Introduction to Deep Learning

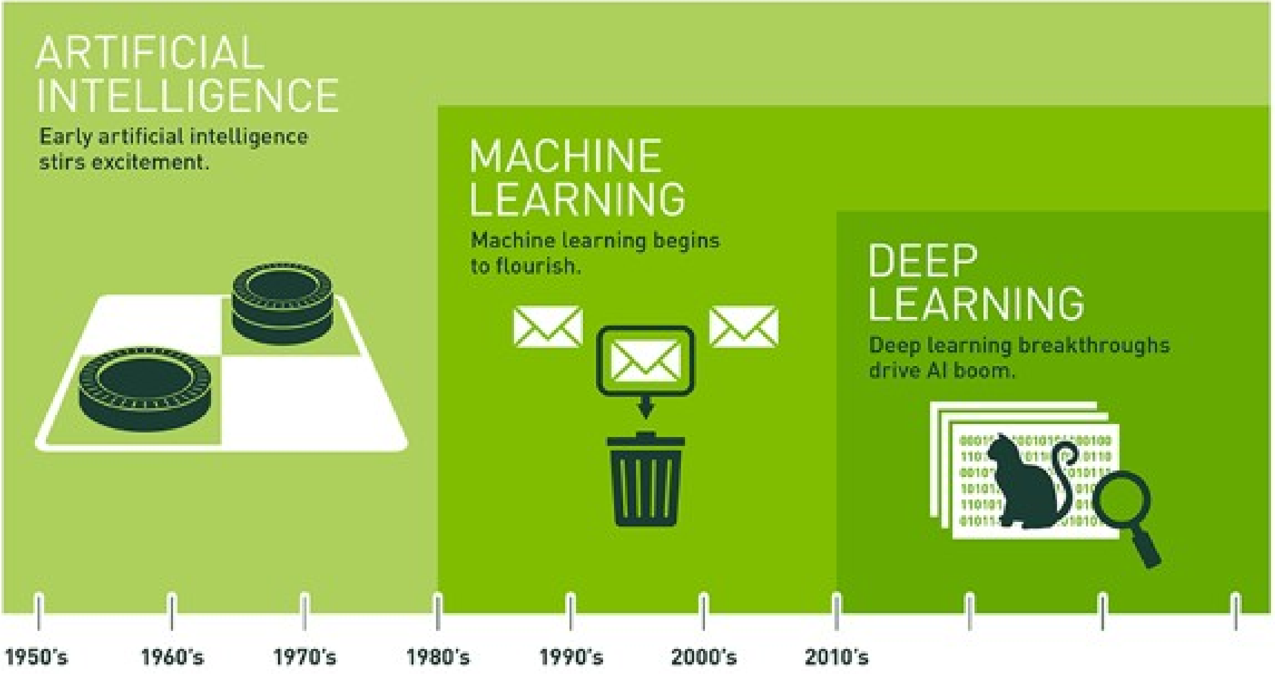
A gentle introduction

Massimiliano Ruocco
Adj. Associate Professor, DART Group, IDI
Senior Researcher, Sintef Digital

massimiliano.ruocco@ntnu.no

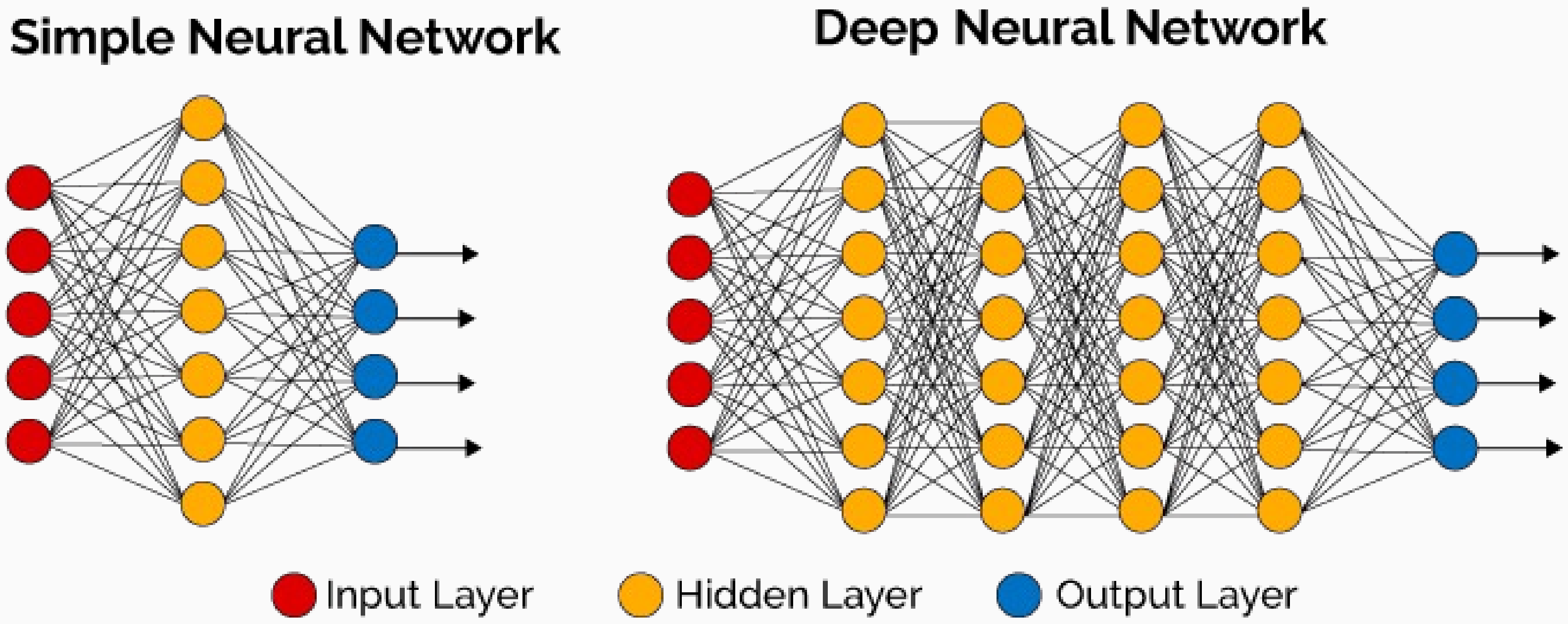
Introduction

Introduction: *AI vs ML vs DL*



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

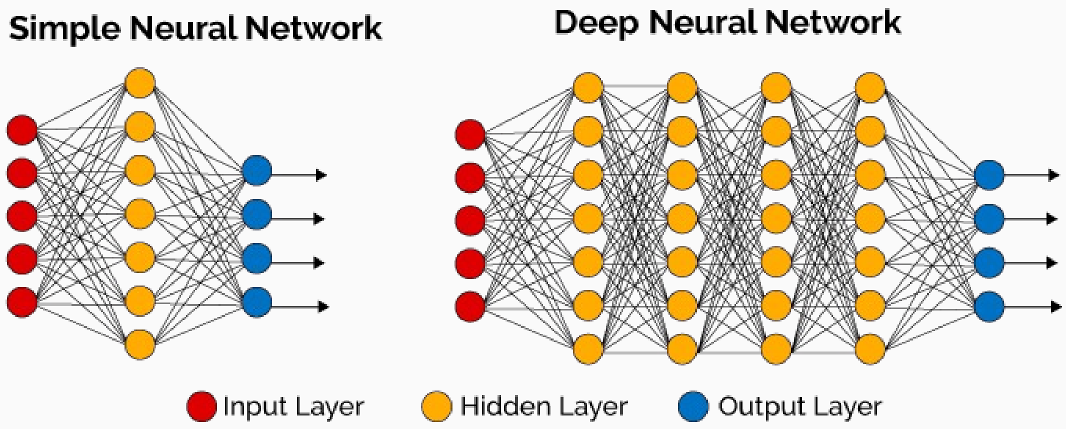
Introduction: *Deep Network*



Introduction: *Deep Learning*

- (+) Efficiently learning from high-dimensional data
- (+) State of the art in Computer Vision/Speech Recognition/NLP tasks
- (+) Representation learning

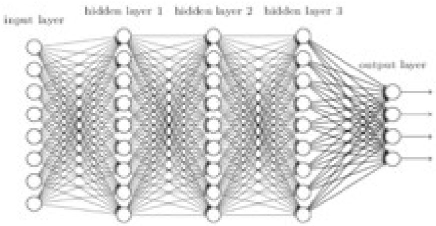
- (-) Data-greedy
- (-) Training Computationally intensive
- (-) Hyperparameter tuning



Introduction: *Main Architectures*

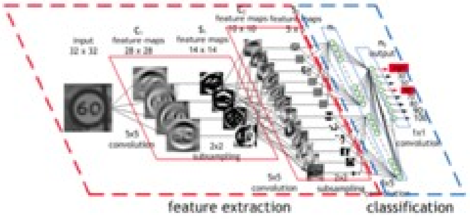
providing lift for classification and forecasting models

Deep Neural Networks



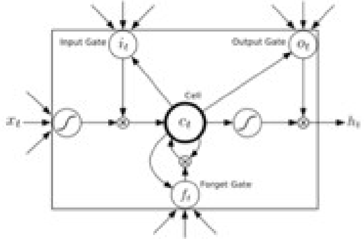
feature extraction and classification of images

Convolutional Neural Networks



for sequence of events, language models, time series, etc.

