MATHS OVERVIEW

Key references

1. V. I. Smirnov. A course of higher mathematics: volume 1
2. N. S. Piskunov. Differential and Integral Calculus: volume 1
3. N. S. Piskunov. Differential and Integral Calculus: volume 2
4. V. E. Gmurman. Fundamentals of Probability Theory and Mathematical Statistics
5. V. A. Ilyin, E. G. Poznyak. Linear Algebra

I. Probability. Part 1

a) **Defining probability**

Naïve definition:

$$P(A) = \frac{\text{№ } of\ outcomes\ within\ a\ subset\ (A)}{\text{№ } of\ outcomes\ within\ a\ set\ (S)}$$

Sample space (S) – a set of all possible outcomes of an experiment.
An event (A) – a subset of the sample space. Thus, $A \subset B, x \in A, A = \{x_1, x_2, \dots, x_n\}$.

Assumptions:

- All outcomes are equally likely;
- There are finitely many outcomes (finite sample space).

Non-naïve definition:

Probability space – entity that consists of S (sample space) and P (a function that takes an event A as its input and gives the output between 0 and 1: $P(A) \in [0 \dots 1]$) and obeys two assumptions:

- $P(\emptyset) = 0, P(S) = 1$: the probability of the empty set is 0 and the probability of the full space is 1;
- $P(\cup_{n=1}^{\infty} A_n) = \sum_{n=1}^{\infty} P(A_n)$ if $A_1, A_2, \dots A_n$ are disjoint (these subsets do not overlap).

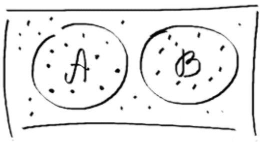
Also $P(A)$ is the probability of any one of the outcomes in A happening.

Additional info:

- What is the probability of the sample space?
- What is probability by Penn State?

b) **Disjoint and non-disjoint events**

Disjoint events – events that cannot happen at the same time, i.e. mutually exclusive $P(A \cap B) = 0$.



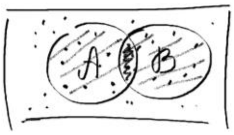
$P(A \cup B)$ is the probability that either the event A happens or the event B happens or both (but in this case it's impossible):

$$P(A \cup B) = P(A) + P(B)$$

$$P(A \cup B) = \frac{m_1 + m_2}{n} = \frac{m_1}{n} + \frac{m_2}{n} = P(A) + P(B)$$

EXAMPLE: when tossing a coin, we cannot get heads and tails simultaneously.

Non-disjoint events – events that can occur simultaneously $P(A \cap B) \neq 0$.



$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

$$P(A) = P(A \cap B) + P(A \cap \bar{B})$$
$$P(B) = P(A \cap B) + P(\bar{A} \cap B)$$

$$P(A \cup B) = \overbrace{P(A \cap B) + P(A \cap \bar{B})}^{P(A)} + \overbrace{P(A \cap B) + P(\bar{A} \cap B)}^{P(B)} - P(A \cap B)$$

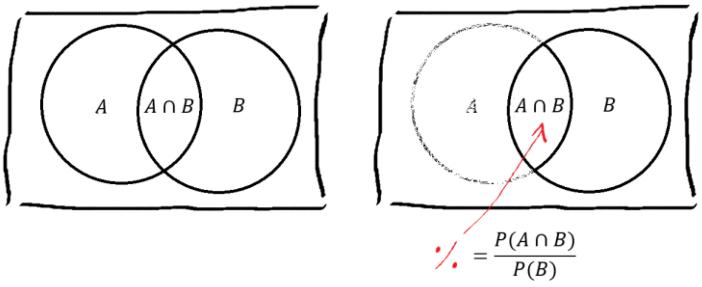
EXAMPLE: a chess piece can be white and a bishop at the same time.

c) **Dependent and independent events. Conditional probability. The product rule of probability**

Dependent events affect each other, i.e. after an even A happens, the probability of an event B changes.

Conditional probability – if an even A happened, what is the probability of an event B happening?

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$



EXAMPLE: once you take one card from the deck there are simply less cards that you can pick from. As a result, the probability of taking a specific card (or a card of a specific rank) after you've already taken one changes.

The product rule of probability for dependent events – we can derive it from the formula of conditional probability

$$P(A \cap B) = P(B)P(A|B)$$
$$P(B \cap A) = P(A)P(B|A)$$
$$P(A_1 \cap A_2 \cap \ldots \cap A_n) = P(A_1)P(A_2|A_1)P(A_3|A_1 \cap A_2) \ldots P(A_n|A_1 \cap A_2 \ldots \cap \ldots A_{n-1})$$

The product rule of probability for independent events – if events are independent, $P(A|B) = P(A)$ and $P(B|A) = P(B)$. Thus:

$$P(A \cap B) = P(A)P(B)$$

d) **Some notes on exclusive and dependent events**

- If events are disjoint, they are dependent because if A happens, the probability of B happening is 0;
- If events are non-disjoint, they can be either dependent or independent.

f) **Random variable**

RV – a function that maps each outcome from the **sample space** (the set of all possible outcomes) to a real number. RV is a way to encode outcomes numerically. So, an RV is neither random, nor a variable, it is a function:

- Unlike regular variable, an RV cannot be replaced with a single (fixed) value;
- Randomness comes from conducting an experiment whose outcomes we cannot (fully) control, but not from an RV which is just a function.

EXAMPLE: if we flip a coin, the sample space is $\{H, T\}$. Then, we can encode these outcomes as 1 and 0 respectively. This encoding (mapping / function) is a random variable X: $X(H) = 1, X(T) = 0$.

Realisation of an RV – informally, it is the value we actually observed from the set of possible outcomes that the variable can take. This definition is needed to distinguish between the collection of all the things that may happen (sample space) and what actually happened. For instance, once a die lands, an RV X maps the outcome to a real number.

Probability distribution – a function that yields the probability of an RV taking a certain value (fall within a specified range), so it operates on realizations. Extending the example given earlier: how likely it is that the value of X is equal to 1 $P(X = 1)$?

Random sample – is a collection of i.i.d. RVs $X_1, X_2, \dots, X_n$. The data we observed are realisations $x_1, x_2, \dots, x_n$ of these i.i.d. RVs.

…

---

e) **Discrete random variable**

Discrete RV – RV is discrete if it takes a countable number of distinct values.

Probability distribution of a discrete RV – an array of probabilities associated with each outcome / value of a RV $P(X = x)$. For example, if X is the number of heads we can get after flipping a coin many times, then $P(X = 5)$ is the probability of getting exactly 5 heads.

Probability mass function – a function that gives the probability that a discrete RV is exactly equal to some value.

f) **Continuous random variable**

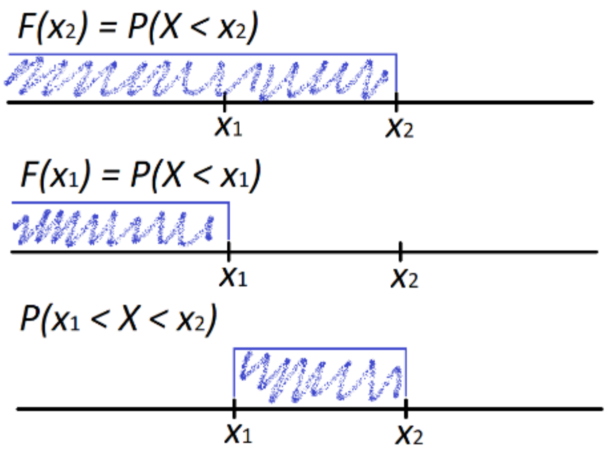
Continuous RV – RV is continuous if it takes an infinite number of distinct values. For example, the hight of a person is a continuous RV since it could be any non-negative value.

Probability distribution of a continuous RV is usually described via CDF and PDF.

Cumulative distribution function of a continuous RV (CDF) – the probability that X will take a value less than or equal to x: $F(x) = P(X < x)$

Some important properties:

- Since $F(x)$ is a probability, by definition we have $0 \leq F(x) \leq 1$;
- $P(a \leq X < b) = F(b) - F(a)$;



$$P(X < x_2) = P(X < x_1) + P(x_1 \leq X < x_2)$$
$$P(X < x_2) - P(X < x_1) = P(x_1 \leq X < x_2)$$
$$F(x_2) - F(x_1) = P(x_1 \leq X < x_2)$$

We can also explore this property from a different angle:

$$\int_{x_1}^{x_2} F'(x)dx = F(x_2) - F(x_1) = P(X < x_2) - P(X < x_1) = P(x_1 \leq X < x_2)$$

- $F(x) = P(X < x) = \int_{-\infty}^{x} f(x)dx$, PDF will be described later.

Probability density function of a continuous RV (PDF) – the first derivative of CDF: $F'(x) = f(x)$.

By definition, the derivative of CDF equals to:

$$F'(x) = f(x) = \lim_{\Delta x \to 0} \frac{\overbrace{F(x + \Delta x) - F(x)}^{P(x<X<x+\Delta x)}}{\Delta x}$$

As was shown earlier $F(x + \Delta x) - F(x) = P(x < X < x + \Delta x)$. We divide the probability of being within the interval $(x; x + \Delta x)$ by the length of this interval Δx. Thus, we get the "average" probability or probability density in other words.

Additional info:

- Introduction to Probability, Statistics, and Random Processes
- Random variables by Yale.edu

I. Probability. Part 2

a) Bayes' theorem. Discrete case

- Let's say we are studying an event A – the probability of passing a very difficult maths exam;
- In addition to that, we explore this event A in conjunction with certain conditions B_1 and B_2:
  - if a student is a historian, what is the probability of passing the exam $P(A|B_1)$;
  - if a student is a mathematician, what is the probability of passing the exam $P(A|B_2)$.

- Probabilities $P(A|B_1)$ and $P(A|B_2)$ we usually know. For instance, a research group analysed how students who specialise in different subjects pass their maths exams;
- Our task is to calculate the following probability: $P(B_1|A)$ if a student passed the exam, what is the probability that they are a historian?

Why it is important to go from something we already know $P(A|B_2)$ [if a student is a historian, what is the probability of passing the exam] to this "unknown" probability $P(B_1|A)$ [if a student passed the exam, what is the probability that they are a historian]?

At first glance, $P(B_1|A)$ should be low since if a student passed this hard maths exam, they are more likely to be a mathematician. So, we may expect $P(B_1|A) < P(B_2|A)$. Especially, if given $P(A|B_1)$ and $P(A|B_2)$ are, for example, 0.1 and 0.9 respectively.

However, if, for whatever reason, we have 10,000 historians and 10 mathematicians, then $P(B_1|A)$ will be larger than $P(B_2|A)$ simply because of the sheer number of historians. So, when we contextualise $P(A|B_1)$ and $P(A|B_2)$, taking into account the conditions of our experiment, in our case it is the number of students in each stream, the picture can change drastically.

$$P(B_i|A) = \frac{P(B_i)P(A|B_i)}{\sum_{i=1}^{n} P(B_i)P(A|B_i)} = \frac{P(B_i)P(A|B_i)}{P(A)}$$

$$\underbrace{\left.\begin{aligned} P(A\cap B_i) &= P(B_i)P(A|B_i) \\ P(B_i\cap A) &= P(A)P(B_i|A) \end{aligned}\right\} P(A\cap B_i) = P(B_i\cap A)}$$

$$\downarrow$$

$$P(B_i)P(A|B_i) = P(A)P(B_i|A)$$

$$P(B_i|A) = \frac{P(B_i)P(A|B_i)}{P(A)}$$

$$P(B_i|A) = \frac{P(B_i)P(A|B_i)}{P(A)} = \frac{P(B_i)P(A|B_i)}{P(B_1)P(A|B_1) + \cdots + P(B_i)P(A|B_i)} = \frac{P(B_i)P(A|B_i)}{\sum_{i=1}^{n} P(B_i)P(A|B_i)}$$

Additional info:

- Visual explanation of Bayes' theorem

b) **Bayes' theorem. Continuous case**

In case of two continuous RVs X and Y, we have:

$$f(x|y) = \frac{f(y|x)f(x)}{f(y)}$$

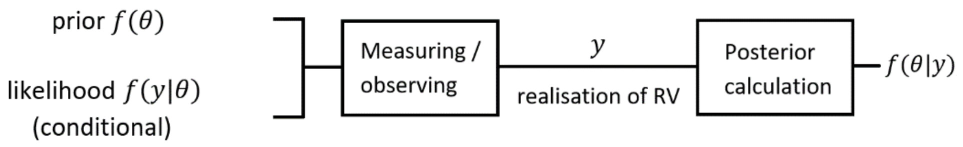
- $f(y|x)$ – the **likelihood**;
- $f(x)$ – the **prior** distribution. So, X is observed on its own – a marginal density function;
- $f(y)$ – the **evidence**. It's another marginal density function, which is usually difficult to estimate, although for a number of problems it is unnecessary;
- $f(x|y)$ – the **posterior** distribution.

c) **Bayesian inference**

- We have an RV Θ with a **prior** distribution $f(\theta)$:
  - This prior could be known from the previous experiments. We could also assume it based on knowledge that we have;
  - For example, what is the distribution of weights $f(w)$ in a NN? If we assume that all of them are equally likely, $f(w)$ will be uniform.

- We have another RV Y which is conditioned on Θ: Y given $\Theta = \theta$. $f(y|\theta)$ is called the **likelihood**;
- Now, we are interested in updating our **prior** $f(\theta)$ using the **likelihood** $f(y|\theta)$ to obtain $f(\theta|y)$, which is the **posterior** distribution:

$$f(\theta|y) = \frac{f(y|\theta)f(\theta)}{f(y)}$$



Before you gather data, Θ follows the **prior** distribution $f(\theta)$. Then, you combine it with information from the obtained data, i.e. the **likelihood** $f(y|\theta)$. So, in Bayesian inference Θ is an RV in contrast to MLE. That is why we also consider the distribution of Θ (**prior**) and not just the **likelihood** $f(y|\theta)$.

For example, we may have a good idea of a possible range for θ and could assign a prior distribution that pushes the θ slightly toward this range of values.

It can be also useful to rewrite Bayes formula in the following way:

$$f(model \mid new\ data) = \frac{f(new\ data \mid model) f(model)}{f(new\ data)}$$

$$f(model \mid new\ data) \propto f(new\ data \mid model) f(model)$$

EXAMPLE #1: given the following probability densities for a feature value x for two classes $c_1$and c_2 with equal priors $p(c_1) = p(c_2)$:

$$p(x|c_1) = \begin{cases} a, & 0 \le x \le 2 \\ 0, & \text{otherwise} \end{cases}; \; p(x|c_2) = \begin{cases} b, & 1.5 \le x \le 2.5 \\ 0, & \text{otherwise} \end{cases}$$

What is the optimal Bayesian classification accuracy that can be achieved?

