

Neural Networks: History and foundation

Amor Ben Tanfous

Aimen Zerroug

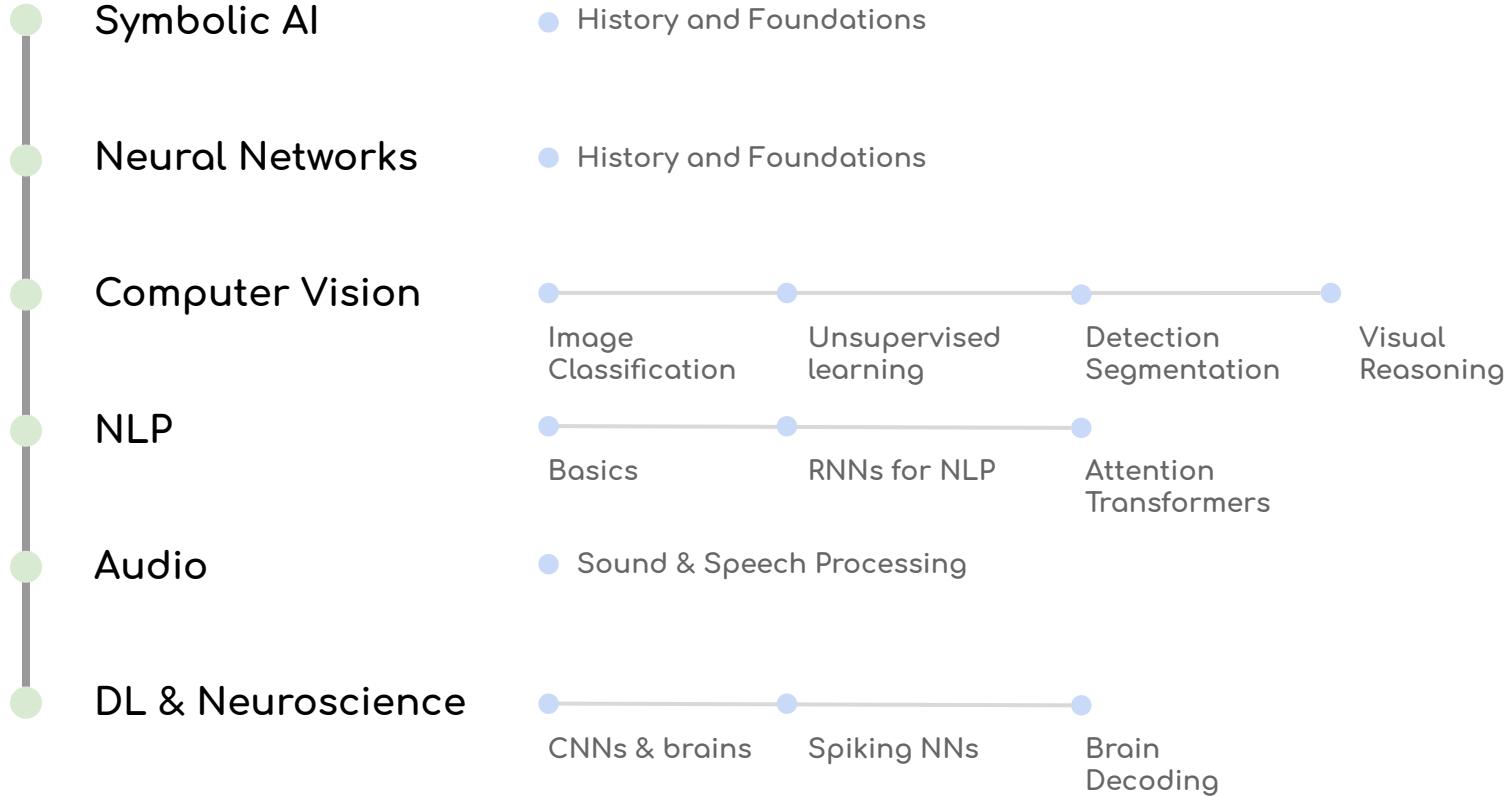
Artificial and Natural Intelligence Toulouse Institute (ANITI)
March 22, 2021



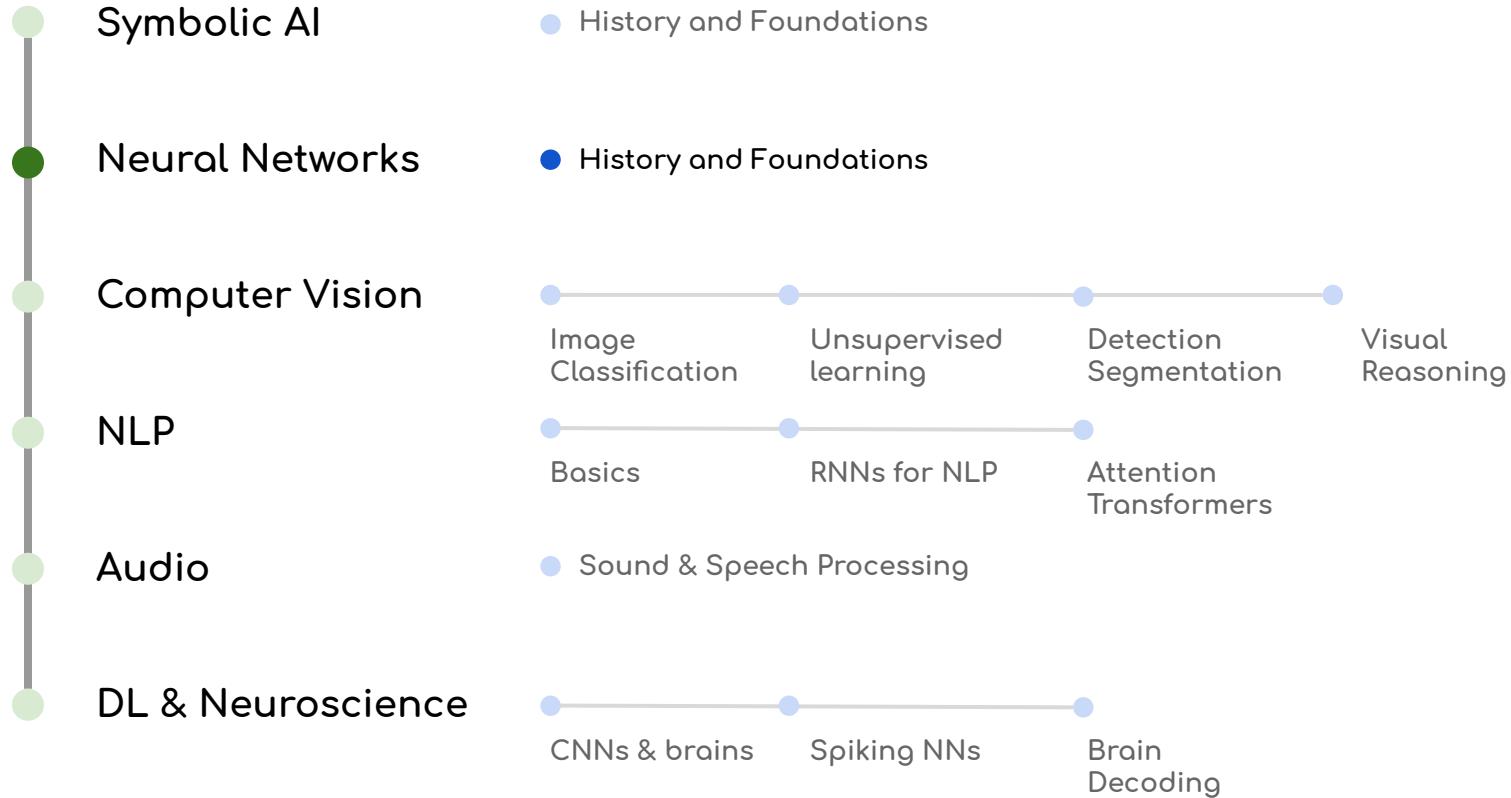
Course Intro

- Symbolic AI
- Neural Networks
- Computer Vision
- NLP
- Audio
- DL & Neuroscience

Course Intro



Course Intro



Outline

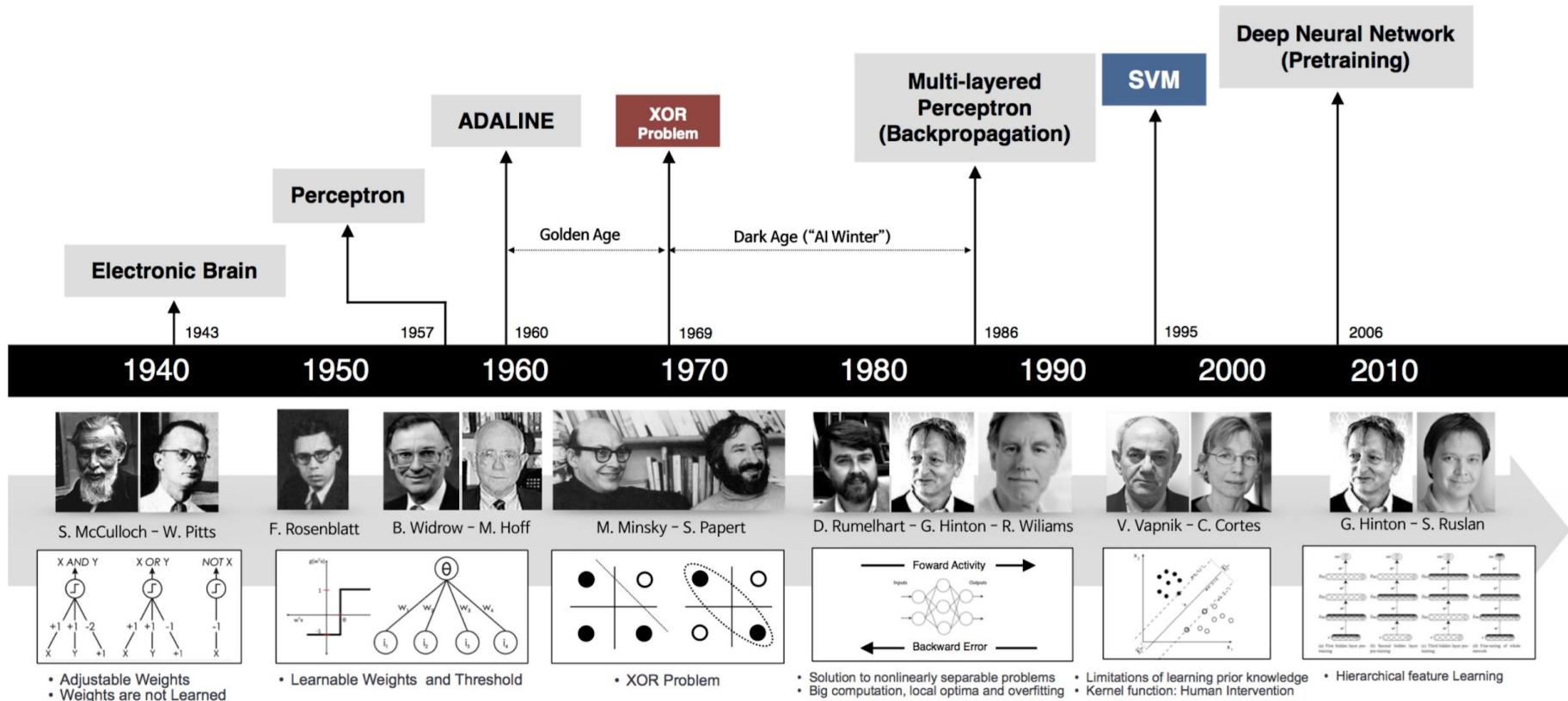
Session 1

- History of neural networks
- Artificial neurons - Perceptrons
- Multi-layer perceptrons (MLPs)
- Optimization and objective functions
- Gradient descent and Back-propagation

