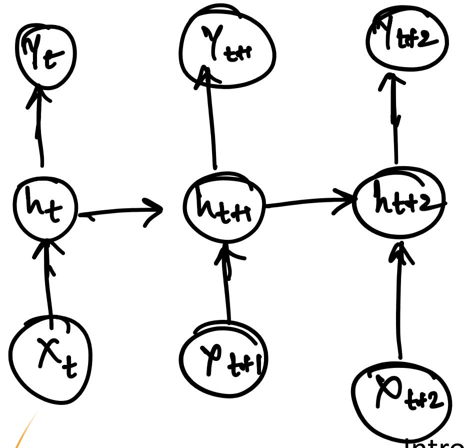


- Captures dependency in input data.
- Same weights at each time step : some weight sharing.

$$h_t = f(h_{t-1}, x_t)$$

$$= \sigma(W^h h_{t-1} + W^x x_{t-1})$$

- Outputs from RNN:



$$y_t = g(h_t, x_t)$$

$$= \sigma(W^h h_t)$$

RNNs: Structure

$$s_t = f(s_{t-1}, w)$$

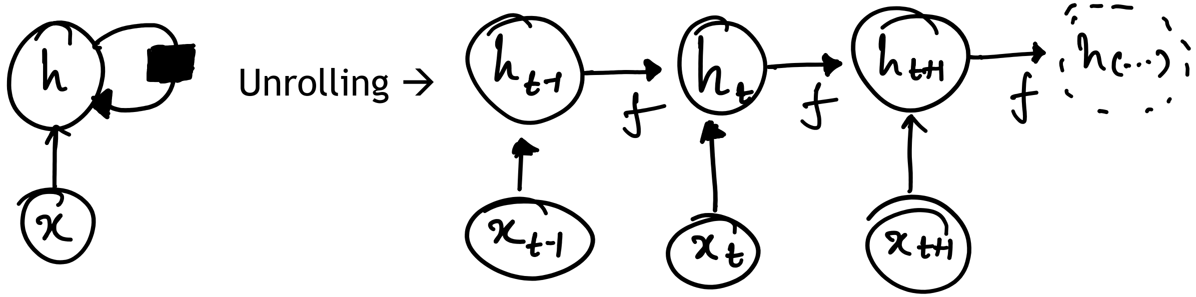
Rewritten

$$h_t = f(h_{t-1}, x_t, w)$$

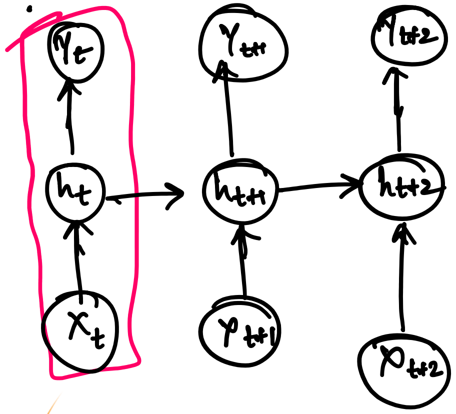
- Recurrence of states. Ex. a dynamical system
- Recursive computation ➔ Computational graph
- When system driven by external input,

  New state contains information about history.

RNNs : Output of node fed back into the hidden nodes (recurrent, cyclic structure)



- Captures dependency in input data.
- Same weights at each time step : some weight sharing.
- Outputs from RNN:



$$h_t = f(h_{t-1}, x_t)$$

$$= \sigma(W^h h_{t-1} + W^x x_{t-1})$$

$$y_t = g(h_t, x_t)$$

$$= \sigma(W^h h_t)$$

RNNs: Structure

- Recurrence of states. Ex. a dynamical system
- Recursive computation ➔ Computational graph
- When system driven by external input,

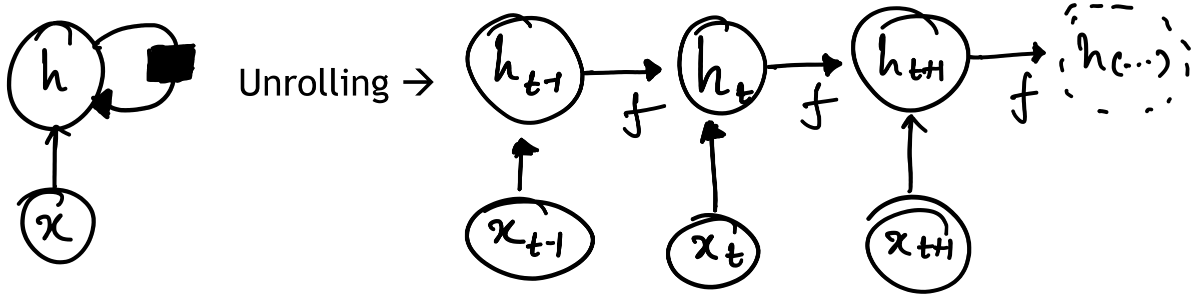
  New state contains information about history.

$$s_t = f(s_{t-1}, w)$$

Rewritten

$$h_t = f(h_{t-1}, x_t, w)$$

RNNs : Output of node fed back into the hidden nodes (recurrent, cyclic structure)

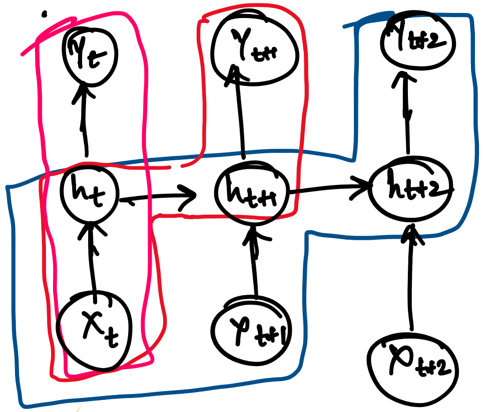


- Captures dependency in input data.
- Same weights at each time step : some weight sharing.

$$h_t = f(h_{t-1}, x_{t-1})$$

$$= \sigma(W^h h_{t-1} + W^x x_{t-1})$$

- Outputs from RNN:



$$y_t = g(h_t, x_t)$$

$$= \sigma(W^h h_t)$$

RNNs: Structure

- Depending upon application:
  - *h* needs to be rich,
  - capture all historical trends {cyclicity, seasonality, trend, fluctuations, global/local}

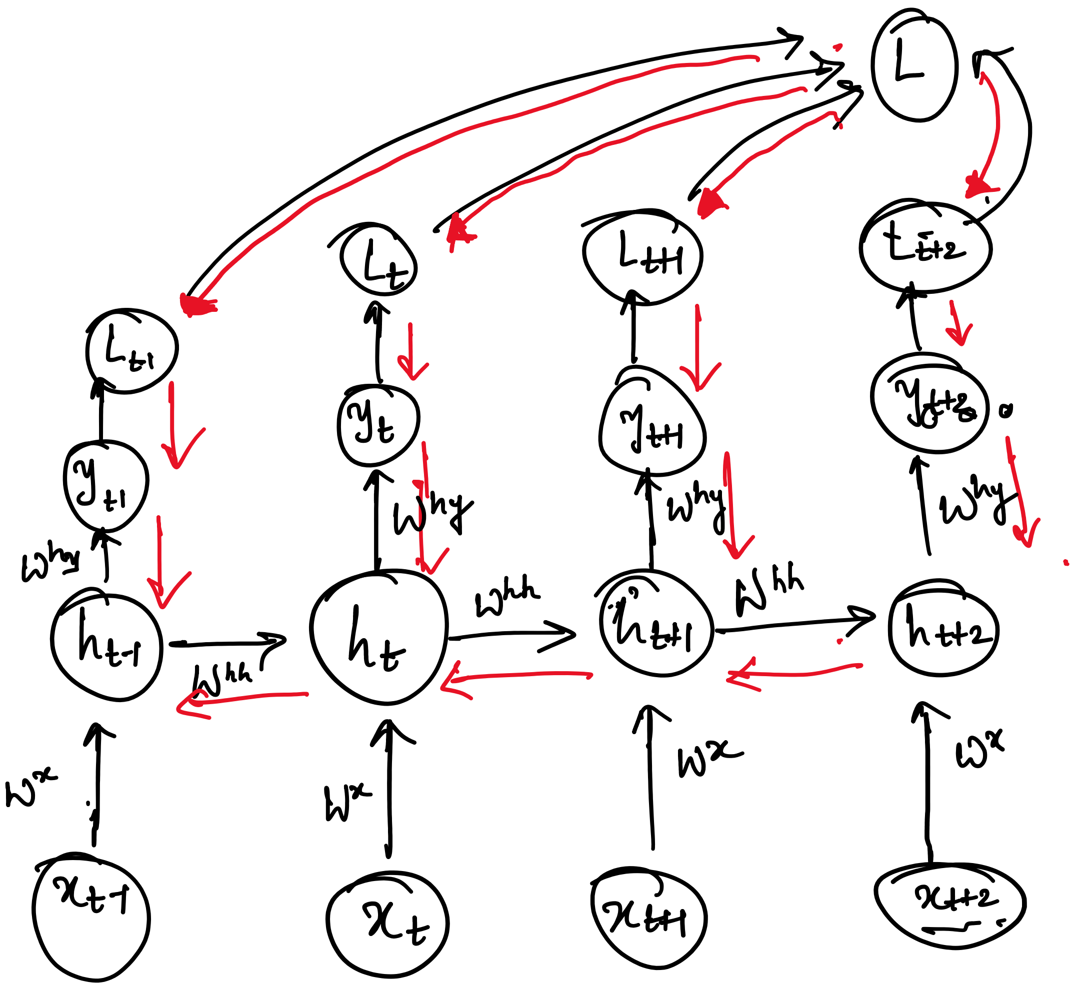
- Advantage:
  - learnt model has same size (regardless of input size)
  - possible to use same transition function *f*

RNNs: Structure

- Depending upon application:
  - *h* needs to be rich,
  - capture all historical trends {cyclicity, seasonality, trend, fluctuations, global/local}

- Advantage:
  - learnt model has same size (regardless of input size)
  - possible to use same transition function *f*

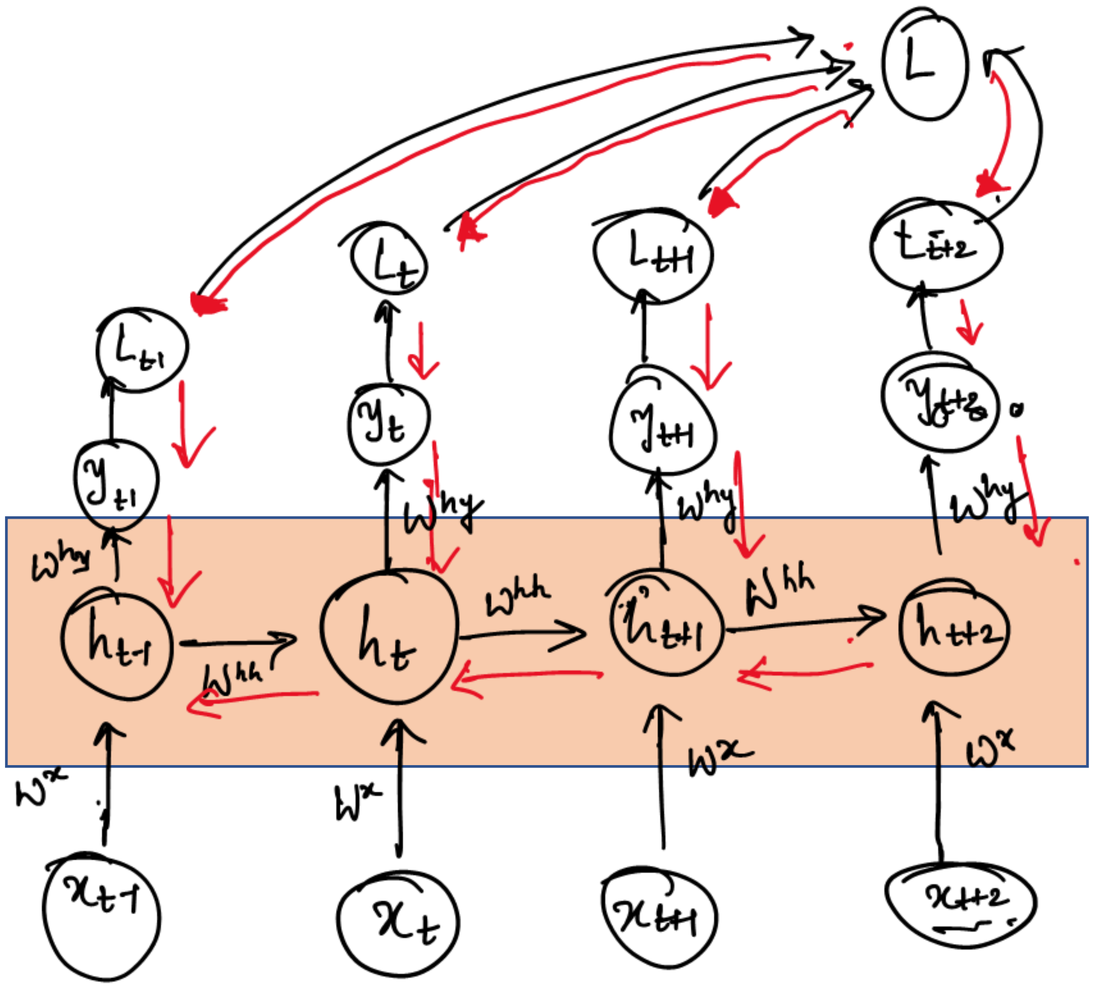
- Learning → Back-propagation through time (BPTT)
  - errors calculated/back-propagated over time = BP over unrolled network
  - gradients calculated in time.
  - Training slower than MLP:
    - repeated multiplication of weights in sequence length
    - repeated product of derivative of activation function.



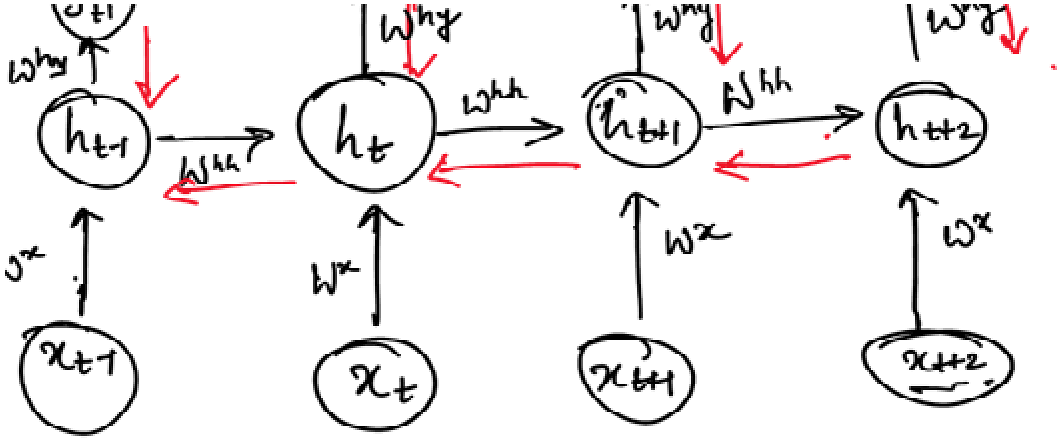
Challenges:

Vanishing gradients: Many values <<1

- activation gradient products
- small weights
- negligible gradient ➔ negligible learning.



Challenges:



Vanishing gradient problem: Many values <<1

- activation gradient products
- small weights
- negligible gradient ➔ negligible learning.

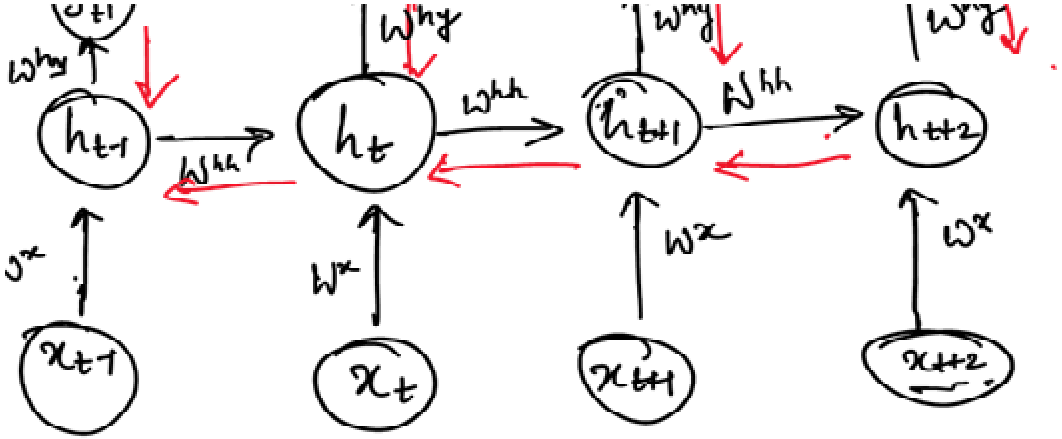
Long range Learning:

- hidden units modify with new information
- vanishing gradient problem ➔ new information not preserved over long ranges.
  - time series forecasting: seasonality etc.