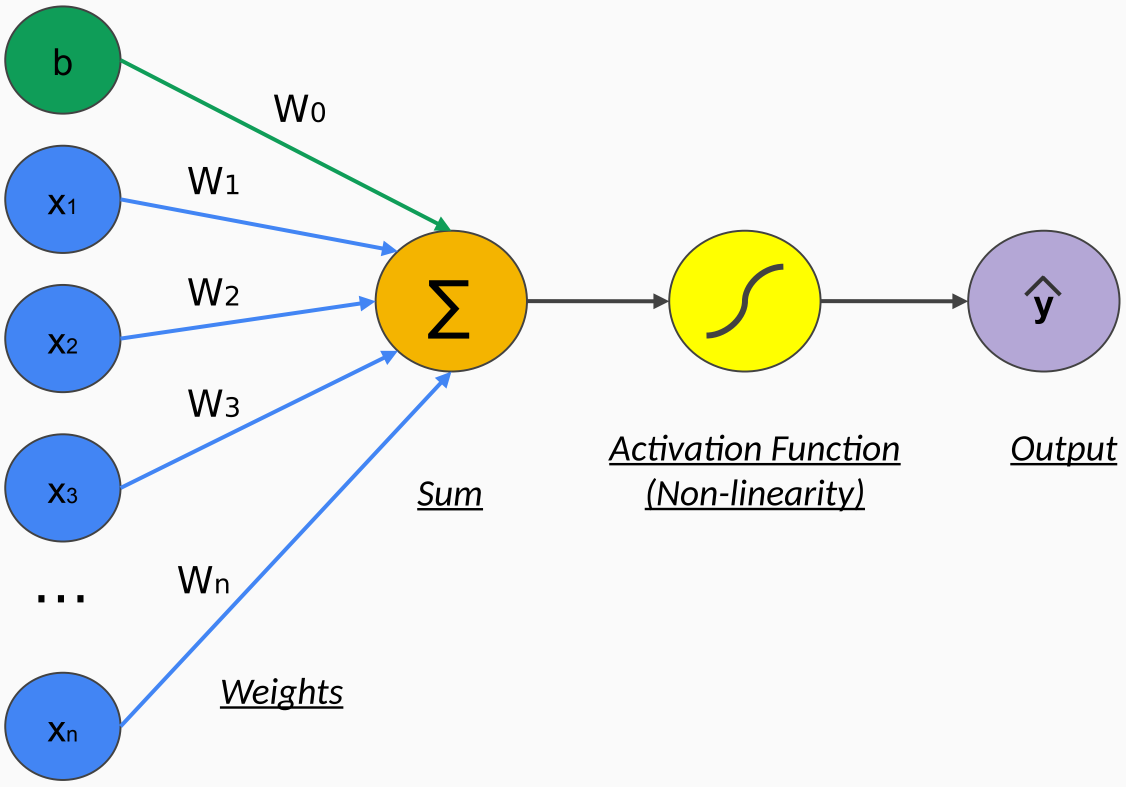
Recurrent Neural Networks



*https://dzone.com/articles/demystifying-ai-machine-learning-and-deep-learning

Deep Neural Network

The Perceptron - Forward Pass

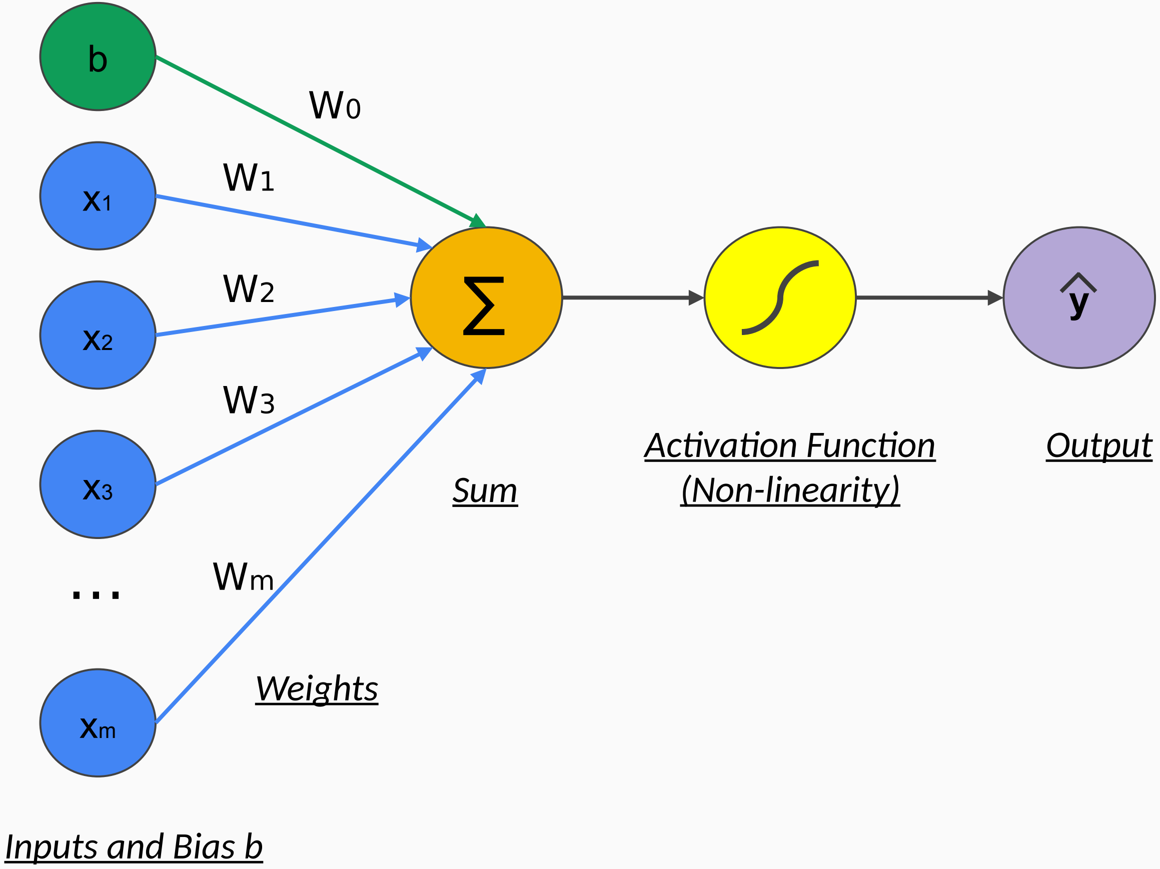


Inputs and Bias b

Activation Function

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

The Perceptron - Forward Pass

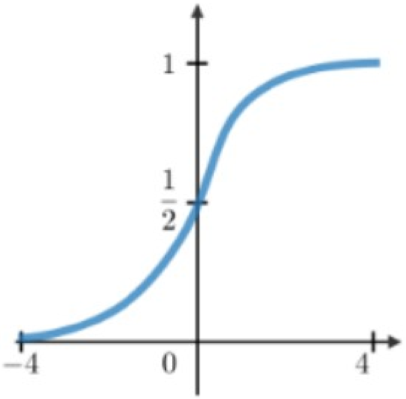
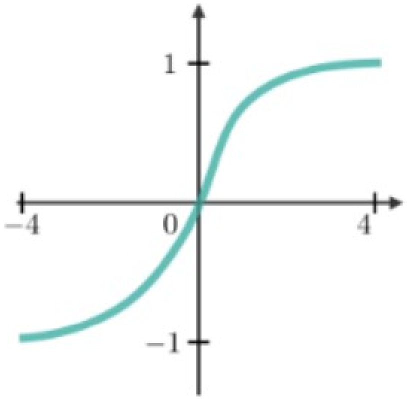
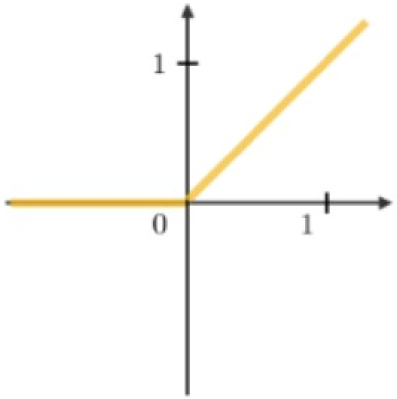


Activation Function

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

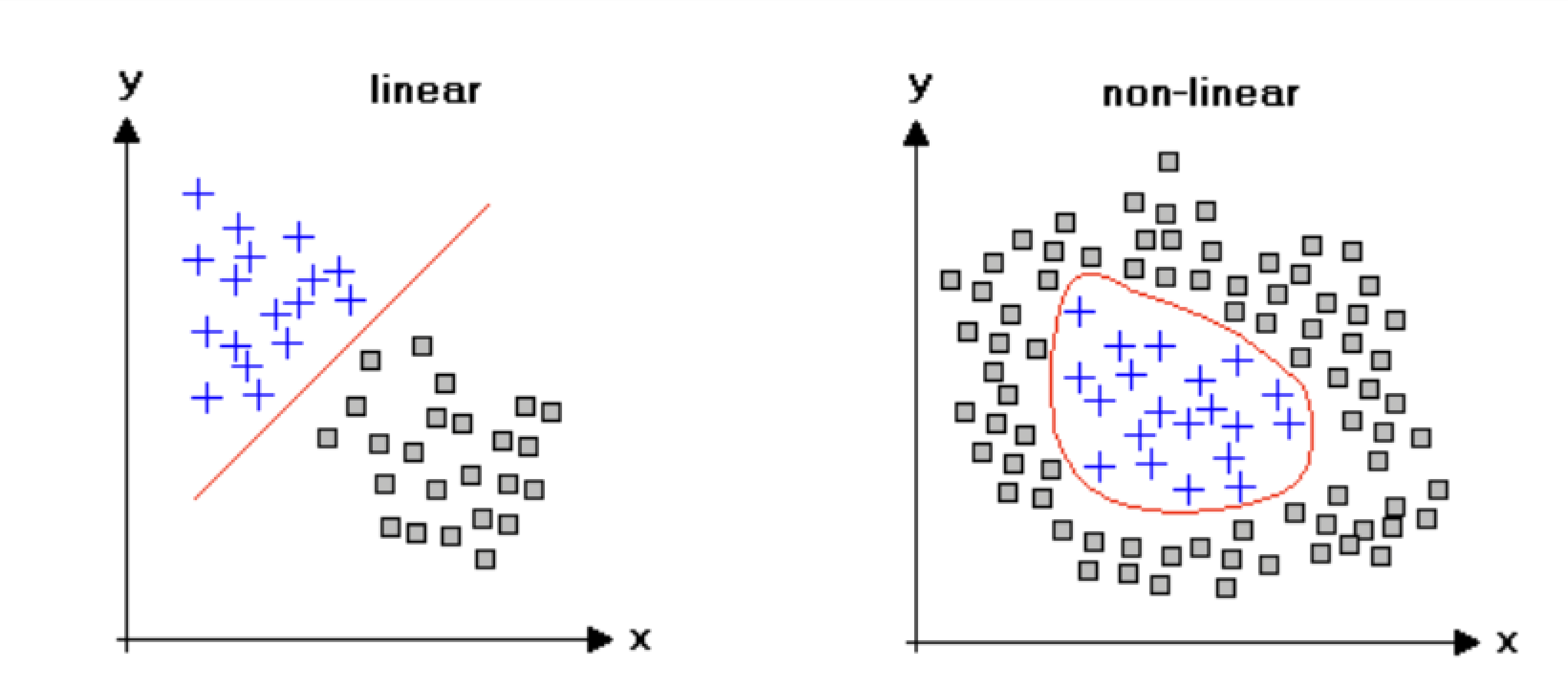
$$\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \text{ and } \boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

The Perceptron - Activation Functions

Sigmoid	Tanh	RELU
$g(z) = \dfrac{1}{1 + e^{-z}}$	$g(z) = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	$g(z) = \max(0, z)$
		

*https://studymachinelearning.com/activation-functions-in-neural-network/

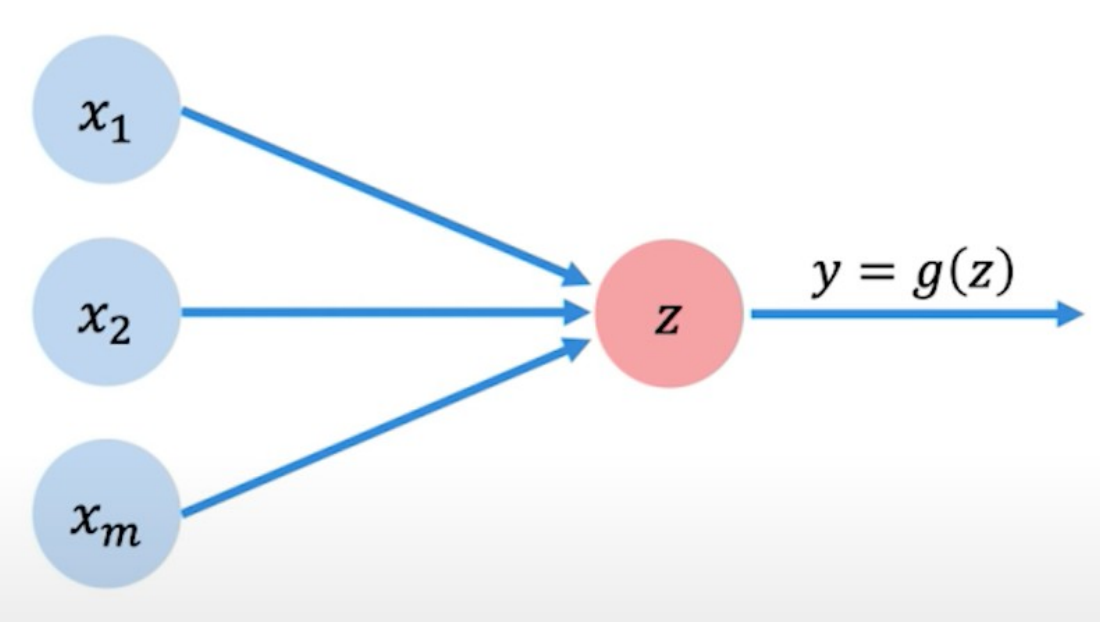
The Perceptron - Activation Functions and Non-Linearity



*https://studymachinelearning.com/activation-functions-in-neural-network/

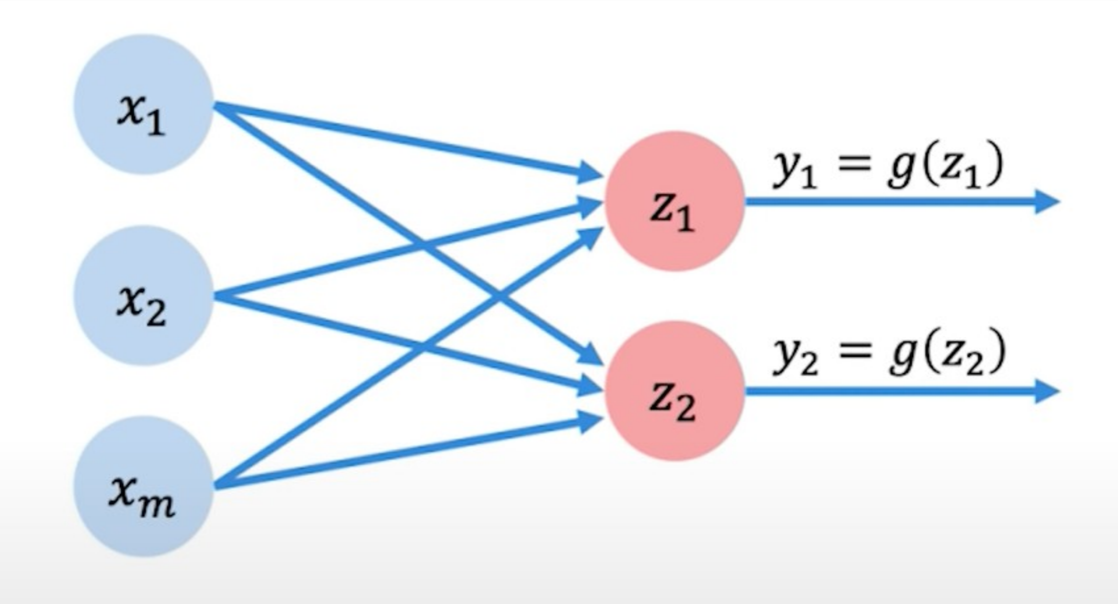
The Perceptron - Building a Deep Neural Network