Session 2

- How to design neural networks
- Choosing the architecture (CNN, RNN)
- Choosing the loss function
- Training a neural network
- Practical examples

Neural networks milestones



The beginning of Neural nets (1940s-1960s)

Artificial neuron: McCulloch & Pitt's neuron model (1943)

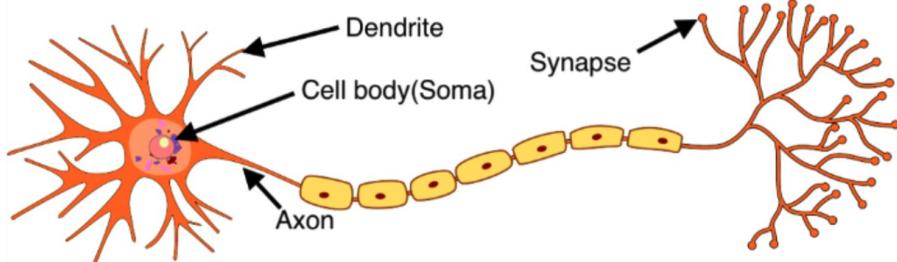
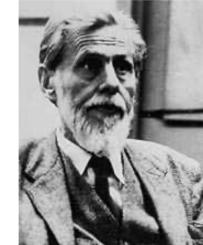
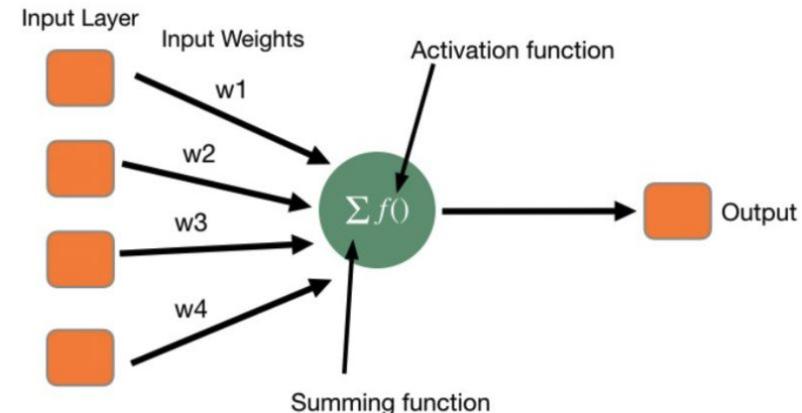


Image source: Wikimedia Commons

Schematic of a biological neuron



Model of an artificial neuron

The beginning of Neural nets (1940s-1960s)

Artificial neuron: McCulloch & Pitt's neuron model (1943)

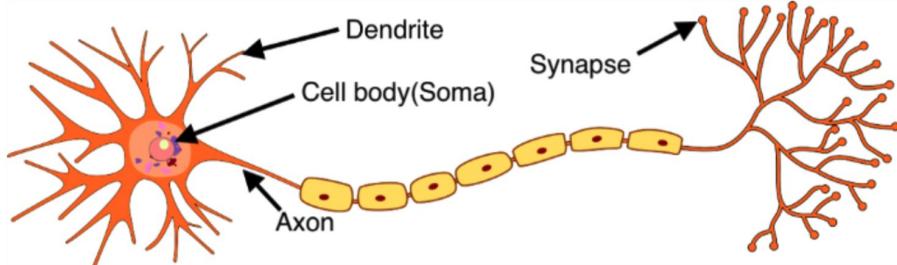
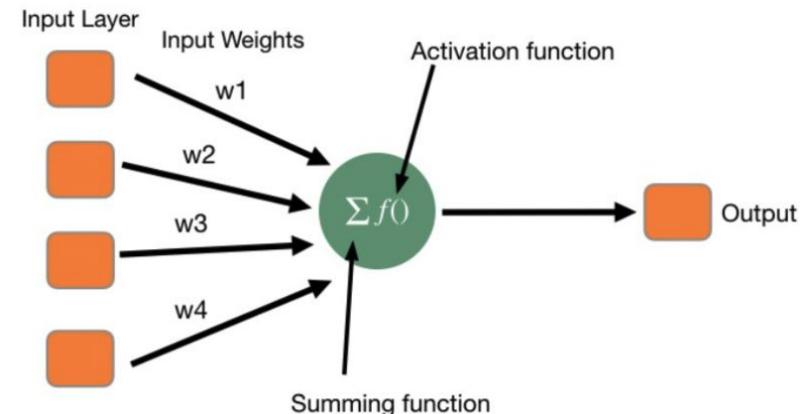


Image source: Wikimedia Commons

Schematic of a biological neuron



$$\text{output} = \begin{cases} 0 & \text{if } t > \sum_{i=0}^n w_i x_i \\ 1 & \text{otherwise} \end{cases}$$

Single-layer neuron examples

- An artificial neuron could solve **linear** logical problems: AND, OR, NOT

AND

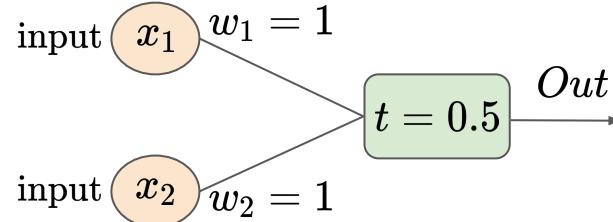
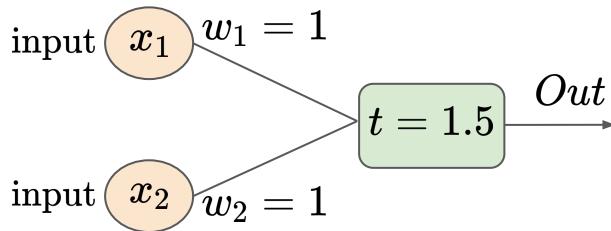
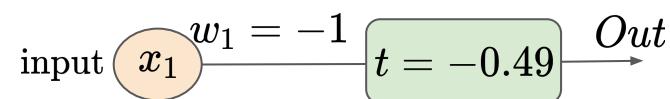
x_1	x_2	<i>Out</i>
0	0	0
0	1	0
1	0	0
1	1	1

OR

x_1	x_2	<i>Out</i>
0	0	0
0	1	1
1	0	1
1	1	1

NOT

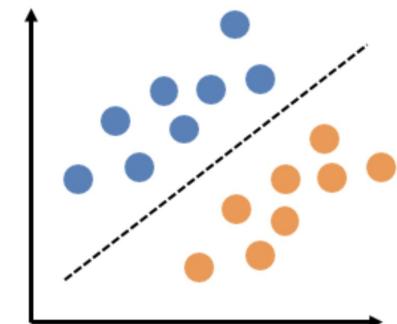
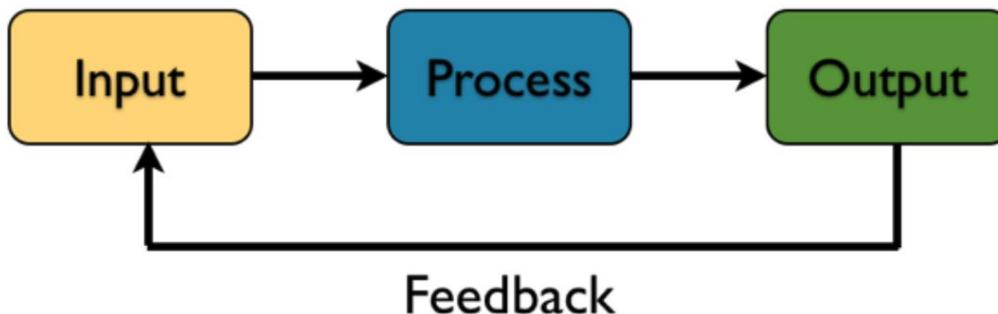
x_1	<i>Out</i>
0	1
1	0



Perceptron: A learning algorithm for the neuron model

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton Project Para.*

- Automatic learning of weights
- Supervised learning of binary classifiers
- Could recognize letters and numbers



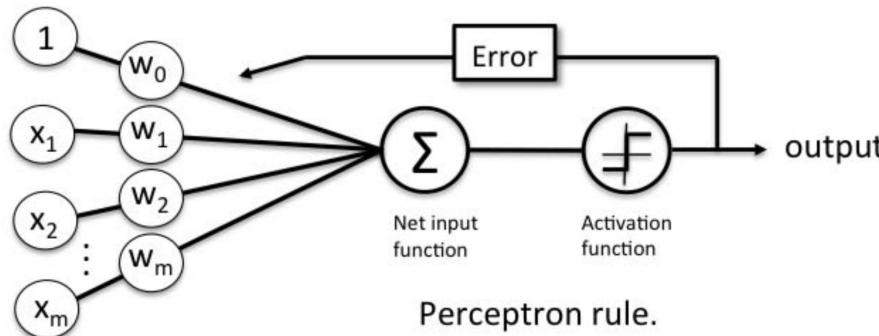
The Perceptron Learning Algorithm

Let $D = (\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \dots, \langle x_n, y_n \rangle) \in (\mathbb{R}^m \times \{0, 1\})^n$

1. Initialise w_i with random small values
2. For every training epoch:

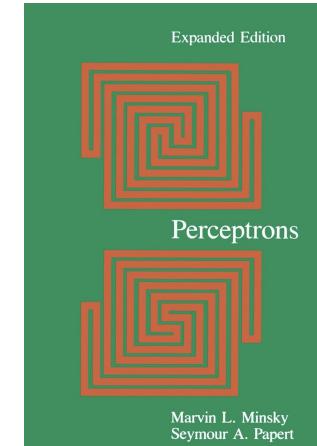
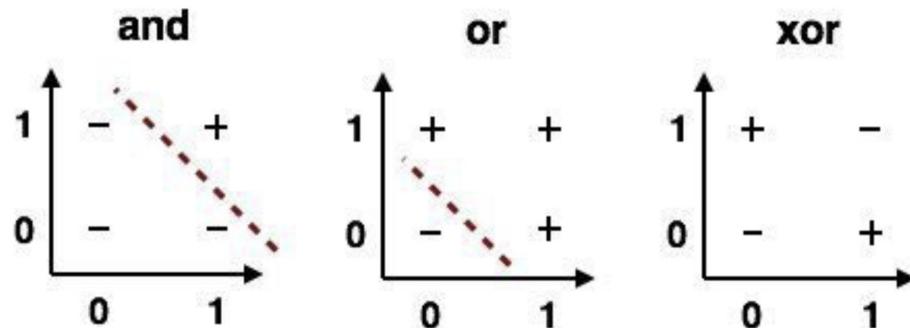
For every sample $\langle x_i, y_i \rangle \in D$:

- (a) $\hat{y} := \sigma(x_i \times w)$ ← Compute output (prediction)
- (b) $err := (y_i - \hat{y}_i)$ ← Compute error
- (c) $w := w + err \times x_i$ ← Update parameters



The first AI winter

Minsky and Papert (1969) show that the perceptron can't even solve the XOR problem