Challenges:

Vanishing gradient problem: Many values <<1

- activation gradient products
- small weights
- negligible gradient ➔ negligible learning.



Long range Learning:

- hidden units modify with new information
- vanishing gradient problem ➔ new information not preserved over long ranges.
  - time series forecasting: seasonality etc.
  - machine translation: relation of first word to context
  - prognostics: prediction of state of health at long time range

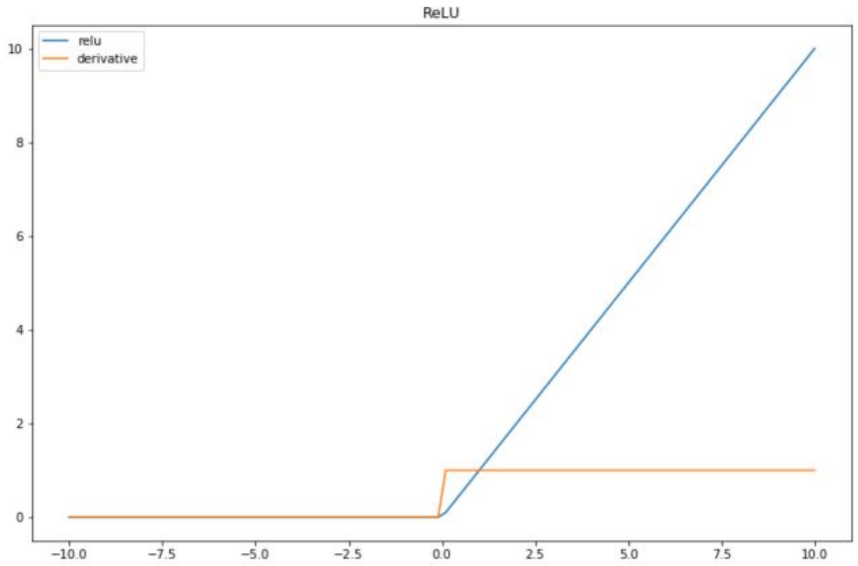
I am from London but I live in Paris so I speak fluent **…….. .**

Prediction Drift:

- next step prediction ➔ recurrence of *h* learnt
- long range prediction → recurrence of *h* over multiple steps
- error cumulation over multiple time steps

Solution:

- Efficient parameter initialization
- Non-saturating activation functions: ReLU, Leaky ReLu…
- Gradient clipping



- Gated Cells:
  - “control” the information flow
  - allow more useful information, forget non-useful information…
  - track information through many time steps to filter out the useless ones.

Long Short Term Memory (LSTMs)

LSTMs (Hochreiter & Schmidhuber 1997)

Gated RNNs: let selective information through

$$h_t = f(h_{t-1}, x_t)$$
$$= \sigma(W^h h_{t-1} + W^x x_{t-1})$$

cleverly designed

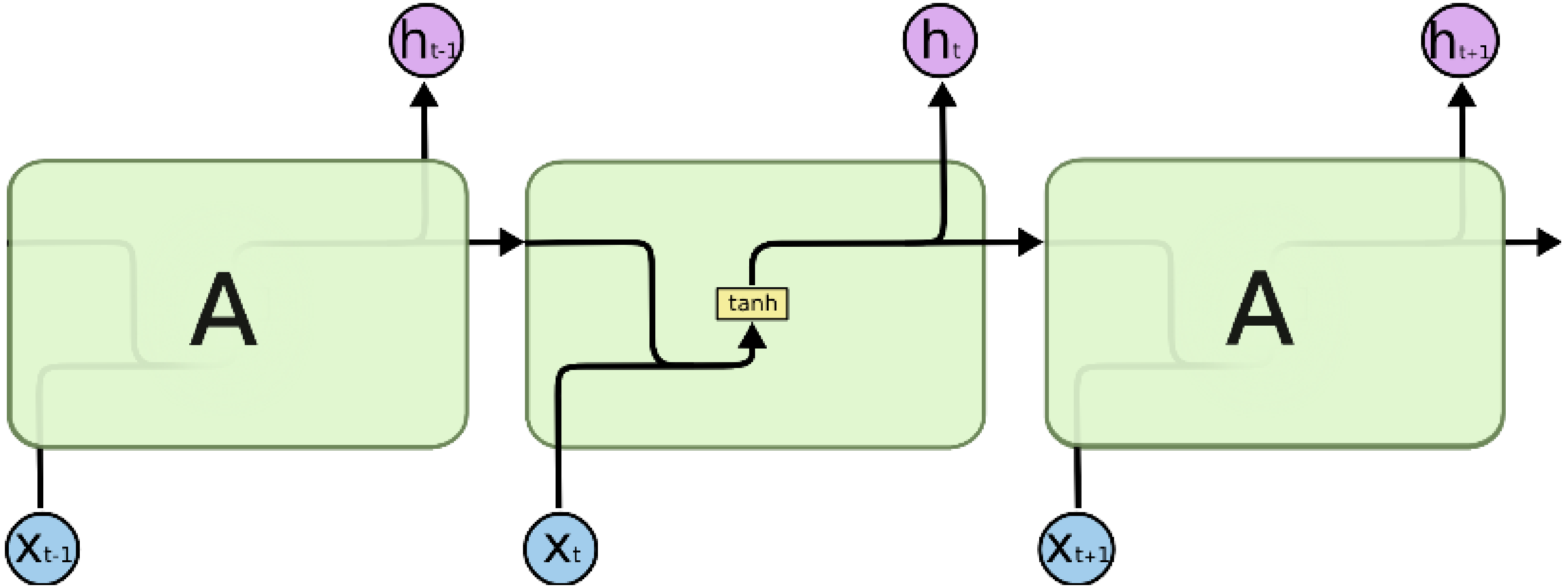
LSTMs (Hochreiter & Schmidhuber 1997)

Gated RNNs: let selective information through

$$h_t = f(h_{t-1}, x_t)$$
$$= \sigma(W^h h_{t-1} + W^x x_{t-1})$$

cleverly designed

RNNS:



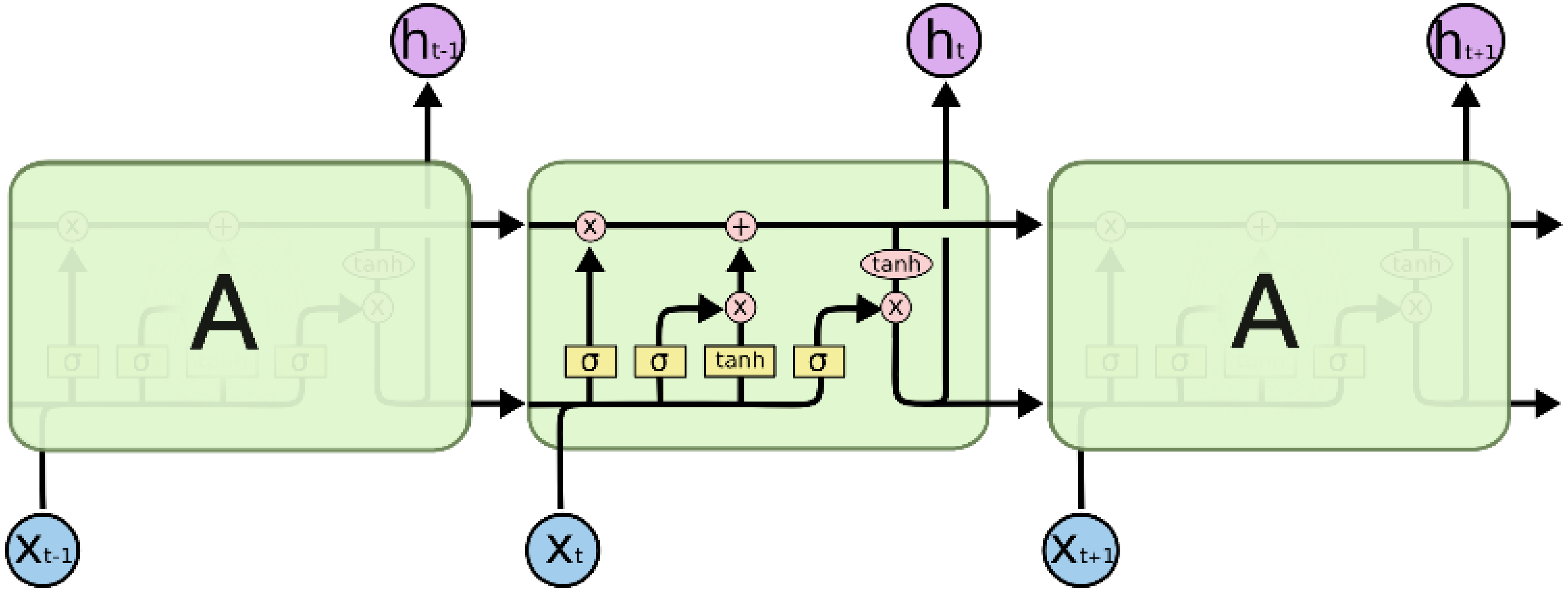
LSTMs (Hochreiter & Schmidhuber 1997)

$$h_t = f(h_{t-1}, x_t)$$
$$= \sigma(W^h h_{t-1} + W^x x_{t-1})$$

cleverly designed

Gated RNNs: let selective information through

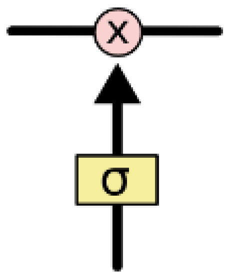
LSTMs:



LSTMs (Hochreiter & Schmidhuber 1997)

Gated RNNs: let selective information through

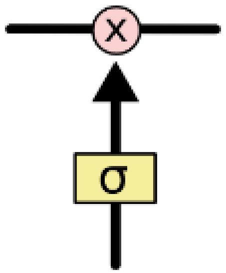
Gates:



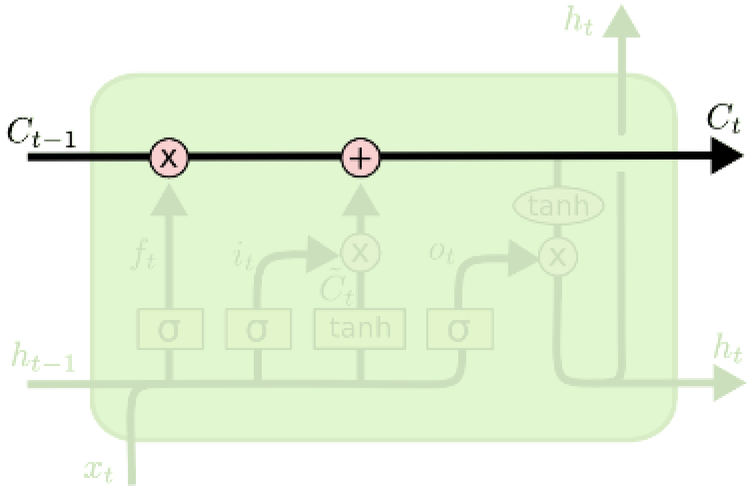
LSTMs (Hochreiter & Schmidhuber 1997)

Cell state: let selective information through

Gates:



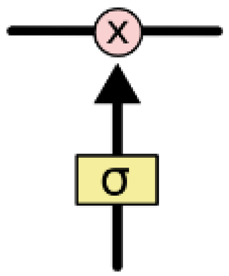
Cell state : Information highway.



LSTMs (Hochreiter & Schmidhuber 1997)

Cell state: let selective information through

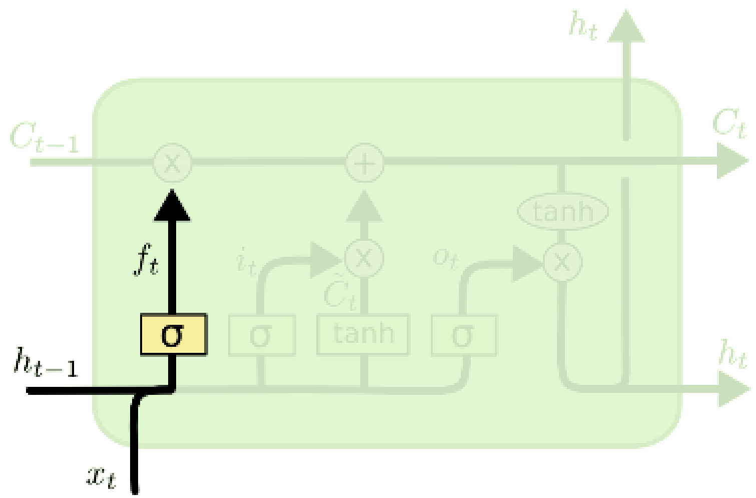
Gates:



$$f_t = \sigma(W^f [h_{t-1}, x_t] + b_f)$$

Cell state : Information highway.

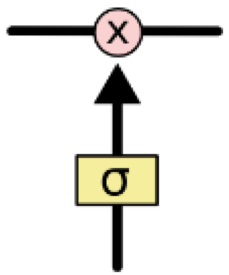
1. Forget:



LSTMs (Hochreiter & Schmidhuber 1997)

Cell state: let selective information through

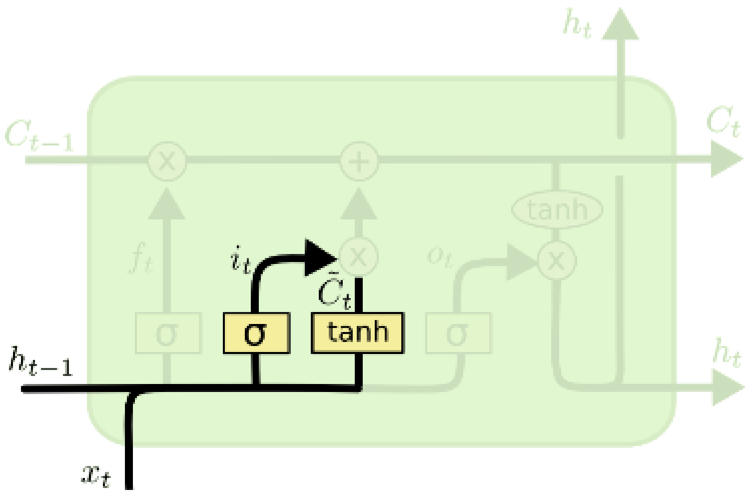
Gates:



Cell state : Information highway.

1. What to Forget:

2. what to Store:



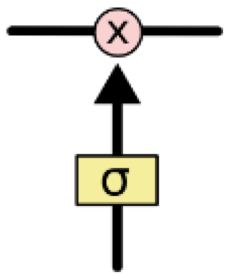
$$i_t = \sigma(W^i[h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W^C[h_{t-1}, x_t] + b_C)$$

LSTMs (Hochreiter & Schmidhuber 1997)

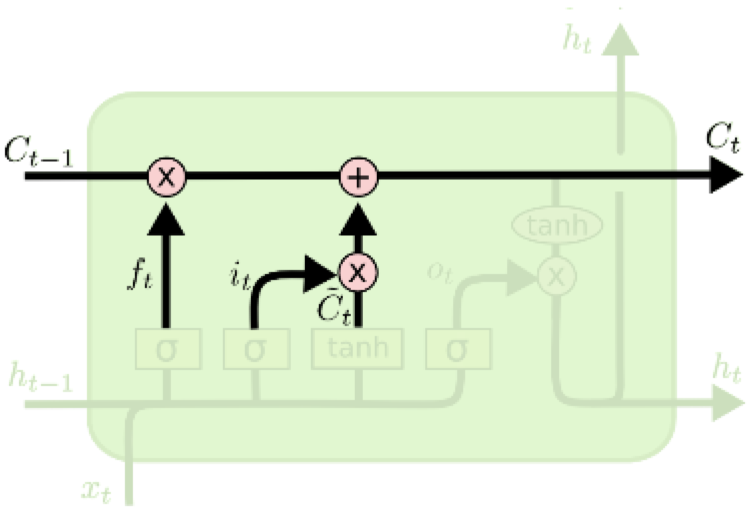
Cell state: let selective information through

Gates:



Cell state : Information highway.

