Data Science: Principles and Practice

Lecture 4: Deep Learning, Part I

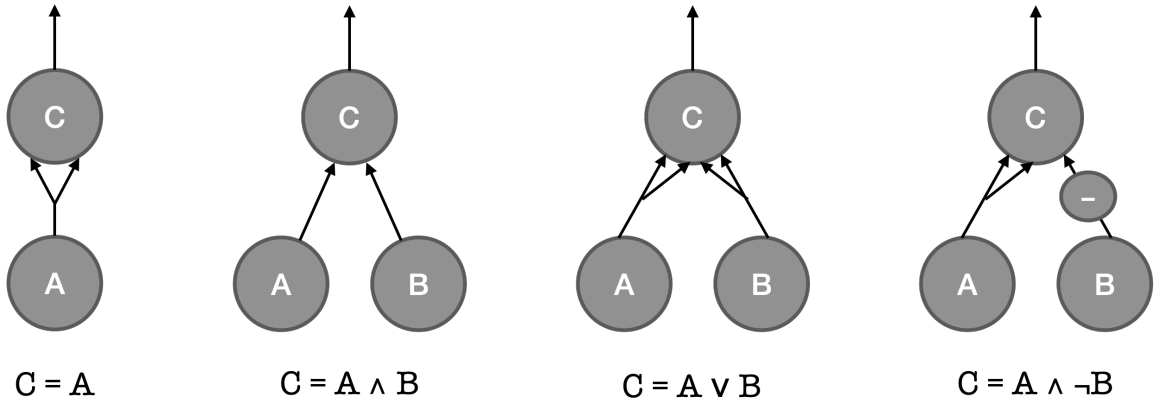
Ekaterina Kochmar[1]

[1] Based on slides by Marek Rei

Fundamentals of Neural Networks

Simple artificial neuron

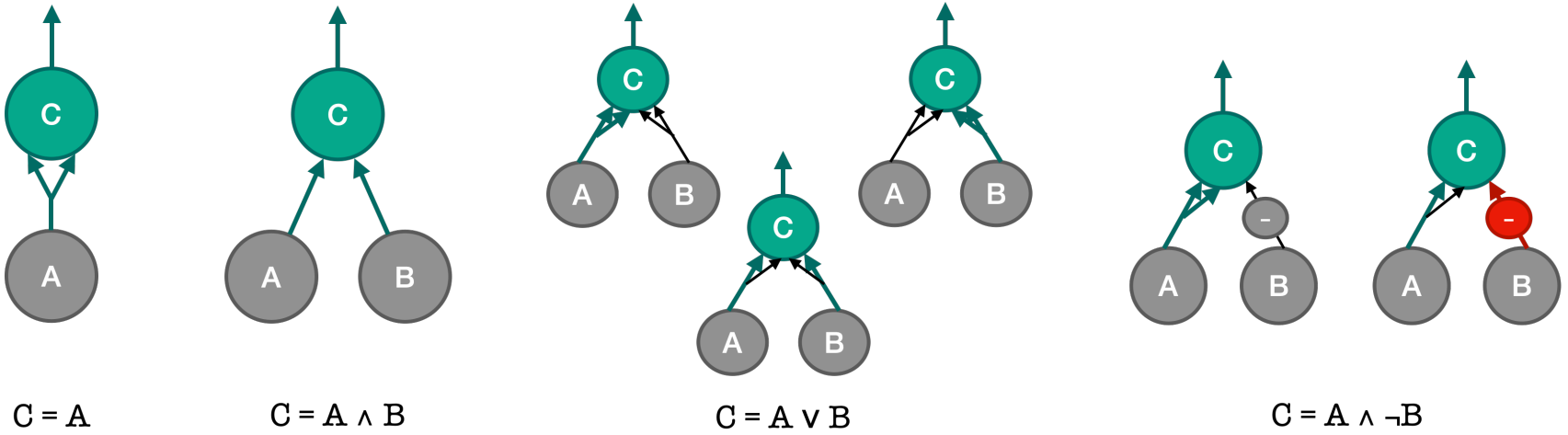
- Threshold Logic Unit, or Linear Threshold Unit, by McCulloch and Pitts[1]
- Has one or more binary (on/off) inputs and one binary output
- Activates its output when more than a certain number of its inputs are active
- Even with such a simplified model it is possible to build a network of artificial neurons that computes any logical proposition



McCulloch and Pitts (1943). "*A logical calculus of the ideas immanent in nervous activity*"

Simple artificial neuron

Assume that a neuron is activated when at least two of its inputs are active:



Geron (2017). "*Hands-On Machine Learning with Scikit-Learn & TensorFlow*"

Simple artificial neuron

Quiz time:

Can you draw an ANN that computes $A \oplus B$, where $\oplus$ is the XOR operation,

i.e. $A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$?

Recap: Perceptron

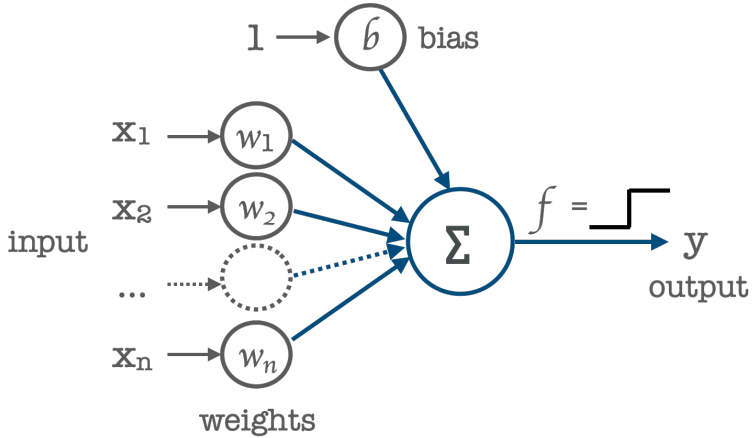
$$\hat{y}^{(i)} = \begin{cases} 1, & \text{if } w \cdot x^{(i)} + b > 0 \\ 0, & \text{otherwise} \end{cases}$$

where:

- $w \cdot x^{(i)}$ is the dot product of the weight vector w and the feature vector $x^{(i)}$ for instance i, i.e.

$$\sum_{j=1}^{n} w_j x_j^{(i)}$$

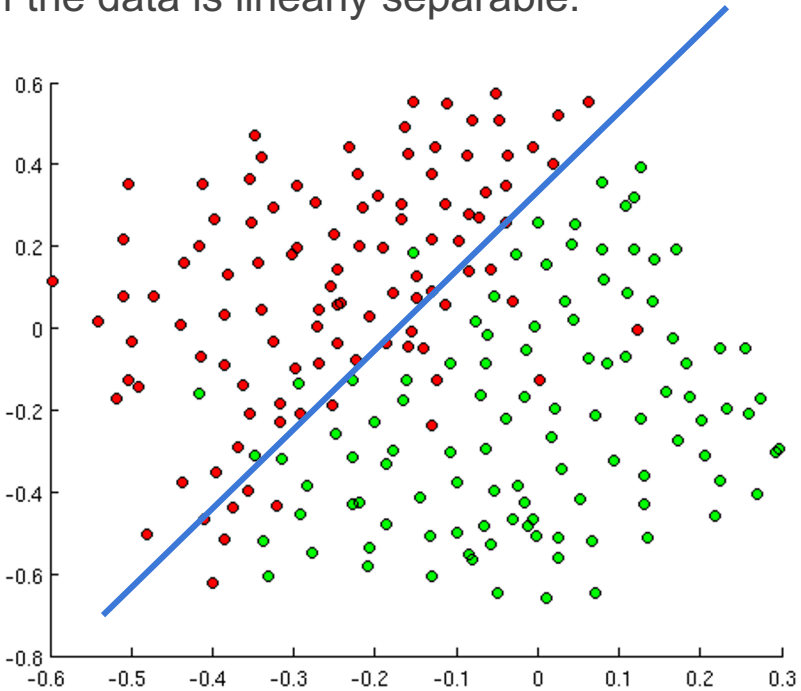
- b is the bias term
- f is a *Heaviside step function*