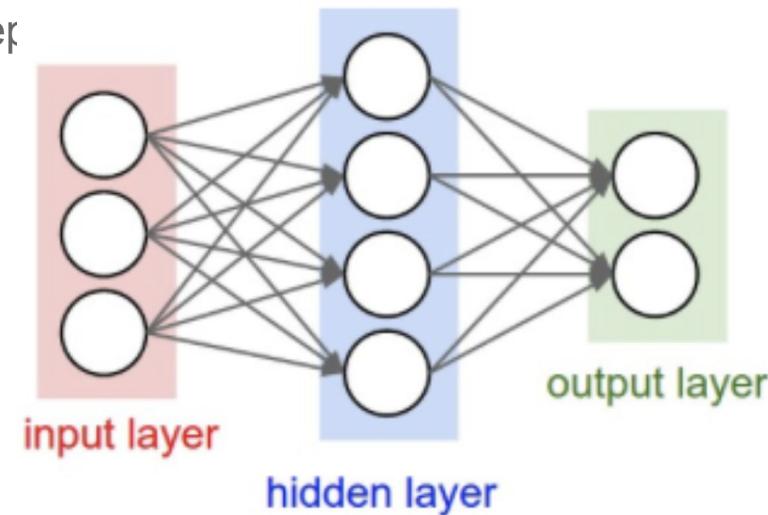
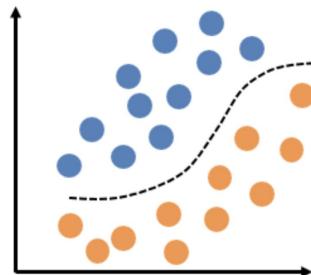


⇒ Kills research on neural nets for the next 15-20 years

Multilayer perceptrons (1980's)

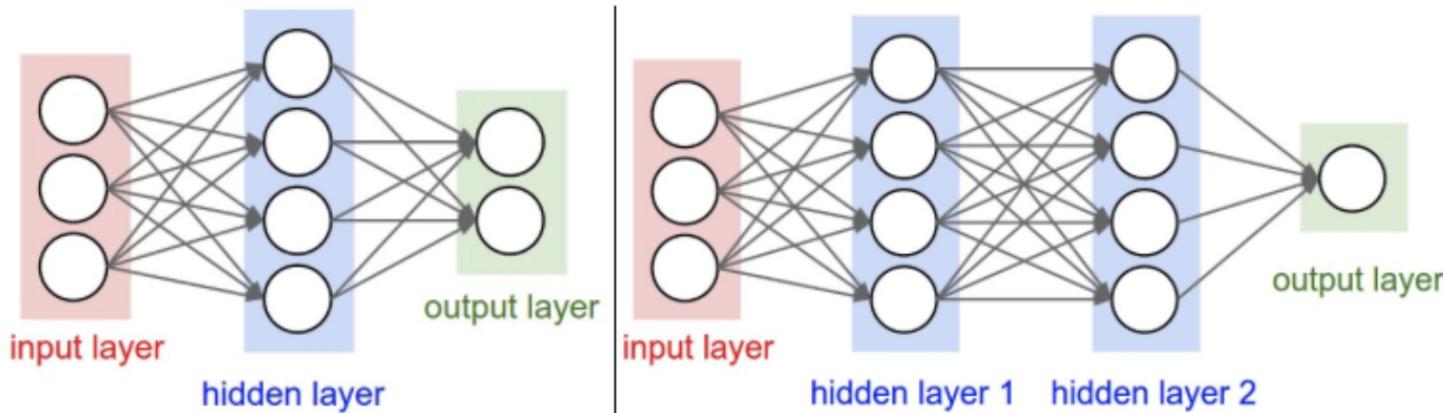
Solution to the XOR problem: Multilayer perceptrons

- Composed of: **input layer**, **hidden layer(s)** and an **output layer**.
- Each node (of hidden and output layers) is a neuron that uses a **nonlinear activation function**.
- It can distinguish data that is **not linearly separable**.



Multilayer perceptrons (1980's)

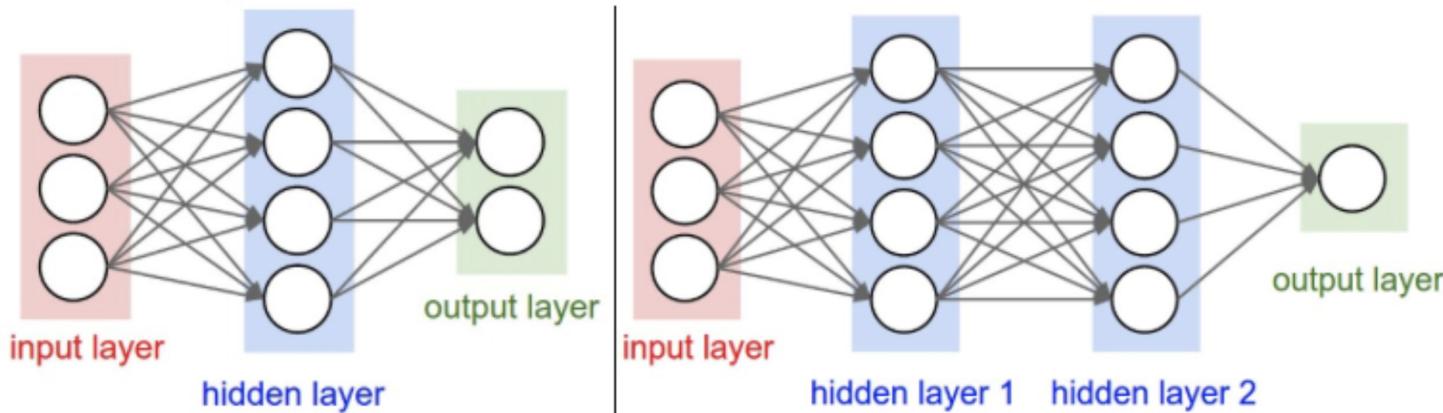
Can add more layers to increase capacity of the network



- New problem: MLPs are hard to train!

Multilayer perceptrons (1980's)

Can add more layers to increase capacity of the network



- New problem: MLPs are hard to train!
- ⇒ Solution: The **Backpropagation** algorithm

The Backpropagation algorithm

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California,
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,
Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure¹.

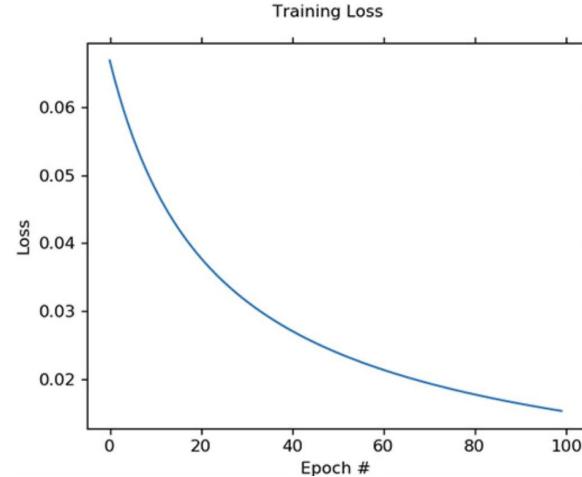
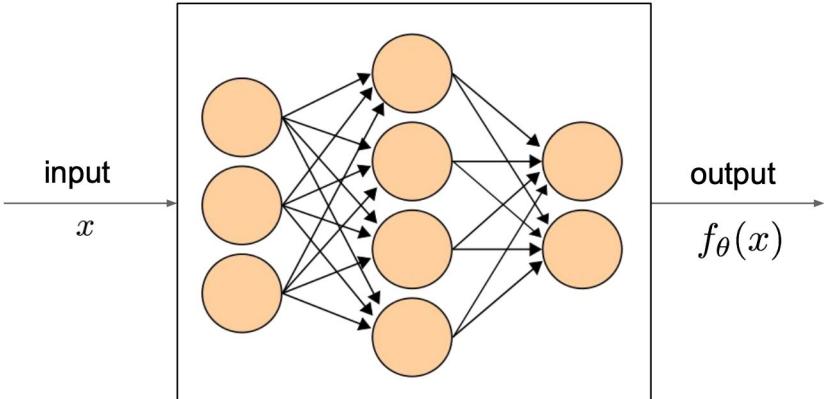


Rumelhart, Hinton, and Williams (1986) introduced Backpropagation to train MLPs

Principle: Computing the gradient of the cost function w.r.t the weights of the network

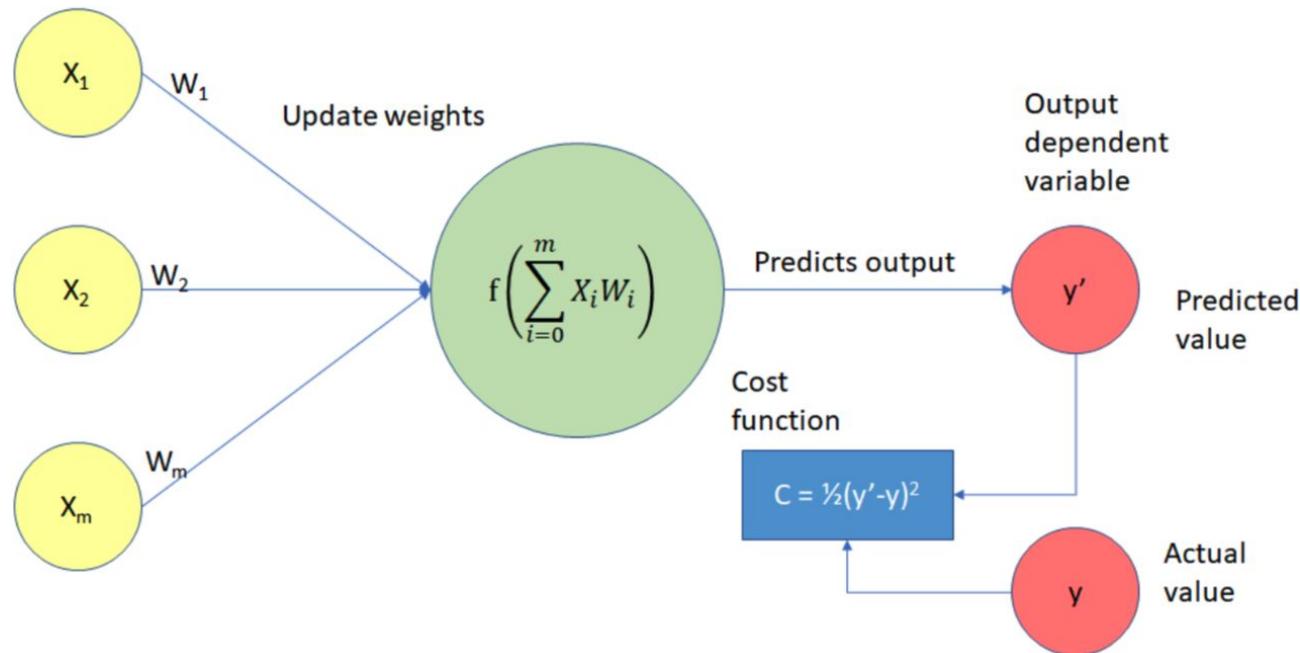
Neural Network learning as optimization

- Mapping a set of inputs to a set of outputs from training data
- Learning is cast as an **optimization** problem to make good enough predictions
- Training with **gradient descent**



Cost functions

- A cost function is a measure of error between predictions and true values



Cost functions

- A cost function is a measure of error between predictions and true values
- Guides the training process to find a set of weights that minimizes its value