1. What to Forget:
2. what to Store:
3. Update old cell state:

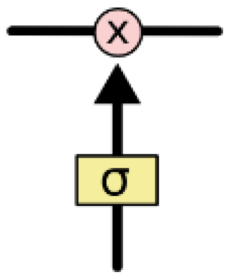


$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

LSTMs (Hochreiter & Schmidhuber 1997)

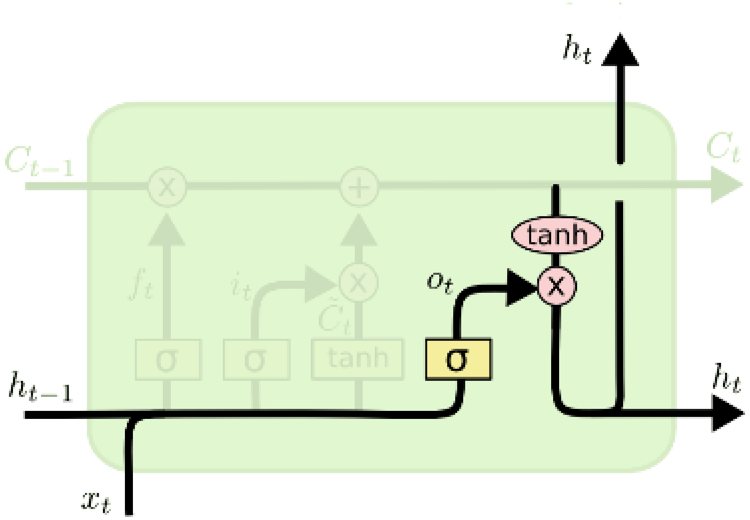
Cell state: let selective information through

Gates:



Cell state : Information highway.

1. What to Forget:
2. What to Store:
3. Update old cell state:
4. Generate output:



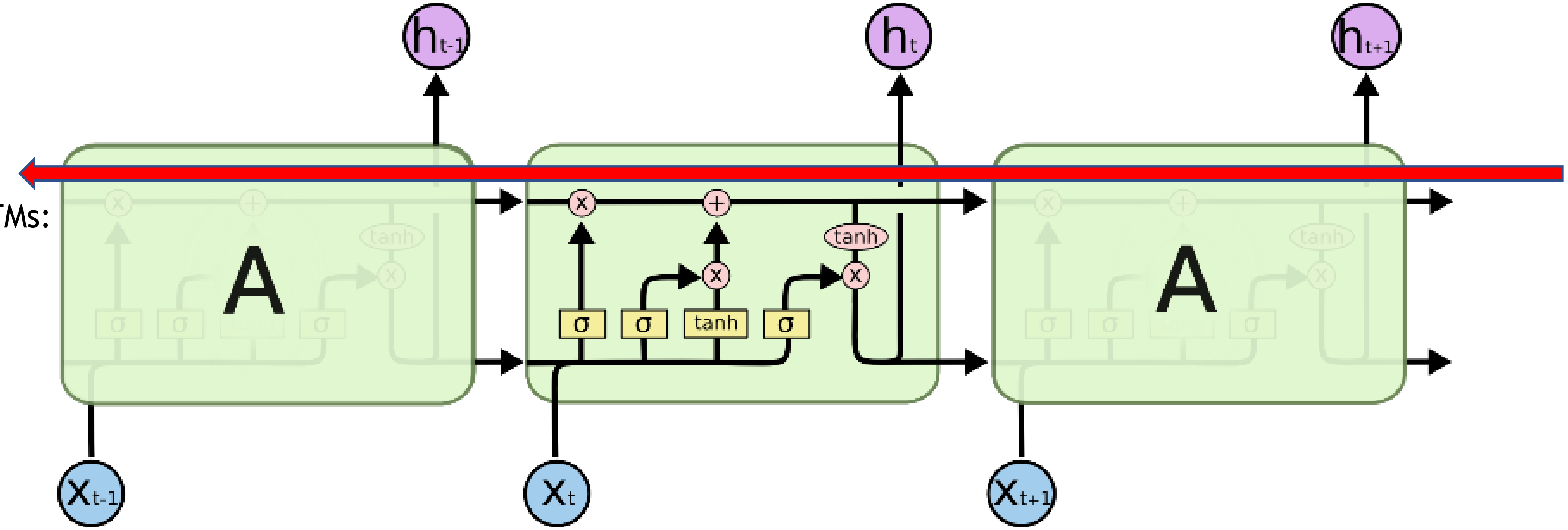
$$o_t = \sigma(W^o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

LSTMs (Hochreiter & Schmidhuber 1997)

Gated RNNs: let selective information through

Backpropagation: Uninterrupted gradient flow
Learning:
Faster than RNNs,
Long range dependency conserved..

LSTMs:

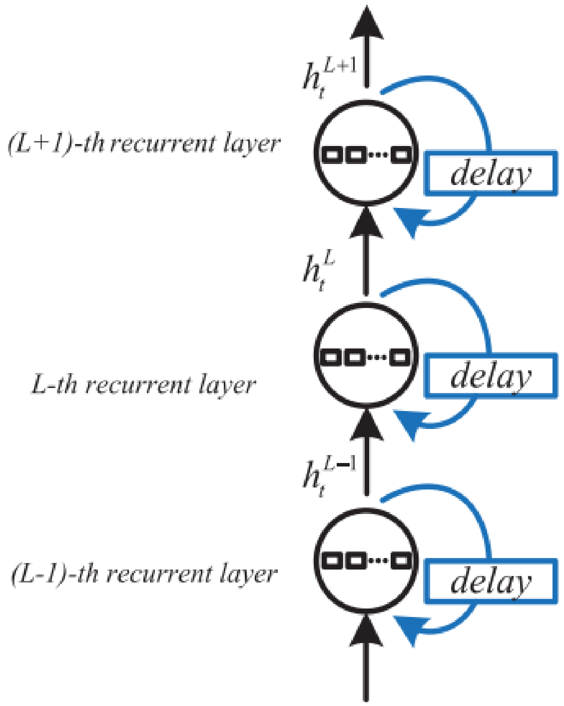


$$f_t = \sigma(W^f[C_{t-1}, h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W^i[C_{t-1}, h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W^o[C_{t-1}, h_{t-1}, x_t] + b_o)$$

LSTM Variants:

- Peephole connections
- Gated Recurrent Units (GRUs) (Cho et al. 2014)
- etc.

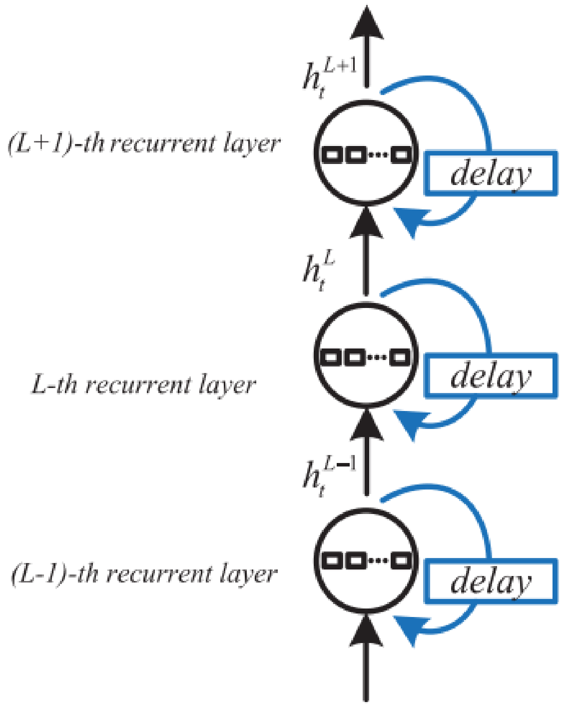
Deep (Stacked) LSTMs (Fernández, Graves, & Schmidhuber,2007):



LSTM Variants:

- Peephole connections
- Gated Recurrent Units (GRUs) (Cho et al. 2014)
- etc.

Deep (Stacked)LSTMs (Fernández, Graves, & Schmidhuber,2007):



Unroll

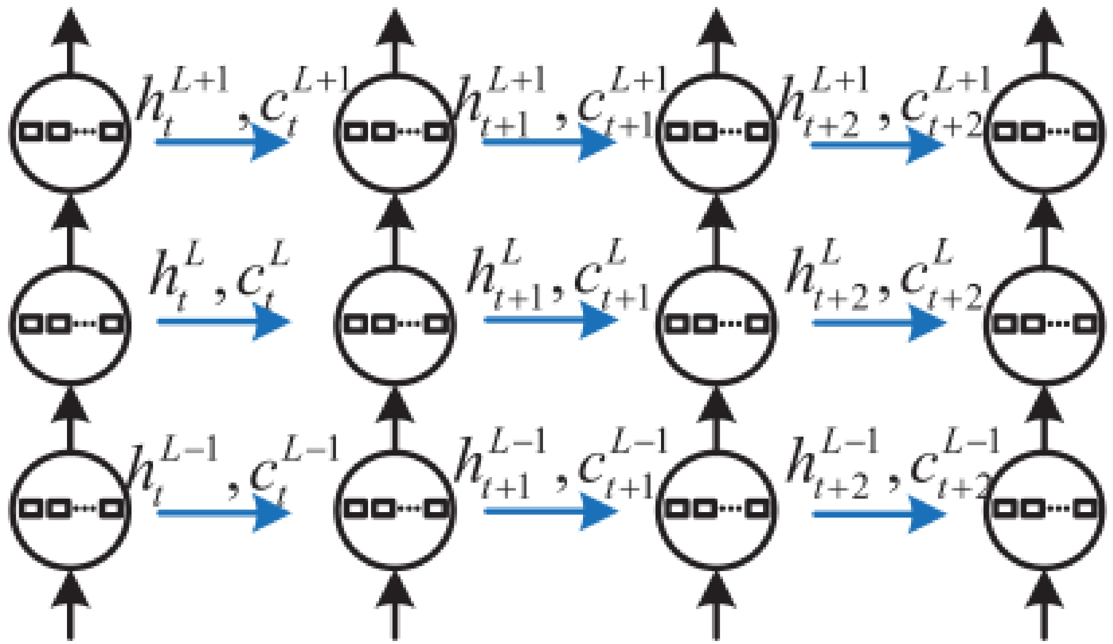


Image credits: Fernández, Graves, & Schmidhuber,2007

Deep LSTMs

- Advantages over RNNs:
  - Learn long term dependencies easily.
  - Avoid vanishing gradient problem through easy information flow.

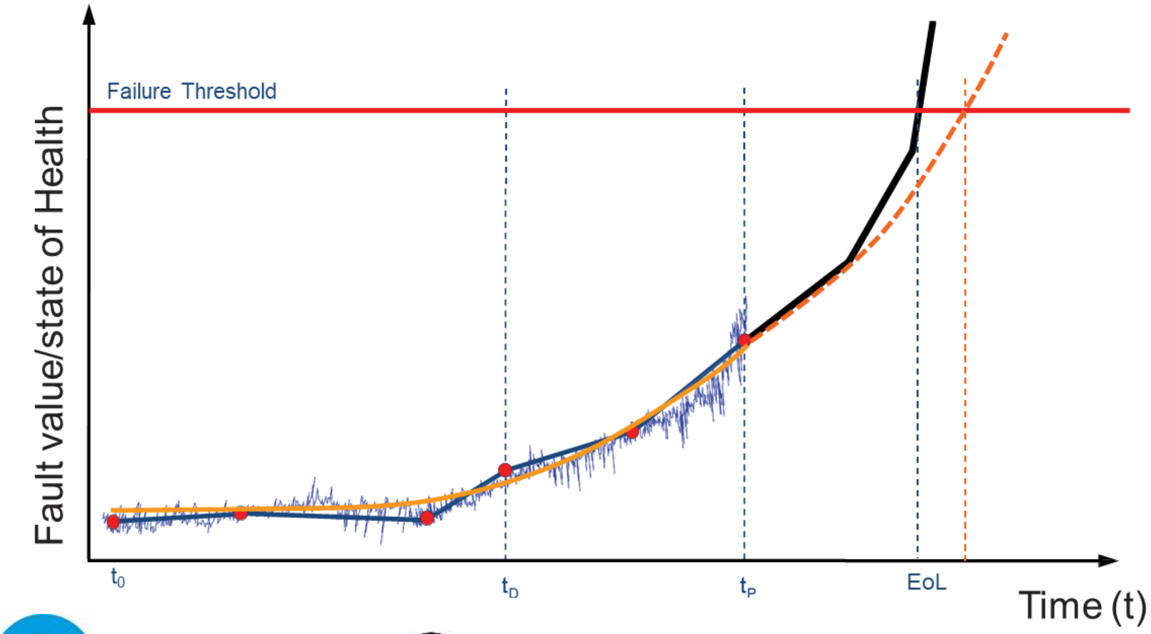
- Replaced RNNs for Identification of Non-linear systems (dynamical systems).
  - Benchmarking performance LSTM > RNN > MLP > CNN (different datasets/ factors)

(A Richard et al. 2019)

Application:
Prognostics and Deep Learning

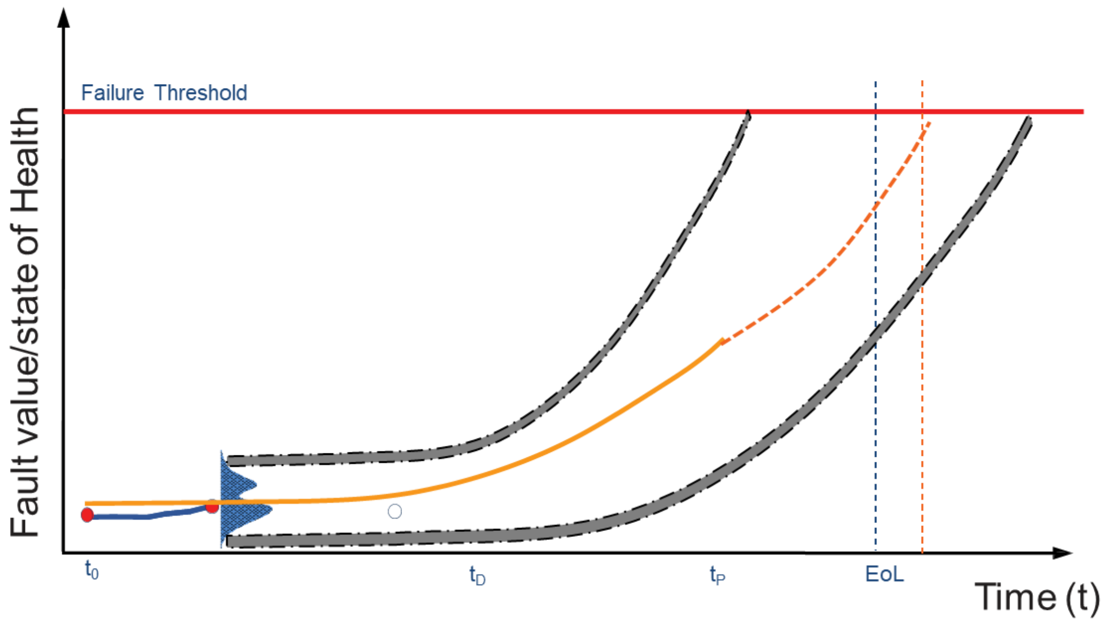
System degradation

- Machines (dynamical systems) degrade with:
  - time
  - operational load cycles
  - operational conditions etc.



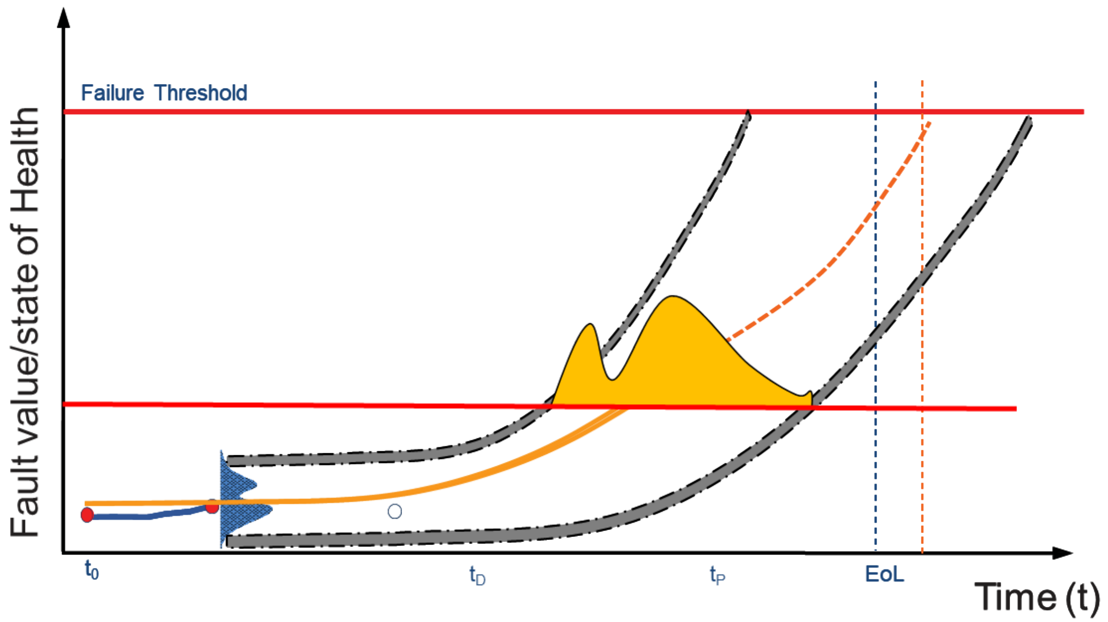
Prognostics

- Prognostics:
  - Estimate (state of health) → identification of degradation model.