Solution:

- Relative frequencies of c_1 and c_2 are equal, thus $P(c_1) = P(c_2) = 0.5$ (the number of observations for each class should be roughly equal)
- Since $\int_a^b p(x|y)dx = 1$ then:

$$\int_0^2 adx = 1 \to a = 0.5; \int_{1.5}^{2.5} bdx = 1 \to b = 1$$

- Thus, conditional PDFs:

$$p(x|c_1) = \begin{cases} 0.5, & 0 \le x \le 2 \\ 0, & \text{otherwise} \end{cases}; \; p(x|c_2) = \begin{cases} 1, & 1.5 \le x \le 2.5 \\ 0, & \text{otherwise} \end{cases}$$

- Class probabilities (given data):

$$\boldsymbol{P}(c_1|x) = \frac{p(x|c_1)\boldsymbol{P}(c_1)}{p(x)} = \frac{p(x|c_1)\boldsymbol{P}(c_1)}{\sum_{c_i} p(x|c_i)P(c_i)} = \frac{0.5 \cdot 0.5}{0.5 \cdot 0.5 + 1 \cdot 0.5} = \frac{0.25}{0.75} = \frac{1}{3}$$

$$\boldsymbol{P}(c_2|x) = 1 - \boldsymbol{P}(c_1|x) = \frac{2}{3}$$

- The decision rule is the following: for a given x if $\boldsymbol{P}(c_1|x) > \boldsymbol{P}(c_2|x)$, we classify this sample as c_1 and vice versa:

a) For $0 \leq x \leq 1.5$: we predict c_1 with 100% accuracy

b) For $1.5 < x \leq 2$: c_2 is more likely, thus, we will predict every observation as c_2, which will lead to incorrectly classifying c_1 in x% of the cases

c) For $2 \leq x \leq 2.5$: we predict c_2 with 100% accuracy

- Let's find the error rate for (b). For that, we will have to consider how many observations of class c_1 we will misclassify, which depends on the distribution $p(x) = \sum_{c_i} p(x|c_i)P(c_i)$:

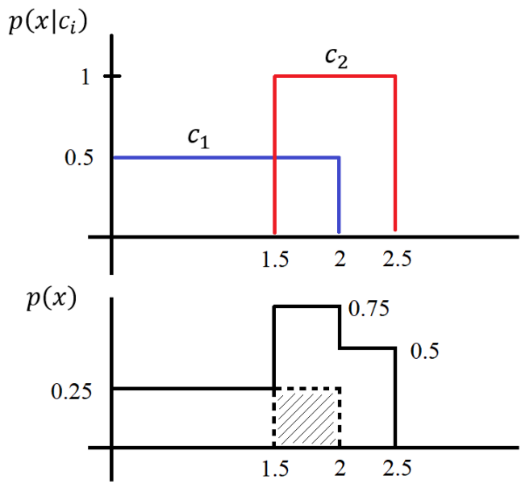
$$p(x) = \begin{cases} \sum_{c_i} p(x|c_i)P(c_i) = 0.5 \cdot 0.5, & 0 \leq x \leq 1.5 \\ \sum_{c_i} p(x|c_i)P(c_i) = 0.5 \cdot 0.5 + 1 \cdot 0.5, & 1.5 < x \leq 2 \\ \sum_{c_i} p(x|c_i)P(c_i) = 1 \cdot 0.5, & 2 < x \leq 2.5 \\ 0, & otherwise \end{cases} = \begin{cases} 0.25, & 0 \leq x \leq 1.5 \\ 0.75, & 1.5 < x \leq 2 \\ 0.5, & 2 < x \leq 2.5 \\ 0, & otherwise \end{cases}$$

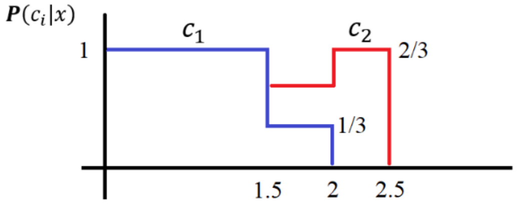
Since for $1.5 < x \leq 2$ we have a mix of classes, and c_1 is "sacrificed" in favour of c_2, thus, the error rate is:

$$\frac{p(x|c_1)P(c_1)}{\sum_{c_i} p(x|c_i)P(c_i)} \int_{1.5}^{2} p(x)dx = \frac{0.5 \cdot 0.5}{0.5 \cdot 0.5 + 1 \cdot 0.5} \int_{1.5}^{2} 0.75dx = \frac{1}{3} \cdot (0.75 \cdot 0.5) = 0.125$$

Which is equivalent to $\int_{1.5}^{2} 0.25dx = 0.25 \cdot 0.5 = 0.125$

$$Accuracy = 1 - 0.125 = 0.875$$





EXAMPLE #2: given the following probability densities for a feature value x for two classes $c_1$and c_2 with unequal priors $p(c_2) = 2\,p(c_1)$:

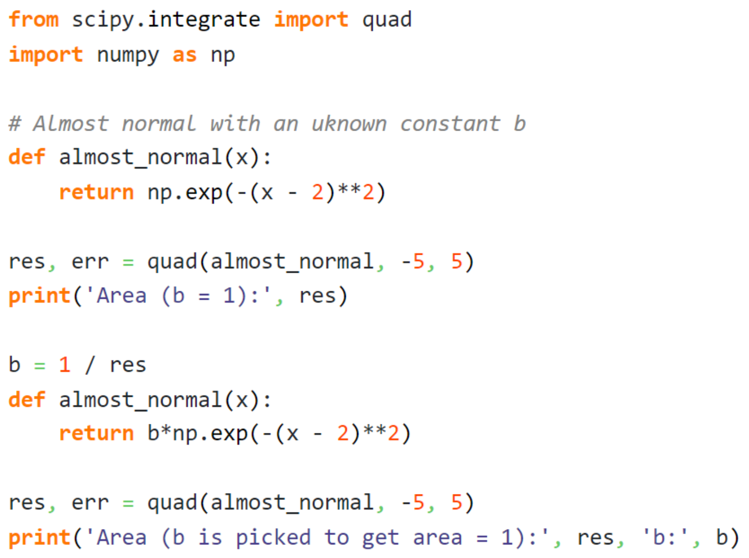
$$p(x|c_1) = \begin{cases} a, & 0 \le x \le 2 \\ 0, & \text{otherwise} \end{cases};\ p(x|c_2) = \ b\,e^{-(x-2)^2}$$

What is the optimal Bayesian classification accuracy that can be achieved?

- Relative frequencies of c_1 and c_2 are proportional, thus $P(c_1) = 2P(c_2)$. So, $P(c_1) = \frac{1}{3}$ and $P(c_2) = \frac{2}{3}$ (while sampling, we'll get a roughly 33 to 66 ratio of observations for each class respectively)
- Since $\int_a^b p(x|y)dx = 1$ then:

$$\int_0^2 adx = 1 \rightarrow a = 0.5; \int_{-\infty}^{\infty} be^{-(x-2)^2}dx = 1 \rightarrow b = 0.56$$

While integrating over $be^{-(x-2)^2}$, I used $\pm 5\sigma$ where $\sigma = 1$ as the integral's boundaries:



```
from scipy.integrate import quad
import numpy as np

# Almost normal with an uknown constant b
def almost_normal(x):
    return np.exp(-(x - 2)**2)

res, err = quad(almost_normal, -5, 5)
print('Area (b = 1):', res)

b = 1 / res
def almost_normal(x):
    return b*np.exp(-(x - 2)**2)

res, err = quad(almost_normal, -5, 5)
print('Area (b is picked to get area = 1):', res, 'b:', b)
```

PDF	Area	Parameter
$be^{-(x-2)^2}$	1.772	$b = 1$
$be^{-(x-2)^2}$	1	$b = 0.564$
$\frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$	1	$-$

- Thus, conditional PDFs:

$$p(x|c_1) = \begin{cases} 0.5, & 0 \leq x \leq 2 \\ 0, & \text{otherwise} \end{cases}; \ p(x|c_2) = \ 0.56e^{-(x-2)^2}$$

- Class probabilities (given data):

$$\boldsymbol{P}(c_1|x) = \frac{p(x|c_1)\boldsymbol{P}(c_1)}{p(x)} = \frac{p(x|c_1)\boldsymbol{P}(c_1)}{\sum_{c_i} p(x|c_i)P(c_i)} = \frac{0.5 \cdot \frac{1}{3}}{0.5 \cdot \frac{1}{3} + 0.56e^{-(x-2)^2} \cdot \frac{2}{3}} = \frac{0.167}{0.37e^{-(x-2)^2} + 0.167} = \frac{1}{2.26e^{-(x-2)^2} + 1}$$

$$\boldsymbol{P}(c_2|x) = 1 - \boldsymbol{P}(c_1|x) = \frac{2.26e^{-(x-2)^2}}{2.26e^{-(x-2)^2} + 1}$$

- The decision rule is the following: if $\frac{\boldsymbol{P}(c_1|x)}{\boldsymbol{P}(c_2|x)} > 1$ then c_1 else c_2. In our case, $\frac{1}{2.26e^{-(x-2)^2}} > 1$, so:

    a) For $\frac{1}{2.26e^{-(x-2)^2}} > 1$ to hold true x should be $x < 1.1$ or $x > 2.9$

    b) We should also take into account the following constraint for c_1: $0 \leq x \leq 2$

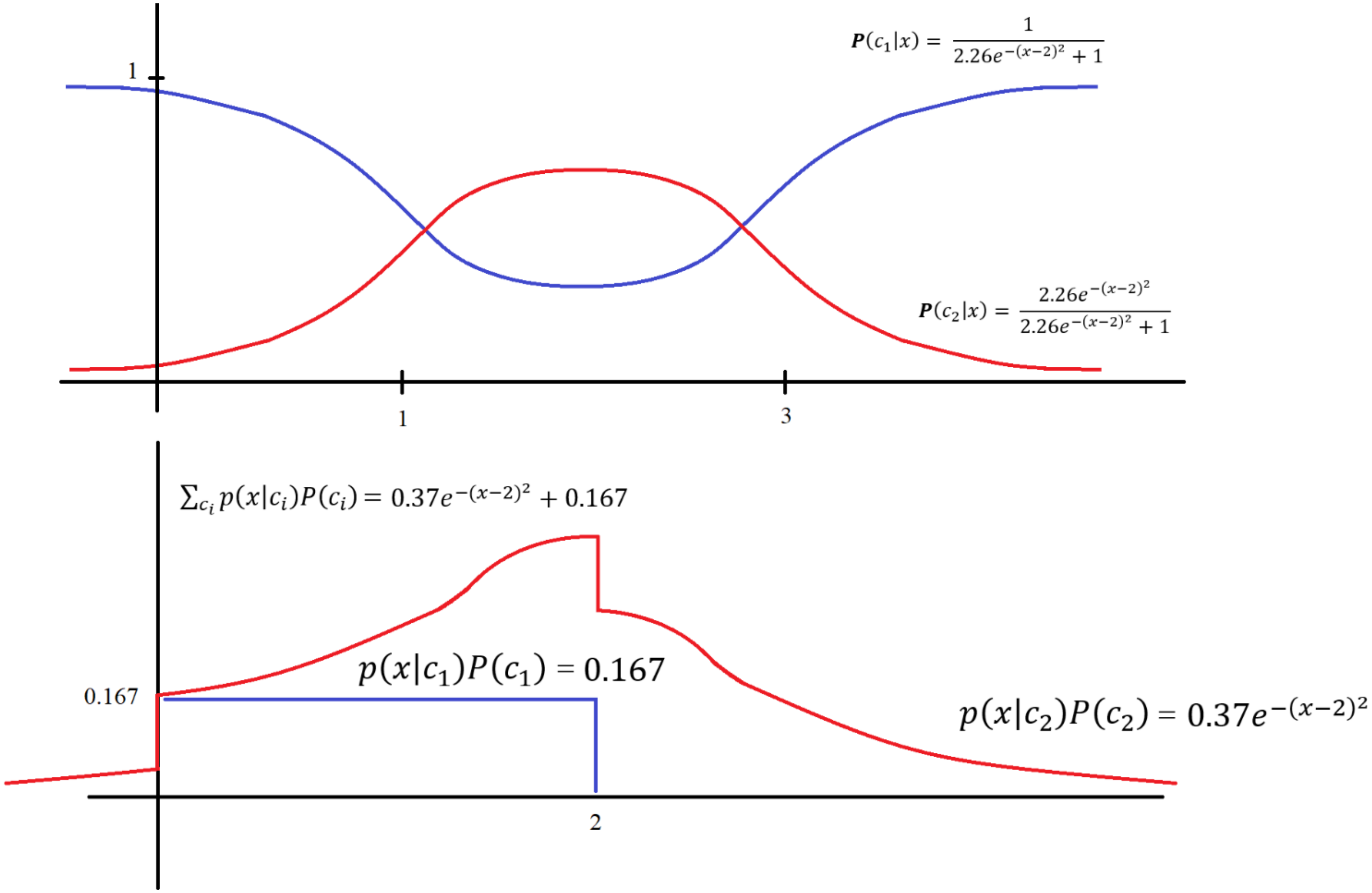
- The error rate of c_2 for $0 < x < 1.1$ is:

$$\int_0^{1.1} p(x|c_2)P(c_2)dx = \int_0^{1.1} 0.37e^{-(x-2)^2} dx = 0.0656$$

- The error rate of c_1 for $1.1 \leq x < 2$ is:

$$\int_{1.1}^{2} p(x|c_1)P(c_1)dx = \int_{1.1}^{2} 0.167dx = 0.1499$$

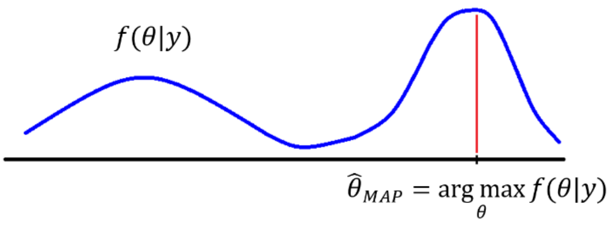
$$Accuracy = 1 - 0.0656 - 0.1837 = 0.7845$$


$P(c_1|x) = \frac{1}{2.26e^{-(x-2)^2} + 1}$
1
$P(c_2|x) = \frac{2.26e^{-(x-2)^2}}{2.26e^{-(x-2)^2} + 1}$
1
3
$\sum_{c_i} p(x|c_i)P(c_i) = 0.37e^{-(x-2)^2} + 0.167$
$p(x|c_1)P(c_1) = 0.167$
0.167
$p(x|c_2)P(c_2) = 0.37e^{-(x-2)^2}$
2

d) **Maximum a posteriori estimation**

After getting the **posterior** distribution $f(\theta|y)$, we may want to make a point estimate. It's done via the **maximum a posteriori** (MAP) estimation. We choose such θ so that $f(\theta|y)$ is maximised:

$$\hat{\theta}_{MAP} = \arg\max_{\theta} f(\theta|y) = \arg\max_{\theta} \frac{f(y|\theta)f(\theta)}{f(y)} = \arg\max_{\theta} f(y|\theta)f(\theta)$$



Since we want to maximise $f(\theta|y) = \frac{f(y|\theta)f(\theta)}{f(y)}$ and $f(y)$ is positive and doesn't depend on θ, we can safely ignore it.

e) Maximum likelihood estimation

Given a probabilistic **model** with **parameters** θ and the observed **data** y $(y|x)$, we want to find such θ so that the joint density of data as a function of θ is maximised. In other words, under the assumed model the observed data must be most probable. Likelihood is a measure of the extent to which observed data provides support for particular values of a parameter in a parametric model.