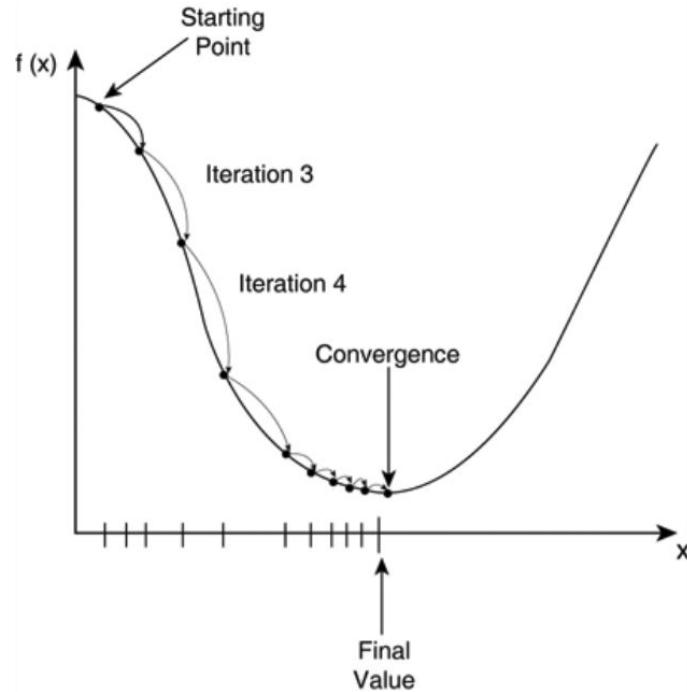
Some examples:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

test set predicted value actual value

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

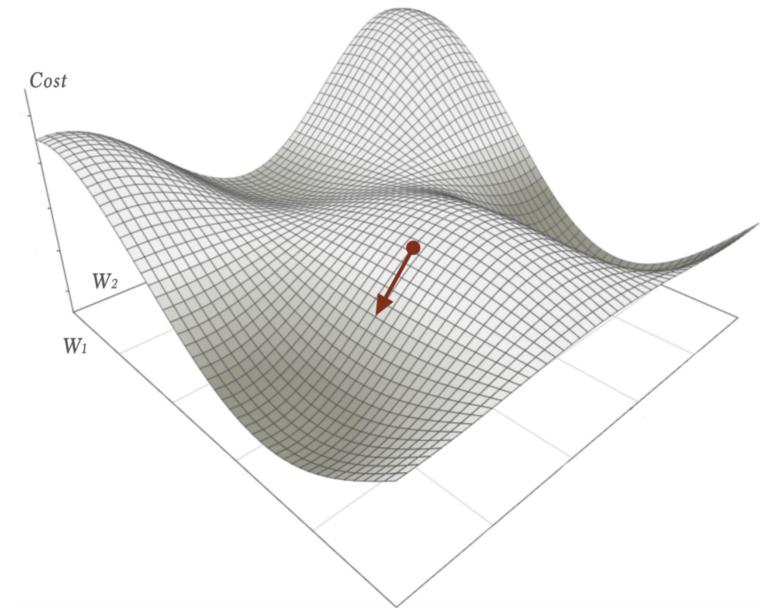
test set predicted value actual value



Gradient Descent

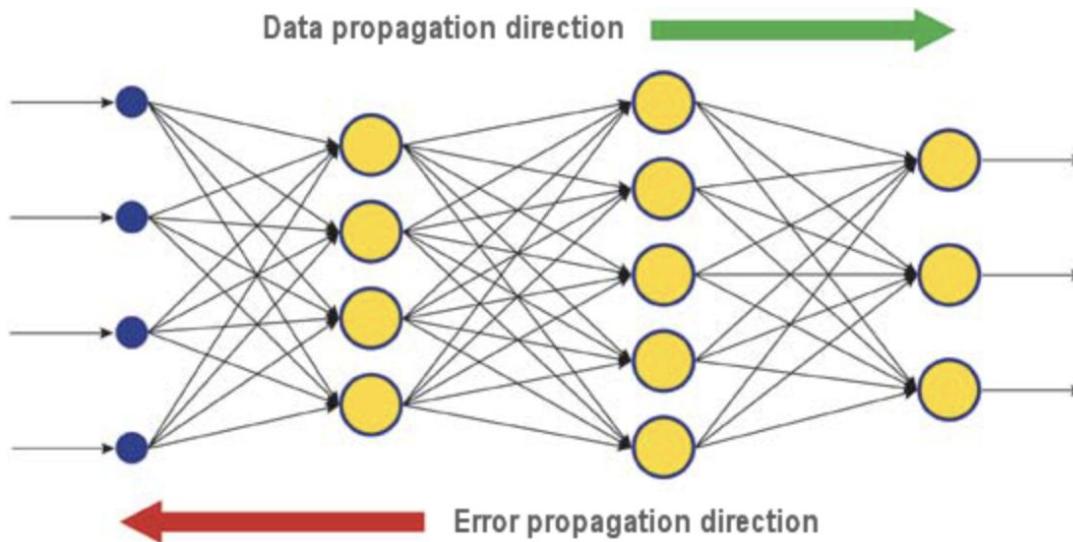
- Finding minimum of the cost function C
- Negative gradient ∇C points in the direction where the function decreases most rapidly
- Calculate new weights: $W^+ = W - \eta \nabla C$

$$\begin{bmatrix} w_1^+ \\ w_2^+ \\ \vdots \\ w_n^+ \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_n} \end{bmatrix}$$



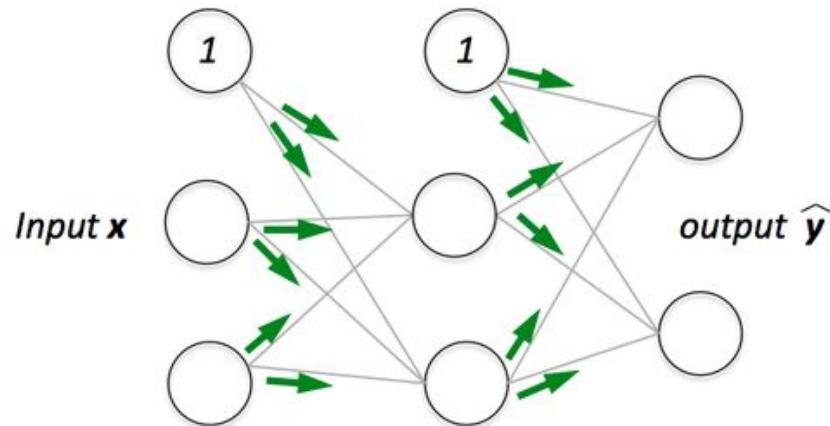
How Backpropagation works

- Minimization through **gradient descent** requires computing the gradient
- Backpropagation: way to compute the gradient by applying the **chain rule**



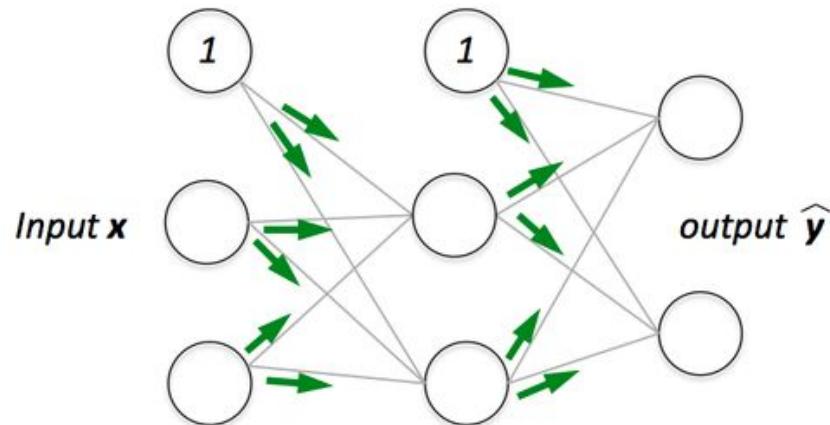
How Backpropagation works

- 1) Forward pass: propagate data through the network to get predictions



How Backpropagation works

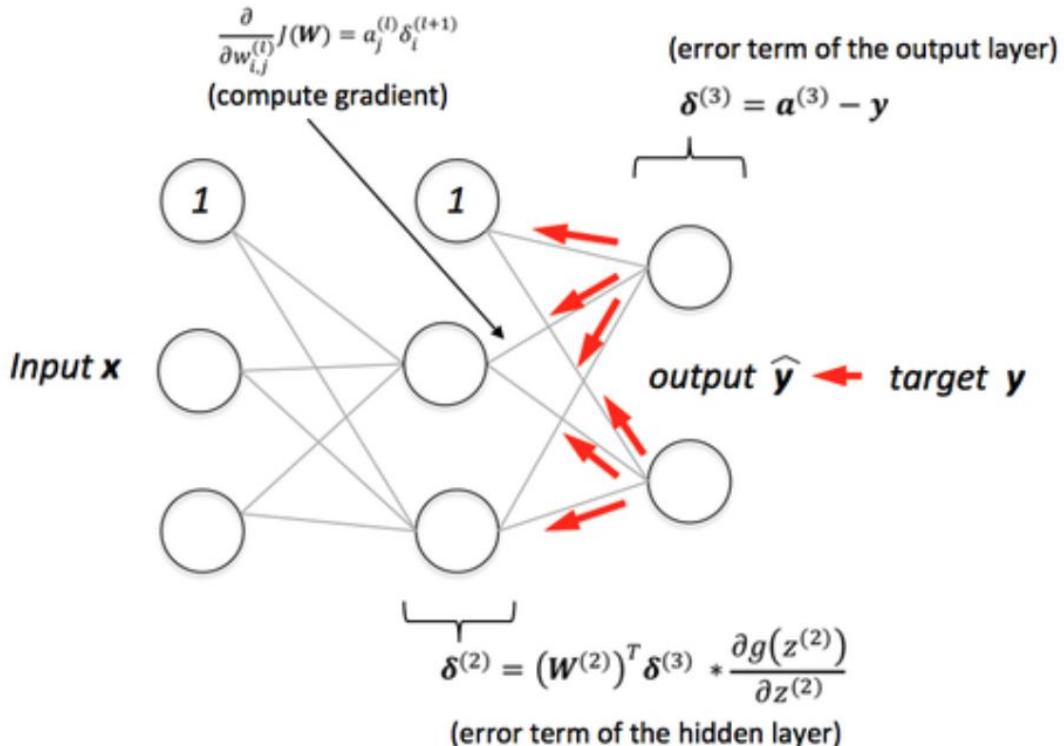
- 1) Forward pass: propagate data through the network to get predictions
- 2) Calculate the total error w.r.t desired outputs



How Backpropagation works

- 1) Forward pass: propagate data through the network to get predictions
- 2) Calculate the total error w.r.t desired outputs
- 3) Backward pass:

a) Compute partial derivatives of the error w.r.t each weight $\frac{dE}{dw_{ij}}$ by applying the chain rule

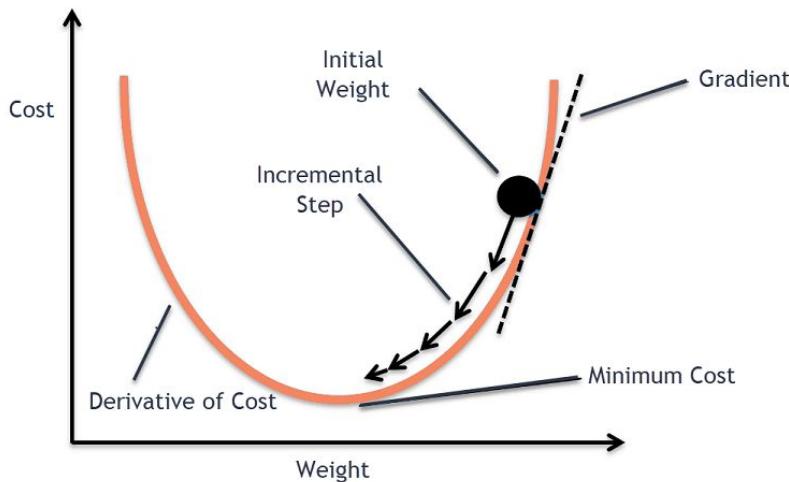
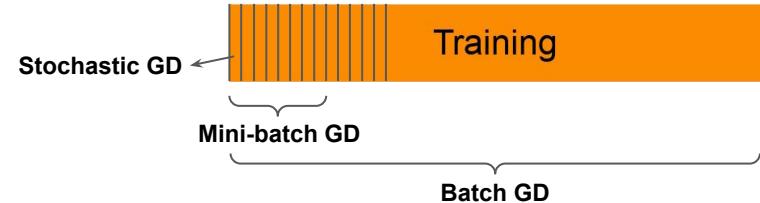


b) Update weights:

$$w_i = w_i - \eta \frac{dE}{dw_{ij}}$$

Training with Gradient descent and BP

- batch GD, stochastic, or mini-batch?
- SGD in DL generally refers to mini-batch GD