Simplified Perceptron



$$z = w_0 + \sum_{j=1}^{m} x_j \, w_j$$

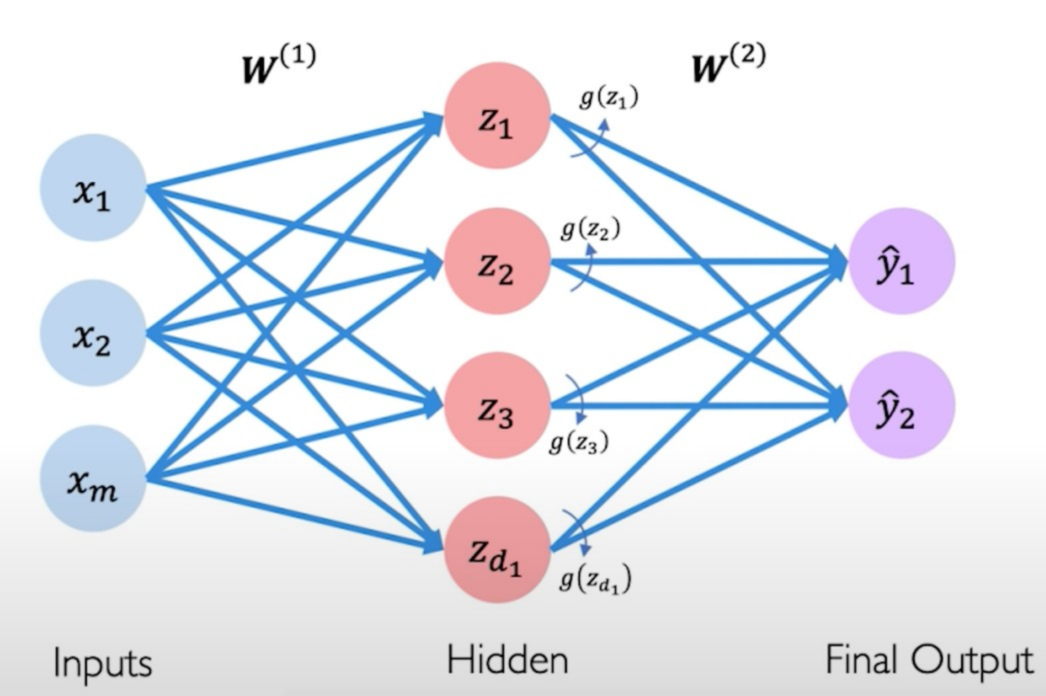
Multi-Output Perceptron (**Dense Layers**)



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

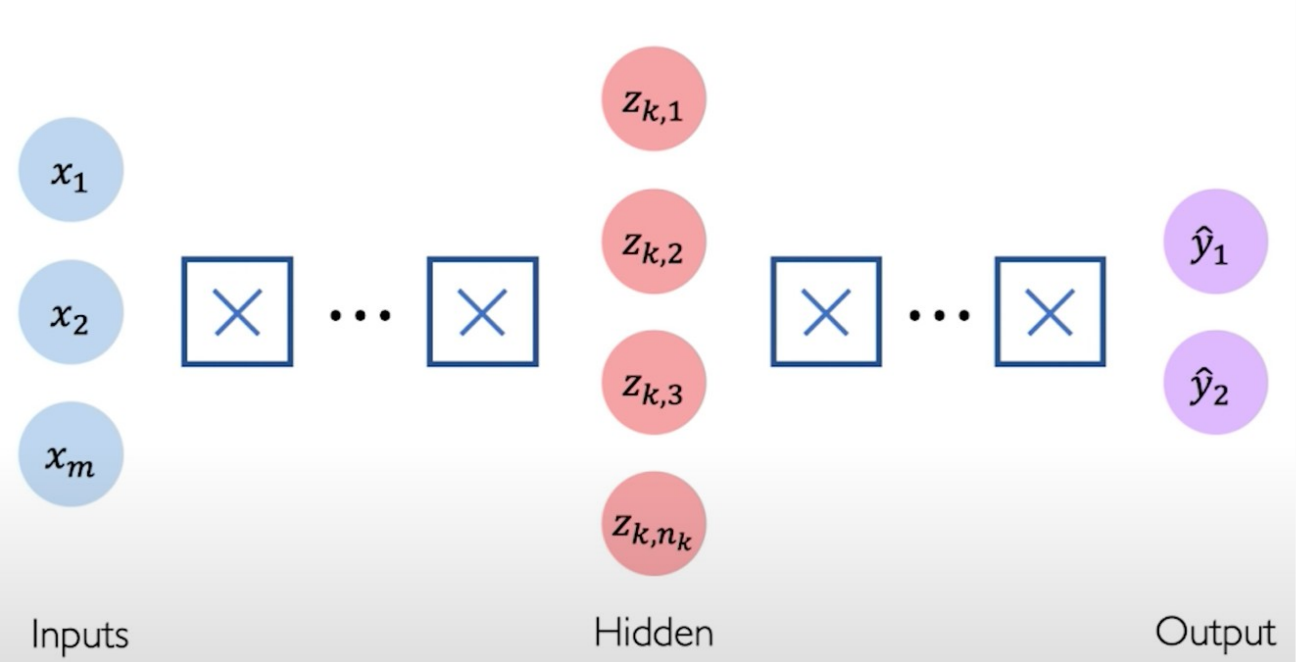
The Perceptron - Building a Deep Neural Network

Single Layer Neural Network



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) \, w_{j,i}^{(2)}\right)$$

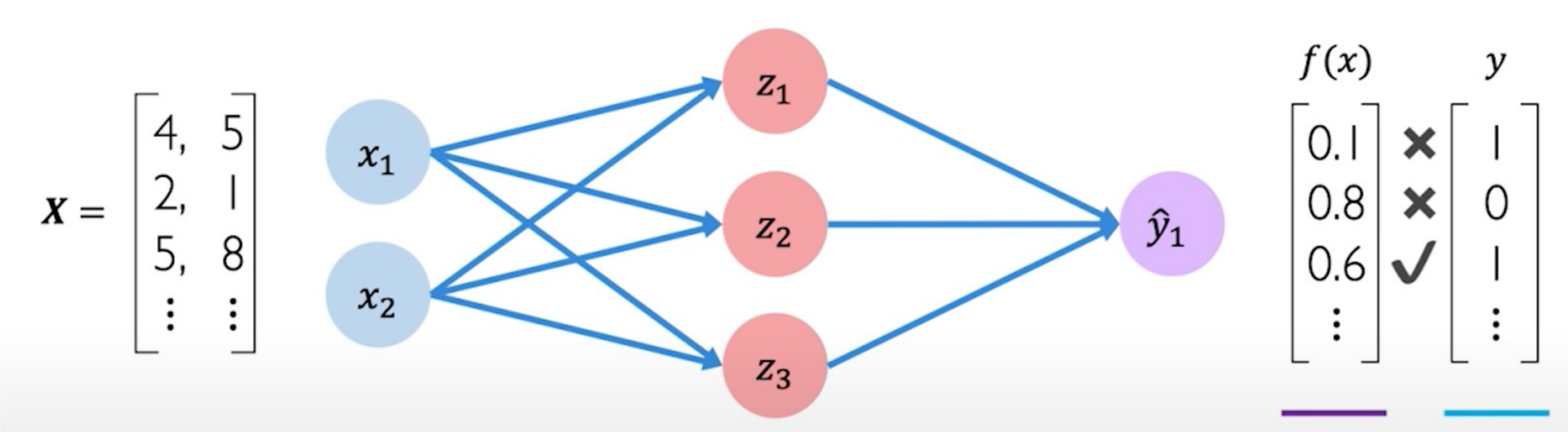
Deep Neural Networks



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Train a Neural Network - The Loss Function

Quantifying the **Loss** (over the entire training set)



Cost/Loss Function

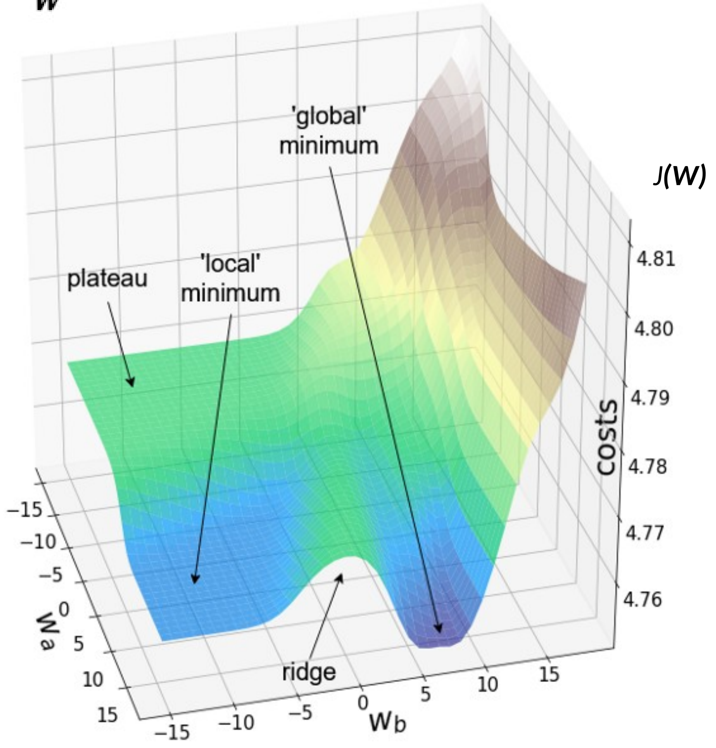
$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\Big(\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\Big)$$

Training a Neural Network - Lear

Gradient Descent

1. Initialize weights randomly $\sim\mathcal{N}(0,\sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$

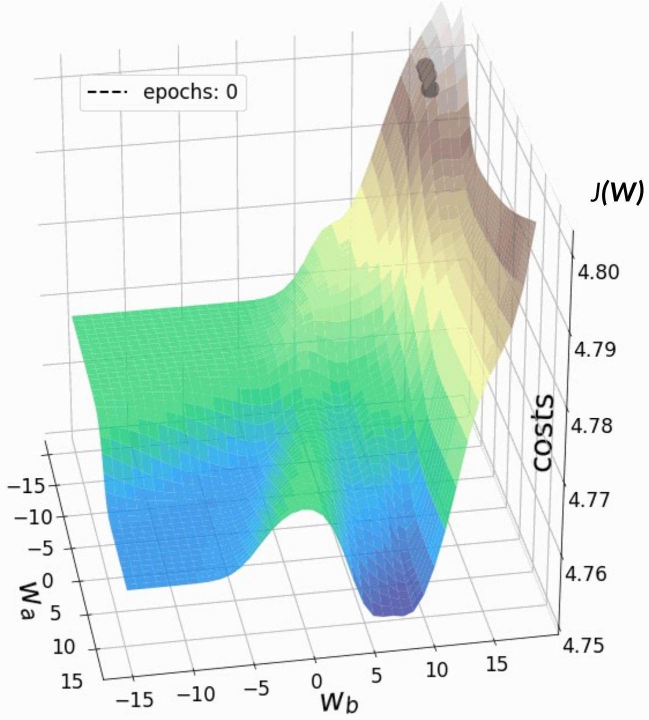


Training a Neural Network - Loss Optimization

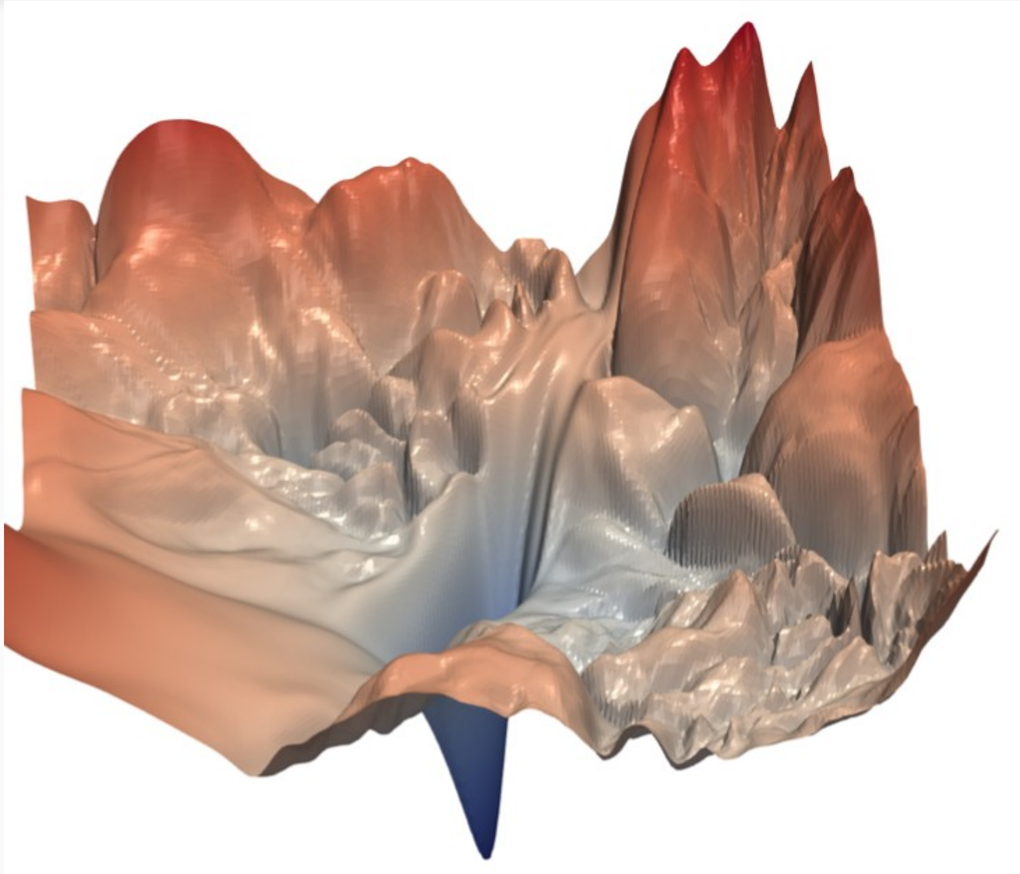
$$W^* = \operatorname{argmin} J(W)$$

Gradient Descent

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights



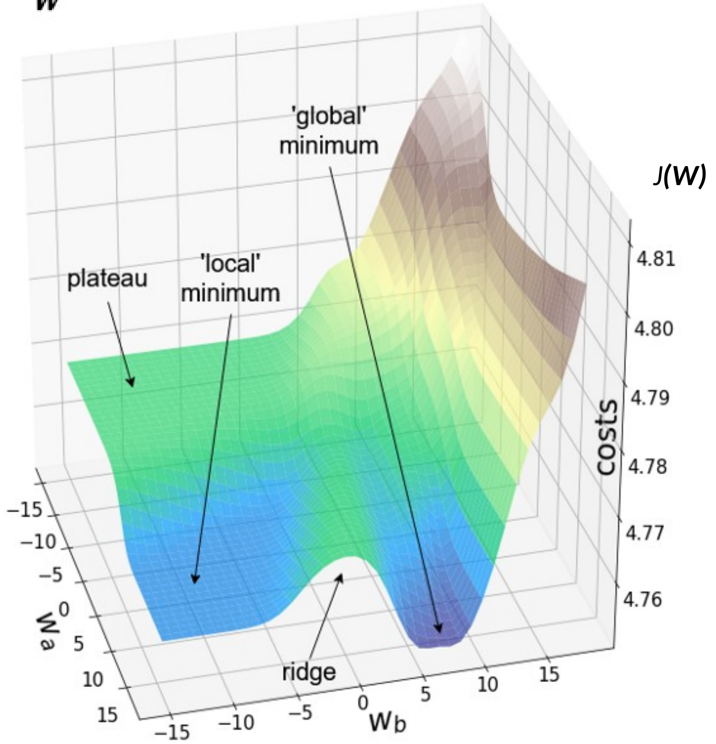
Training a Neural Network - The Loss Landscape



