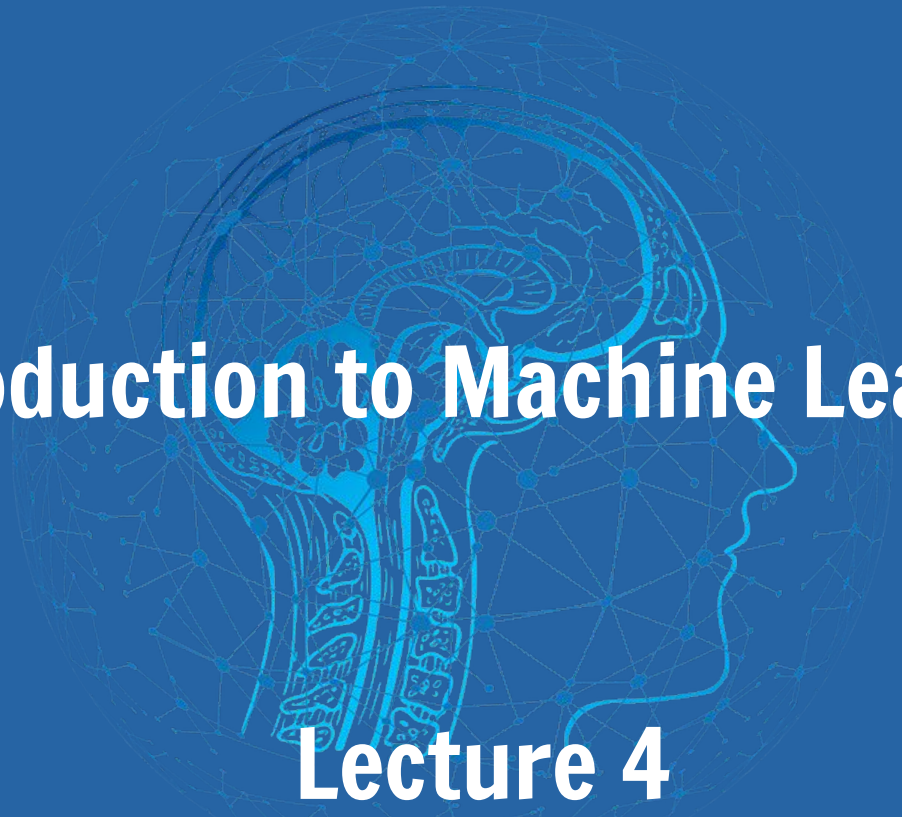
Imperial College London



Introduction to Machine Learning

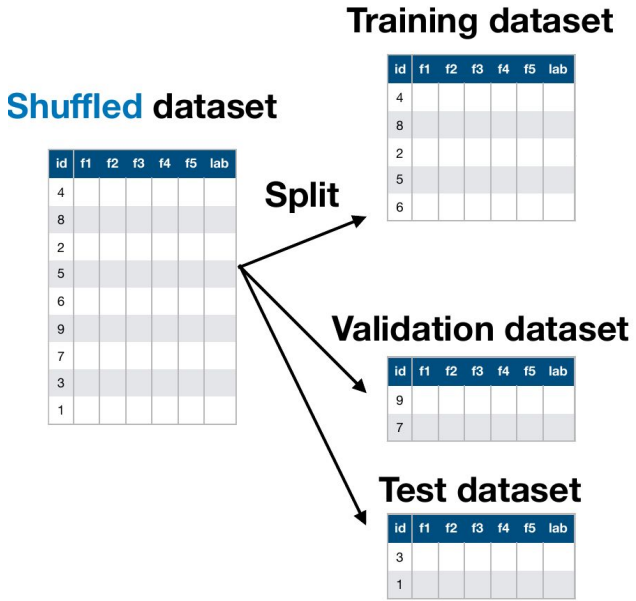
Lecture 4

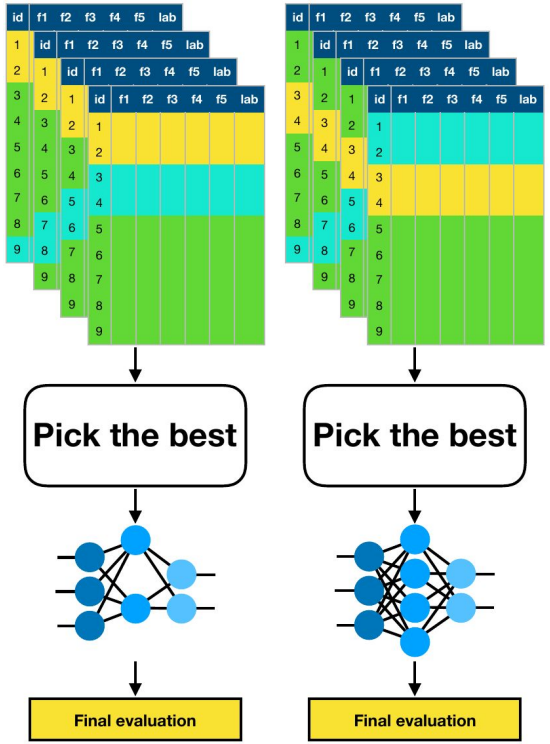
Antoine Cully & Marek Rei & Josiah Wang

In the previous lecture...

Recap: Evaluation

Setting up evaluation: either using train/dev/test datasets or cross-validation





Validation dataset used for tuning hyper-parameter and select the optimal model configuration.

Test dataset used for evaluate the model performance, and the result should be averaged value of all evaluation metrics

Recap: Evaluation metrics

Evaluation metrics: accuracy, precision, recall, F-measure.
Looking at the confusion matrix.

Results from the test dataset

Id	Labels	Predictions
1	+	+
2	+	+
3	+	-
4	+	+
5	-	-
6	-	+
7	-	-
8	-	+

Confusion Matrix

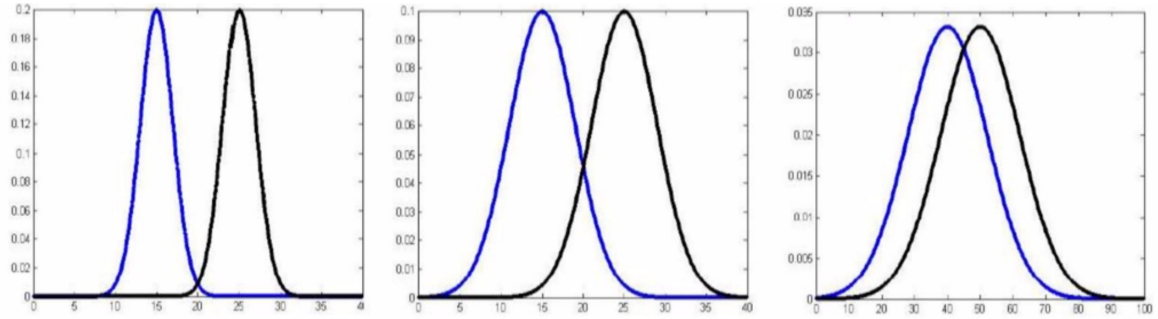
	Class 1 Predicted	Class 2 Predicted
Class 1 Actual	**TP**: True Positive	**FN**: False Negative
Class 2 Actual	**FP**: False Positive	**TN**: True Negative

Recap: Measuring uncertainty

Confidence intervals show the area where the true value is expected to be with N% probability.

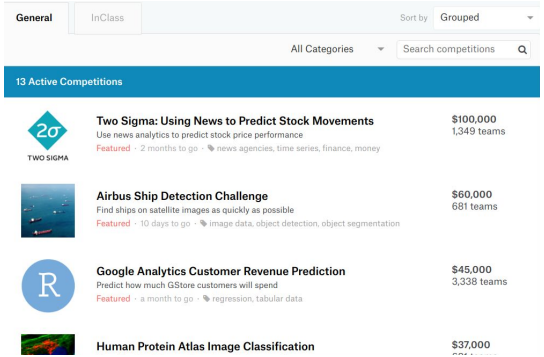
$$error_S(h) \pm Z_N \sqrt{\frac{error_S(h) * (1 - error_S(h))}{n}}$$

Statistical significance tests help us estimate whether two systems are actually different or whether the performance differences might be due to sampling error.

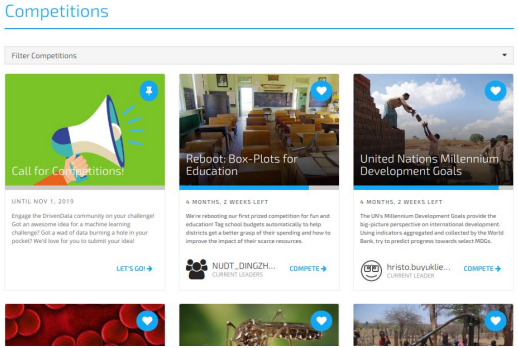


Today...

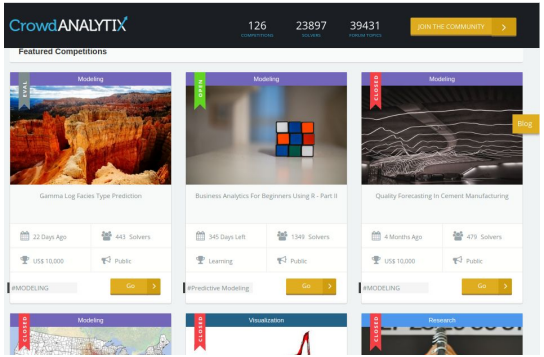
Open ML competitions



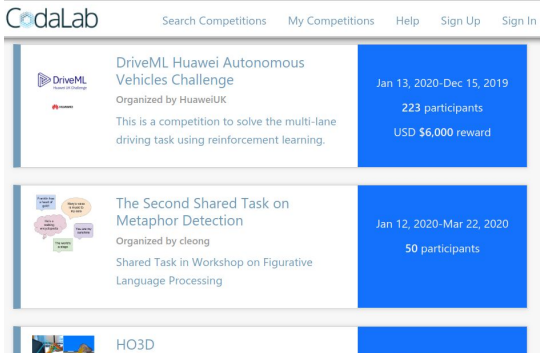
kaggle.com



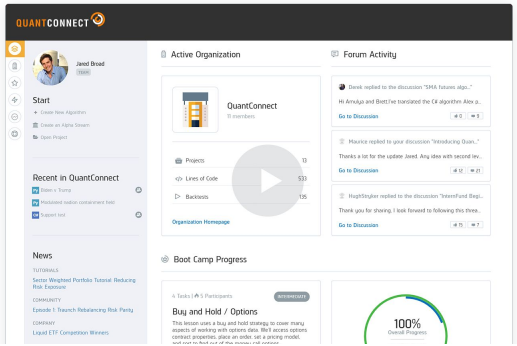
drivendata.org



www.crowdanalytix.com/community



competitions.codalab.org



quantconnect.com

Code snippets from today's lecture

master ic-intro-to-ml-2020 / Lecture4_NNets1.ipynb Go to file ...

marekrei Created using Colaboratory Latest commit ccceae9 1 minute ago History

1 contributor

806 lines (806 sloc) 179 KB Raw Blame

Open in Colab

Lecture 4: Artificial Neural Networks I

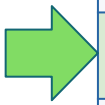
Loading the necessary libraries and the dataset.