We have IDD RVs $Y_1, Y_2, \dots, Y_n$, and we collect observations (realisations) $y_1, y_2, \dots, y_n$ drawn from a parametric probability distribution f with unknown parameters θ – this is our probabilistic model. We define the joint PDF, and call it the **likelihood**:

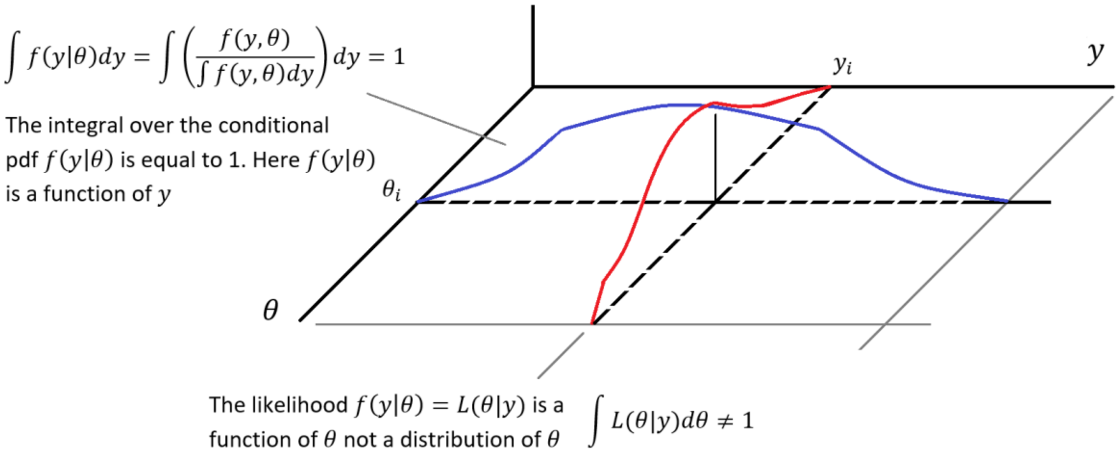
$$L(\theta|y) = f(y|\theta) = f(y;\theta) = \prod_{i=1}^{n} f(y_i|\theta)$$

Here, θ is not an RV. It is an unknown parameter. So, θ doesn't have any probabilistic meaning assigned to it – we do not incorporate any assumptions about the distribution of θ in our calculations. The likelihood function $f(y|\theta)$ is frequently written as $f(y;\theta)$ or $f_\theta(y)$ to indicate that θ is not an RV.

Turning back to Bayes theorem $f(\theta|y) = \frac{f(y|\theta)f(\theta)}{f(y)}$. There are two roles that $f(y|\theta)$ plays:

- $f(y|\theta)$ is a (conditional) probability distribution of y, i.e. $\int f(y|\theta)dy = 1$;
- $f(y|\theta)$ is just a function of θ, i.e. $\int f(y|\theta)d\theta \neq 1$, which is the **likelihood** $L(\theta|y)$.

Once data y is observed, it's "fixed". Thus, $f(y|\theta)$ becomes the **likelihood** when we calculate the **posterior**. And that is why it is equivalent to $L(\theta|y) = f(y|\theta)$ in MLE.


$\int f(y|\theta)dy = \int \left(\frac{f(y,\theta)}{\int f(y,\theta)dy}\right)dy = 1$
The integral over the conditional pdf $f(y|\theta)$ is equal to 1. Here $f(y|\theta)$ is a function of y
y_i
y
θ_i
θ
The likelihood $f(y|\theta) = L(\theta|y)$ is a function of θ not a distribution of θ
$\int L(\theta|y)d\theta \neq 1$

Instead of $L(\theta|y)$ the loglikelihood is normally used $l(\theta|y) = \ln L(\theta|y)$. Since logarithms are strictly increasing functions, maximising $l(\theta|y)$ is equivalent to maximising $L(\theta|y)$. Taking the logarithm has a number of useful properties, for example $\ln(\prod_{i=1}^{n} f(y_i|\theta)) = \sum_{i=1}^{n} f(y_i|\theta)$.

Finally, we choose such θ so that $l(\theta|y)$ is maximised:

$$\hat{\theta}_{MLE} = \arg\max_{\theta} l(\theta|y)$$

MLE and MAP:

- If the prior is uniform, then $f(\theta) = const$ and $\hat{\theta}_{MAP} = \hat{\theta}_{MLE}$:

$$\hat{\theta}_{MAP} = \arg\max_{\theta} f(y|\theta)f(\theta) = \arg\max_{\theta} f(y|\theta)\,; \; \hat{\theta}_{MLE} = \arg\max_{\theta} l(\theta|y)$$

- If the prior is normally distributed with $\mu = 0$ and σ^2 (we want to put constraints on our parameters making them smaller, i.e. on average they should be around $\mu = 0$), then $f(\theta) = N(0, \sigma^2)$ and:

$$\hat{\theta}_{MAP} = \arg\max_{\theta} f(y|\theta)N(0,\sigma^2) \;\; = \arg\max_{\theta} \log\big(f(y|\theta)\big) + \log\left(N(0,\sigma^2)\right)$$

Where $\log\left(N(0,\sigma^2)\right) = -\frac{1}{2\sigma^2}\sum w^2$, which is L2 regularisation. This expression is normally parametrized by some value λ which controls whether the parameter estimation is driven by the likelihood or the prior.

Additional info:

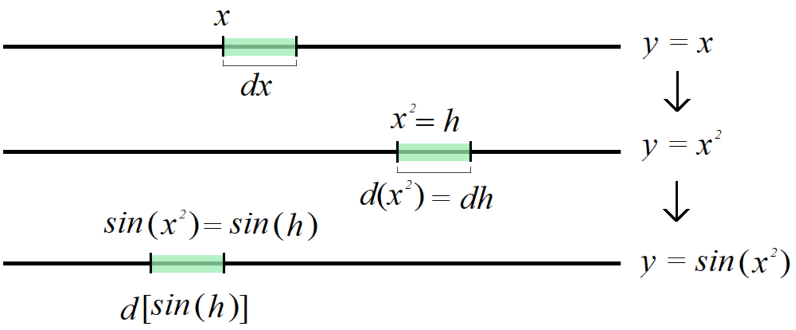
- Bayesian inference by MIT
- MAP. Introduction to Probability, Statistics, and Random Processes
- Wiki: MAP estimation
- MLE vs MAP. Introduction to Probability, Statistics, and Random Processes
- Bayes theorem and MLE #1
- Bayes theorem and MLE #2
- Bayes theorem and MLE #3
- MLE vs MAP vs EM
- Likelihood vs probability

II. Derivatives

a) **Derivative of a composite function**

If we have the following function $y = f(z)$, where $z = g(x)$, then:

$$y' = f'(z)g'(x) = \frac{dy}{dz}\frac{dz}{dx}$$



$$d[sin(h)] = cos(h)dh$$
$$d[sin(x^2)] = cos(x^2)d(x^2) = cos(x^2)2xdx$$

EXAMPLE: $y = (3x+2)^2$. Let's replace the inner function with $z = 3x + 2$, we get $y = (z)^2$. Thus, $\frac{dy}{dx} = \frac{d(z)^2}{dz}\frac{dz}{dx} = \frac{d(z)^2}{dz}\frac{d(3x+2)}{dx} = 2z \cdot 3 = 6(3x+2)$.

b) **Partial derivative**

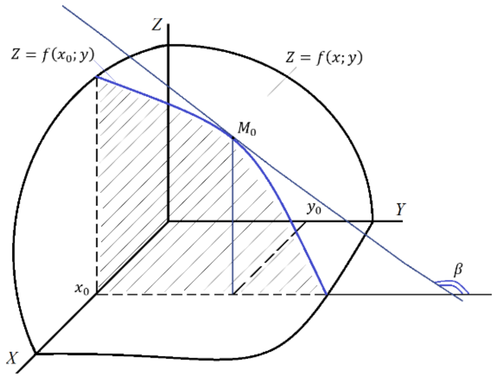
If we have a function of 2 independent variables $z = f(x, y)$, then a partial derivative with respect to x can be defined as follows:

$$f_x'(x,y) = \frac{\partial z}{\partial x} = \lim_{\Delta x \to 0} \frac{\Delta z_x}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x, y) - f(x,y)}{\Delta x}$$

A partial derivative with respect to x is the rate of change of a function f along the x-axis. All other variables are held constant, and, as a result, our function f depends only on x.

EXAMPLE: $z = 3x + 2y$. So, $\frac{\partial z}{\partial x} = (3x)' + (2y)'$, since y is a constant $\frac{\partial z}{\partial x} = 3 + 0 = 3$.

Suppose we have a function $z = f(x, y)$. Let's pick a specific value $x = x_0$ and illustrate the function $z = f(x_0, y)$:

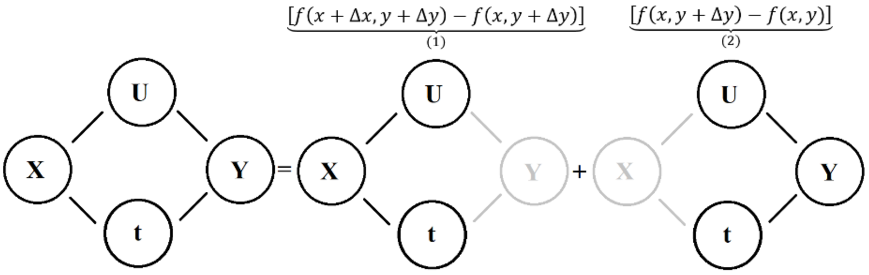


It can be seen that $z = f(x_0, y)$ depends only on y, and we can now calculate the derivative with respect to y as we normally would.

c) **Derivative of a composite function. Multivariate case**

Suppose we have a function $u = f(x, y)$ and variables x and y depend on t. Then, the derivative with respect to t is equal to:

$$\frac{du}{dt} = \frac{\partial u}{\partial x}\frac{dx}{dt} + \frac{\partial u}{\partial y}\frac{dy}{dt}$$


$\underbrace{[f(x+\Delta x, y+\Delta y) - f(x, y+\Delta y)]}_{(1)}$
$\underbrace{[f(x, y+\Delta y) - f(x, y)]}_{(2)}$
U
X
Y
t
=
U
X
Y
t
+
U
X
Y
t

Formula is derived based on the Lagrange's theorem [a].

e) **Directional derivative**

If we want to explore the rate of change of a function in any given direction rather than along a specific axis (x-axis, for example), we will need to introduce the concept of a **directional derivative** which is built on the idea of **partial derivatives**. If we have a function of 2 independent variables $z = f(x, y)$, then the directional derivative along the direction $\overline{u}$ is:

$$\nabla_{\overline{u}} f(x, y) = \frac{\partial z}{\partial x} u_1 + \frac{\partial z}{\partial y} u_2$$

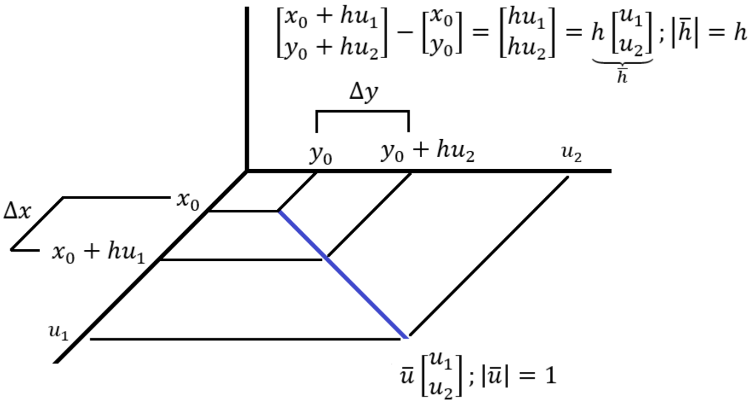
In this case, $\bar{u} = (u_1, u_2)$ is a unit vector $|\bar{u}| = 1$ that determines the direction of our derivative.

We choose a direction $\bar{u} = (u_1, u_2)$, where $|\bar{u}| = 1$ (a unit vector) because we only use it to determine the direction in which we are going []. We start at some point $M(x_0, y_0)$ and go in the direction of $\bar{u}$. As a result, we can parametrise independent variables:

$$x(h) = x_0 + hu_1;\ y(h) = y_0 + hu_2$$

By changing h we move from the point $M(x_0, y_0)$ along the $\bar{u}$ direction. Thus, just like with ordinary derivatives, we can use the limit definition:

$$\nabla_{\bar{u}} f(x, y) = \lim_{h \to 0} \frac{f(x + hu_1, y + hu_2) - f(x, y)}{h}$$


$\begin{bmatrix} x_0 + hu_1 \\ y_0 + hu_2 \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} hu_1 \\ hu_2 \end{bmatrix} = h \underbrace{\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}}_{\bar{h}}; |\bar{h}| = h$
Δy
y_0
$y_0 + hu_2$
u_2
Δx
x_0
$x_0 + hu_1$
u_1
$\bar{u} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}; |\bar{u}| = 1$

However, in order to compute this derivative, a little more work is needed. We know that $z = f(x, y)$ depends on x and y that, in turn, depend on h. Consequently, $z = f(x, y)$ depends on h and is a composite function. The derivative of a composite function of multiple variables (2 in our case) is equal to:

$$\frac{dz}{dh} = \frac{\partial z}{\partial x}\frac{dx}{dh} + \frac{\partial z}{\partial y}\frac{dy}{dh}$$

$$\frac{dx}{dh} = (x_0 + hu_1)' = u_1;\ \frac{dy}{dh} = (y_0 + hu_2)' = u_2$$

$$\frac{dz}{dh} = \nabla_{\bar{u}} f(x, y) = \frac{\partial z}{\partial x} u_1 + \frac{\partial z}{\partial y} u_2$$

Additional info:

- Wiki: what is directional derivative?
- Video lecture by Dr. Trefor Bazett
- Video lecture by Prof. Leonard

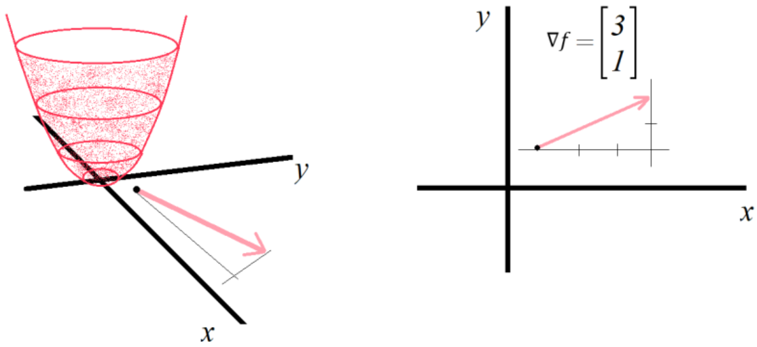
f) **Gradient**

It is a vector that shows the direction of the fastest increase:

$$\nabla f = \nabla f(w_1, w_2, \dots w_n) = \left[\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \dots, \frac{\partial f}{\partial w_n}\right]^T$$

For instance, $\nabla f(1, 1, \dots, 1) = [0.3, 2, \dots, -0.5]^T$:

- $\frac{\partial f}{\partial w_2}$ is more important than both $\frac{\partial f}{\partial w_1}$ and $\frac{\partial f}{\partial w_n}$, while $\frac{\partial f}{\partial w_n}$ is more important than $\frac{\partial f}{\partial w_1}$;
- Standing at the input $(1, 1, \dots, 1)$ and moving along this direction ∇f, increases the function $f(\dots)$ most quickly, but, on top of that, changes to the variable w_2 is more important than changes to the variable w_1 ($2 > 0.3$), at least in the neighbourhood of the input;
- Calculating a directional derivative with respect to the direction $\bar{u}$, we weigh $\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \dots, \frac{\partial f}{\partial w_n}$ based on $u_1, u_2, \dots, u_n$. If, however, $\bar{u}$ points in the direction of ∇f, we weigh $\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \dots, \frac{\partial f}{\partial w_n}$ in the "best" way possible, that is we give them weights according to their contributions to the rate of change of $f(\dots)$ – the larger the contribution, the greater the weight:



Also, if $\bar{u} = \nabla f$, a directional derivative will be equal to the magnitude of ∇f:

$$\nabla_{\bar{u}} f(x, y) = |\nabla f|$$

Earlier, we defined a directional derivative in the following way:

$$\nabla_{\bar{u}} f(x, y) = \frac{\partial z}{\partial x} u_1 + \frac{\partial z}{\partial y} u_2$$

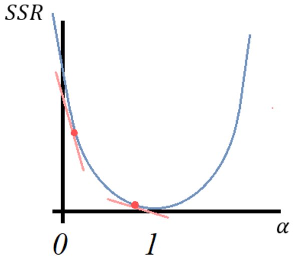
And $\bar{u} = (u_1, u_2)$ is a unit vector pointing in any given direction. Since $\nabla_{\bar{u}} f(x, y)$ is a dot product of two vectors (u_1, u_2) and $\nabla f = \left(\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}\right)$, so we can rewrite it as follows:

$$\nabla_{\bar{u}} f(x, y) = |\nabla f| \cdot |\bar{u}| \cdot \cos\alpha = |\nabla f| \cdot \cos\alpha$$

So, what will happen if we try maximising this quantity? The maximum value of $\cos \alpha = 1$, which is the case when $\alpha = 0$, in other words, when the angle between ∇f and $\bar{u}$ is 0 – they point in the same direction. Thus, $\nabla_{\bar{u}} f(x, y)$ is maximised when $\bar{u} = \nabla f$.

