Deep learning for medical imaging

Olivier Colliot, PhD
Research Director at CNRS
Co-Head of the ARAMIS Lab – www.aramislab.fr
PRAIRIE – Paris Artificial Intelligence Research Institute

Maria Vakalopoulou, PhD
Assistant Professor at CentraleSupelec
Center for Visual Computing

Master 2 - MVA

Course website: http://www.aramislab.fr/teaching/DLMI-2019-2020/
Piazza (for registered students): https://piazza.com/centralesupelec/spring2020/mvadlmi/

Part 2 – Classification (and regression)

Introduction

Outputs y are classes: **classification** problem

Labelled training examples

Learning

Input x

f

$y = f(x)$

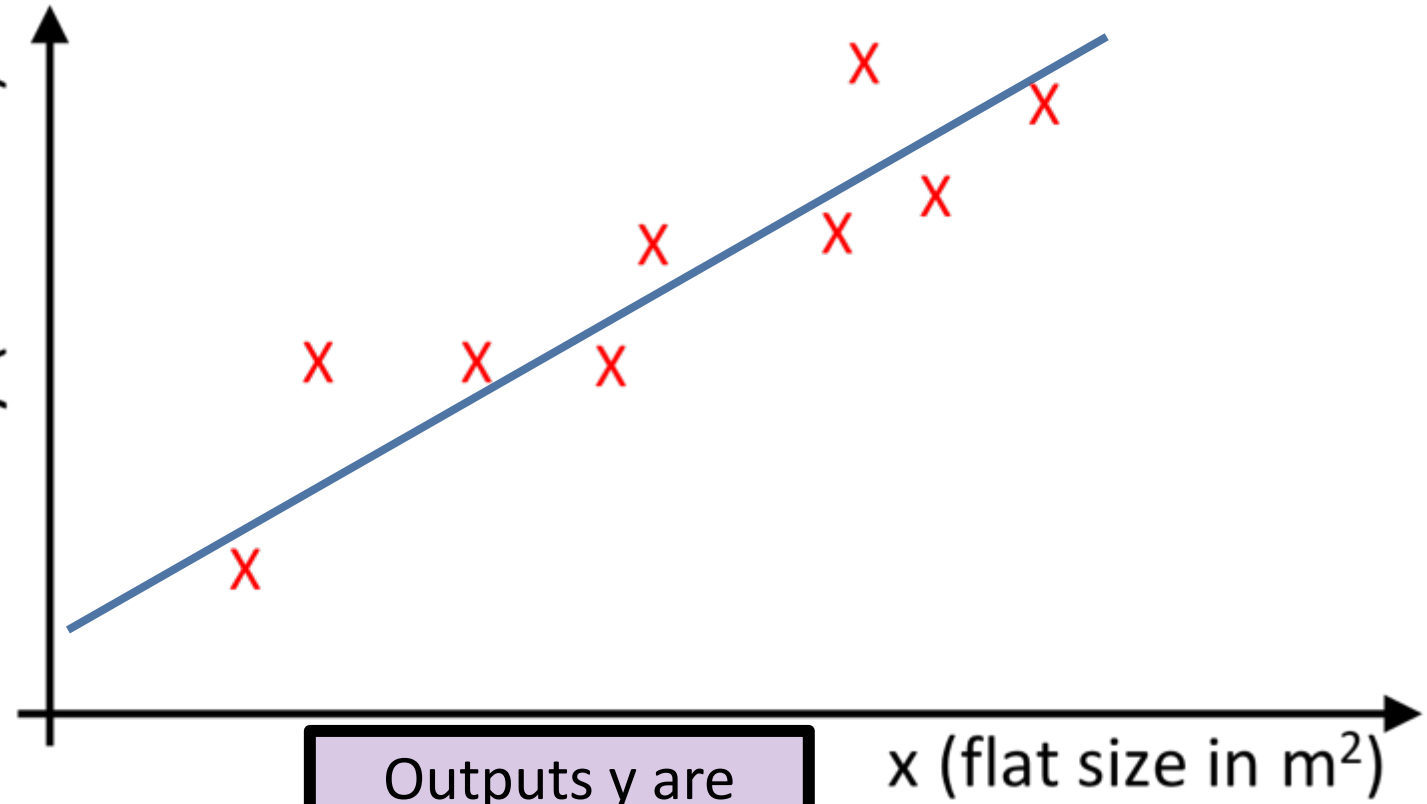
Beaver

Output y

$$\hat{f} = arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, f(x^{(i)})\right)$$

Loss function: measures error in prediction

Introduction



Outputs y are numbers: **regression** problem

Part 2 – Classification (and regression)
2.1 Linear and logistic regression in 10 minutes

So, linear regression is AI?

Sally Hudson
@SallyLHudson

Suivre

"When you're fundraising, it's AI.
When you're hiring, it's ML.
When you're implementing, it's linear regression.

Multivariate linear regression

Training set

features

i	$x_1^{(i)}$ Size of flat (in m^2)	$x_2^{(i)}$ Number of rooms	$x_3^{(i)}$ Floor	y_i Cost (in €)
1	100	5	1	940,000
2	53	3	5	510,000
3	25	1	6	280,000
…				
n				

Multivariate linear regression

Training set

features

i	$x_1^{(i)}$ Size of flat (in m^2)	$x_2^{(i)}$ Number of rooms	$x_3^{(i)}$ Floor	y_i Cost (in €)
1	100	5	1	940,000
2	53	3	5	510,000
3	25	1	6	280,000
…				
n				

Input $\mathbf{x}$ is a vector $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$, indexed by j

Model: $y = w_0 + \sum_{j=1}^{p} w_j x_j$

Reduced notation: $y = w_0 + \boldsymbol{w}^T \mathbf{x}$

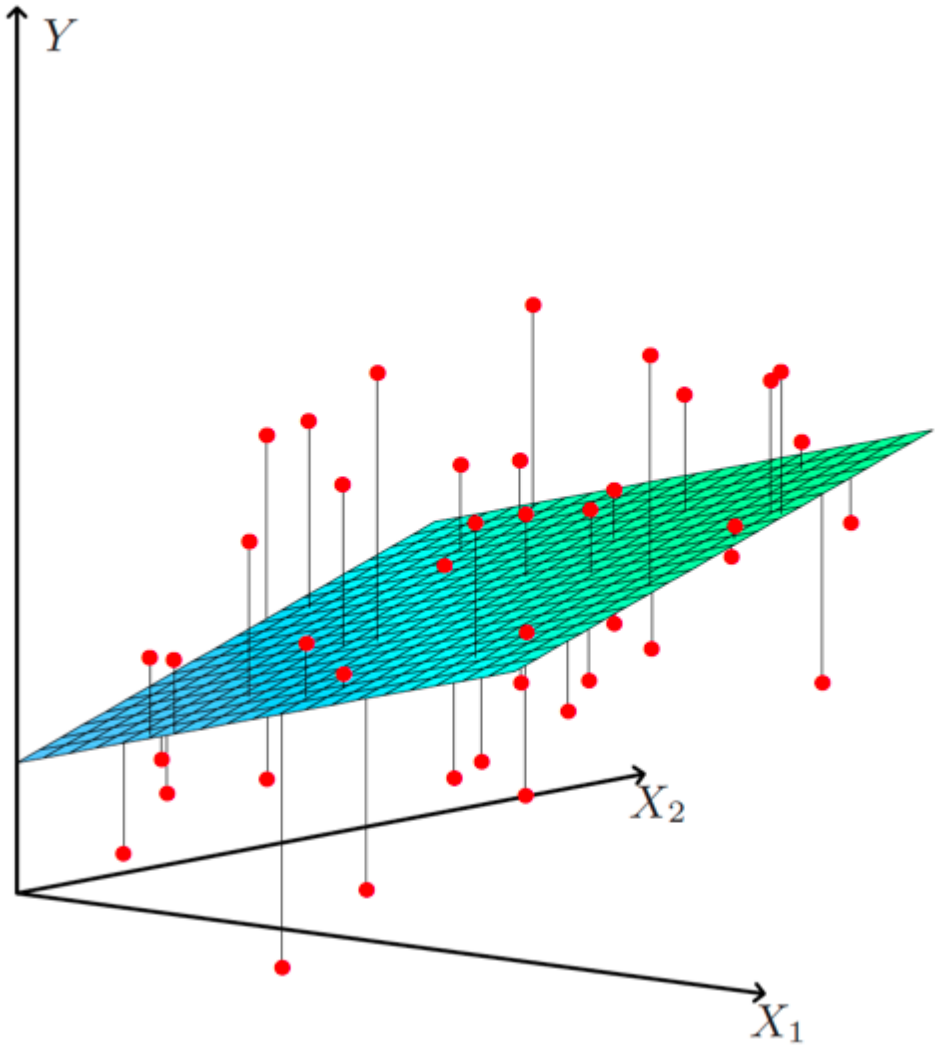
Often, to simplify the notations, we will integrate w_0 into $\boldsymbol{w}$.

Input $\mathbf{x}$ is a vector $\boldsymbol{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}$

Model: $y = \sum_{j=0}^{p} w_j x_j$

Reduced notation: $y = \boldsymbol{w}^T \mathbf{x}$

Multivariate linear regression



Model: $y = \sum_{j=0}^{p} w_j x_j$

Loss: $\ell(y, f(x)) = (y - f(x))^2$

Multivariate linear regression

Input $\mathbf{x}$ is a vector $\boldsymbol{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}$

Model: $y = \sum_{j=0}^{p} w_j x_j$

Reduced notation: $y = \boldsymbol{w}^T \mathbf{x}$

Training samples (inputs): $\mathbf{x^{(1)}}, \ldots, \mathbf{x^{(n)}}$

vectors $\mathbf{x^{(i)}} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_p^{(i)} \end{bmatrix}$

In matrix notation: $\mathbf{X} = \begin{bmatrix} 1 & \ldots & 1 \\ x_1^{(1)} & \ldots & x_p^{(1)} \\ \vdots & & \vdots \\ x_1^{(n)} & \ldots & x_p^{(n)} \end{bmatrix}$

Training samples (outputs): $\mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$

Cost function:

$$J(f) = \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - w_0 - \sum_{j=1}^{p} w_i x_i^{(j)} \right)$$

$$J(f) = \frac{1}{n} (\mathbf{y} - \mathbf{X}\boldsymbol{w})^T (\mathbf{y} - \mathbf{X}\boldsymbol{w})$$

Multivariate linear regression

we have an exact solution:

$$\hat{\boldsymbol{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

But the matrix $\mathbf{X}^T\mathbf{X}$ needs to be invertible, which is not the case if:

- There are redundant input variables (e.g. size in meters and size in feet)
- There are more input variables than training samples ($p > n$)

Multivariate linear regression

- **Summary**
  - **Multivariate** linear regression is a technique for predicting a continuous output Y from **multiple** inputs
  - It uses a linear model
  - The cost function is convex
  - It has a single global minimum
  - The solution can be found in an exact manner

Classification

The output y is a class.

Suppose we have K classes, then we could write $y \in \{C_1, \ldots, C_K\}$.

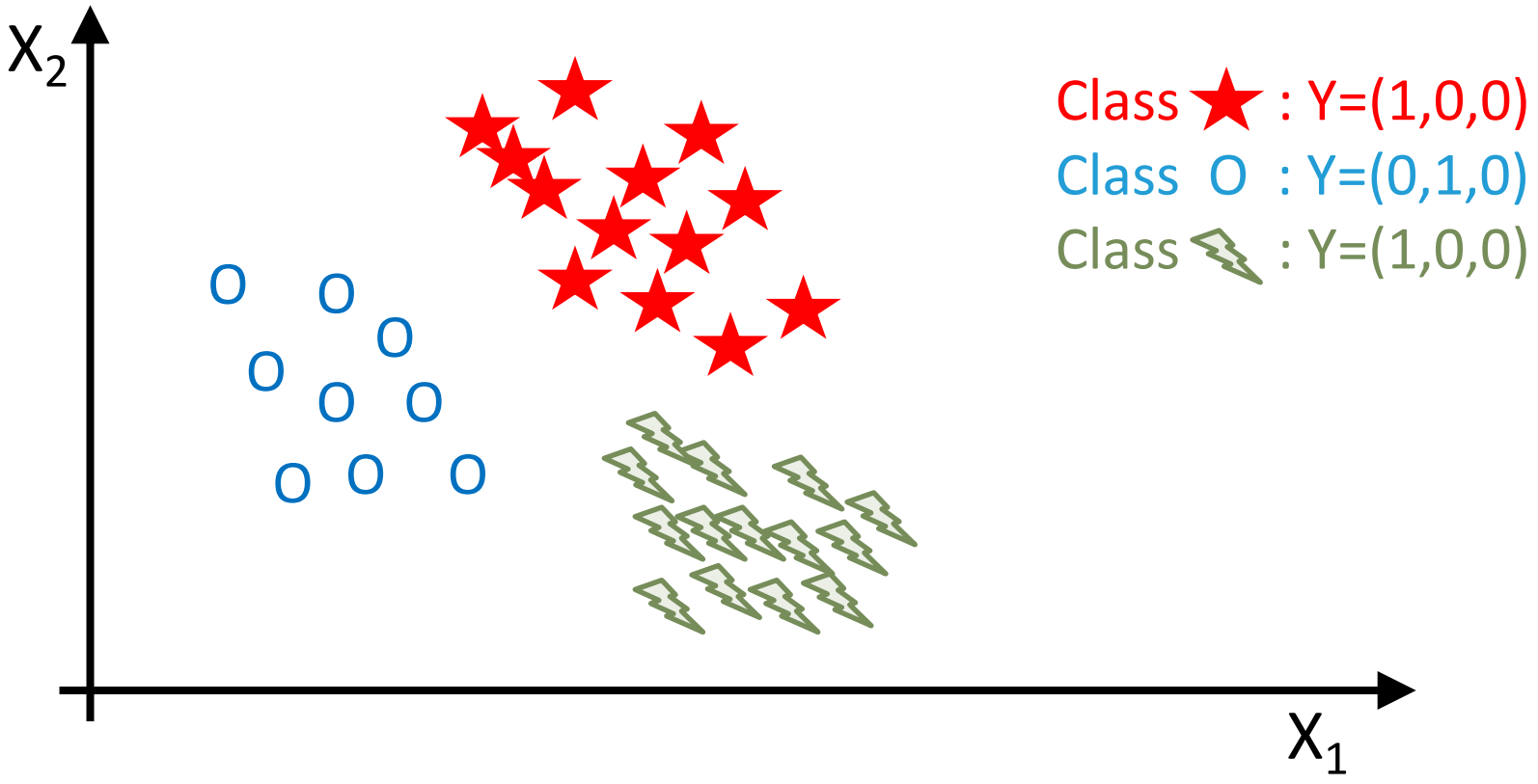
We need to encode this numerically.

Coding $y \in \{1, \ldots, K\}$ would not be meaningful because there is no order between the different classes.

Therefore, we will use indicator variables. The indicator y_k for class C_k is such that $y_k = 1$ if Y belongs to class C_k and 0 otherwise.

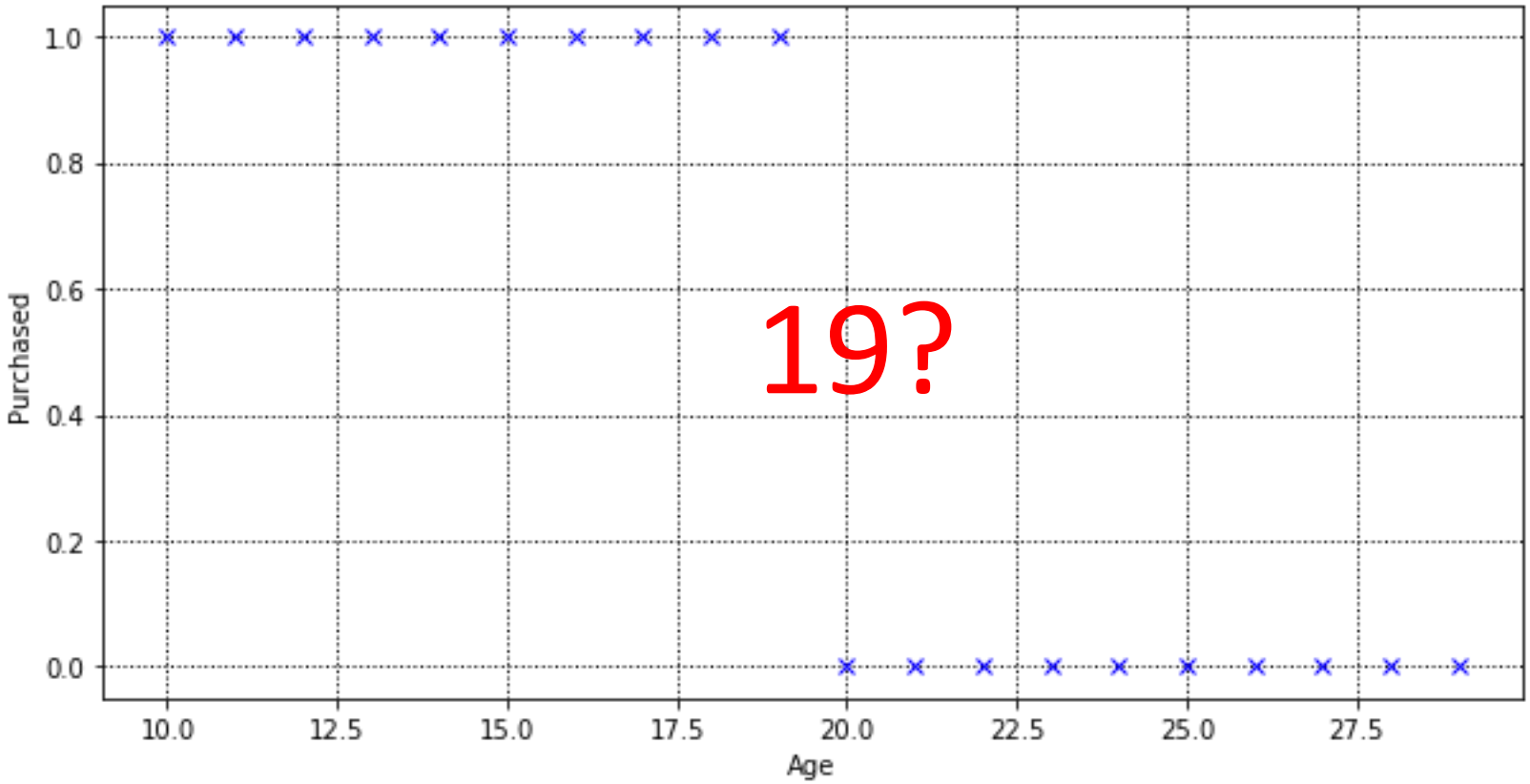
Thus, the ouput is the vector $\mathbf{y} = (y_1, \ldots, y_K)$.

Example: three classes



Classification

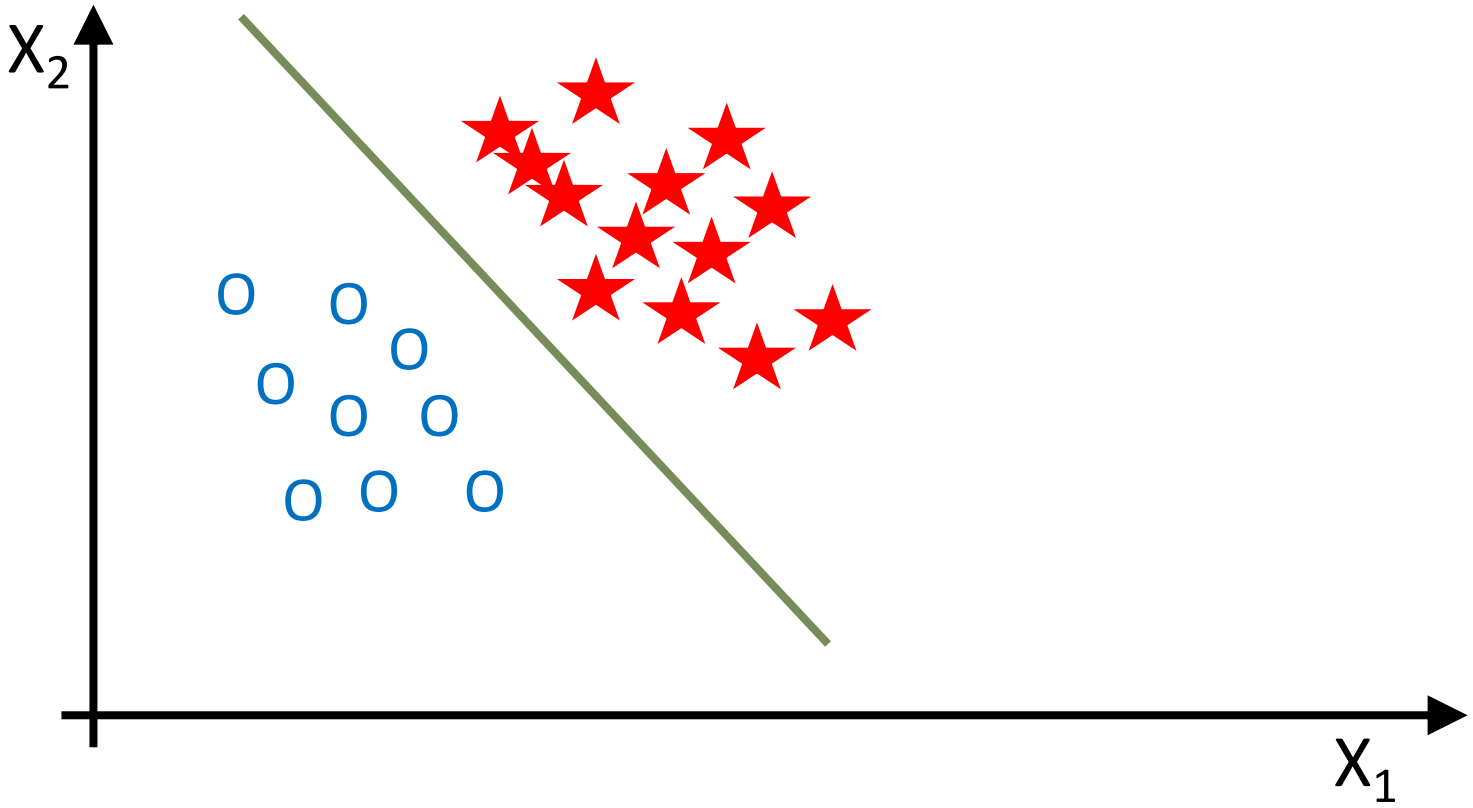
What we want is a decision boundary



With 1 variable, it is a number

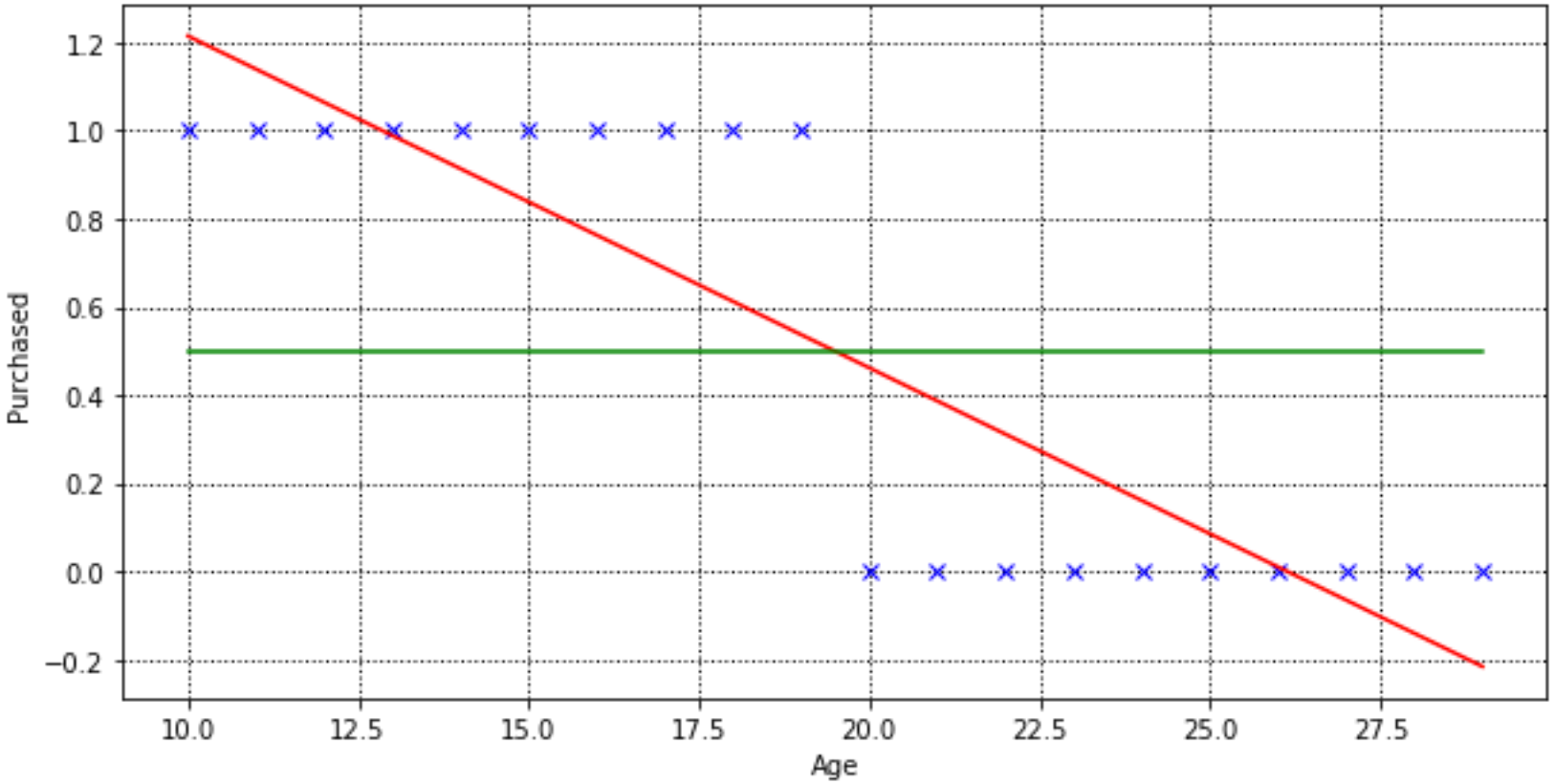
Decision boundary

What we want is a decision boundary



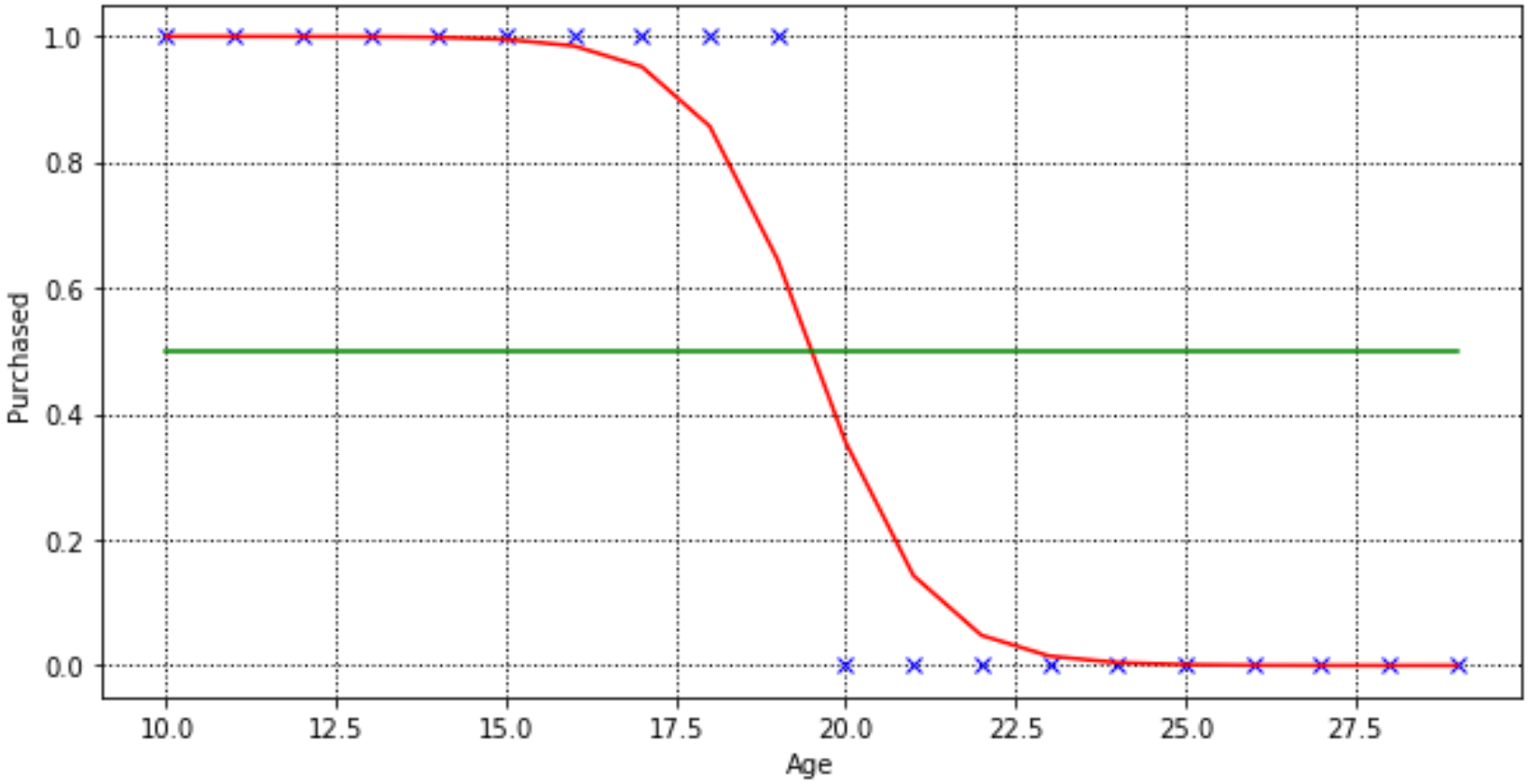
With 2 variables and 2 classes, it is a line if the separation is linear

Coming back to regression



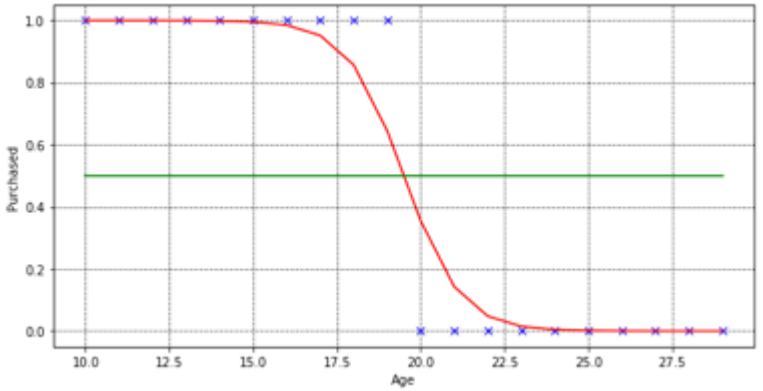
Problem: Y should be binary and not continuous

Coming back to regression



Logistic regression

Activation function



Logistic regression

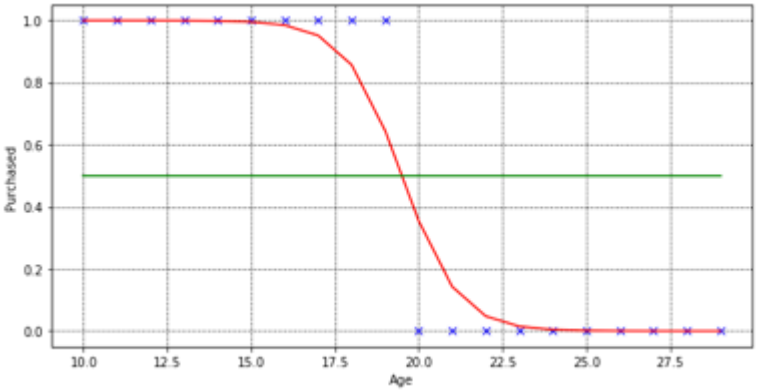
Activation function

Remember the model of linear regression:
$y = f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$

Now we want $0 \leq f(\boldsymbol{x}) \leq 1$

Define $g(z) = \frac{1}{1+e^{-z}}$

This is called a logistic function.