We are using an example dataset containing statistics about different countries.

In [57]:
```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing
from sklearn.preprocessing import PolynomialFeatures

data = pd.read_csv('https://raw.githubusercontent.com/marekrei/ic-intro-to-ml-2020/master/data/country-stats.csv')
data.head()
```

https://github.com/marekrei/ic-intro-to-ml-2020/blob/master/Lecture4_NNets1.ipynb

Course plan



	Lecture
Week 2	Introduction to ML + Classification
Week 3	Instance-based Learning + Decision Trees
Week 4	Machine Learning Evaluation
Week 5	Artificial Neural Networks I
Week 6	Artificial Neural Networks II
Week 7	Unsupervised Learning
Week 8	Genetic Algorithms

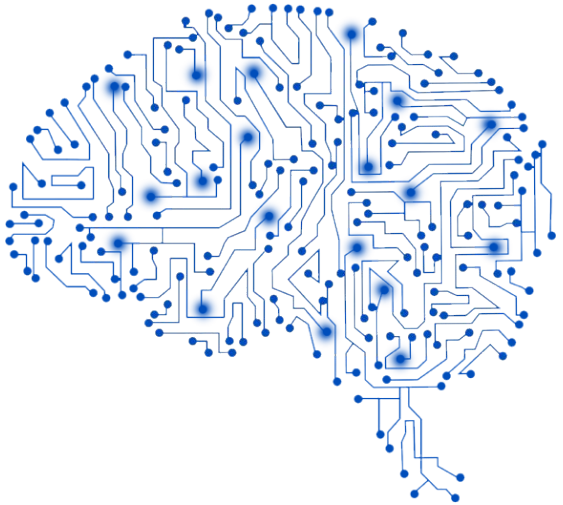
Today's lecture

- Introduction to neural networks
  - Some examples
  - A brief background
- Linear regression
  - Optimising with gradient descent
- A single neuron
  - Extending linear regression
- Multi-layer networks
- Activation functions

The Rise of Neural Networks

Neural Networks and Deep Learning

Artificial neural networks are a class of machine learning algorithms. Architectures of connected neurons, usually optimised with gradient descent.



Deep learning refers to using neural network models with multiple hidden layers. Complex models trained with large datasets.

The Rise of Deep Learning

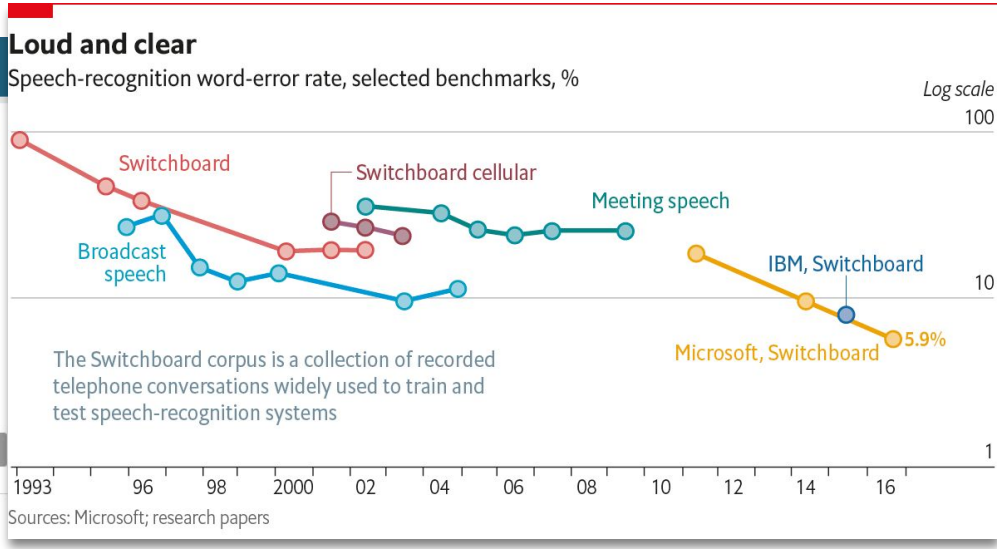
BUSINESS INSIDER UK TECH

Microsoft's voice-recognition tech is now better than even teams of humans at transcribing conversations

Matt Weinberger
Aug. 21, 2017, 7:30 PM 667

FACEBOOK LINKEDIN TWITTER

Follow Business Insider:



In October 2016, in a big milestone for artificial intelligence, Microsoft

http://uk.businessinsider.com/microsoft-research-beats-humans-at-speech-transcription-2017-8
https://www.economist.com/technology-quarterly/2017-05-01/language

The Rise of Deep Learning

ars TECHNICA SUBSCRIPTIONS SIGN IN

TECH —

Google's AlphaGo AI beats world's best human Go player

Ke Jie tried to use AlphaGo's own moves and lost.

SEBASTIAN ANTHONY - 5/23/2017, 2:20 PM

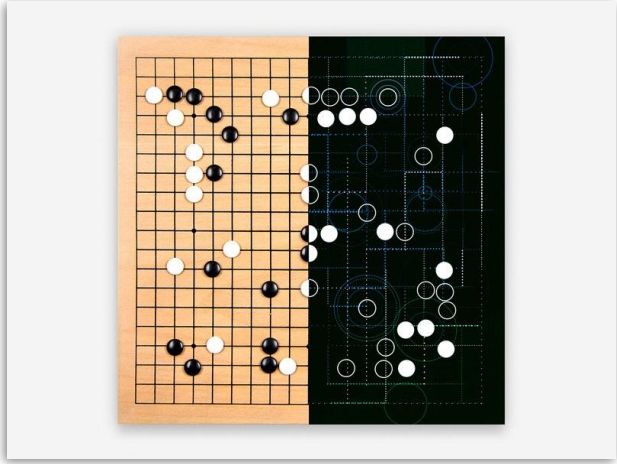
STR/AFP/Getty Images

Enlarge / China's 19-year-old Go player Ke Jie (L) prepares to make a move during the first match against Google's artificial intelligence program AlphaGo in Wuzhen, east China's Zhejiang province on May 23, 2017.

DeepMind's AlphaGo AI has defeated Ke Jie in the first round of a best-of-three Go match in China. A video of the match is embedded below. Ke Jie was defeated by just a half a point—the closest margin possible—but scoring versus AlphaGo is a little bit disingenuous: DeepMind's AI doesn't try to win by a large margin; it just plots the surest route to victory, even if it's only by half a point.

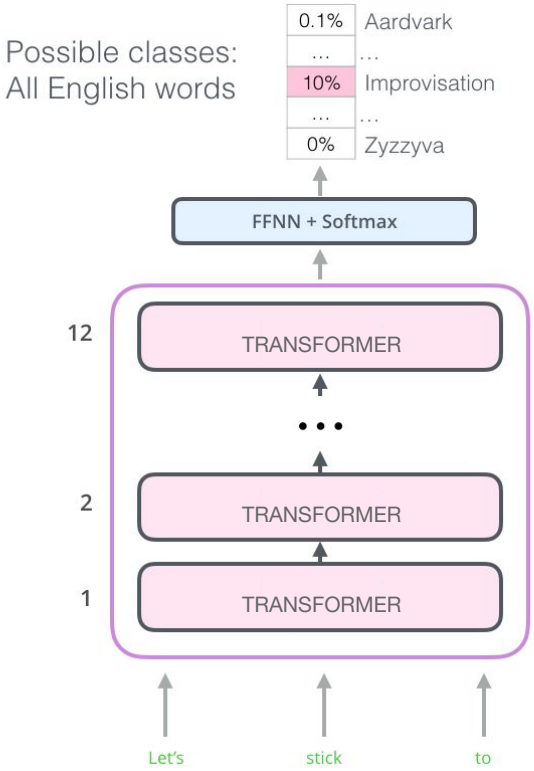
Ke Jie is generally considered to be the world's best human Go player, but he wasn't expected to win; AlphaGo defeated the Chinese 19-year-old earlier in the year during an unbeaten online 60-match victory streak.

Today's real-life match was a little different, though. According to DeepMind cofounder Demis Hassabis, Ke Jie "used the ideas

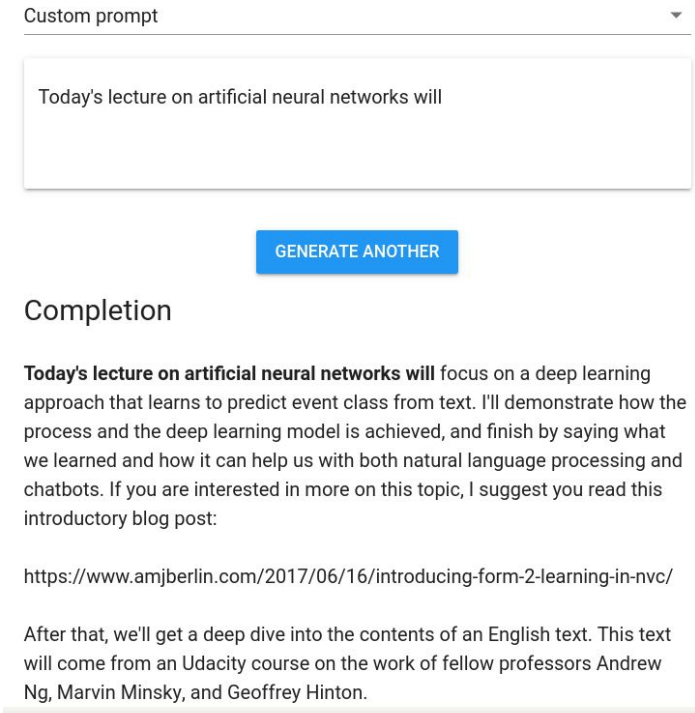


https://arstechnica.com/gadgets/2017/05/googles-alphago-ai-beats-worlds-best-human-go-player/

Language generation



https://github.com/openai/gpt-2
https://blog.floydhub.com/gpt2/



https://talktotransformer.com/

vectorized words, then predict what the next word will be.