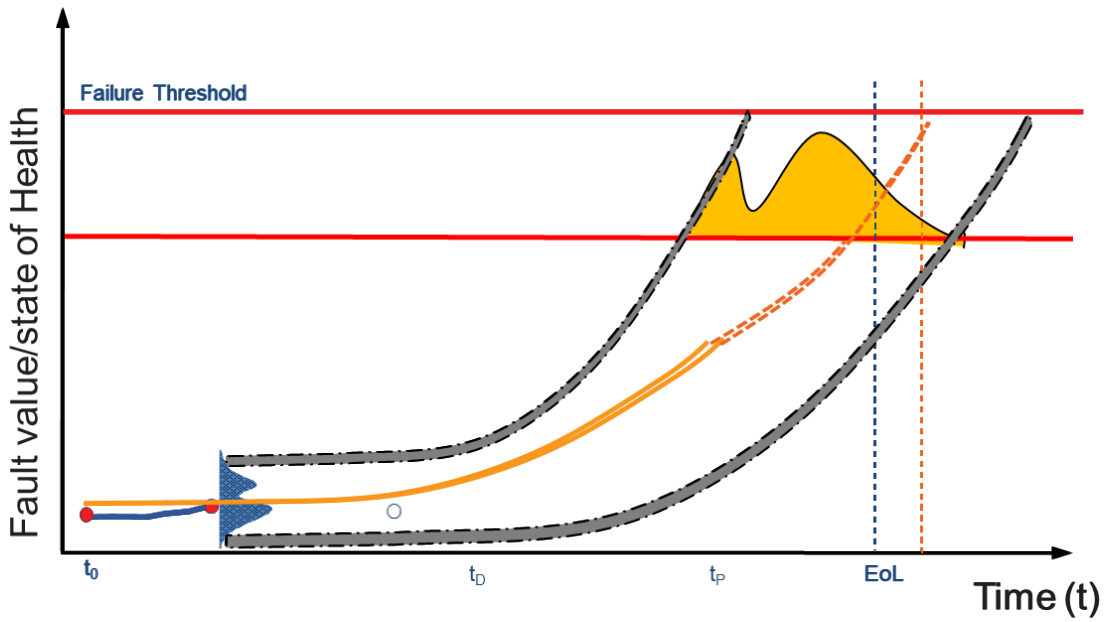
Prognostics

- Prognostics:
  - Estimate (state of health) → identification of degradation model.
  - Prediction of future health + Remaining Useful Life (RUL)



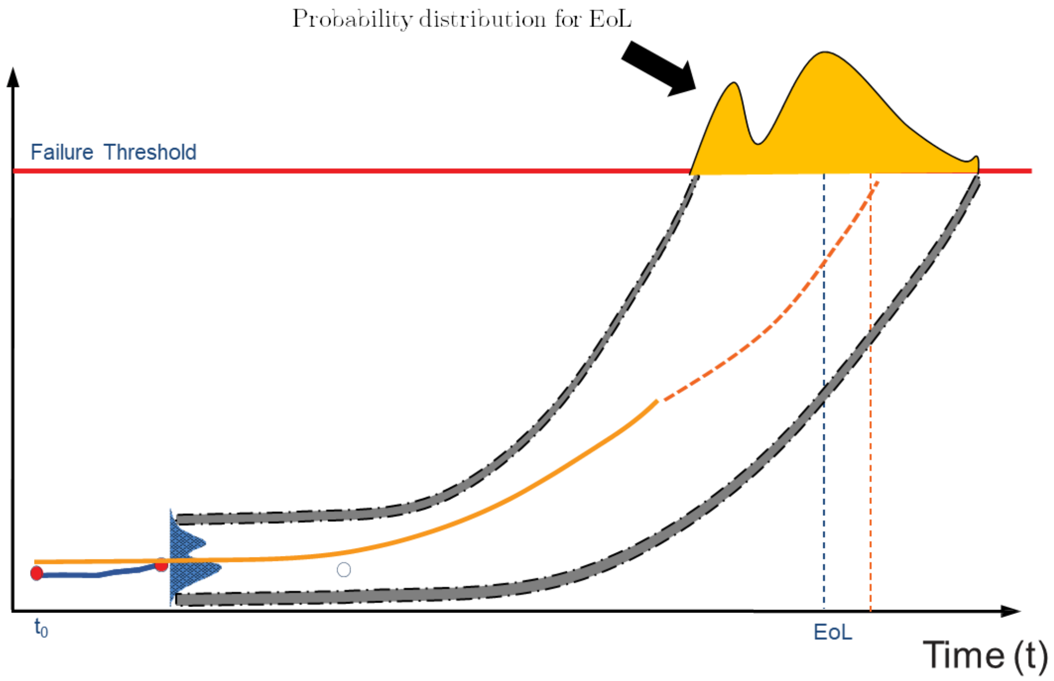
Prognostics

- Prognostics:
  - Estimate (state of health) → identification of degradation model.
  - Prediction of future health + Remaining Useful Life (RUL)



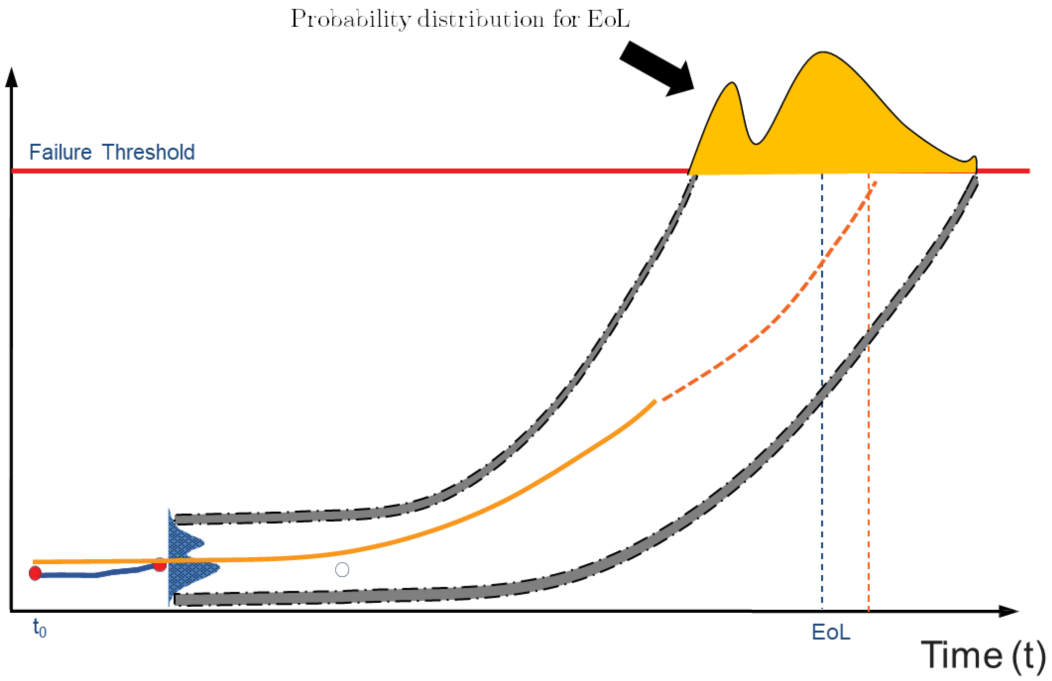
Prognostics

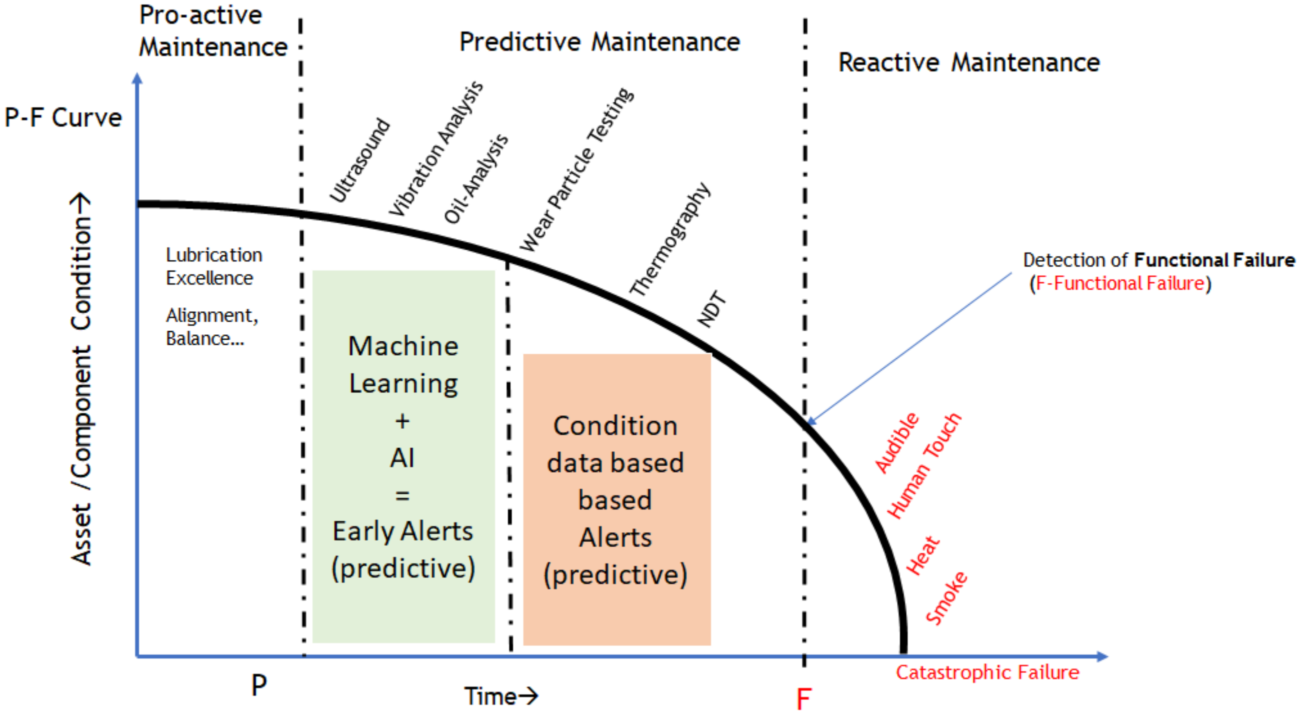
- Prognostics:
  - Estimate (state of health) → identification of degradation model.
  - Prediction of future health + Remaining Useful Life (RUL)



Prognostics

- Prognostics:
  - Estimate (state of health) → identification of degradation model.
  - Prediction of future health + **Remaining Useful Life (RUL)**
  - Evaluate: Decision "when failure occurs ???" "what maintenance strategy"



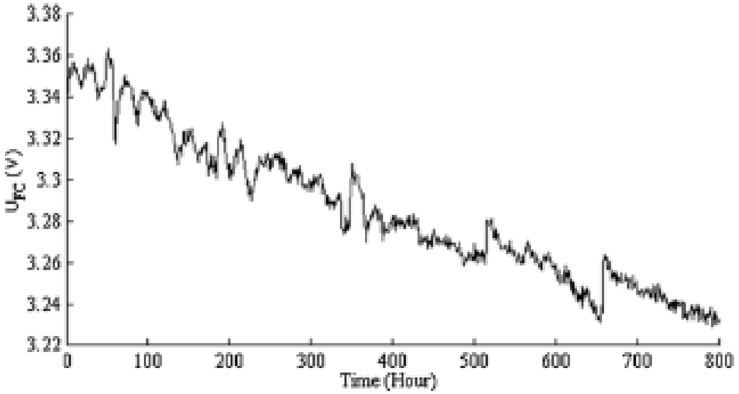


Degradation Data

- Degradation:
  - unknown, non-linear varying dynamics
  - sensor data: non-stationary process → trend, seasonality, cyclic etc.
  - depends on qualitative+ quantitative factors.

Degradation Data

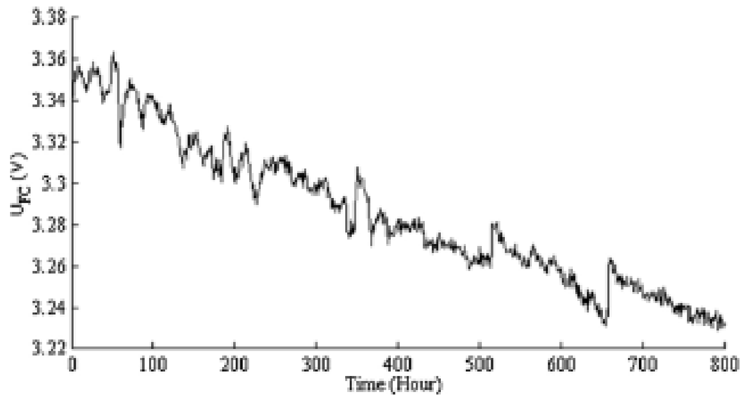
- Degradation:
  - unknown, non-linear varying dynamics
  - sensor data: non-stationary → trend, seasonality, cyclic etc.
  - depends on qualitative+ quantitative factors.



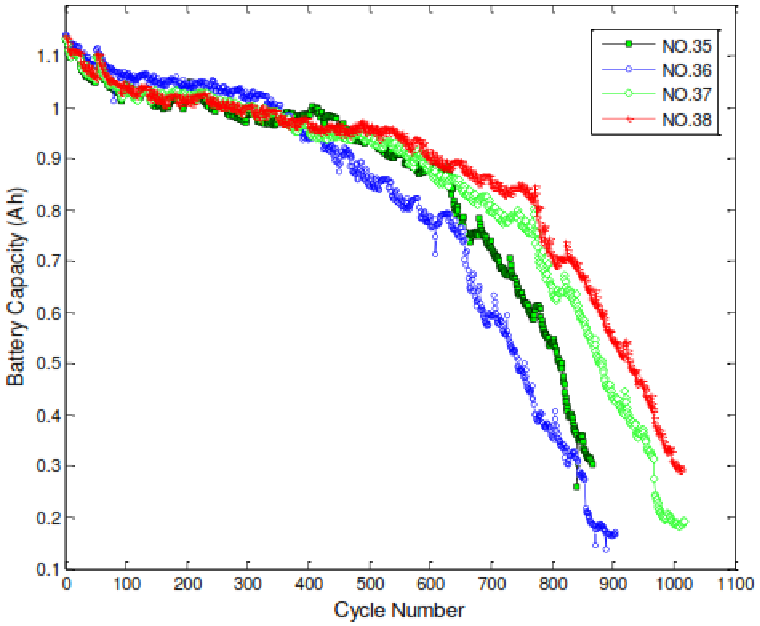
PEM Fuel Cell degradation (Jha et al. 2016)

Degradation Data

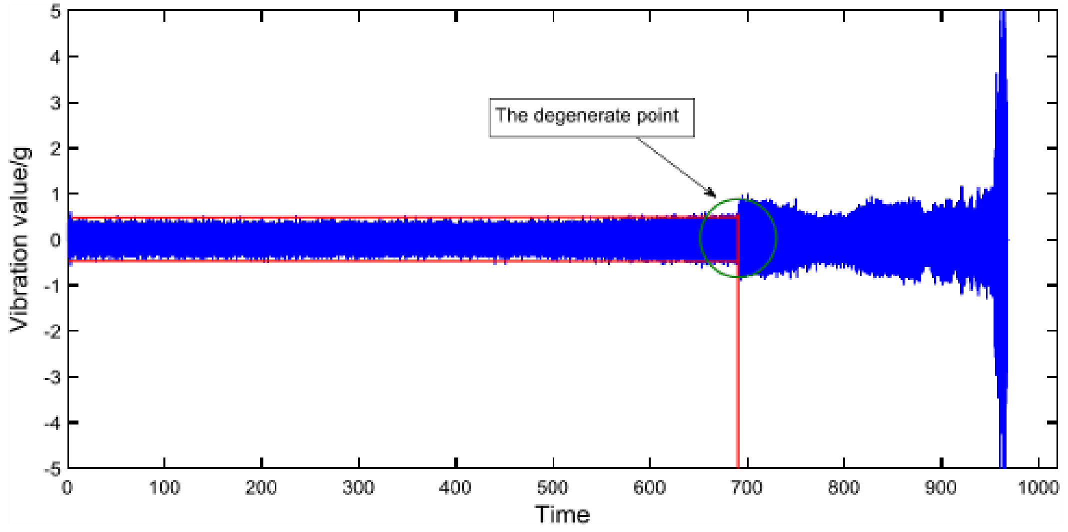
- Degradation:
  - unknown, non-linear varying dynamics
  - sensor data: non-stationary → trend, seasonality, cyclic etc.
  - depends on qualitative+ quantitative factors.