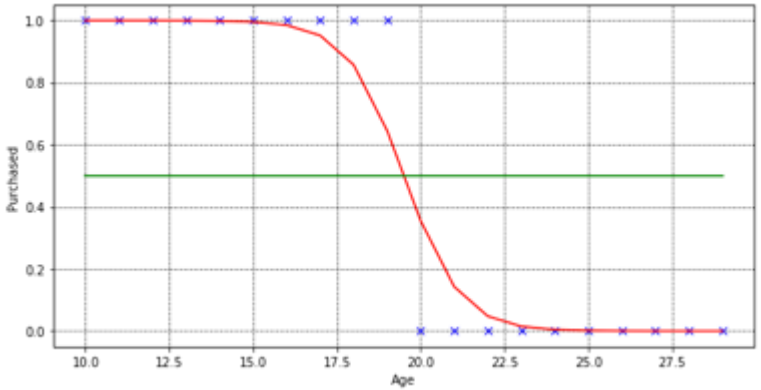


Note: il belongs to the more general class of sigmoid functions (functions which have an S shape).
For example: $\tanh(z)$, $\arctan(z)$
These will be called activation functions in neural networks.

$f(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{w}^T \boldsymbol{x}}}$

Logistic regression

Probabilistic interpretation

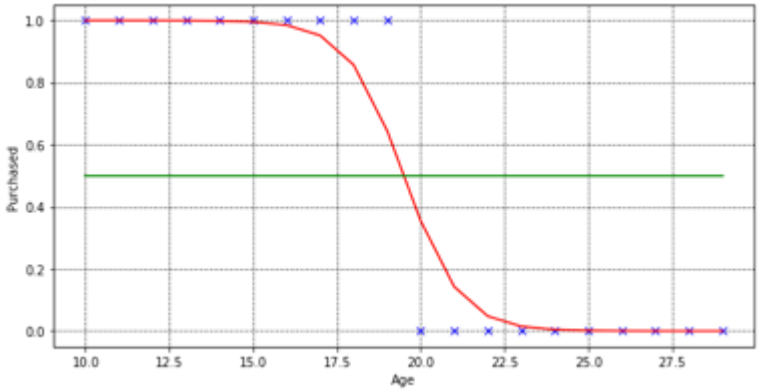


Logistic regression

Probabilistic interpretation

f(x) is the probability that Y=1 given the input x

In our example: probability that the client has purchased given his age



$$f(\boldsymbol{x}) = P(Y = 1 \mid \boldsymbol{x}, \boldsymbol{w})$$

For instance, $f(\boldsymbol{x}) = P(purchased \mid age, model)$

$P(Y = 0 \mid \boldsymbol{x}, \boldsymbol{w}) = 1 - f(X)$

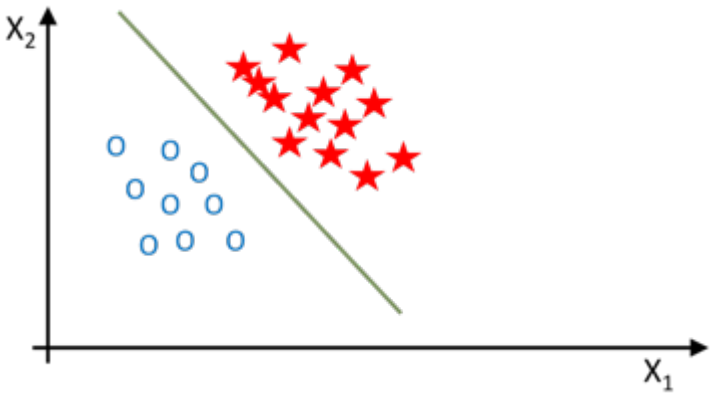
Logistic regression

Discriminant function

Predict Y=1 if $f(\boldsymbol{x}) \geq 0.5 \iff \boldsymbol{w}^T\boldsymbol{x} \geq 0$

Line

This is a **linear classification** technique even though σ is a non-linear function



Logistic regression

Cost function

Cost function:

$$J(f) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, f(x^{(i)})\right)$$

Loss $\ell(y, f(x))$

Measures the difference between the predicted output and the true output

Learning:

$$\hat{f} = arg\min_{f \in \mathcal{F}} J(f)$$

Logistic regression

Loss

$$\ell(y, f(x))$$

Low if prediction is correct

High if prediction is wrong

Logistic regression

Loss

if $y = 1$, $\ell(y, f(x))$ should be low when $f(x)$ is high

if $y = 0$, $\ell(y, f(x))$ should be low when $f(x)$ is low

if $y = 1$, $\ell(y, f(x)) = -\log(f(x))$

if $y = 0$, $\ell(y, f(x)) = -\log(1 - f(x))$

This can be summarized as

$$\ell(y, f(x) = -y \log(f(x) - (1 - y) log(1 - f(x))$$

Logistic regression

Cost function

$$\ell(y, f(x) = -y \log(f(x) - (1-y) log(1 - f(x))$$

$$J(f) = -\frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} \log(f(x^{(i)}) + (1 - y^{(i)}) log(1 - f(x^{(i)})) \right)$$

Logistic regression

Learning

Learning:

$$\hat{f} = arg\min_{f \in \mathcal{F}} J(f)$$

$$J(f) = -\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}\log(f(x^{(i)}) + (1-y^{(i)})log(1-f(x^{(i)}))\right)$$

There is no exact solution (unlike for linear regression)

We will have to find an approximate solution (using an optimization algorithm)

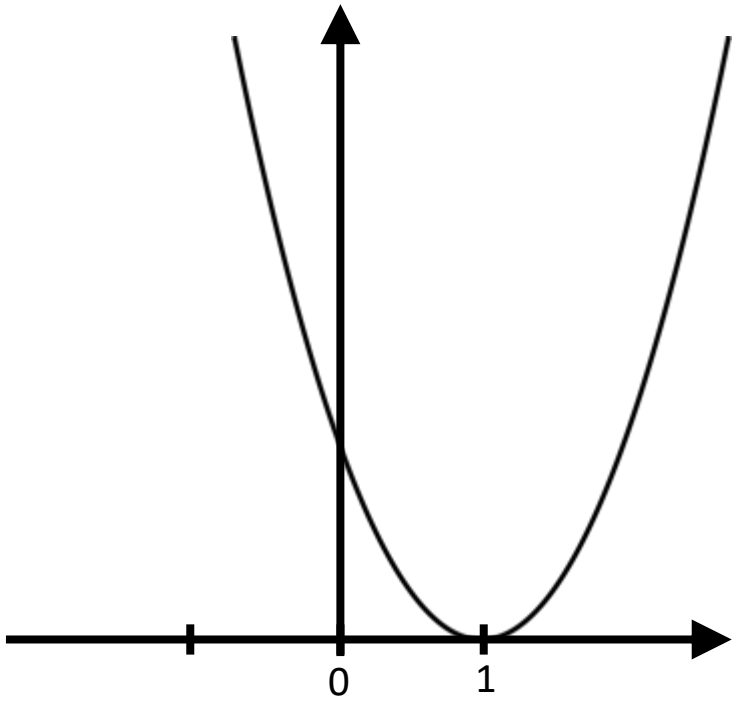
Here, we will use the **gradient descent algorithm**

Gradient descent

Idea:

- Start with an initial value of w_1
- Keep changing w_1 so that it reduces $J(w_1)$

Case with one parameter

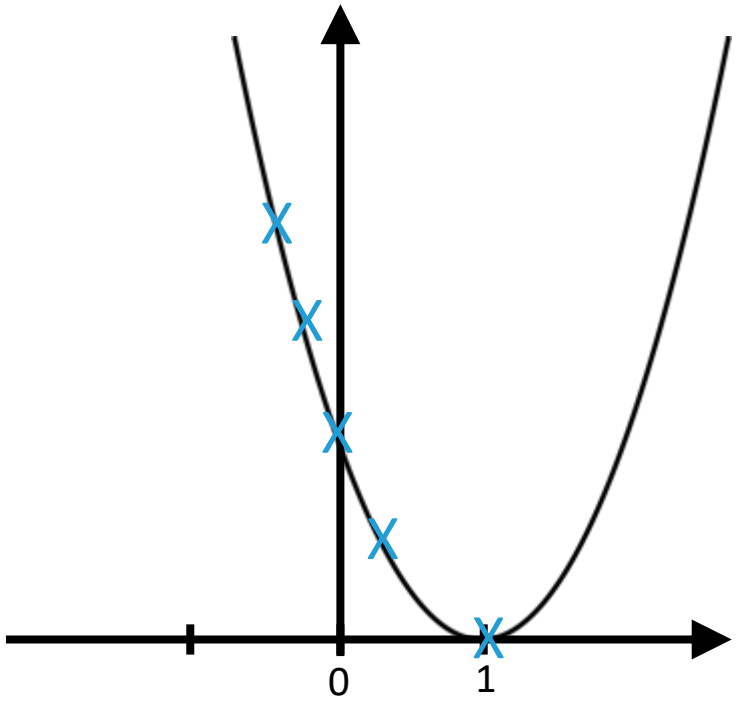


Gradient descent

Idea:

- Start with an initial value of w_1
- Keep changing w_1 so that it reduces $J(w_1)$

Case with one parameter

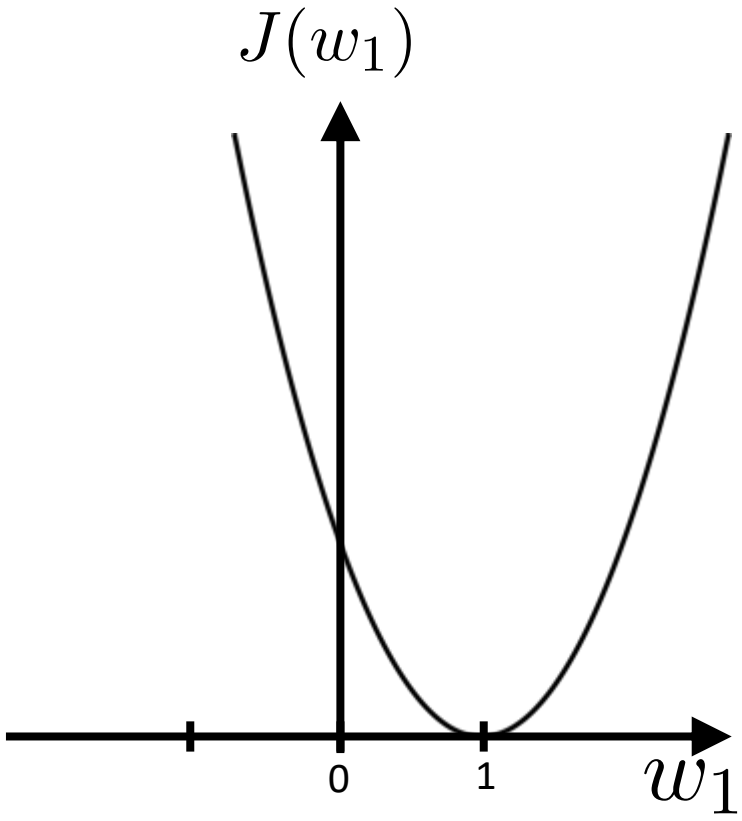


Gradient descent

How do we update w_1 ?

- Use the derivative of $J(w_1)$

Case with one parameter

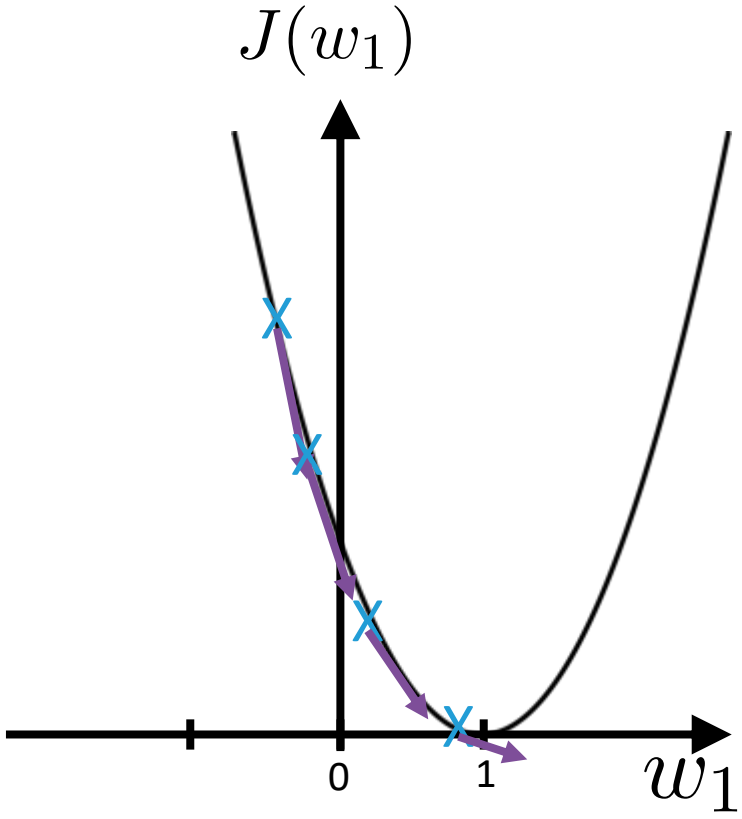


Gradient descent

How do we update w_1 ?

- Use the derivative of $J(w_1)$ $\frac{dJ}{dw_1}$

Case with one parameter



Gradient descent

Gradient descent algorithm

Repeat until convergence {

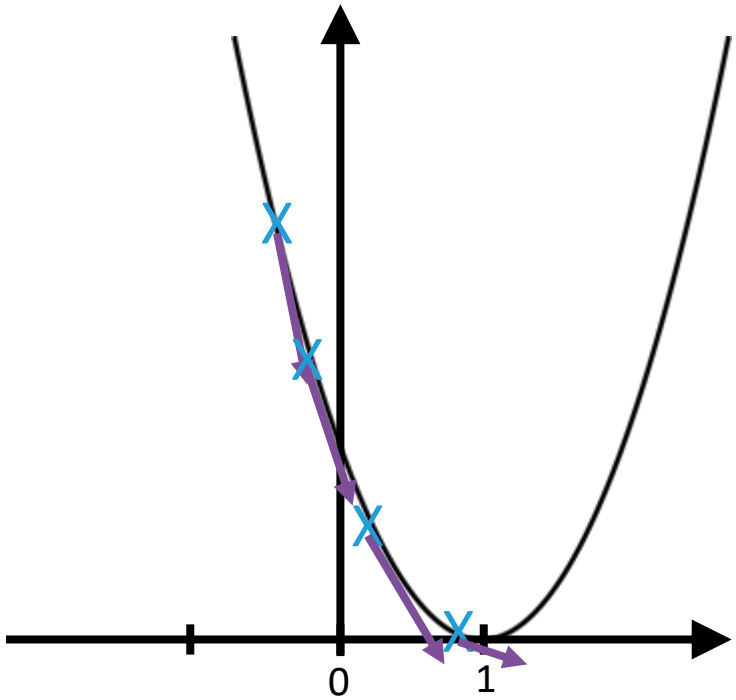
$$w_1 \leftarrow w_1 - \eta \frac{dJ}{dw_1}$$

}

derivative

Learning rate

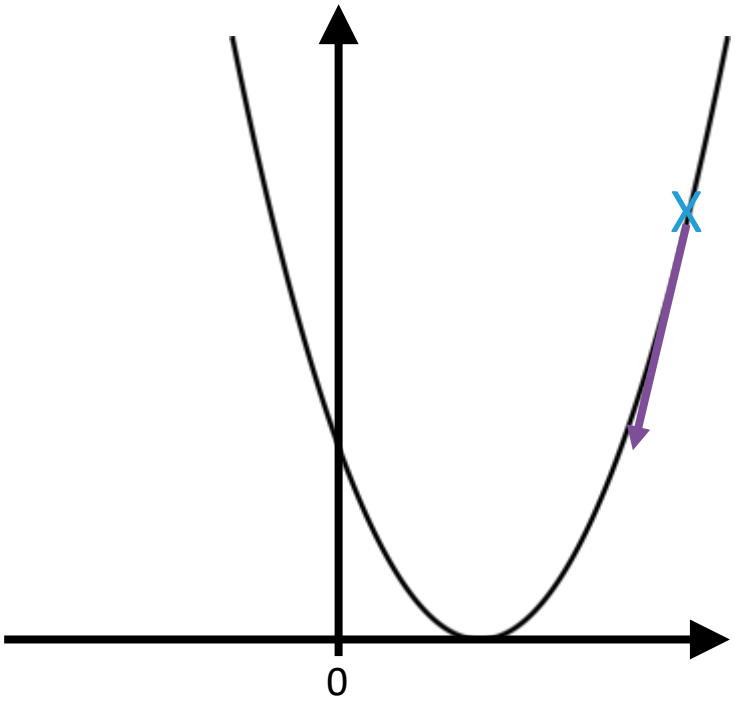
Case with one parameter



Gradient descent

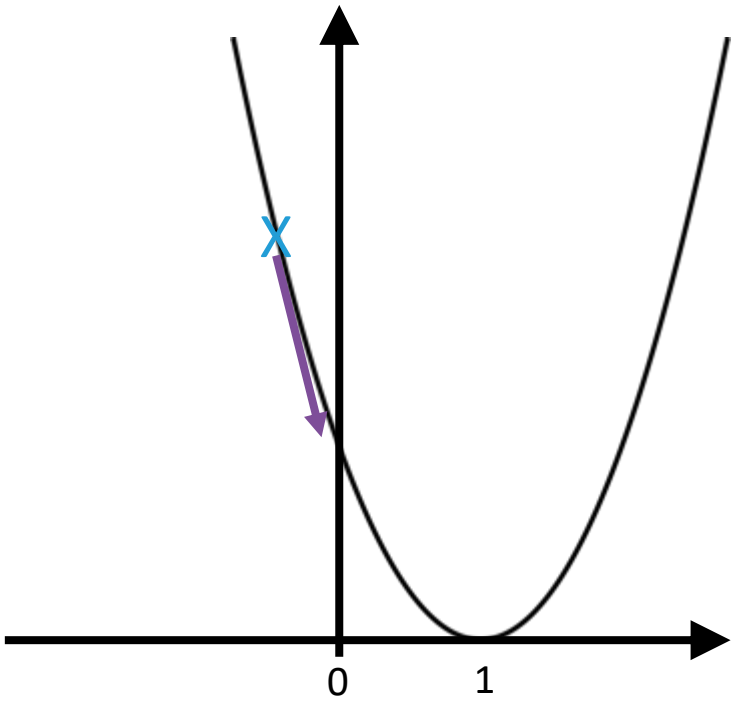
$$w_1 \leftarrow w_1 - \eta \frac{dJ}{dw_1}$$

positive



$$w_1 \leftarrow w_1 - \eta \frac{dJ}{dw_1}$$

negative

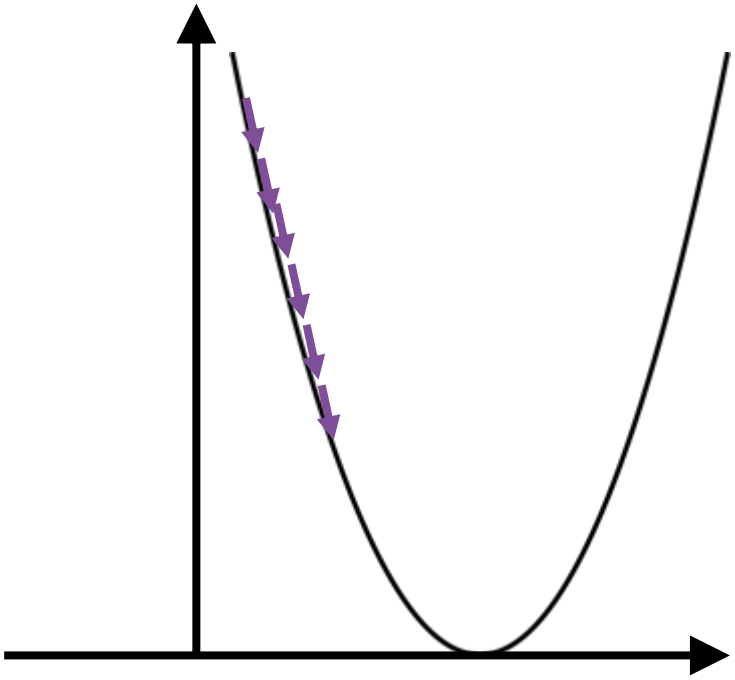


Gradient descent

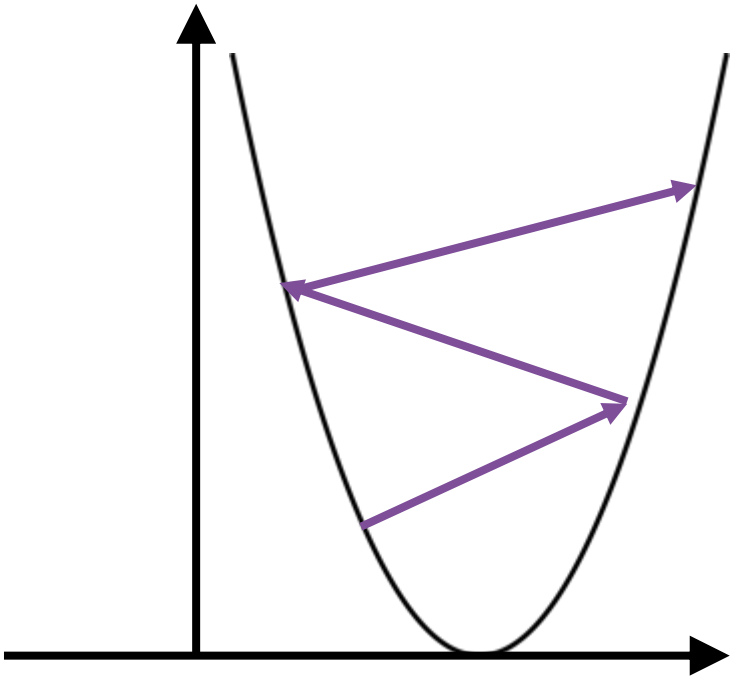
How to choose the learning rate.

$$w_1 \leftarrow w_1 - \eta \frac{dJ}{dw_1}$$

- If η is too small: slow to converge



- If η is too large: may diverge



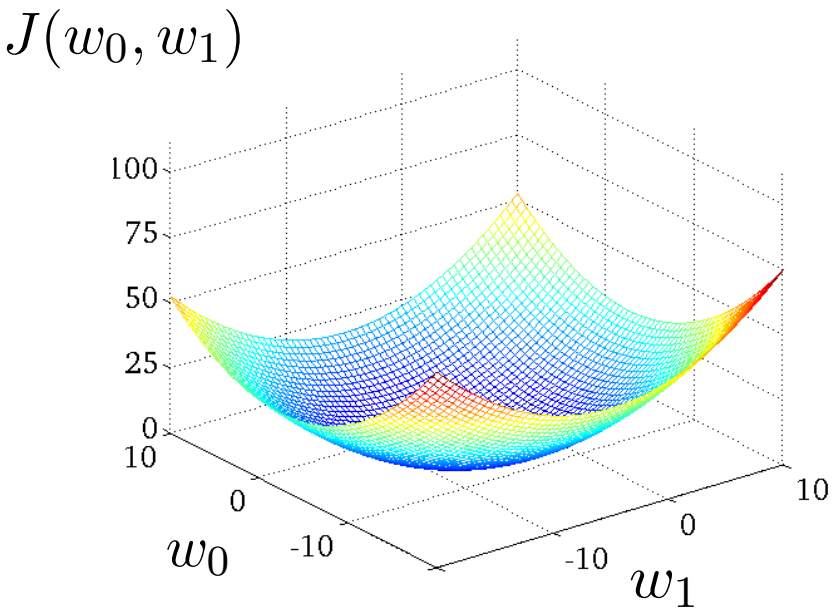
Gradient descent

Cost function $J(w_0, w_1)$

Partial derivatives $\frac{\partial J}{\partial w_0}$ $\frac{\partial J}{\partial w_1}$

Gradient $\nabla J = \left(\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}\right)$ $\boldsymbol{\nabla} J$ or $\frac{\partial J}{\partial \boldsymbol{w}}$

Case with two parameters



Gradient descent

Gradient descent algorithm

Repeat until convergence {

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla J$$

gradient

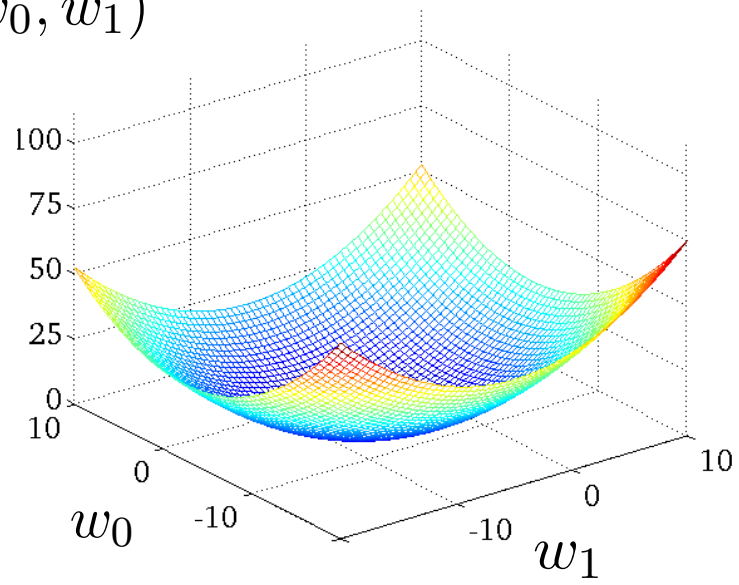
}

Learning rate

Case with two parameters

$J(\beta_1)$

$J(w_0, w_1)$



Gradient descent

Gradient descent algorithm

Repeat until convergence {

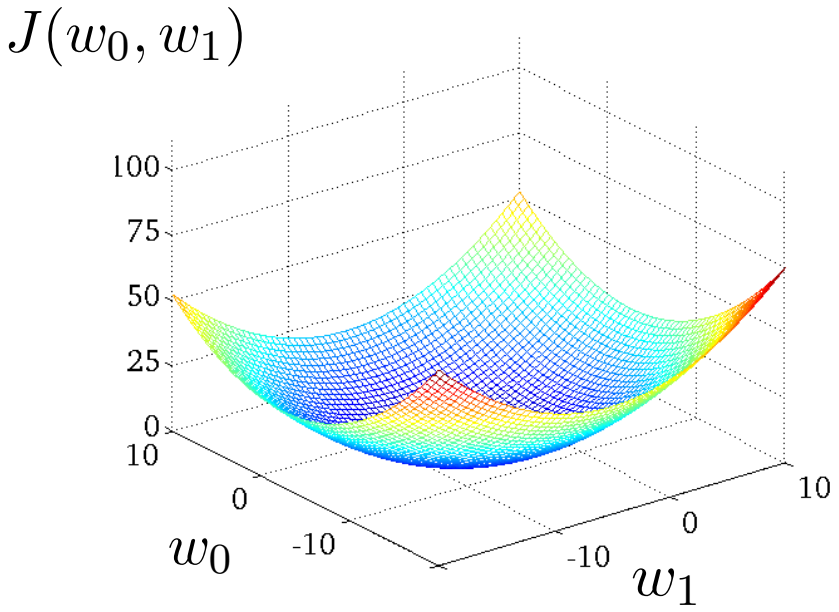
$$w_0 \leftarrow w_0 - \eta \frac{\partial J}{\partial w_0}$$

$$w_1 \leftarrow w_1 - \eta \frac{\partial J}{\partial w_1}$$

}

The two updates need to be simultaneous (i.e. do not use the new version of w0 to compute the update of w1)

Case with two parameters

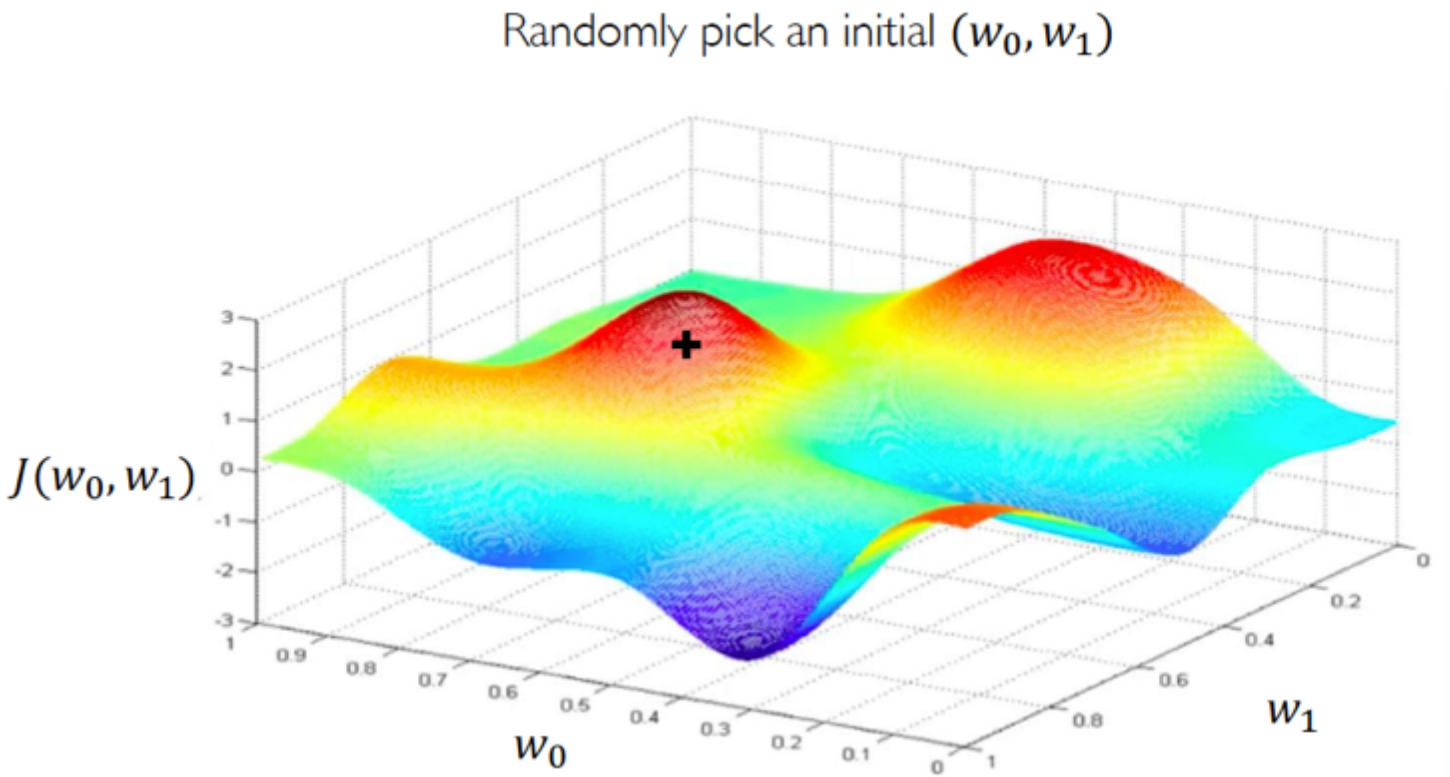


Gradient descent

Logistic regression has a convex cost function.