Some useful resources

<http://neuralnetworksanddeeplearning.com/chap1.html>

<https://towardsdatascience.com/part-2-gradient-descent-and-backpropagation-bf90932c066a>

<https://towardsdatascience.com/a-concise-history-of-neural-networks-2070655d3fec#.ekc89166m>

<https://people.idsia.ch/~juergen/who-invented-backpropagation.html>

Training a neural network

Data

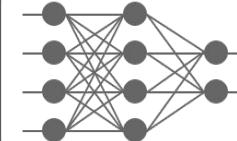
Data Samples
and Labels

$$\{(x_1, y_1), (x_2, y_2), \dots\}$$

Task

Input and output
 $x_i \rightarrow y_i$

Architecture



Loss function

$$loss = diff(prediction, label)$$

Optimization

prediction → backpropagation → gradient descent

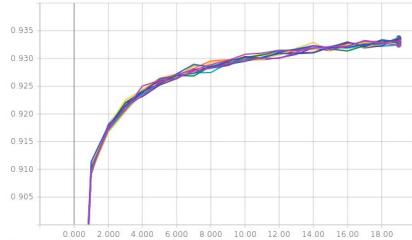


Training a neural network

Data

Handwritten Digit Dataset

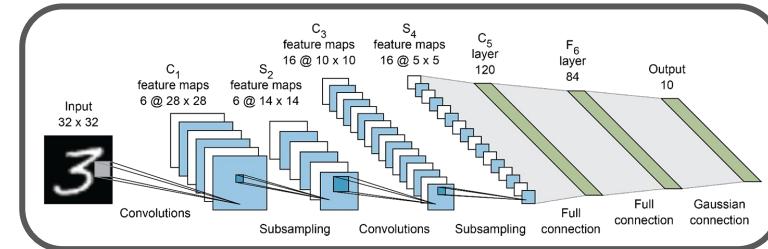
```
0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9
```



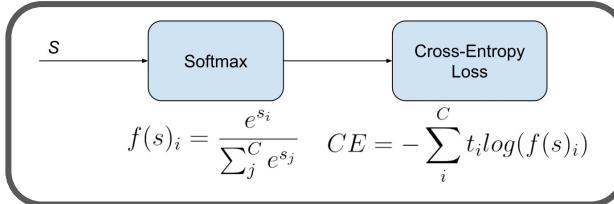
Task



Architecture



Loss function

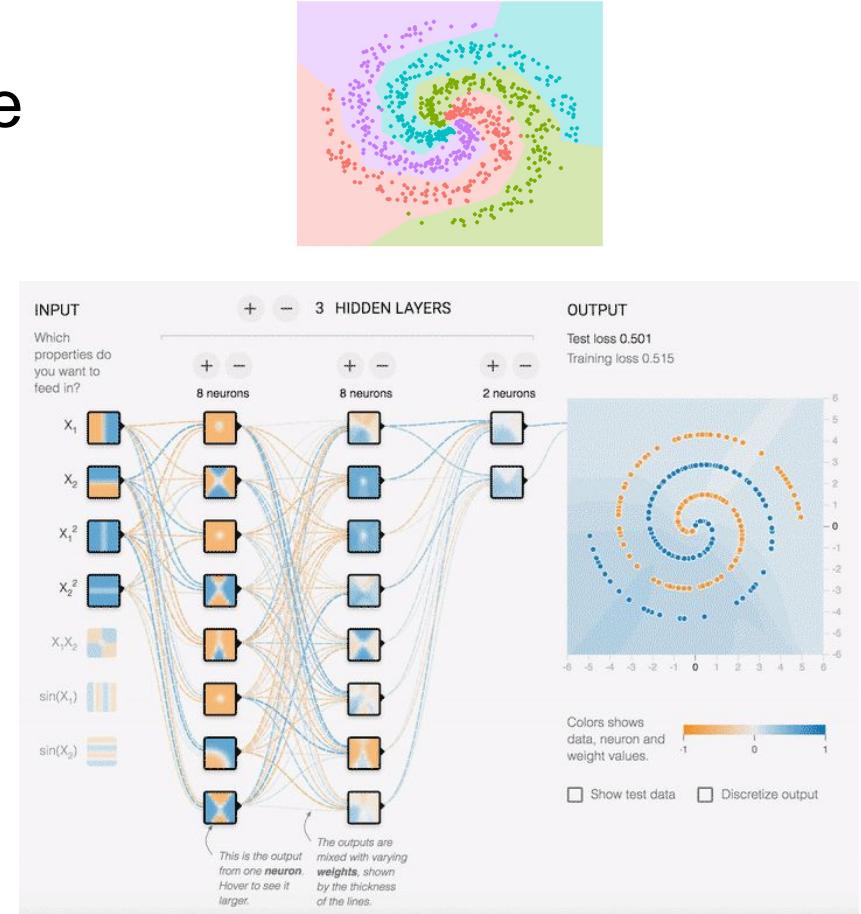
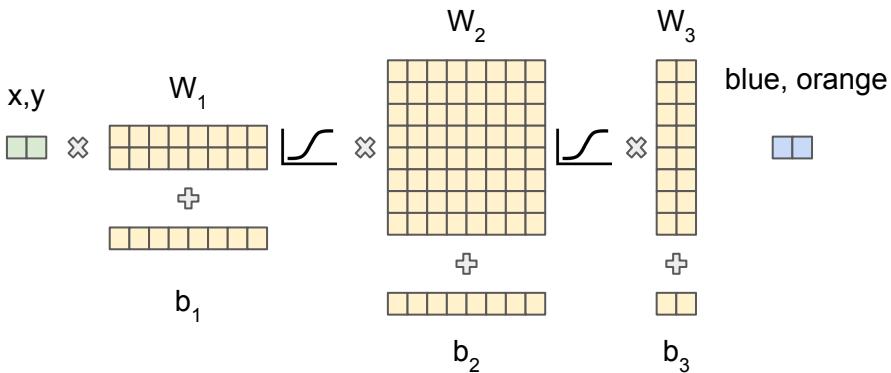


Optimization



How to choose the architecture

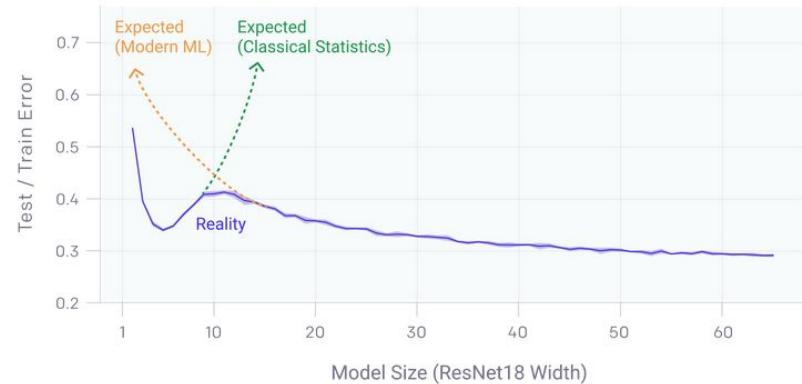
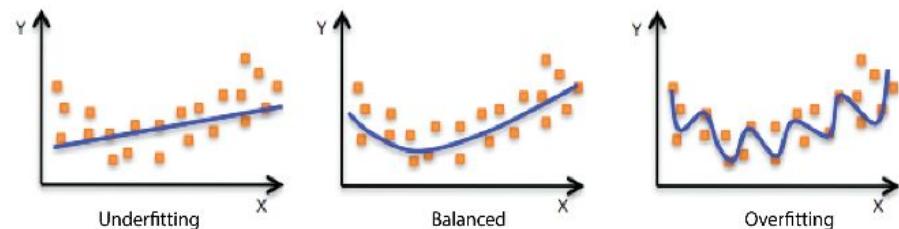
Multi-layer perceptrons **MLPs** are the standard solution for data with simple structure (ex. tabular data).



How to choose the architecture

What's important ?

- Generalization
 - Fitting the data distribution
 - Fitting unseen examples
- Efficiency
 - Memory (how large is the model ?)
 - Time (how long does it take to train ?)
 - Data (how many training samples does it need ?)



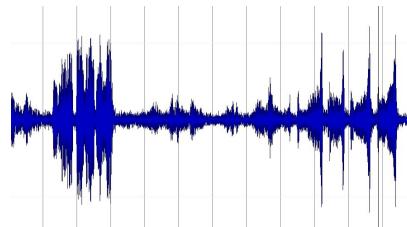
How to choose the architecture

Do MLPs work for all types of data ? **Yes, but not efficiently**

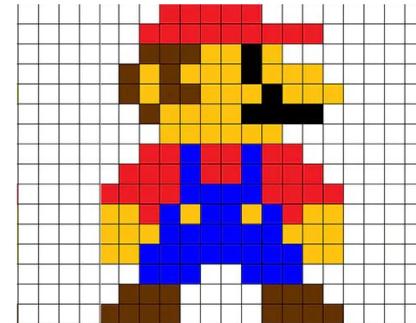
text

I am a student

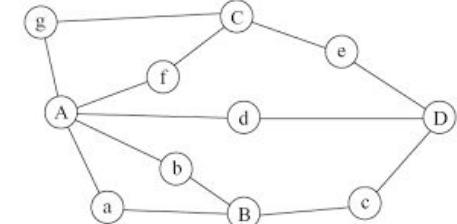
time
sequences



images

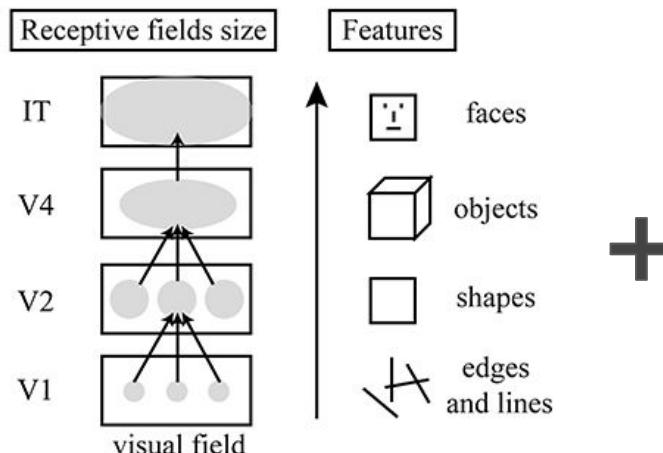


graphs



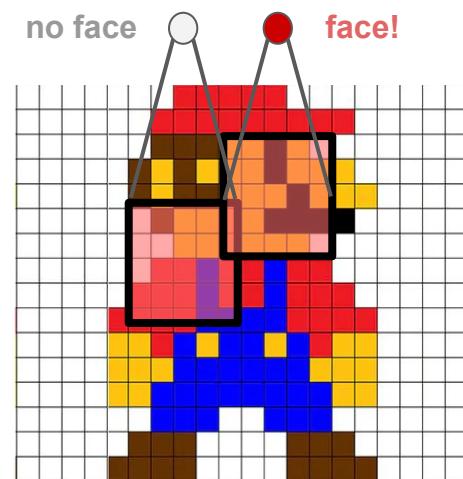
Inductive bias : an architectural assumption or constraint

Convolutional Neural Network (CNN)