Video editing

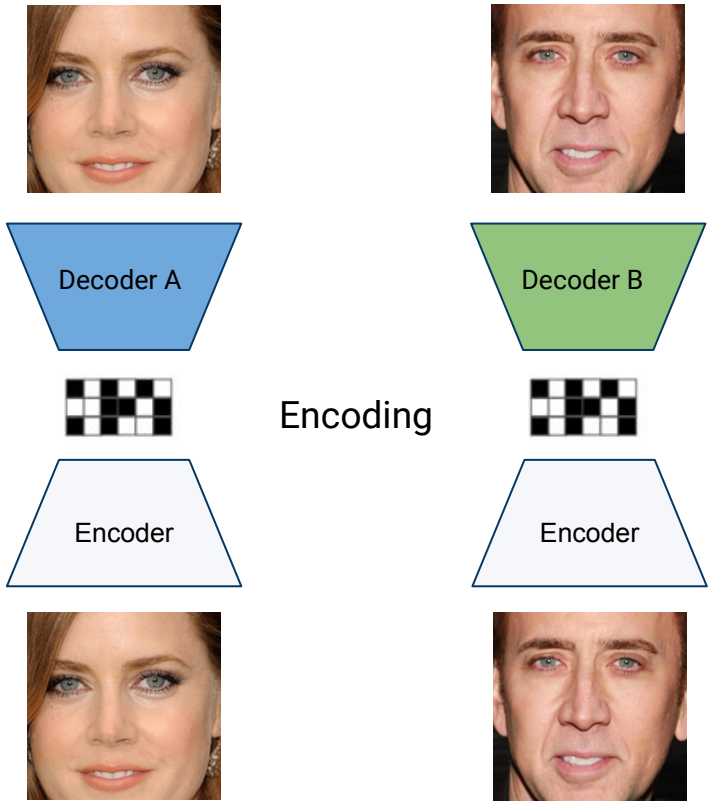


Image Integration using encoder/decoder:

Use same encoder to encode different faces into vector representation, but when decoding, use decoder B to decode the vector representation A so to generate a new face.

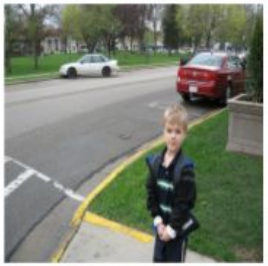
https://www.kdnuggets.com/2018/03/exploring-deepfakes.html
https://www.youtube.com/watch?v=v5NaLxlcpZ0

Combining images and text

there is a cat sitting on a shelf .

a plate with a fork and a piece of cake .

a black and white photo of a window .



a young boy standing on a parking lot next to cars .

a wooden table and chairs arranged in a room .

a kitchen with stainless steel appliances .

this is a herd of cattle out in the field .

a car is parked in the middle of nowhere .

a ferry boat on a marina with a group of people .

Kiros et al. (2014) Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models

But why now?

2012 - AlexNet wins ImageNet, Krizhevsky

2006 - Restricted Boltzmann Machine, Hinton

1998 - ConvNets for OCR, LeCun

1997 - LSTM, Hochreiter & Schmidhuber

1974 - backpropagation, Werbos

1958 - perceptrons, Rosenblatt

The theory was there before, but the conditions are now better for putting it into action.

1. Big Data

- Large datasets for training
- Better methods for storing and managing data

2. Faster Hardware

- Graphics Processing Units (GPUs)
- Faster CPUs
- More affordable

3. Better Software

- Better Optimization Algorithms
- Automatic Differentiation Libraries

Linear Regression

Supervised learning

Dataset: $\{< x^{(1)}, y^{(1)} >, < x^{(2)}, y^{(2)} >, \ldots, < x^{(N)}, y^{(N)} >\}$

Input features: $x^{(1)}, x^{(2)}, \ldots, x^{(N)}$

Known (desired) outputs: $y^{(1)}, y^{(2)}, \ldots, y^{(N)}$

Our goal: Learn the mapping $f : X \to Y$

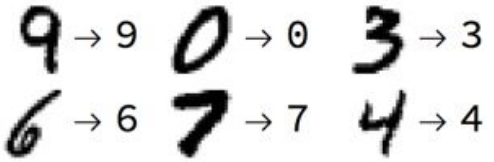
such that $y^{(i)} = f(x^{(i)})$

for all $i = 1, 2, 3, \ldots, N$

Continuous vs Discrete Problems

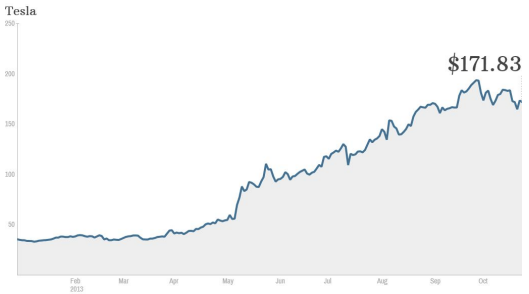
Classification: the desired labels are **discrete**

- Handwritten digits → digit label
- User tweets → detect pos/neg sentiment



Regression: the desired labels are **continuous**

- House size and age → price
- Company earnings, revenue → stock price

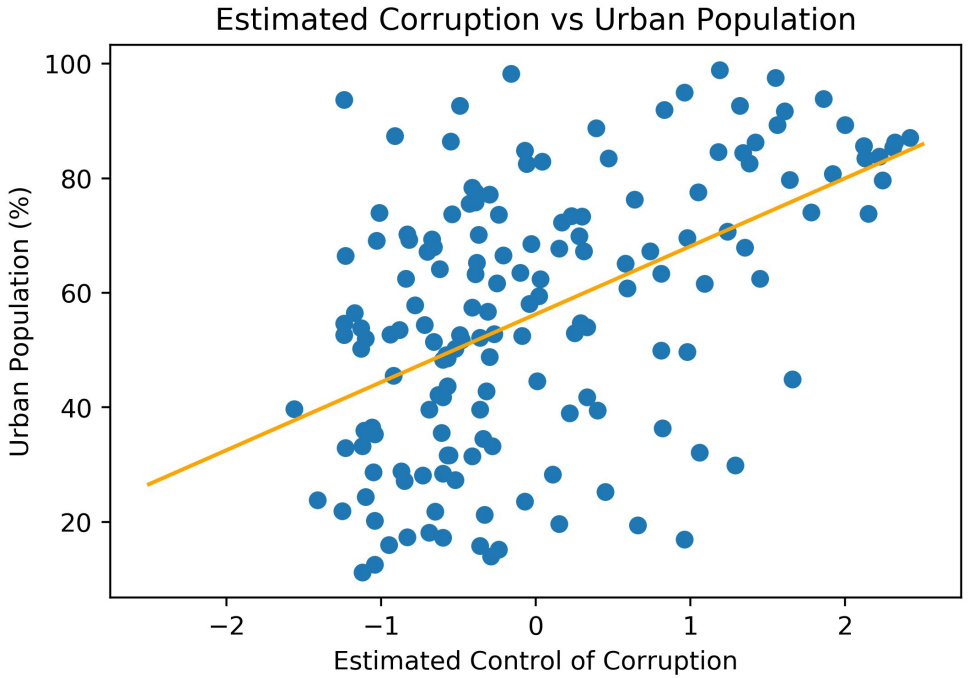


Regression or classification?

- Model the salary of baseball players
- Find what object is in a photo
- Predicting election results

Linear Regression

Simple linear regression - with one input variable



$$y = ax + b$$

Slope controls the angle

Intercept / bias controls the height

Linear Regression

	Country Name	GDP per Capita (PPP USD)	Enrolment Rate, Tertiary (%)
0	Afghanistan	1560.67	3.33
1	Albania	9403.43	54.85
2	Algeria	8515.35	31.46
3	Antigua and Barbuda	19640.35	14.37
4	Argentina	12016.20	74.83
5	Armenia	8416.82	48.94
6	Australia	44597.83	83.24
7	Austria	43661.15	71.00
8	Azerbaijan	10125.23	19.65
9	Bahrain	24590.49	33.46
10	Bangladesh	1883.05	13.15
11	Barbados	26487.77	60.84
12	Belgium	39751.48	69.26

x : GDP per Capita

y : Enrolment Rate

$$\hat{y} = ax + b$$

How do we find the best values for **a** and **b**?

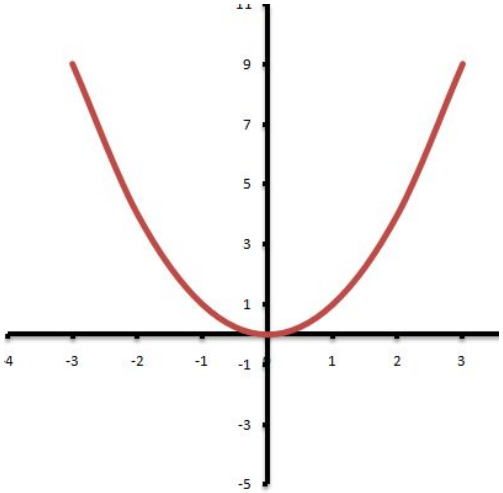
Loss Function

First, let's define what "best" actually means for us.

Our loss function (*cost function*) is the **sum-of-squares**:

$$E = \frac{1}{2}\sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

$$E = \frac{1}{2}\sum_{i=1}^{N}\left(ax^{(i)} + b - y^{(i)}\right)^2$$



Large penalty for outliers while allowing small losses

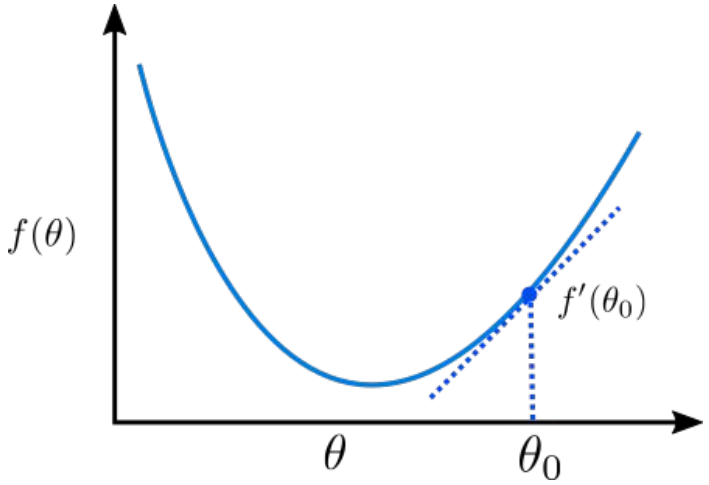
- Smaller value of E means our predictions are close to the real values
- Individual large errors incur a large exponential penalty
- Many small errors are acceptable and get a small loss
- Easily differentiable function

Updating parameters with derivatives

Derivatives show us how to change each parameter value to get a smaller loss.

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

The derivative of a function shows the direction of the slope.



Intuitively, the derivative of a function with respect to a parameter $\frac{\partial f}{\partial \theta}$ tells us how f changes when θ does.