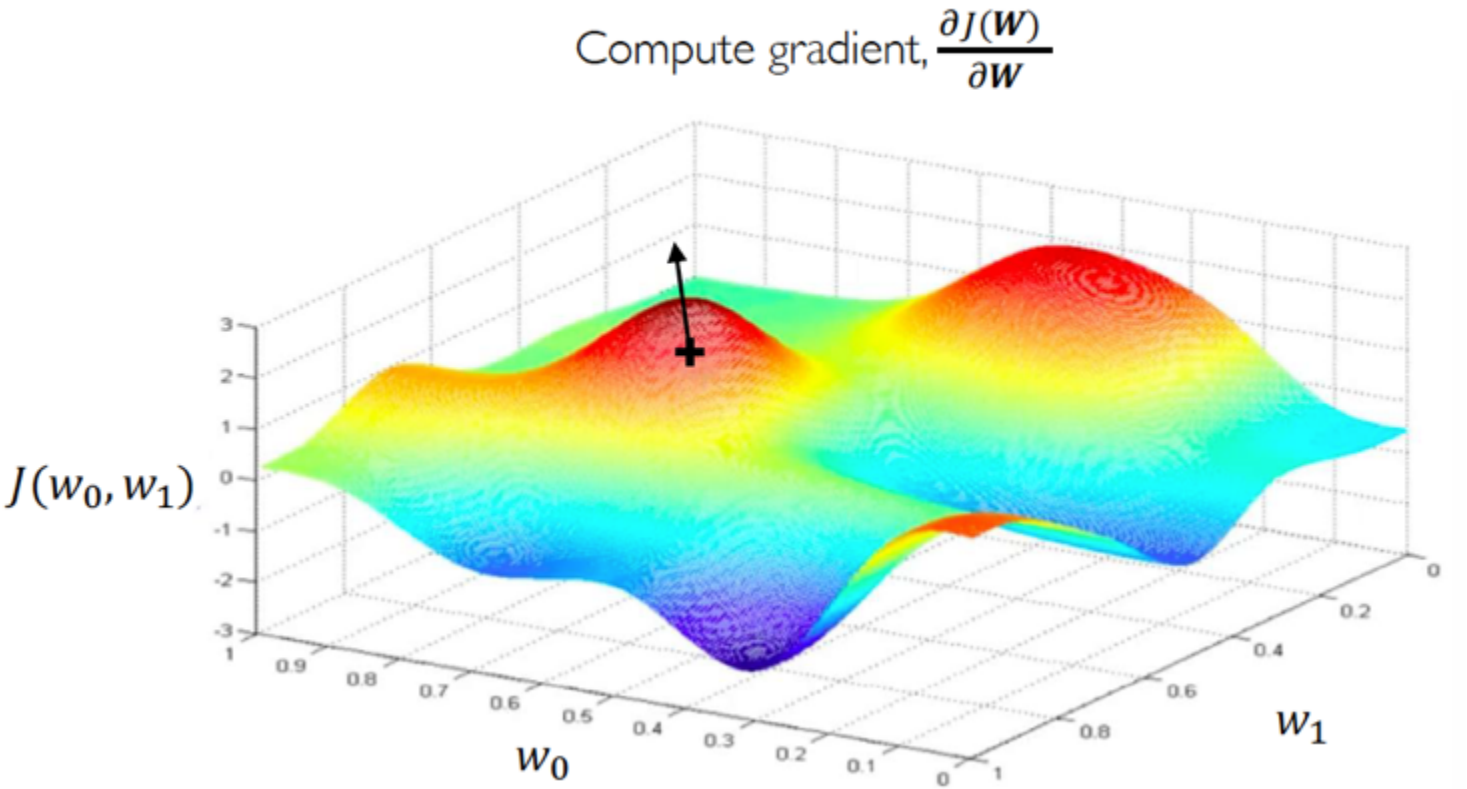
Gradient descent can also be applied to non-convex costs (but no garantee to find the global minimum)

Gradient descent



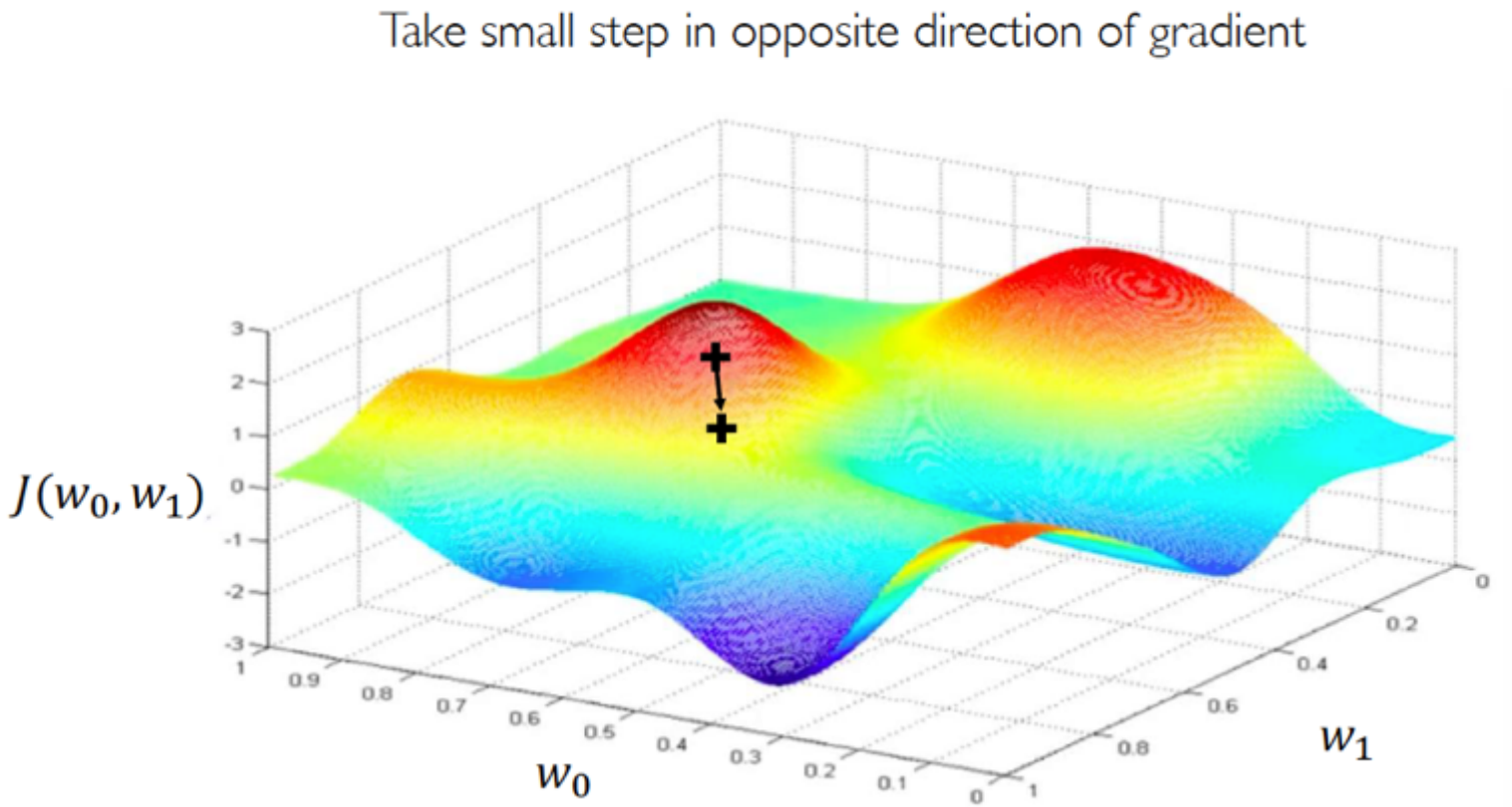
Source: http://introtodeeplearning.com/

Gradient descent



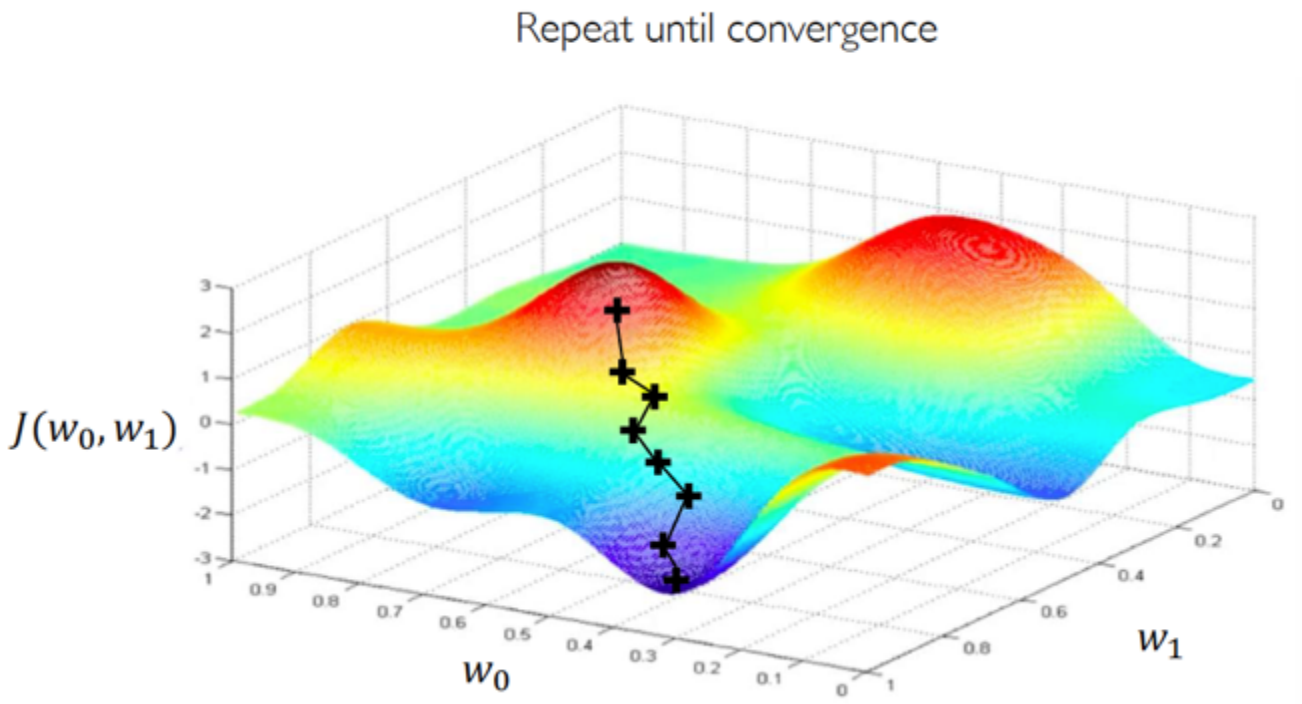
Source: http://introtodeeplearning.com/

Gradient descent



Source: http://introtodeeplearning.com/

Gradient descent



Source: http://introtodeeplearning.com/

Summary

Input variable (multivariate): $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$

Output: y

Model: f, $y = f(x)$ The "artificial intelligence"

Model parameters: $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix}$, indexed by j

Model, depending on parameters: $f(\boldsymbol{x}; \boldsymbol{w})$

Summary

Input variable (multivariate): $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$

Input features

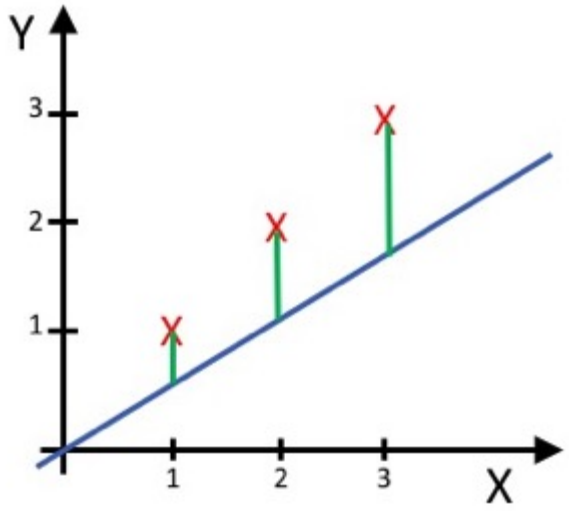
Output: y

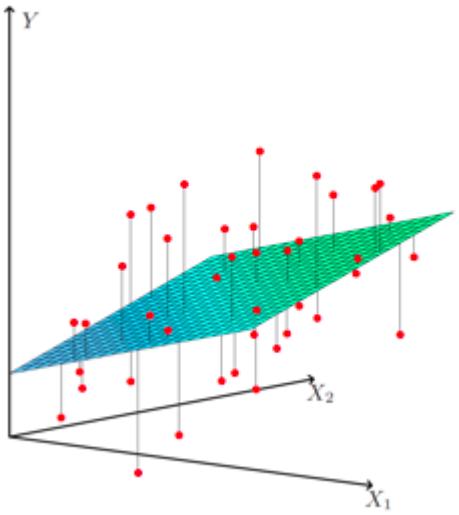
Model: f, $y = f(x)$

The "artificial intelligence"

Loss: $\ell(y, x)$

Quantifies how much the prediction is far from the true output





Summary

Input variable (multivariate): $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$

Input features

Output: y

Model: $f,\ y = f(x)$

The "artificial intelligence"

Loss: $\ell(y, x)$

Quantifies how much the prediction is far from the true output

Cost function:

$$J(f) = \frac{1}{n} \sum_{i=1}^{n} \ell \left(y^{(i)}, f(x^{(i)}) \right)$$

How far are we from the true output across all training examples ?

Learning:

$$\hat{f} = arg \min_{f \in \mathcal{F}} J(f)$$

Learning: find the model with the minimal error

Optimization algorithm: method to find the minimum

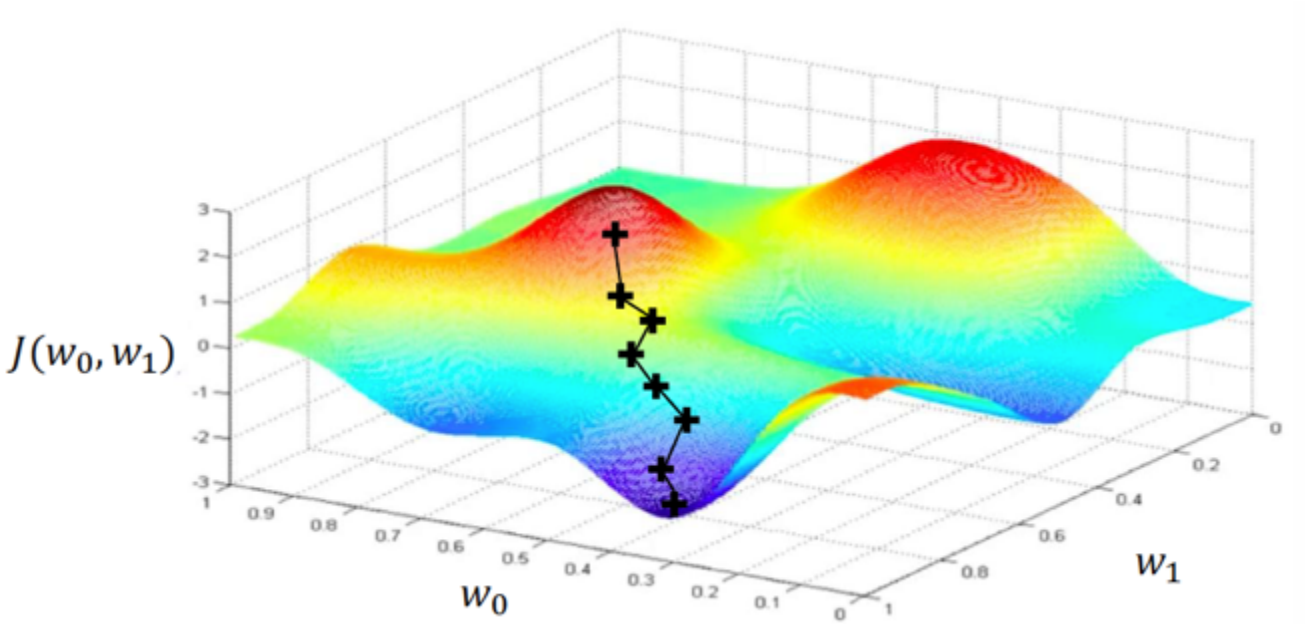
Summary

Learning:

$$\hat{f} = arg\min_{f \in \mathcal{F}} J(f)$$

Learning: find the model with the minimal error

Optimization algorithm: method to find the minimum



Part 2 – Classification and regression
2.2 Introduction to deep learning

Introduction

Labelled training examples

$(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$

Learning

Input

x

f

$y = f(x)$

Beaver

Output

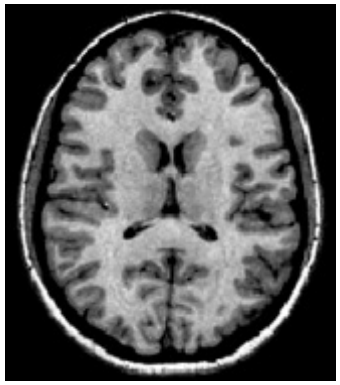
y

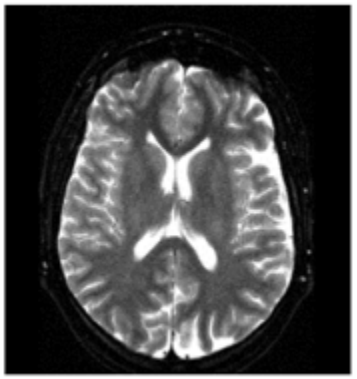
$$\hat{f} = arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, f(x^{(i)})\right)$$

Introduction

- There are many cases where the inputs are not numbers

Natural images





Magnetic resonance images

1. 'To be, or not to be: that is the question'

(Hamlet Act 3, Scene 1)

Text

DNA sequence

Learning features

Labelled training examples

$$(x_1, y_1), \ldots, (x_N, y_N)$$

Learning

Input

X

$$Y = f(\phi(X))$$

Beaver

Output

Y

$$\hat{f} = arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \ell\left(y_i, f\left(\phi(x_i)\right)\right)$$

ϕ is the feature extractor

Learning features

Labelled training examples

Learning

Input

X

$Y = f(\phi(X))$

Beaver

Output

Y

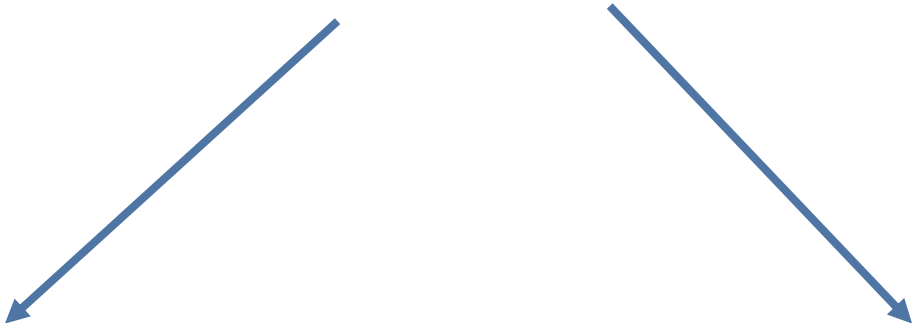
ϕ is the feature extractor

ϕ maps the set of inputs to the set of features $\mathbb{R}^p$

$$\hat{f} = arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \ell\left(y_i, f\left(\phi(x_i)\right)\right)$$

Learning features

How to define ϕ?



Define it "manually"

Feature engineering
"Hand-crafted" features

Learn it

This is the approach used in deep learning but not only

Learning features

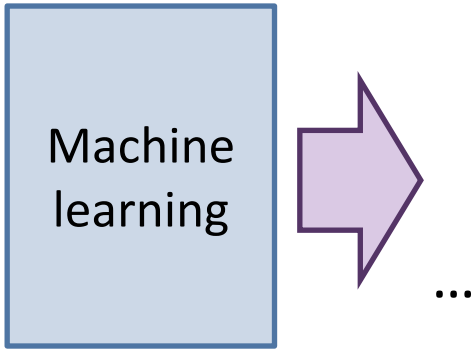
Cat

Beaver

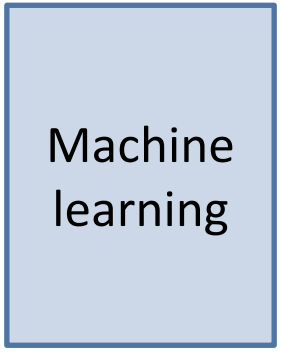
Ouput independent

Machine learning

Deep learning



...

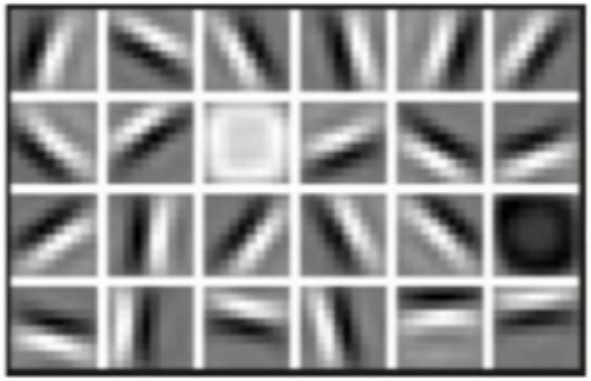


Cat

Beaver

Learning features

Low Level Features



Lines & Edges

Mid Level Features

Eyes & Nose & Ears

High Level Features

Facial Structure

Source: http://introtodeeplearning.com/

What is deep learning?

- Good old Neural Networks, with more layer/modules
- Automatic extraction of features
- Non-linear, hierarchical, abstract representations of data
- Flexible models with any input/output type and size

Source: https://m2dsupsdlclass.github.io/lectures-labs/slides/01_intro_to_deep_learning/index.html#7

Why now?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

Neural Networks date back decades, so why the resurgence?

1. Big Data

- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware

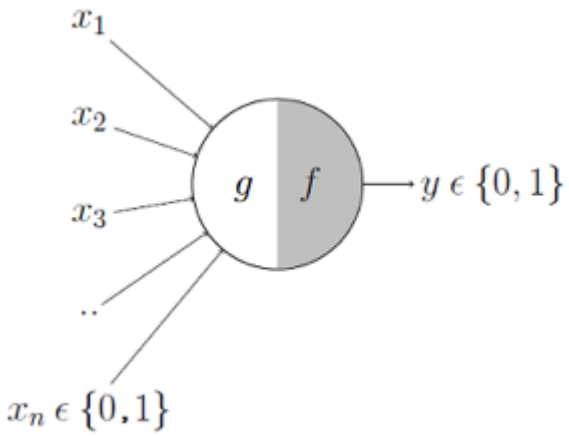
- Graphics Processing Units (GPUs)
- Massively Parallelizable

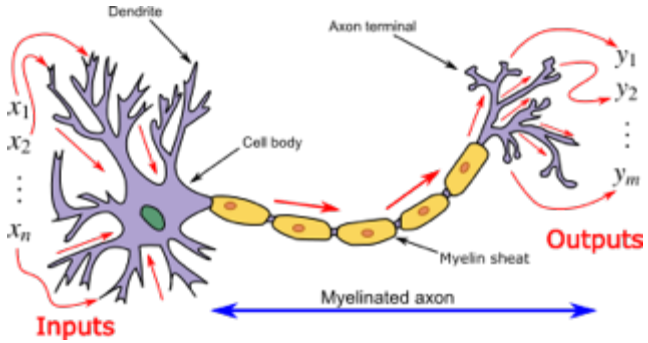
Source: http://introtodeeplearning.com/

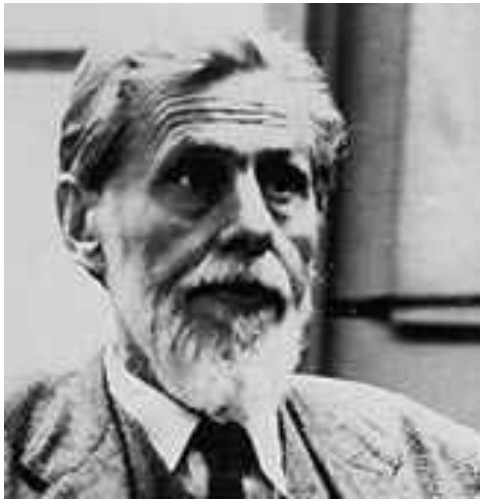
Part 2 – Classification and regression
2.3 Perceptron

1943: artificial neuron model

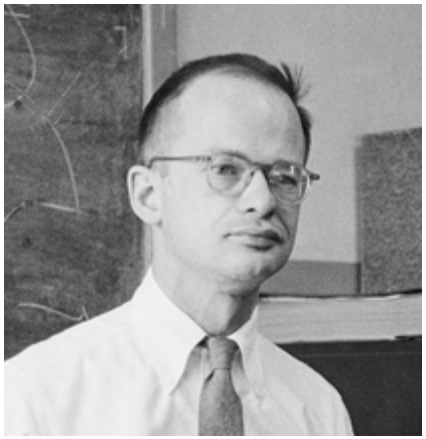
- **McCulloch-Pitt neuron**
  - Artificial neuron model







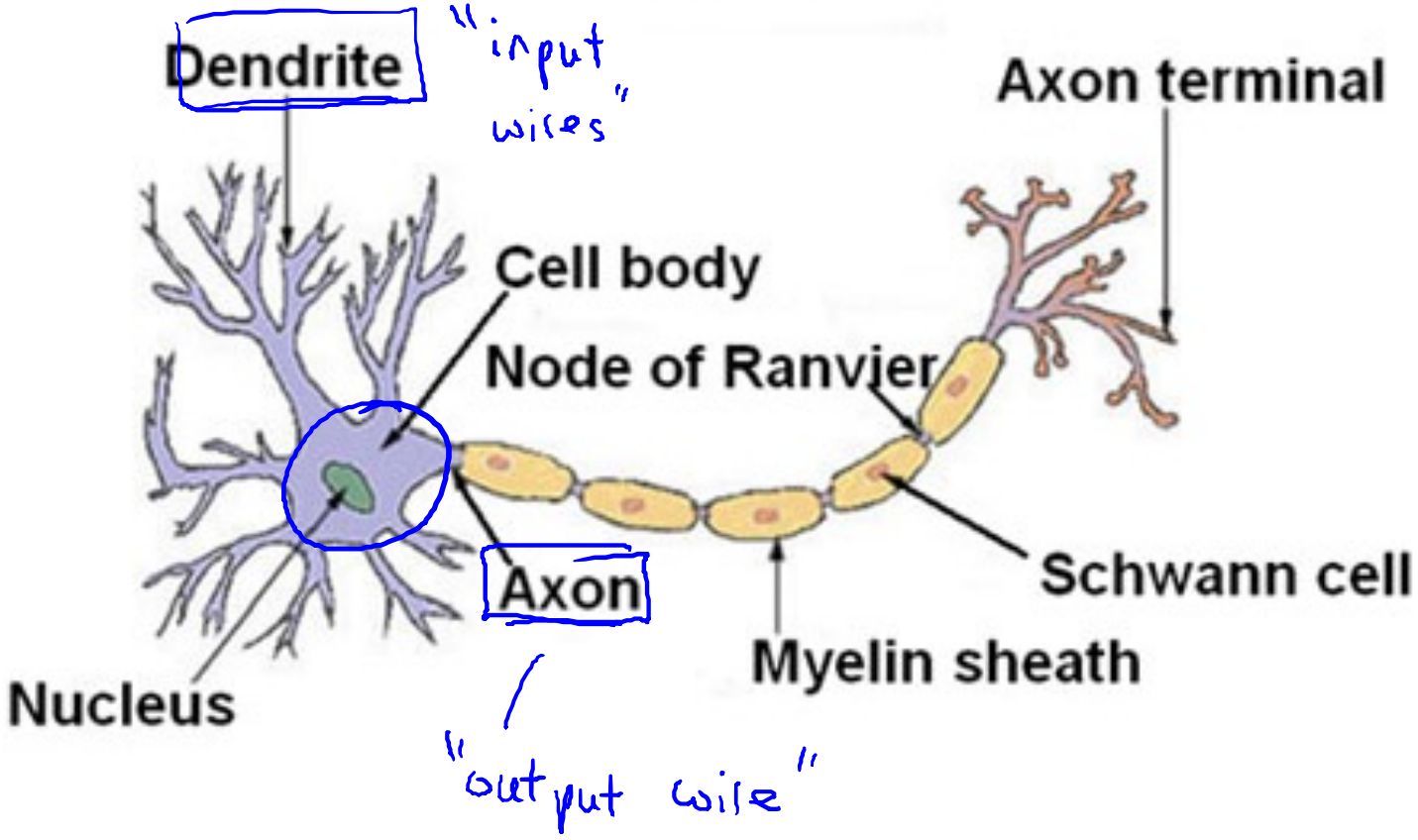
Warren McCulloch



Walter Pitts

Source: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

Neuron



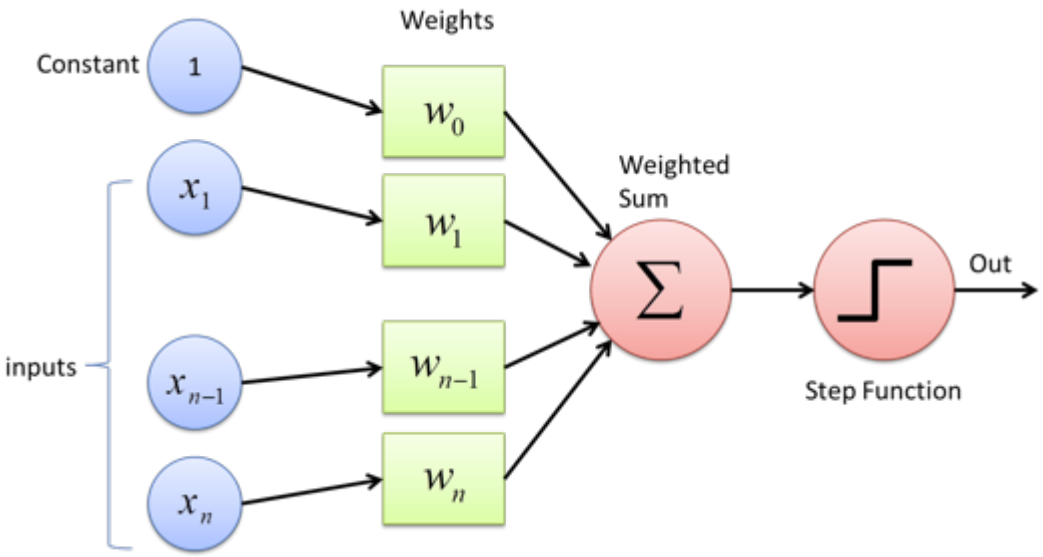
Source: http://introtodeeplearning.com/

1958: The perceptron

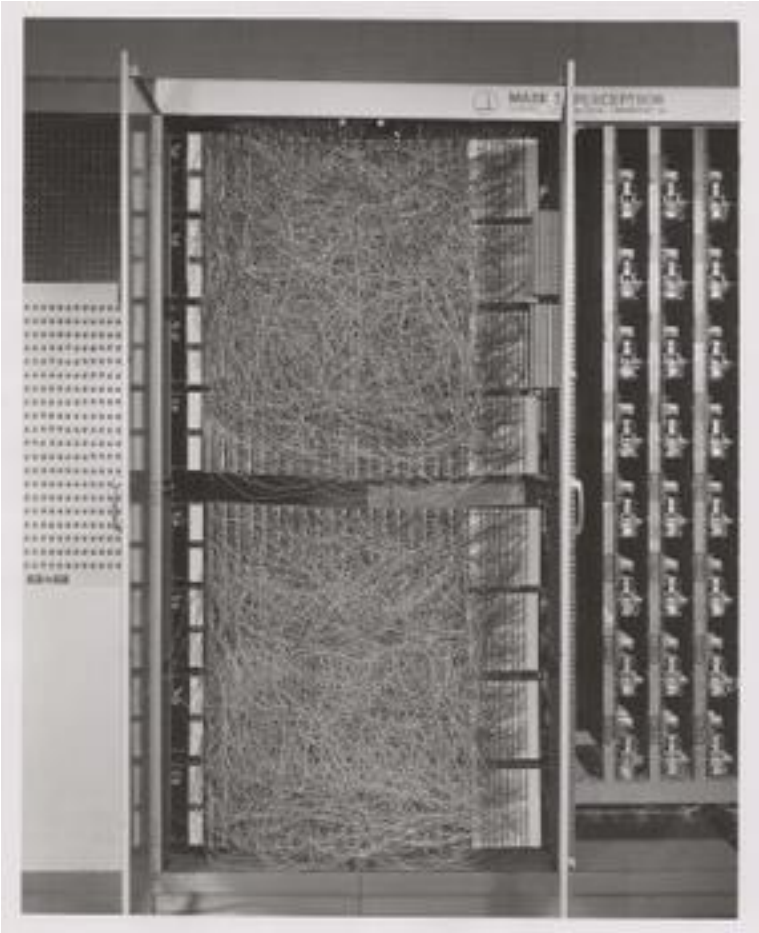
- **Artificial neural network**

- Could recognize letters and numbers

Frank Rosenblatt



1958: The perceptron



Mark I Perceptron machine, hardware implementation of the perceptron algorithm**. It was connected to a camera** with 20×20 cadmium sulfide photocells to make a 400-pixel image. To the right, arrays of **potentiometers** that implemented the adaptive weights.