Important: it's crucial to remember that $\frac{dy}{dx}$ doesn't just show us the rate of change, it also tells us whether y is decreasing or increasing, in other words the direction in which the function is changing. So, not only do we know how fast we move but also what is the trajectory of our movement.

EXAMPLE #1:

№	Step	Example	Note
1	Define a loss function. For example: $SSR = \sum [y_i - f(x_i;\ \theta_1, \theta_2, \dots, \theta_n)]^2$	$f(x_i;\ \theta_1, \theta_2, \dots, \theta_n) = \alpha + 0.64x_i$	The function that predicts y_i is $\alpha + 0.64x_i$. It has one parameter α that we need to optimise
2	Find the gradient. In this case, it's just a partial derivative $\frac{\partial\ SSR}{\partial \alpha}$	$\frac{\partial\ SSR}{\partial \alpha} = \sum -2[y_i - (\alpha + 0.64x_i)]$ Let's say $\{y_i; x_i\} = \{(1.4, 0.5); (1.9, 2.3); (3.2, 2.9)\}$. So $\frac{\partial\ SSR}{\partial \alpha} = 3\alpha - 5.7$. We found ∇SSR for this dataset and can now optimise α	If the value of $\frac{\partial\ SSR}{\partial \alpha}$ is large, then the rate of change is large, then the slope is steep, and, as a result, we are still far away from the local extremum of a function. The closer we are to the local extremum, the closer the $\frac{\partial\ SSR}{\partial \alpha}$ is to 0, and we need to take smaller steps in order to not miss the target (that is why the size of a step should be related to the slope)
			
3	Initialise your guess of α	$\alpha = 0;\ \frac{\partial\ SSR}{\partial \alpha}(0) = -5.7$	
4	Figure out the size of the step: $step_{size_\alpha} = l_{rate} \cdot \frac{\partial\ SSR}{\partial \alpha}(0)$	If $l_{rate} = 0.1$: $step\_size_\alpha = 0.1 \cdot (-5.7) = -0.57$	by how much you are going to change your parameter α to get a better value that will make us closer to the local extremum

5	Now, we can calculate a new value for our parameter $\alpha_{new} = \alpha - step\_size_{\alpha}$	$\alpha_{new} = 0 - (-0.57) = 0.57$	
6	We got our new value for α, and we can find the ∇SSR in this point	$\frac{\partial\, SSR}{\partial \alpha}(0.57) = -2.3$	The absolute value of the ∇SSR became smaller, so we got closer to the local minimum
7	Calculate: $step\_size_{\alpha}$, α_{new}, ∇SSR	$step\_size_{\alpha} = 0.1 \cdot (-2.3) = -0.23$ $\alpha_{new} = 0.57 - (-0.23) = 0.8$ $\frac{\partial\, SSR}{\partial \alpha}(0.8) = \cdots$	As ∇SSR is getting smaller, $step\_size_{\alpha}$ is also getting smaller. When it’s close to 0, the algorithm stops

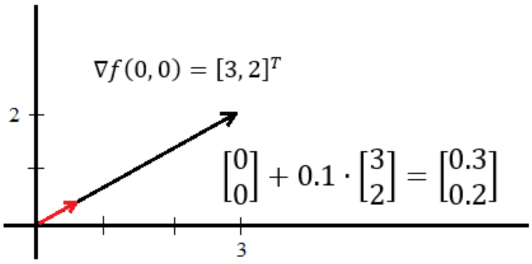
EXAMPLE #2:

- Define a loss function. For example, $SSR = \sum[y_i - f(x_i;\ \theta_1, \theta_2, \dots, \theta_n)]^2$, let's say that $f(x_i;\ \theta_1, \theta_2, \dots, \theta_n) = \alpha + \beta x_i$;
- Find both partial derivatives:

$$\begin{cases} \dfrac{\partial\, SSR}{\partial \alpha} = \sum -2[y_i - (\alpha + \beta x_i)] \\ \dfrac{\partial\, SSR}{\partial \beta} = \sum -2x_i[y_i - (\alpha + \beta x_i)] \end{cases}$$

- We pick α and β randomly ($\alpha = 0$ and $\beta = 1$) and calculate $\frac{\partial\, SSR}{\partial \alpha}(0,1) = -1.6$ and $\frac{\partial\, SSR}{\partial \beta}(0,1) = -0.8$, getting $\nabla f(0,1) = [-1.6, -0.8]^T$;
- Calculate $step\_size_\alpha = 0.01 \cdot (-1.6) = -0.016$ and $step\_size_\beta = 0.01 \cdot (-0.8) = -0.008$. The process is repeated…

Why do we multiply by the learning rate?


$\nabla f(0,0) = [3,2]^T$
2
3
$\begin{bmatrix}0\\0\end{bmatrix} + 0.1 \cdot \begin{bmatrix}3\\2\end{bmatrix} = \begin{bmatrix}0.3\\0.2\end{bmatrix}$

Additional info:

- Video by 3Blue1Brown – what is gradient descent?
- Video by StatQuest – what is gradient descent?
- Why do we multiply by the learning rate?

III. Integrals (1 variable)

a) **Antiderivative (primitive)** – a function F is an antiderivative of a function f if $F'(x) = f(x)$.

EXAMPLE: $f(x) = 2x$. What is the antiderivative $(?)' = 2x$? $F(x) = x^2$.

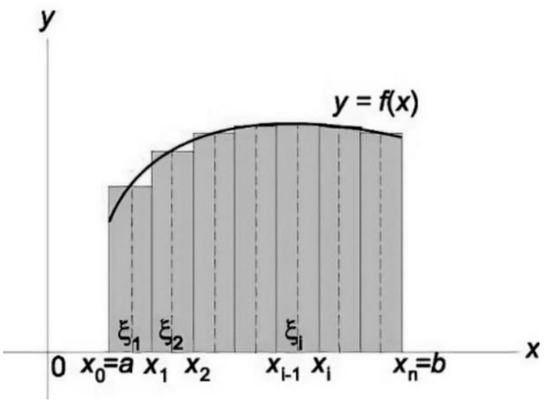
b) **Indefinite integral** – a set of all antiderivatives of f: $F(x) + C$, where C is an arbitrary constant. This definition is not the only one. Later, when explaining the connection between a definite and indefinite integral, it will be shown that an integral with the variable upper limit is also an antiderivative (primitive).

$$\int f(x)dx = F(x) + C$$

Why do we add a constant C?

$$[F(x) + C]' = F'(x) + 0 = f(x)$$

c) **Definite integral** – informally, it is the sum of infinitely many (infinitely small) rectangles or stripes. This sum represents the area under a curve.



Given a function $f(x)$ that is continuous on the interval $[a, b]$ we divide the interval into n subintervals of the width, Δx, and within each interval choose a point ξ_i:

$$\int_a^b f(x)dx = \lim_{\lambda \to 0} \sum_{i=0}^{n-1} f(\xi_i)\Delta x_i$$

Additional info:

- Indefinite integrals by Utexas.edu
- The connection between definite and indefinite integrals
- Definite integral by Utexas.edu

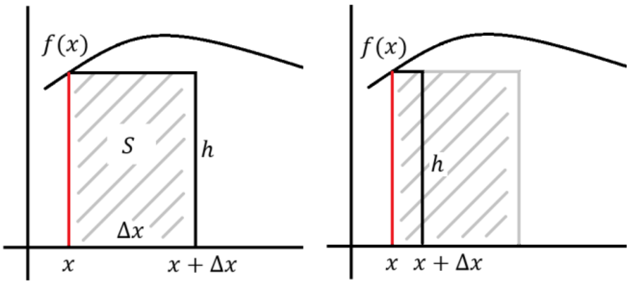
d) **Newton-Leibniz theorem. Fundamental theorem of calculous** – theorem that links a definite integral to antiderivatives:

$$\int_a^b f(t)dt = F(b) - F(a)$$

An integral with a variable upper limit, can be seen as a function of x – an area under the curve that changes based on x:

$$F(x) = S_{a,x}(x) = \int_a^x f(t)dt$$

When the area of a rectangle is divided by its base, we get the height. In case of $\frac{\Delta S_{a,x}}{\Delta x}$ we will get the height h of a rectangle, and this height will get closer and closer to $f(x)$ as $\Delta x \to \infty$:



$$f(x) = \lim_{\Delta x \to 0} \frac{\Delta S_{a,x}}{\Delta x};\ S_{a,x}{}'(x) = F'(x) = f(x)$$

Consequently, an integral with a variable upper limit is one of the possible antiderivatives (primitives) of $f(x)$. We got one $F(x)$, but we are interested in a set $F(x) + C$. Suppose $\Phi(x)$ is any given antiderivative from this set:

$$\Phi(x) = F(x) + C;\ \Phi(x) = \int_a^x f(t)dt + C$$

Let's consider the following edge cases:

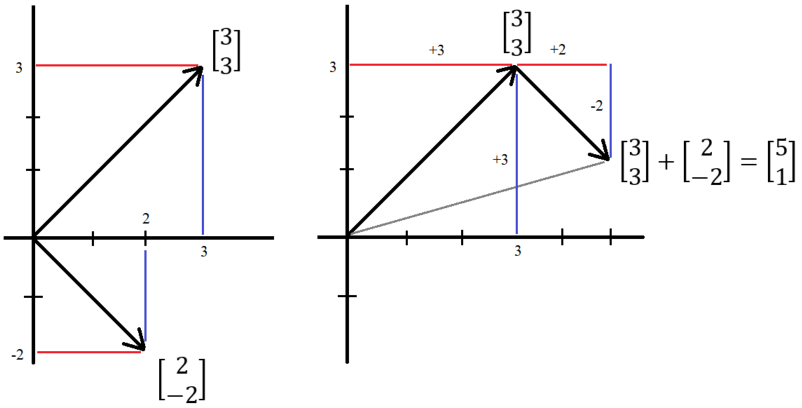
$$\Phi(a) = \int_a^a f(t)dt + C;\ \Phi(a) = C;\ \Phi(b) = \int_a^b f(t)dt + \mathrm{C};\ \int_a^b f(t)dt = \Phi(b) - \Phi(a)$$

IV. Vectors. Matrices

a) **Vector addition** – a new vector whose coordinates are equal to the sum of corresponding components of summed vectors:

$$\begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

There are plenty of real-word examples that showcase why should we add vectors in such a way. When two (or more) physical forces act at one point, it's not enough to take into account only their magnitude – the direction matters as well:



3
$\begin{bmatrix} 3 \\ 3 \end{bmatrix}$
2
3
-2
$\begin{bmatrix} 2 \\ -2 \end{bmatrix}$
+3
$\begin{bmatrix} 3 \\ 3 \end{bmatrix}$
+2
3
-2
+3
$\begin{bmatrix} 3 \\ 3 \end{bmatrix} + \begin{bmatrix} 2 \\ -2 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$
3

If move in the direction of $\begin{bmatrix} 3 \\ 3 \end{bmatrix}$ and $\begin{bmatrix} 2 \\ -2 \end{bmatrix}$ at the same time, you will end up at the point $\begin{bmatrix} 5 \\ 1 \end{bmatrix}$. Thus, simultaneous movement can be expressed as a consecutive one: we first move in the direction of $\begin{bmatrix} 3 \\ 3 \end{bmatrix}$, when we get there, we proceed by going 2 units along the x-axis and -2 units along the y-axis.

b) **Multiplying a vector by a scalar** – a new vector whose components are multiplied by a given scalar:

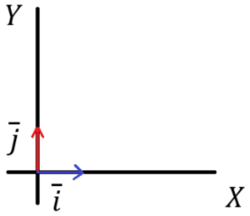
$$\begin{bmatrix} \lambda a_1 \\ \lambda a_2 \end{bmatrix} = \lambda \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

There are a lot of scenarios that make this definition useful. For instance, velocity is a vector that has a magnitude and a direction. If you want to move 3 times as fast, you can multiply its magnitude by 3.

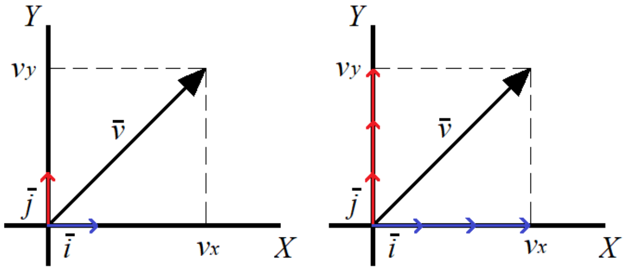
c) **Basis** – a set of linearly independent vectors that spans (allows us to create) a vector space.

For example, in a 2-dimensional space (R^2) we need 2 linearly independent vectors to "build everything else", i.e. construct any other vector in this space that we want. If vectors are not linearly independent, we basically won't have enough information to explore the entire vector space.

Let's consider a plane and unit vectors $\bar{\imath}$ and $\bar{\jmath}$:



Using these 2 vectors we can create any other vector in this space $\bar{v} = v_1\bar{\imath} + v_2\bar{\jmath}$. Because we have only two dimensions to explore, having tow linearly independent vectors is enough:



We first scale vectors $\bar{\imath}$ and $\bar{\jmath}$ by appropriate constants and then add them up. This basis is called the **standard basis** $\vec{e} = \{\bar{\imath}, \bar{\jmath}\}$.

d) **Function, transformation. Linear transformation**

Function – mapping (relating) elements of a set X to a set Y. Each element from X gets exactly one element from Y.

- **Functional notation**: $f(x) = x^2$
- **Arrow notation**: $\begin{matrix} f: \mathbb{R} \to \mathbb{R} \\ f: x \mapsto x^2 \end{matrix}$. This function has the same **domain** and **codomain** $\mathbb{R}$.

More generally, $f: X \to Y$ means f maps a set to a set (where a function operates). $f: x \mapsto y$ means that f maps an element of one set to an element of another set (what a function does). For instance:

$$\begin{matrix} f: X \to Y \\ f: x \mapsto y \end{matrix} \Rightarrow \begin{matrix} f: \{1,2,3\} \to \{4,5,6\} \\ f: 1 \mapsto 5 \end{matrix}$$

We can also map sets of different dimensionalities, i.e. $f: \mathbb{R}^2 \to \mathbb{R}^3$.

Additional info:

- Arrow notation #1
- Arrow notation #2

Vector transformation – functions that operate on vectors: T.