Updating parameters with derivatives

What is the partial derivative of the loss function with respect to each of the parameters?

$$E = \frac{1}{2}\sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

$$E = \frac{1}{2}\sum_{i=1}^{N}\left(ax^{(i)} + b - y^{(i)}\right)^2$$

$$\begin{aligned}
\frac{\partial E}{\partial a} &= \frac{\partial}{\partial a}\frac{1}{2}\sum_{i=1}^{N}\left(ax^{(i)} + b - y^{(i)}\right)^2 \\
&= \frac{1}{2}\sum_{i=1}^{N}\frac{\partial}{\partial a}\left(ax^{(i)} + b - y^{(i)}\right)^2 \\
&= \sum_{i=1}^{N}\left(ax^{(i)} + b - y^{(i)}\right)x^{(i)} \\
&= \sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)x^{(i)}
\end{aligned}$$

Updating parameters with derivatives

What is the partial derivative of the loss function with respect to each of the parameters?

$$E = \frac{1}{2}\sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

$$E = \frac{1}{2}\sum_{i=1}^{N}\left(ax^{(i)} + b - y^{(i)}\right)^2$$

$$\begin{aligned}\frac{\partial E}{\partial b} &= \frac{\partial}{\partial b}\frac{1}{2}\sum_{i=1}^{N}\left(ax^{(i)} + b - y^{(i)}\right)^2 \\ &= \sum_{i=1}^{N}\left(ax^{(i)} + b - y^{(i)}\right) \\ &= \sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)\end{aligned}$$

Differentiation Refresher

Definition of derivative: $f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$

Constant rule: $\frac{d}{dx}(c) = 0$

Power rule: $\frac{d}{dx}(x^n) = nx^{n-1}$

Exponential (e): $\frac{d}{dx}e^x = e^x$

Exponential (general): $\frac{d}{dx}a^x = a^x \ln(a)$

Natural log: $\frac{d}{dx}(ln(x)) = \frac{1}{x}$

General log: $\frac{d}{dx}(log_a(x)) = \frac{1}{xln(a)}$

Differentiation Refresher

Constant multiple rule: $\frac{d}{dx}[cf(x)] = cf'(x)$

Sum rule: $\frac{d}{dx}[f(x) + g(x)] = f'(x) + g'(x)$

Difference rule: $\frac{d}{dx}[f(x) - g(x)] = f'(x) - g'(x)$

Product rule: $\frac{d}{dx}[f(x)g(x)] = f(x)g'(x) + g(x)f'(x)$

Quotient rule: $\frac{d}{dx}[\frac{f(x)}{g(x)}] = \frac{g(x)f'(x) - f(x)g'(x)}{[g(x)]^2}$

Chain rule: $\frac{d}{dx}f(g(x)) = f'(g(x))g'(x)$

Chain rule: $\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx}$

Training Linear Regression Models

Gradient Descent

Gradient descent: Repeatedly update parameters a and b by taking small steps in the negative direction of the partial derivative.

$$a := a - \alpha \frac{\partial E}{\partial a} \qquad b := b - \alpha \frac{\partial E}{\partial b}$$

$$a := a - \alpha \sum_{i=1}^{N} \left(a x^{(i)} + b - y^{(i)} \right) x^{(i)}$$

$$b := b - \alpha \sum_{i=1}^{N} \left(a x^{(i)} + b - y^{(i)} \right)$$

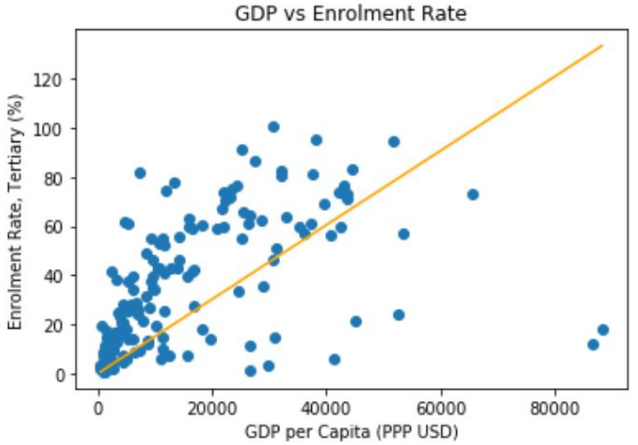
α : learning rate / step size

This same algorithm drives nearly all of the modern neural network models.

Gradient Descent

Implementing gradient descent in linear regression by hand.

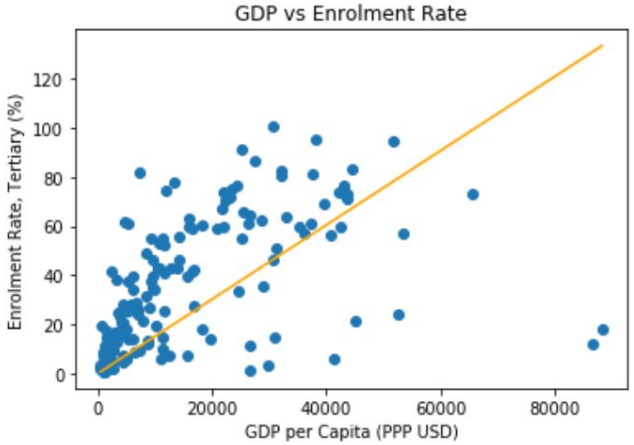
```
X = data["GDP per Capita (PPP USD)"].values
Y = data["Enrolment Rate, Tertiary (%)"].values

a = 0.0
b = 0.0
learning_rate = 1e-11

for epoch in range(5):
    update_a = 0.0
    update_b = 0.0
    error = 0.0
    for i in range(len(Y)):
        y_predicted = a * X[i] + b
        update_a += (y_predicted - Y[i])*X[i]
        update_b += (y_predicted - Y[i])
        error += np.square(y_predicted - Y[i])
    a = a - learning_rate * update_a
    b = b - learning_rate * update_b
    rmse = np.sqrt(error / len(Y))
    print(epoch, a, b, rmse, sep='\t')
```




```
epoch:0  a:0.0012  b:0.000000055  RMSE:43.6071
epoch:1  a:0.0015  b:0.000000079  RMSE:27.7650
epoch:2  a:0.0015  b:0.000000098  RMSE:27.1220
epoch:3  a:0.0015  b:0.000000116  RMSE:27.1017
epoch:4  a:0.0015  b:0.000000133  RMSE:27.1010
```

Gradient Descent

A more compact version, operating over all the datapoints at the same time.




```
X = data["GDP per Capita (PPP USD)"].values
Y = data["Enrolment Rate, Tertiary (%)"].values

a = 0.0
b = 0.0
learning_rate = 1e-11

for epoch in range(5):
    y_predicted = a * X + b
    a = a - learning_rate * ((y_predicted - Y)*X).sum()
    b = b - learning_rate * (y_predicted - Y).sum()
    rmse = np.sqrt(np.square(y_predicted - Y).mean())
    print(epoch, a, b, rmse, sep='\t')
```

```
epoch:0  a:0.0012  b:0.000000055  RMSE:43.6071
epoch:1  a:0.0015  b:0.000000079  RMSE:27.7650
epoch:2  a:0.0015  b:0.000000098  RMSE:27.1220
epoch:3  a:0.0015  b:0.000000116  RMSE:27.1017
epoch:4  a:0.0015  b:0.000000133  RMSE:27.1010
```

X is a vector, so there is no need to iterate the entire vector and sum

The Gradient

It can be more convenient to work with vector notation.

The gradient is a **vector of all partial derivatives**.

For a function $f : \mathbb{R}^K \rightarrow \mathbb{R}$, the gradient is

$$\nabla_\theta f(\theta) = \begin{bmatrix} \frac{\partial f(\theta)}{\partial \theta_1} \\ \frac{\partial f(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\theta)}{\partial \theta_K} \end{bmatrix}$$

The Analytical Solution

Solving the single-variable linear regression with the analytical solution. No iteration needed.

$$X = \begin{bmatrix} x^{(1)} & 1.0 \\ x^{(2)} & 1.0 \\ \vdots & \vdots \\ x^{(N)} & 1.0 \end{bmatrix} \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix} \quad \theta = \begin{bmatrix} a \\ b \end{bmatrix}$$

Setting the derivative to zero -> finding the minimum

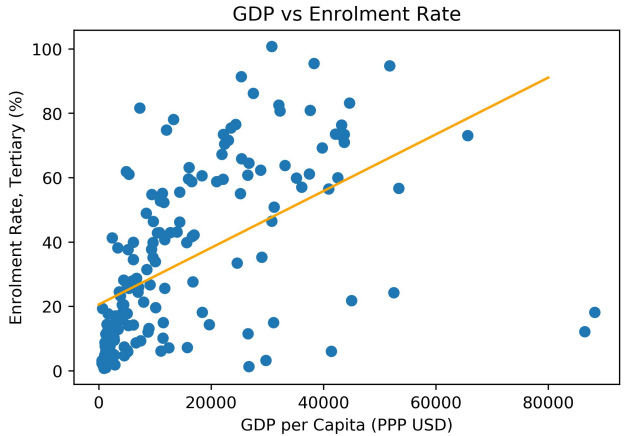
$$\nabla_\theta E(\theta) = X^T(X\theta - y) = 0$$

$$\implies \theta^* = (X^T X)^{-1} X^T y$$

Great for directly finding the optimal parameter values.
Not so great for large problems: matrix inversion has cubic complexity.