Funded by US Navy

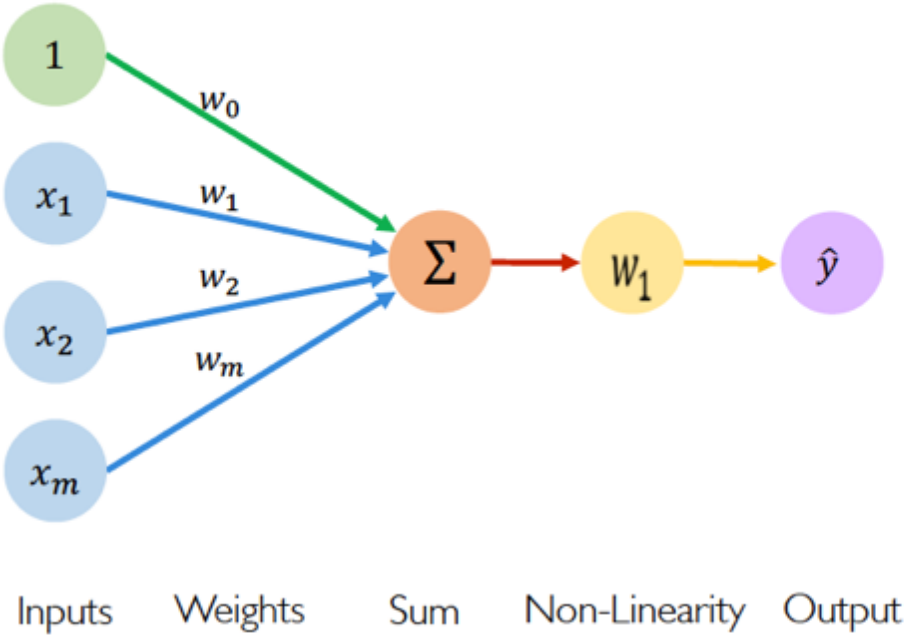
https://en.wikipedia.org/wiki/Perceptron

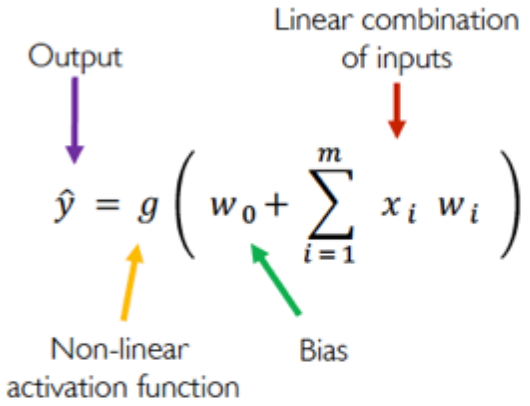
1958: The perceptron

The New York Times

WASHINGTON, July 7 (UPI) -- The Navy revealed the embryo of an electronic computer today that it expects **will be able to walk, talk, see, write, reproduce itself and be conscious of its existence**.

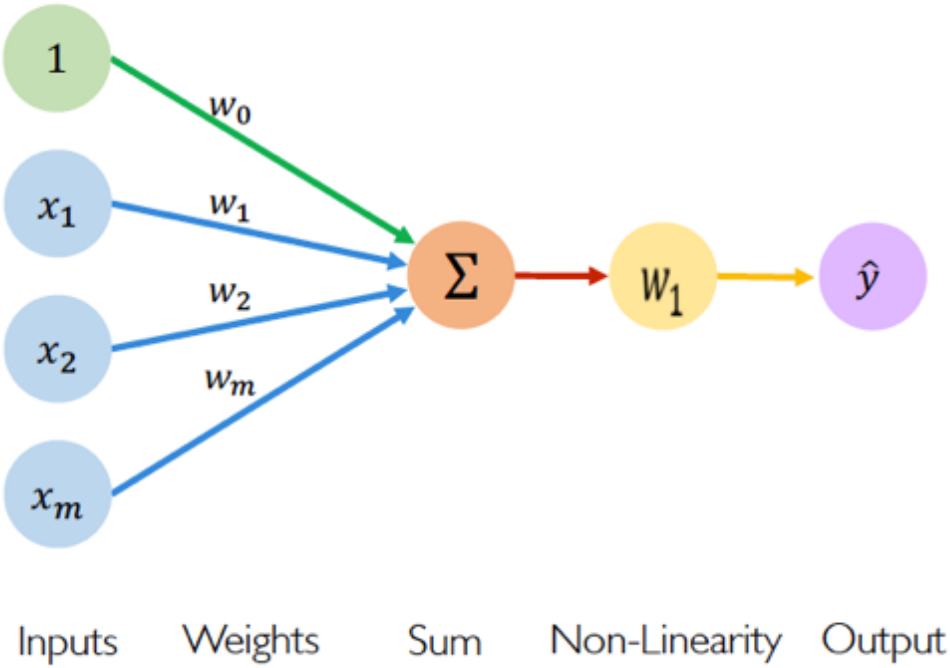
Perceptron





Source: http://introtodeeplearning.com/

Perceptron



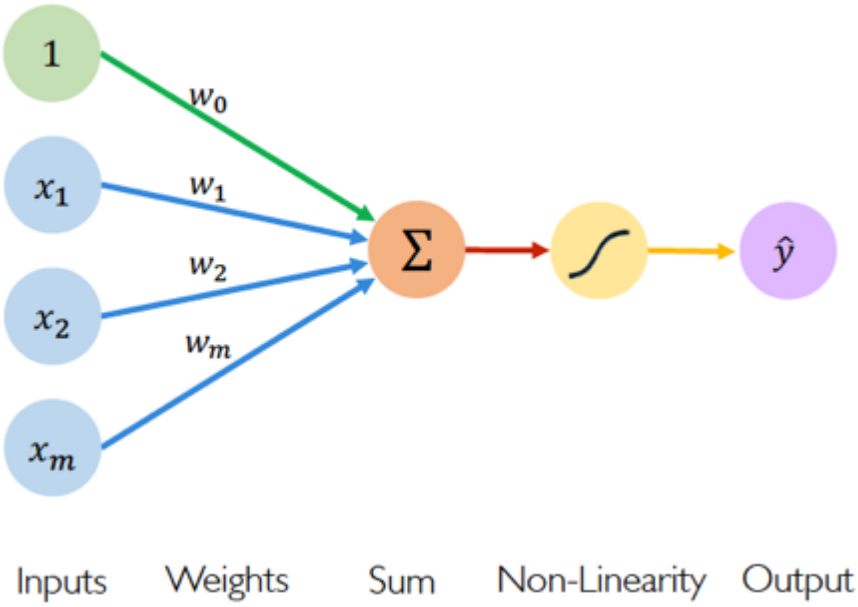
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$$

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

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

Source: http://introtodeeplearning.com/

Perceptron

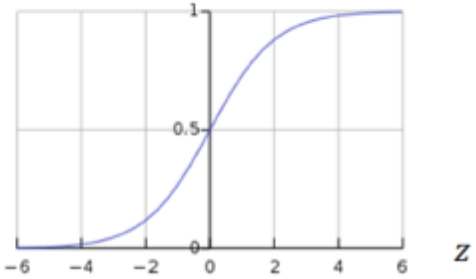


Activation Functions

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

- Example: sigmoid function

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

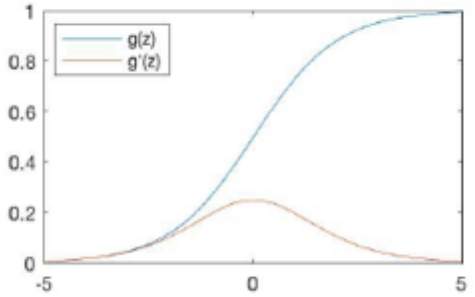


Similar to logistic regression

Source: http://introtodeeplearning.com/

Common activation functions

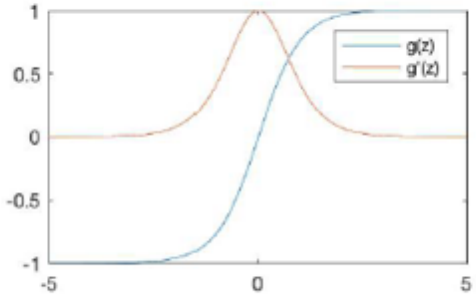
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

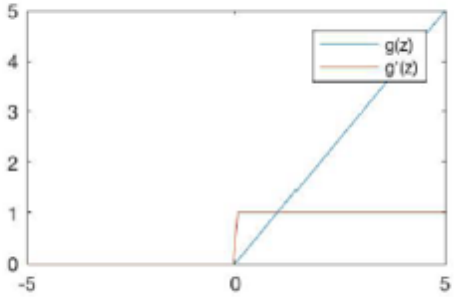
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



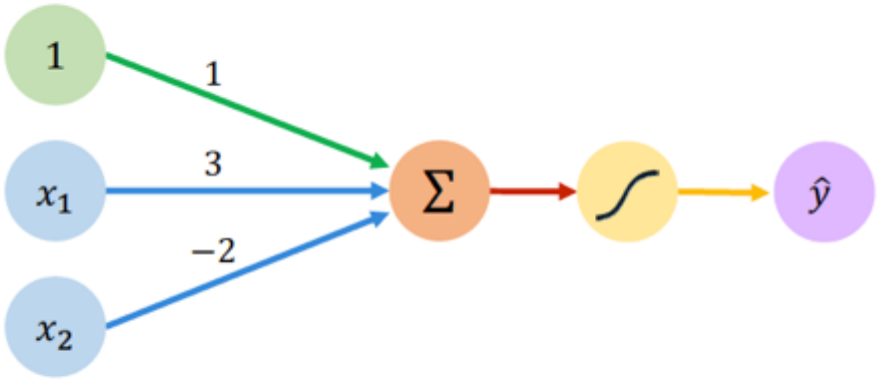
$$g(z) = \max(0, z)$$

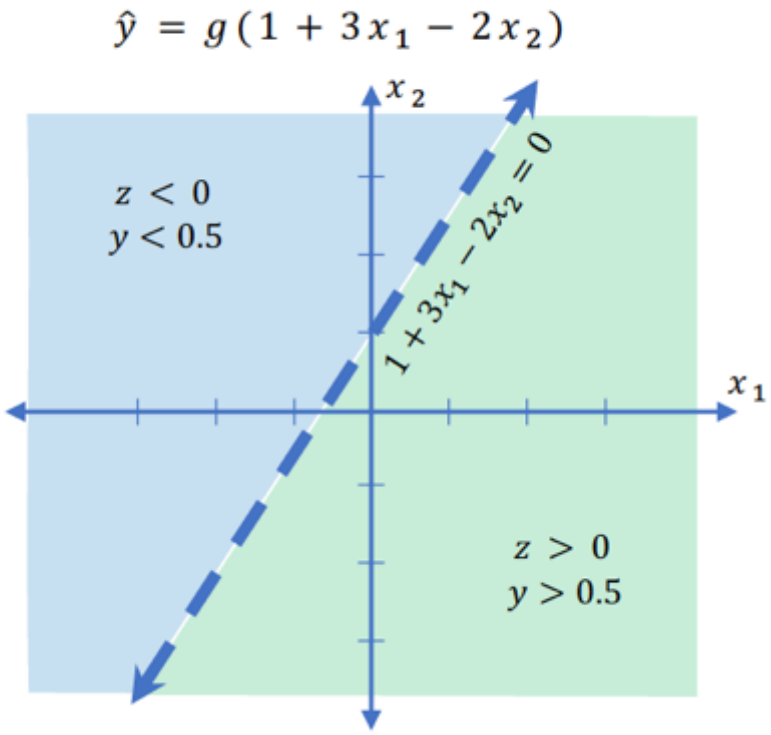
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Source: http://introtodeeplearning.com/

Perceptron

As logistic regression, the (single-layer) perceptron is a linear classifier

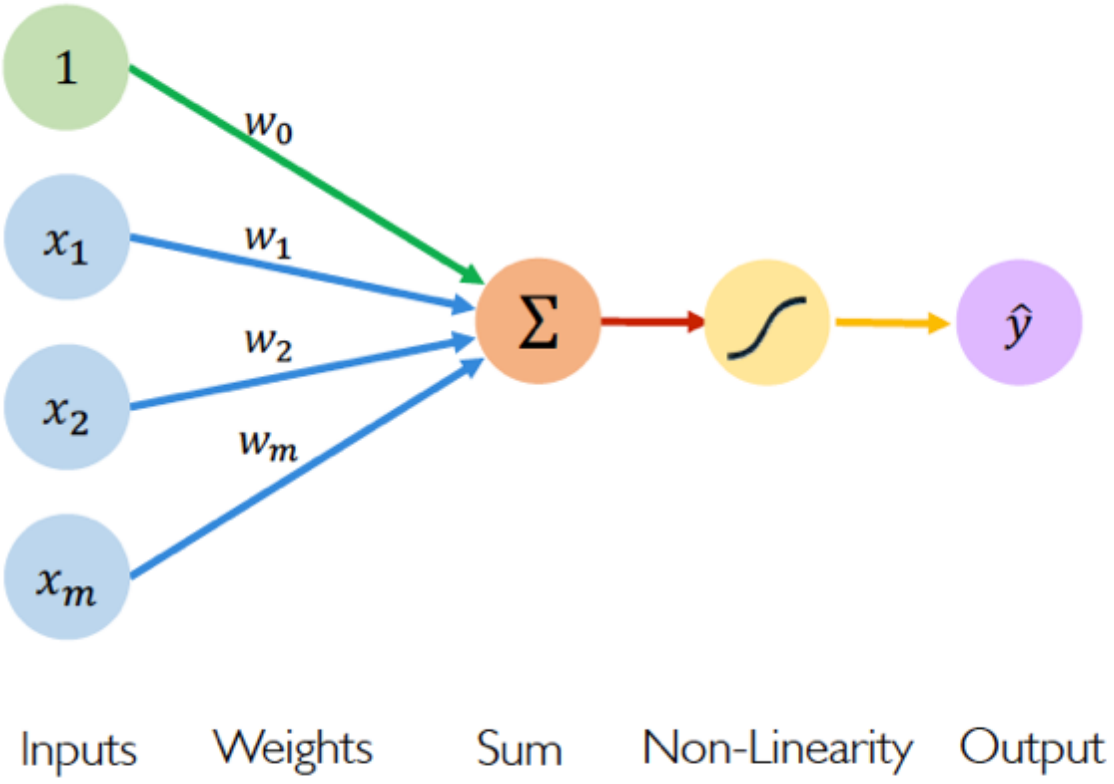




Source: http://introtodeeplearning.com/

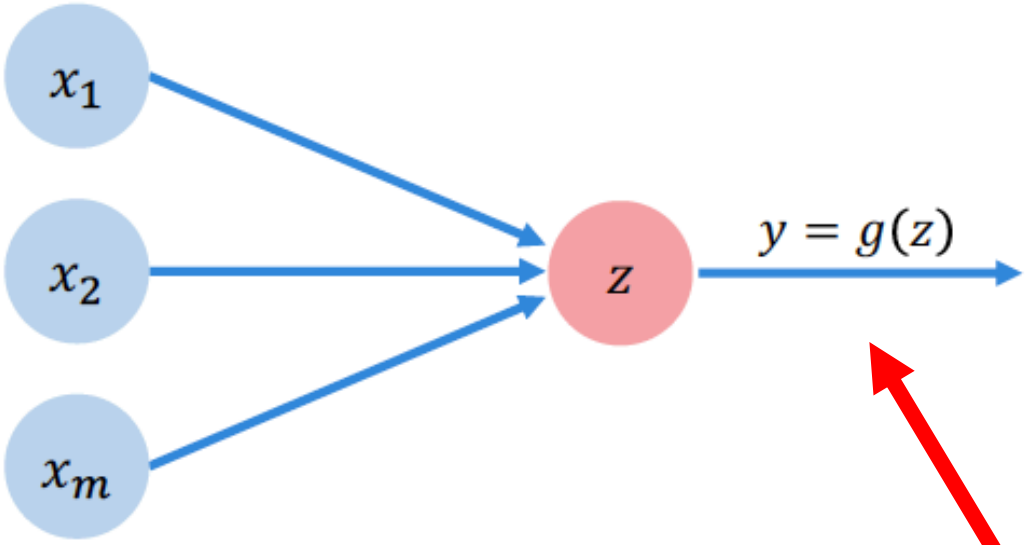
Part 2 – Classification and regression
2.4 Multi-layer perceptron

Perceptron



Source: http://introtodeeplearning.com/

Perceptron (simplified notation)

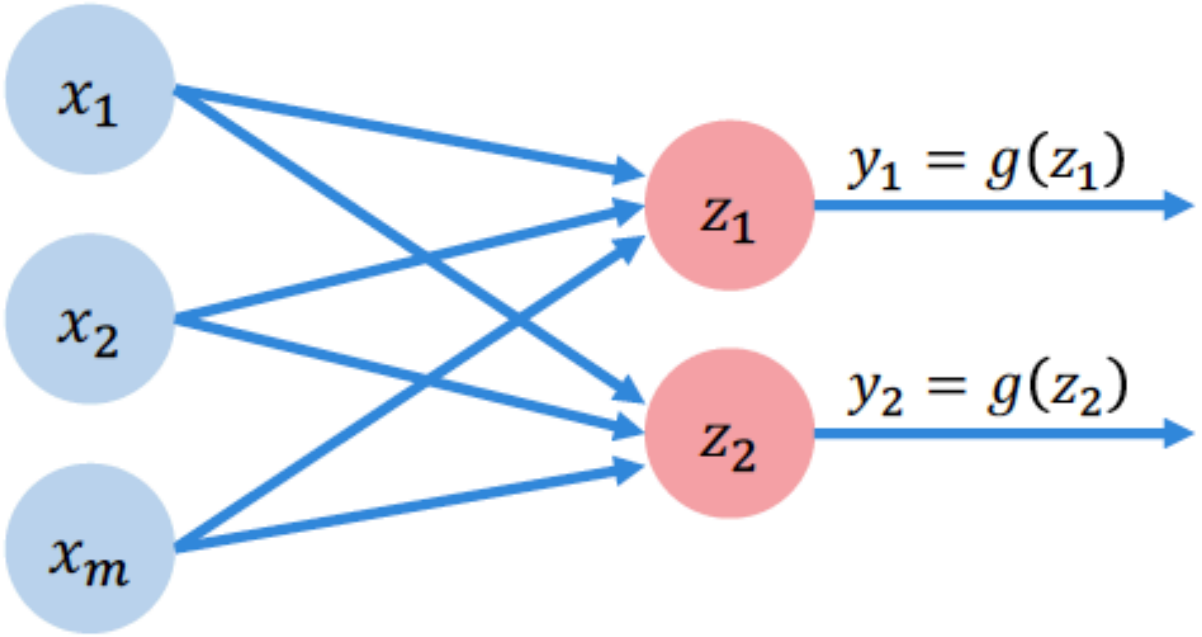


The linearity is now denoted in this arrow instead of as a node

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

Source: http://introtodeeplearning.com/

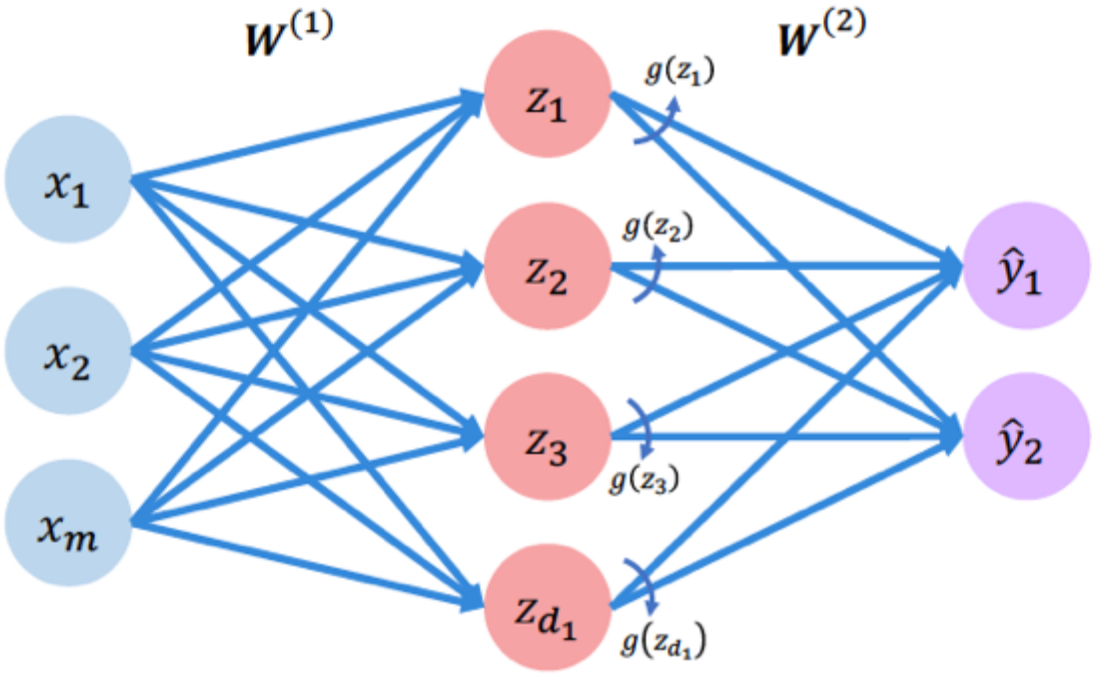
Multiple outputs



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

Source: http://introtodeeplearning.com/

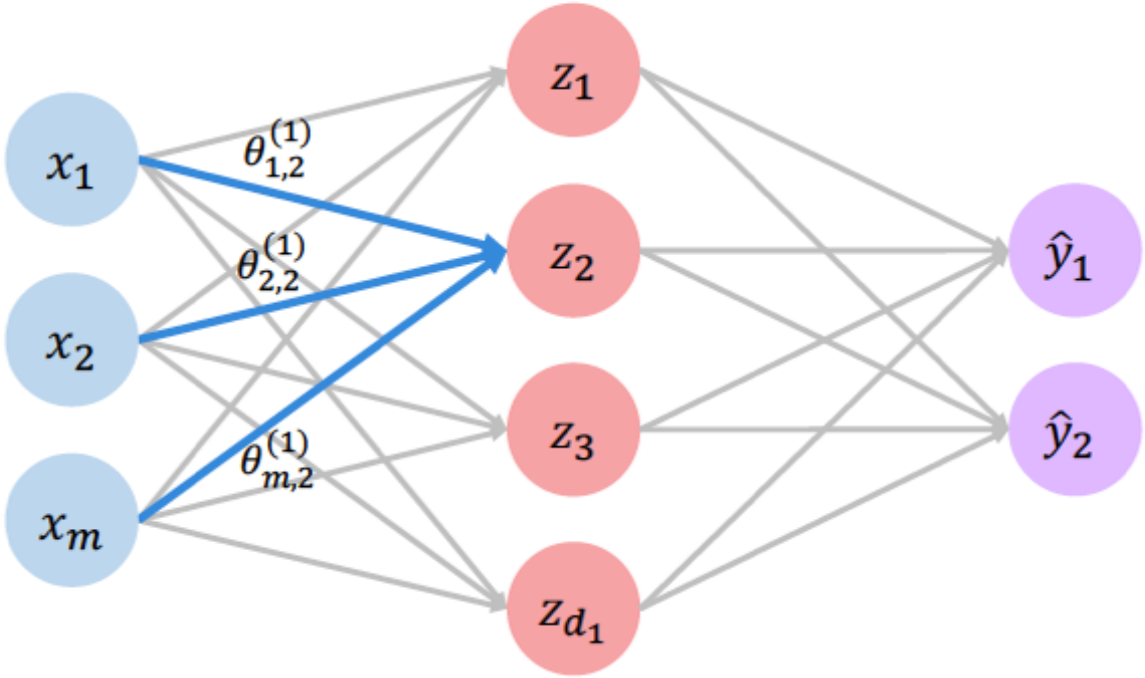
One hidden layer



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

Source: http://introtodeeplearning.com/

One hidden layer

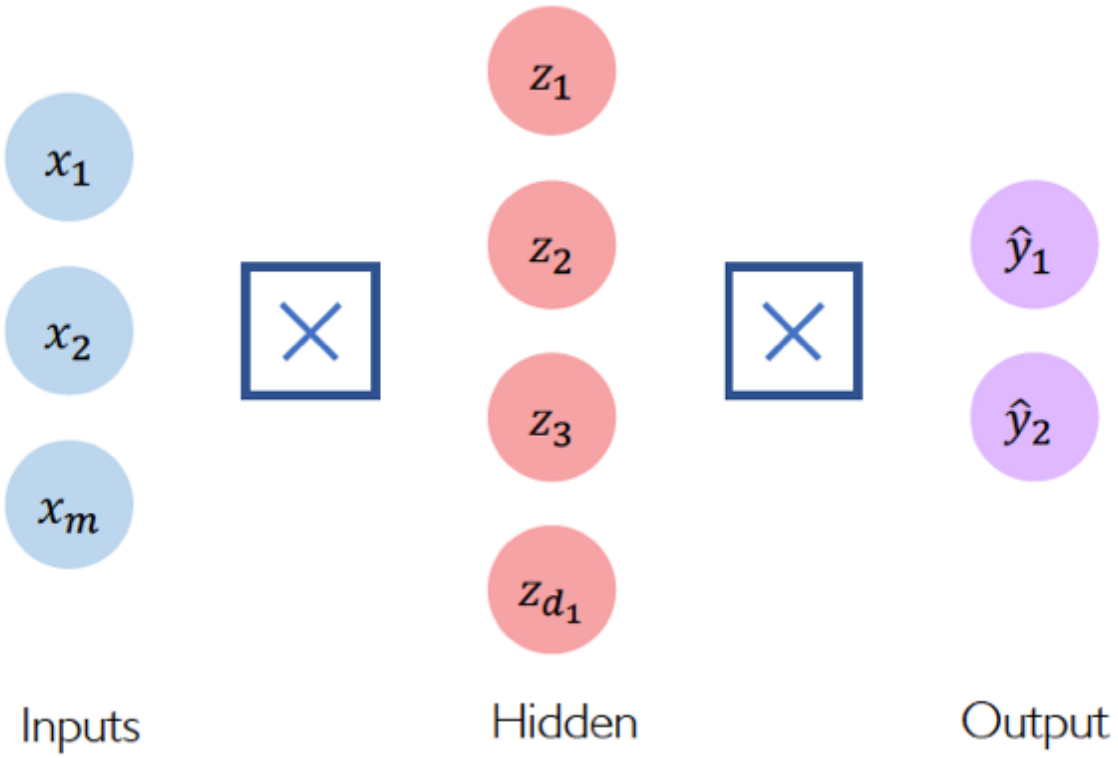


$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}$$

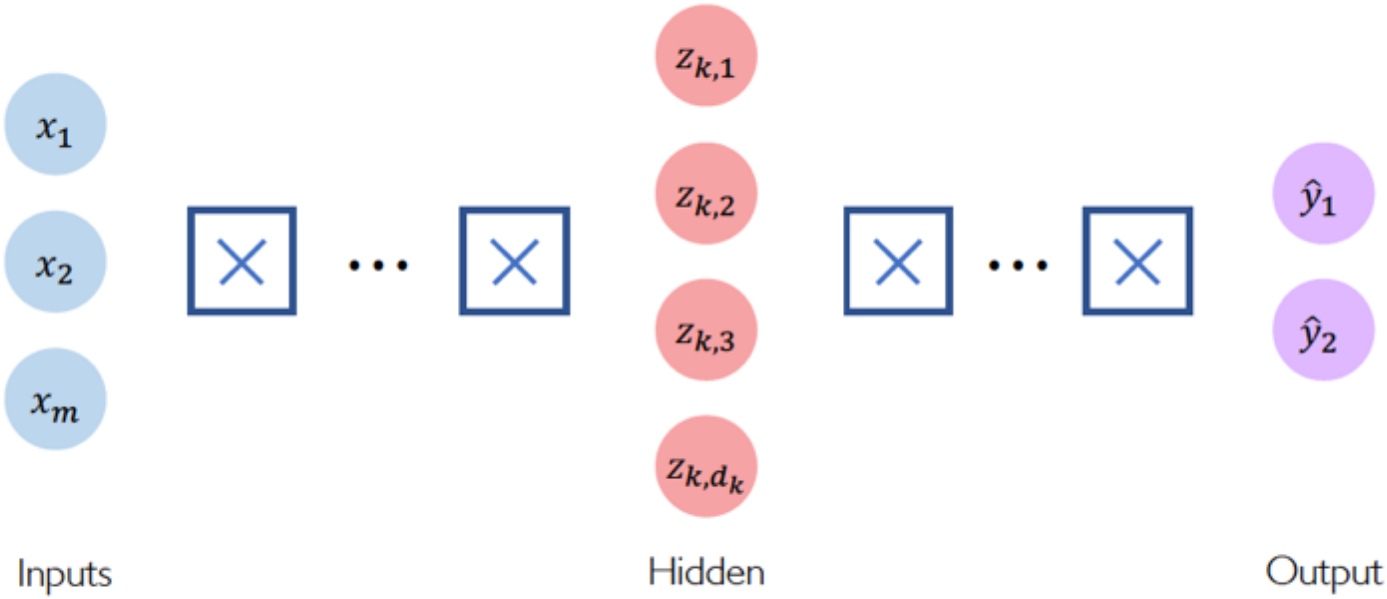
Source: http://introtodeeplearning.com/

Simplified notation



Source: http://introtodeeplearning.com/

Deep neural network

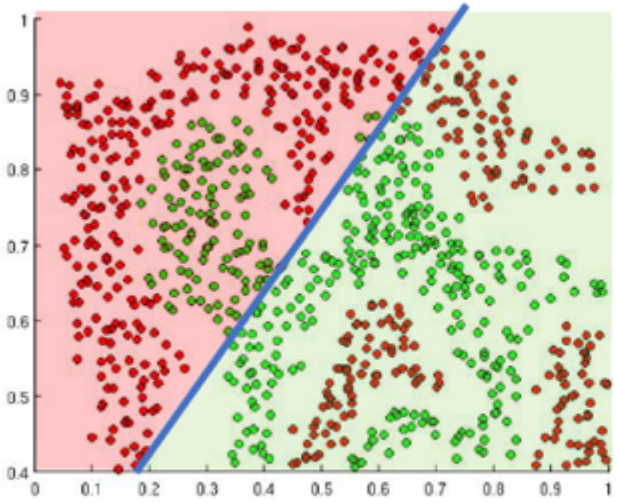


$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

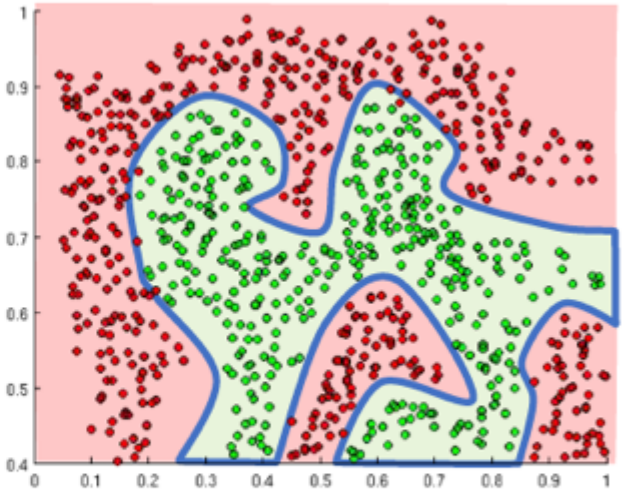
Source: http://introtodeeplearning.com/

Non-linearities

A neural network with multiple layers can learn non-linear boundaries because we use non-linear activation functions



Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Source: http://introtodeeplearning.com/

Part 2 – Classification and regression
2.5 Loss and cost function

Loss for classification

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

Same as in logistic regression

Called cross-entropy loss

Source: http://introtodeeplearning.com/

Loss for regression

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; \boldsymbol{W}\right)\right)^2$$

Actual Predicted

Same as in linear regression

Least squares loss

Source: http://introtodeeplearning.com/

Part 2 – Classification and regression
2.6 Training

Loss and cost function

Learned function: $\hat{f}$

Cost function:

$$J(f) = \frac{1}{N} \sum_{i=1}^{N} \ell\left(y_i, f\left(x_i\right)\right)$$

Learning:

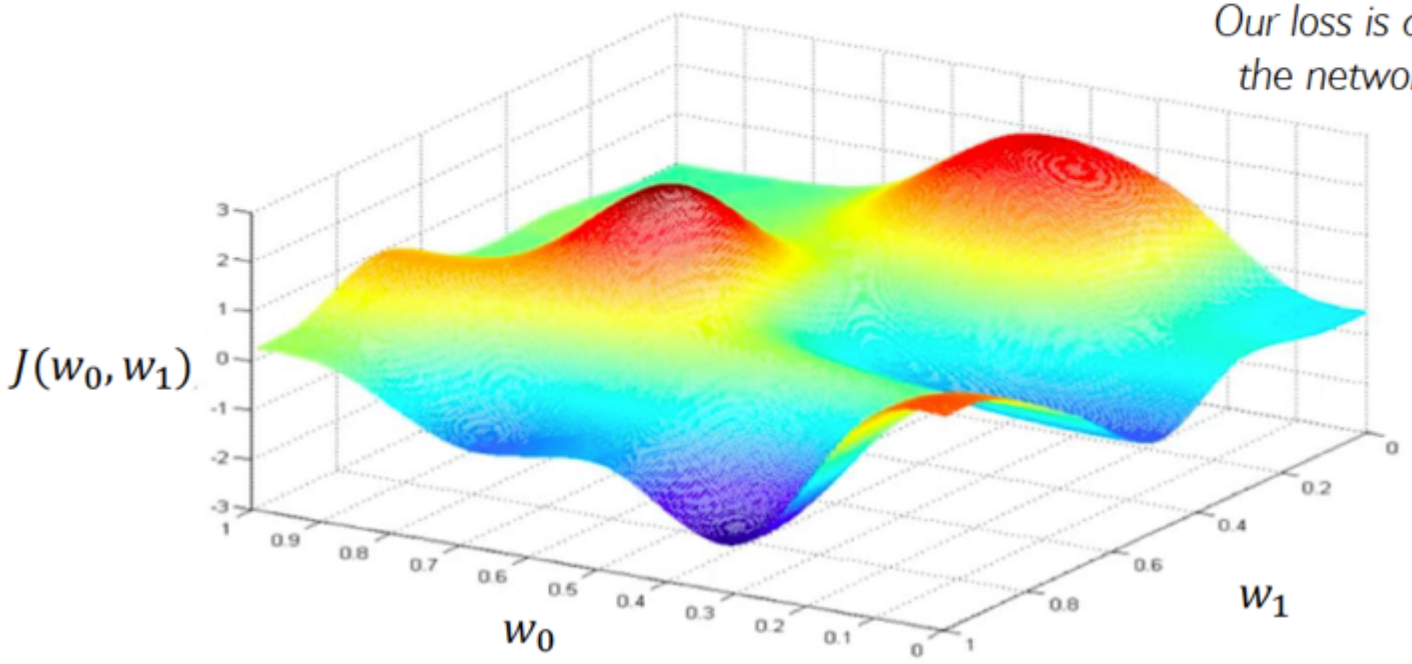
$$\hat{f} = arg \min_{f \in \mathcal{F}} J(f)$$

Source: http://introtodeeplearning.com/

Optimization

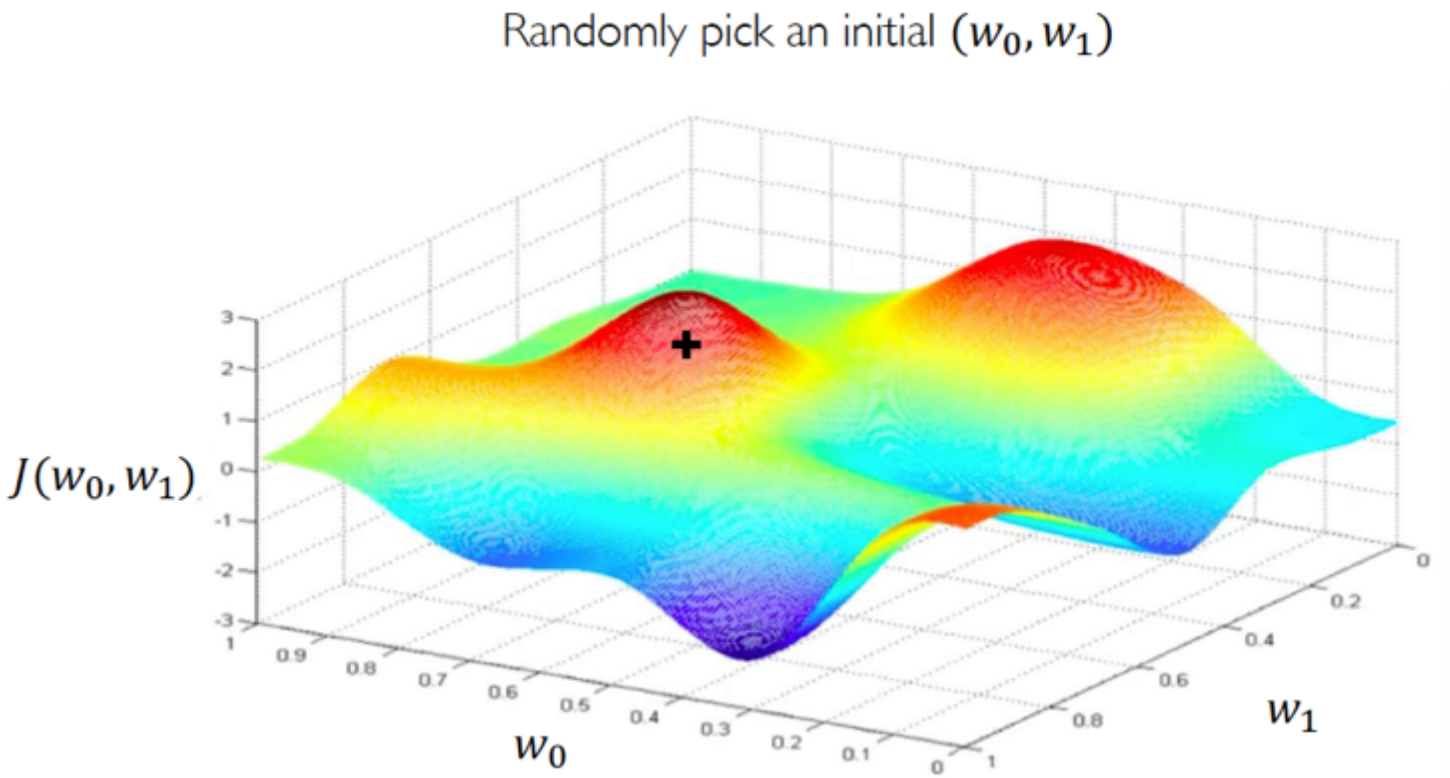
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$

Remember:
Our loss is a function of the network weights!



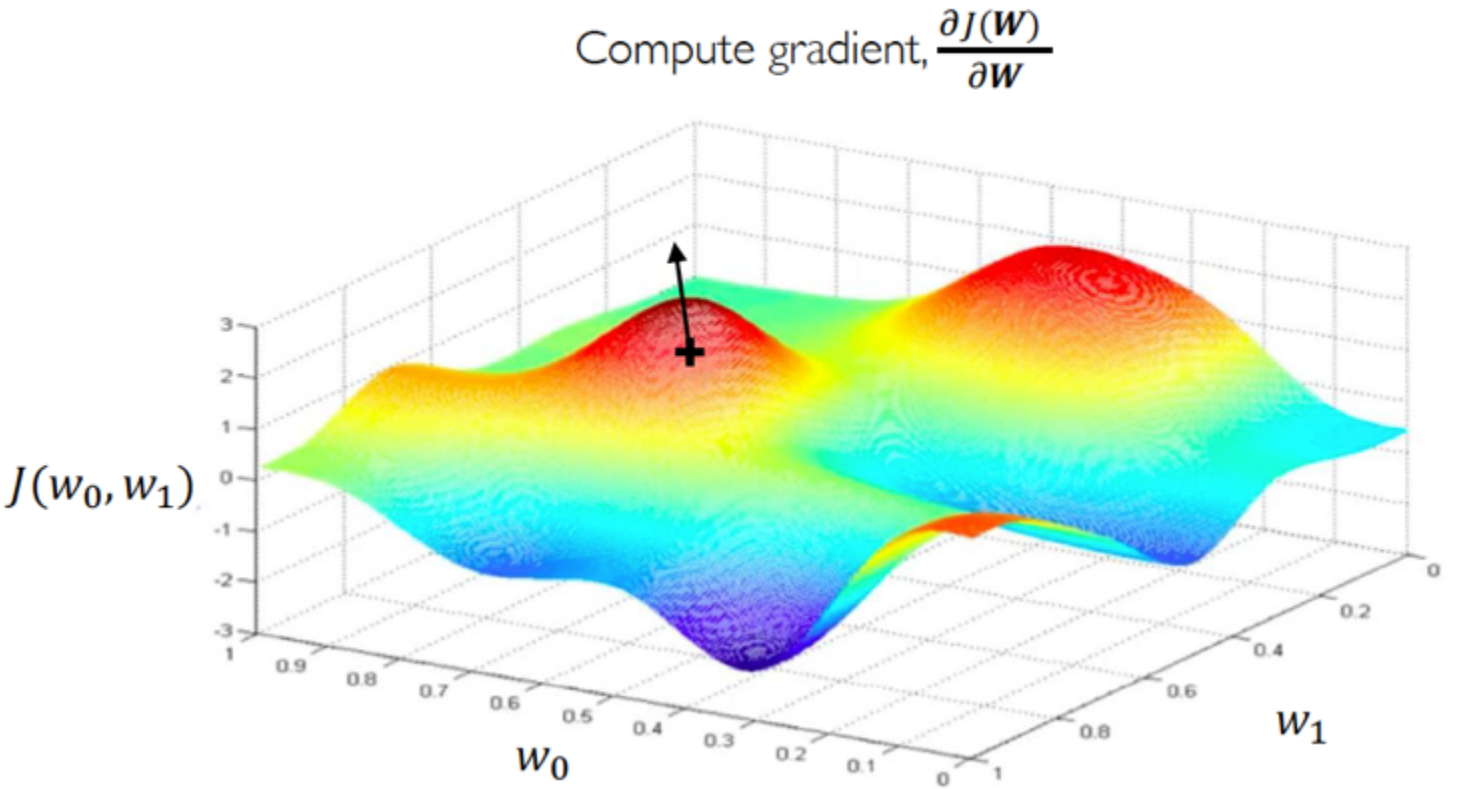
Source: http://introtodeeplearning.com/

Optimization



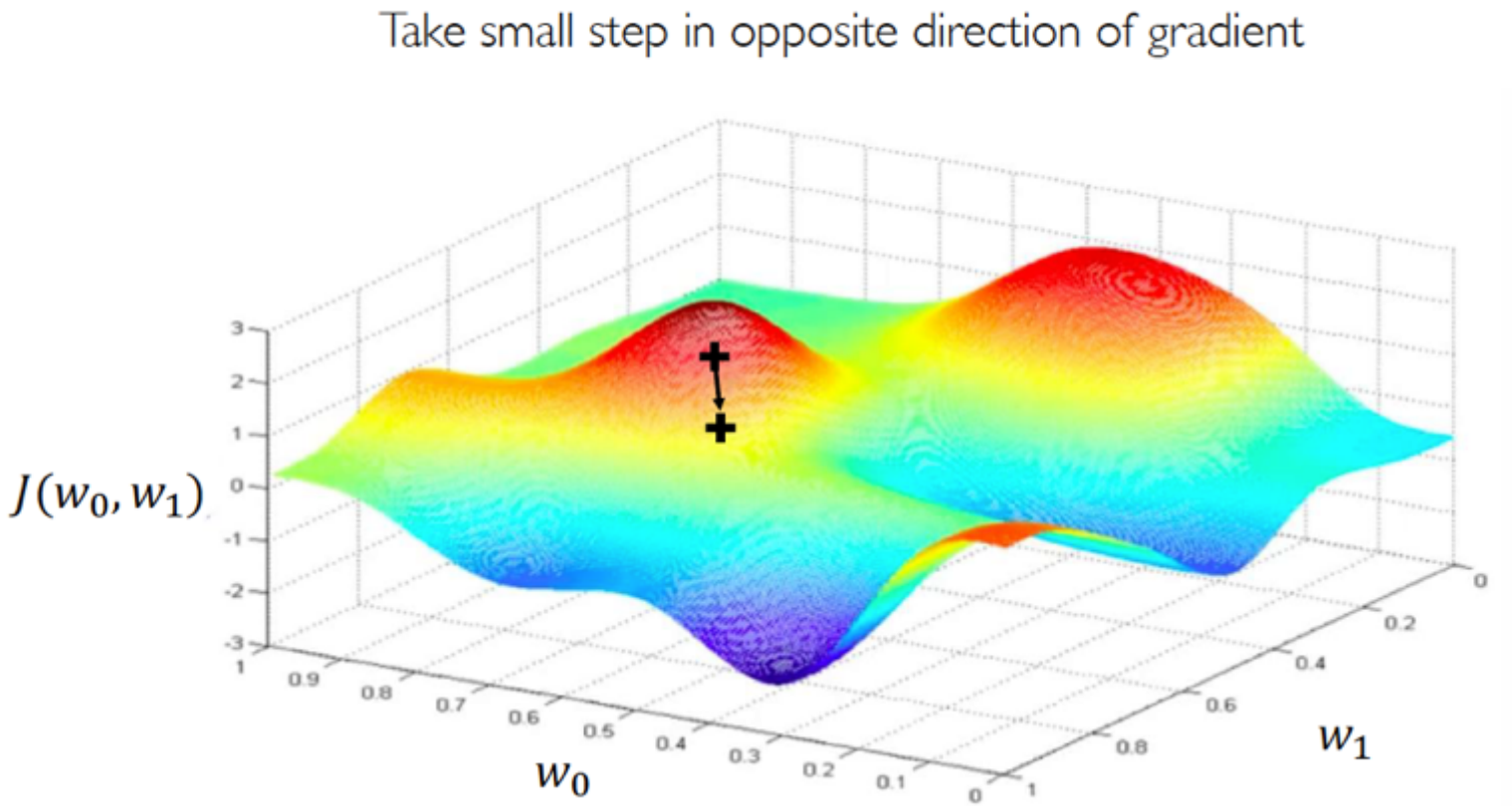
Source: http://introtodeeplearning.com/

Optimization – gradient descent



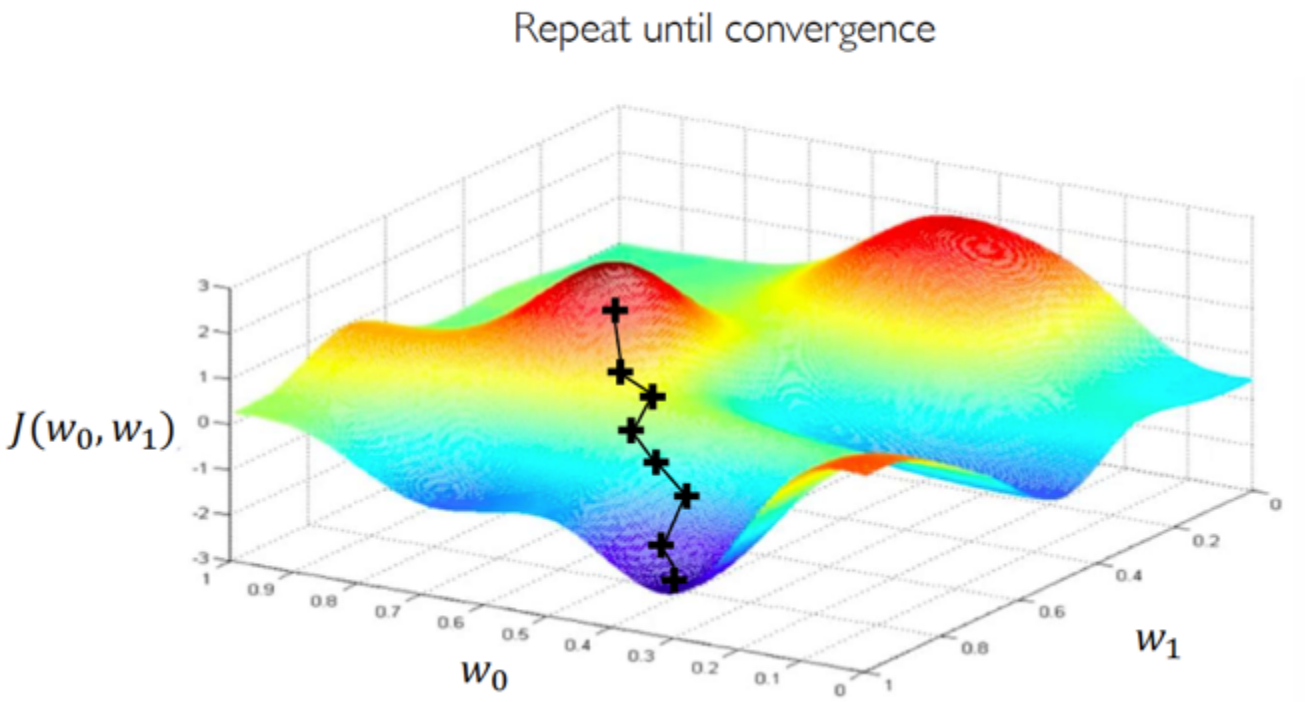
Source: http://introtodeeplearning.com/

Optimization – gradient descent



Source: http://introtodeeplearning.com/

Optimization – gradient descent

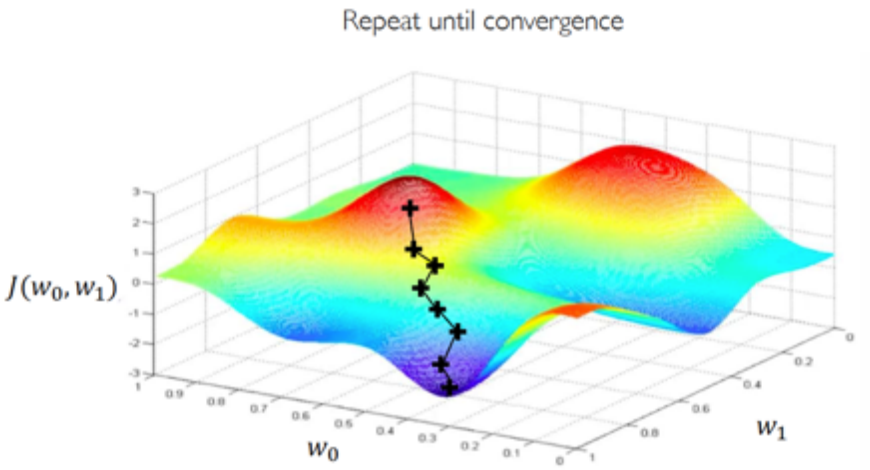


Source: http://introtodeeplearning.com/

Optimization – gradient descent

Algorithm

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



Source: http://introtodeeplearning.com/

Optimization - backpropagation

Source: http://introtodeeplearning.com/

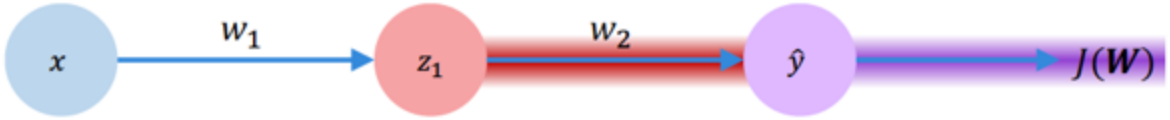
Optimization - backpropagation

$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} =$$

Let's use the chain rule!

Source: http://introtodeeplearning.com/

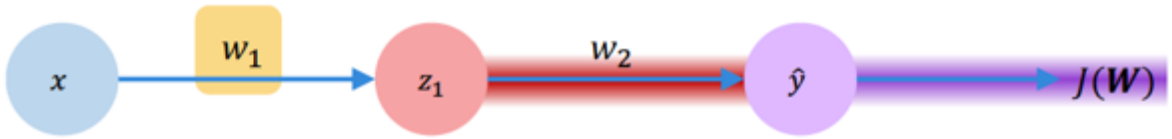
Optimization - backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

Source: http://introtodeeplearning.com/

Optimization - backpropagation



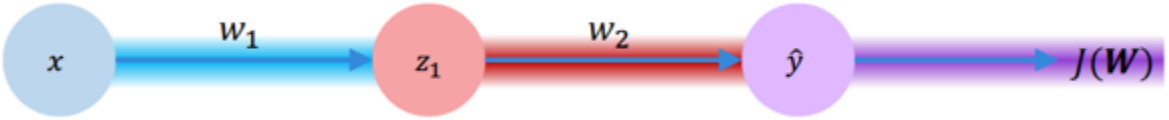
$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

Apply chain rule!

Source: http://introtodeeplearning.com/

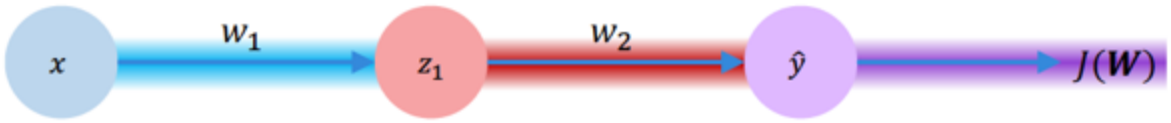
Optimization - backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Source: http://introtodeeplearning.com/

Optimization - backpropagation

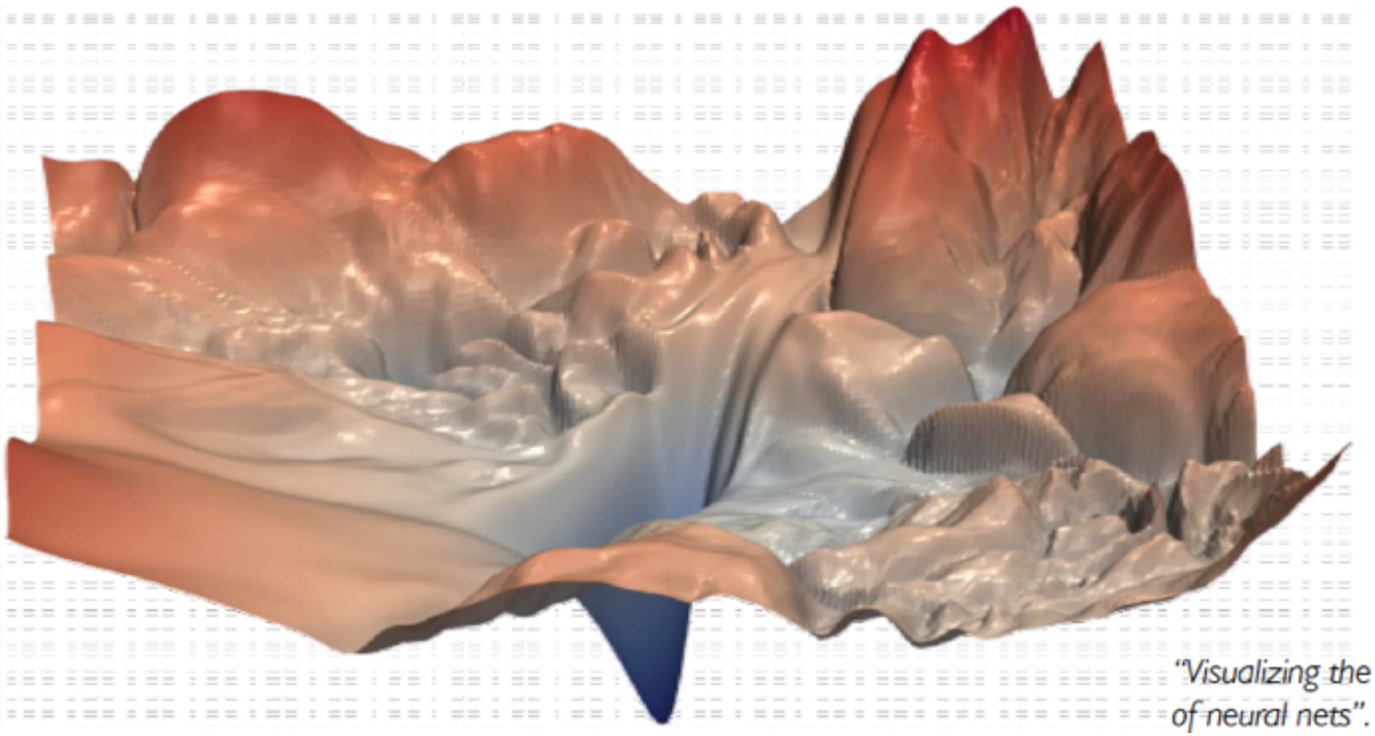


$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for **every weight in the network** *using gradients from later layers*

Source: http://introtodeeplearning.com/

Training is difficult



"Visualizing the loss landscape of neural nets". Dec 2017.

Source: http://introtodeeplearning.com/

Learning rate

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

How can we set the learning rate?

- If η is too small: slow to converge, may be trapped in local minima
- If η is too large: may diverge

Source: http://introtodeeplearning.com/

Adaptive learning rates

- Learning rates are no longer fixed
- Can be made larger or smaller depending on:
  - how large gradient is
  - how fast learning is happening
  - size of particular weights
  - etc...

Source: http://introtodeeplearning.com/

Adaptive learning rates

- Momentum
- Adagrad
- Adadelta
- Adam
- RMSProp

Qian et al. "On the momentum term in gradient descent learning algorithms." 1999.

Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.

Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.

Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.

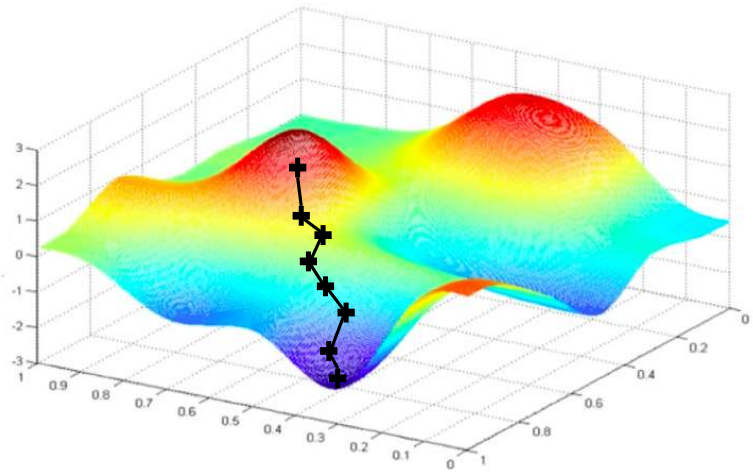
Source: http://introtodeeplearning.com/

Stochastic gradient descent

Classical gradient descent

Algorithm

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



Source: http://introtodeeplearning.com/

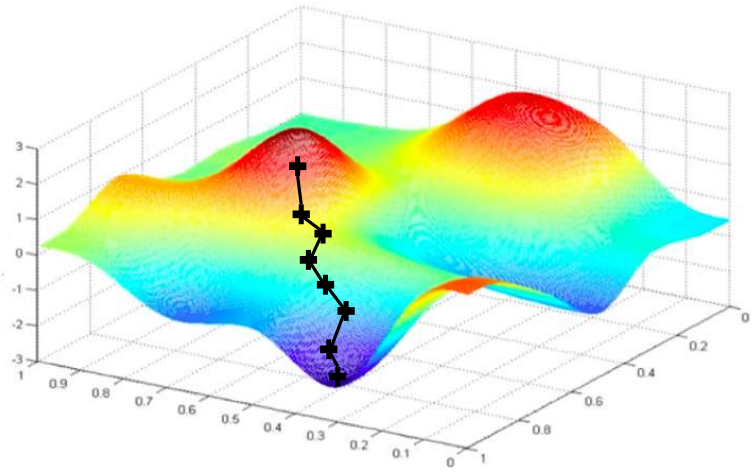
Stochastic gradient descent

Classical gradient descent

Algorithm

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

Can be very computational to compute!



Source: http://introtodeeplearning.com/

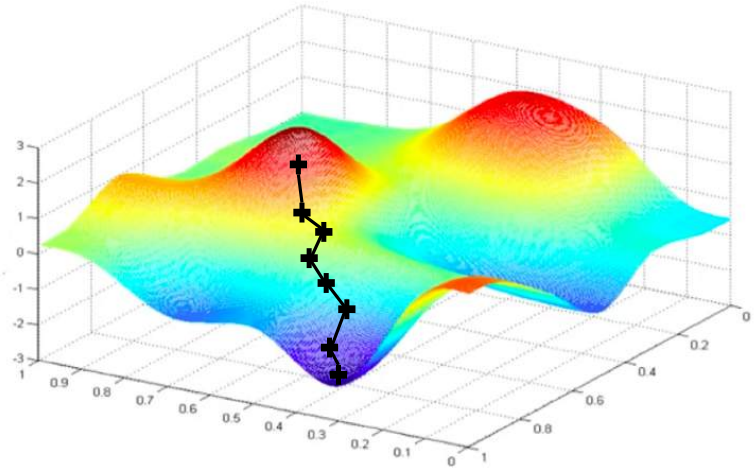
Stochastic gradient descent

Stochastic gradient descent with one sample

Algorithm

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

Easy to compute but **very noisy**



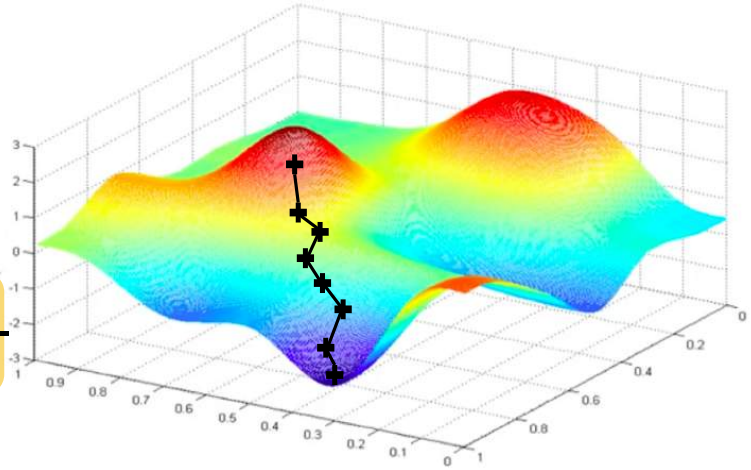
Source: http://introtodeeplearning.com/

Stochastic gradient descent

Stochastic gradient descent with several samples (mini-batch)

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

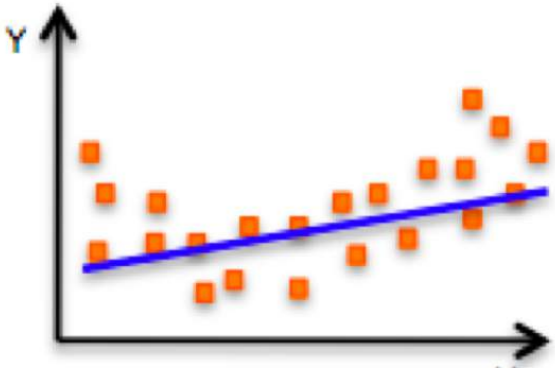


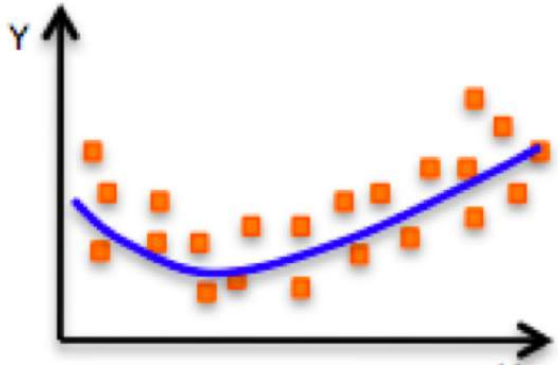
Fast to compute and a much better estimate of the true gradient!