Since vectors are also members of sets, we can have functions that takes vectors. For instance, $T: \mathbb{R}^n \to \mathbb{R}^m$. Such functions are vector-valued. Let's consider a specific example:

$$T: \mathbb{R}^3 \to \mathbb{R}^2$$
$$T: \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \mapsto \begin{bmatrix} x_1 + 2x_2 \\ 3x_3 \end{bmatrix}; \; T\left(\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

Linear transformation – is a transformation $T: \mathbb{R}^n \to \mathbb{R}^m$ that obeys 2 rules:

$$T(\overline{a} + \overline{b}) = T(\overline{a}) + T(\overline{b})$$
$$T(\lambda\overline{a}) = \lambda T(\overline{a})$$

Here $\overline{a}, \overline{b} \in \mathbb{R}^n$.

EXAMPLE: let's consider the following transformation $T(x_1, x_2) = (x_1 + x_2, 3x_1)$. $\overline{a} = [a_1, a_2]^T$ and $\overline{b} = [b_1, b_2]^T$.

$$T(\overline{a} + \overline{b}) = T(a_1 + b_1, a_2 + b_2) = (a_1 + b_1 + a_2 + b_2, 3a_1 + 3b_1)$$
$$T(\overline{a}) = T(a_1, a_2) = (a_1 + a_2, 3a_1)$$
$$T(\overline{b}) = T(b_1, b_2) = (b_1 + b_2, 3b_1)$$
$$T(\overline{a}) + T(\overline{b}) = (a_1 + a_2, 3a_1) + (b_1 + b_2, 3b_1)$$

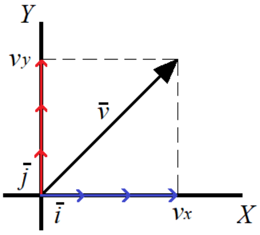
$$T(\lambda\overline{a}) = T(\lambda a_1, \lambda a_2) = (\lambda a_1 + \lambda a_2, 3\lambda a_1)$$
$$\lambda T(\overline{a}) = \lambda T(a_1, a_2) = \lambda(a_1 + a_2, 3a_1)$$

This transformation is linear.

e) **Matrix vector multiplication**

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = a \begin{bmatrix} x_{11} \\ x_{21} \end{bmatrix} + b \begin{bmatrix} x_{12} \\ x_{22} \end{bmatrix} = \begin{bmatrix} ax_{11} & bx_{12} \\ ax_{21} & bx_{22} \end{bmatrix}; \; AX = B$$

One example of this definition is the decomposition of a vector $\overline{v}$ via the standard basis $\vec{e} = \{\overline{\imath}, \overline{\jmath}\}$:



In this case, we would have $\overline{v} = 3\bar{i} + 3\bar{j}$ or alternatively $\overline{v} = 3\begin{bmatrix}1\\0\end{bmatrix} + 3\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}3\\3\end{bmatrix}$. Thus, we have:

$$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\begin{bmatrix}3\\3\end{bmatrix} = 3\begin{bmatrix}1\\0\end{bmatrix} + 3\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}3\\3\end{bmatrix}$$

Basis vectors $\bar{i}$ and $\bar{j}$ are scaled by corresponding values in a column vector and then added together. We can also think of this example as multiplying $\overline{v}$ by the identity matrix – a primitive case of a linear transformation, i.e. the vector isn't transformed.

---

f) **Matrix vector multiplication as a linear transformation**

Suppose we have a matrix $A = \underbrace{[\bar{v}_1, \bar{v}_2, \dots, \bar{v}_n]}_{m\times n}$ where $\bar{v}_i \in R^m$. We also have a vector $X \in R^n$. Now, let's perform matrix vector multiplication:

$$\underbrace{A}_{m\times n} \cdot \underbrace{X}_{n\times 1} = \underbrace{B}_{m\times 1}$$

We can see that this operation results in $\mathbb{R}^n \to \mathbb{R}^m$. Thus, it can be seen that this operation is a transformation – it takes some vector X from $\mathbb{R}^n$ and it maps it to some vector from $\mathbb{R}^m$:

$$T: \mathbb{R}^n \to \mathbb{R}^m$$
$$T(X) = \underbrace{A}_{m\times n} \cdot \underbrace{X}_{n\times 1} = \underbrace{B}_{m\times 1}$$

EXAMPLE:

…

Additional info:

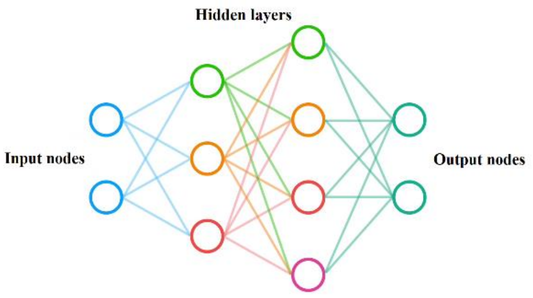
- Geometric interpretation of non-square matrices
- Intuition behind matrix multiplication #1
- Intuition behind matrix multiplication #2
- Linear algebra by Khan Academy

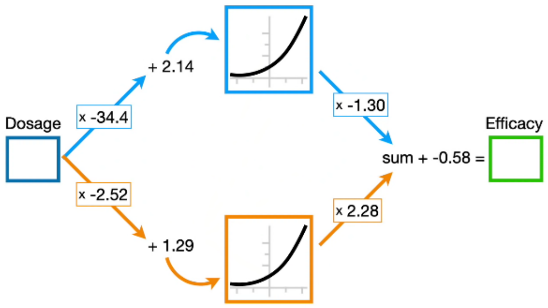
ARTIFICIAL NEURAL NETWORKS

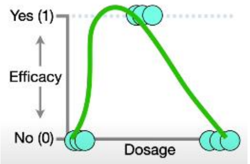
Acknowledgment

Hat tip to StatQuest and 3Blue1Brown for a great, intuitive introduction to neural networks

I. Basics. Intuition

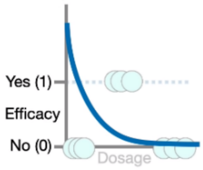


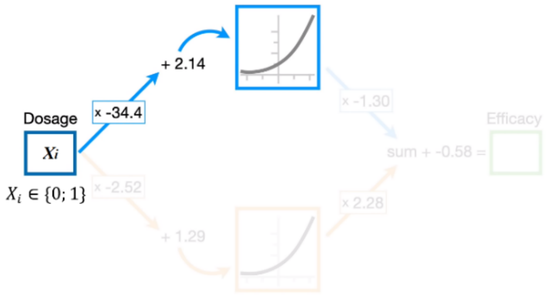




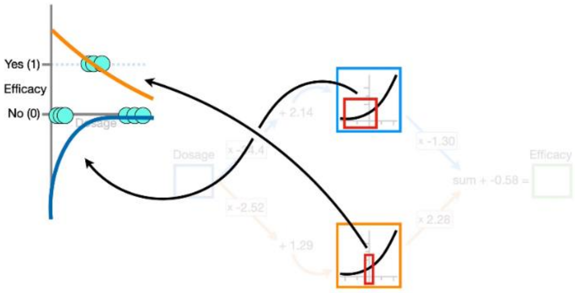
To approximate highly complex non-linear relationships between X and Y, we use:

- **Input nodes** that take x_i values as their input;
- **Weights** and **biases** that connect **input nodes** and **hidden layers** (consist of **nodes**);
- The defined **hidden layers** use the input (from the **input nodes**) that was transformed via multiplying it by **weights** and adding **biases** – x_i^{trns}. These parameters are optimised via gradient descent and **back propagation** – x_i that are most relevant to predicting y_i have larger **weights**;
- To use the transformed input x_i^{trns}, **hidden layers** have some sort of **activation function** – the building block that is necessary to approximate the complex nature of y_i – that takes the transformed input x_i^{trns} and outputs some y_i^{act} values;

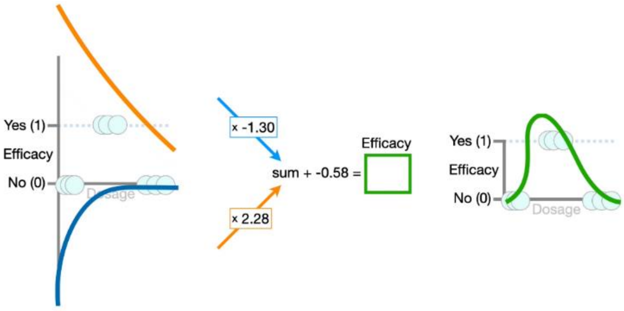




- Different **weights** and **biases** for each connection between the **input nodes** and the **hidden layers** result in each **node** in the **hidden layers** using different portions of the activation function. This creates various shapes that we can combine to fit any data we want:



- We finally scale y_i^{act} by **weights** that represent the connection between the **nodes** in the **hidden layers** and the **output nodes** and sum the results up:



This neural network starts with identical activation functions in each **node** of **hidden layers**, but the **weights** and **biases** "slice / flip / stretch" outputs of these functions to create new shapes that we later add together to get something entirely new.

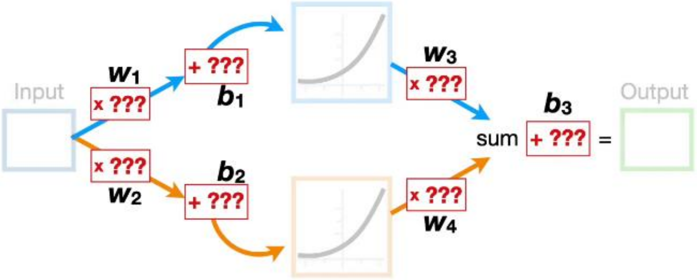
Key references:

- What is a neural network by StatQuest

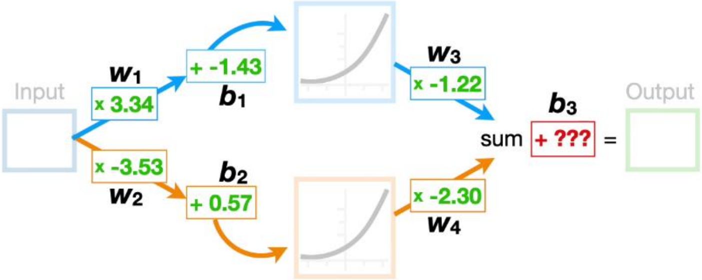
II. Activation functions

Sigmoid –
ReLU –
Tanh –
ArgMax –
SoftMax –

III. Backpropagation. Gradient. 1 parameter



1. Backpropagation starts with the last parameters and works itself backwards to estimate other parameters. In this example, let's assume that we don't know only b_3;



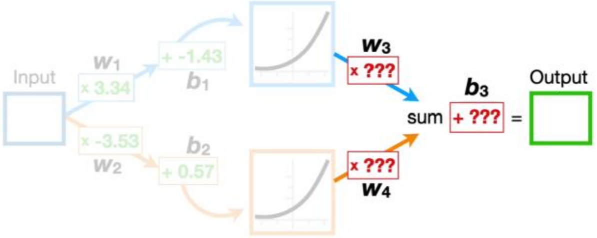
2. We calculate the gradient of our loss function (SSR in this example) with respect to b_3:

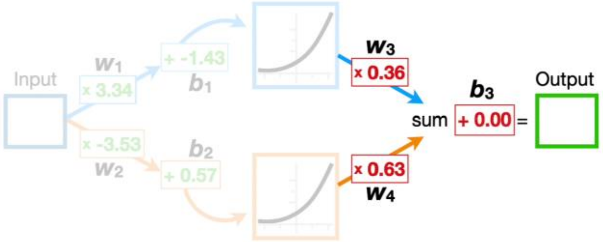
- $\nabla f = \frac{\partial\, SSR}{\partial b_3} = \frac{\partial\, \sum [y_i - \hat{y}_i]^2}{\partial b_3} = \underbrace{\sum -2[y_i - \hat{y}_i] \cdot 1}_{\frac{\partial\, SSR}{\partial \hat{y}_i} \cdot \frac{\partial\, \hat{y}_i}{\partial b_3}}$;
- We initialise $b_3 = 0$: $\nabla f(b_3 = 0) = -15.7$ (let's assume we've plugged in some data);
- Following that, we calculate the $step\_size_{b_3} = 0.1 \cdot (-15.7) = -1.57$;
- We calculate the new $b_3^{new} = b_3 - step\_size_{b_3} = 0 - (-1.57) = 1.57$;
- We use $b_3^{new} = 1.57$ to calculate $\hat{y}_i$, getting a new value for $\nabla f(b_3 = 1.57) = -6.26$;
- We keep calculating step sizes, getting new values for b_3 and smaller and smaller ∇f values. Soon enough, $\nabla f \to 0$ and thus $step\_size_{b_3} \to 0$ – we have found the optimal value for b_3. In this case, $b_3^{opt} = 2.61$.

Additional info:

- Backpropagation. Main ideas by brilliant.org
- How neural networks learn by 3Blue1Brown
- Backpropagation. Main ideas by StatQuest

IV. Backpropagation. Gradient. Multiple parameters

№	Step	Example	Note
Last 3 parameters			
1	Define a loss function. For example: $SSR = \sum [y_i - f(x_i;\ \theta_1, \theta_2, \dots, \theta_n)]^2$	$SSR = \sum (y_i - \hat{y}_i)^2$	$\hat{y}_i$ depends on all b_i and w_i that we don't know
2	For each activation function, let's use the following notation: $(x_{k,i}; y_{k,i})$, where k – an index of an activation function, i – an index of an observation	For instance, for the 1st activation function we have $(x_{1,i}; y_{1,i})$	$x_{1,i}$ – values of exogenous variables that we feed to our 1st activation function. $y_{1,i}$ – outputs of the 1st activation function
3	In this case, our predicted values $\hat{y}_i$ depend on 3 parameters	$\hat{y}_i = w_3 \cdot y_{1;i} + w_4 \cdot y_{2;i} + b_3$	…
 Input w_1 x 3.34 + -1.43 b_1 x -3.53 w_2 b_2 + 0.57 w_3 x ??? x ??? w_4 b_3 Output sum + ??? =			
4	We calculate the gradient of our loss function (SSR in this example) with respect to w_3, w_4, b_3	$\nabla f = \left[\frac{\partial\ SSR}{\partial b_3}, \frac{\partial\ SSR}{\partial w_3}, \frac{\partial\ SSR}{\partial w_4}\right]^T$ $\frac{\partial\ SSR}{\partial b_3} = \sum -2 \cdot (y_i - \hat{y}_i) \cdot 1$ $\frac{\partial\ SSR}{\partial w_3} = \sum -2 \cdot (y_i - \hat{y}_i) \cdot y_{1,i}$ $\frac{\partial\ SSR}{\partial w_4} = \sum -2 \cdot (y_i - \hat{y}_i) \cdot y_{2,i}$	…

	We initialise b_i (usually 0) and pick w_i, which can be done by sampling values from $\mathrm{N}(0, 1)$	Let’s say $b_3 = 0$, $w_3 = 0.36$ and $w_4 = 0.63$	There are other ways of initialising parameters
 Input w_1 x 3.34 + -1.43 b_1 x -3.53 w_2 b_2 + 0.57 w_3 x 0.36 b_3 sum + 0.00 = Output x 0.63 w_4			
5	- For chosen parameters, we calculate the ∇f; - We calculate the $step\_size = l\_rate \cdot \nabla f$; - We can now get new values for our parameters $w_i^{new} = w_i^{old} - step\_size$; - Then, the ∇f is recalculated for the new parameters; - Repeat the process until $\hat{y}_i$ don’t improve or other criteria are met.	$\nabla f \; = [1.9, 2.58, 1.26]^T$ $step\_size = [0.1 \cdot 1.9, 0.1 \cdot 2.58, 0.1 \cdot 1.26]^T$ $step\_size = [0.19, 0.258, 0.126]^T$ $\begin{bmatrix} b_3^{new} \\ w_3^{new} \\ w_4^{new} \end{bmatrix} = \begin{bmatrix} 0 \\ 0.36 \\ 0.63 \end{bmatrix} - \begin{bmatrix} 0.19 \\ 0.258 \\ 0.126 \end{bmatrix} = \begin{bmatrix} -0.19 \\ 0.10 \\ 0.50 \end{bmatrix}$	…

V. Backpropagation. Key concepts

a) **Core idea of backpropagation**

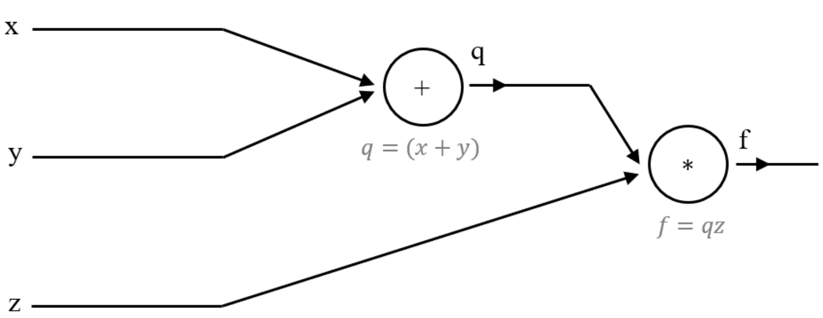
We don't want to expand a loss function and drive the ∇f directly because:

- It is not possible to standardise this process, making it modular. Every time we have a different model, loss function, activation function, we will have to derive ∇f from scratch;
- It also requires a lot of calculations and some derivatives reuse parts from previous steps.