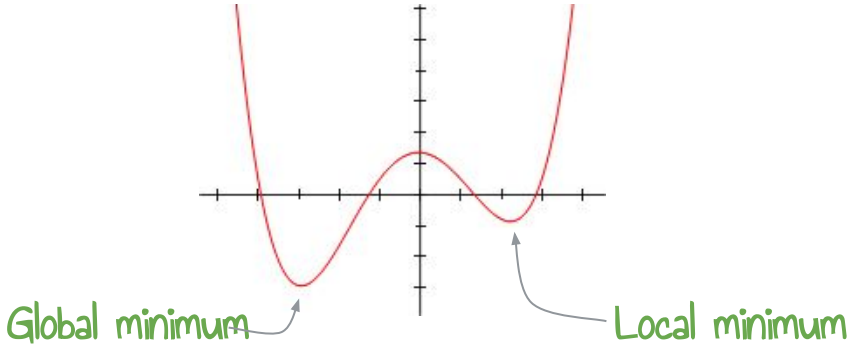
Analytical Solution with Scikit-Learn

```
from sklearn.linear_model import LinearRegression

model = LinearRegression(fit_intercept=True)
X = data["GDP per Capita (PPP USD)"].values.reshape(-1,1)
Y = data["Enrolment Rate, Tertiary (%)"]
model.fit(X, Y)

mse = np.square(Y - model.predict(X)).mean()
print("RMSE: " + str(np.sqrt(mse)))
```



RMSE: 22.630490998345973

Analytical Solution => Find global minimum
Gradient Descent => Might be local minimum because it only finds a place where nearby points are all greater/smaller than the minimum



Multiple Linear Regression

Simple linear regression: 1 input feature

Multiple linear regression: many input features

Output label

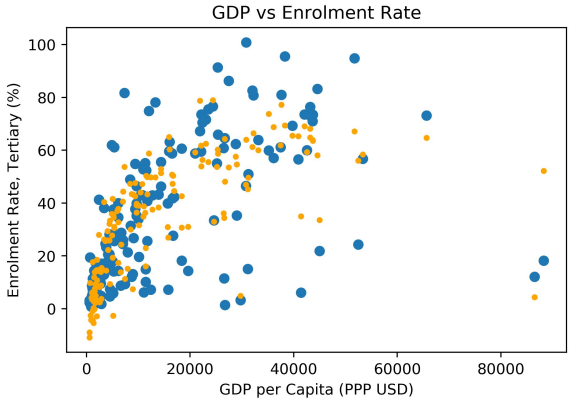
Input features

	GDP per Capita (PPP USD)	Population Density (persons per sq km)	Population Growth Rate (%)	Urban Population (%)	Life Expectancy at Birth (avg years)	Fertility Rate (births per woman)	Infant Mortality (deaths per 1000 births)	Unemployment, Total (%)	Estimated Control of Corruption (scale -2.5 to 2.5)	Estimated Government Effectiveness (scale -2.5 to 2.5)	Internet Users (%)	Enrolment Rate, Tertiary (%)
0	1560.67	44.62	2.44	23.86	60.07	5.39	71.0	8.5	-1.41	-1.40	5.45	3.33
1	9403.43	115.11	0.26	54.45	77.16	1.75	15.0	14.2	-0.72	-0.28	54.66	54.85
2	8515.35	15.86	1.89	73.71	70.75	2.83	25.6	10.0	-0.54	-0.55	15.23	31.46
3	19640.35	200.35	1.03	29.87	75.50	2.12	9.2	8.4	1.29	0.48	83.79	14.37
4	12016.20	14.88	0.88	92.64	75.84	2.20	12.7	7.2	-0.49	-0.25	55.80	74.83

$$y^{(i)} = \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \theta_3 x_3^{(i)} + \ldots + \theta_K x_K^{(i)} + \theta_{K+1}$$

Multiple Linear Regression

```
model = LinearRegression(fit_intercept=True)
X = data.copy().drop(["Country Name",
                      "Enrolment Rate, Tertiary (%)"],
                     axis=1)
Y = data["Enrolment Rate, Tertiary (%)"]

model.fit(X, Y)

mse = np.square(Y - model.predict(X)).mean()
print("RMSE: " + str(np.sqrt(mse)))
```

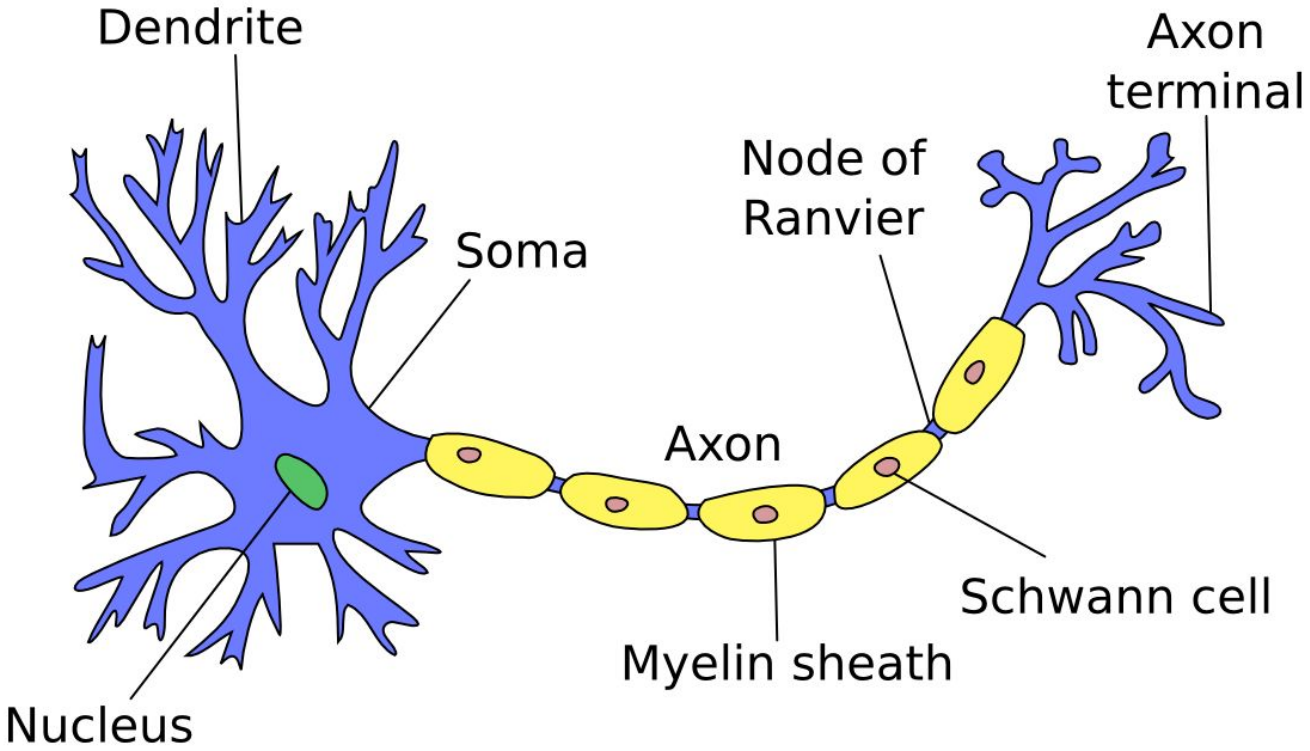


RMSE: 14.40196

With more features/information, the model is able to predict more accurate

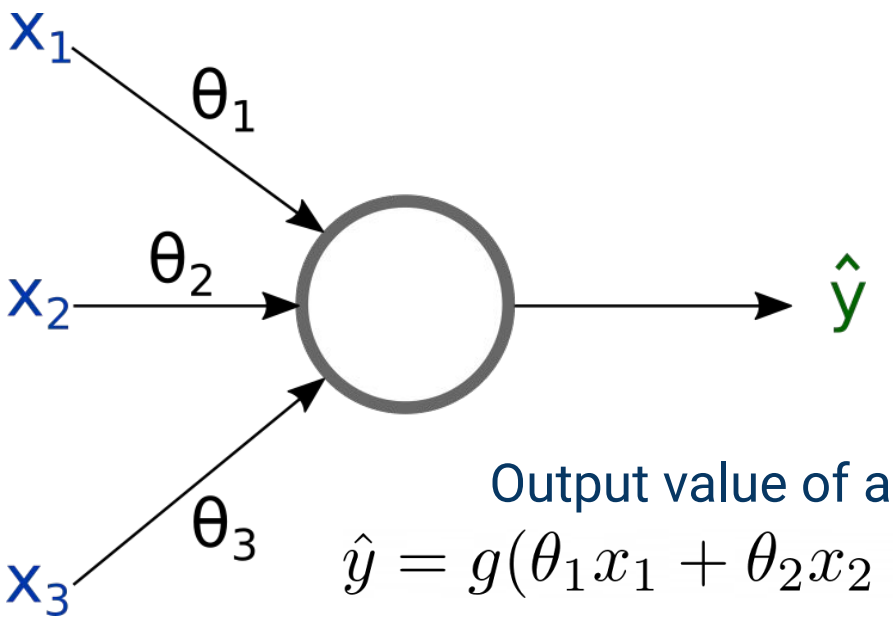
Neuron

Biological neuron



Takes in signals from the dendrites. Under specific conditions releases its own signal from the axon.

Artificial neuron



Features:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

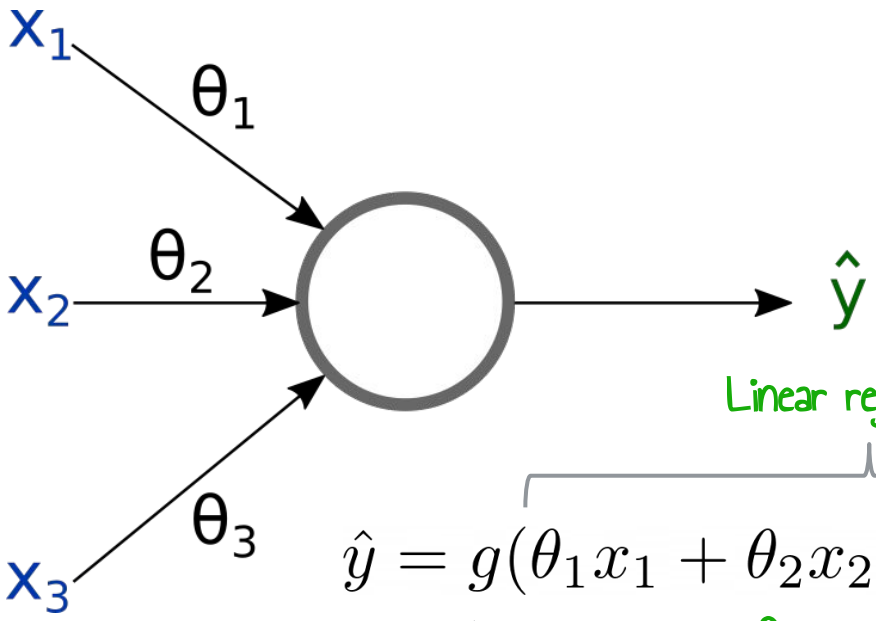
Parameters:

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Output value of a neuron:

$$\hat{y} = g(\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + b)$$

Artificial neuron



Features:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

1

Parameters:

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

4

Linear regression

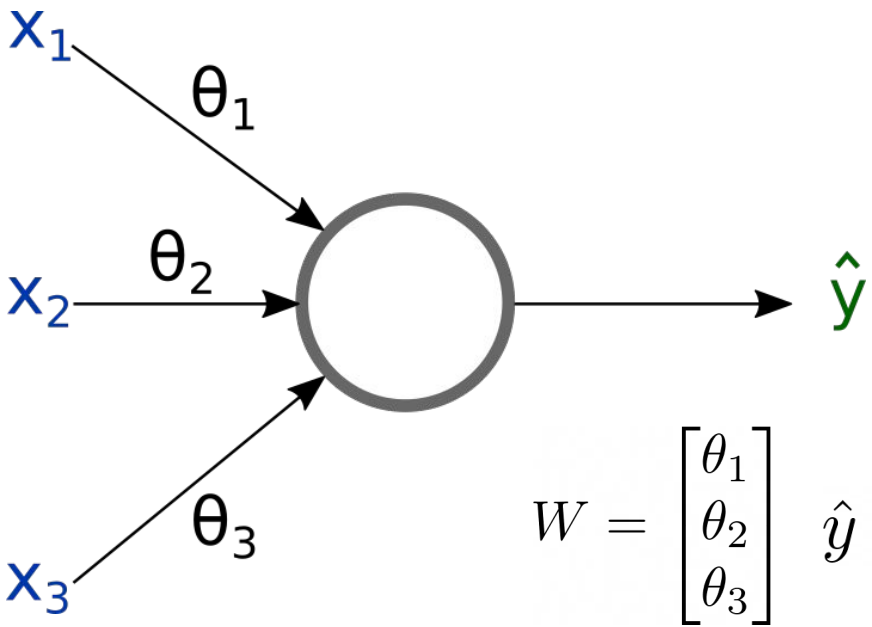
$$\hat{y} = g(\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + b)$$

Often implicit.
We can add an extra input feature with value 1 instead.

b => Biased term, can be reformulated by adding another input feature

Activation function
E.g., ReLU

Artificial neuron



Features:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Parameters:

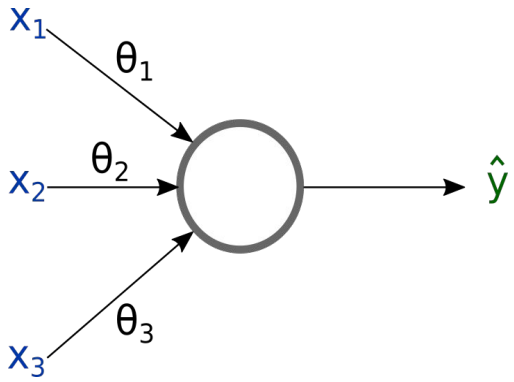
$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$W = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \quad \hat{y} = g(W^T x)$$

$$W \in \mathbb{R}^{K \times 1} \quad x \in \mathbb{R}^{K \times 1} \quad \hat{y} \in \mathbb{R}^{1 \times 1}$$

Logistic Activation Function

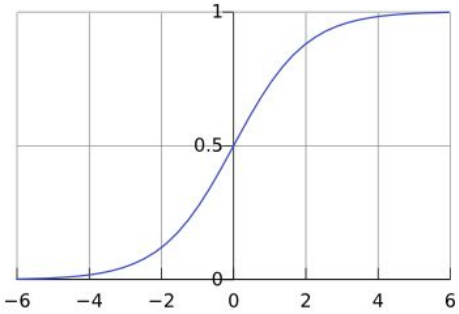
$$\hat{y} = g(\sum_k \theta_k x_k)$$



The **logistic function**, aka the **sigmoid** function.

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

$$z \in [-\infty, \infty] \qquad \hat{y} \in [0, 1]$$



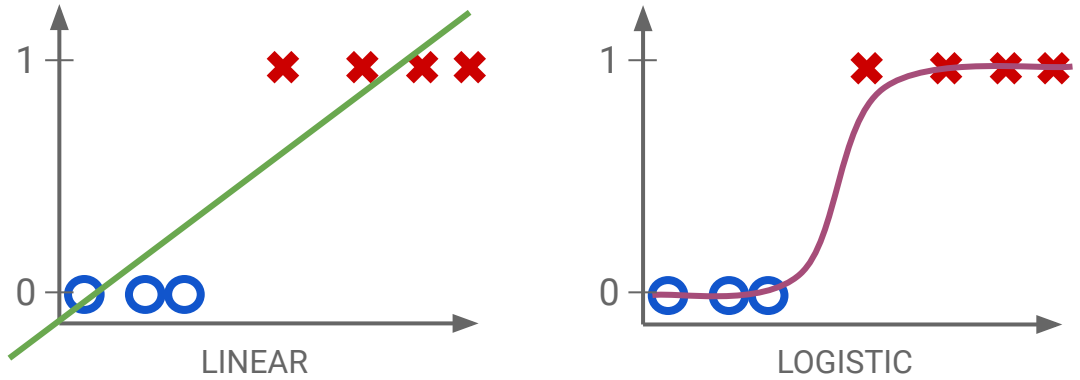
The value calculated from the summation will be the input z of the activation function, and squeezed into range [0, 1]

Logistic Regression

Not actually regression

In logistic regression, we pass the regression output through a logistic function. This allows us to do binary classification (0 or 1).

$$\hat{y} = g(\sum_k \theta_k x_k) \qquad g(z) = \frac{1}{1 + e^{-z}}$$

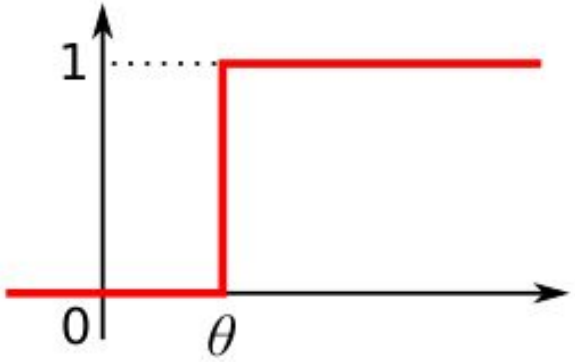


Logistic Regression => Only for classification, and uses sigmoid function as activation function`

The model is optimised using gradient descent.
More details on that in the next lecture.

Perceptron

Perceptron is an algorithm for supervised binary classification



It is an early version of an artificial neuron

Two classes: 0 and 1

It uses the **threshold function** as the activation function

$$h(x) = f(W^T x) = \begin{cases} 1 & \text{if } W^T x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron learning rule

$$\theta_i \leftarrow \theta_i + \alpha(y - h(x))x_i$$

- If the desired output y is equal to our prediction $h(x)$, then the right hand side of the summation becomes zero and the weights stay the same. Intuitively, don't fix it if it isn't broken.

- If $y = 1; h(x) = 0$ then the weight θ_i is increased when the corresponding input x_i is positive and decreased when it is negative. By doing so we want to make $W^T x$ bigger since the desired output is larger than our prediction.

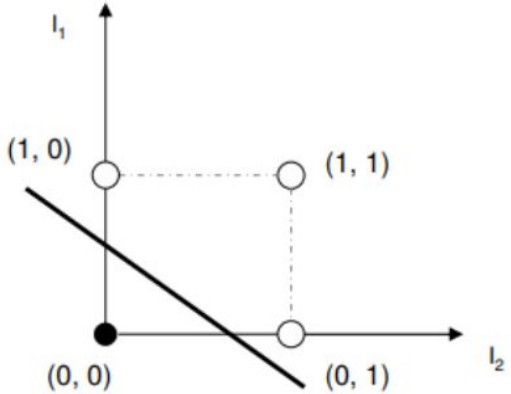
Similar to gradient descent

- If $y = 0; h(x) = 1$ then we want to decrease the summation so we do the opposite.

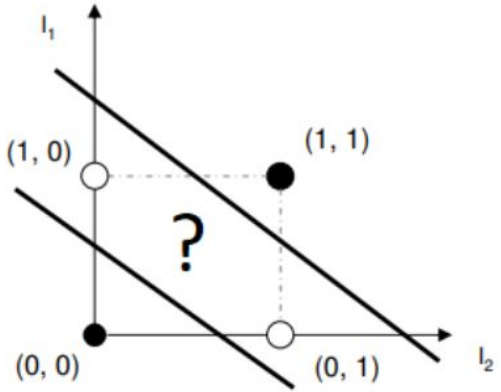
Perceptron

We can learn any linearly separable function using a perceptron.

OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0

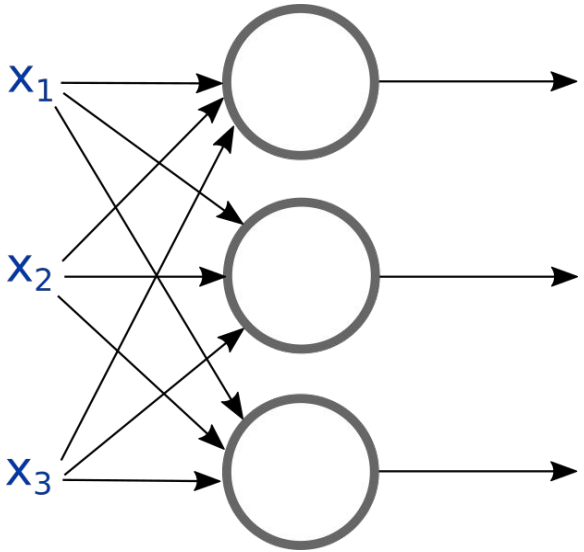


But the activation function is very sharp and non-differentiable, so we don't use it in more complex neural networks.

Multi-layer networks

Connecting the Neurons

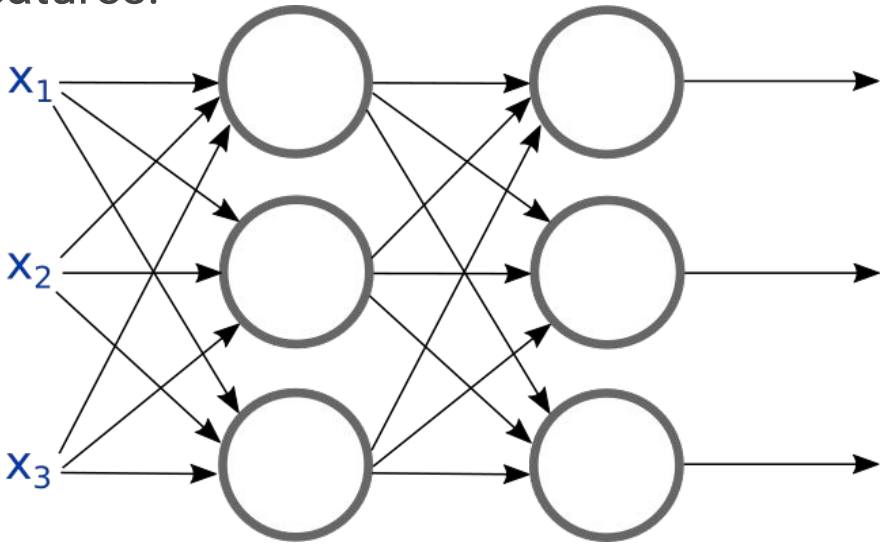
We can connect multiple neurons in parallel - each one will learn to detect something different.