*[Visualizing the Loss Landscape of Neural Nets](), *Li et Al*(2018)

Training a Neural Network - The importance of Learning Rate

$$W^* = \underset{W}{\operatorname{argmin}}\, J(W)$$



Gradient Descent

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

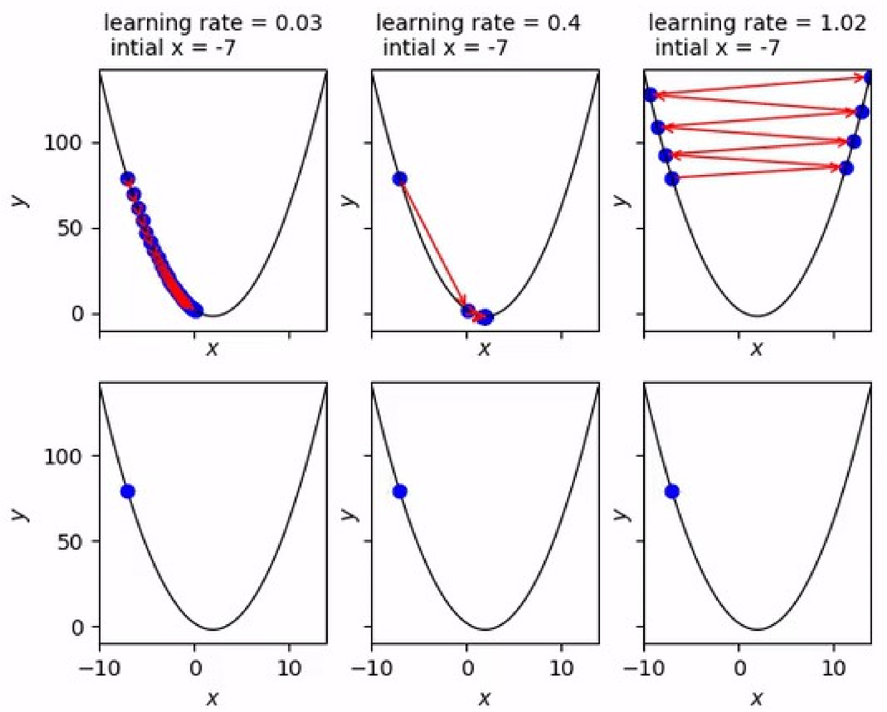
Learning Rate

Training a Neural Network - The Importance of Learning Rate

Gradient Descent

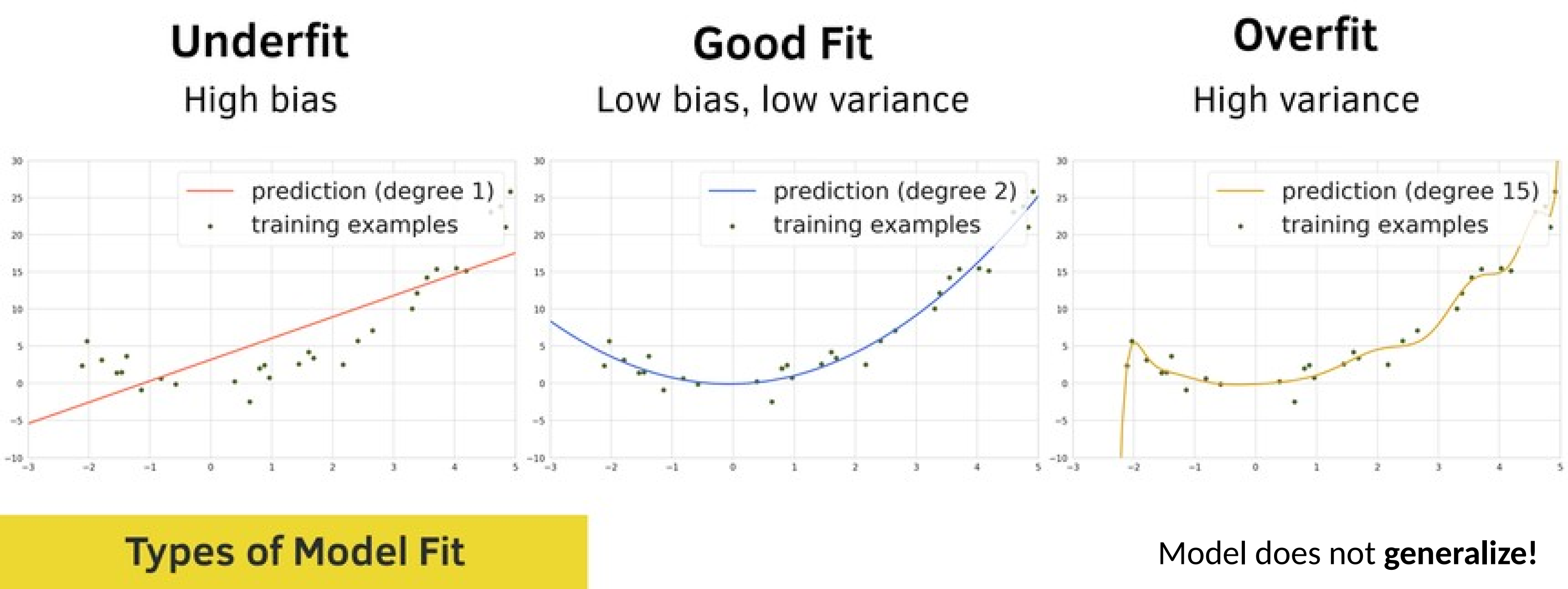
1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Learning Rate



*https://www.kaggle.com/code/ohseokkim/bird-species-standing-on-the-shoulders-of-giant

Training a Neural Network - Overfitting and Regularization



Model does not **generalize!**

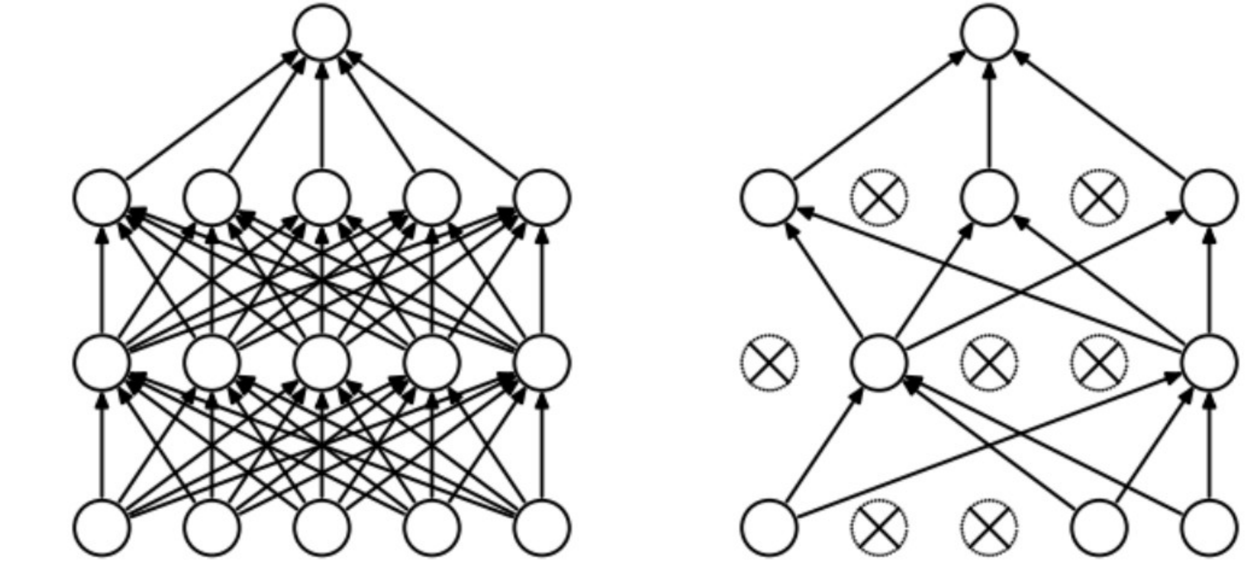
*https://curiousily.com/posts/hackers-guide-to-fixing-underfitting-and-overfitting-models/

Training a Neural Network - Overfitting and Regularization

Deal with **Overfitting** through **Regularization**

(to improve generalization of the model on unseen data)

DROPOUT



(a) Standard Neural Net   (b) After applying dropout.

Quick Summary

- **Perceptron** as building block of Deep Neural Network
- **Optimization** through backpropagation
- **Learning Rate**
- **Regularization**

Convolutional Neural Network

Introduction

Class of Deep **Feed-Forward** ANN Specialized
for processing data with a **grid-like** topology
(i.e.: time series data, image data, language)

Convolution Operation

- **1-D Convolution**

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

Convolution Operation

- **1-D Convolution**

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

Feature Map

Input

Kernel

Convolution Operation

- **1-D Convolution**

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

- Example

x

1	2	4	1

w

1 1	3	

Convolution Operation

- **1-D Convolution**

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

- Example

x

1	2	4	1

w

1 1	3	

x

1	2	4	1

1 1	3	

s

Convolution Operation

- **1-D Convolution**

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

- Example

x	1	2	4	1

w	1 1	3	

x	1	2	4	1
	1 1	3		
s	5			

s(0) = ~~x[-1]*w[0]~~ + x[0]*w[1] + x[1]*w[2] = ~~0*1~~ + 1*3 + 2*1 = 3 + 2 = **5**

Convolution Operation

- **1-D Convolution**

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

- Example

x	1	2	4	1

w	1 1	3	

x	1	2	4	1
	1 1	3		
s	5	11		

s(1) = x[0]*w[0] + x[1]*w[1] + x[2]*w[2] = 1*1 + 2*3 + 4*1 = 1 + 6 + 4 = **11**

Convolution Operation

- **1-D Convolution**

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

- Example

x

1	2	4	1

w

1 1	3	

x

1	2	4	1

	1 1	3	

s

5	11	15	

s(2) = x[1]*w[0] + x[2]*w[1] + x[3]*w[2] = 2*1 + 4*3 + 1*1 = 2 + 12 + 1 = **15**

Convolution Operation

- **1-D Convolution**

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

- Example

x

1	2	4	1

w

1 1	3	

x

1	2	4	1
		1 1	3

s

5	11	15	7

s(3) = x[2]*w[0] + x[3]*w[1] + ~~x[4]*w[2]~~ = 4*1 + 1*3 + ~~0*1~~ = 4 + 3 = **7**

Convolution Operation

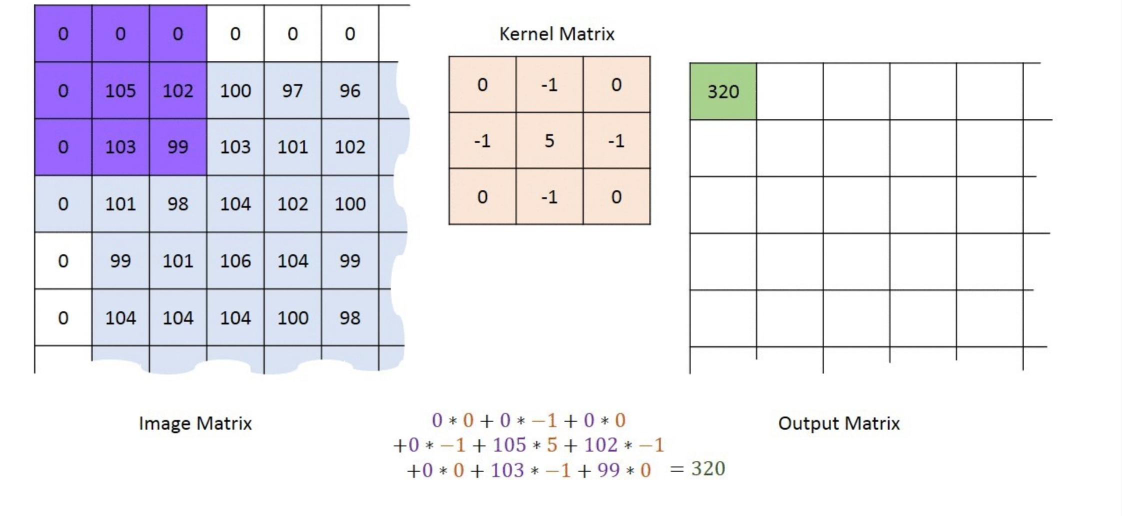
- **2-D Convolution**

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n)$$

Convolution Operation

- **2-D Convolution**

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$



Convolution with horizontal and vertical strides = 1

Credit: http://machinelearninguru.com/computer_vision/basics/convolution/convolution_layer.html