PEM Fuel Cell degradation (Jha et al. 2016)



Lithium-ion battery degradation,
Center for Advanced Life Cycle Engineering (CALCE) in University of Maryland (He W., Williard N., Osterman M., & Pecht M., 2011)

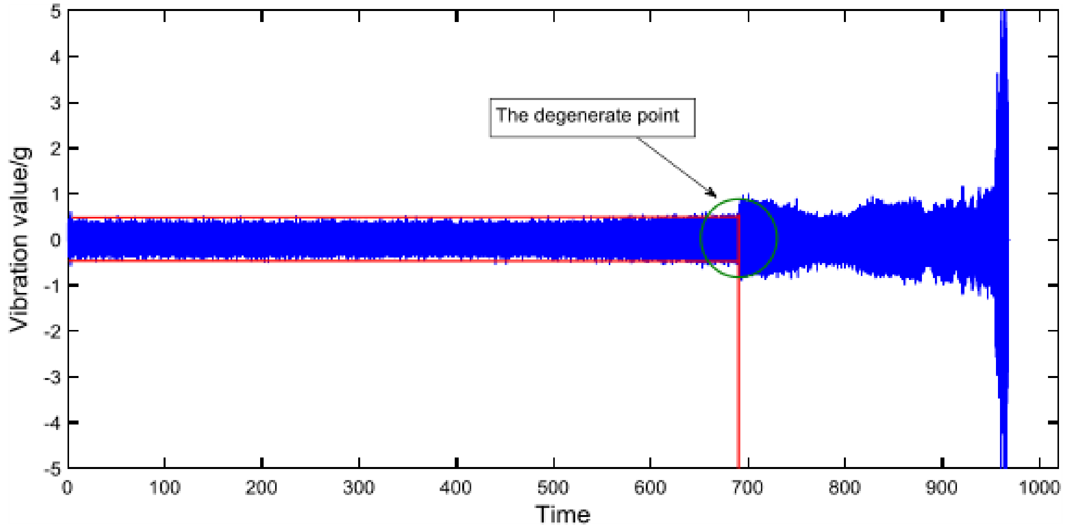


Roller bearing degradation (PRONOSTIA platform)

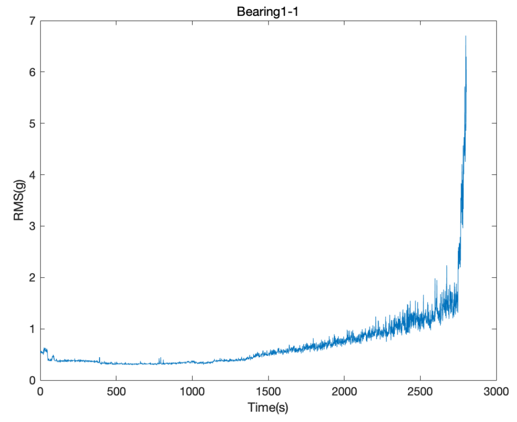
Degradation Data

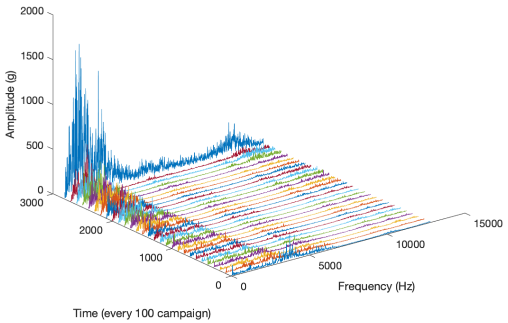
- Degradation:
  - unknown, non-linear varying dynamics
  - sensor data: non-stationary process → trend, seasonality, cyclic etc.
  - depends on qualitative+ quantitative factors.

- Raw degradation data → Hidden features / representation:
  - Spatially varying
  - Temporally varying
  - Multimodal characteristics



Roller bearing degradation (PRONOSTIA platform)





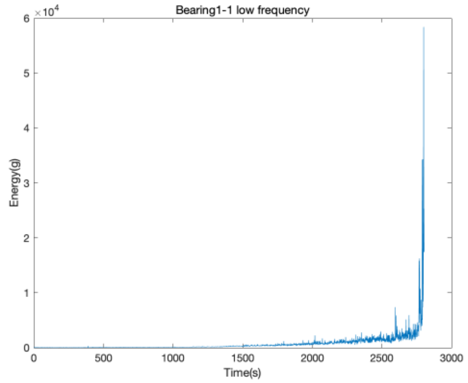


Photo: Report of Jha

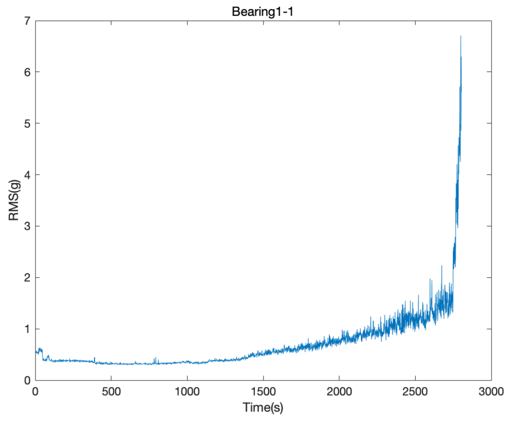
Degradation Data

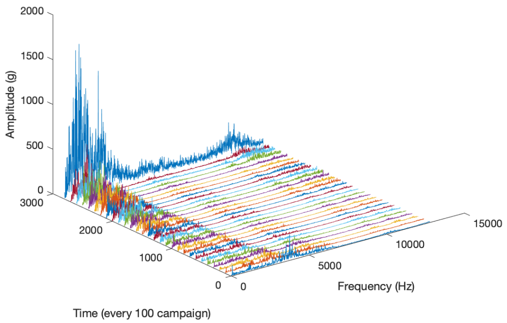
- Degradation:
  - unknown, non-linear varying dynamics
  - sensor data: non-stationary process → trend, seasonality, cyclic etc.
  - depends on qualitative+ quantitative factors.

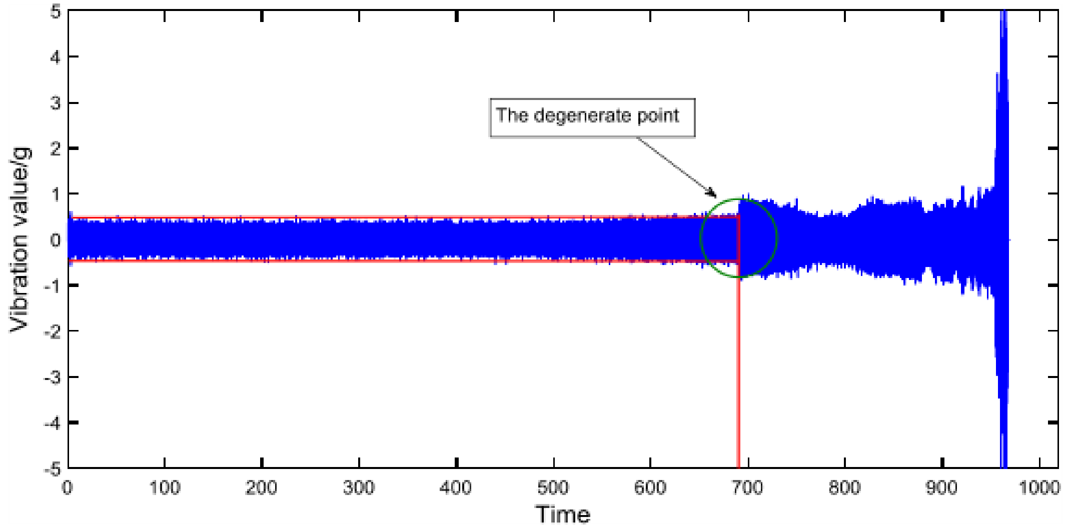
Deep LSTMs

- Raw degradation data → Hidden features / representation:
  - Spatially varying
  - Temporally varying
  - Multimodal characteristics

CNNs







Roller bearing degradation (PRONOSTIA platform)

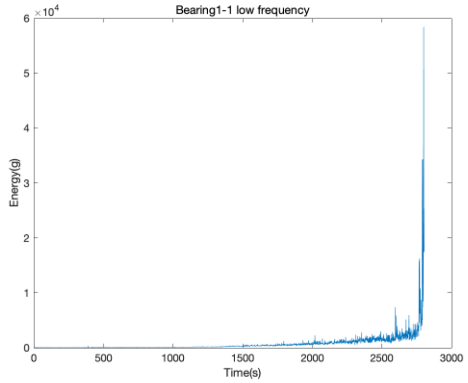
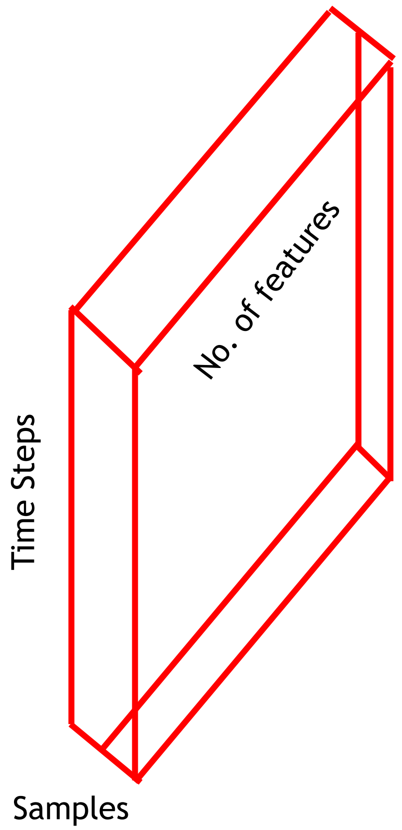


Photo: Report of Jha

Deep LSTMs for Prognostics

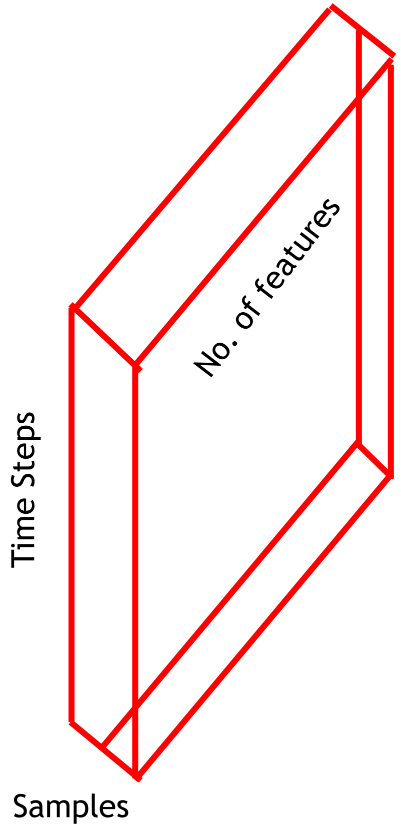
Basic Architecture



3D- Input

Deep LSTMs for RUL prediction

Basic Architecture



3D- Input

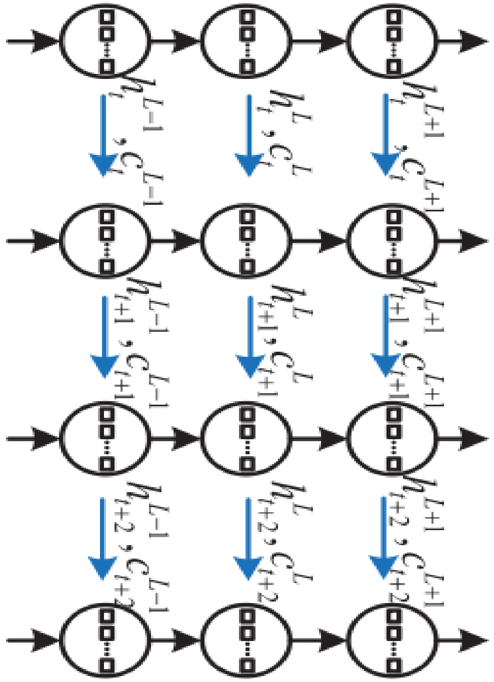
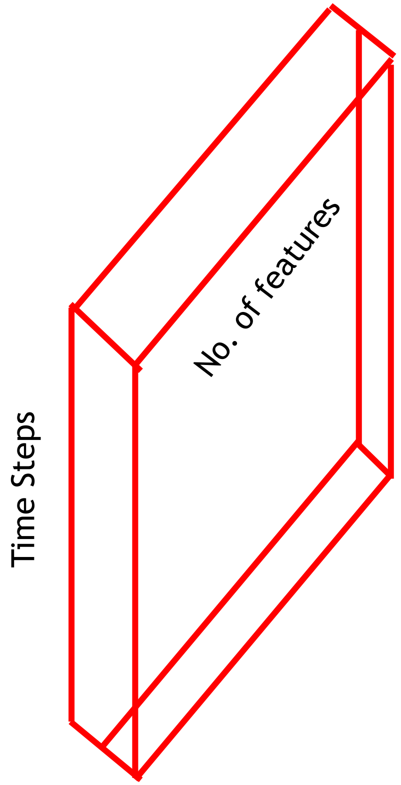


Image credits: Fernández, Graves, & Schmidhuber,2007

Deep LSTM
+ dropout schemes

Deep LSTMs for RUL prediction

Basic Architecture: LSTMs: Temporal features + FNNs: Map features in RULs



3D- Input

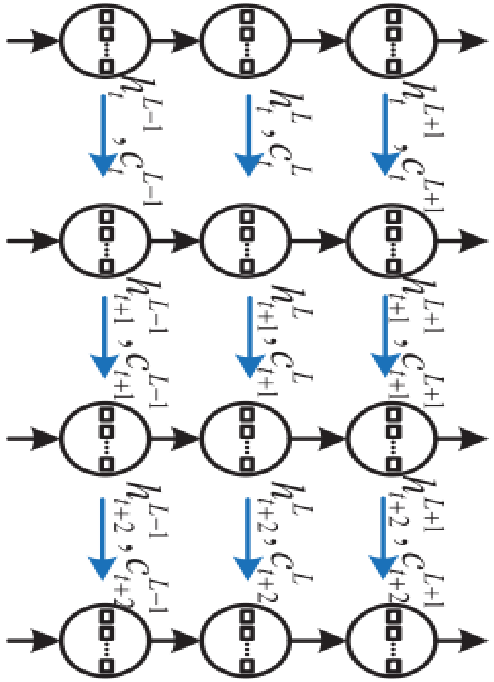
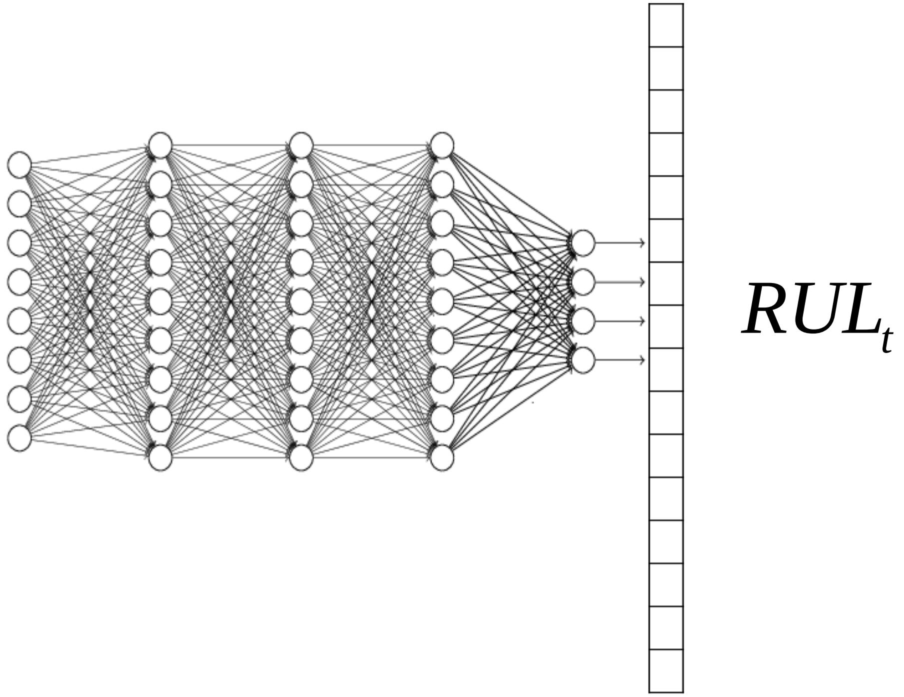


Image credits: Fernández, Graves, & Schmidhuber,2007

Deep LSTM
+ dropout schemes



Fully connected Layer

Target Vector

Deep LSTMs for RUL prediction

- Degradation data➔ Time Series sequence ➔ segmented into sliding windows.
- Each sliding window is assigned a target RUL value [Zeng et al, 2017]

$X = [X_1, X_2, \ldots, X_t, \ldots X_{T-1}]$ to estimate RUL_{T-1}

$X = [X_1, X_2, \ldots, X_t, \ldots X_{T-2}]$ to estimate RUL_{T-2}

Deep LSTMs for RUL prediction

- Degradation data → Time Series sequence → segmented into sliding windows.
- Each sliding window is assigned a target RUL value [Zeng et al, 2017]

$X = [X_1, X_2, ..., X_t, ... X_{T-1}]$ to estimate RUL_{T-1}

$X = [X_1, X_2, ..., X_t, ... X_{T-2}]$ to estimate RUL_{T-2}

$[X_t, X_{t-1}, \ldots, X_{t-d+1}], \epsilon \mathbb{R}^d$

$[\mathrm{RUL}_{t+L}, \mathrm{RUL}_{t+L+1}, \ldots, \mathrm{RUL}_n]$

Many variants exist!