This is a **Linear Threshold Unit**

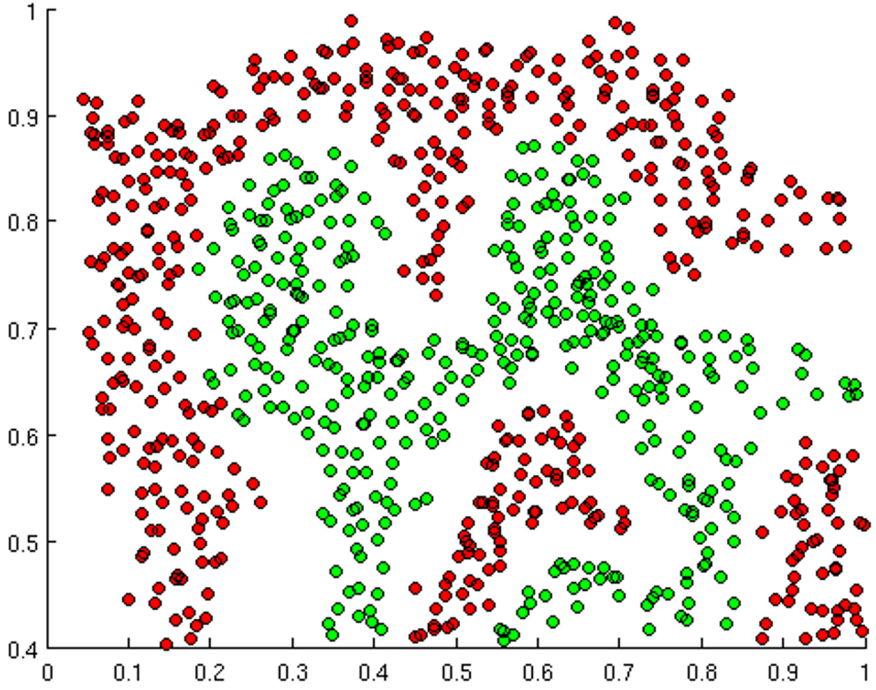
Linear separability of classes

Linear models are great if the data is linearly separable.



Linear separability of classes

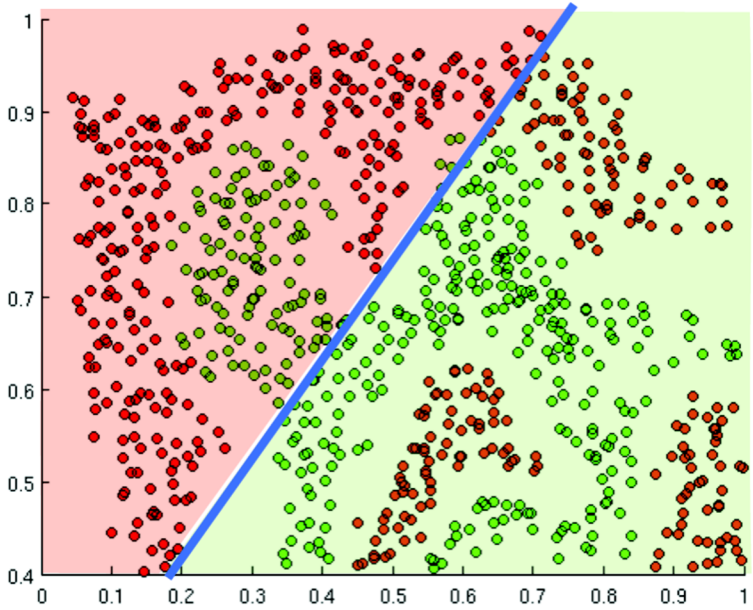
… but often that is not the case.

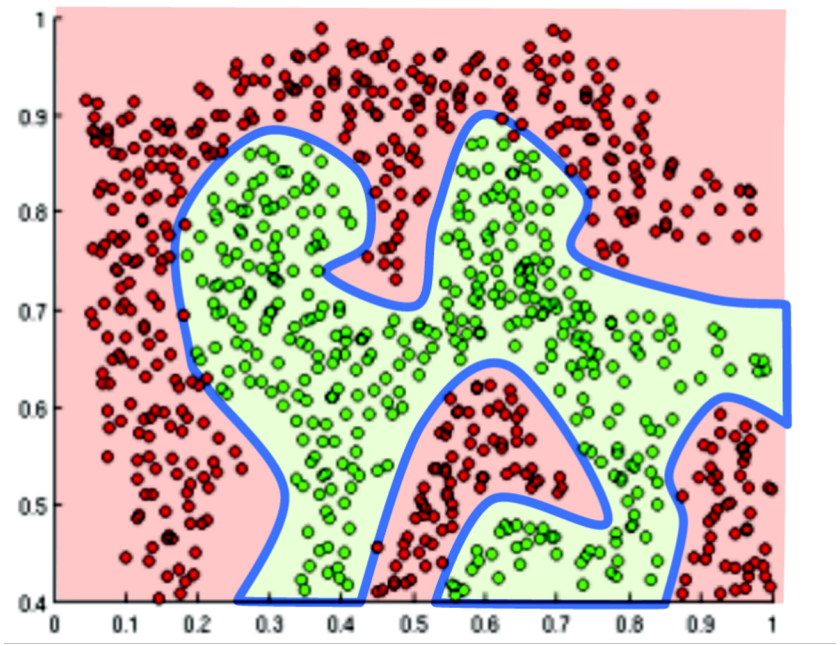


http://introtodeeplearning.com/

Linear separability of classes

Linear models are not able to capture complex patterns in the data.



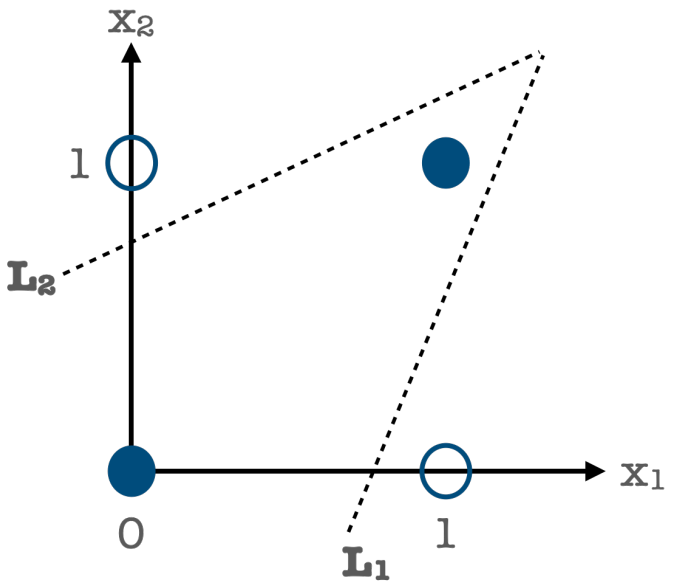


Recap: Non-linearly separable data

Consider the following classic example of the XOR problem $y = x_1 \oplus x_2$

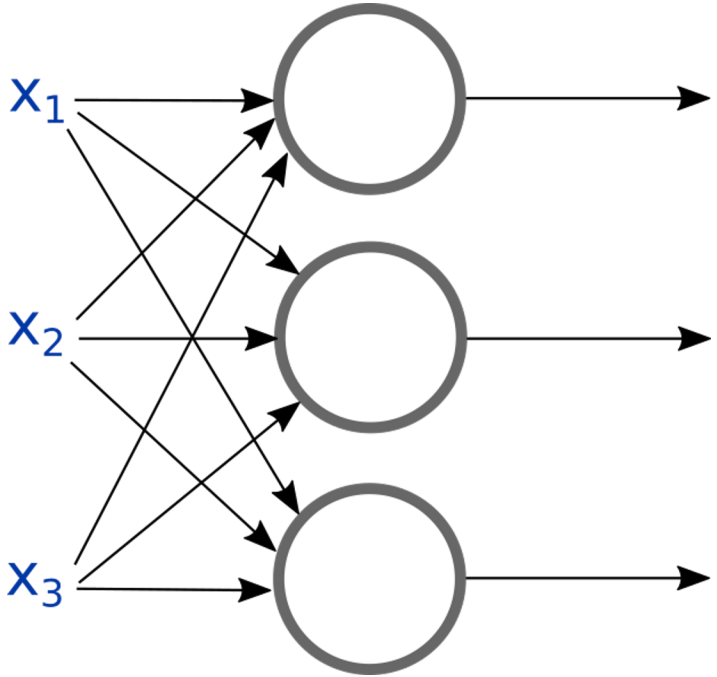
x_1	x_2	y
0	0	0
0	1	**1**
1	0	**1**
1	1	0