Convolution Operation

- **2-D Convolution**

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

- Different Kernels/Filters are used in Image processing for revealing characteristics of the input

Convolution Operation

- **2-D Convolution**

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

- Different Kernels/Filters are used in Image processing for revealing characteristics of the input

*

-1	-1	-1
-1	8	-1
-1	-1	-1

→

Edge Detection

Convolution Operation

- **2-D Convolution**

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

- Different Kernels/Filters are used in Image processing for revealing characteristics of the input

$$* \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} \longrightarrow$$

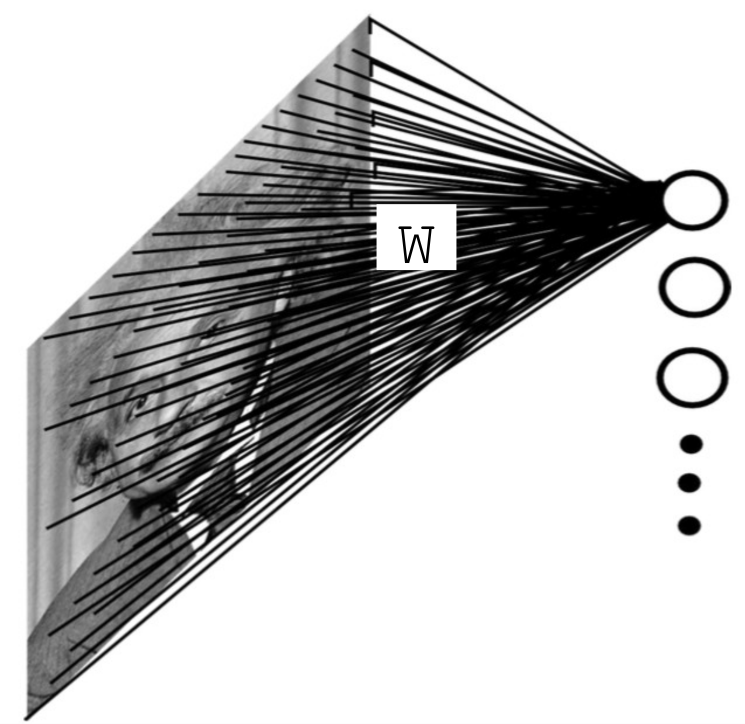
Blurring

Convolution - Properties

- Sparse Interactions
- Parameter sharing
- Equivariant Representation

Sparse Interactions - *Full Connectivity VS Sparse Connectivity*

- **Full Connectivity**: Fully connected NW on the whole image

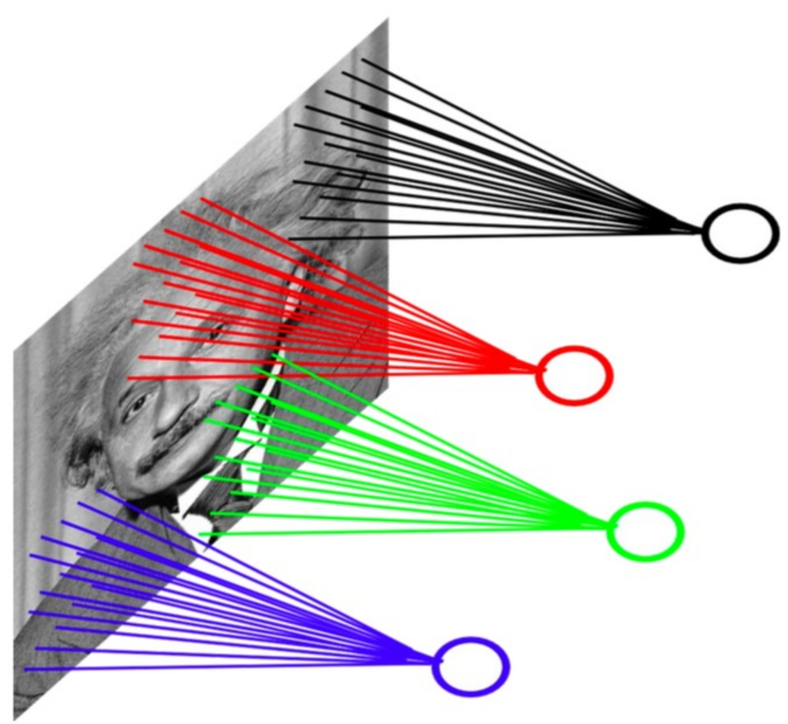


200x200 Image → **40.000** Pixels in *input*

400.000 *hidden* units → **40Kx400K** = **16B** *parameters*

Sparse Interactions - *Fully Connectivity VS Sparse Connectivity*

- **Sparse Connectivity**: locally connected NW on the whole image



200x200 Image → **40.000** Pixels in *input*

400.000 *hidden* units → **40Kx400K** = **16B** *parameters*

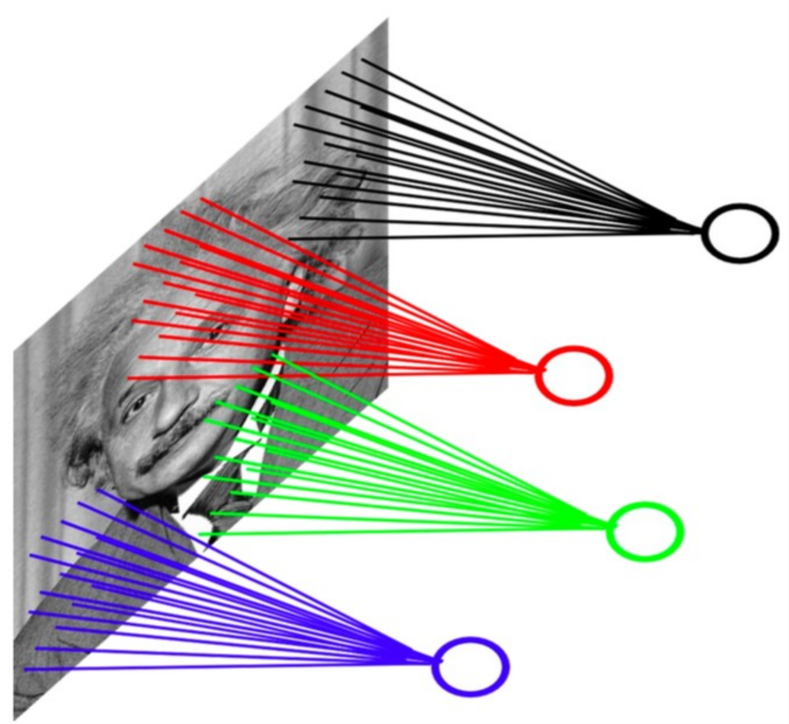
200x200 Image → **40.000** Pixels in *input*

10x10 *fields*

400.000 *hidden* units → **40Kx100** = **40M** *parameters*

Sparse Interactions - *Fully Connectivity VS Sparse Connectivity*

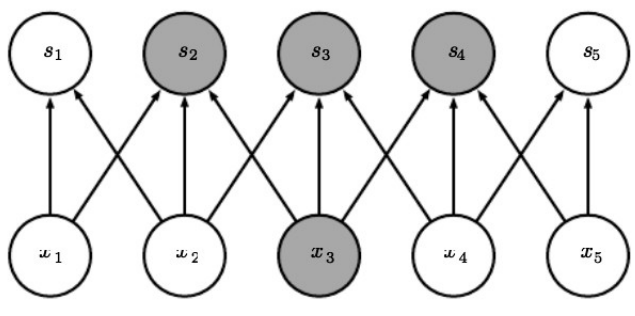
- **Sparse Connectivity**: locally connected NW on the whole image



Sparse Connectivity - Pros

- **Fewer parameters** stored for the **model**
- Reducing **memory** requirements
- Improving **statistical** efficiency
- **Less** operations for **computing** the **output**
- *O(mxn)* → *O(kxn)*

Sparse Interactions - *Fully Connectivity VS Sparse Connectivity*

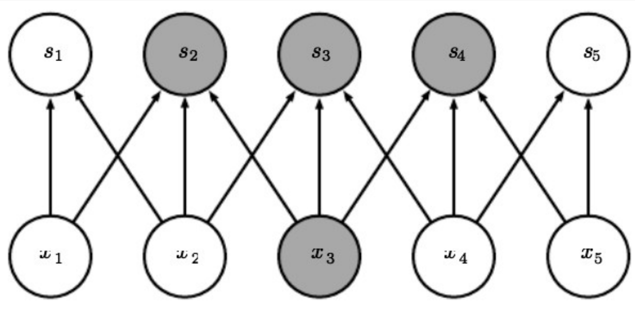


Sparse Connectivity

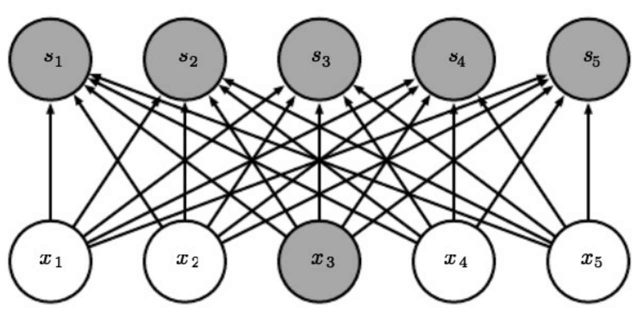
Sparse Connectivity - Pros

- **Fewer parameters** stored for the **model**
- Reducing **memory** requirements
- Improving **statistical** efficiency
- **Less** operations for **computing** the **output**
- *O(mxn) → O(kxn)*

Sparse Interactions - *Fully Connectivity VS Sparse Connectivity*



Sparse Connectivity

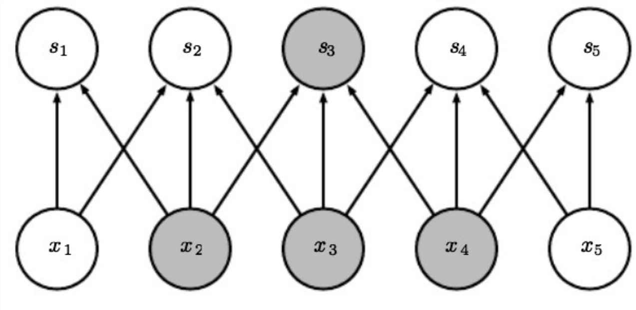


Full Connectivity

Sparse Connectivity - Pros

- **Fewer parameters** stored for the **model**
- Reducing **memory** requirements
- Improving **statistical** efficiency
- **Less** operations for **computing** the **output**
- *O(mxn) → O(kxn)*

Sparse Interactions - *Fully Connectivity VS Sparse Connectivity*

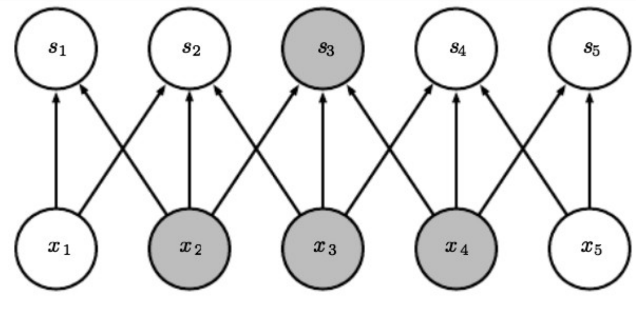


Sparse Connectivity

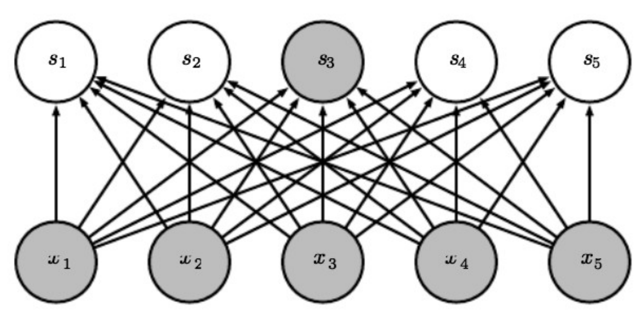
Sparse Connectivity - Pros

- **Fewer parameters** stored for the **model**
- Reducing **memory** requirements
- Improving **statistical** efficiency
- **Less** operations for **computing** the **output**
- *O(mxn) → O(kxn)*