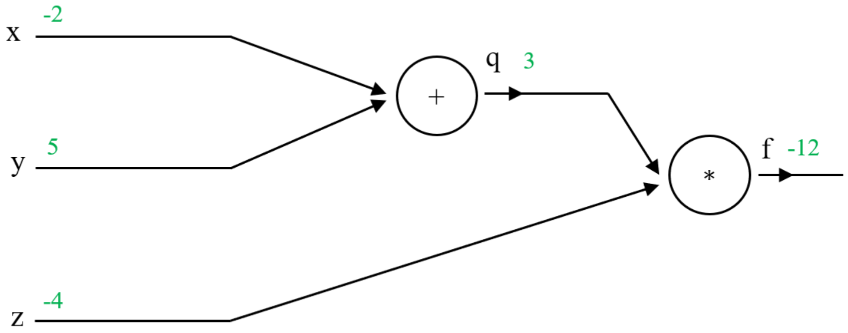
As a result, it is better to rely on **computational graphs**:

1. **Forward pass**: when we use a **computational graph**, we move forward from **inputs** to each consecutive **output** – from left to right. We need this pass to get the final value for $f(x, y, z)$:

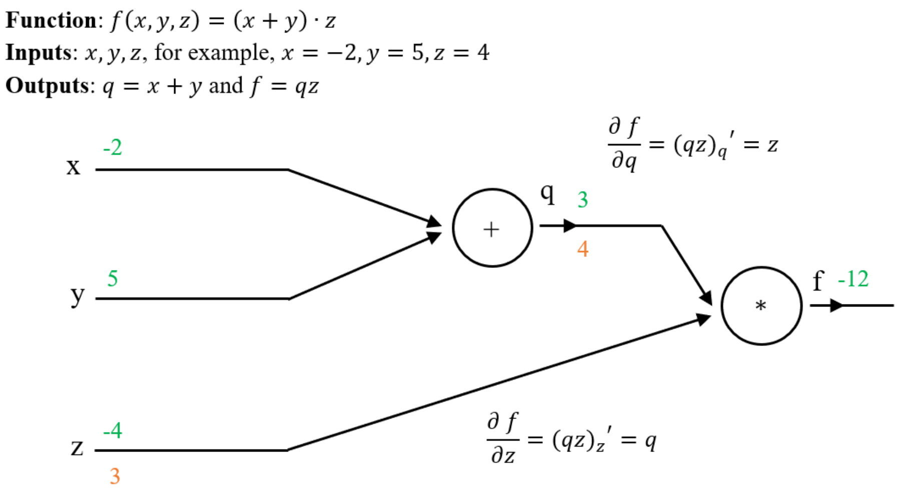
Function: $f(x, y, z) = (x + y) \cdot z$
Inputs: x, y, z, for example, $x = -2, y = 5, z = 4$
Outputs: $q = x + y$ and $f = qz$



Function: $f(x, y, z) = (x + y) \cdot z$
Inputs: x, y, z, for example, $x = -2, y = 5, z = 4$
Outputs: $q = x + y$ and $f = qz$

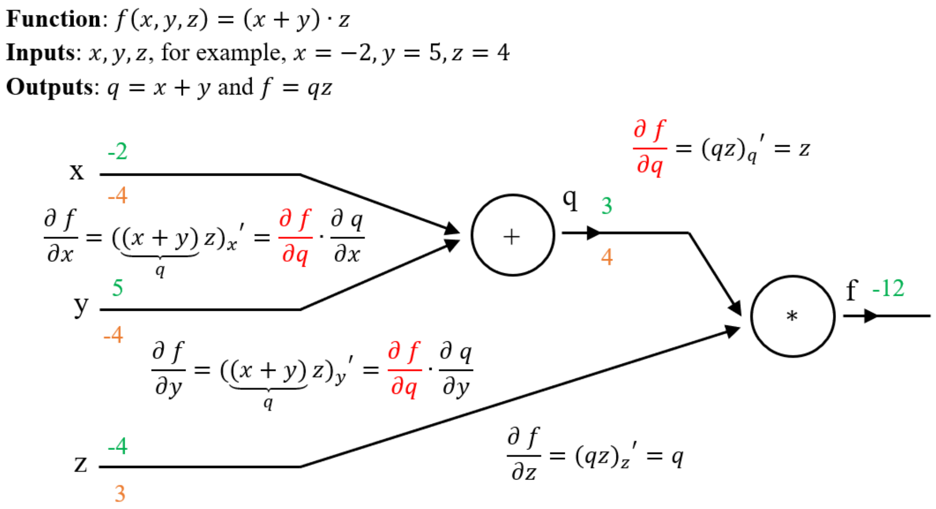


2. **Backward pass**: we want to compute the ∇f with respect to x, y, z, going backwards – from right to left:



Function: $f(x, y, z) = (x + y) \cdot z$
Inputs: x, y, z, for example, $x = -2, y = 5, z = 4$
Outputs: $q = x + y$ and $f = qz$

x -2
y 5
z -4
3
+
q 3
4
*
f -12
$\frac{\partial f}{\partial q} = (qz)_q{}' = z$
$\frac{\partial f}{\partial z} = (qz)_z{}' = q$



Function: $f(x, y, z) = (x + y) \cdot z$
Inputs: x, y, z, for example, $x = -2, y = 5, z = 4$
Outputs: $q = x + y$ and $f = qz$

x -2
-4
$\frac{\partial f}{\partial x} = ((\underbrace{x + y}_{q})z)_x{}' = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x}$
y 5
-4
$\frac{\partial f}{\partial y} = ((\underbrace{x + y}_{q})z)_y{}' = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y}$
z -4
3
+
q 3
4
*
f -12
$\frac{\partial f}{\partial q} = (qz)_q{}' = z$
$\frac{\partial f}{\partial z} = (qz)_z{}' = q$

So, we calculate derivatives in the following order:

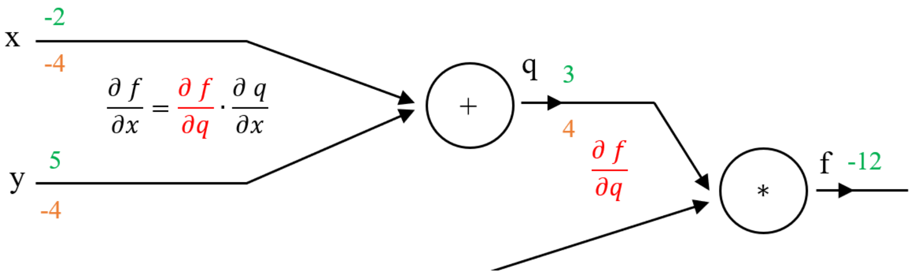
$$\frac{\partial f}{\partial z} = (qz)_z{}' = q$$

$$\frac{\partial f}{\partial q} = (qz)_q{}' = z$$

$$\frac{\partial f}{\partial x} = ((\underbrace{x + y}_{q})z)_x{}' = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x} = (qz)_q{}' \cdot (x + y)_x{}' = z$$

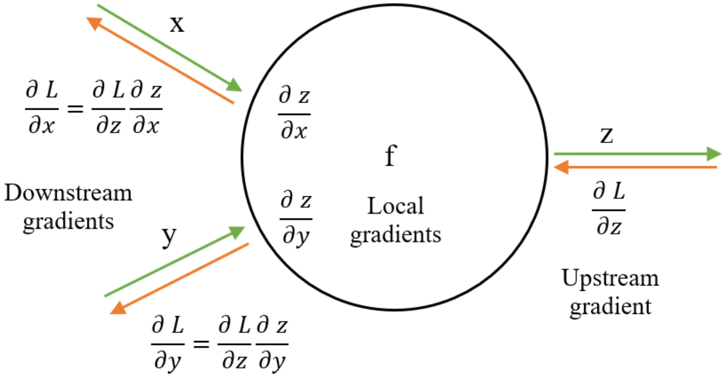
$$\frac{\partial f}{\partial y} = ((\underbrace{x + y}_{q})z)_y{}' = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y} = (qz)_q{}' \cdot (x + y)_y{}' = z$$

Let's consider one of the final derivatives with respect to x:


x
-2
-4
y
5
-4
$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x}$
+
q
3
4
$\frac{\partial f}{\partial q}$
*
f
-12

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x}$$

$\frac{\partial f}{\partial x}$ – **downstream gradient**, $\frac{\partial q}{\partial x}$ – **local gradient**, $\frac{\partial f}{\partial q}$ – **upstream gradient**. Thus, a **computational graph**, can be generalised:


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
Downstream gradients
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
Local gradients
z
$\frac{\partial L}{\partial z}$
Upstream gradient

- During the **forward pass**, we will compute the **output** z and pass it to the next node;
- At the end of the **forward pass**, the final loss L will be calculated, and the backpropagation is going to be initialised;
- At some point, this node will get a signal from the upstream of the graph in the form of **upstream gradient** $\frac{\partial L}{\partial z}$ (how much the loss L changes if we adjust the local output z);
- We can now compute local gradients $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$;
- Using this information, we can compute **downstream gradients** $\frac{\partial L}{\partial x}$ and $\frac{\partial L}{\partial y}$. These gradients can be passed along to other nodes down the graph;
- At the end, we will compute gradients of the loss L with respect to all the parameters. Thus, using this modular approach, we can achieve the global understanding of our function without reasoning about the global structure of our function.

Additional info:

- Backpropagation by Michigan Online

A composite function – 1 variable: we have $f(u) = f[g(x)]$. We want to find the derivative with respect to x:

$$(f[g(x)])' = \left(f(u)\right)' \cdot (g(x))'; \; \frac{df}{dx} = \frac{df}{du} \cdot \frac{du}{dx}; \; \frac{d\, f[g(x)]}{dx} = \frac{d\, f(u)}{du} \cdot \frac{d\, g(x)}{dx}$$

A composite function – 2 variables: we have $f(u, v) = f[g(x), \varphi(x)]$. We want to find the derivative with respect to x:

$$\frac{df}{dx} = \frac{\partial f}{\partial u} \cdot \frac{du}{dx} + \frac{\partial f}{\partial v} \cdot \frac{dv}{dx}$$

In certain cases, functions may look like this: $f[g(x), \varphi(y)]$. Thus, when we calculate the derivative with respect to x, we get: $\frac{df}{dx} = \frac{\partial f}{\partial u} \cdot \frac{du}{dx} + 0$

b) **Autograd**

1. **Mapping an input vector to a scalar**. We have an input vector $X = [x_1, x_2, x_3, \dots x_n]^T$ and some function $y = f(X)$:

$$f: \mathbb{R}^n \to \mathbb{R}$$
$$f: (x_1, x_2, \dots x_n) \mapsto f(x_1, x_2, \dots x_n)$$

$$f: \mathbb{R}^3 \to \mathbb{R}$$
$$f: (x_1, x_2, \dots x_n) \mapsto \sum_{i=1}^{n} x_i$$

If we calculate the ∇f of this function, we will get:

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n}\right]^T = \left[\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \dots, \frac{\partial y}{\partial x_n}\right]^T = \begin{bmatrix} 1 + 0 + \cdots + 0 \\ 0 + 1 + \cdots + 0 \\ \vdots \\ 0 + 0 + \cdots + 1 \end{bmatrix} = [1,1, \dots ,1]^T$$

```
x = torch.tensor((1.0, 5.0, 3.0, 4.0), requires_grad=True)
y = x.sum()

y.backward()
x.grad

output: tensor([1., 1., 1., 1.])
```

2. **Mapping an input vector to an output vector**. We have an input vector $X = [x_1, x_2, x_3, \dots x_n]^T$ and some function $Y = f(X)$:

$$f: \mathbb{R}^n \to \mathbb{R}^m$$
$$f: (x_1, x_2, \dots x_n) \mapsto (f_1(x_1, x_2, \dots x_n), f_2(x_1, x_2, \dots x_n), \dots , f_m(x_1, x_2, \dots x_n))$$

In this case, we will need to compute a Jacobian matrix:

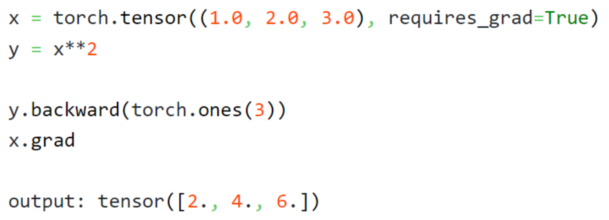
$$\mathrm{J} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}\right]^T = \left[\frac{\partial Y}{\partial x_1}, \frac{\partial Y}{\partial x_2}, \frac{\partial Y}{\partial x_3}\right]^T = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix}$$

Let’s consider the following example:

$$f\colon \mathbb{R}^3 \to \mathbb{R}^3$$
$$f\colon (x_1, x_2, x_3) \mapsto (x_1^2, x_2^2, x_3^2)$$

Note, $f_1(x_1, x_2, x_3)$ doesn’t depend on x_2 and x_3, $f_2(x_1, x_2, x_3)$ doesn’t depend on x_1 and x_3 etc. Also, vector V should match the dimensions of J (columns to rows):

$$J = \begin{bmatrix} 2x_1 & 0 & 0 \\ 0 & 2x_2 & 0 \\ 0 & 0 & 2x_3 \end{bmatrix}; J^T V = \begin{bmatrix} 2x_1 & 0 & 0 \\ 0 & 2x_2 & 0 \\ 0 & 0 & 2x_3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2x_1 \\ 2x_2 \\ 2x_3 \end{bmatrix}$$



```
x = torch.tensor((1.0, 2.0, 3.0), requires_grad=True)
y = x**2

y.backward(torch.ones(3))
x.grad

output: tensor([2., 4., 6.])
```

Additional info:

- Pytorch autograd
- Wiki: Jacobian matrix

VI. SGD

a) **Stochastic gradient decent (SGD)** – the actual gradient calculated for the entire dataset is replaced by its estimate through the usage of random data subsets.