$y = x_1 \oplus x_2$



Connecting the neurons

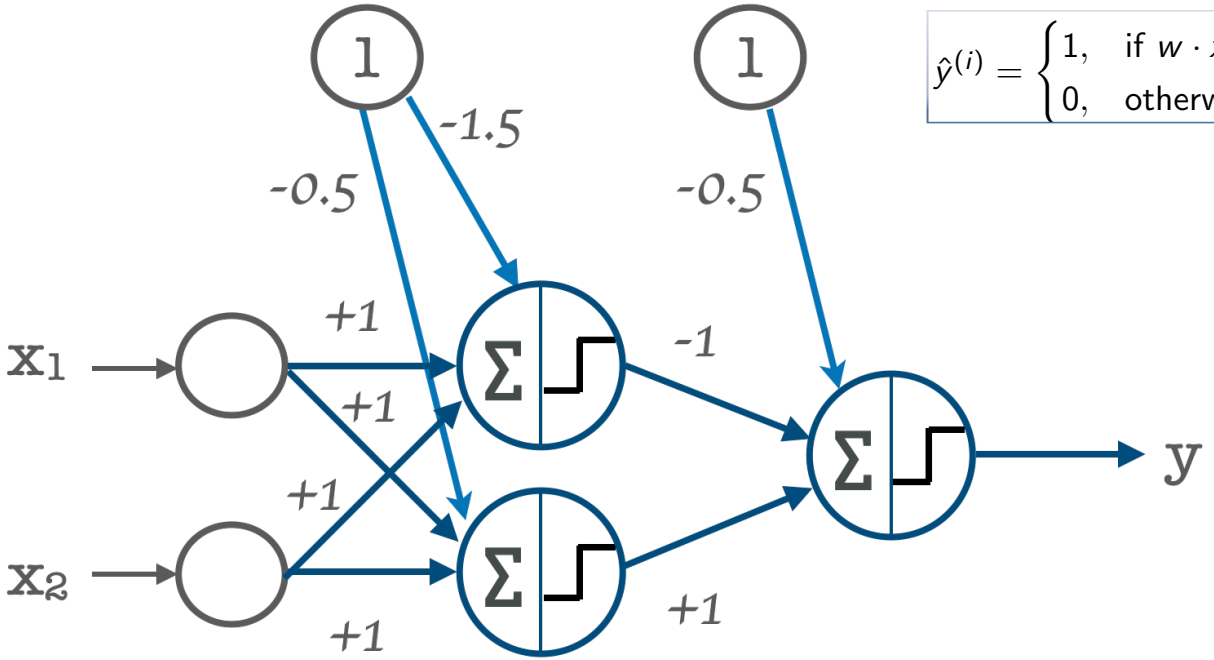
We can connect multiple neurons in parallel – each one will learn to detect something different.



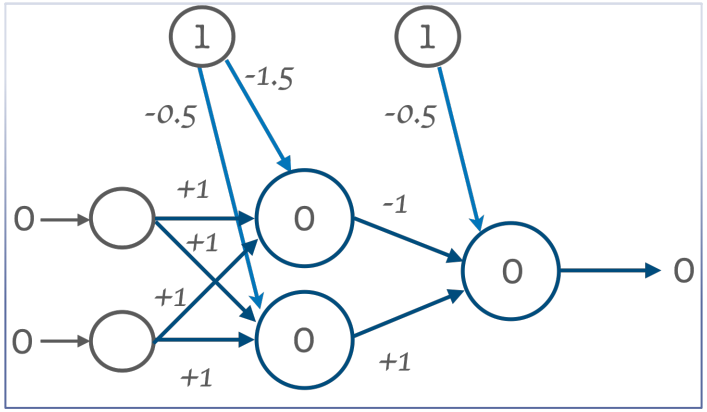
Connecting the neurons

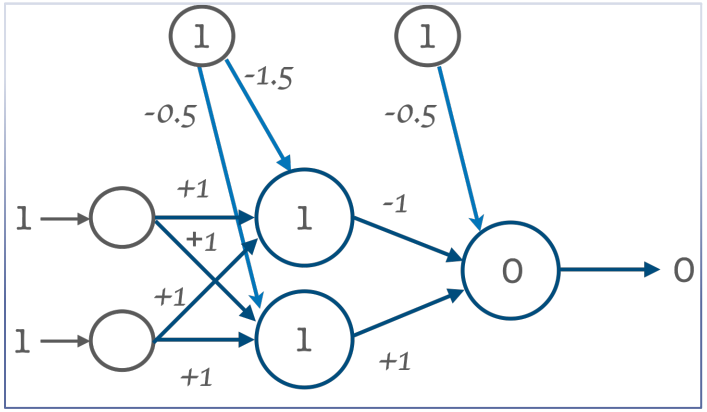
Each node will learn to detect something different: e.g., one hidden node – whether at least one input is 1, and another – whether both are 1

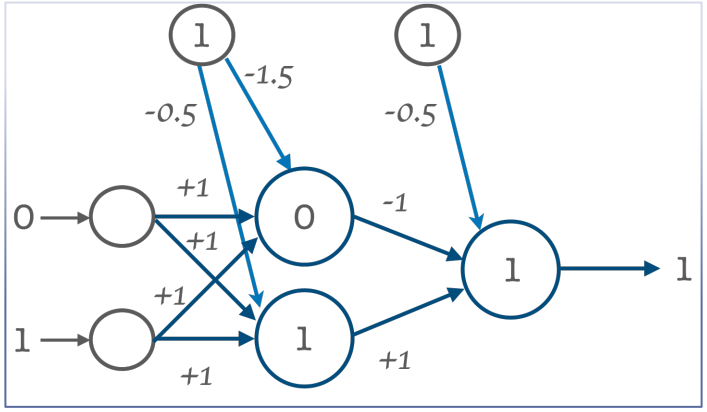
$$\hat{y}^{(i)} = \begin{cases} 1, & \text{if } w \cdot x^{(i)} + b > 0 \\ 0, & \text{otherwise} \end{cases}$$