Sparse Interactions - *Fully Connectivity VS Sparse Connectivity*



Sparse Connectivity

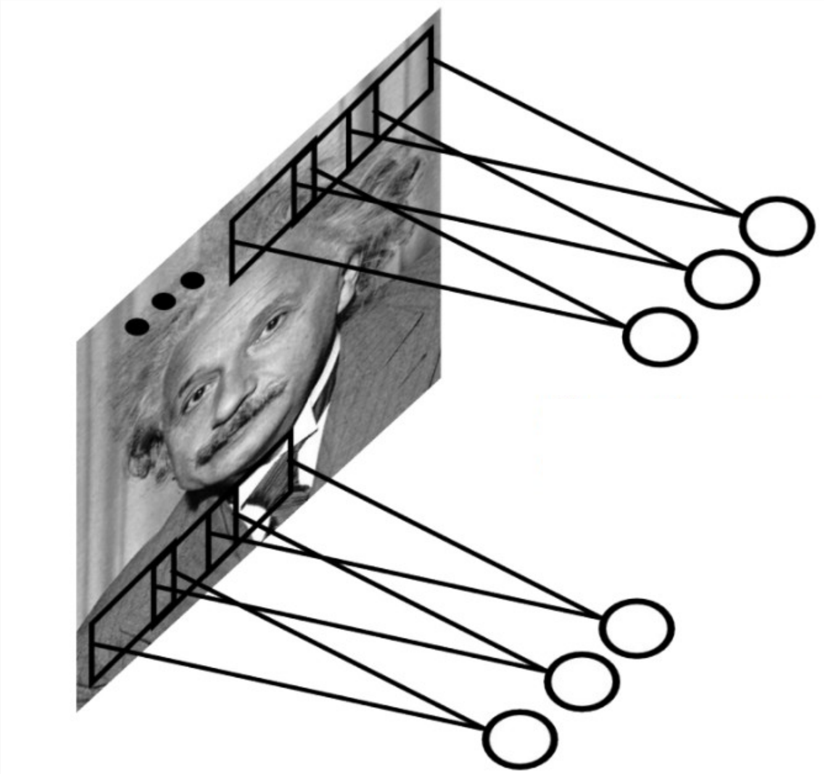


Full Connectivity

Sparse Connectivity - Pros

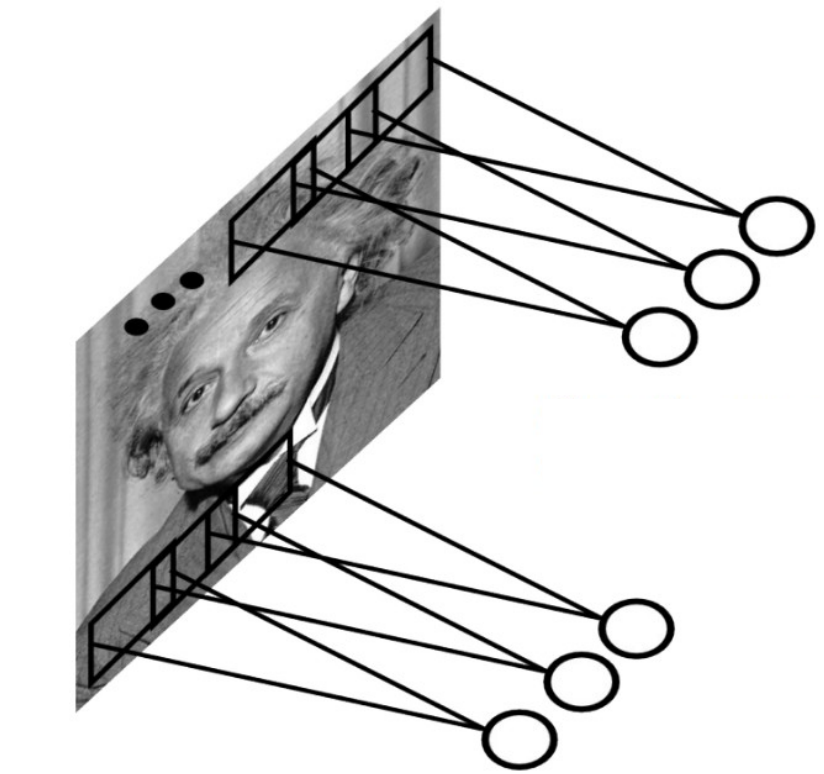
- **Fewer parameters** stored for the **model**
- Reducing **memory** requirements
- Improving **statistical** efficiency
- **Less** operations for **computing** the **output**
- *O(mxn) → O(kxn)*

Parameter Sharing



- **Re-used kernel**
- Each member of the **kernel** used at **every position** of the **input**
- Only one set of parameters are learned for all the locations
- **Less** operations for **computing** the **output**
- *O(mxn) → O(kxn) for the forward propagation*

Parameter Sharing causes **Equivariance** (to Translation)

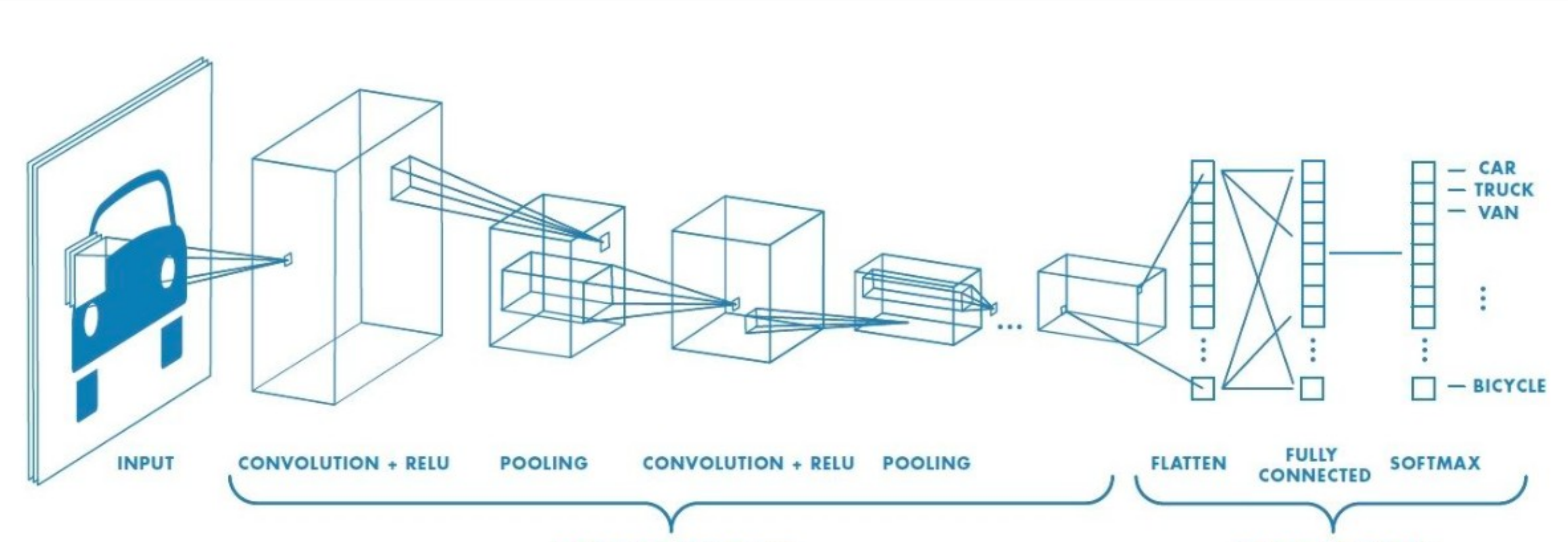


- **Re-used kernel**
- Each member of the **kernel** used at **every position** of the **input**
- Only one set of parameters are learned for all the locations
- **Less** operations for **computing** the **output**
- *O(mxn) → O(kxn) for the forward propagation*
- **Translation-Invariant**

Convnets - Layers

- Input
- Convolutional Stage
- Nonlinearity (e.g. *ReLU*)
- Pooling
- ...
- Fully Connected Layer

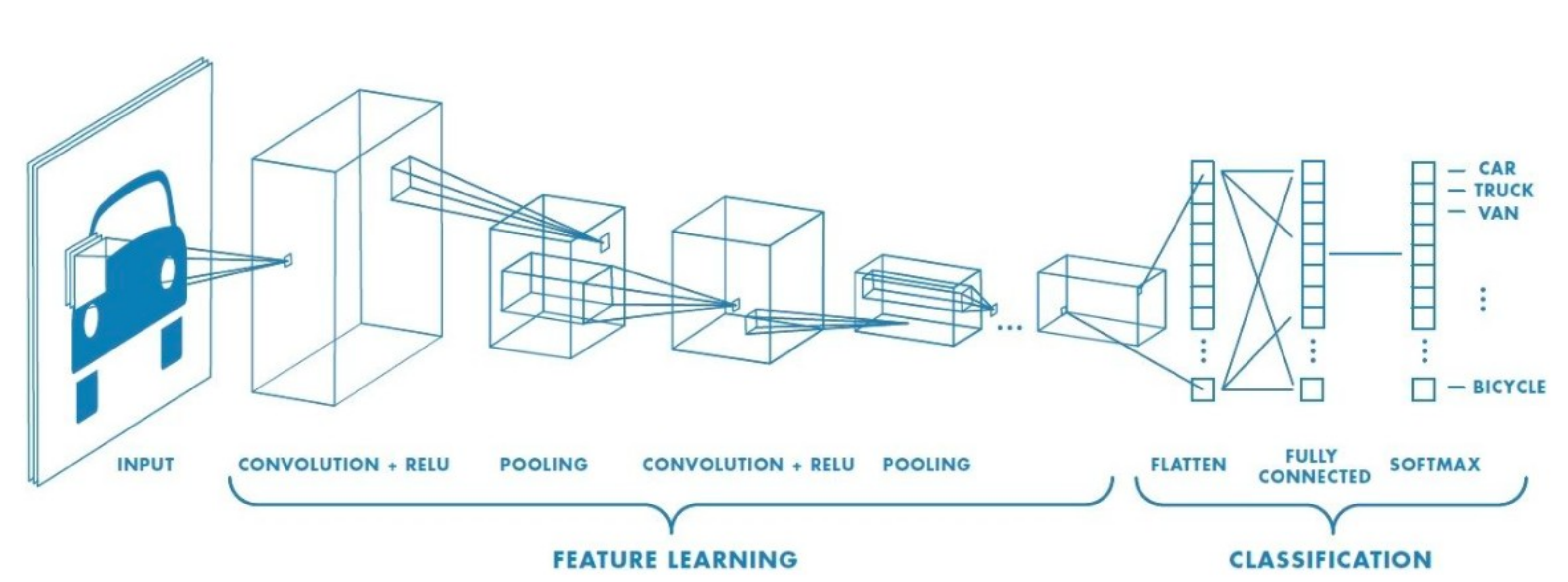
Convolutional Layer



Convnets - Layers

- Input
- Convolutional Stage
- Nonlinearity (e.g. *ReLU*)
- Pooling
- …
- Fully Connected Layer

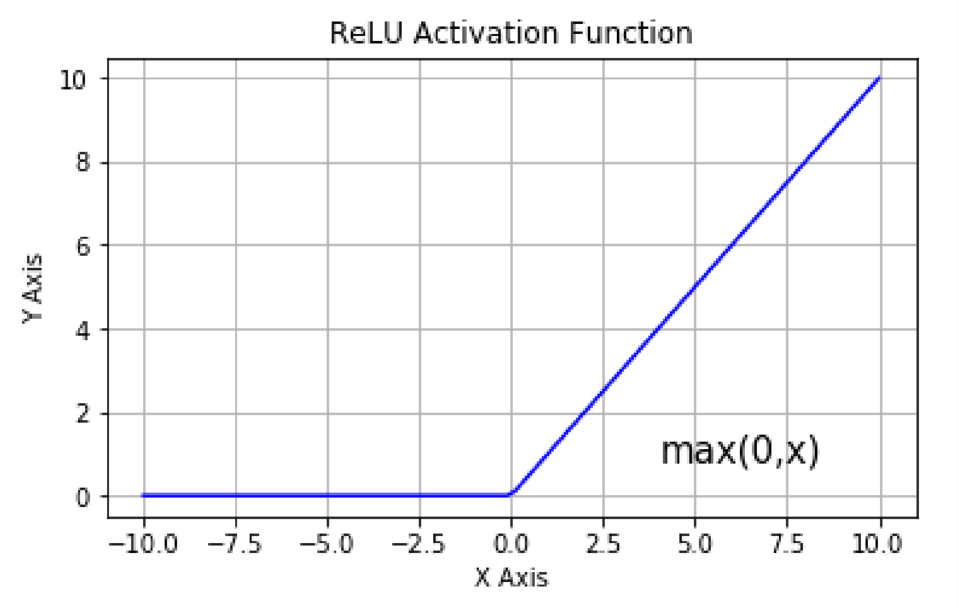
Convolutional Layer



Convolutional Layer

*https://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/

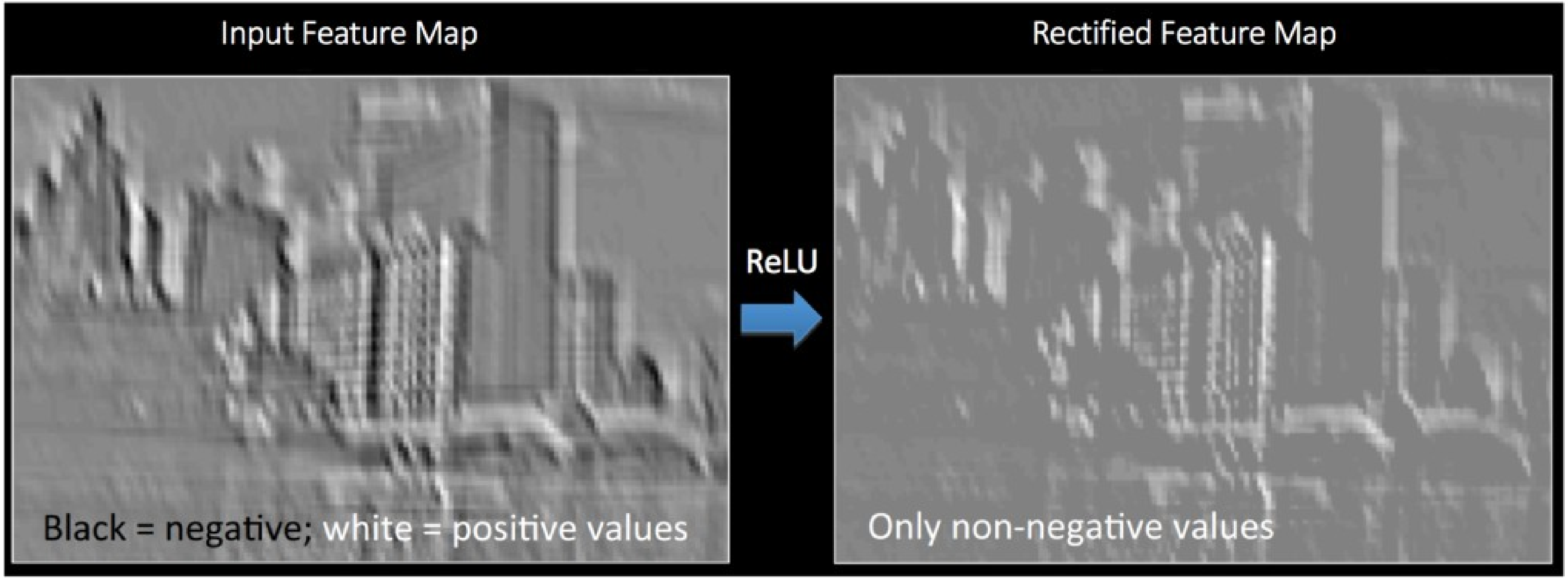
Nonlinearity - ReLU



- *Rectified Linear Unit*
- Introducing **Non-Linearity** to the ConvNet
- **Replace** all negative input values to **zero**