Training tuples:

$([X_t, X_{t-1}, \ldots, X_{t-d+1}], \mathrm{RUL}_{t+L})$

$\widehat{\mathrm{RUL}_{t+L}} = \phi(X_t, X_{t-1}, \ldots, X_{t-d+1})$

Loss Calculation : Error based cost function

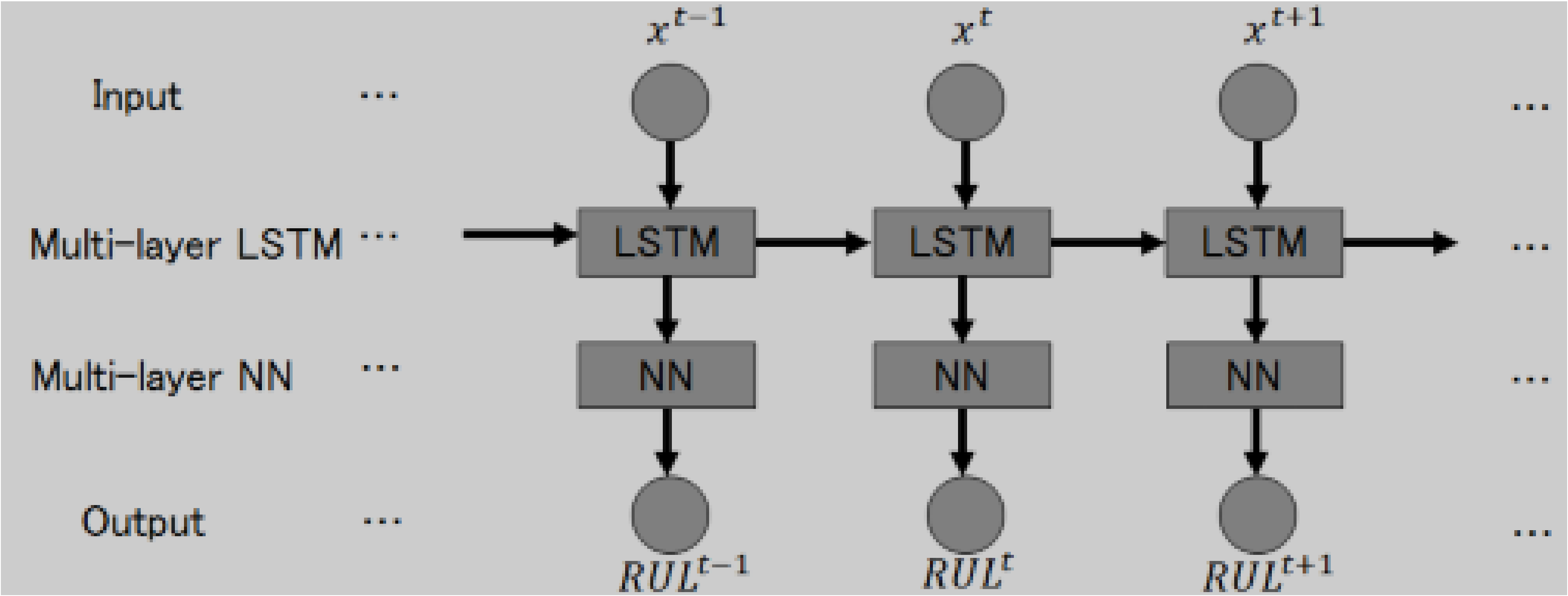
$$J = \sum_t \|(RUL_{est}^t - RUL_{calc}^t\|^2$$

Some issues:

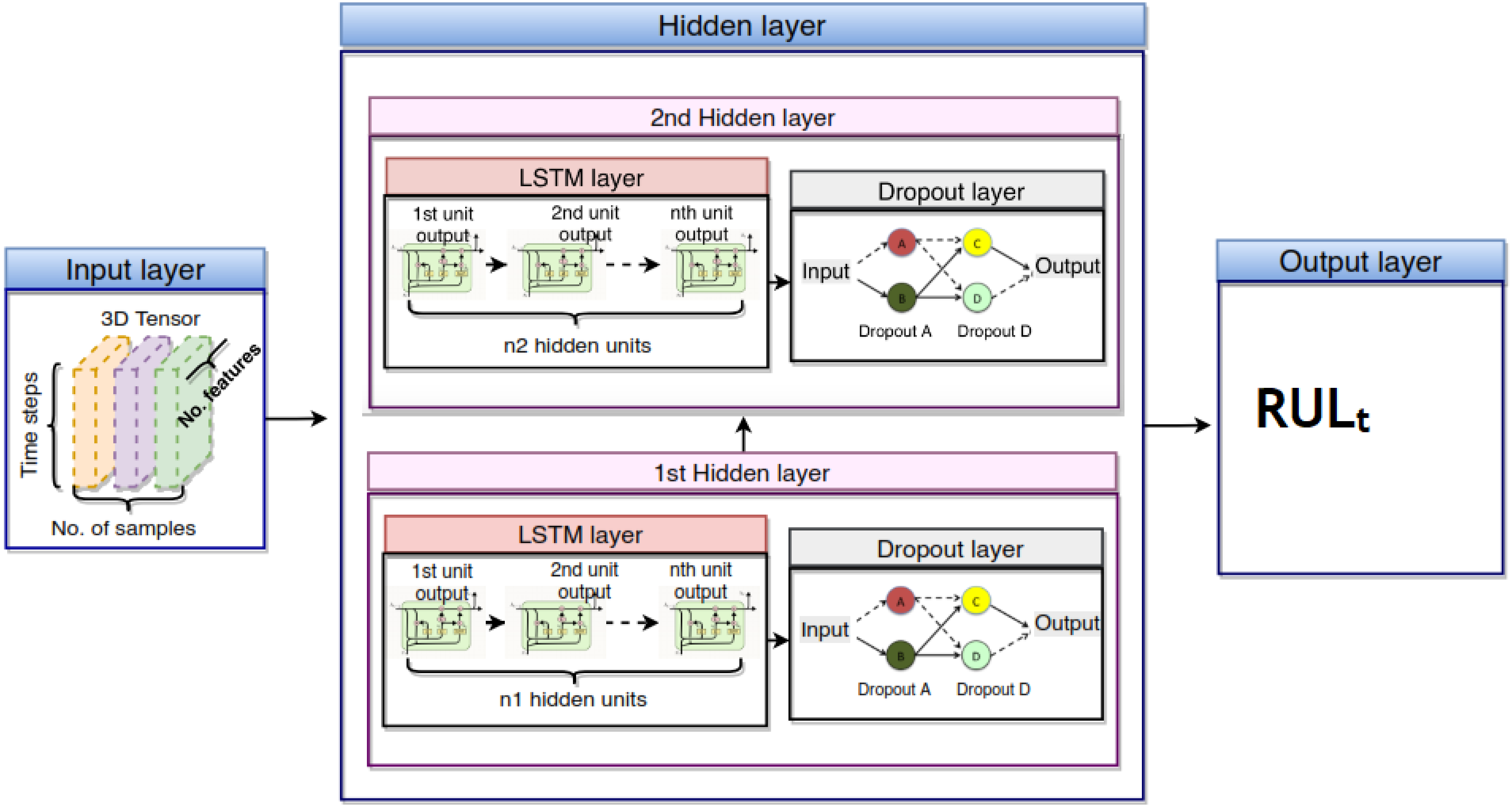
- Independent Windows → to assure assumption of i.i.d
- Dependent windows → claim more realistic.

LSTM training

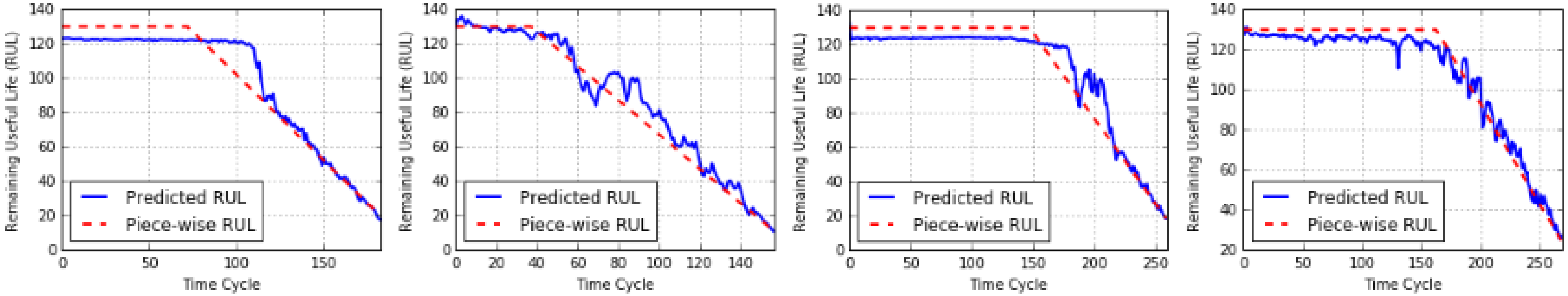
- Inputs : Sensor data at time *t*
- *Output*: RUL at time *t*



RUL prediction training How??



RUL prediction Example: C-MAPSS dataset (NASA)



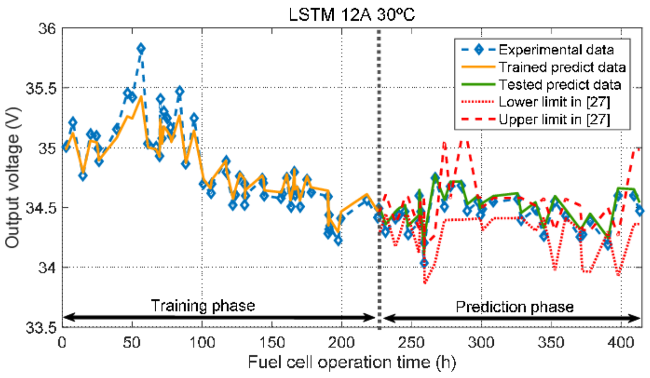
(a) FD001 example (b) FD002 example (c) FD003 example (d) FD004 example

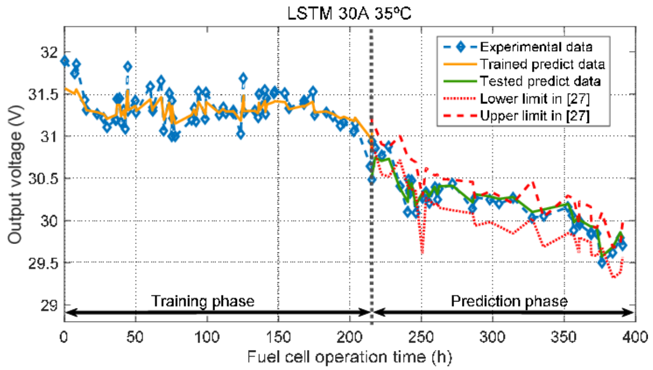
RUL estimation at each time stamp using Deep LSTM (use sequence $X = [\mathbf{x}^1, ..., \mathbf{x}^t, ..., \mathbf{x}^{T-1}]$ to estimate RUL at time $T-1$, with the true RUL as $RUL+1$; use sequence $X = [\mathbf{x}^1, ..., \mathbf{x}^t, ..., \mathbf{x}^{T-2}]$ to estimate RUL at time $T-2$, with the true RUL as $RUL+2$, etc..)

MAPSS stands for 'Commercial Modular Aero-Propulsion System Simulation' and it is a tool for the simulation of realistic large commercial turbofan engine data.

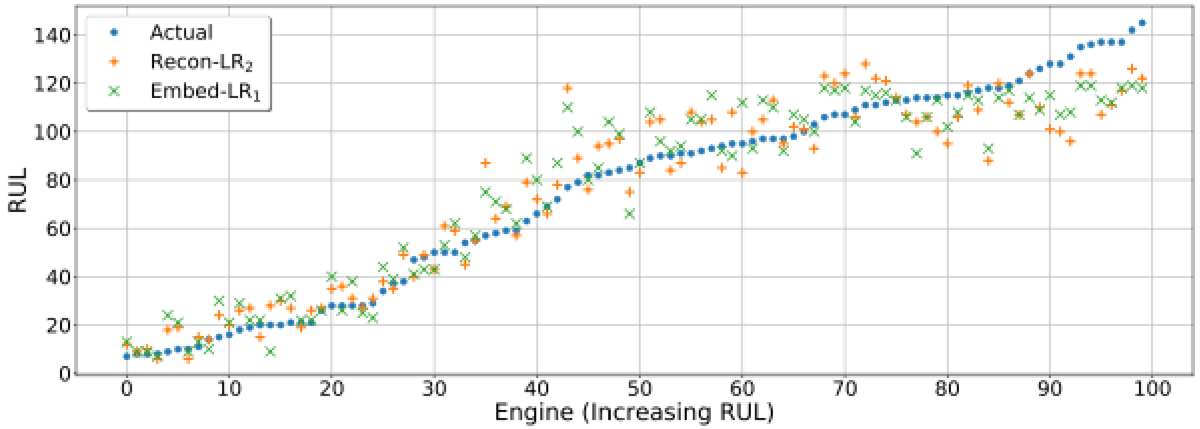
The fault was injected at a given time in one of the flights and persists throughout the remaining flights, effectively increasing the age of the engine.