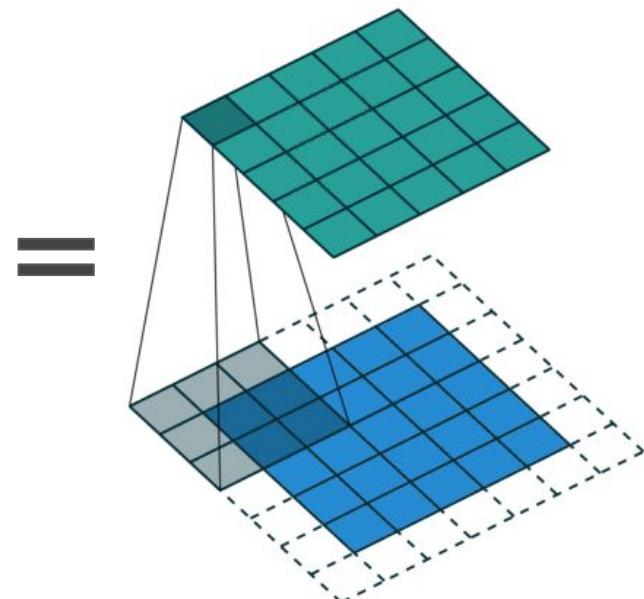


Local Connectivity

Hierarchical Processing

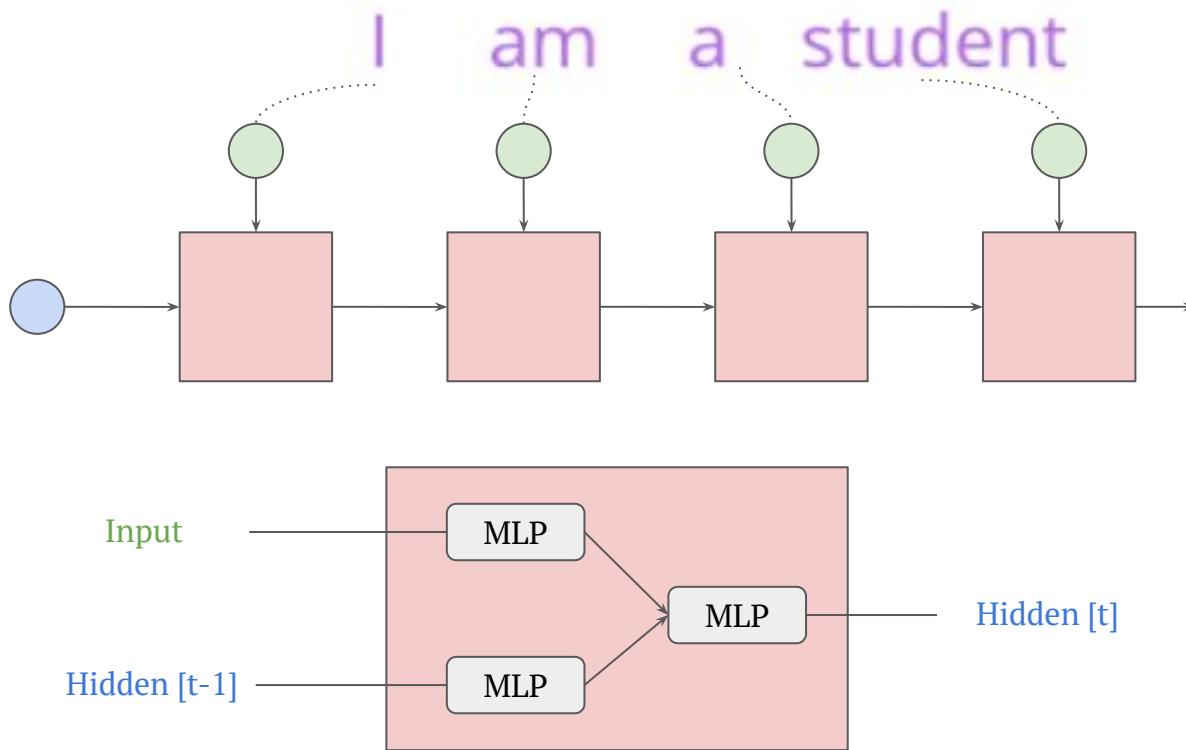


Weight sharing



Convolution

Recurrent neural networks (RNN)

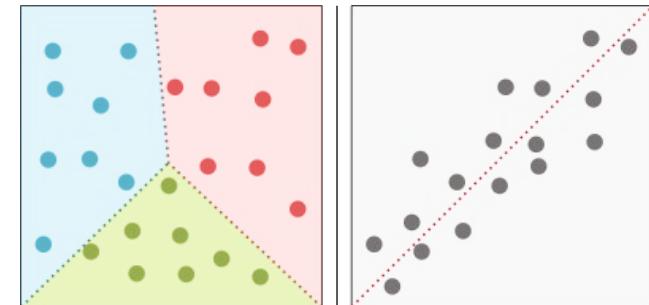
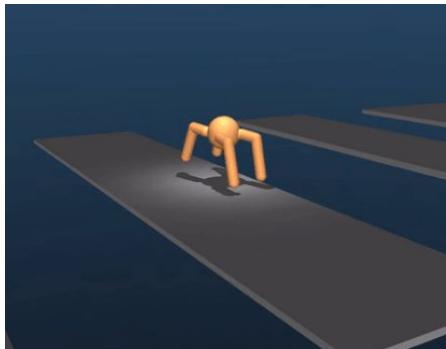


How to choose the loss

The choice of the loss is crucial !

What's important in the loss design?

- Adaptability to the problem (correlates with performance metrics)
- Continuous and differentiable
- Numerically stable



Classification

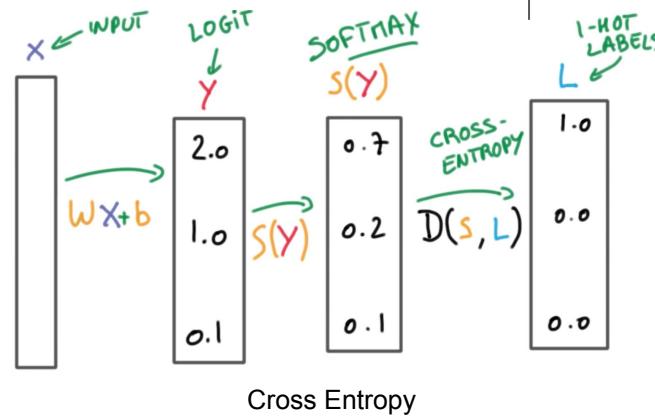
Regression

Label : Categorical (discrete)

Continuous

Examples : Cross Entropy
Hinge Loss

Mean Square Error (L2)
Mean Absolute Error (L1)



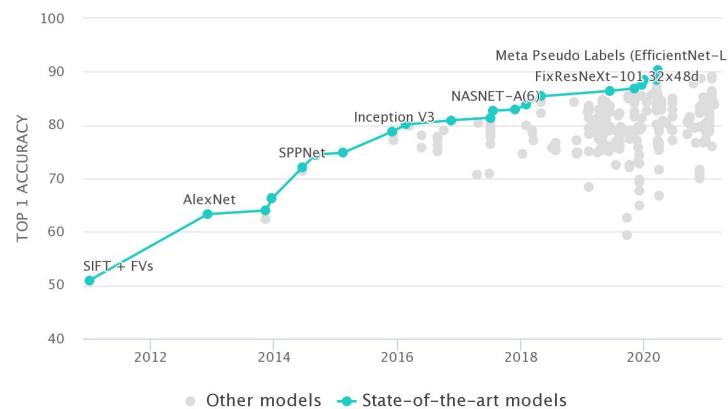
Training and Evaluation

Training: optimization of the model

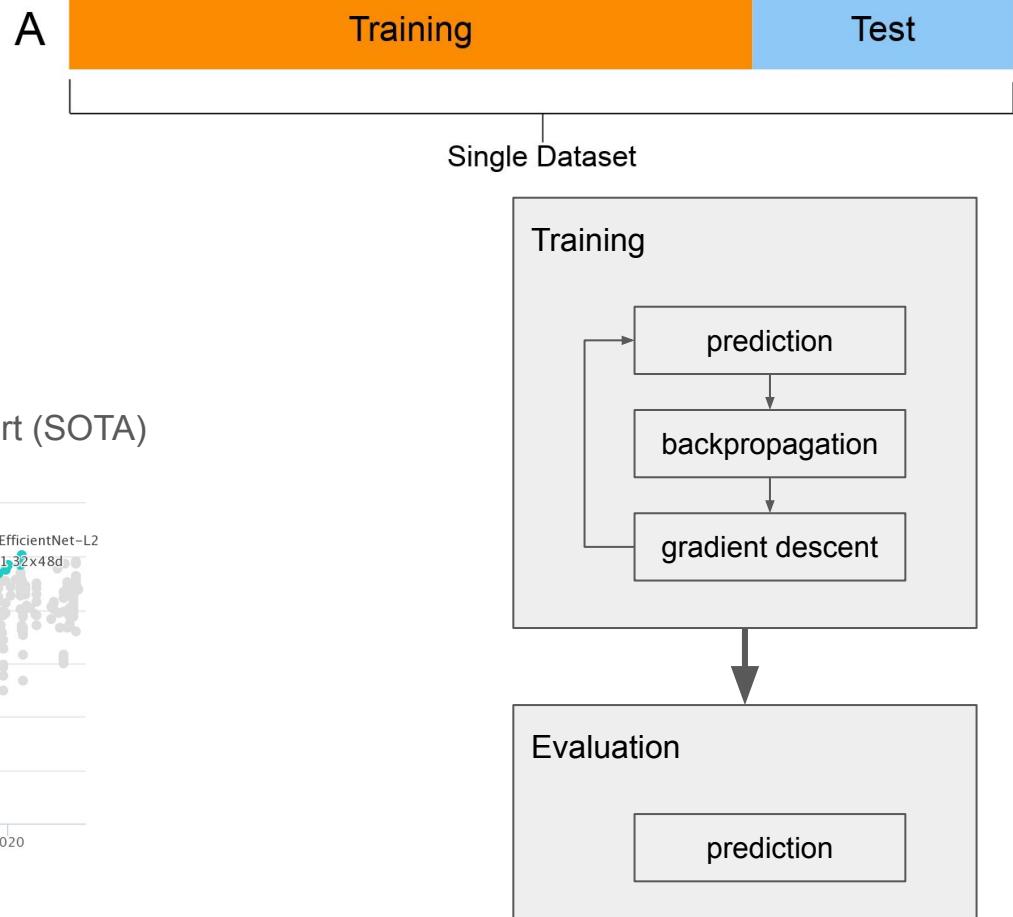
Evaluation: testing generalization

Metrics: Loss and accuracy

Models with the highest accuracy are state of the art (SOTA)



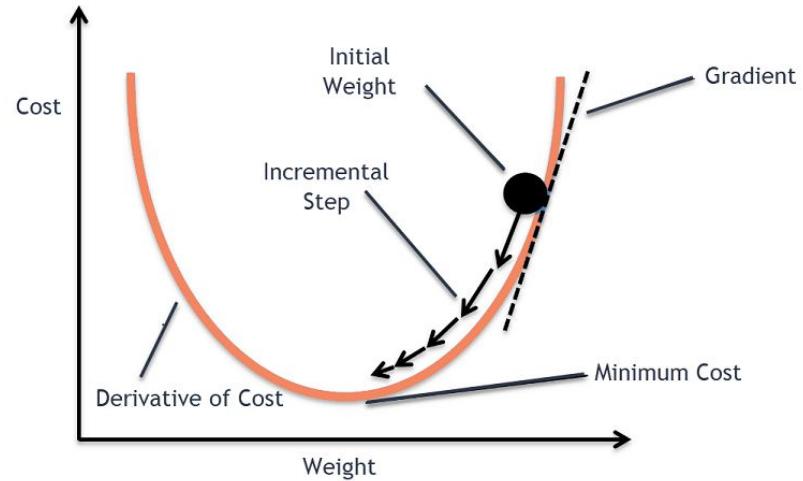
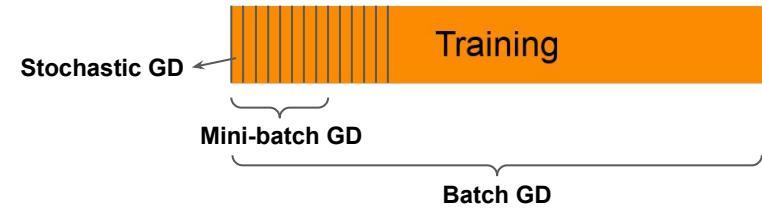
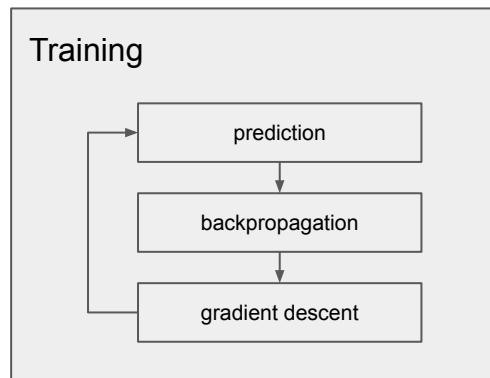
Imagenet Benchmark on paperswithcode.com