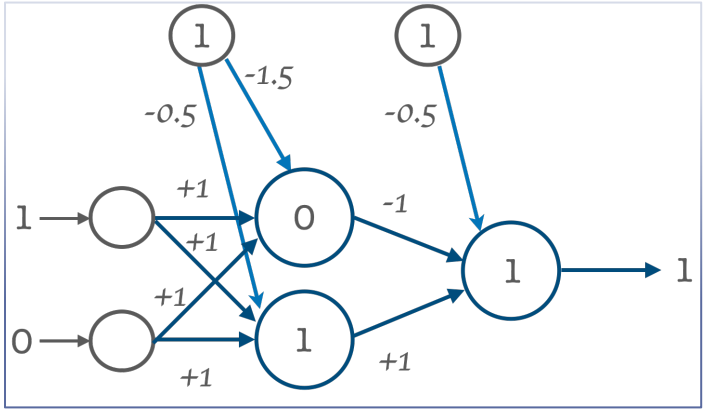


Multilayer Perceptron (MLP) on XOR problem





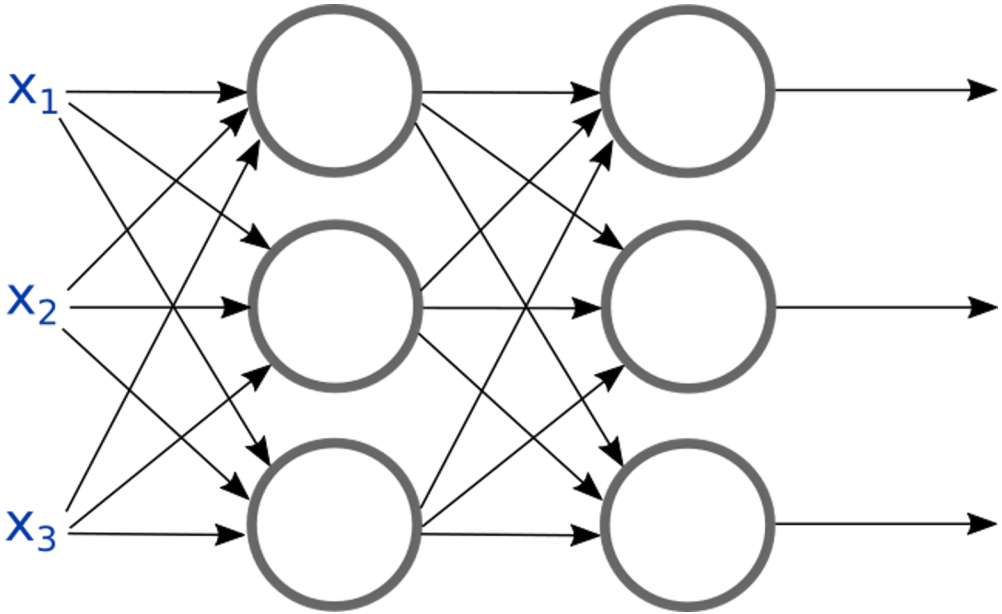




Multilayer Perceptron

Not actually a perceptron

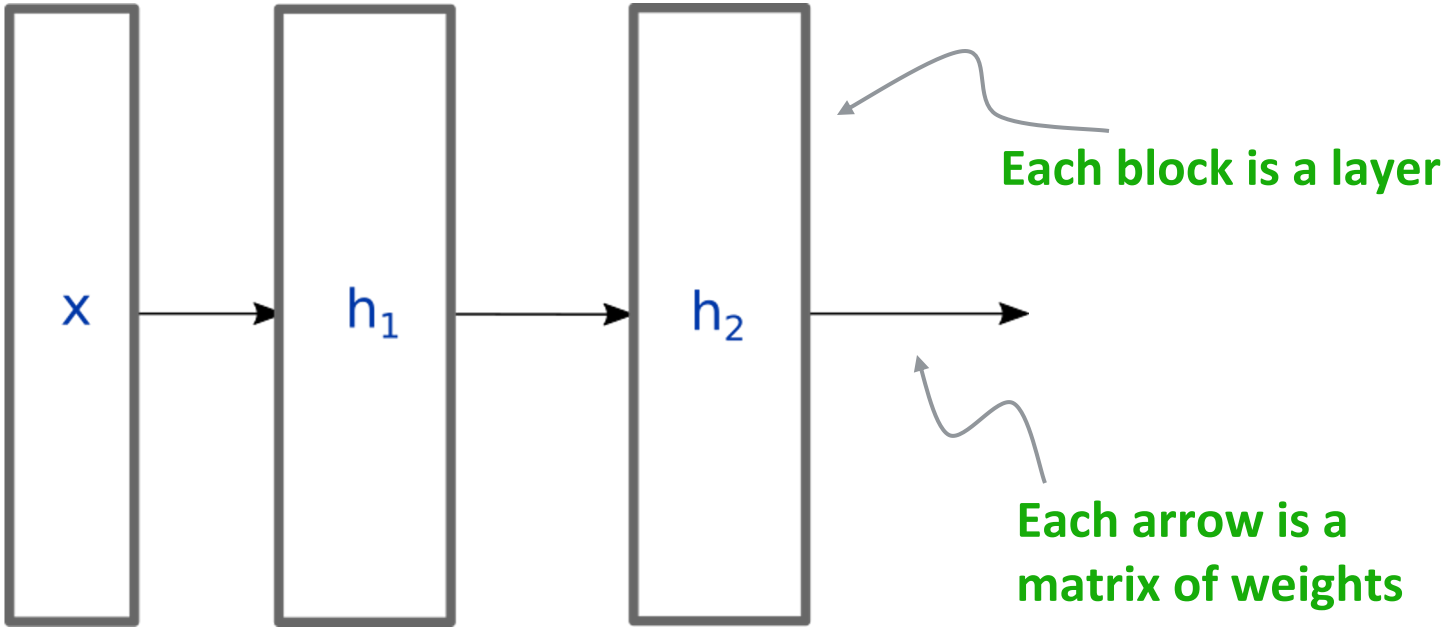
We can connect neurons in sequence in order to learn from higher-order features.



An MLP with sufficient number of neurons can theoretically model an arbitrary function over an input.

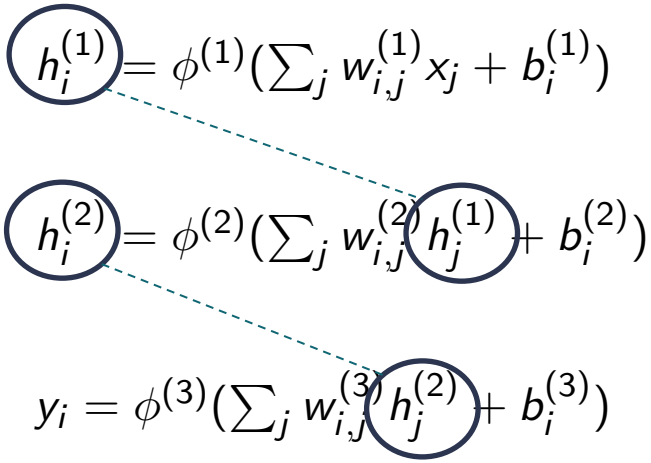
Multilayer Perceptron

We can connect neurons in sequence in order to learn from higher-order features.



An MLP with sufficient number of neurons can theoretically model an arbitrary function over an input.

Mathematical Definition



Fully connected network

- Each unit in the first hidden layer $h_i^{(1)}$ receives activations from each input unit x_j multiplied with the weight relevant for this pair of units $w_{i,j}^{(1)}$ plus unit's own bias $b_i^{(1)}$
- Second layer $h_i^{(2)}$ units receive activations from the first layer units $h_j^{(1)}$
- Different activation functions $\phi^{(1)}\ldots\ \phi^{(3)}$ may be used at each level

Non-linear Activation Functions

- **Logistic function**:

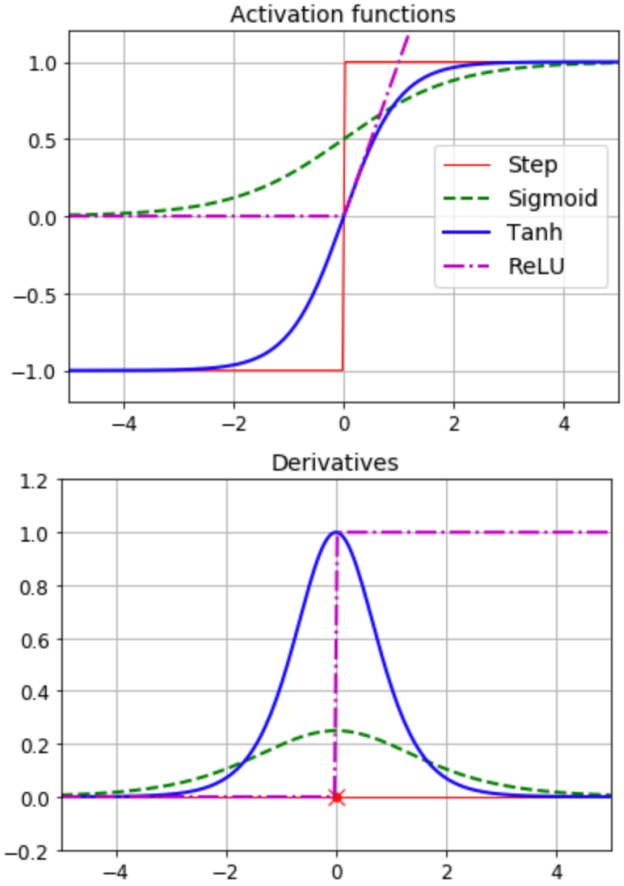
$$\sigma(z) = \frac{1}{1+exp(-z)}$$

- **Hyperbolic tangent function**:

$$\tanh(z) = 2\sigma(2z) - 1$$

- **Rectified linear unit (ReLU) function**:

$$ReLU(z) = max(0, z)$$

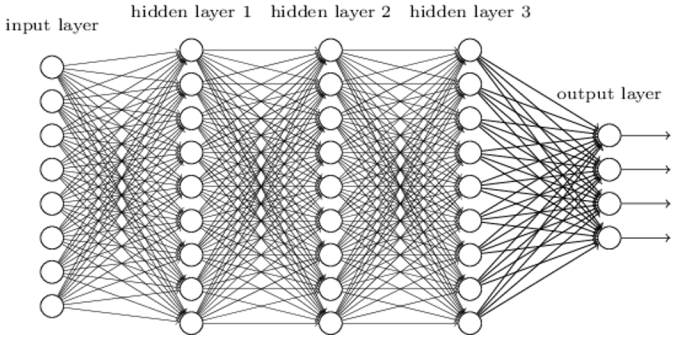


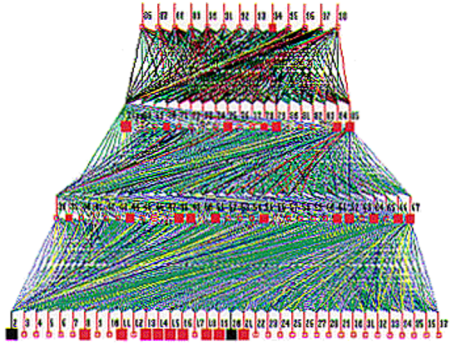
Neural Network Hyperparameters

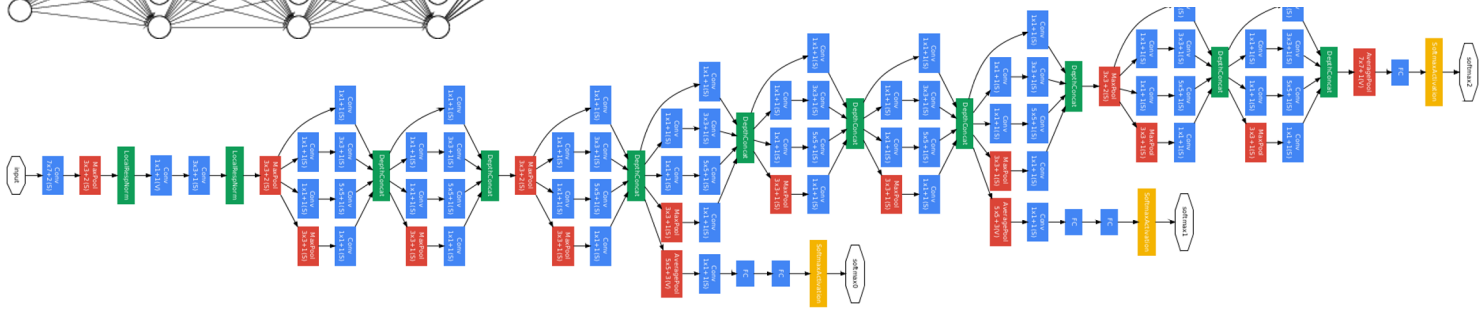
- **Depth** – number of hidden layers
- **Width** – number of units per hidden layer
- **Activation functions**

Deep Neural Networks

In practice we train neural networks with thousands of neurons and millions (or billions) of trainable weights.