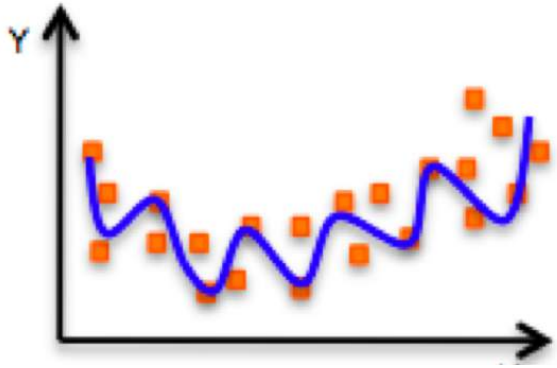
Source: http://introtodeeplearning.com/

Part 2 – Classification and regression
2.6 Overfitting and regularization

The problem of overfitting







Underfitting
Model does not have capacity to fully learn the data

Ideal fit

Overfitting
Too complex, extra parameters, does not generalize well

Source: http://introtodeeplearning.com/

Regularization

What is it?

Technique that constrains our optimization problem to discourage complex models

Source: http://introtodeeplearning.com/

Regularization

Regularization

What is it?

Technique that constrains our optimization problem to discourage complex models

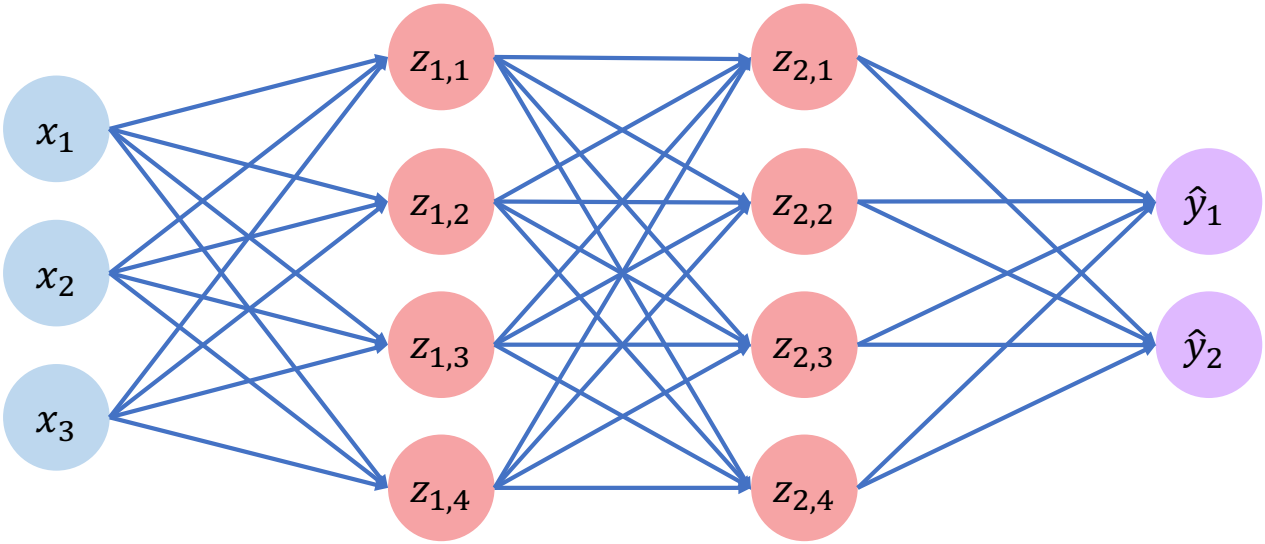
Why do we need it?

Improve generalization of our model on unseen data

Source: http://introtodeeplearning.com/

Regularization: dropout

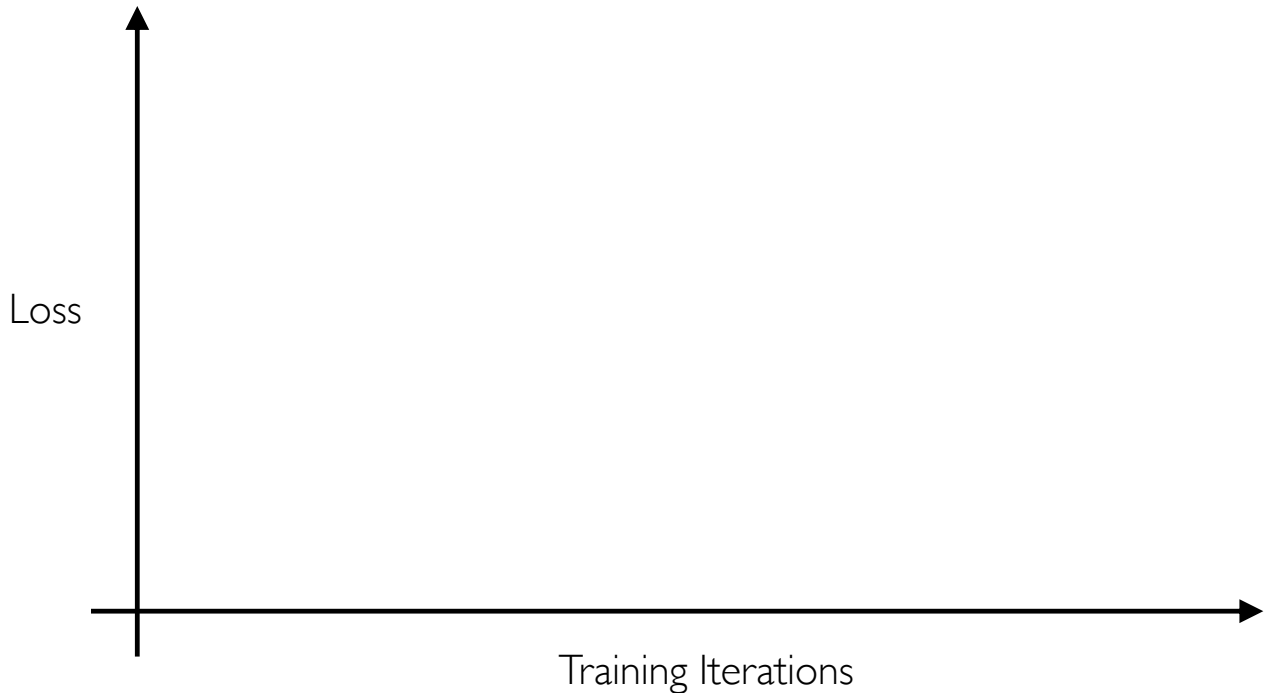
- During training, randomly set some activations to 0



Source: http://introtodeeplearning.com/

Regularization: early stopping

- Stop training before we have a chance to overfit



Source: http://introtodeeplearning.com/

Regularization: early stopping

- Stop training before we have a chance to overfit

Source: http://introtodeeplearning.com/

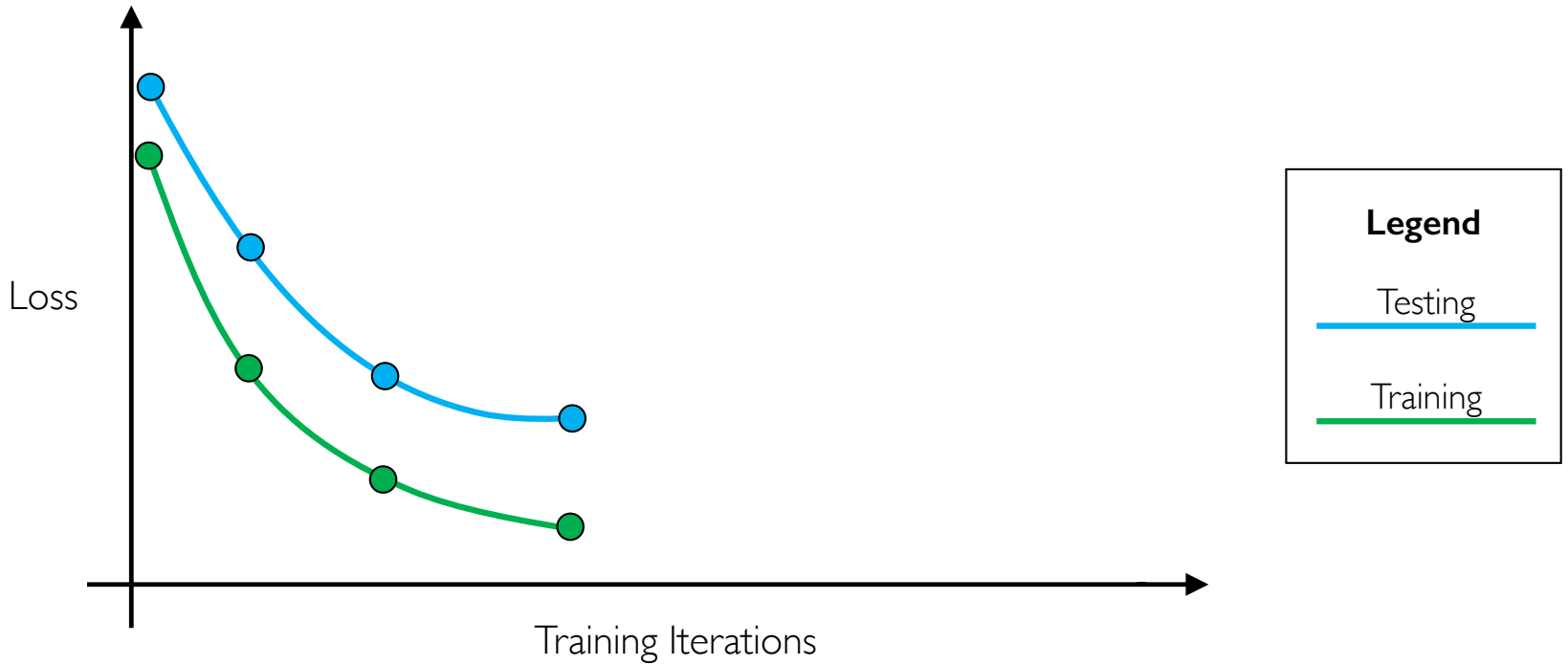
Regularization: early stopping

- Stop training before we have a chance to overfit

Source: http://introtodeeplearning.com/

Regularization: early stopping

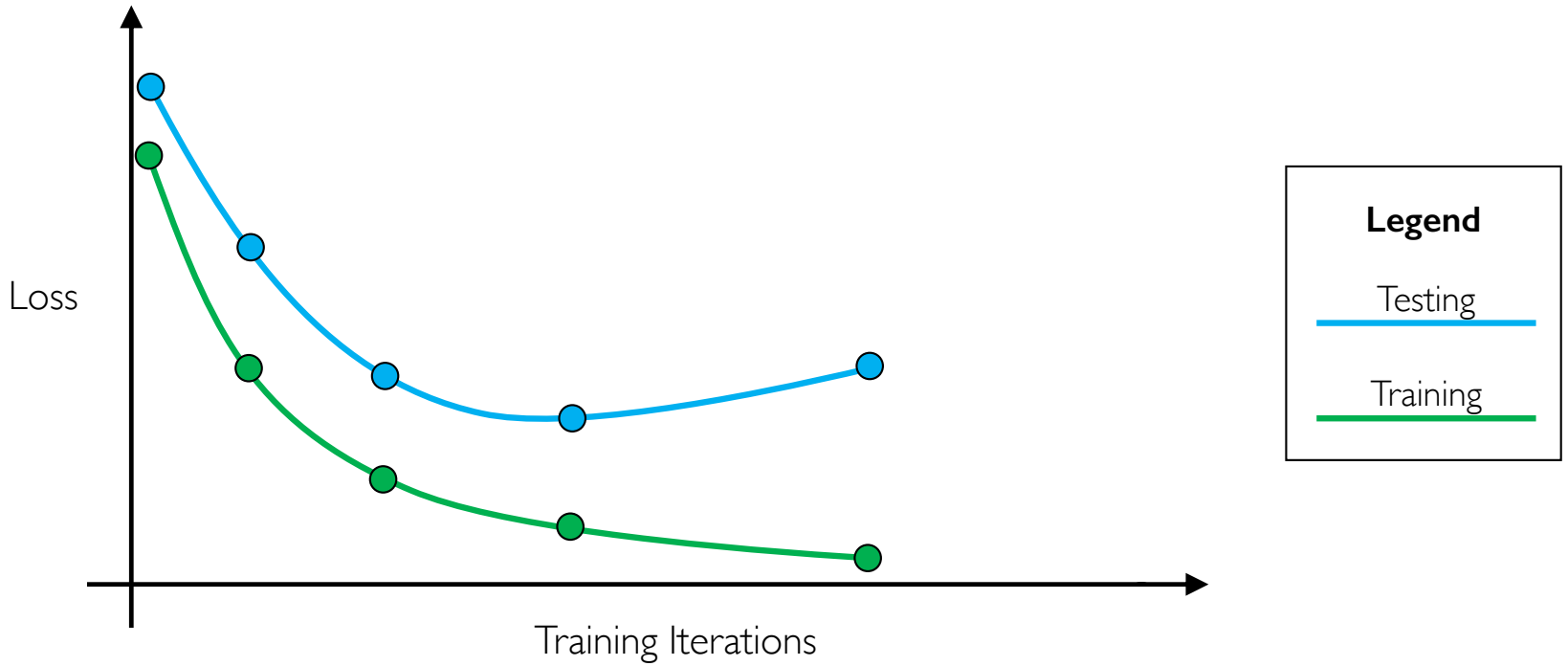
- Stop training before we have a chance to overfit



Source: http://introtodeeplearning.com/

Regularization: early stopping

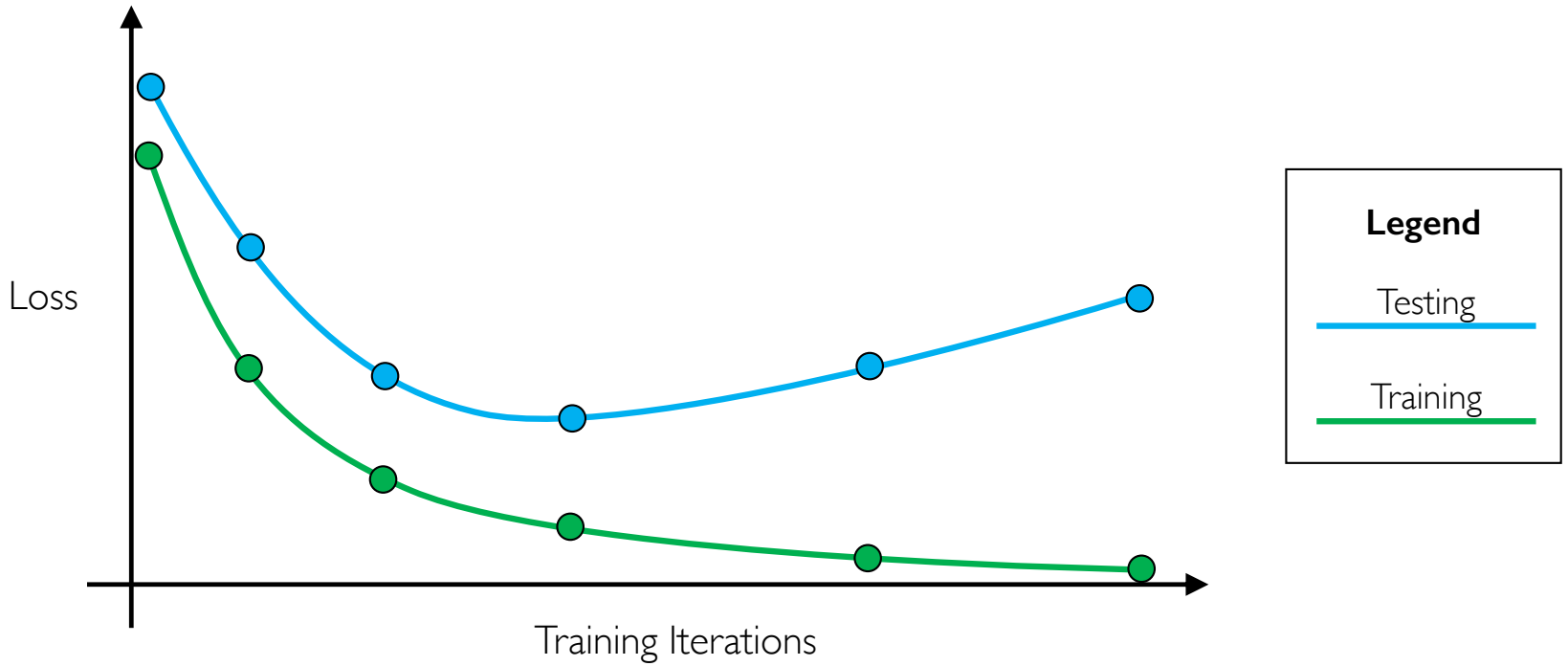
- Stop training before we have a chance to overfit



Source: http://introtodeeplearning.com/

Regularization: early stopping

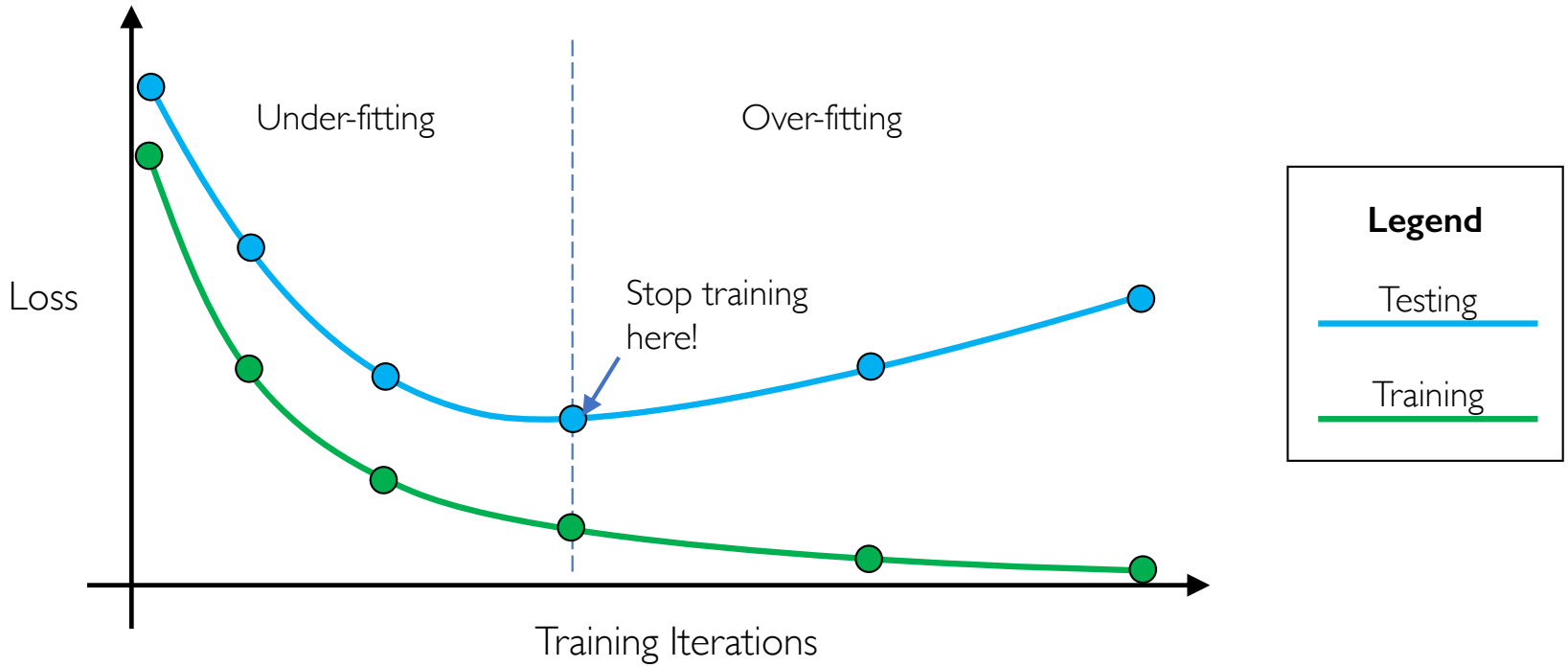
- Stop training before we have a chance to overfit



Source: http://introtodeeplearning.com/

Regularization: early stopping

- Stop training before we have a chance to overfit



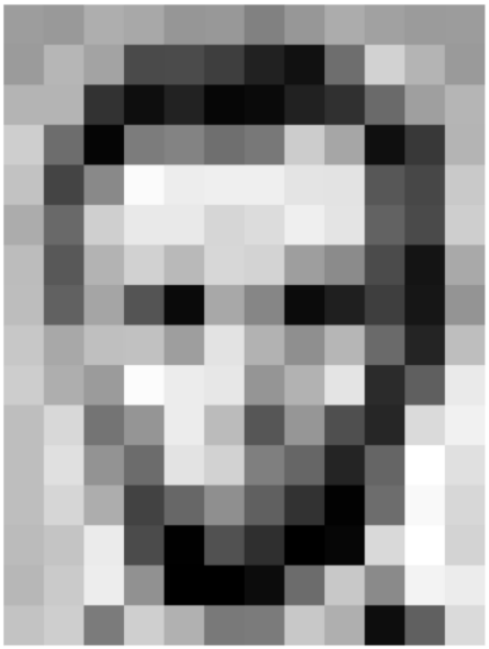
Source: http://introtodeeplearning.com/

Part 2 – Classification and regression
2.6 CNNs

CNN

- Specific type of neural network
  - Well adapted to image data
- State-of-the-art for many image analysis tasks
- Inspired from biological vision (visual cortex)

Images



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

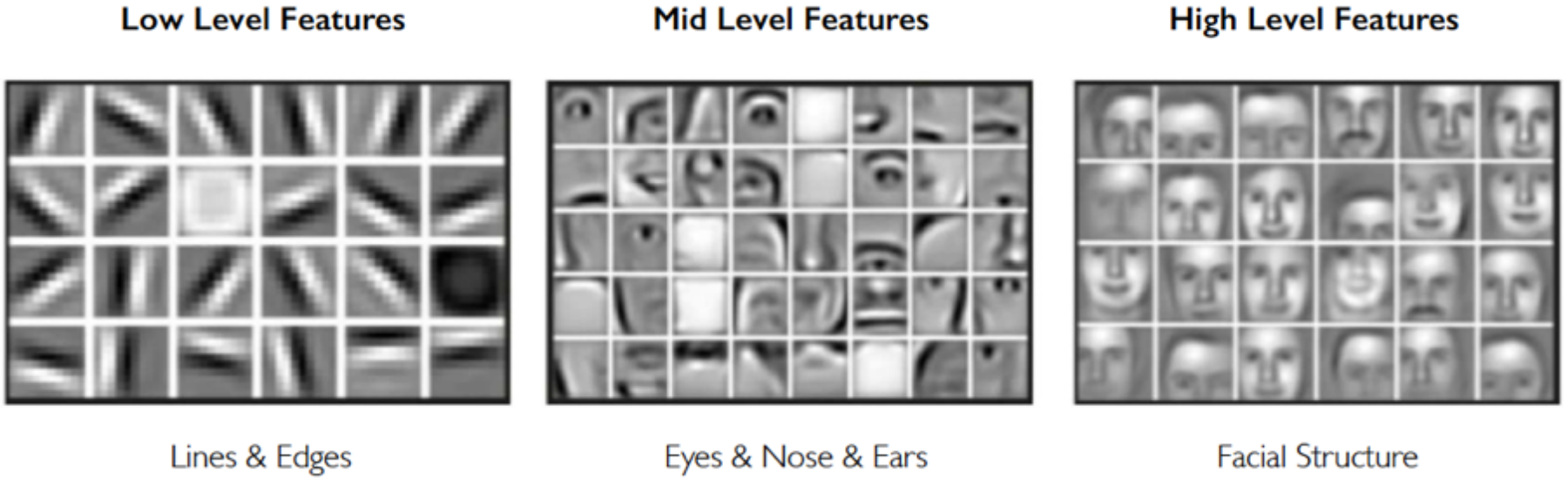
What the computer sees

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

An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image

Source: http://introtodeeplearning.com/

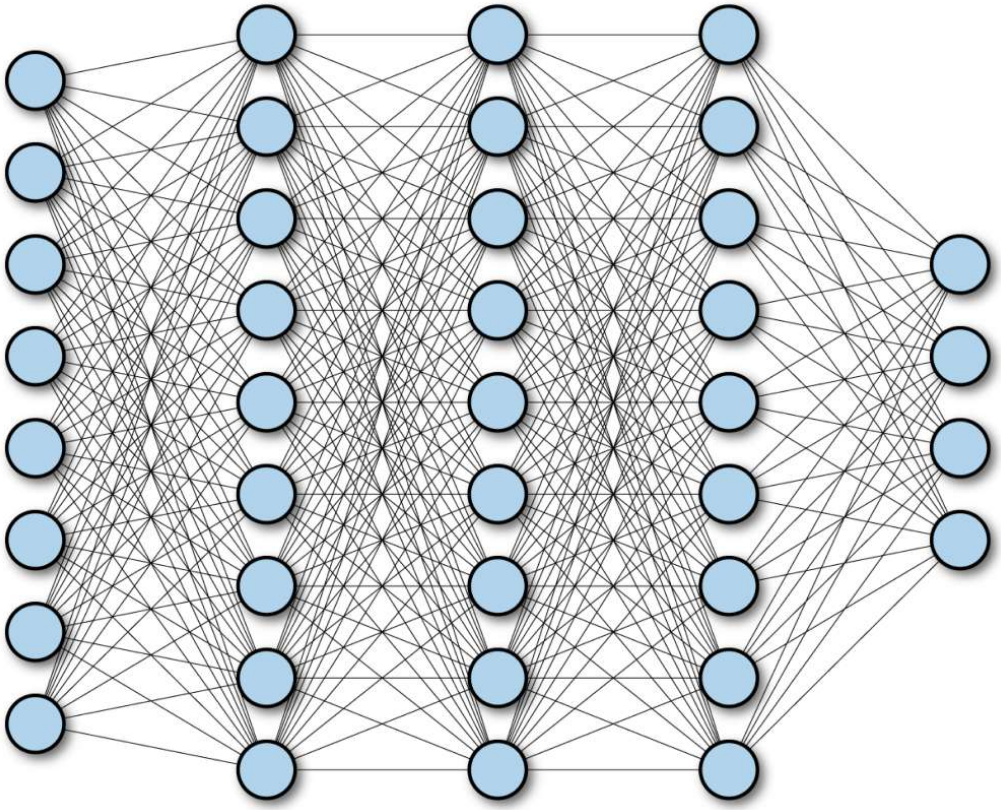
Learning visual features



Source: http://introtodeeplearning.com/

Fully connected networks

What we have seen so far



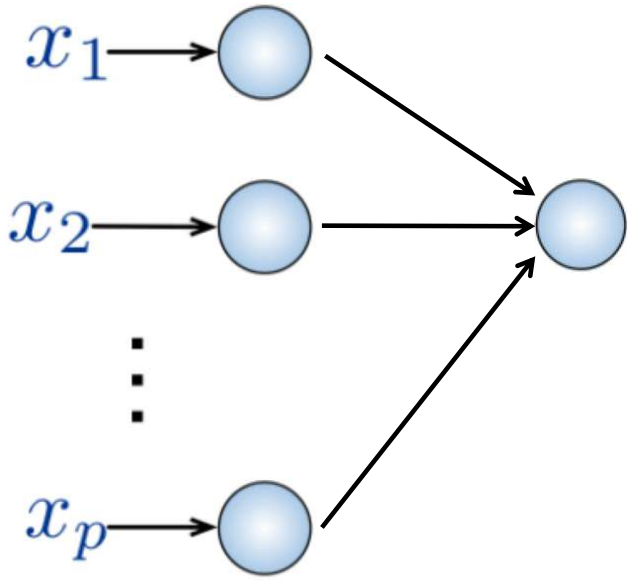
Source: http://introtodeeplearning.com/

Fully connected networks

What we have seen so far

Input:

- 2D image
- Vector of pixel values



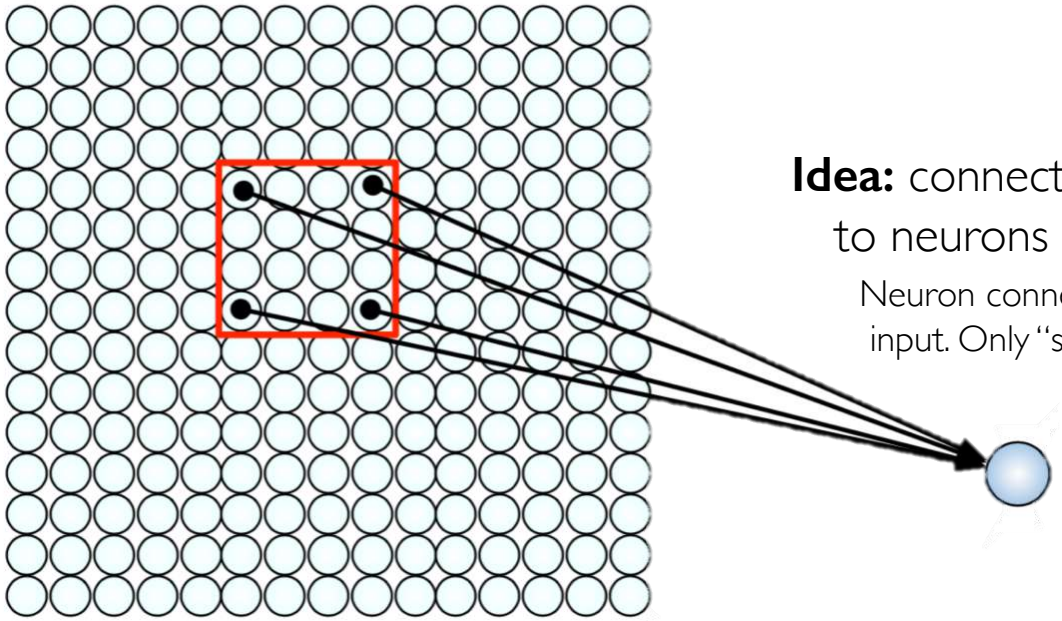
Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

Source: http://introtodeeplearning.com/

CNN

Input: 2D image.
Array of pixel values

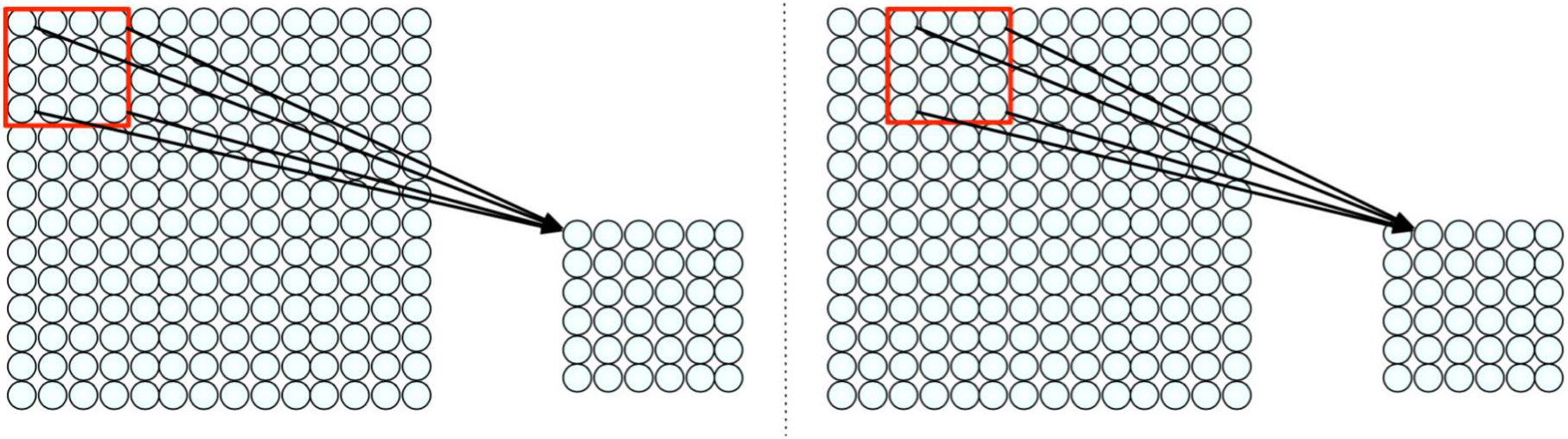


Idea: connect patches of input to neurons in hidden layer.

Neuron connected to region of input. Only "sees" these values.