Some applications:





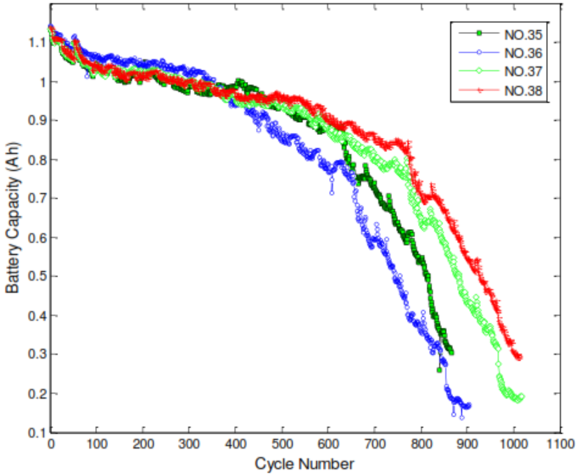
PEM Fuel Cell degradation

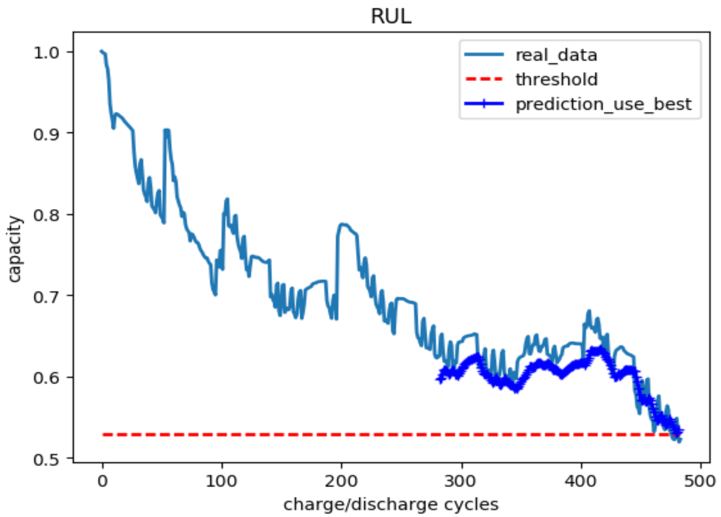


[Gugulothu et al 2017]

Engine prognostics (NASA) : CMAPSS
'Commercial Modular Aero-Propulsion System Simulation'
[Zhang et al, 2017]

- unknown non-linear dynamics,
- non-stationary (multi modal degradation,
- multiple modes of degradation)





Lithium-ion battery RUL prediction

(He W., Williard N., OstermanM., & Pecht M., 2011)

CNNs for Prognostics

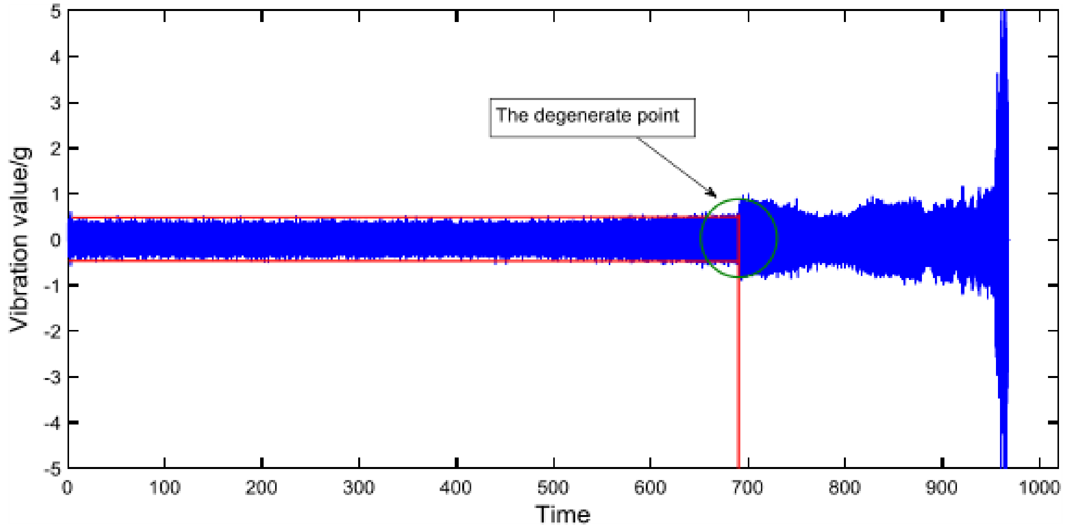
- LSTMs: good sequence learning

but good input sequence needs to be provided!!

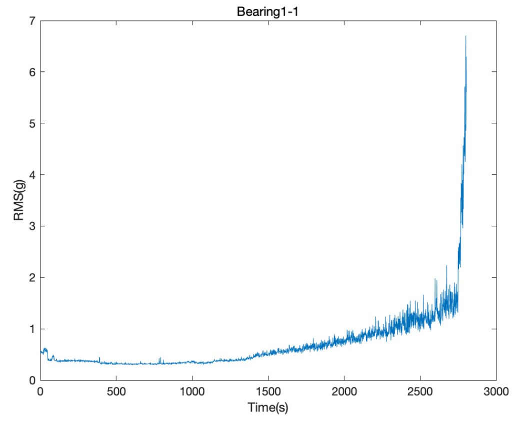
- Feature extraction needs domain knowledge.
- Labelled data → difficult !

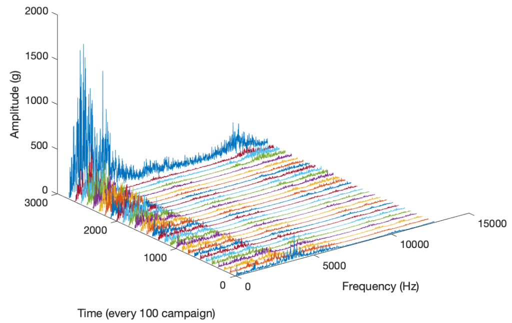
- CNNs → Hidden features / representation of sequence:
  - Spatially varying
  - Temporally varying
  - Multimodal characteristics

CNNs



Roller bearing degradation (PRONOSTIA platform)





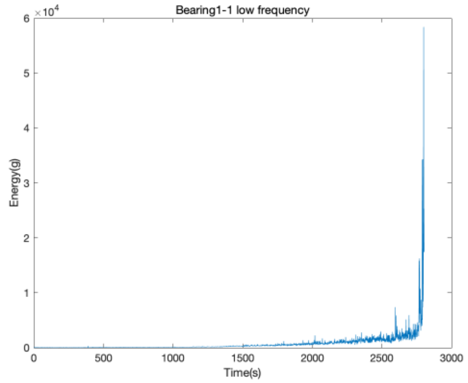
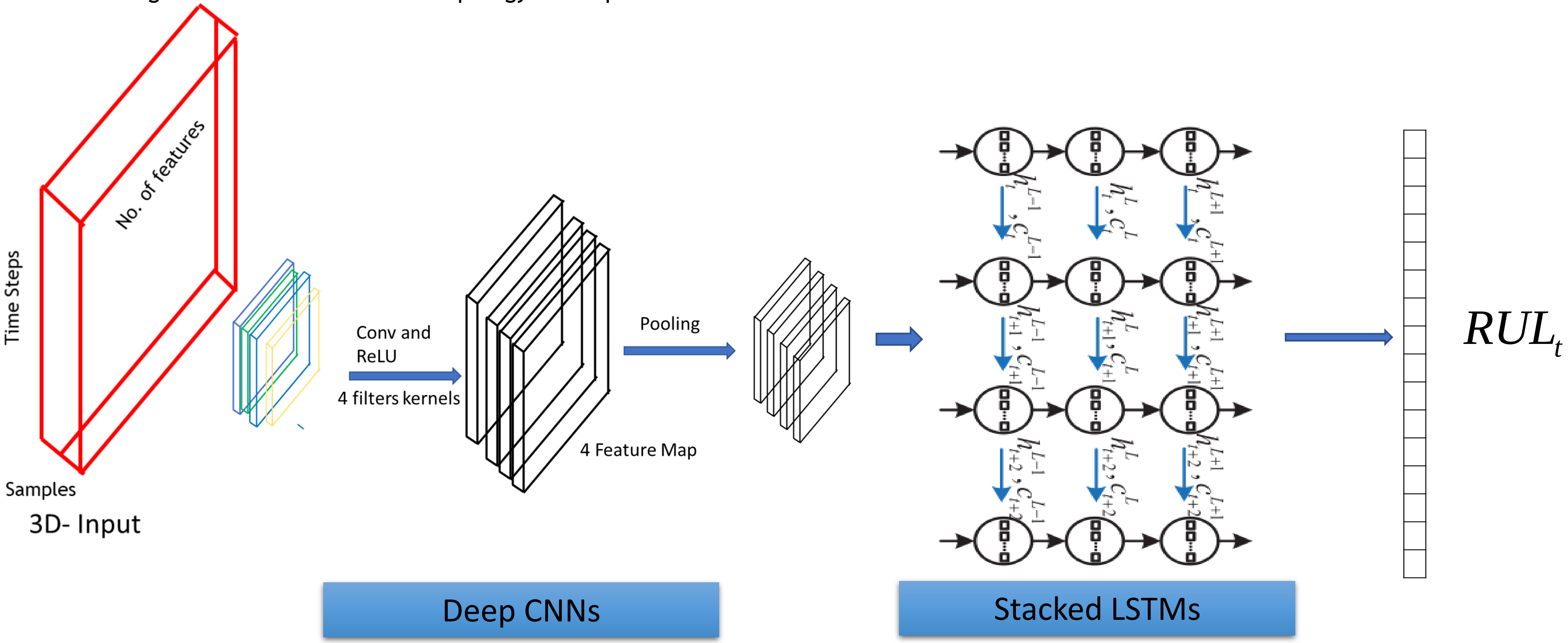


Photo: Report of Jha

JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

CNNs for Prognostics

- CNNs → Traditionally, 2D-3D structured data for face/object recognition
- Prognostics → 3D structured topology for sequence data



CNNs for Prognostics

- Automatically learn feature representation, hidden multimodal distributions

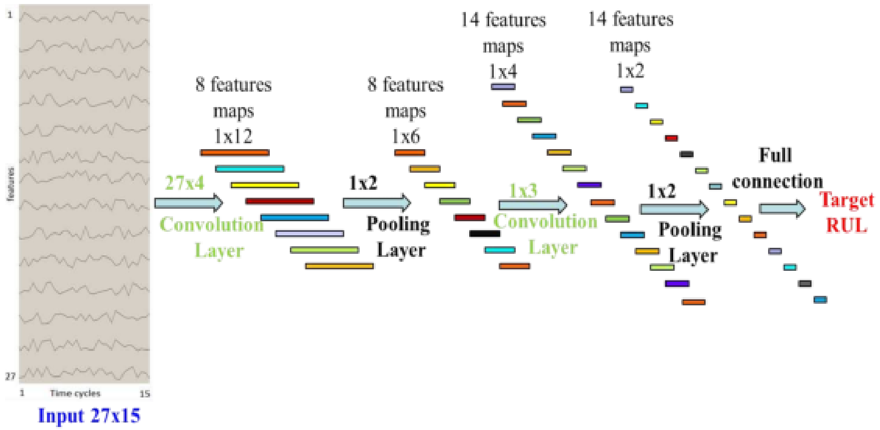
[Liu et al., 2017] [Jing et al., 2017] [Li et al., 2018]

&

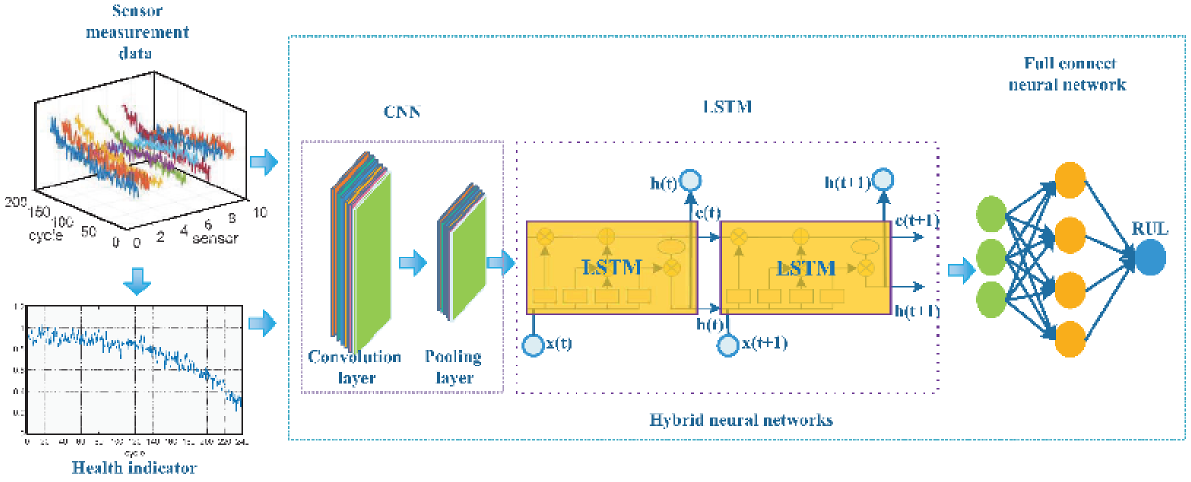
- Efficient learning with multi-variate sequential (time series) data.

[Babu et al., 2016]

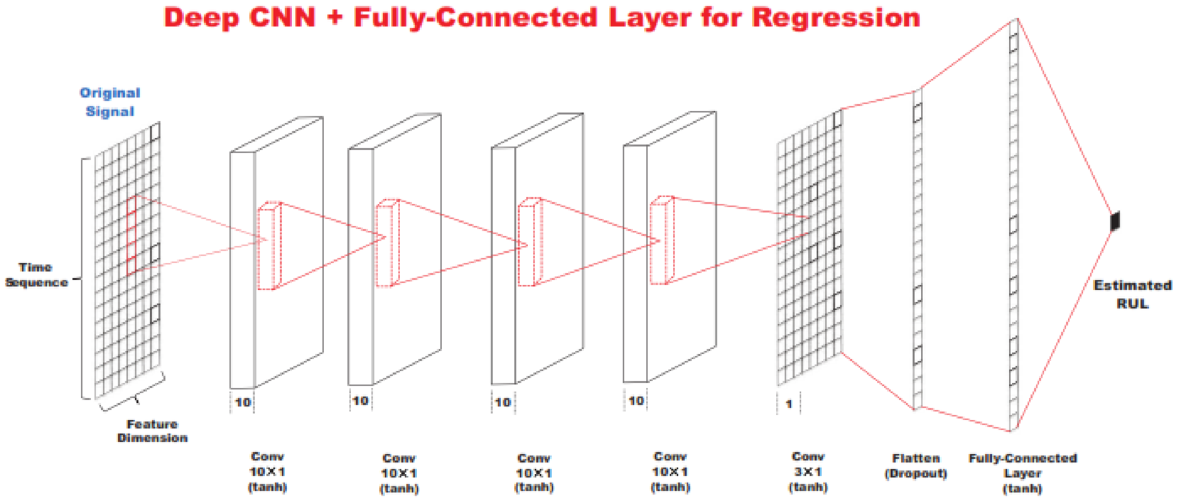
- Hybrid structure



[Babu et al., 2016]

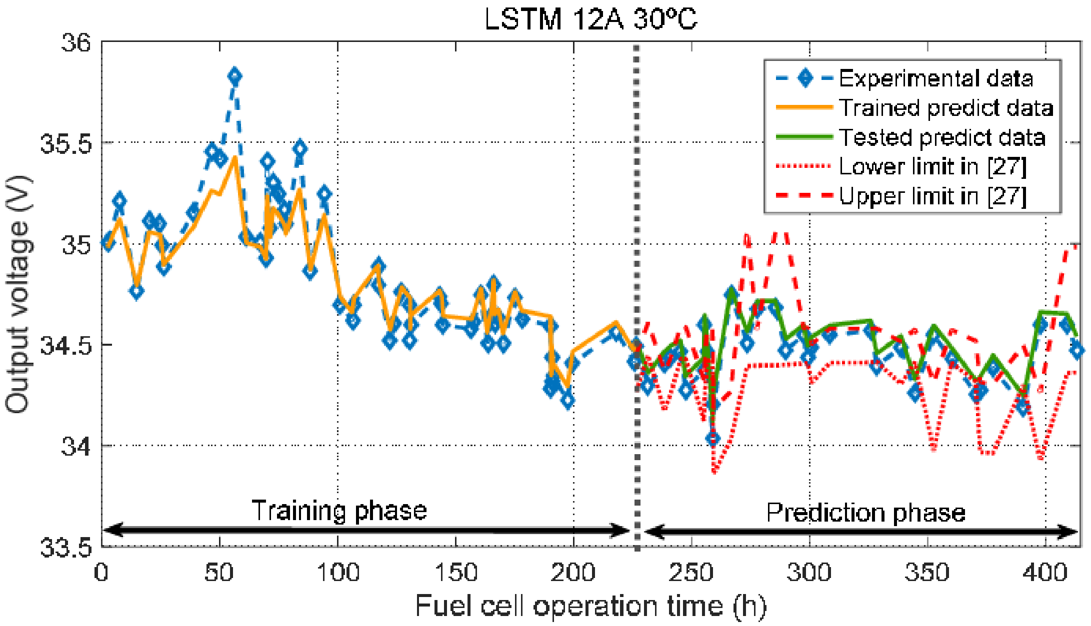


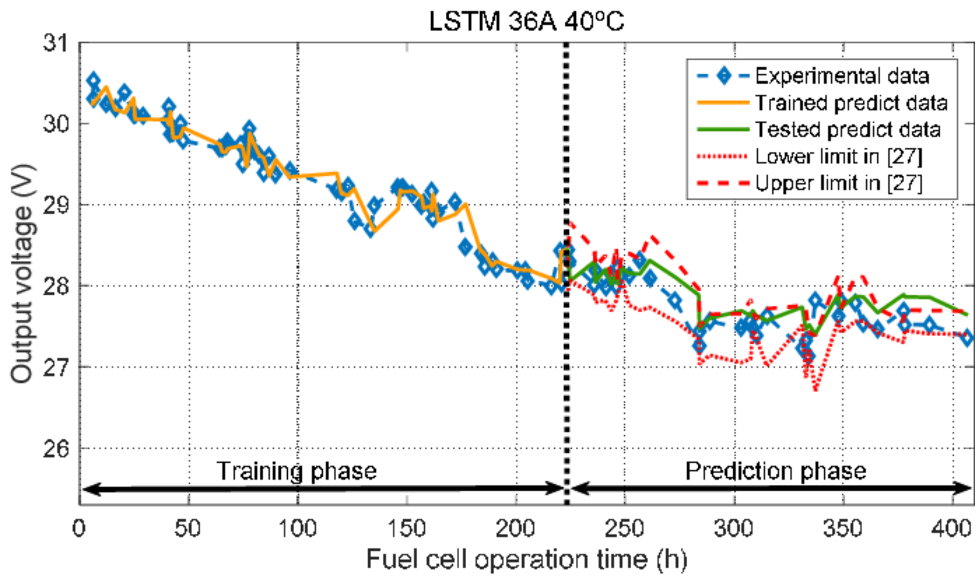
[Kong et al. 2019]