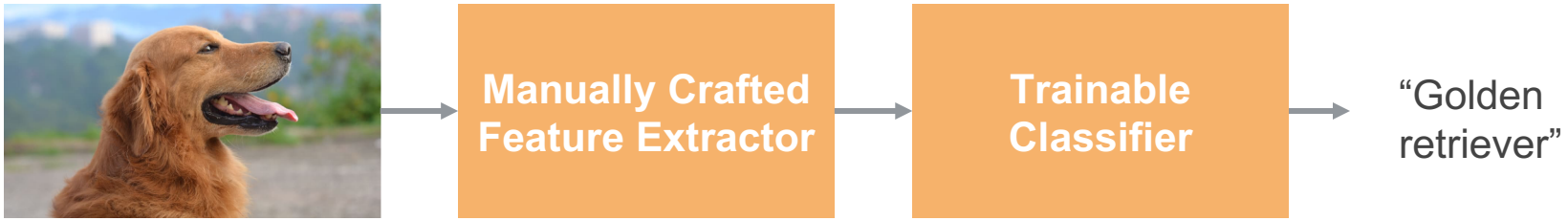




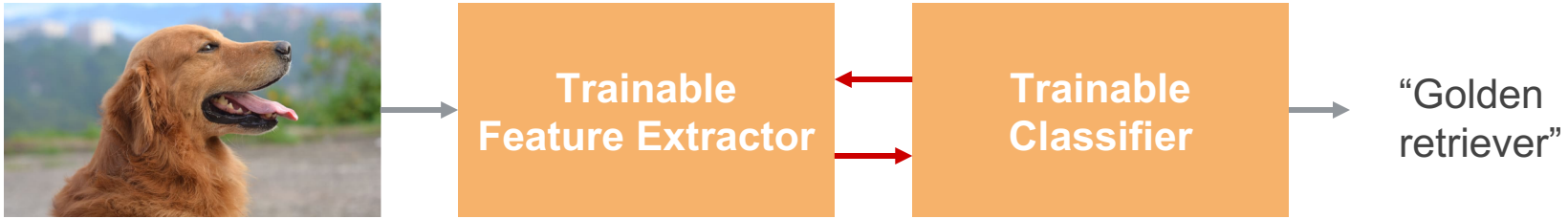


Learning Representations & Features

Traditional pattern recognition

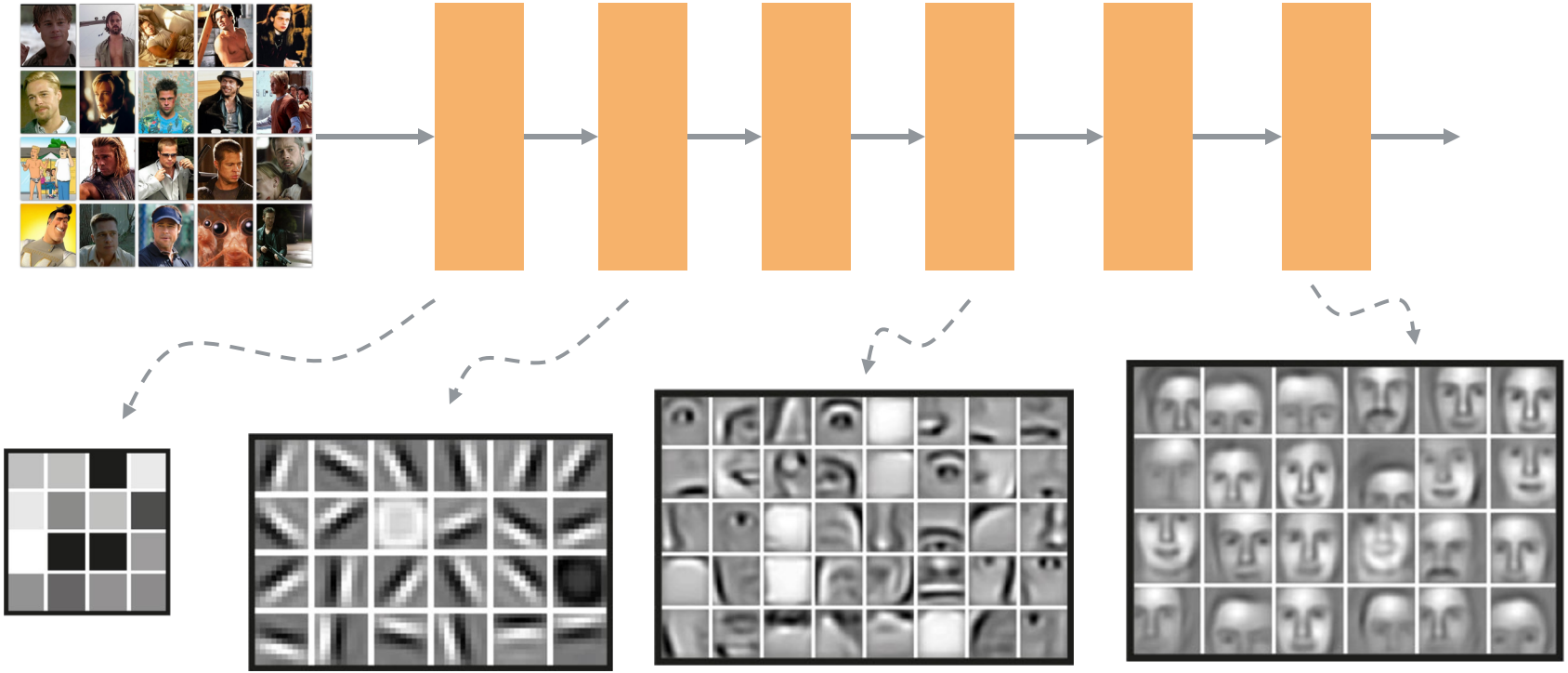


End-to-end training: Learn useful features also from the data



Learning Representations & Features

Automatically learning increasingly more complex feature detectors from the data.



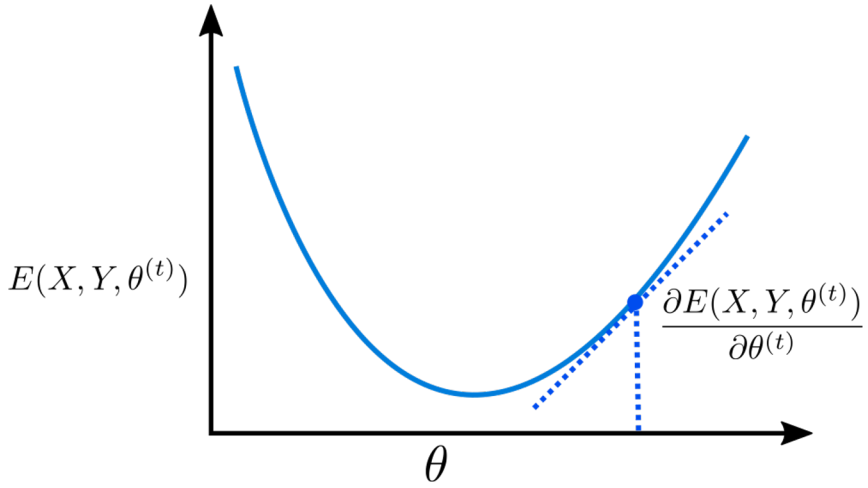
Neural Network Optimization

Optimizing Neural Networks

Define a **loss function** that we want to minimize

Update the parameters using **gradient descent**, taking small steps in the direction of the gradient (going downhill on the slope).

All the operations in the network need to be **differentiable**.

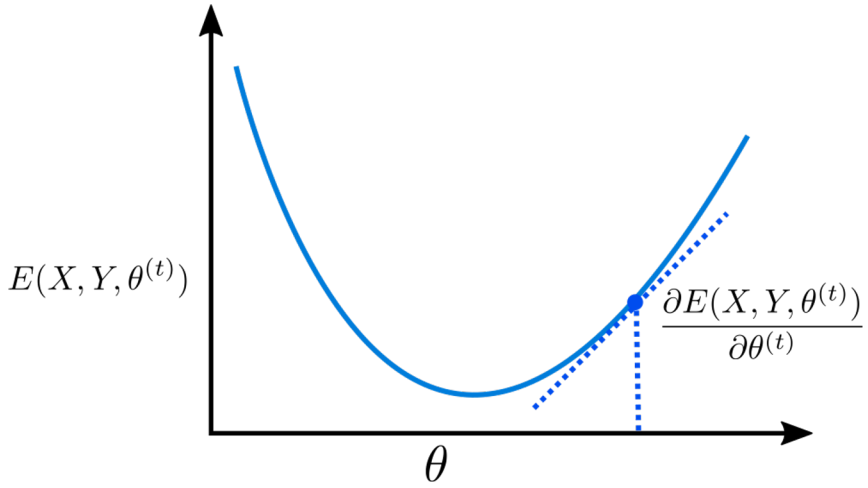


$$\theta_i^{(t+1)} = \theta_i^{(t)} - \alpha \frac{\partial E}{\partial \theta_i^{(t)}}$$

Gradient Descent

Algorithm

1. Initialize weights randomly
2. Loop until convergence:
3. Compute gradient based on the whole dataset
4. Update weights
5. Return weights

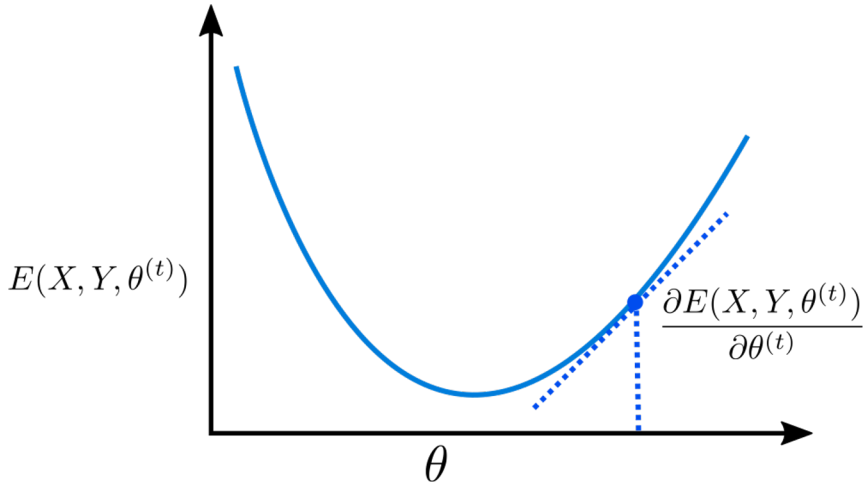


In practice, datasets are often too big for this

Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly
2. Loop until convergence:
3. Loop over **each datapoint**:
4. Compute gradient based on the datapoint
5. Update weights
6. Return weights