Source: http://introtodeeplearning.com/

CNN



Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.
How can we **weight** *the patch to detect particular features?*

Source: http://introtodeeplearning.com/

CNN

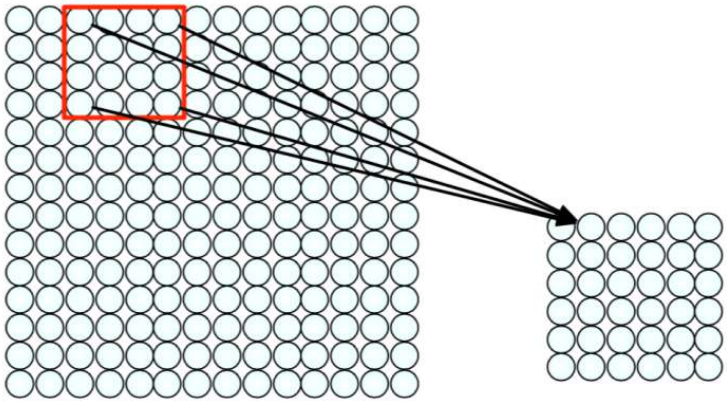
1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) Spatially **share** parameters of each filter
(features that matter in one part of the input should matter elsewhere)

Source: http://introtodeeplearning.com/

CNN



- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

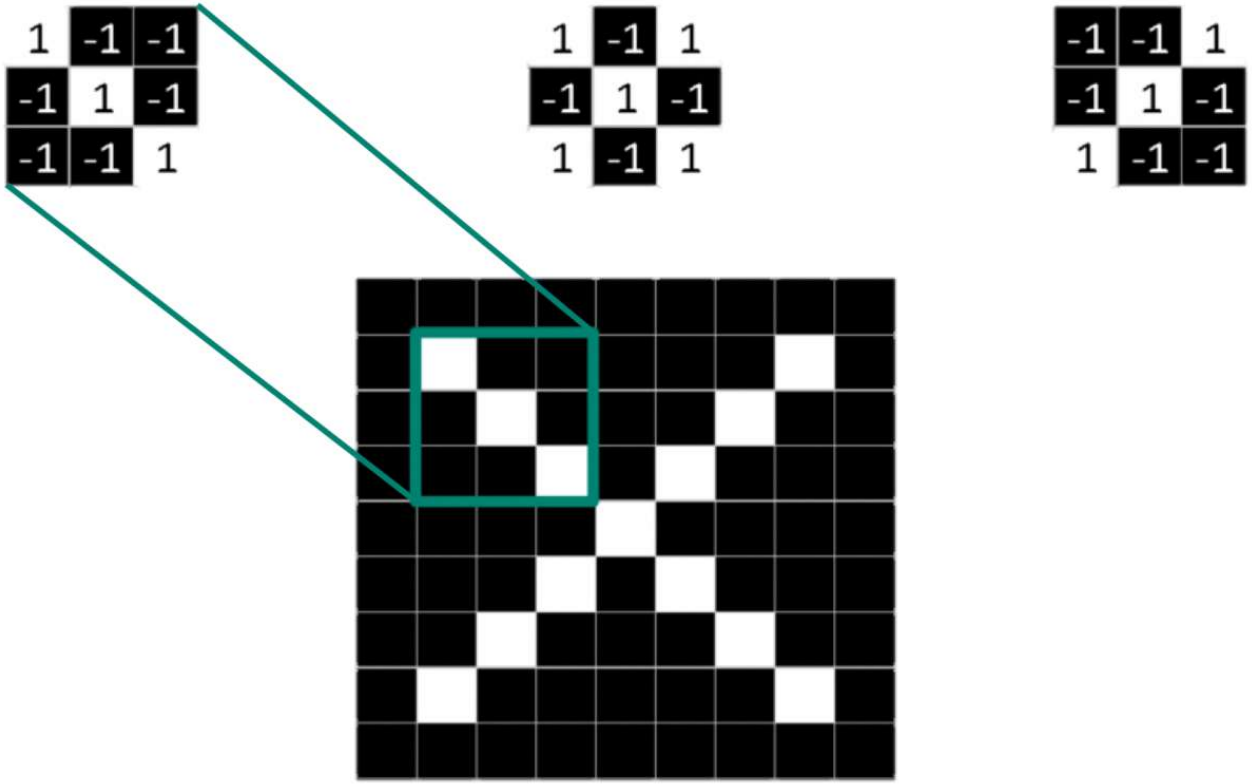
2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

Source: http://introtodeeplearning.com/

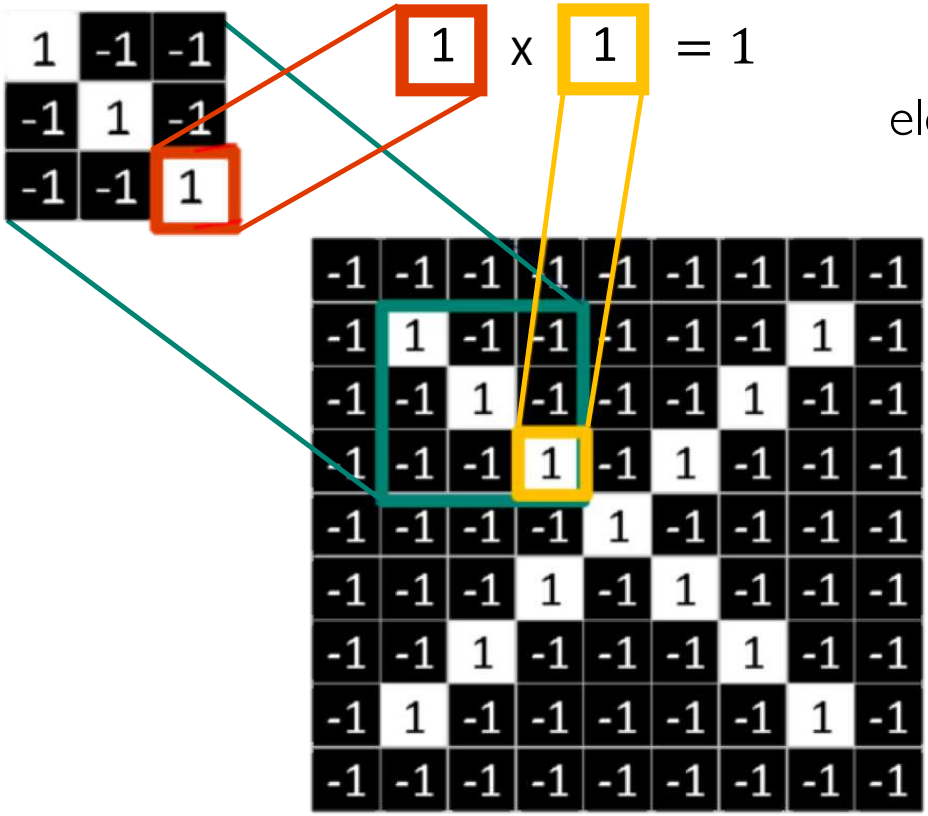
Filters to Detect X Features

filters



Source: http://introtodeeplearning.com/

The Convolution Operation



element wise multiply

add outputs

$$\odot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = 9$$

Source: http://introtodeeplearning.com/

Producing Feature Maps

Original

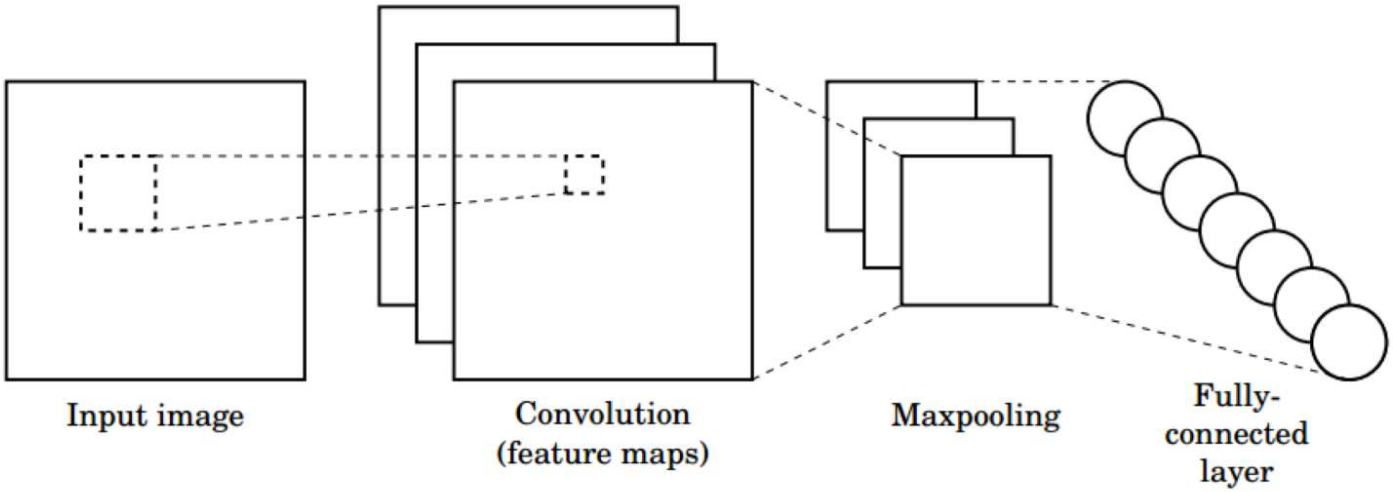
Sharpen

Edge Detect

"Strong" Edge Detect

Source: http://introtodeeplearning.com/

CNNs for Classification

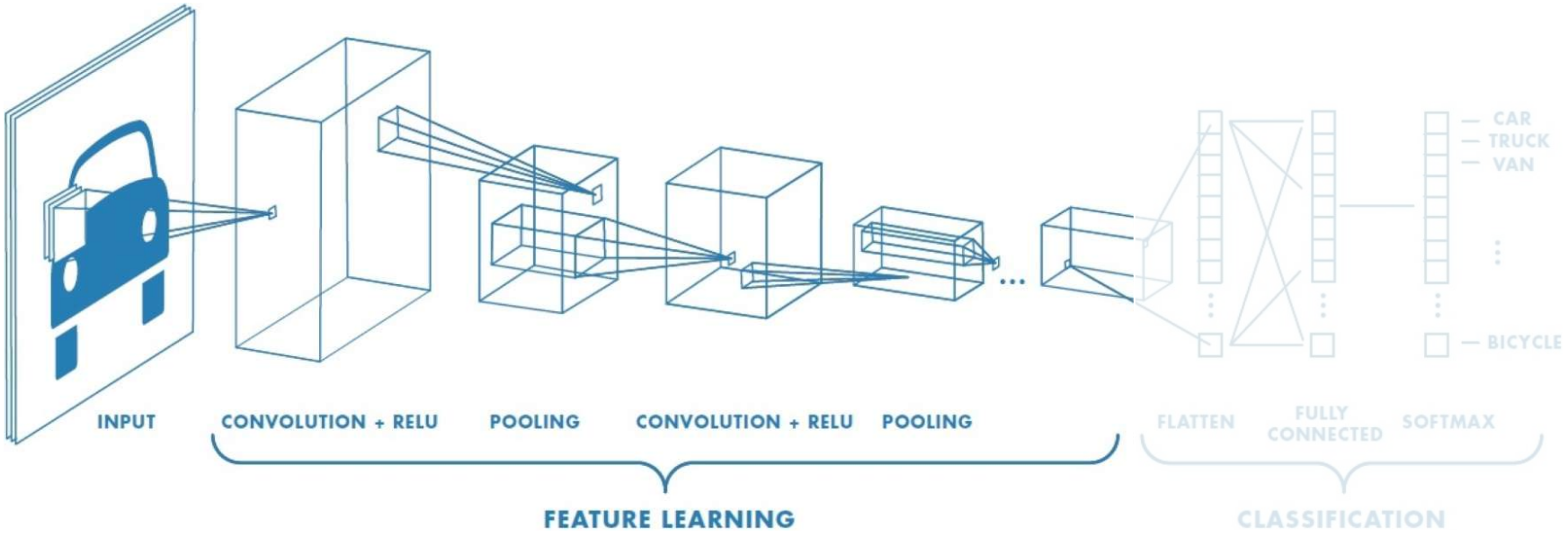


1. **Convolution**: Apply filters with learned weights to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

Train model with image data.
Learn weights of filters in convolutional layers.

Source: http://introtodeeplearning.com/

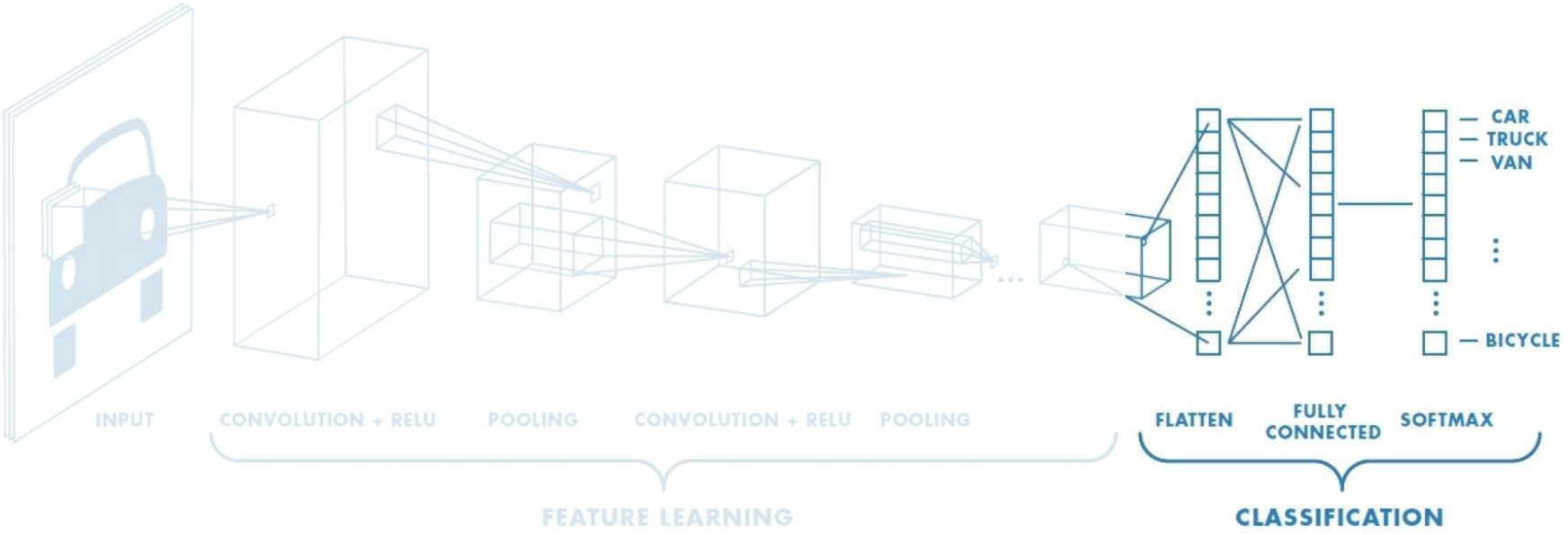
CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

Source: http://introtodeeplearning.com/

CNNs for Classification: Class Probabilities



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

Source: http://introtodeeplearning.com/

Part 2 – Classification and regression
2.7 Examples in medical imaging

Classification of chest radiographs

RESEARCH ARTICLE

Deep learning for chest radiograph diagnosis: A retrospective comparison of the CheXNeXt algorithm to practicing radiologists

Pranav Rajpurkar[1‡*], Jeremy Irvin[1‡], Robyn L. Ball[2], Kaylie Zhu[1], Brandon Yang[1], Hershel Mehta[1], Tony Duan[1], Daisy Ding[1], Aarti Bagul[1], Curtis P. Langlotz[3], Bhavik N. Patel[3], Kristen W. Yeom[3], Katie Shpanskaya[3], Francis G. Blankenberg[3], Jayne Seekins[3], Timothy J. Amrhein[4], David A. Mong[5], Safwan S. Halabi[3], Evan J. Zucker[3], Andrew Y. Ng[1☯], Matthew P. Lungren[3☯]

PLOS | MEDICINE

Citation: Rajpurkar P, Irvin J, Ball RL, Zhu K, Yang B, Mehta H, et al. (2018) Deep learning for chest radiograph diagnosis: A retrospective comparison of the CheXNeXt algorithm to practicing radiologists. PLoS Med 15(11): e1002686. https://doi.org/10.1371/journal.pmed.1002686

Classification of chest radiographs

Data

The ChestX-ray14 dataset [14] was used to develop the deep learning algorithm. The dataset is currently the largest public repository of radiographs, containing 112,120 frontal-view (both posteroanterior and anteroposterior) chest radiographs of 30,805 unique patients. Each image in ChestX-ray14 was annotated with up to 14 different thoracic pathology labels that were chosen based on frequency of observation and diagnosis in clinical practice. The labels for each image were obtained using automatic extraction methods on radiology reports, resulting in 14 binary values per image, where 0 indicates the absence of that pathology and 1 denotes the presence (multiple pathologies can be present in each image). We partitioned the dataset into training, tuning, and validation (see S1 Table for statistics of dataset splits used in this study).

The training set was used to optimize network parameters, the tuning set was used to compare and choose networks, and the validation set was used to evaluate CheXNeXt and radiologists. There is no patient overlap among the partitions.

Classification of chest radiographs

Radiologist annotations

A validation set of 420 frontal-view chest radiographs was selected from ChestX-ray14 for radiologist annotation. The set was curated to contain at least 50 cases of each pathology according to the original labels provided in the dataset by randomly sampling examples and iteratively updating the selected examples by sampling from the examples labeled with the underrepresented pathologies. The radiographs in the validation set were annotated by 3 independent board-certified cardiothoracic specialist radiologists (average experience 15 years, range 5–28 years) for the presence of each of the 14 pathologies. The majority vote of their annotations was taken as a consensus reference standard on each image. To compare to the algorithm, 6 board-certified radiologists from 3 academic institutions (average experience 12 years, range 4–28 years) and 3 senior radiology residents also annotated the validation set of 420 radiographs for all 14 labels. All radiologists individually reviewed and labeled each of the images using a freely available image viewer with capabilities for picture archiving and communication system features such as zoom, window leveling, and contrast adjustment. Radiologists did not have access to any patient information or knowledge of disease prevalence in the data. Labels were entered into a standardized data entry program, and the total time to complete the review was recorded. The Stanford International Review Board (IRB) approved this study, and all radiologists consented to participate in the labeling process.

Classification of chest radiographs

Algorithm development

The deep learning algorithm, called CheXNeXt, is a neural network trained to concurrently detect the 14 pathologies in frontal-view chest radiographs. Neural networks are functions with many parameters that are structured as a hierarchy of layers to model different levels of abstraction. In this study, the selected architecture was a convolutional neural network, a particular type of neural network that is specially designed to handle image data. By exploiting a parameter sharing receptive field, convolutional neural networks scan over an image to learn features from local structure and aggregate the local features to make a prediction on the full image. The neural network used in this study is a 121-layer DenseNet architecture [15] in which each layer is directly connected to every other layer within a block. For each layer, the feature maps of all preceding layers are used as inputs, and its own feature maps are passed on to all following layers as inputs.

Once specifying the neural network architecture, the parameters are automatically learned from a large amount of data labeled with the presence or absence of each pathology. The learning process consists of iteratively updating the parameters to decrease the prediction error, which is computed by comparing the network's prediction to the known annotations on each image. By performing this procedure using a representative set of images, the resulting network can make predictions on previously unseen frontal-view chest radiographs.

Classification of chest radiographs

Table 1. Radiologists and algorithm AUC with CIs.

Pathology	Radiologists (95% CI)	Algorithm (95% CI)	Algorithm – Radiologists Difference (99.6% CI)[a]	Advantage
Atelectasis	0.808 (0.777 to 0.838)	0.862 (0.825 to 0.895)	0.053 (0.003 to 0.101)	Algorithm
Cardiomegaly	0.888 (0.863 to 0.910)	0.831 (0.790 to 0.870)	–0.057 (–0.113 to –0.007)	Radiologists
Consolidation	0.841 (0.815 to 0.870)	0.893 (0.859 to 0.924)	0.052 (–0.001 to 0.101)	No difference
Edema	0.910 (0.886 to 0.930)	0.924 (0.886 to 0.955)	0.015 (–0.038 to 0.060)	No difference
Effusion	0.900 (0.876 to 0.921)	0.901 (0.868 to 0.930)	0.000 (–0.042 to 0.040)	No difference
Emphysema	0.911 (0.866 to 0.947)	0.704 (0.567 to 0.833)	–0.208 (–0.508 to –0.003)	Radiologists
Fibrosis	0.897 (0.840 to 0.936)	0.806 (0.719 to 0.884)	–0.091 (–0.198 to 0.016)	No difference
Hernia	0.985 (0.974 to 0.991)	0.851 (0.785 to 0.909)	–0.133 (–0.236 to –0.055)	Radiologists
Infiltration	0.734 (0.688 to 0.779)	0.721 (0.651 to 0.786)	–0.013 (–0.107 to 0.067)	No difference
Mass	0.886 (0.856 to 0.913)	0.909 (0.864 to 0.948)	0.024 (–0.041 to 0.080)	No difference
Nodule	0.899 (0.869 to 0.924)	0.894 (0.853 to 0.930)	–0.005 (–0.058 to 0.044)	No difference
Pleural thickening	0.779 (0.740 to 0.809)	0.798 (0.744 to 0.849)	0.019 (–0.056 to 0.094)	No difference
Pneumonia	0.823 (0.779 to 0.856)	0.851 (0.781 to 0.911)	0.028 (–0.087 to 0.125)	No difference
Pneumothorax	0.940 (0.912 to 0.962)	0.944 (0.915 to 0.969)	0.004 (–0.040 to 0.051)	No difference

[a]The AUC difference was calculated as the AUC of the algorithm minus the AUC of the radiologists. To account for multiple hypothesis testing, the Bonferroni-corrected CI (1 – 0.05/14; 99.6%) around the difference was computed.

The nonparametric bootstrap was used to estimate the variability around each of the performance measures; 10,000 bootstrap replicates from the validation set were drawn, and each performance measure was calculated for the algorithm and the radiologists on these same 10,000 bootstrap replicates. This produced a distribution for each estimate, and the 95% bootstrap percentile intervals (2.5th and 97.5th percentiles) are reported.

Abbreviations: AUC, area under the receiver operating characteristic curve; CI, confidence interval.

Classification of chest radiographs

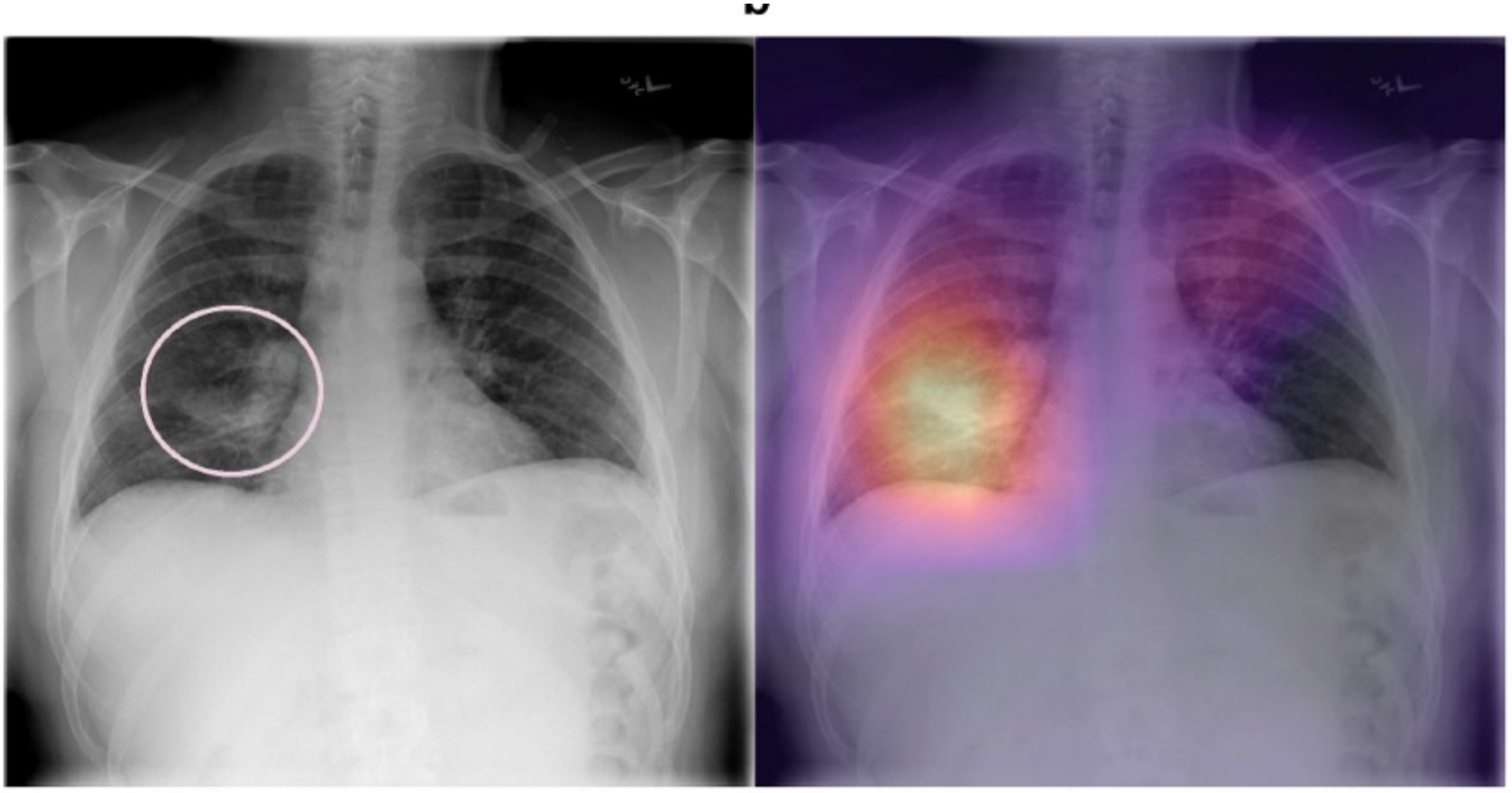
Table 1. Radiologists and algorithm AUC with CIs.

Pathology	Radiologists (95% CI)	Algorithm (95% CI)
Atelectasis	0.808 (0.777 to 0.838)	0.862 (0.825 to 0.895)
Cardiomegaly	0.888 (0.863 to 0.910)	0.831 (0.790 to 0.870)
Consolidation	0.841 (0.815 to 0.870)	0.893 (0.859 to 0.924)
Edema	0.910 (0.886 to 0.930)	0.924 (0.886 to 0.955)
Effusion	0.900 (0.876 to 0.921)	0.901 (0.868 to 0.930)
Emphysema	0.911 (0.866 to 0.947)	0.704 (0.567 to 0.833)

Classification of chest radiographs

Pathology	Advantage
Atelectasis	Algorithm
Cardiomegaly	Radiologists
Consolidation	No difference
Edema	No difference
Effusion	No difference
Emphysema	Radiologists
Fibrosis	No difference
Hernia	Radiologists
Infiltration	No difference
Mass	No difference
Nodule	No difference
Pleural thickening	No difference
Pneumonia	No difference
Pneumothorax	No difference

Classification of chest radiographs

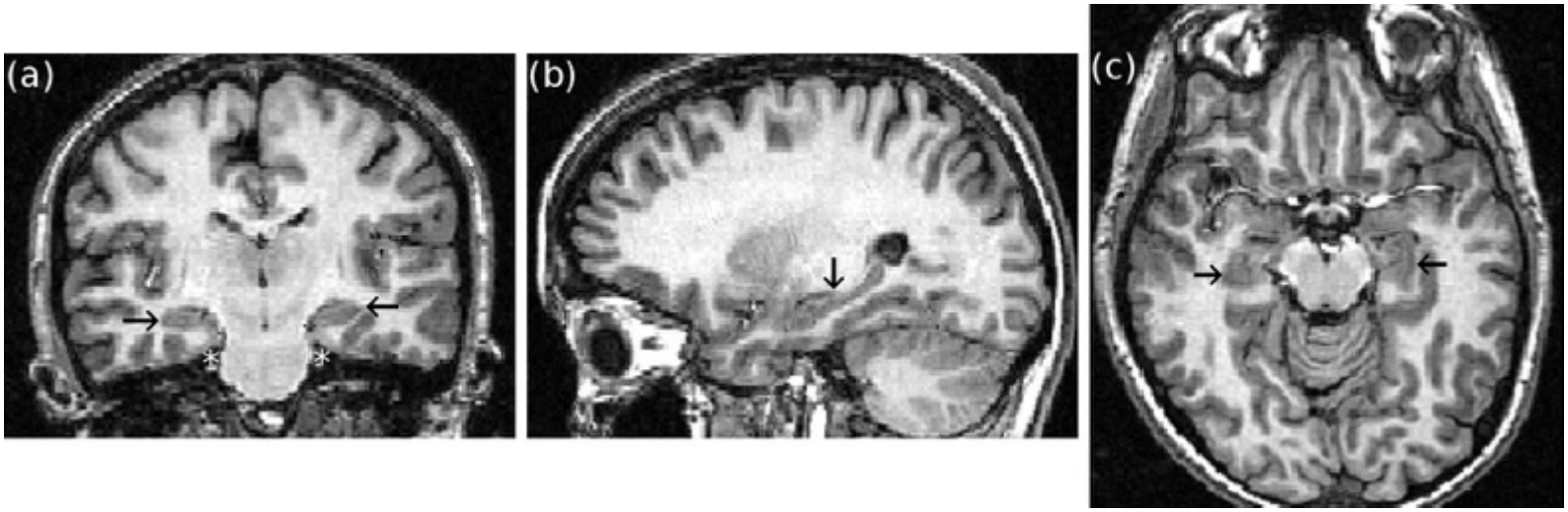


Interpreting network predictions

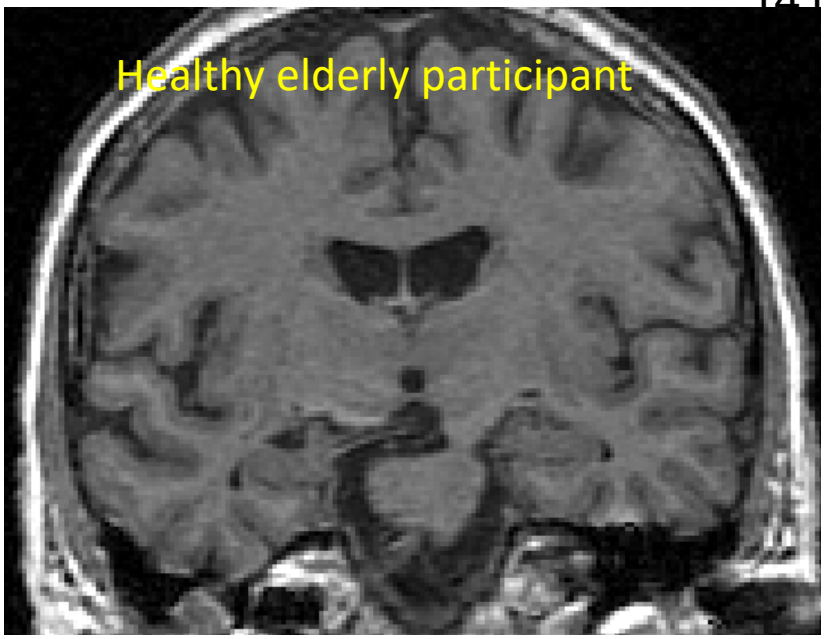
Example in brain image classification

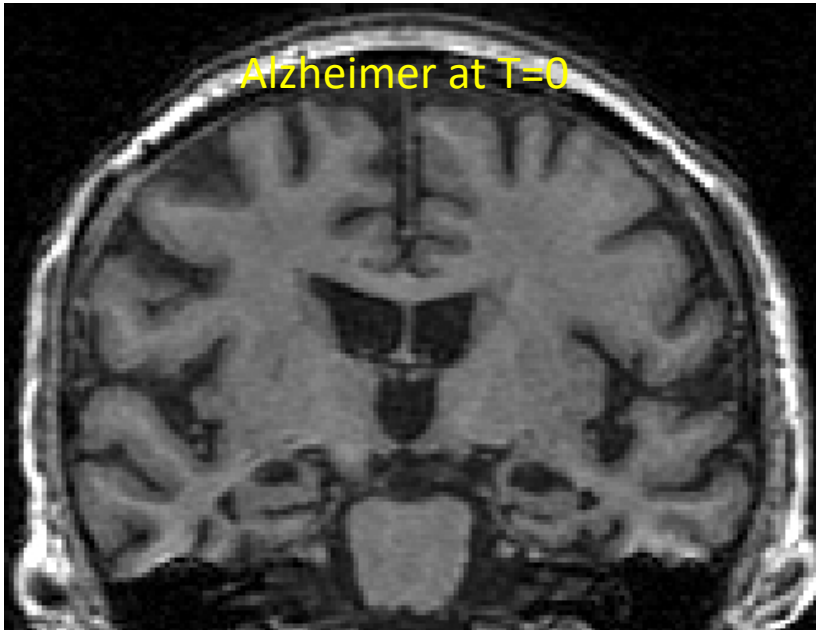
- Alzheimer's disease
  - Neurodegenerative disease
    - Loss of neurons and connections