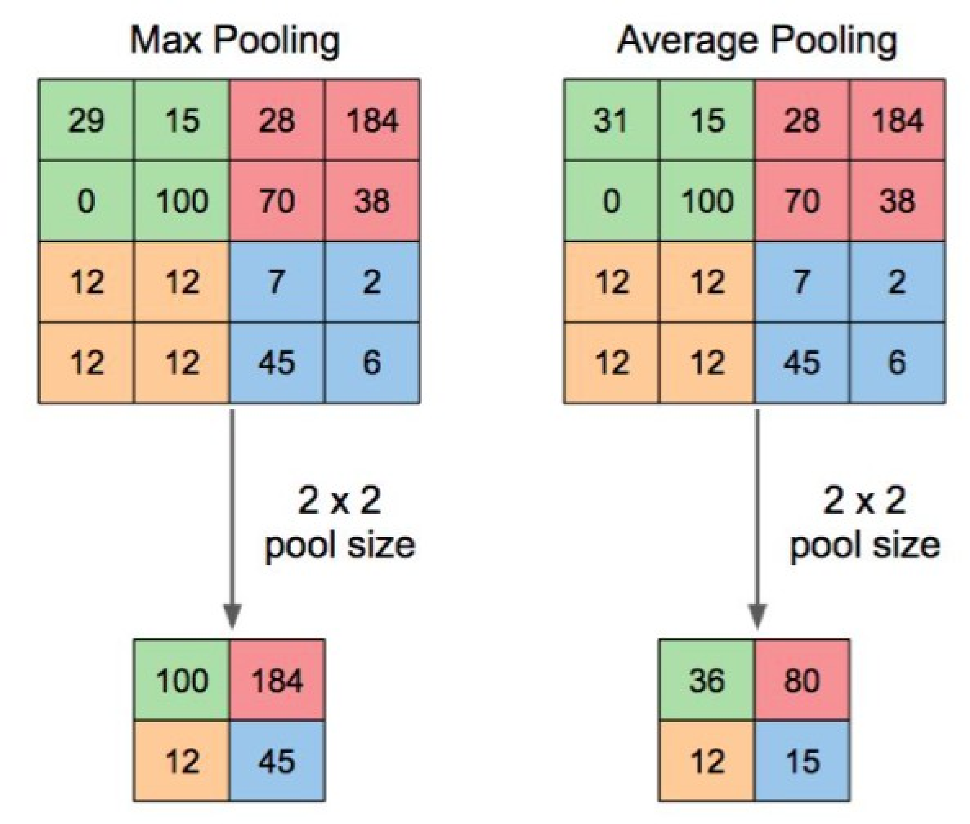
*https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/

Nonlinearity - ReLU



**Visualizing and Understanding Convolutional Neural Networks Matthew (2013)* - D. Zeiler, Rob Fergus

Pooling Layer



- Applied to a **feature map** after the non-linearity (ReLU)
- Providing **summary statistics** of the **nearby output**
- **maximum/average** output within a rectangular **neighborhood**
- Subsampling effect

Fully Connected Layer + Softmax

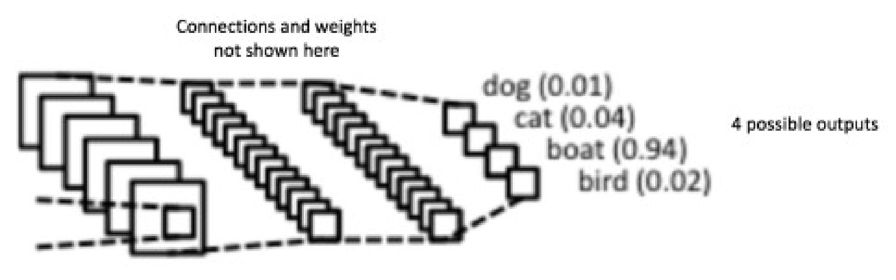
Fully Connected Layer

- Generally a **MLP** and **every neuron** in the previous **layer** is **connected** to **every neuron** in the **next**
- Use features extracted from the convolutional layers for performing **classification**

Softmax

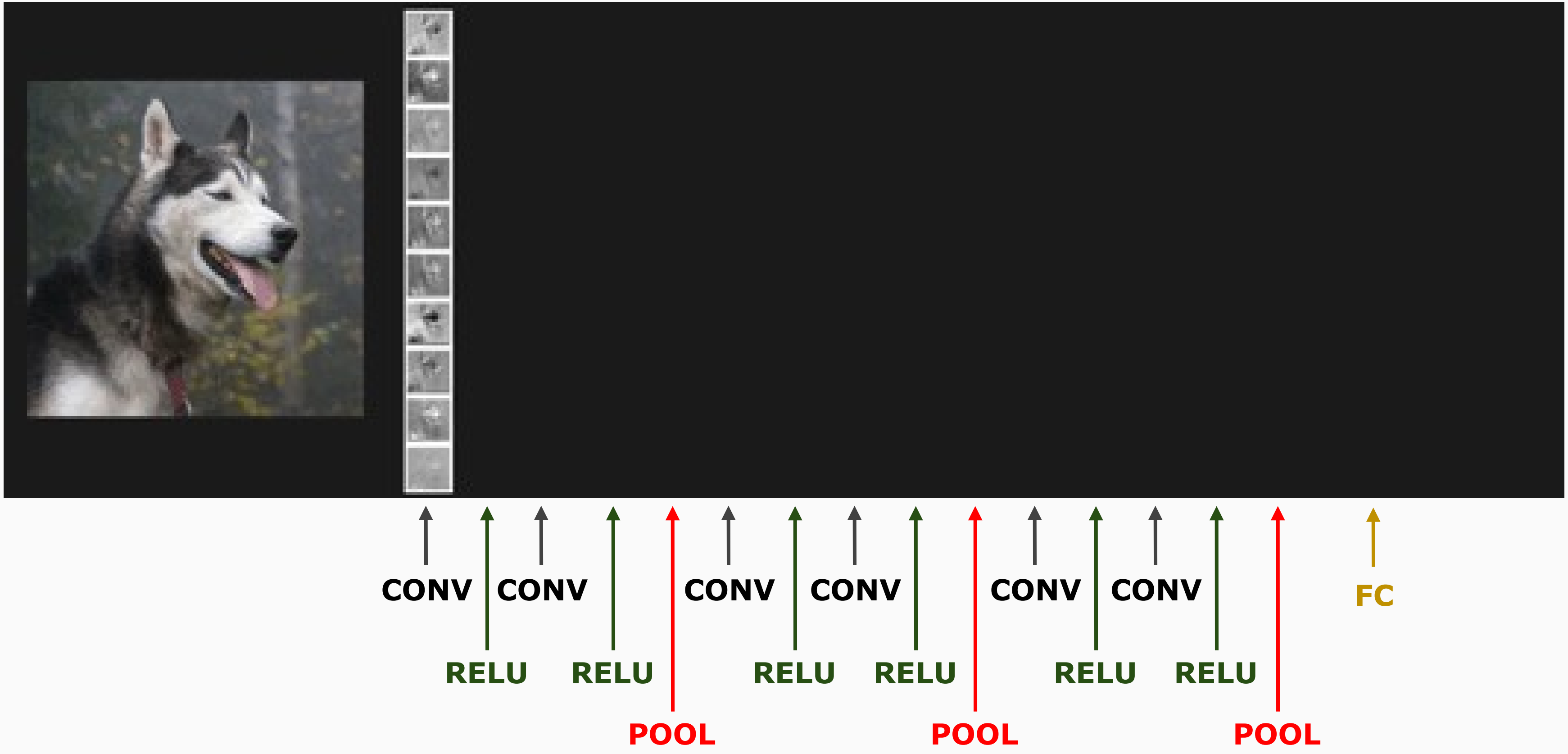
$$P(y=j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

- Applied to the **output** of the **FC** layer
- Producing a **discrete probability distribution** layer



Fully Connected layers

Convnet in action



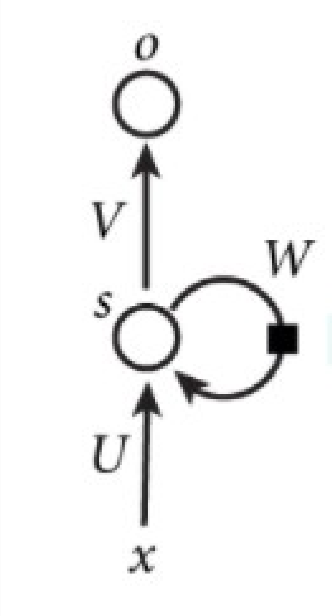
Summary

- Computationally more efficient
- Translation invariance
- Improving memory requirements

Recurrent Neural Network

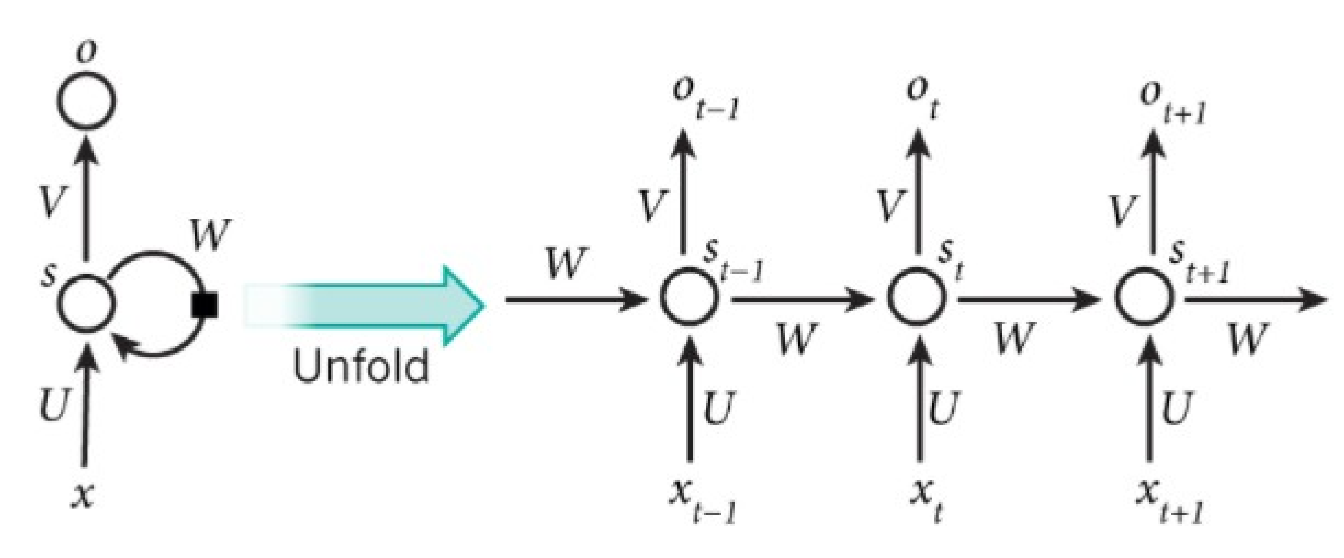
Recurrent Neural Network: Intro

- RNN: family of NN for processing **sequential data**
- **Example**: *predicting the next word of a sentence*
- **Recurrent**: performing the **same task** for every element of the sequence
- **Output**: dependent on previous computation
- RNN have **memory**