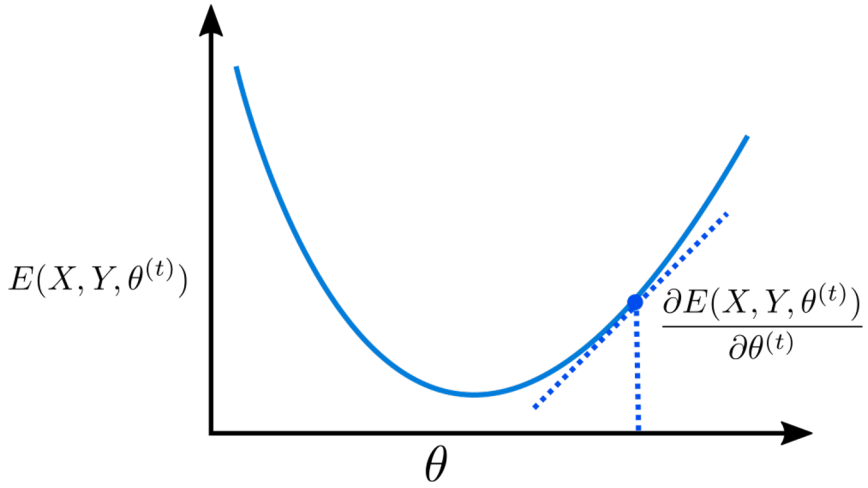


Very noisy to take steps based only on a single datapoint

Mini-batch Gradient Descent

Algorithm

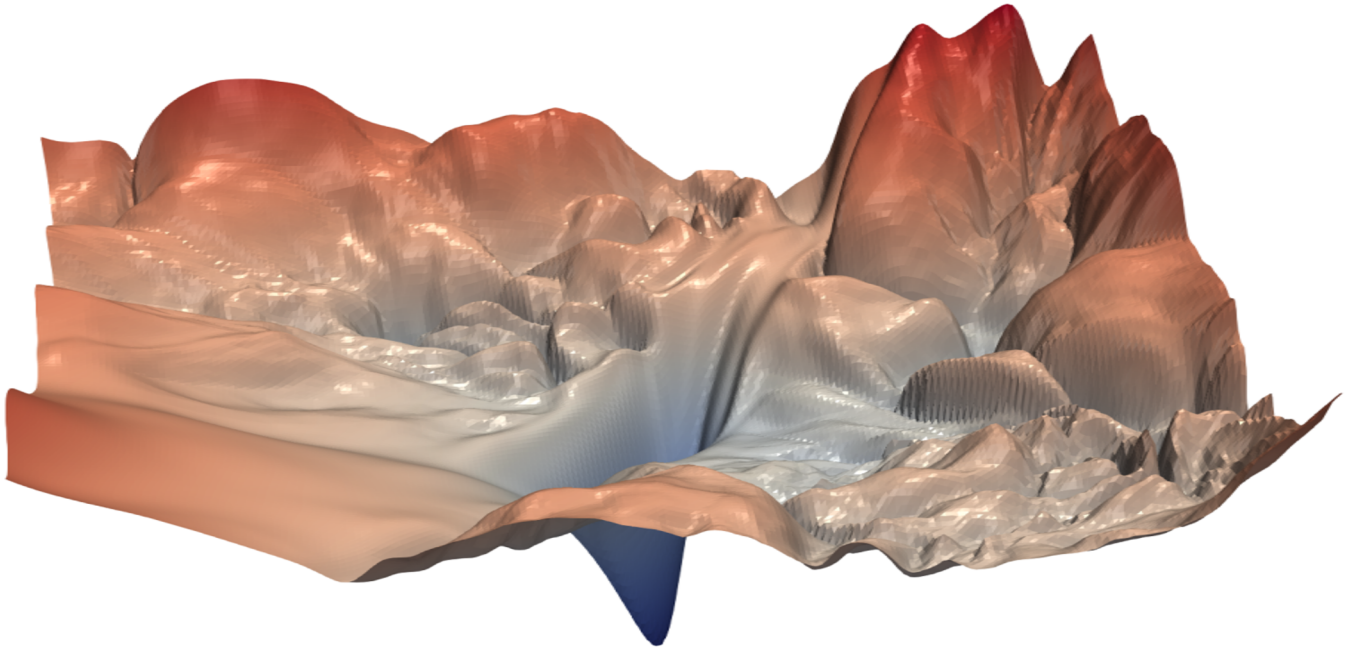
1. Initialize weights randomly
2. Loop until convergence:
3. Loop over **batches of datapoints**:
4. Compute gradient based on the batch
5. Update weights
6. Return weights



This is what we mostly use in practice

Optimizing Neural Networks

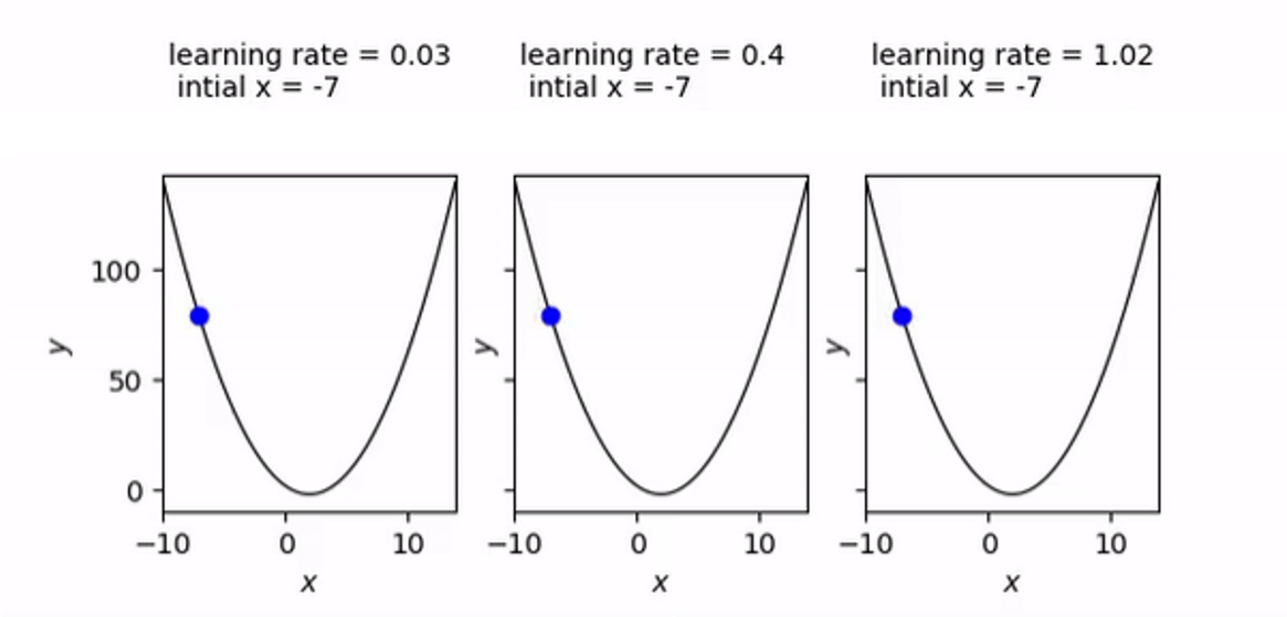
Neural networks have very complex loss surfaces and finding the optimum is difficult.