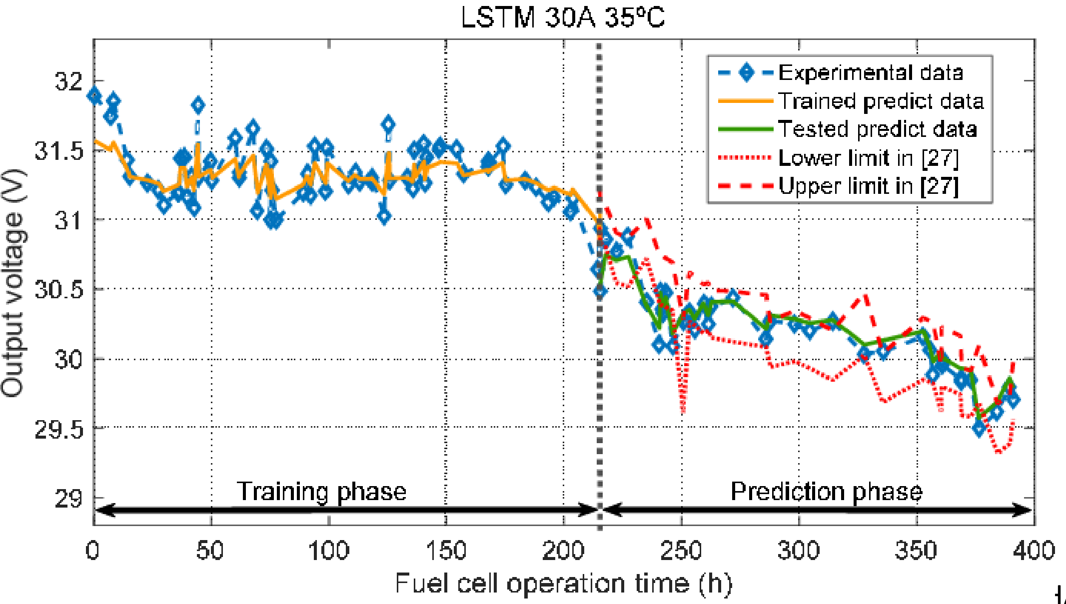


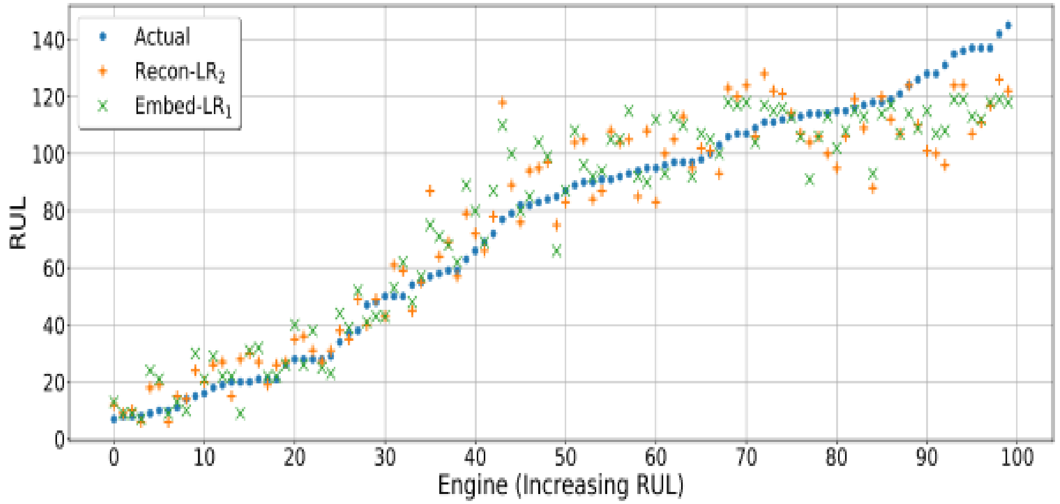
[Liu et al., 2017]

PEM Fuell Cell Degradation



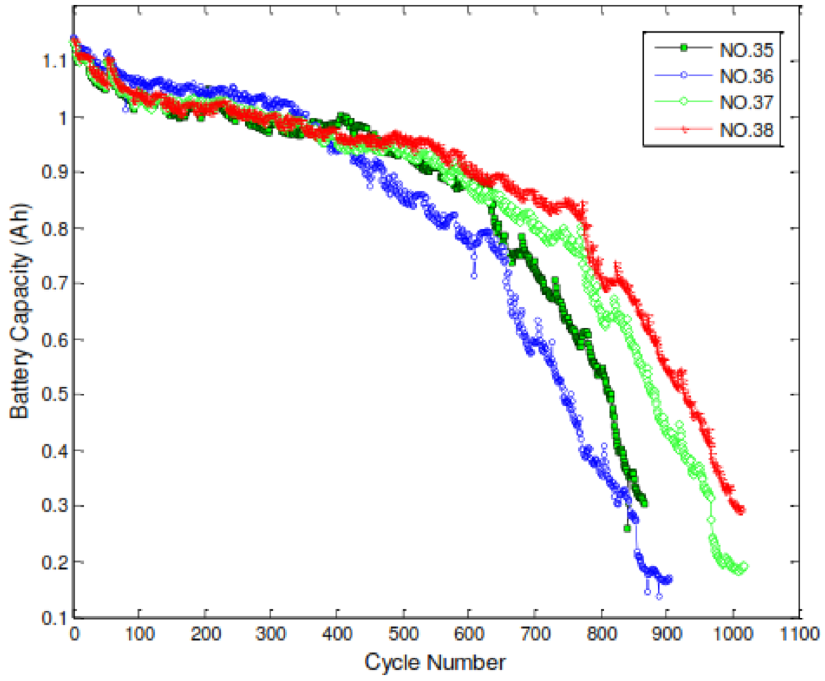






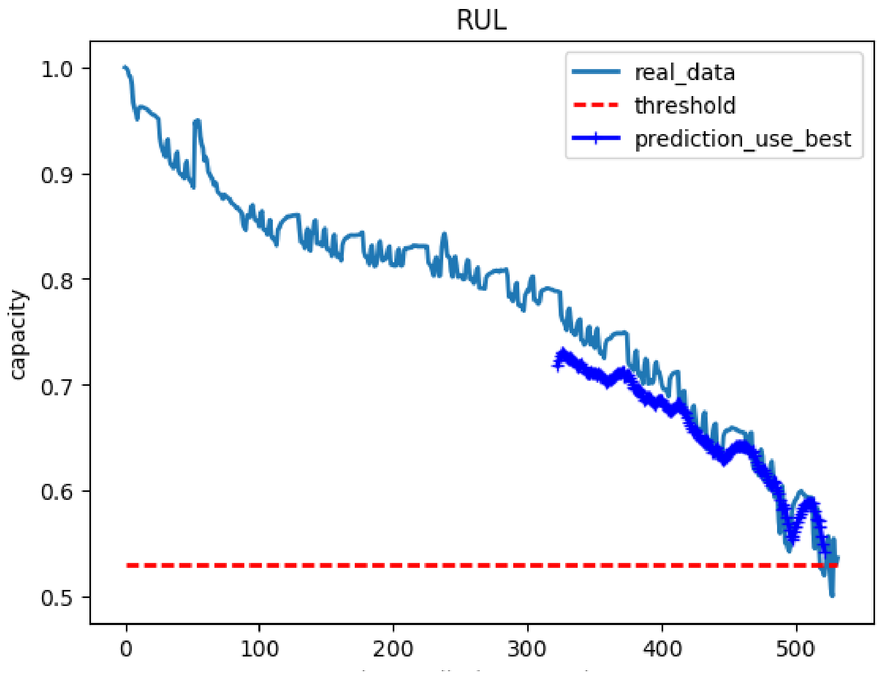
Gugulothu et al.

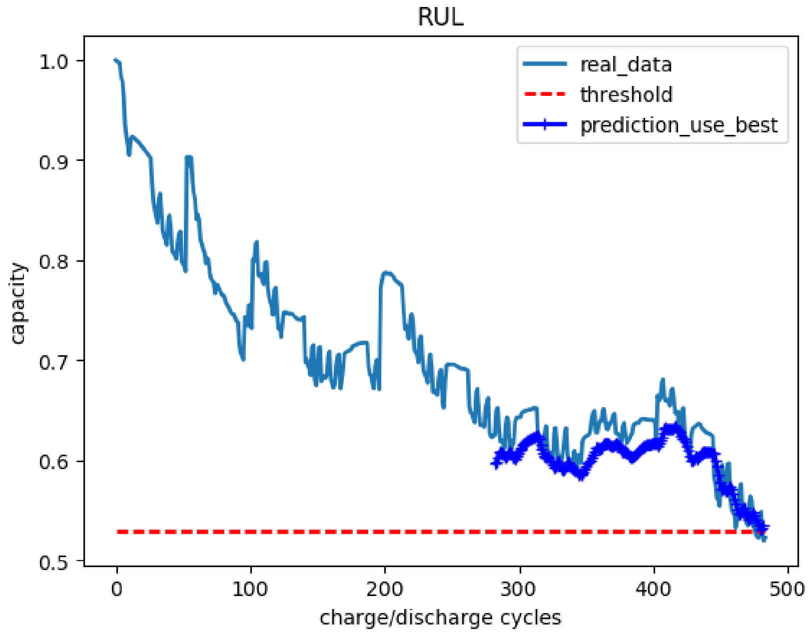
Battery degradation RUL prediction ((He W., Williard N., OstermanM., & Pecht M., 2011)



Lithium-ion battery RUL prediction

LSTM based RUL prediction





POLYTECH NANCY
UNIVERSITÉ DE LORRAINE
CRAN

JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

Basic Approach

Raw Data

⇩

Data Preparation :
- fill missing values
- data sorting..
- Test set, Train Set

⇩

Dimensionality Reduction

Principle Component Analysis (PCA),
t-Distributed Stochastic Neighbor Embedding (t-SNE)

⇩

Feature Extraction

- Time domain
- frequency (FFT)
- Time-frequency (Wavelet transforms)
- STFFT

⇩

Supervised /Unsupervised Training of Classifier

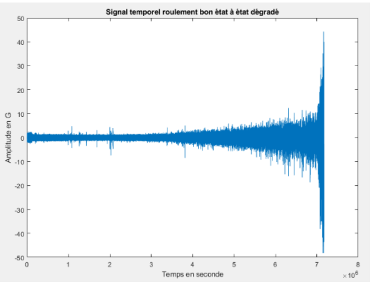
⇩

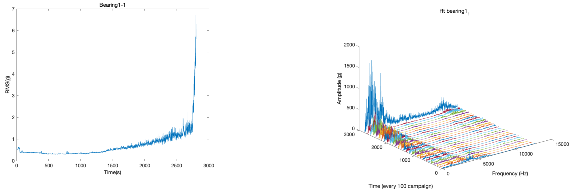
Cross validation

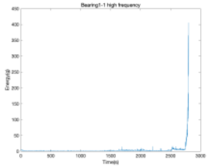
⇩

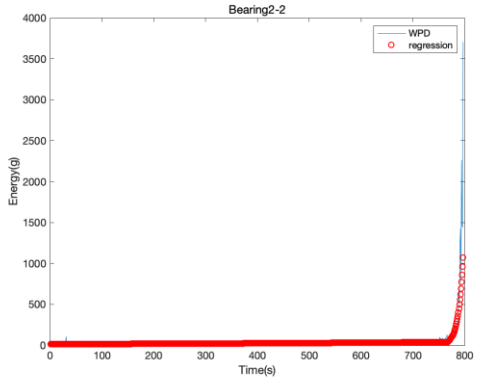
Trained classifier Model

Offline









New Data → Classification/Prediction → Decision / Output

AI enhances Tribology , KTH, Sweden
Email: *mayank-shekhar.jha [at] univ-lorraine.fr*

References

Kattan, Ali, and Rosni Abdullah. "Training of feed-forward neural networks for pattern-classification applications using music inspired algorithm." *International Journal of Computer Science and Information Security* 9.11 (2011): 44.

Taig, Efrat, and Ohad Ben-Shahar. "Gradient Surfing: A New Deterministic Approach for Low-Dimensional Global Optimization." *Journal of Optimization Theory and Applications* 180.3 (2019): 855-878.

Guest, D., Cranmer, K., & Whiteson, D. (2018). Deep learning and its application to LHC physics. *Annual Review of Nuclear and Particle Science*, *68*, 161-181.

Sirunyan, A. M., Tumasyan, A., Adam, W., Ambrogi, F., Asilar, E., Bergauer, T., ... & Del Valle, A. E. (2019). Search for the Higgs boson decaying to two muons in proton-proton collisions at s= 13 TeV. *Physical review letters*, *122*(2), 021801.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). **Human-level control through deep reinforcement learning**. *Nature*, *518*(7540), 529.

Cully, Antoine, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. 2015. “Robots That Can Adapt like Animals.” ***Nature*** 521 (7553): 503.

Park, Y., & Kellis, M. (2015). Deep learning for regulatory genomics. *Nature biotechnology*, *33*(8), 825.

Gugulothu, N., TV, V., Malhotra, P., Vig, L., Agarwal, P., & Shroff, G. (2017). Predicting remaining useful life using time series embeddings based on recurrent neural networks. *arXiv preprint arXiv:1709.01073*.

Sutton, Richard S., and Andrew G. Barto. *Introduction to reinforcement learning*. Vol. 135. Cambridge: MIT press, 1998.

Bibliography

- Rueckert, E., Nakatenus, M., Tosatto, S., & Peters, J. (2017, November). Learning inverse dynamics models in o (n) time with lstm networks. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)* (pp. 811-816). IEEE.
- Fernández, S., Graves, A., & Schmidhuber, J. (2007b). An application of recurrent neural networks to discriminative keyword spotting. In *Proceedings of the International Conference on Artificial Neural Networks (pp. 220–229). Berlin: Springer*
- Liu, N., Li, L., Hao, B., Yang, L., Hu, T., Xue, T., & Wang, S. (2019). Modeling and simulation of robot inverse dynamics using LSTM-based deep learning algorithm for smart cities and factories. *IEEE Access*, *7*, 173989-173998.
- Gugulothu, N., TV, V., Malhotra, P., Vig, L., Agarwal, P., & Shroff, G. (2017). Predicting remaining useful life using time series embeddings based on recurrent neural networks. *arXiv preprint arXiv:1709.01073*.
- *Systems*. Springer, Cham, 2017. 233-270Jha, Mayank-Shekhar; 2017. “Algorithm Architectures for Intelligent Communications, Control and Monitoring Systems, **Rolls Royce University Technology Centre** Sheffield.”
- Jha, Mayank Shekhar, et al. "Particle filter based hybrid prognostics of proton exchange membrane fuel cell in bond graph framework." *Computers & Chemical Engineering* 95 (2016): 216-230.
- Jha, Mayank S., Geneviève Dauphin-Tanguy, and B. Ould-Bouamama. "Particle Filter Based Integrated Health Monitoring in Bond Graph Framework." *Bond Graphs for Modelling, Control and Fault Diagnosis of Engineering* .

Bibliography

- Learning algorithms for classification: A comparison on handwritten digit recognition. Neural networks Stat Mech Perspect 261:276
- Simonyan K, Zisserman A (2015) VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. ICLR 75:398–406. doi: 10.2146/ajhp170251
- Lin M, Chen Q, Yan S (2013) Network In Network. 1–10. doi: 10.1109/ASRU.2015.7404828
- He K, Zhang X, Ren S, Sun J (2015) Deep Residual Learning for Image Recognition. Multimed Tools Appl 77:10437–10453. doi: 10.1007/s11042-017-4440-4
- Khan, A., Sohail, A., Zahoora, U. *et al.* A survey of the recent architectures of deep convolutional neural networks. *Artif Intell Rev* **53,** 5455-5516 (2020)
- Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp 249–256
- Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
- Richard, A., Mahé, A., Pradalier, C., Rozenstein, O., & Geist, M. (2019). A Comprehensive Benchmark of Neural Networks for System Identification.
- Woo, J., Park, J., Yu, C., & Kim, N. (2018). Dynamic model identification of unmanned surface vehicles using deep learning network. *Applied Ocean Research*, *78*, 123-133.

Bibliography

- Zheng, S., Ristovski, K., Farahat, A., & Gupta, C. (2017, June). Long short-term memory network for remaining useful life estimation. In *2017 IEEE international conference on prognostics and health management (ICPHM)* (pp. 88-95). IEEE.
- Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.