Recurrent Neural Network: Intro

- Unfolding RNN



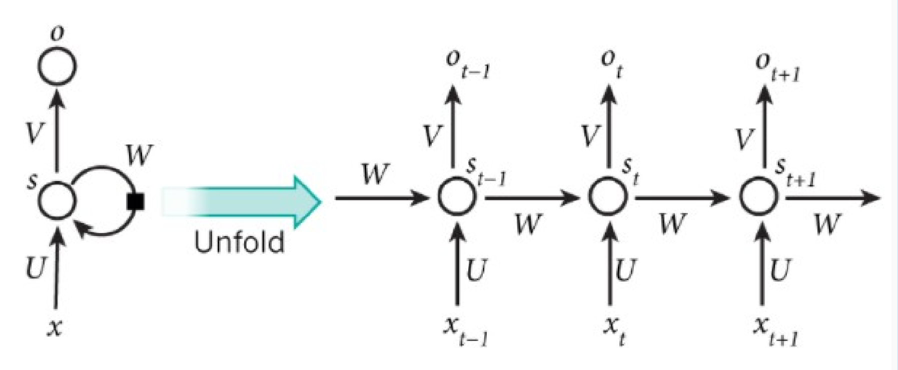
Recurrent Neural Network: Intro

- x_t : **input** at timestamp t
- s_t : **hidden** state

$$s_t = f(Ux_t + Ws_{t-1})$$

- o_t : **output** at timestamp t

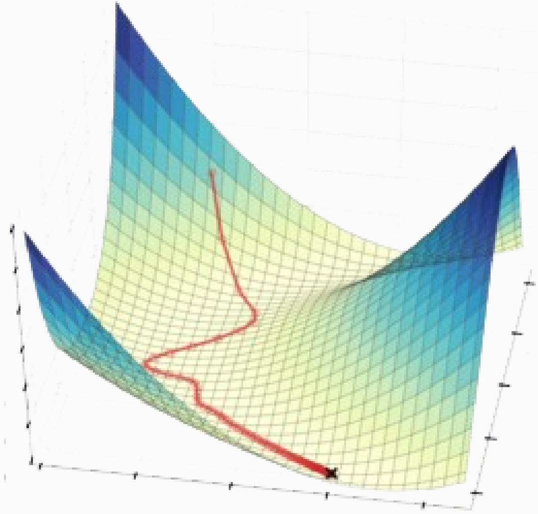
$$o_t = \mathrm{softmax}(Vs_t)$$



Recurrent Neural Network: Training

- **Learning** the **parameters**: *U,V,W*
- **SGD**: Stochastic Gradient Descent
  - Minimizing the **total loss** of the training data
  - **Iterative** process
  - Nudge the parameters in the **directions** of the **gradients**

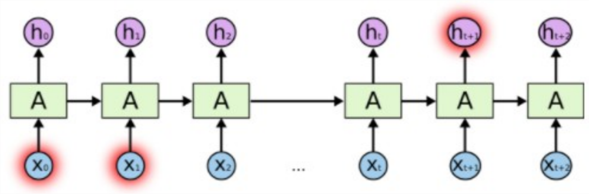
$$\frac{\partial L}{\partial U}, \frac{\partial L}{\partial V}, \frac{\partial L}{\partial W},$$



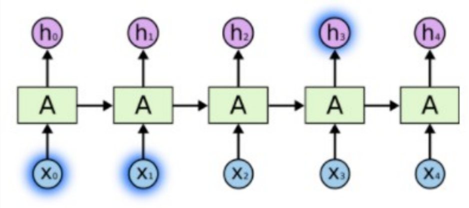
- **BPTT:** Backpropagation Through Time
  - **Modified** version of **backpropagation algorithm** for **computing** the **gradients**

Long-Term Dependency Problem

- **Example**: *Prediction of next word*
  - *"The clouds are in the" → ? ["Sky"]*



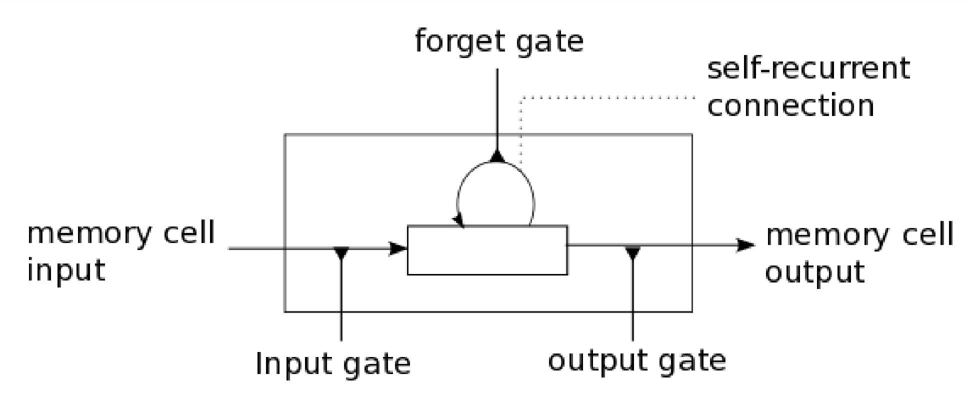
  - *"I grew up in Italy (...) I speak fluent" → ? ["Italian"]*



- **Vanishing Gradient Problem**
  - Gradients become too large or too small during the iterative process of parameter learning

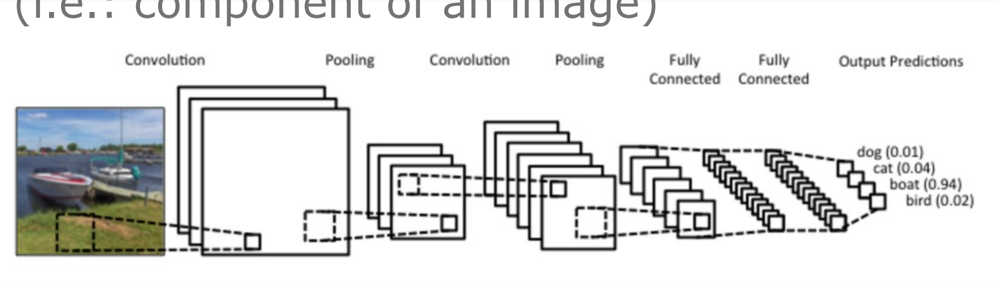
LSTM - Long Short-Term Memory

- Designed to **handle long-term** dependency
- **Memory cell** unit
  - *Forget Gate*: information to throw away (in the cell state)
  - *Input Gate*: information to store (in the cell state)
  - *Output Gate:* what to output

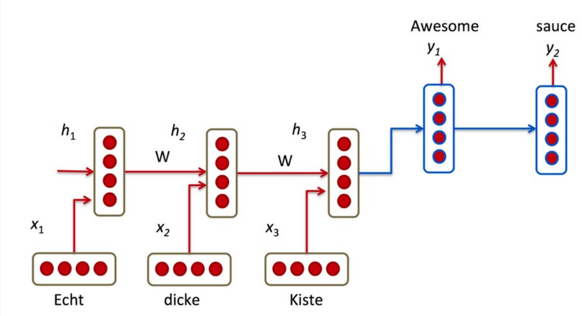


RNN vs CNN

- **CNN**: Neural network able to recognize patterns across the space (i.e.: component of an image)



- **RNN**: NN able to capture pattern from sequential data



- CNN + RNN in **joint** architectures!

Recurrent Neural Network: Applications

- Sentence Modelling
- Click Prediction
- Location Prediction
- Language Translation
- Sentiment Analysis
- Image Captioning and Description
- Speech Recognition
- Question/Answering Systems
- Text Generation

EXTRA: Transformer architecture

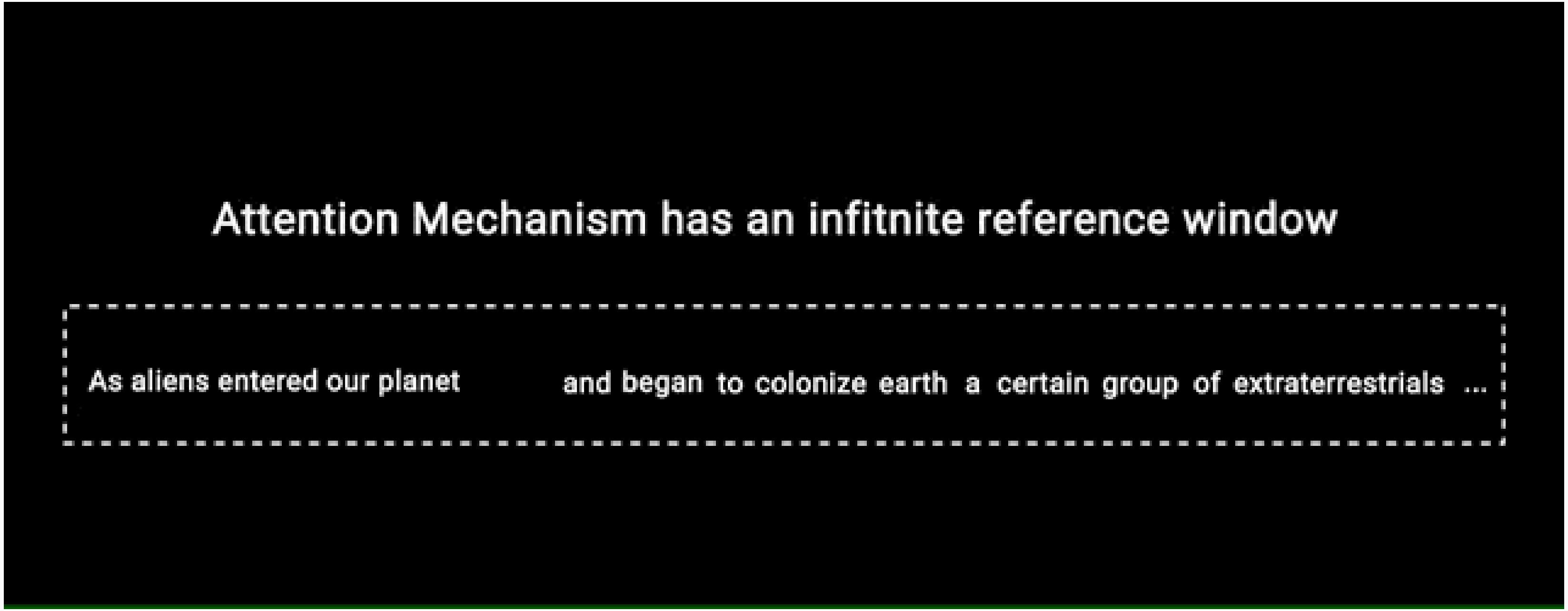
- Vanishing and exploding gradient problem
- LSTM struggle in capturing long-term dependencies
- RNN-based architectures prevents efficient parallelization when encoding

Self Attention +
Positional Encoding

Transformer

EXTRA: Transformer architecture

- Attention mechanism enable Transformer to have **very long term memory**



EXTRA: Transformer architecture

- The Vanilla Transformer

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

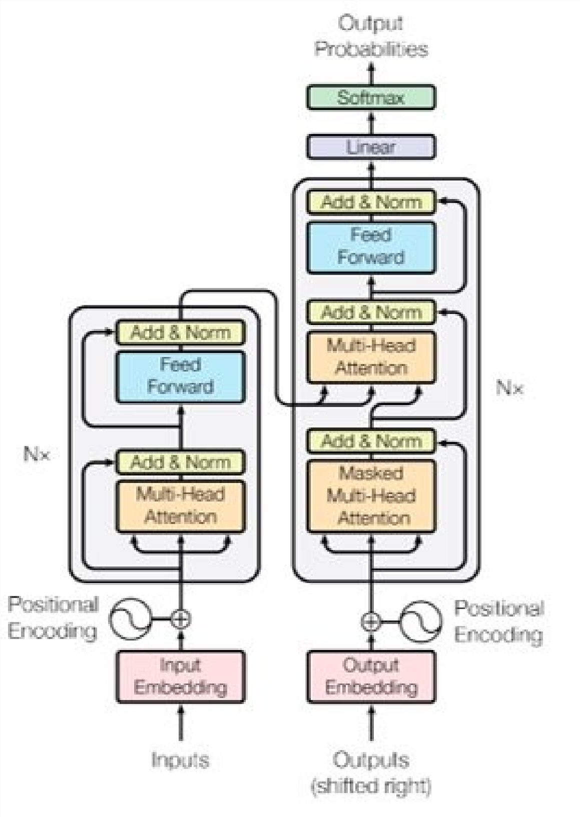
Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com



Conclusion

- Recurrent Neural are able to model **sequences**
- **Training** RNN is hard because the **vanishing problem**
- **LSTM** tackle the **Long-Term Dependency** Problem
- Mostly useful in **NLP** related **problems**

Summary

- Deep Neural Network can learn **extremely complex patterns**
- **CNN** suitable for learning **pattern across space** (i.e., images)
- **RNN** suitable for **sequence-wise** kind of data

References

- **Deep Learning Book** [CHAPTER 9 and 10],
  Ian Goodfellow, Yoshua Bengio, and Aaron Courville, MIT Press, 2016, link
- All the links in the slides

Extra

- **Master Thesis Available! (massimiliano.ruocco@ntnu.no) (Integrated PhD?)**
  - **Where**:
    - *Sintef Digital / NTNU / Collaboration with industries*
  - **Topics**:
    - *Deep Learning,*
    - *Transformers,*
    - *Generative Adversarial Network,*
    - *Attention Mechanism,*
    - *Image-To-Image Translation,*
    - *Self-Supervised Learning,*
    - *Time Series Analysis*
  - **Applications**:
    - *Healthcare,*
    - *Smart Building,*
    - *Manufacturing,*
    - *Energy,*
    - *Telco*

Introduction to Deep Learning

A gentle introduction

Massimiliano Ruocco
Adj. Associate Professor, DART Group, IDI
Senior Researcher, Sintef Digital

massimiliano.ruocco@ntnu.no