- If only one data point is used (instead of the entire dataset) it is called "on-line" gradient descent;
- If more than one data point is used it is called mini-batch gradient descent.

SGD is not only computationally more efficient, but randomness that it introduces may help in reducing the probability of getting stuck in a saddle point.

b) **Scaling features** – bringing all features to the same scale. If it's not done, a variable with a larger range of values will lead to weights of this feature being updated more even if its actual impact is less significant. A loss function may end up being more difficult to work with – its shape is more elongated along some dimensions and squished along others. As a result, some features will have larger step sizes leading to slower convergence.

c) **SGD with momentum** – this extension of SGD that instead of using only the ∇_i uses exponentially weighted average of the gradients, which reduces oscillation

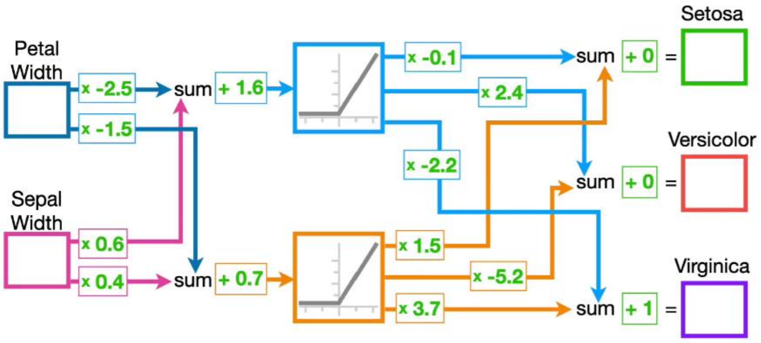
Weight decay is often used in conjunction with other regularization techniques such as dropout and batch normalization. These techniques serve different purposes. Dropout helps in reducing co-adaptation of neurons by randomly dropping out units during training, while batch normalization helps stabilize and accelerate training. These techniques can complement each other and be used together for better results.

Additional info:

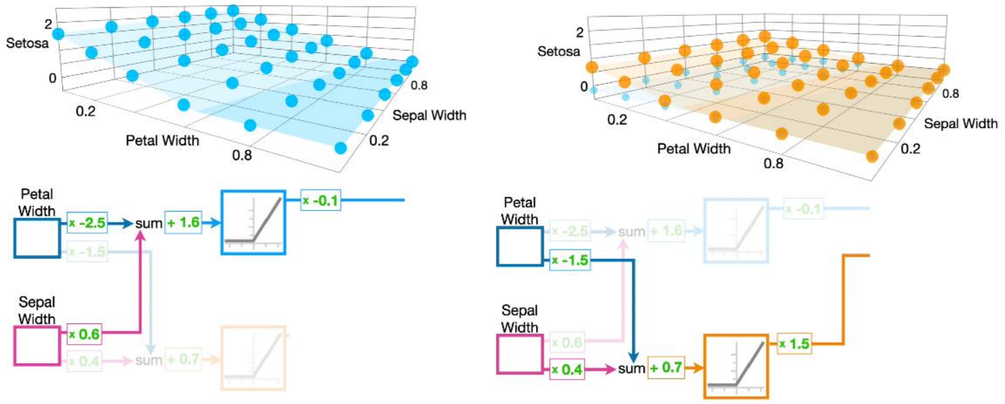
- Wiki: Stochastic gradient descent
- Normalising inputs

VII. Multiple inputs and outputs for an ANN

The basic idea: multiple **inputs** are connected to **hidden layers**, but before we pass them to activation functions, we sum them up. Incoming inputs to the neurons need to be weighted and combined (usually summed but not always), the magnitude of that overall sum can then be passed through a decision function such as the activation functions.



Blue plane – the output of the 1st activation function plotted against 2 exogenous variables.



We than weigh the outputs of activation functions, sum them, add bias and get the final output.

Additional info:

- Multiple inputs and outputs by StatQuest
- Interactions between input variables

VIII. Some regularisation techniques

a)

https://www.youtube.com/watch?v=CPOGlwvVv6o
Batch normalisation –

https://www.google.com/search?q=py+torch+add+weight+regularisation&oq=py+torch+add+weight+regularisation&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCTE4ODAyajBqMagCALACAA&sourceid=chrome&ie=UTF-8#fpstate=ive&vld=cid:e92ad52b,vid:_SlPBbxuqas,st:0

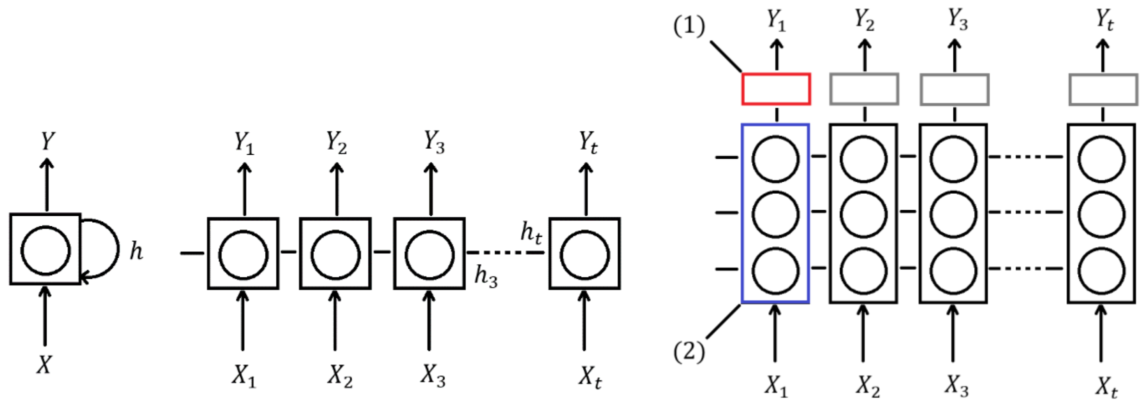
RECURRENT NEURAL NETWORK (RNN)

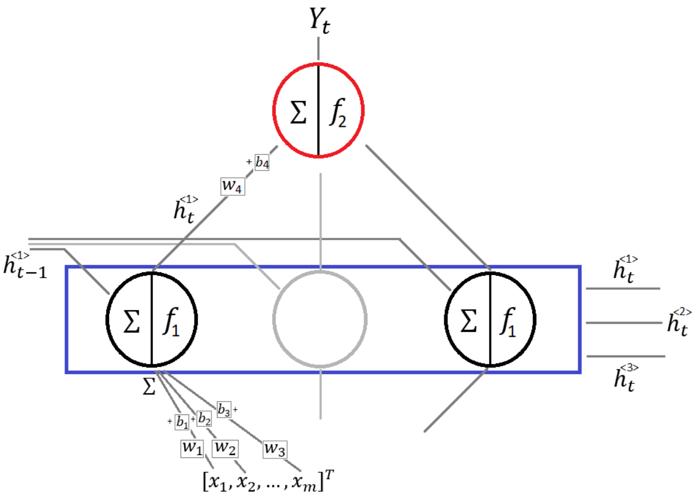
I. Basic implementation

Why ANNs don't work:

- For ANNs the order of inputs doesn't matter – no sequence is assumed;
- For ANNs we cannot easily incorporate data of variable length;

Elements of RNN:





$$h_t = f_1(W_h X_t + U_h h_{t-1} + b_h)$$
$$y_t = f_2(W_y h_t + b_y)$$

- Where X_t – a $\{m \times 1\}$ dimensional input vector (feature vector) at time step t;
- W_h is a $\{n \times m\}$ matrix of weights for X_t, where n is the number of hidden units (neurons). The result of $W_h X_t$ is a $\{n \times 1\}$ vector of sums of weighted variables for each hidden unit;
- b_h is a $\{n \times 1\}$ vector of biases;
- h_{t-1} is $\{n \times 1\}$ a vector of hidden states from the previous time step, and it is multiplied by a weight matrix U_h that has the shape $\{n \times n\}$. So, the output $U_h h_{t-1}$ is a $\{n \times 1\}$ vector;

- After applying an activation function f_1 element wise to the vector $W_h X_t + U_h h_{t-1} + b_h$ we get a vector h_t with the shape $\{n \times 1\}$. This vector is called hidden state that will be passed along to the next time step;
- For each time step, if necessary, we can also calculate y_t. To do so, we usually use a fully connected layer (a linear layer, for example) that takes h_t as its input vector and outputs y_t (could be a scalar or a vector).

Distinct characteristics of RNN:

- All weights and biases are shared across timestamps. This framework allows RNN to generalise well for sequences of different lengths, makes it computationally efficient and eliminates "order constraints";
- Hidden units do not communicate with each other directly via weights. Each hidden unit communicates with itself from previous time steps as well as other units within the hidden layer but indirectly – through shared weights.

Issues RNNs have:

- They use information only from the past (can be a problem in certain tasks);
- They suffer from the vanishing / exploding gradient problem: we multiply h_t by U_h many times (depending on the sequence length). If weights in U_h are > 1 we'll get a very large number and if they are < 1 we get a very small number. These values appear in some derivatives making it impossible to properly use gradient descent. If the value of such a partial derivative is large, the optimisation algorithm is likely to overshoot (miss a local minimum). If the value of such a partial derivative is small, the optimisation algorithm won't converge.

Backpropagation through time:

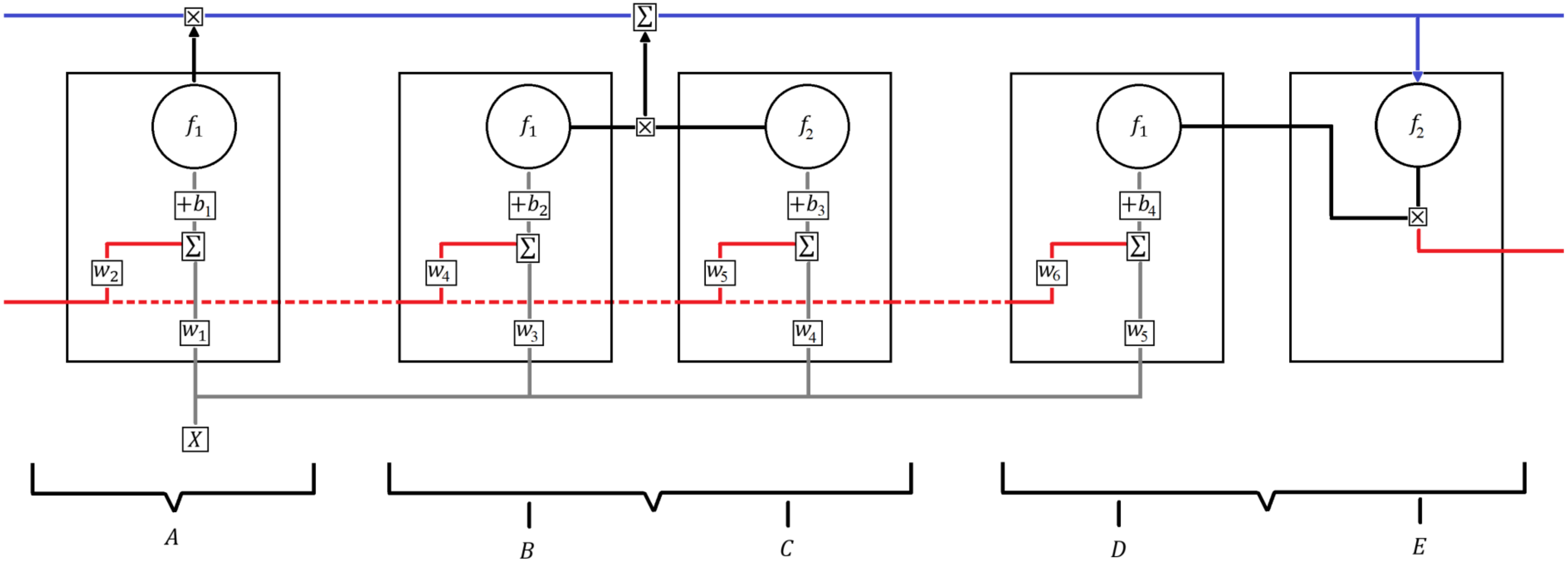
- **Forward pass**: when loss is calculated, it is done for every step y_t and then we sum it up;
- **Backward pass**: since we calculate derivatives for each parameter and these parameters time dependent, we basically go back in time – from the most recent timestamps to the oldest. When derivatives are calculated, RNN is unrolled one step at a time.

Practical issues:

- Although RNNs can take sequences of variable length during prediction, for training you have to establish a fixed length vector using padding so that batching can be used.
- However, during the prediction stage sequences of any given length can be used.

II. LSTM

A more sophisticated architecture is needed to avoid vanishing / exploding gradients. Two separate paths are used: the long-term memory path and the short-term memory path. An RNN's simple hidden unit that outputs $h_t = f_1(W_h X_t + U_h h_{t-1} + b_h)$ is replaced with a more complex one. Thus, to keep it simple, only one unit will be used:



- The blue line is the **cell state** c_t (long-term memory). It doesn't have any weights associated with it which solves the problem of vanishing / exploding gradients;
- The red line is the **hidden state** h_t (short-term memory);

- Section A. **Forget Gate** [**Long-term memory to remember**]: it takes the **previous** h_{t-1} and input X. By using the sigmoid activation function f_1, this gate outputs a value between 0 and 1 which is then multiplied with the **previous** c_{t-1} giving as c_t^*. This gate determines how much of the long-term memory from the previous step to remember;
- Section B and C. **Input Gate**:
    - Section C. **Potential long-term memory**: it takes the **previous** h_{t-1} and input X. By using the tanh activation function f_2, a potential long-term memory is created c_t^{potent};
    - Section B. **Potential long-term memory to remember**: it functions just like the **forget gate**, but this time the **potential long-term memory** gets modified.

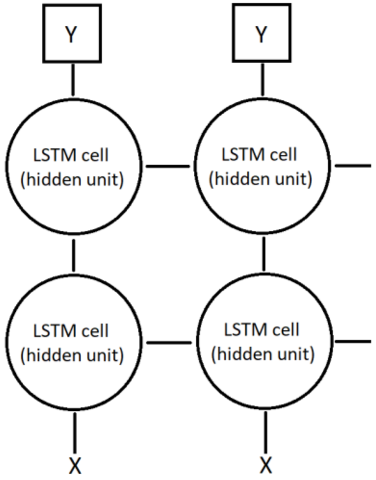
  Finally, we add c_t^* and % c_t^{potent} (modified the long-term memory from the previous state plus the potential long-term memory) and we get a new long-term memory c_t^{new}.

- Section D and E. **Output Gate**:
    - Section E. **Updated short-term memory**: it takes c_t^{new} and multiplies it with the output of the section D to get h_t^{new};
    - Section D. **Short-term memory to remember**. it functions just like the **forget gate**, but this time the **short-term memory** gets modified.