Training

SGD in DL generally refers to mini-batch GD

Epoch: one pass through the dataset

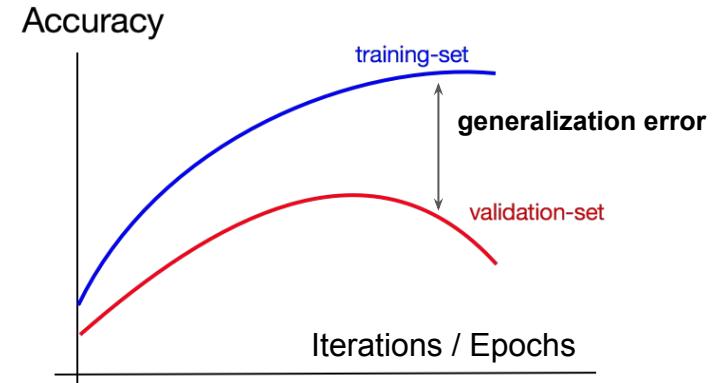
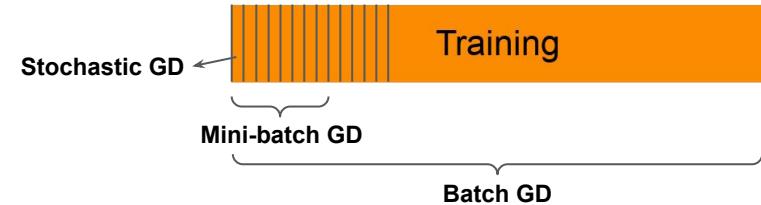


Training

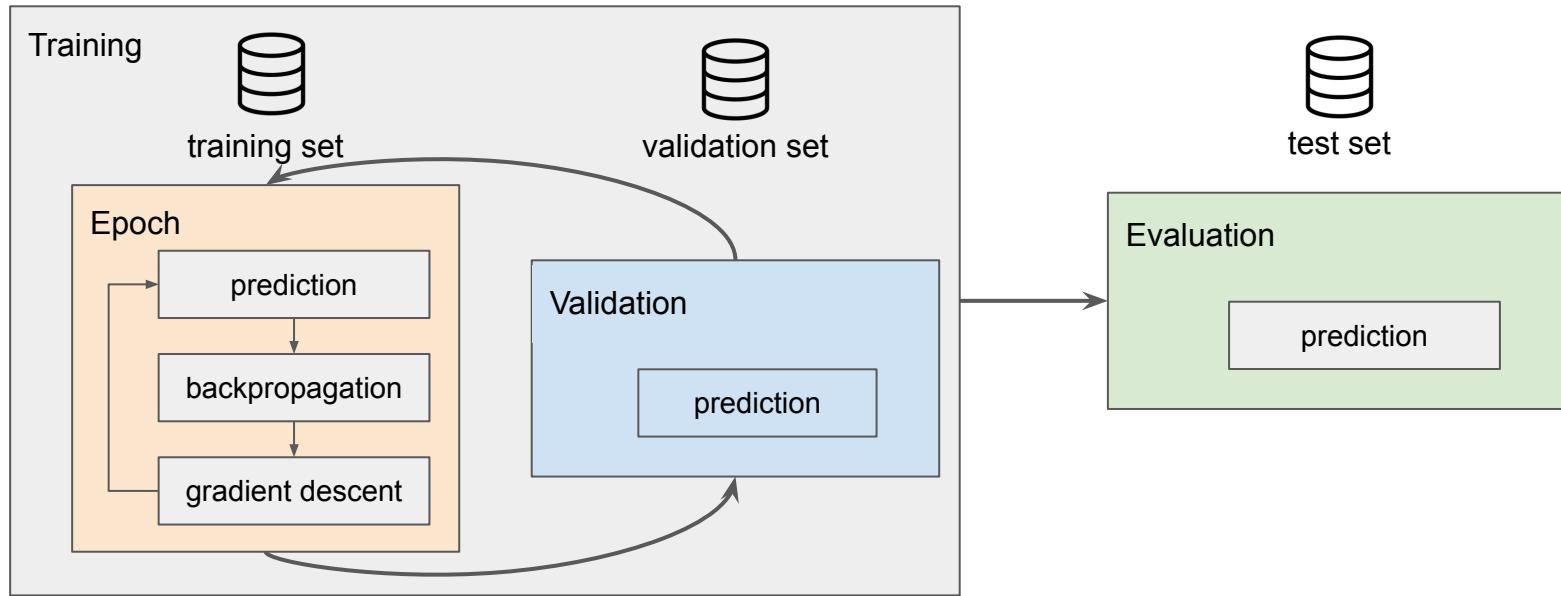
How many iterations/epochs ?

The validation set is used for early stopping

Test set \neq validation set



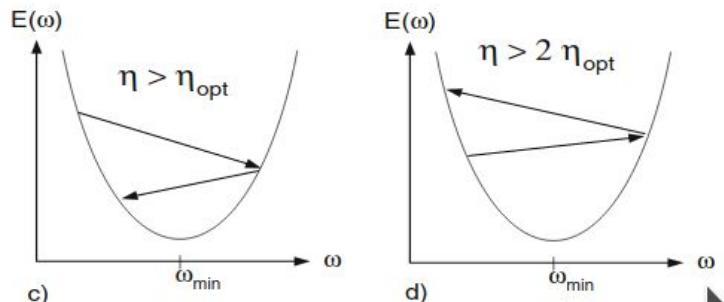
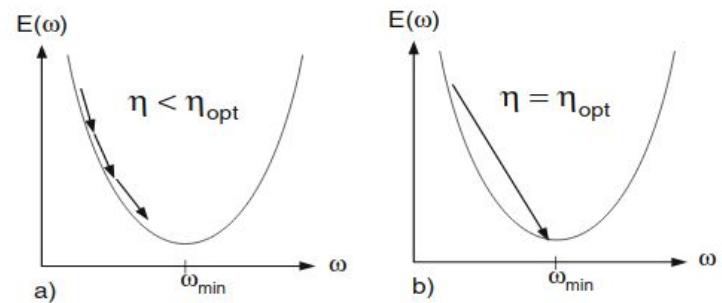
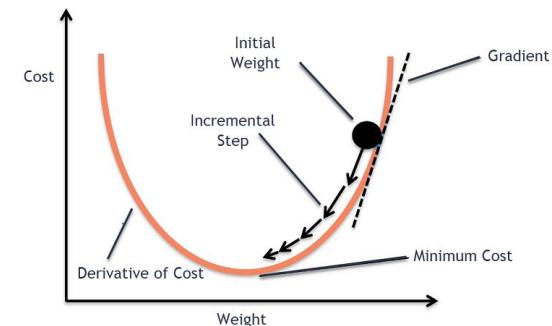
Training and Evaluation



Training

How to find the learning rate ? η

$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$



Optimizers

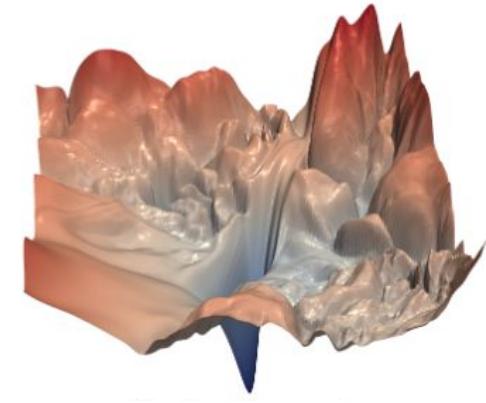
$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

Loss landscapes are not easy to navigate for optimizers

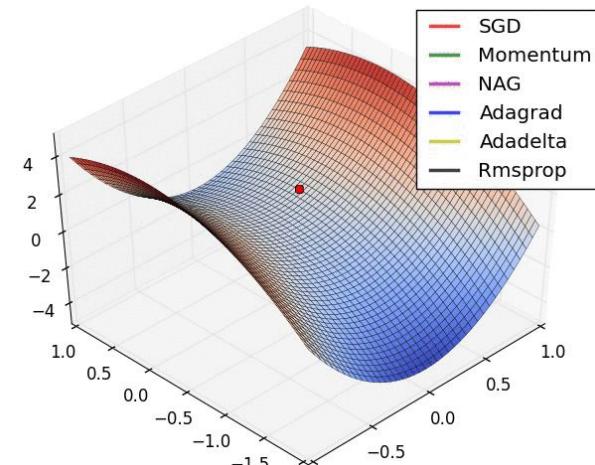
Ideas:

- Use the gradient history of previous timesteps to inform GD in future timesteps
- Adapt the learning rate to each parameter

Adam optimizer is an extension of SGD which makes use of these two ideas. It is currently the most used optimizer in DL after SGD.



This is a 2 parameter example!
Imagine millions



Regularization

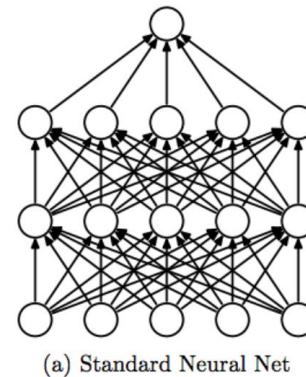
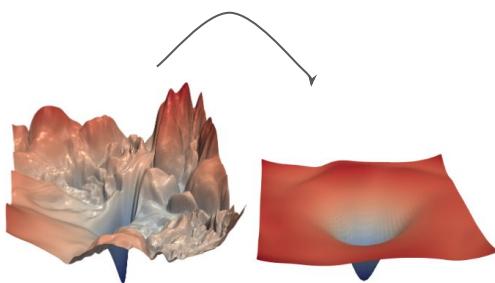
Additional constraints to reduce overfitting

Dropout: stochastically dropping weights during inference

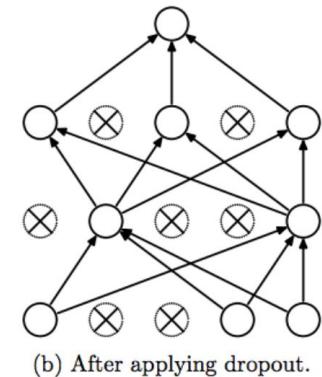
Early stopping: stopping the training as soon as the validation loss starts increasing

Weight penalties (weight decay): L1 norm (Lasso) / L2 norm (ridge). Terms added to the loss.

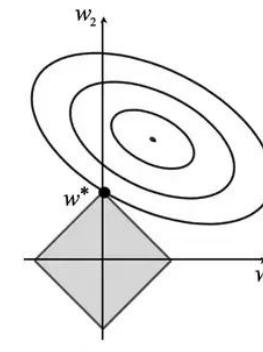
Data augmentations: artificially boosting the number of training samples



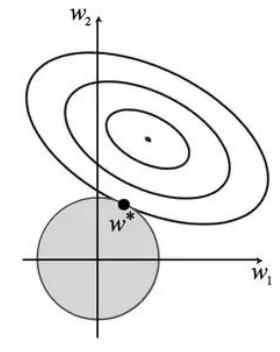
(a) Standard Neural Net



(b) After applying dropout.