Li et al., 2018. “Visualizing the Loss Landscape of Neural Nets”

The Importance of the Learning Rate

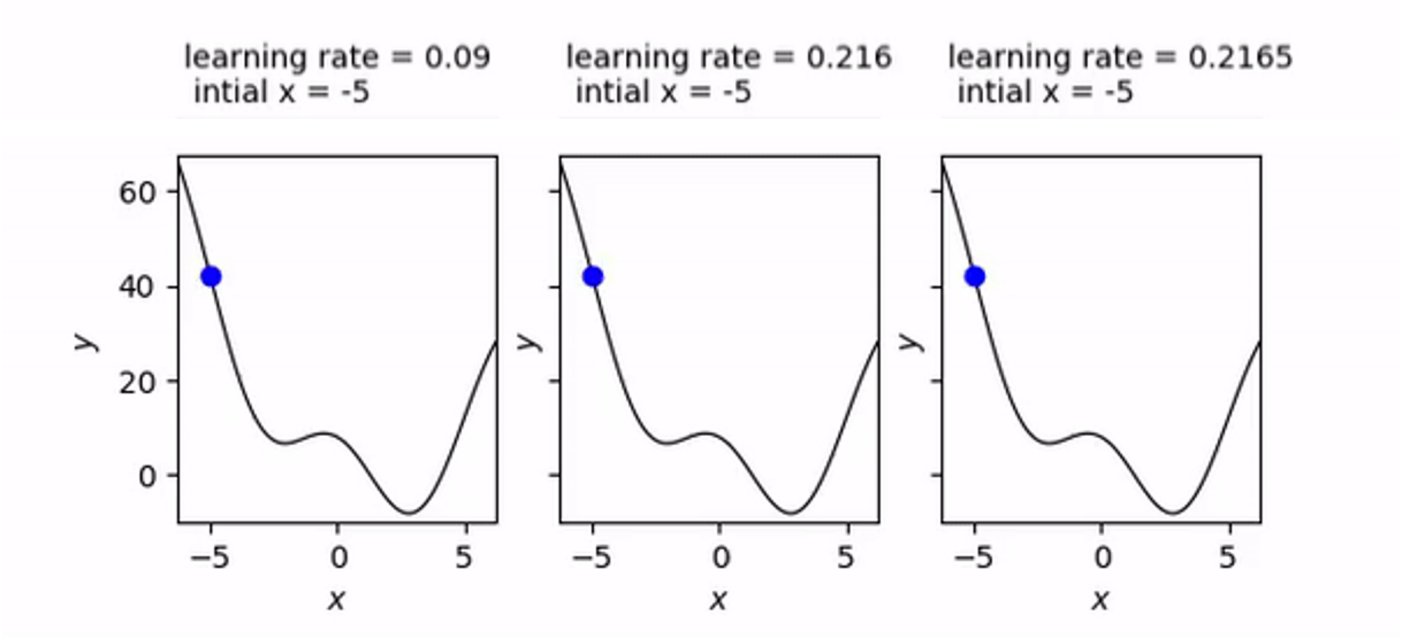
If the learning rate is too low, the model will take forever to converge.
If the learning rate is too high, we will just keep stepping over the optimum values.



https://jed-ai.github.io/opt2_gradient_descent_1/

The Importance of the Learning Rate

A small learning rate can get the model stuck in local minima.
A bigger learning rate can help the model converge better (if it doesn't overshoot).



https://jed-ai.github.io/opt2_gradient_descent_1/

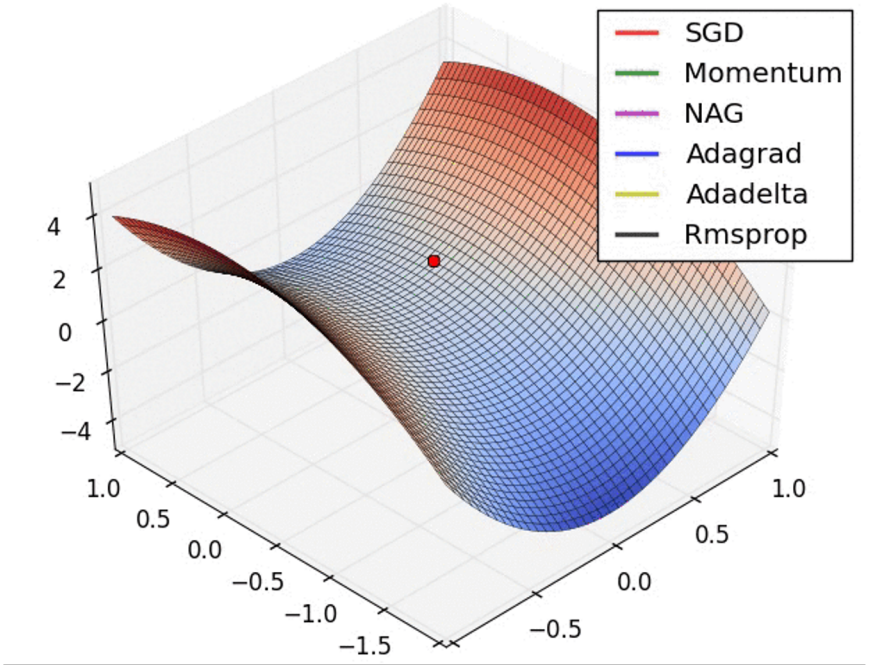
Adaptive Learning Rates

Intuition:

Have a different learning rate for each parameter.

Take bigger steps if a parameter has not been updated much recently.

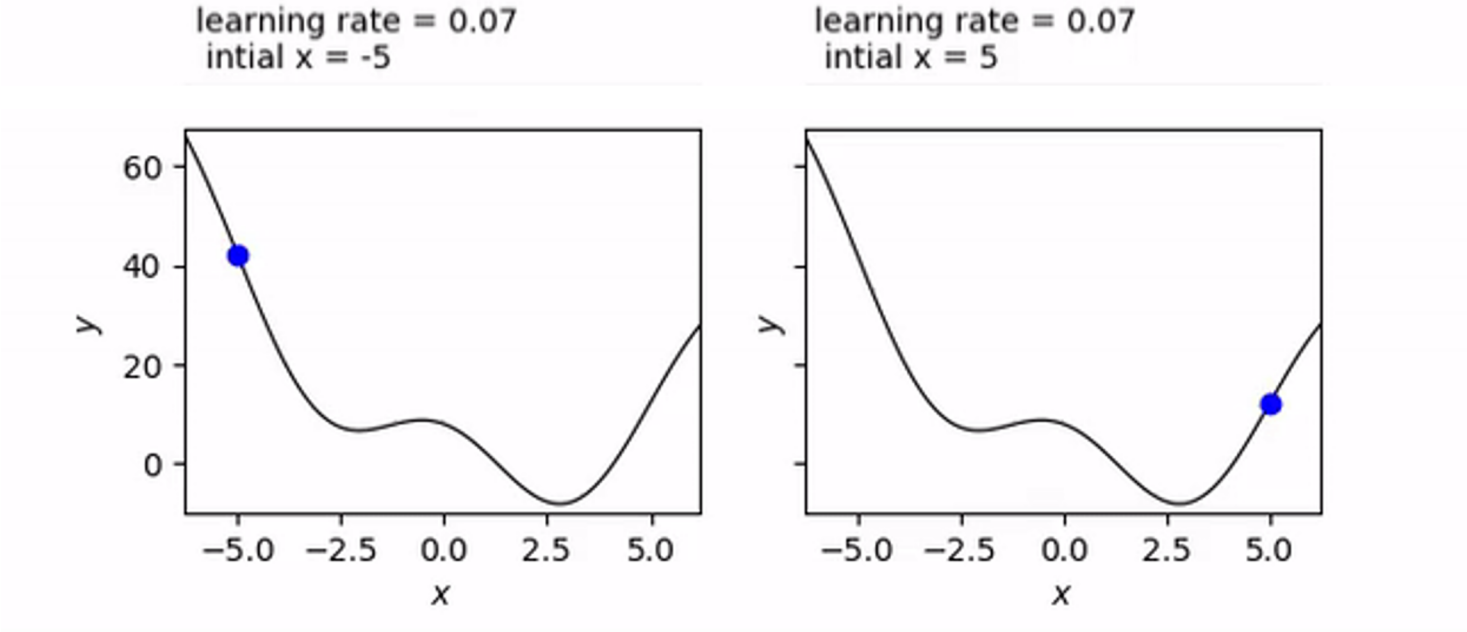
Take smaller steps if a parameter has been getting many big updates.



Alex Radford

Random initialization Matters

All other things being equal, just starting from a different location can lead to a different result.



https://jed-ai.github.io/opt2_gradient_descent_1/