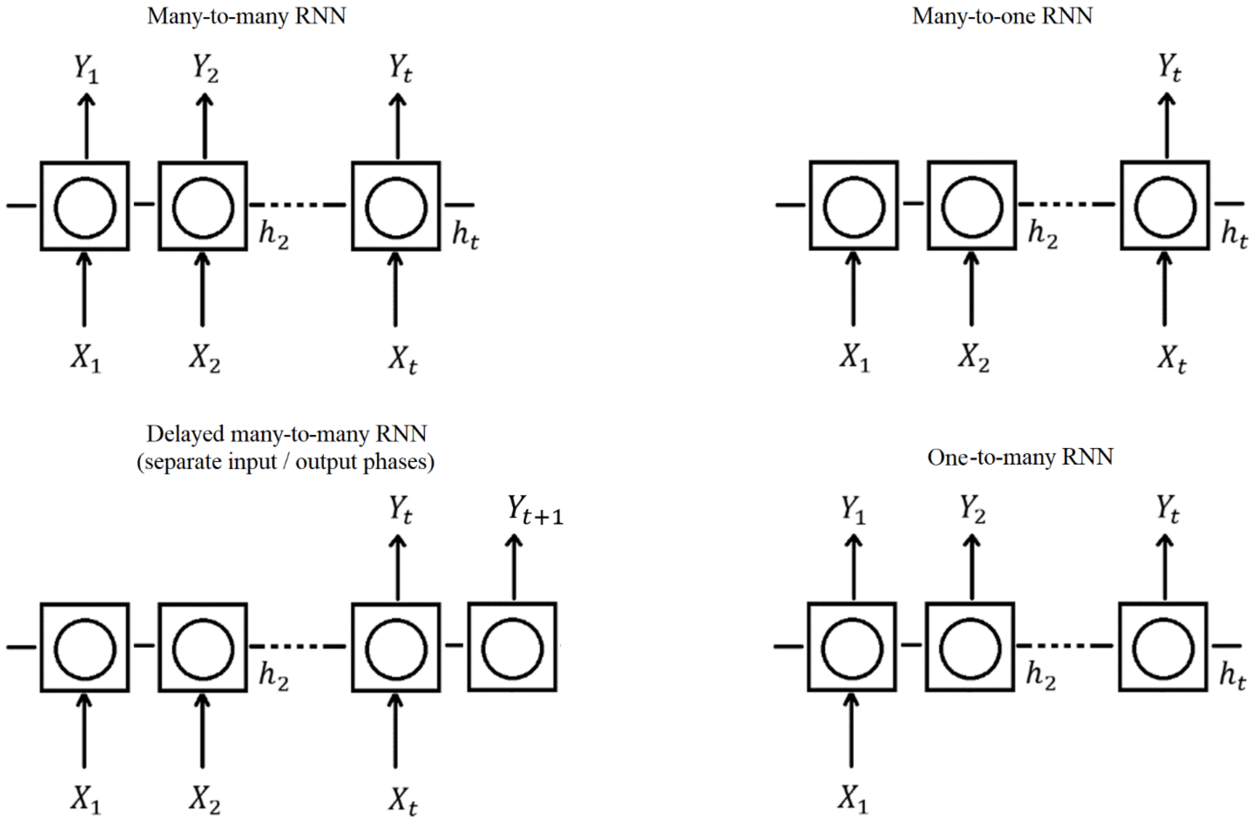
Finally, we get both h_t^{new} and c_t^{new} that are passed to the next step.

Building a more complex model:

- Just like with RNNs, we can use multiple LSTM cells (hidden units) that will be then combined via a fully-connected layer;
- We can also create stacked LSTMs: hidden states (short-term memory) produced by the 1st layer are passed to the 2nd layer as an input.



LSTM types:



- Taking input and generating output happens within the same RNN structure;
- Output is delayed until an entire input sequence is processed.

Additional info:

- RNN by Stanford.edu
- RNN by Oreilly
- Number of cells in RNN
- Why do RNNs share weights #1
- Why do RNNs share weights #2
- Sequences of variable length

III. Encoder-decoder model

? Convolution

a) **Convolution of f and g** – the product of two functions after one is reflected (not always necessary) and shifted.

b) **2D convolution** – element-wise multiplication of 2 matrices (the original matrix and the filter matrix, i.e. kernel), summing up the results, and doing it for all positions of the filter.

It allows us to modify the original matrix. For instance, if the original matrix represents an image, using the filter matrix with equal weights (1/4 for a 2×2 matrix), we can smooth the original image – take the average of nearby pixels.

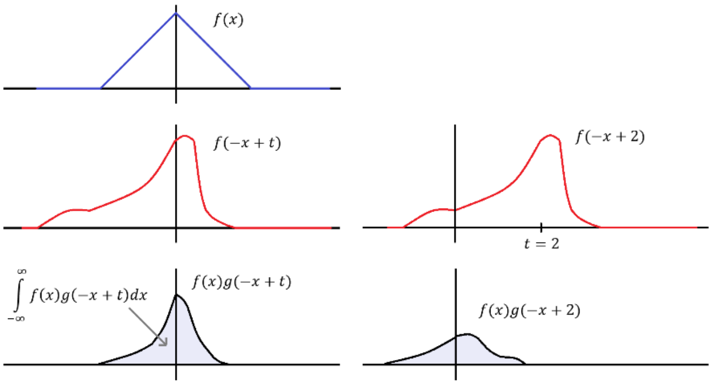
$$(f * g)(x, y) = \sum_{x'=-\infty}^{x} \sum_{y'=-\infty}^{y} f(x', y')g(x' + x, y' + y)$$

Stride – how far the filter moves along any direction. If the stride is large, we will skip pixels (or any other entities we compute the convolution over) and, as a result, reduce the output – downsample.

Padding – increasing the size of an image (or any other entities we compute the convolution over) by adding additional data (0s, other values) so that we could use the filter on the edges as well. Using padding we can keep the original dimensions intact.

In case of a continuous case with a function of a single variable we can define convolution as follows:

$$[f * g](t) = \int_{-\infty}^{\infty} f(x)g(-x + t)dx$$



We use $-x$ to flip the function which can be useful for calculating $P(X + Y)$, for example. However, in ML-related topics like classification using convolutions, it is not necessary. The letter t is used to shift the kernel step by step.

Additional info:

- Intuitive guide to convolution
- Convolutional filter by Programmatically
- Convolutions and kernels by University of Edinburgh
- Visual explanation of convolution

CLASSIFICATION METRICS (SCORING RULES). BINARY CASE

		Actual	
		Positive	Negative
Predicted	Positive	TP (True Positive)	FP (False Positive)
	Negative	FN (False Negative)	TN (True Negative)

<table>
<tr><th colspan="3">I. Accuracy, Precision, Recall, F1</th></tr>
<tr>
<td><b>Accuracy</b>: out of all predictions how many are correct?</td>
<td>

$$Accuracy = \frac{TP + TN}{TP + TN + FN + TN} = \frac{Correct\ preds}{All\ preds}$$

<table>
<tr><td></td><td></td><th colspan="2">Actual</th></tr>
<tr><td></td><td></td><td>healthy</td><td>ill</td></tr>
<tr><td><b>Predicted</b></td><td>healthy</td><td>100</td><td>20</td></tr>
<tr><td></td><td>ill</td><td>30</td><td>50</td></tr>
</table>

$$Accuracy = \frac{100 + 50}{100 + 50 + 30 + 20} = \frac{150}{200}$$

</td>
<td>

<b>Issues</b>:

- If the target class is not balanced, high accuracy can be achieved by just guessing the majority class;
- It is invariant with respect to classes. We cannot target a specific class as more important one.

</td>
</tr>
<tr>
<td><b>Precision</b>: what you are saying is a positive actually is a positive</td>
<td>

$$Precision = \frac{TP}{TP + FP} = \frac{Correct\ positive\ preds}{All\ positive\ preds}$$

<table>
<tr><td></td><td></td><th colspan="2">Actual</th></tr>
<tr><td></td><td></td><td>killer</td><td>citizen</td></tr>
<tr><td><b>Predicted</b></td><td>killer</td><td>100</td><td>20</td></tr>
<tr><td></td><td>citizen</td><td>30</td><td>50</td></tr>
<tr><th colspan="4">Extreme case</th></tr>
<tr><td><b>Predicted</b></td><td>killer</td><td>1</td><td>0</td></tr>
<tr><td></td><td>citizen</td><td>200*</td><td>0</td></tr>
</table>

</td>
<td>No matter what, we want to correctly guess every single <b>predicted killer</b>. However, our model may start being too careful at accusing people of being killers (we won’t catch a lot of killers).</td>
</tr>
</table>

<table>
<tr><td></td><td>

** Murderers we didn't convict – we are being too careful.*

$$Precision_1 = \frac{100}{100+20} = \frac{100}{120} = 83\%$$

$$Precision_2 = \frac{1}{1+0} = \frac{1}{1} = 100\%$$

</td><td></td></tr>
<tr><td>

Recall: actual positive observations are classified correctly

</td><td>

$$Recall = \frac{TP}{TP+FN} = \frac{Correct\ positive\ preds}{All\ positive\ actuals}$$

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2"><b>Actual</b></td></tr>
<tr><td>killer</td><td>citizen</td></tr>
<tr><td rowspan="2"><b>Predicted</b></td><td>killer</td><td>100</td><td>20</td></tr>
<tr><td>citizen</td><td>30</td><td>50</td></tr>
<tr><td colspan="4"><b>Extreme case</b></td></tr>
<tr><td rowspan="2"><b>Predicted</b></td><td>killer</td><td>1</td><td>200*</td></tr>
<tr><td>citizen</td><td>0</td><td>0</td></tr>
</table>

** False alarm – we are being too frivolous with our allegations.*

$$Recall_1 = \frac{100}{100+30} = \frac{100}{130} = 77\%$$

$$Recall_2 = \frac{1}{1+0} = \frac{1}{1} = 100\%$$

</td><td>

No matter what, we want to correctly identify every single **actual killer**. However, our model may start guessing that most people are killers since this metric indicates nothing about another class (we lose public trust – we accuse too many innocent people).

</td></tr>
<tr><td>

F1 Score: a harmonic average of **precision** and **recall**

</td><td>

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

</td><td></td></tr>
<tr><td colspan="3"><b>II. ROC and PR curves. AUC (area under the curve)</b></td></tr>
<tr><td colspan="3">

If a model outputs probabilities, we can choose different thresholds for classifying observations. For example, if the threshold is 0.5 the range $(0, 0.49)$ is associated with the class 0 and the range $(0.5, 1)$ is associated with the class 1.

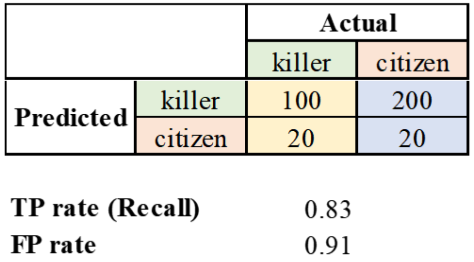
</td></tr>
</table>

ROC curve (receiver operator characteristic): one way to choose the threshold is to analyse a ROC curve, which depicts the interplay between true positive rate (recall) and false positive rate.

$$True\ positive\ rate = \frac{TP}{TP + FN}$$

$$False\ positive\ rate = \frac{FP}{FP + TN}$$

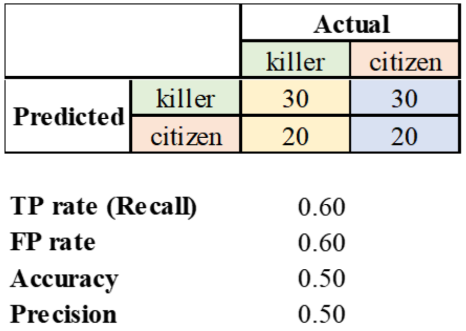
The larger the **true positive rate** (**recall**) is, the better (we identify all actual positives as positives). The smaller the **false positive rate** is, the better (we identify actual negatives as positives – false alarm):



		Actual	
		killer	citizen
Predicted	killer	100	200
	citizen	20	20

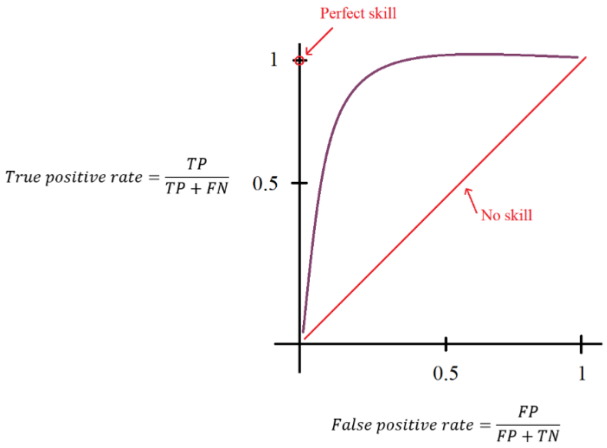
TP rate (Recall)	0.83
FP rate	0.91

If **true positive rate** (**recall**) equals to **false positive rate**, then our models doesn't distinguish between classes at all:



		Actual	
		killer	citizen
Predicted	killer	30	30
	citizen	20	20

TP rate (Recall)	0.60
FP rate	0.60
Accuracy	0.50
Precision	0.50



		Actual	
		killer	citizen
Precision			
Predicted	killer	100	20
	citizen	30	50
Recall			
Predicted	killer	100	20
	citizen	30	50
True / False positive rate			
Predicted	killer	100	20
	citizen	30	50
Reminder			
Predicted	killer	TP	FP
	citizen	FN	TN

ROC AUC

MODEL TRAINING AND EVALUATION

I. Train. Validate. Test. Hyperparameters

a) **Training set** – a sample of data used for training a model (its parameters).

b) **Validation set** – a sample of data used for tuning hyperparameters. It doesn't overlap with the training set and is often replaced by cross-validation.

c) **Test set** – a sample of data that is used to assess the performance of a model on previously unseen data.

d) **Hyperparameter** – a parameter whose value is set before the machine learning process begins. In contrast, the values of other parameters are derived via training. Algorithm hyperparameters affect the speed and quality of the learning process – are used to control the learning process.

e) **Tuning and comparing models**:

<table>
<tr><th>Train inner</th><th>Val inner</th><th>Val outer (testing)</th></tr>
<tr><td>Training models</td><td>Tuning HP</td><td></td></tr>
<tr><td colspan="3">1. Train n models using **train inner**;
2. Use **validation inner** to tune HP;
3. Pick models with the best HP;</td></tr>
<tr><th colspan="2">Train outer</th><th>Val outer (testing)</th></tr>
<tr><td colspan="3">1. Train models using **train outer**;
2. Compare models using **validation outer**.</td></tr>
</table>

In the case of a large dataset, you may want to use a smaller subset of the data for the initial evaluations of hyperparameters and then fine-tuning the best configurations on the full dataset.

Plenty of models require early stopping to properly tune number of trees or epochs. In this case, the process gets a little bit more difficult:

<table>
<tr><th>Train inner</th><th>Val inner 1</th><th>Val inner 2</th><th>Val outer (testing)</th></tr>
<tr><td>Training models</td><td>Tuning HP</td><td>ES</td><td></td></tr>
<tr><td colspan="4">1. Train n models using **train inner**;
2. Use **validation inner 1** to tune HP and **validation inner 2** for early stopping;
3. Pick models with the best HP;</td></tr>
<tr><th colspan="2">Train outer</th><th>Val inner 2</th><th>Val outer (testing)</th></tr>
<tr><td colspan="4">4. Train models using **train outer** and use **validation inner 2** for early stopping (in case of decision trees you may consider training a model on **train outer** +</td></tr>
</table>

validation inner 2 since there is no need in early stopping anymore);

5. Compare models using **validation outer**.

Sometimes combining multiple models can be the best option. Use bagging, stacking, blending, which also requires either cross validated predictions or a validation set.

It may also be reasonable to use early stopping for each CV round particularly when it comes finding the optimal number of epochs for forecasting time series.

Additional info:

- Tuning HP and comparing models
- Properly tuning a neural network
- Training multiple neural networks and combining them
- Early stopping with cross validation