Multilayer Perceptron

Not actually a perceptron

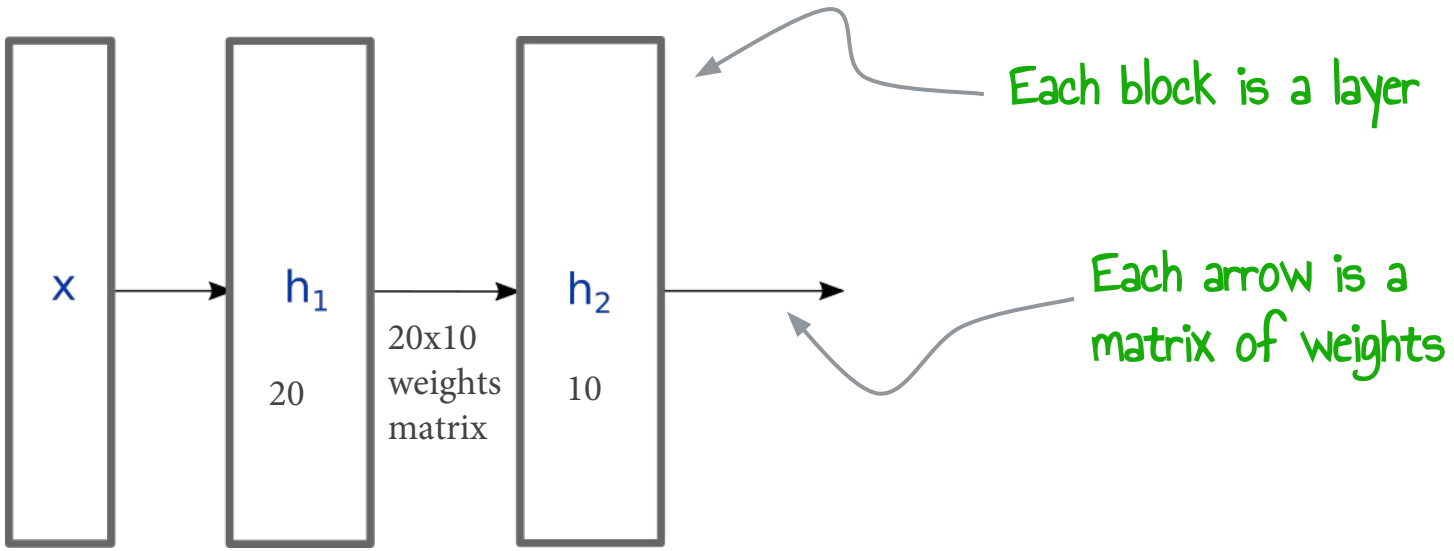
We can connect neurons in sequence in order to learn from higher-order features.



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

Multilayer Perceptron

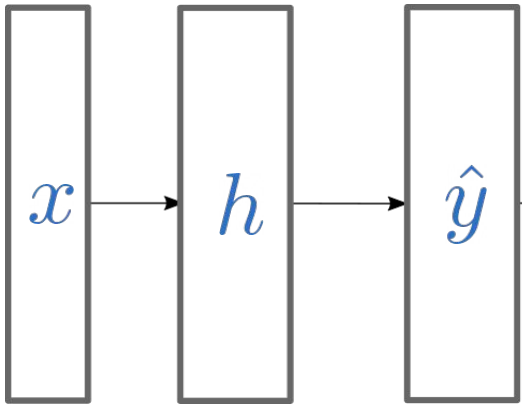
We can connect neurons in sequence in order to learn from higher-order features.



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

Multilayer Perceptron

Extending the neuron formulas to multilayer perceptrons



$$h = g_h(W_h^T x + b_h)$$

$$\hat{y} = g_{\hat{y}}(W_{\hat{y}}^T h + b_{\hat{y}})$$

$$x \in \mathbb{R}^{K \times 1}$$

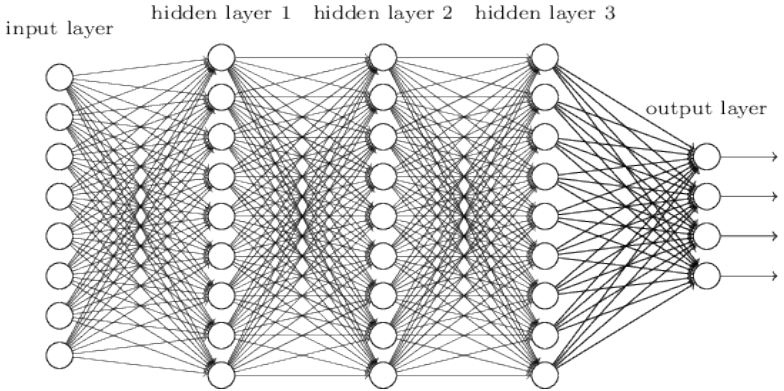
$$h \in \mathbb{R}^{H \times 1} \quad W_h \in \mathbb{R}^{K \times H} \quad b_h \in \mathbb{R}^{H \times 1}$$

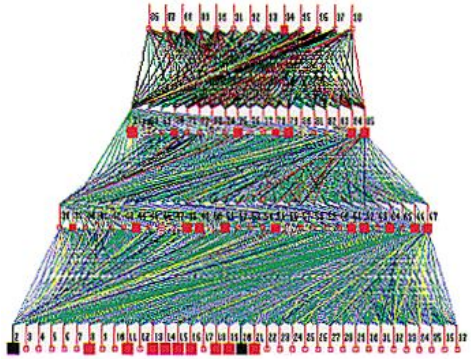
$$\hat{y} \in \mathbb{R}^{C \times 1} \quad W_{\hat{y}} \in \mathbb{R}^{H \times C} \quad b_{\hat{y}} \in \mathbb{R}^{C \times 1}$$

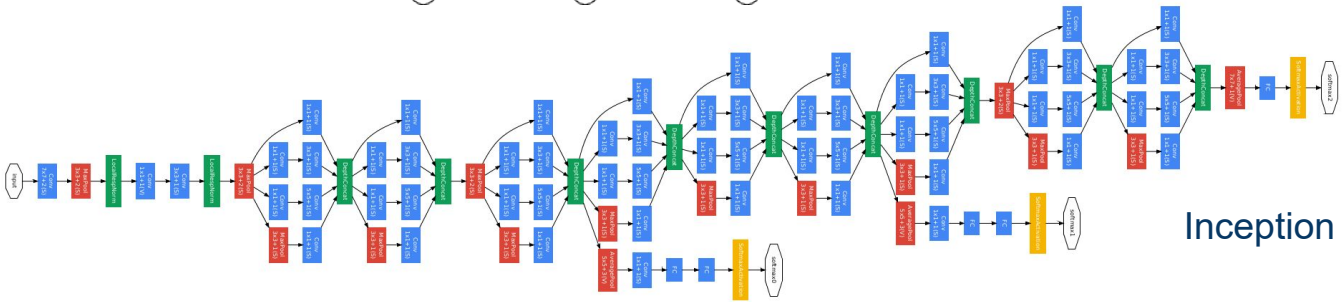
When something isn't working, it's a good idea to check that the matrix dimensions match!

Deep Neural Networks

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



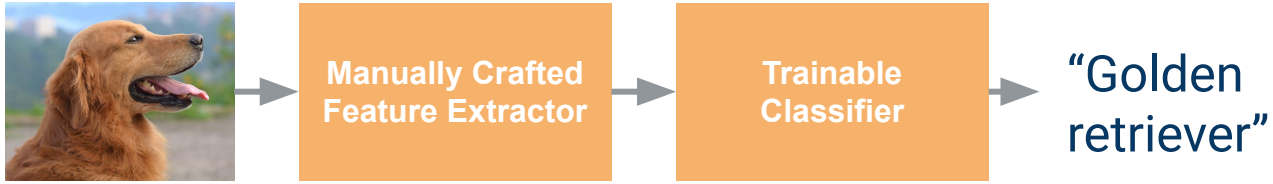




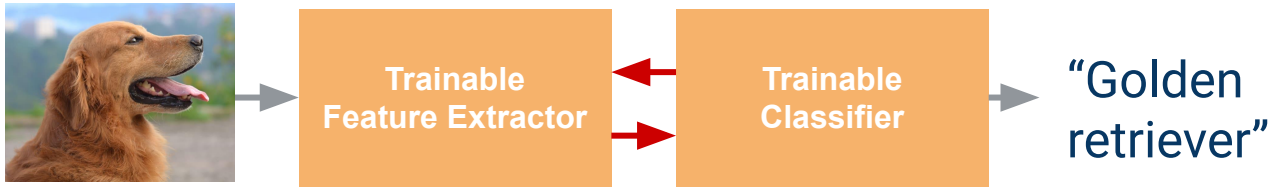
Inception => ImageNet Competition

Learning Representations & Features

Traditional pattern recognition



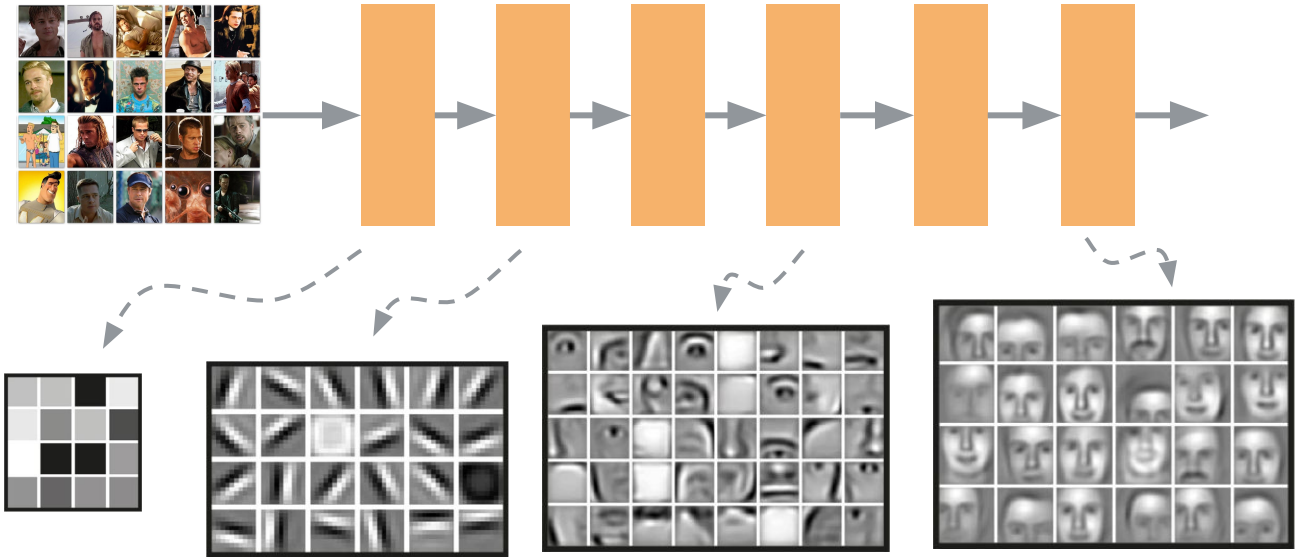
End-to-end training: Learn useful features also from the data



Low Level Neurons

Learning Representations & Features

Automatically learning increasingly more complex feature detectors from the data.