L1
Sparse weights



L2
Smaller weight values

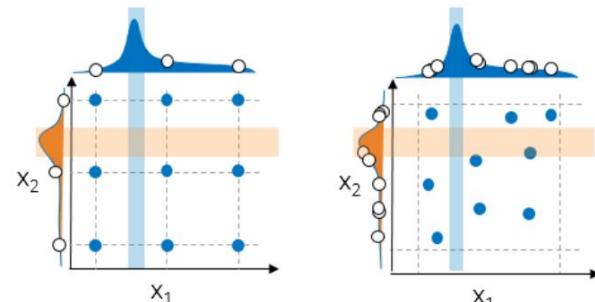
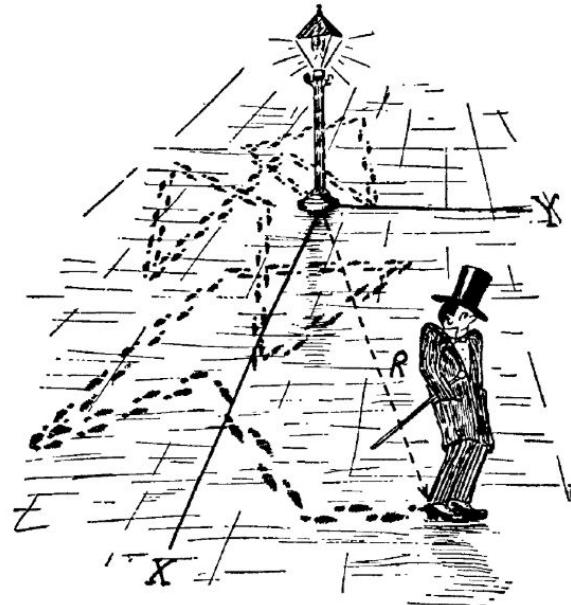
Hyperparameters

All parameters and settings that are set before training:

- Architectural choices: types and number of layers, size of each layer
- Losses/regularization: weights, additional constants
- Optimization: batch size, learning rate, iterations/epochs, schedule

Hyperparameter search. Another optimization problem ?

- Often done manually.
- When resources are available, large scale search is possible



(a) Standard Grid Search

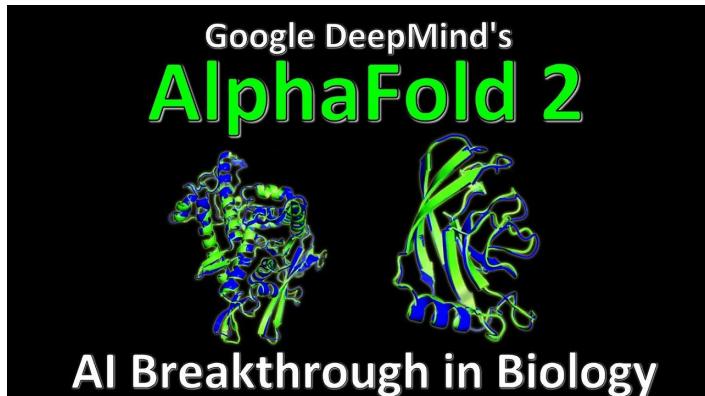
(b) Random Search

The machine learner pipeline

1. Understanding the data and the task
2. Set up the end-to-end training/evaluation skeleton
 - a. a basic architecture
 - b. a standard loss
 - c. Standard optimization pipeline
3. Complexify one thing at a time
4. Regularize
5. Tune Hyperparameters

Practical examples

With this formula you can do all of this



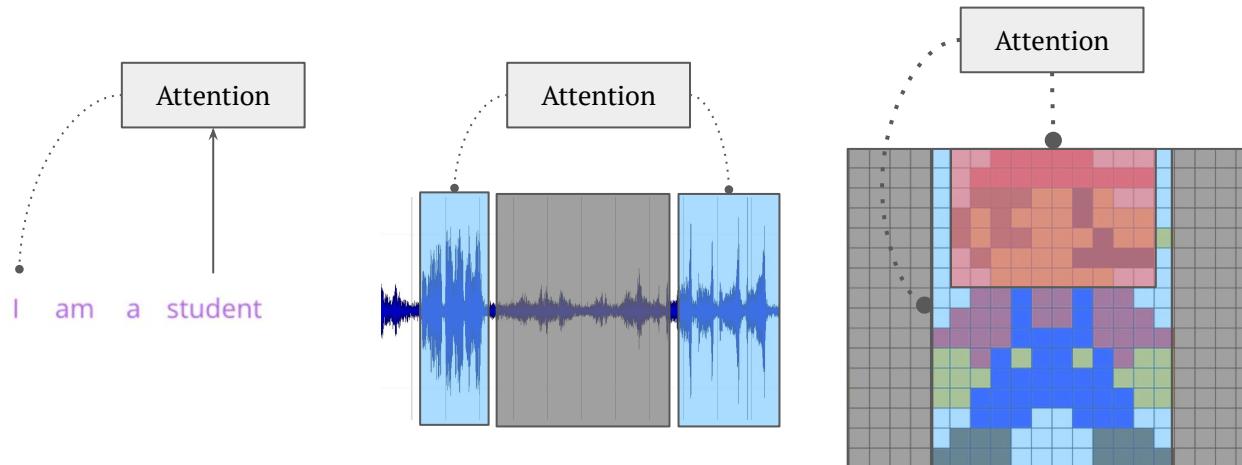
DeepMind AI Reduces
Google Data Centre
Cooling Bill by 40%

This was 5 years ago

Go to notebook

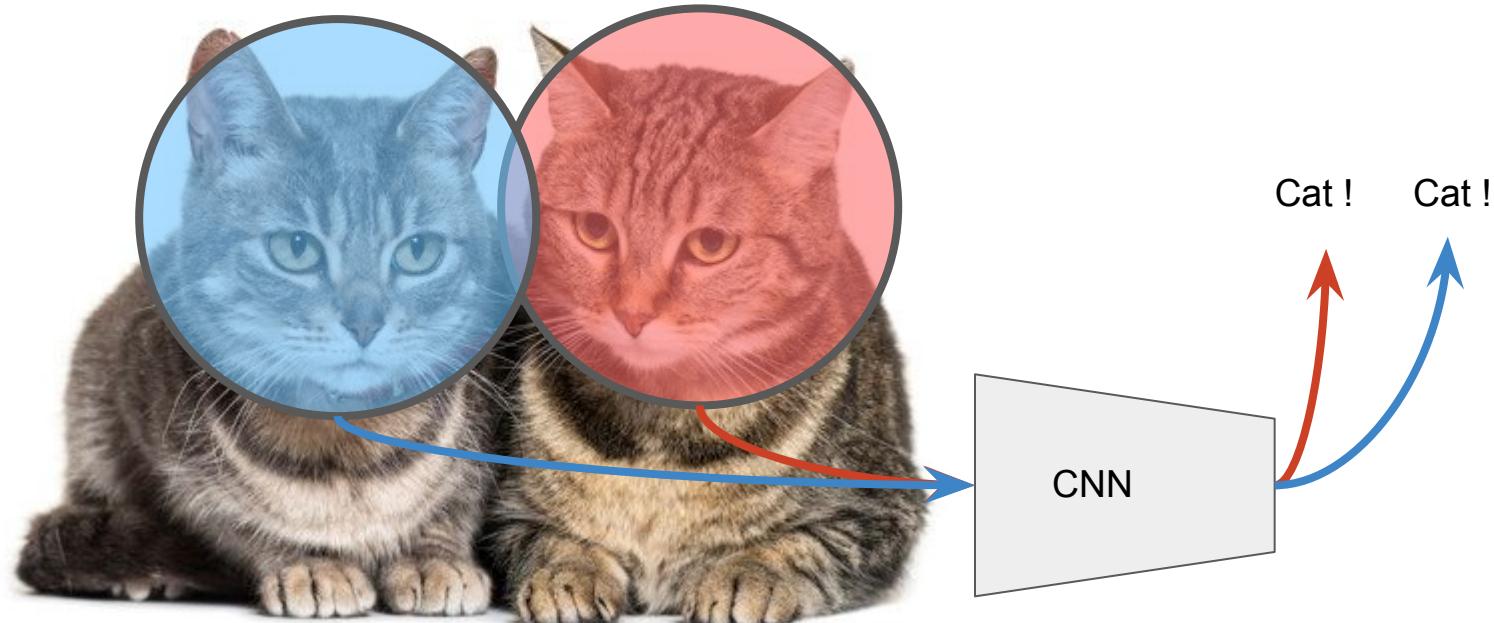
Attention

Attention as a design principle



Convolutional Neural Network (CNN)

Local Connectivity
Hierarchical Processing



Weight Sharing

Using the same to process different regions of the image

Training

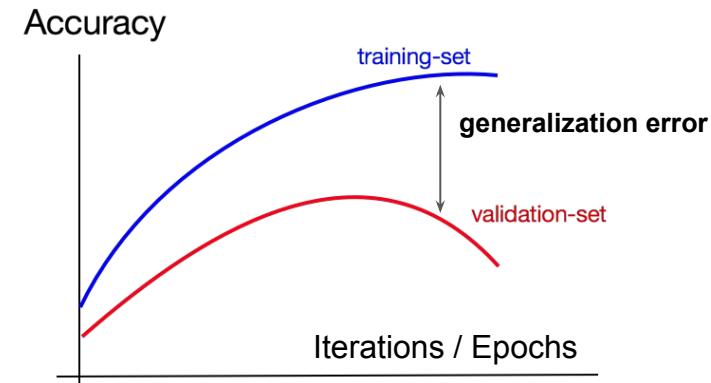
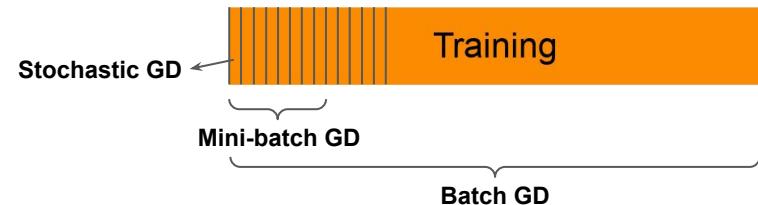
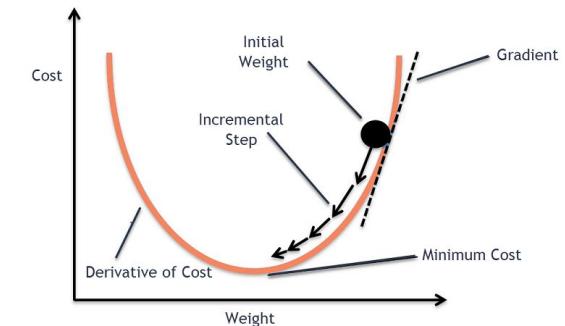
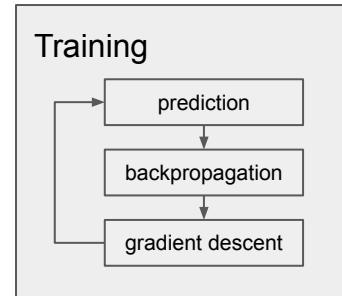
$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

stochastic, mini-batch or batch GD ?

Epoch: one pass through the dataset

How many iterations/epochs ?

How to find the learning rate ? η



Training

stochastic, mini-batch or batch GD

How many iterations ? what's the step size ? how to evaluate ?
(overfitting and underfitting)