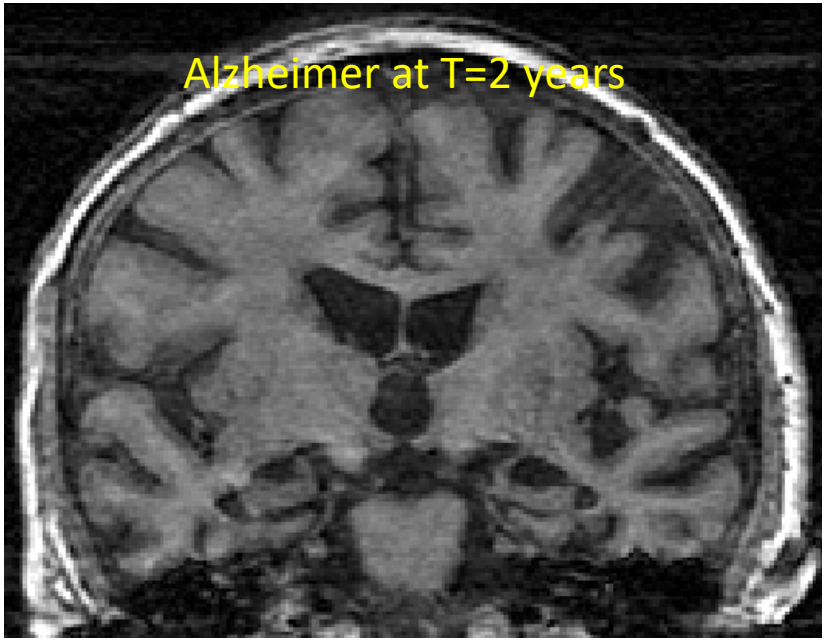
- Anatomical MRI (magnetic resonance imaging)



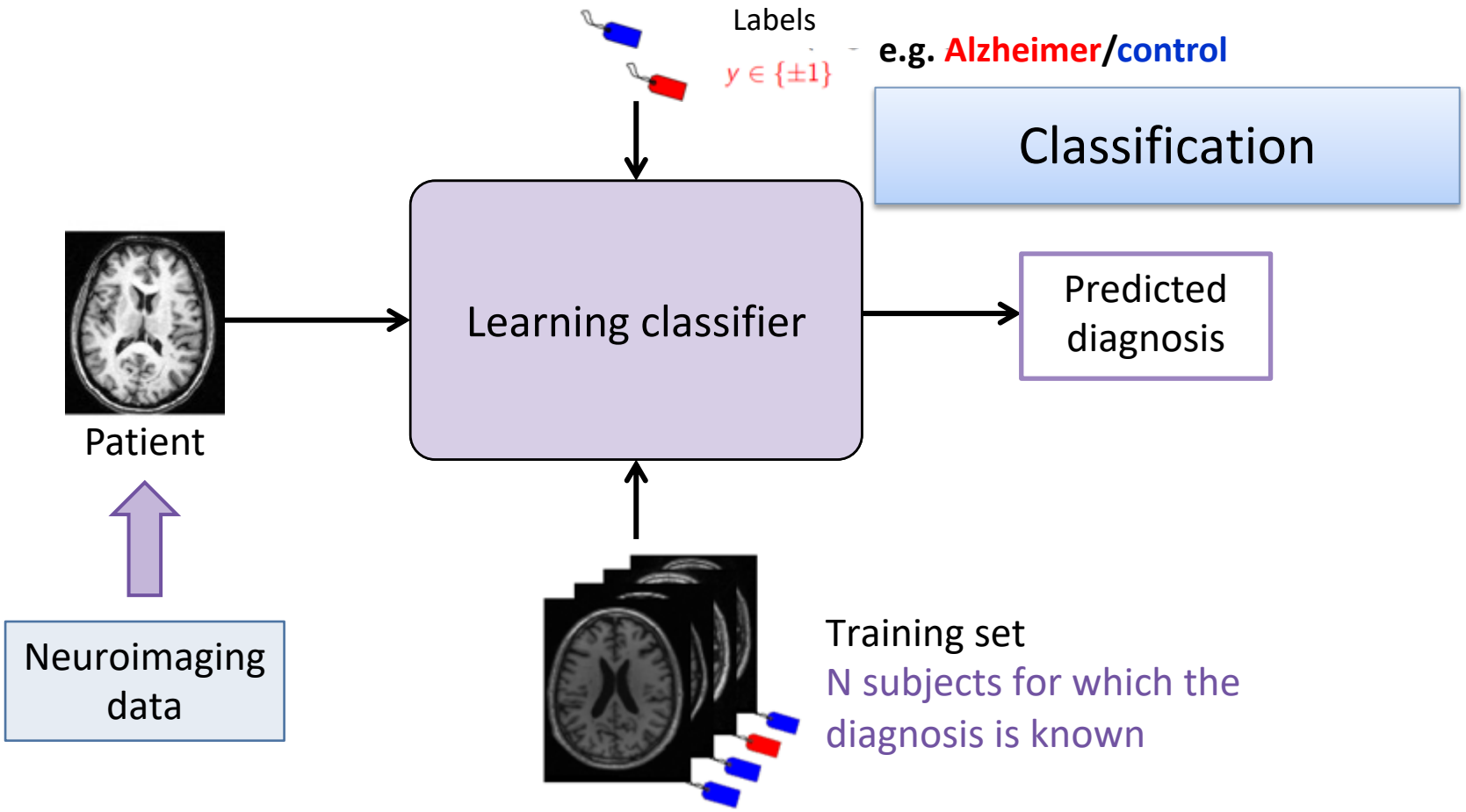
Can we detect that the patient has Alzheimer's disease from MRI?





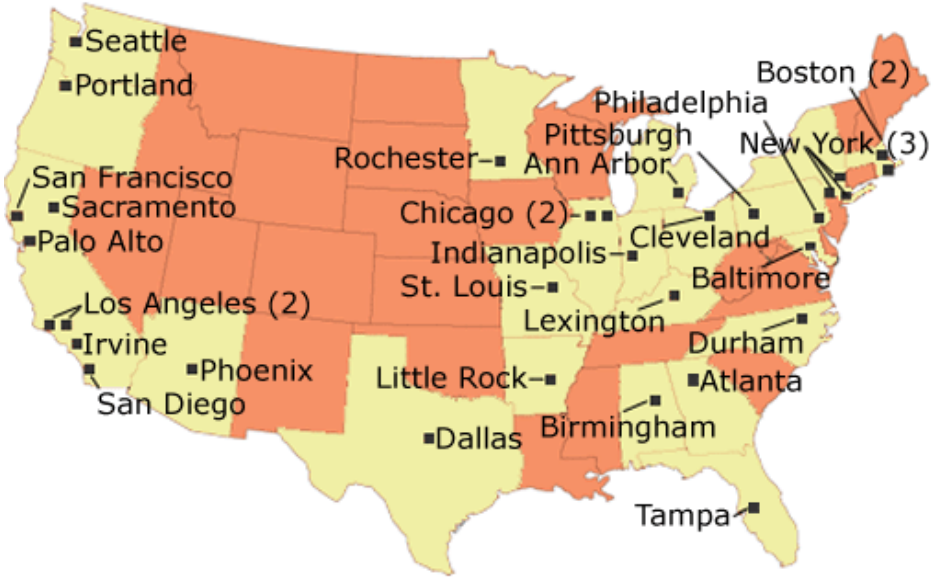


Example in brain image classification



Data

- **Data: ADNI (Alzheimer's Disease NeuroImaging Initiative)**
  - Large multicentric dataset
  - Over 1000 participants
  - Multimodal data: Brain imaging, cognitive scores, biomarkers, genetics
  - Publicly available



Data

Table 2. Summary of participant demographics, mini-mental state examination (MMSE) and global clinical dementia rating (CDR) scores at baseline for ADNI.

	Subjects	Sessions	Age	Gender	MMSE	CDR
CN	330	1 830	74. 4 ± 5.8 [59.8, 89.6]	160 M / 170 F	29.1 ± 1.1 [24, 30]	0: 330
MCI	787	3 458	73.3 ± 7.5 [54.4, 91.4]	464 M / 323 F	27.5 ± 1.8 [23, 30]	0: 2; 0.5: 785
sMCI	298	1 046	72.3 ± 7.4 [55.0, 88.4]	175 M / 123 F	28.0 ± 1.7 [23, 30]	0.5: 298
pMCI	295	865	73.8 ± 6.9 [55.1, 88.3]	176 M / 119 F	26.9 ± 1.7 [23, 30]	0.5: 293; 1: 2
AD	336	1 106	75.0 ± 7.8 [55.1, 90.9]	185 M / 151 F	23.2 ± 2.1 [18, 27]	0.5: 160; 1: 175; 2: 1

Values are presented as mean ± SD [range]. M: male, F: female

3D CNN

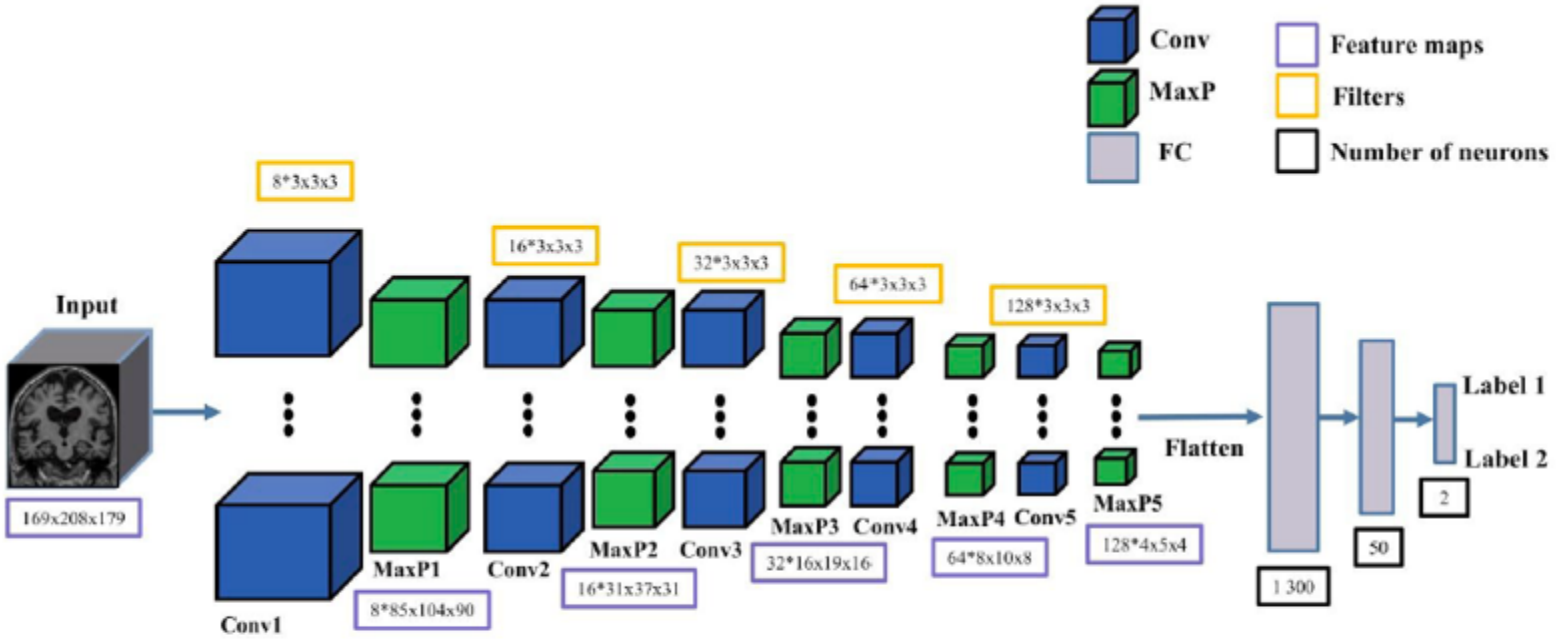


Figure 1: Architecture of the 3D subject-level CNNs. For each convolutional block, we only display the convolutional and max pooling layers. Filters for each convolutional layer represent the number of filters * filter size. Feature maps of each convolutional block represent the number of feature maps * size of each feature map. Conv: convolutional layer; MaxP: max pooling layer; FC: fully connected layer.

3D ROI-based CNN

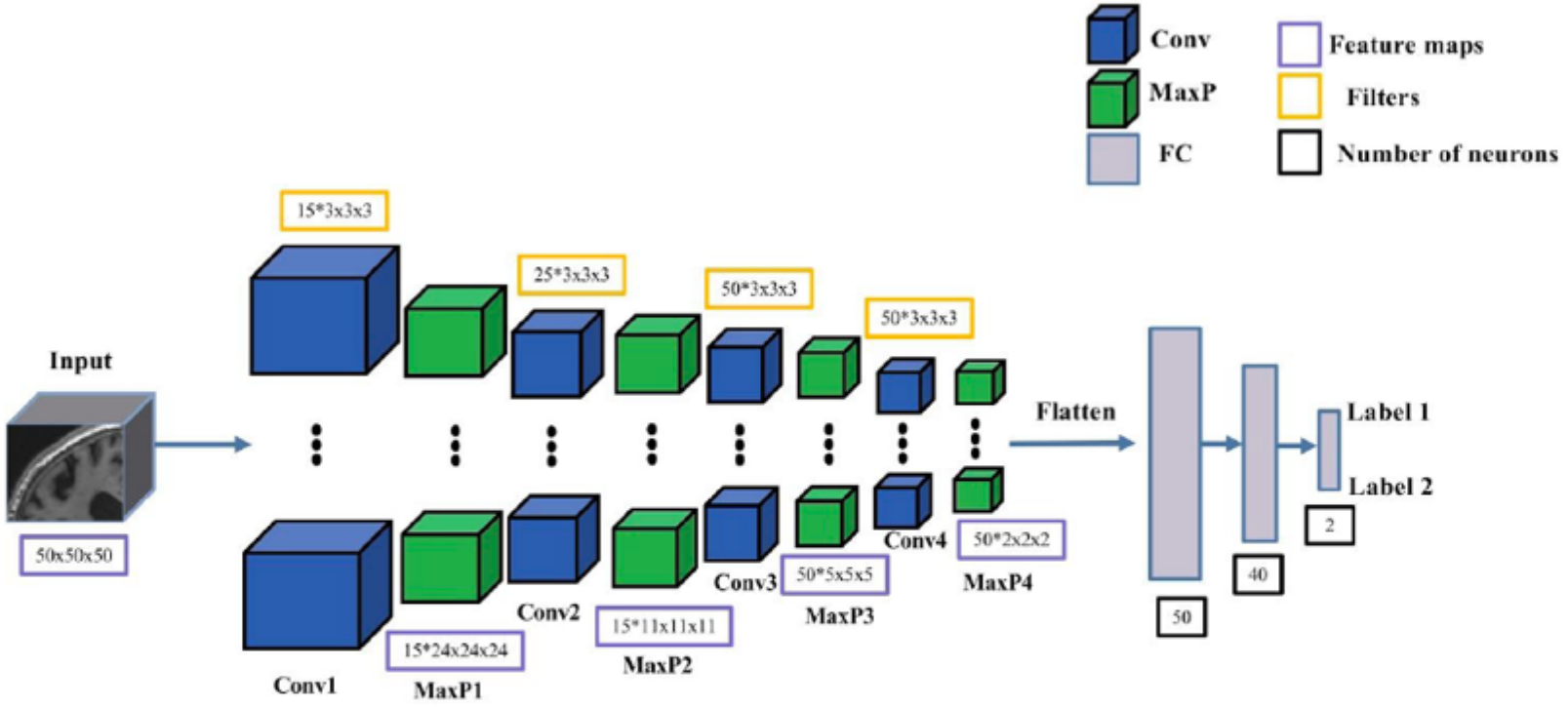


Figure 2: Architecture of the 3D ROI-based and 3D patch-level CNNs. For each convolutional block, we only display the convolutional and max pooling layers. Filters for each convolutional layer represent the number of filters * filter size. Feature maps of each convolutional block represent the number of feature maps * size of each feature map. Conv: convolutional layer; MaxP: max pooling layer; FC: fully connected layer.

2D slice-level CNN (ResNet)

Pretrained on ImageNet

Fine-tuned o Alzheimer's MRI data

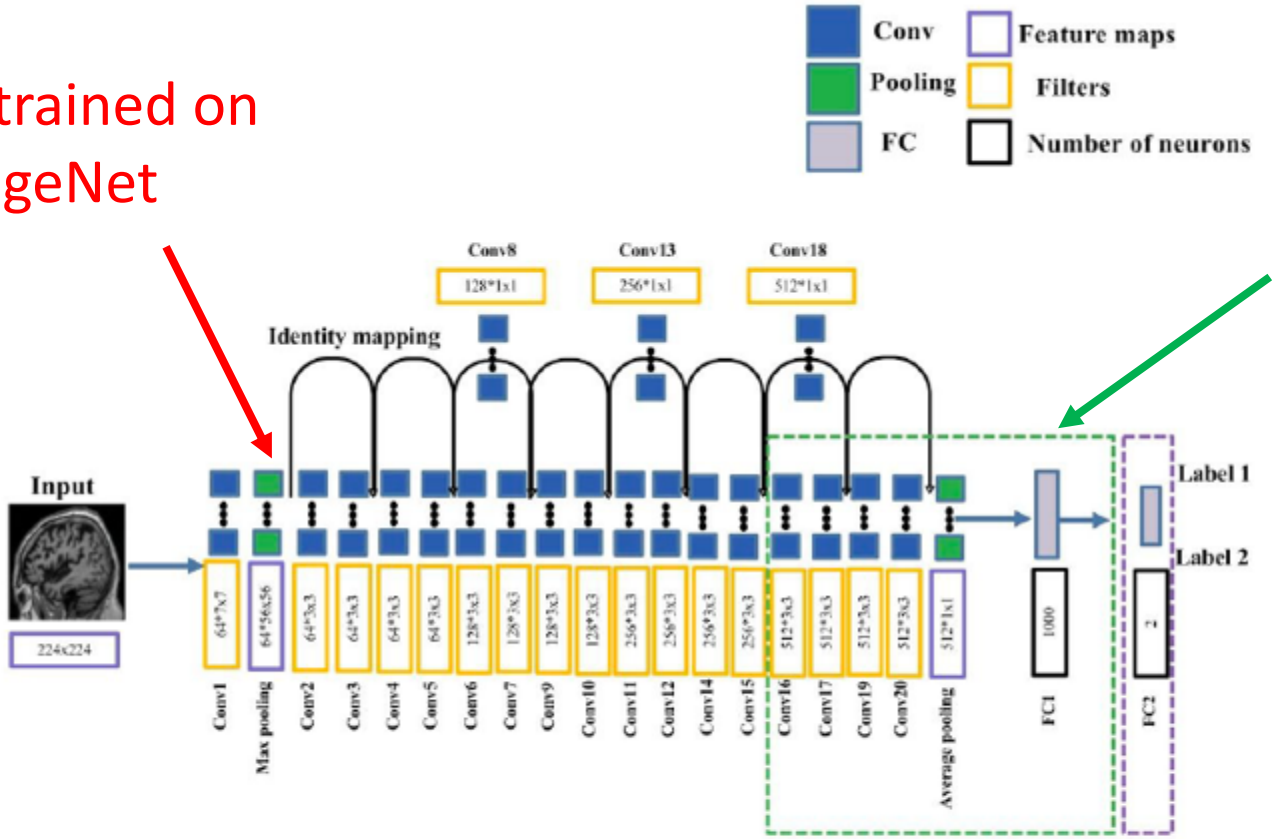
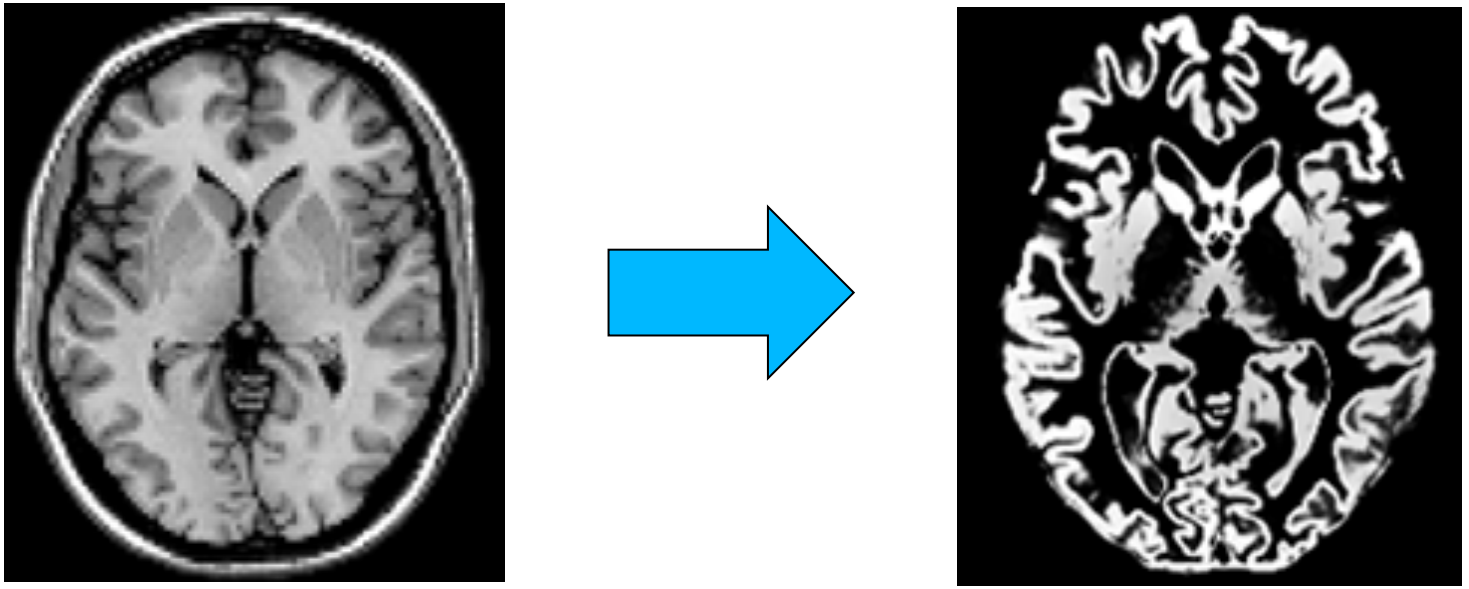


Figure 3: Architecture of the 2D slice-level CNN. An FC layer (FC2) was added on top of the ResNet. The last five convolutional layers and the last FC of ResNet (green dotted box) and the added FC layer (purple dotted box) were fine-tuned and the other layers were frozen during training. Filters for each convolutional layer represent the number of filters * filter size. Feature maps of each convolutional block represent the number of feature maps * size of each feature map. Conv: convolutional layer; FC: fully connected layer.

Feature extraction using domain knowledge

- **Knowledge:** Alzheimer's disease causes the loss of neurons which ultimately results in atrophy of the gray matter of the brain

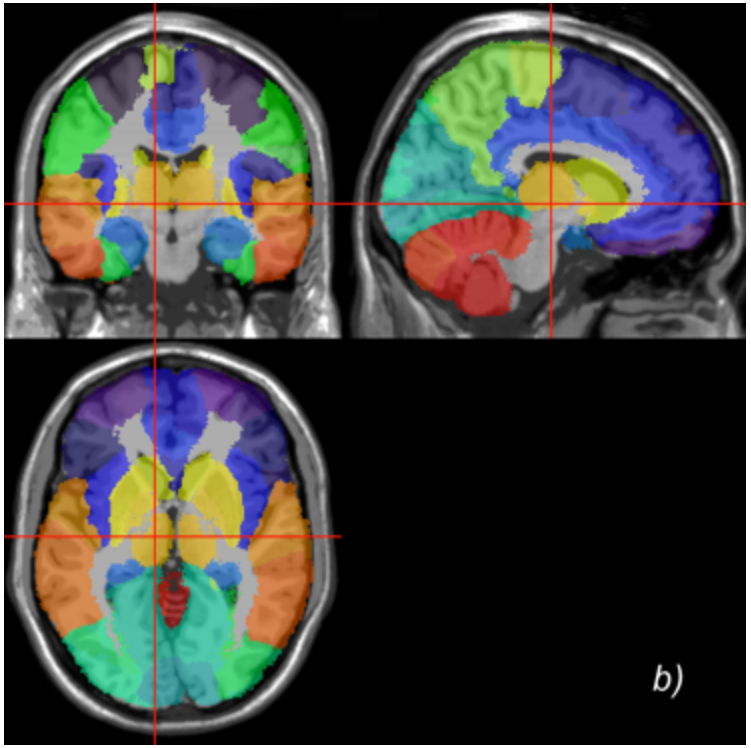
- Idea: quantify the gray matter in the image



Feature extraction using domain knowledge

- **Knowledge:** Alzheimer's disease affects preferentially some specific regions of the brain

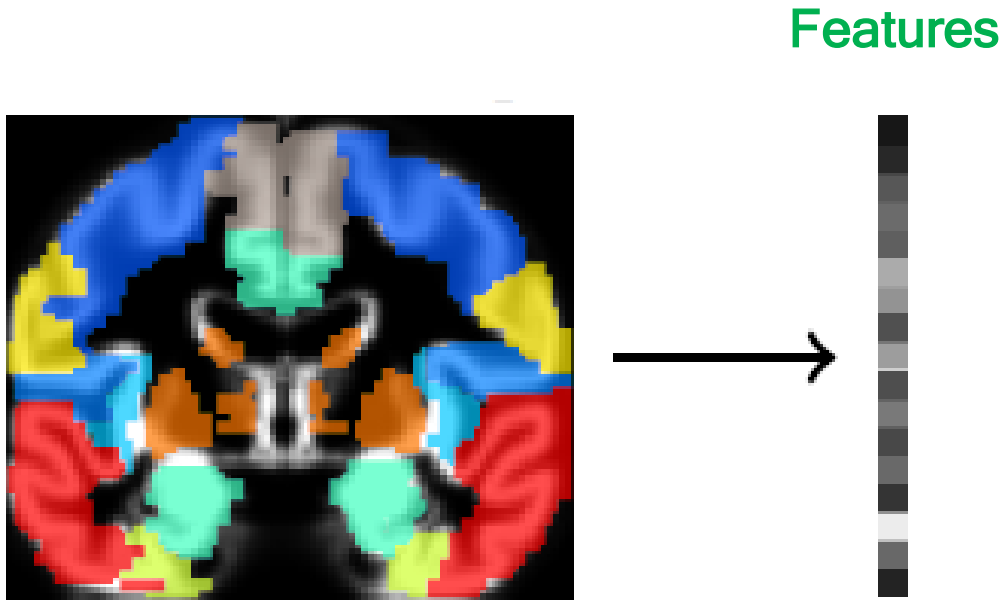
- Idea: parcellate the brain into different regions



All these operations are done automatically by sophisticated image processing algorithms

Feature extraction using domain knowledge

- **Features:** amount of gray matter in each region of the brain

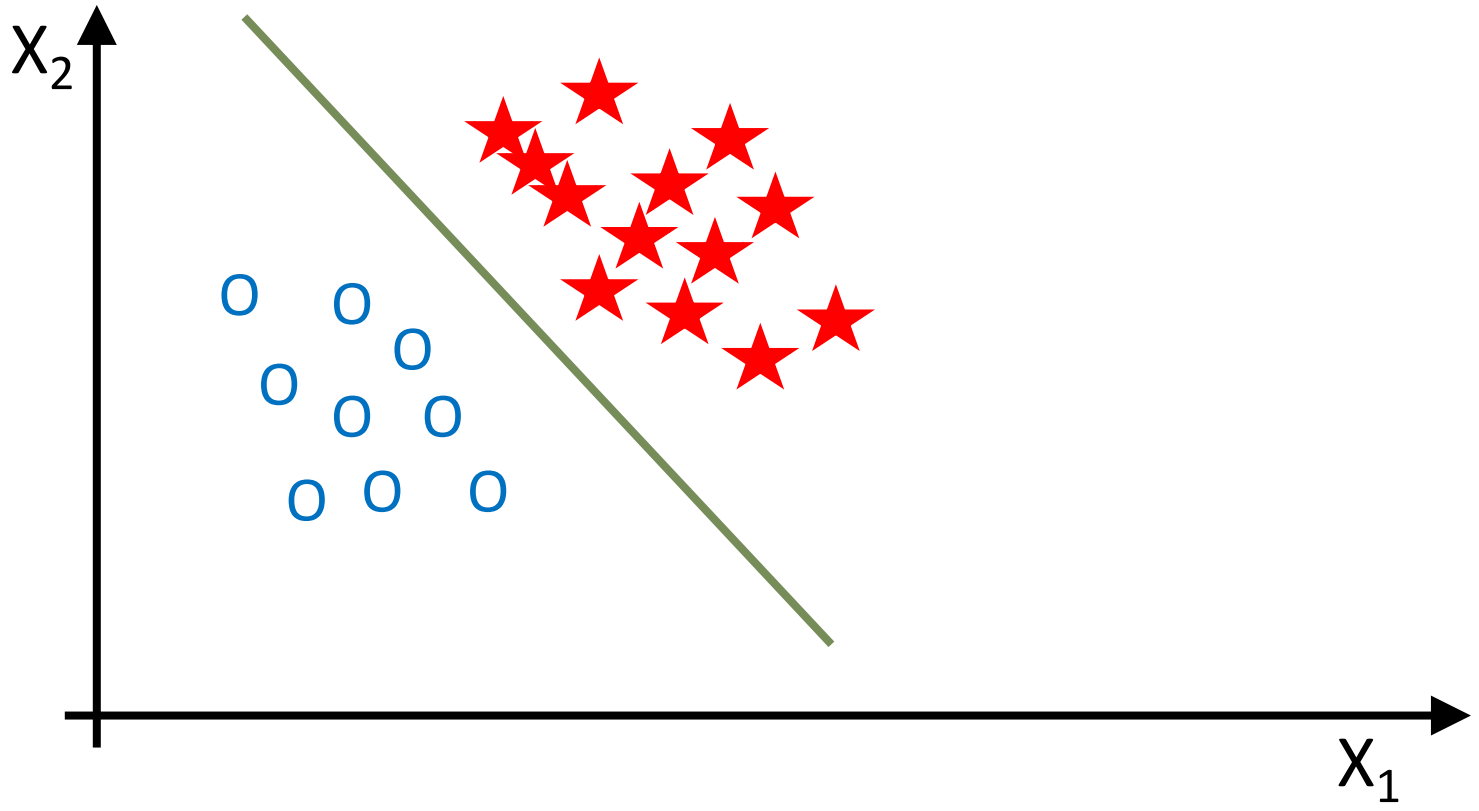


$x \in \mathbb{R}^p$

where p is the number of regions

Classifier

- **Classifier:** linear SVM



Linear SVM vs deep learning

	Alzheimer vs controls	Stable vs progressive MCI
Deep learning: 3D Whole-brain CNN	84%	73%
Deep learning: 3D Hippocampus CNN	86%	73%
Deep learning: 2D ResNet	76%	N/A
Linear SVM	86%	72%

Wen, Thibeau--Sutre et al, *In revision*

Image classification

- **A situation very different from computer vision and from other medical image classifications**

- **Where does it come from?**
  - Not enough samples (hundreds vs millions)
  - Brain images are highly constrained objects
  - Image preprocessing and feature extraction are very mature in this field

20 years of medical image computing have been useful, after all…