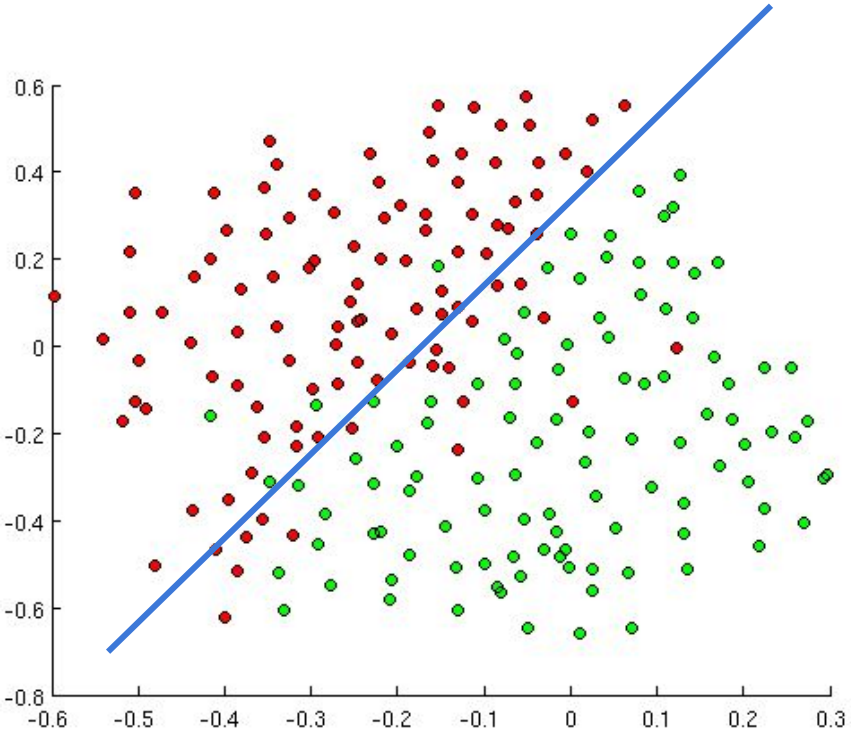


Lee et al. (2009) “Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations”

Activation functions

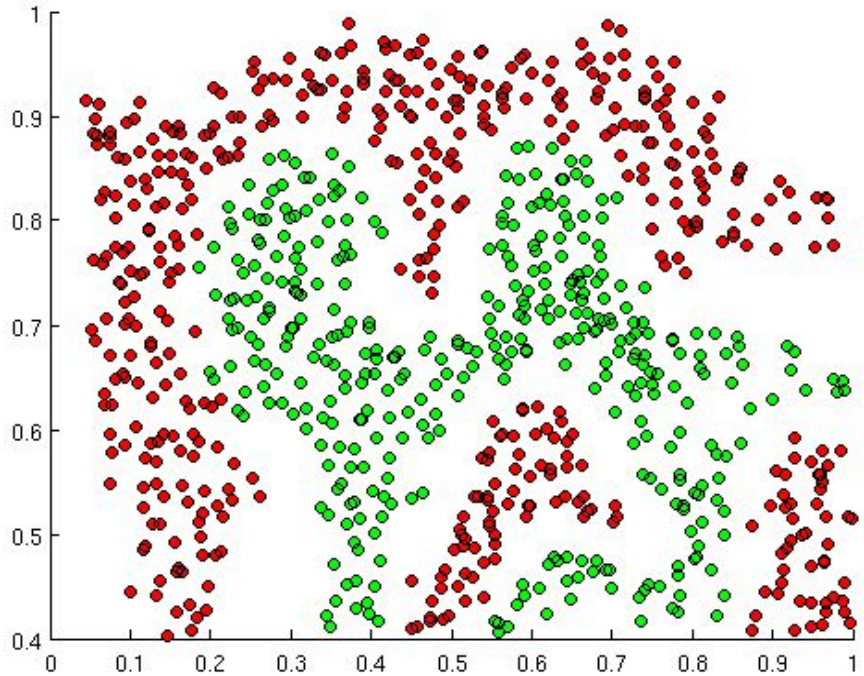
Linear Separability of Data

Linear models are great if the data is linearly separable.



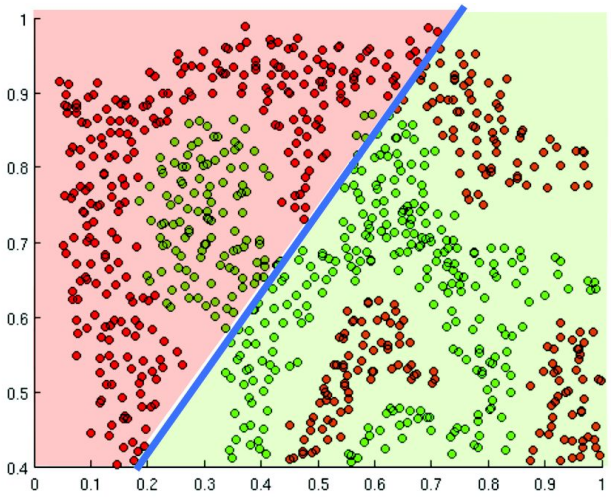
Linear Separability of Data

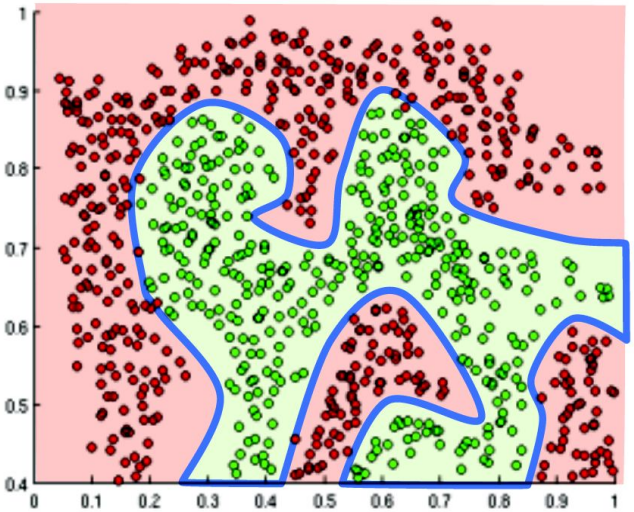
… but often that is not the case.



Linear Separability of Data

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





http://introtodeeplearning.com/

Activation functions

With multi-layer networks, activation functions become increasingly important.

Linear activation: The same as having no activation function. Directly passes on the output of the linear layer.

Similar to the identity function => Output equals to Input

$$f(x) = x \qquad \hat{y} = W^T x$$

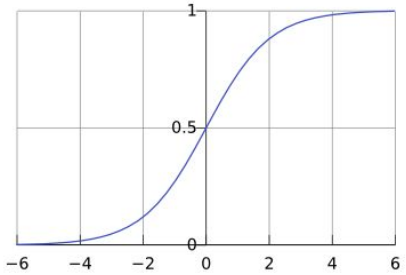
Could we just use linear activation everywhere? **No!**
Multi-layer linear network is equal to a single-layer linear network.

$$\hat{y} = W_1(W_2 x) \qquad \hat{y} = Ux \qquad U = W_1 W_2$$

That is why we need non-linear activations!

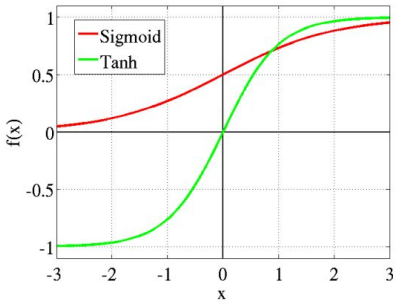
Activation functions

Sigmoid activation: Compresses the output smoothly into the range between 0 and 1. Also called the logistic function. A soft version of the threshold function.



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

Tanh activation: Similar shape to the sigmoid but ranges between -1 and 1. Input 0 also corresponds to output 0.

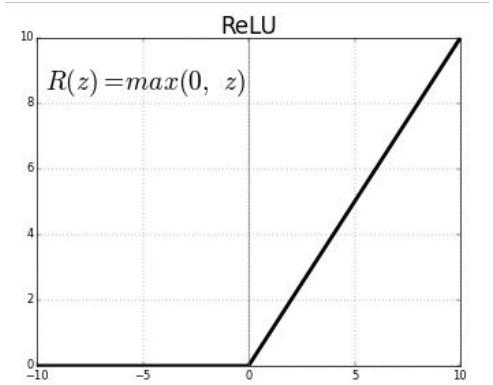


$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Activation functions

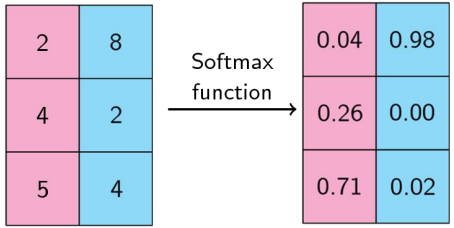
ReLU activation: Rectified linear unit. Linear in the positive part, but still non-linear overall.

$$f(x) = ReLU(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$$



Softmax activation: Scales the inputs into a probability distribution. The largest input will be large, the rest will be small. All output values sum up to 1.

$$softmax(z_i) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$



The output of input 4 in blue vector is comparably smaller than the output of input 8

Activation functions

- Most activation functions are **applied element-wise**: each element is passed through the function independently. Softmax is an exception. Softmax needs the entire vector and scale into a sum to 1

- **Which activation function** to use in hidden layers? ReLU is commonly used for very deep networks. Tanh and sigmoid also work well and can be more robust.

- The **activation of the output layer** should depend on the task. For example:
  - Classifying into two classes -> sigmoid or tanh
  - Predicting an unbounded score -> linear
  - Predicting a probability distribution -> softmax

Feedforward network in PyTorch

```
import torch
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.layer_h = nn.Linear(10, 5) # input->hidden weights
        self.layer_y = nn.Linear(5, 1) # hidden->output weights

    def forward(self, x):
        h = torch.tanh(self.layer_h(x)) # hidden layer with tanh activation
        y = torch.sigmoid(self.layer_y(h)) # output with sigmoid activation
        return y

net = Net() # constructing the network
input = torch.FloatTensor([x for x in range(10)]) # sample input
output = net(input) # executing the network
print(net) # printing network architecture
print(output) # printing output
```

```
Net(
  (layer_h): Linear(in_features=10, out_features=5, bias=True)
  (layer_y): Linear(in_features=5, out_features=1, bias=True)
)
tensor([0.2587], grad_fn=<SigmoidBackward>)
```