1 Machine Learning & Neural Networks

1.1 Learning

Learning is an essential process for dealing with unknown environments. An agent is almost **never omnisicient**.

Learning also makes an agent "aware" of their environment. Rather than trying to change the environment so that they can take their actions, they might **try to use and adapt to the environment** to fulfill their tasks.

Lastly, Learning improves the agents decision mechanisms the longer it is active. No longer will an agent repeat actions hoping for different results (e.g. local extrema).

1.1.1 Methods of Learning

Memorization (Declarative Knowledge):
Accumulates individual facts and their outcomes. This is limited by the time to observe facts and memory to store facts.

Better: **Generalization (Imperative Knowledge):**
Deduce new facts from old facts. Limited by accuracy of deduction process, assumes relations between past and future.

1.1.2 Inductive Learning

Inductive learning is the simplest form of learning. It deduces a trend from examples.

Basic Idea:

- f is the (unknown) target function. $f(x)$ is then called the target
- Examples are defined in the form $(x, f(x))$ e.g. Weather data: (1, Rain)
- Use examples to find a hypothesis h such that $h \approx f$

Hypothesis h is **consistent** if $\boxed{\forall x : h(x) = f(x)}$.

Ockhams Razor

The **best explanation** is the **simplest explanation** that fits the data

Overfitting Avoidance

Maximize the combination of consistency and simplicity.

1.2 Machine Learning

Programming an algorithm to automatically learn a program from data or experience.

Machine Learning is **not synonymous** with AI. Many AI algorithms are based on Machine Learning, but not all.

1.2.1 Machine Learning & Human Learning

Human learning is often

- very data- and knowledge efficient
- a complete multitasking, multi-modal system
- time-inefficient

Machine learning is often inspired by human learning, the goal however, is not to recreate human learning.

- May borrow idead from human learning (e.g. neural networks)

- May perform better or worse

1.2.2 Designing a Learning System

Machine Learning is not the solution to every problem!

1. Do I need a learning approach for my problem?
   - Is there a pattern to detect?
   - Can I solve the problem analytically?
   - Do I have data to train on?
2. What type of problem do we have?
   - How to represent it?
   - Which algorithm to use?
3. Gather and organize data
   - Preprocessing is important
4. Fitting / Training your model
5. Optimization
6. Evaluate and iterate back to step 2

1.2.3 Types of Learning

Supervised Learning

Learning based on labeled datasets. Learns to map inputs to outputs based on pairs in the dataset.

Unsupervised Learning

Learning based on unlabeled datasets. Searches for patterns and similarities in the data.

Reinforcement Learning

Agent acts and receives positive or negative reward. The agent learns to act in a way that maximizes the reward.

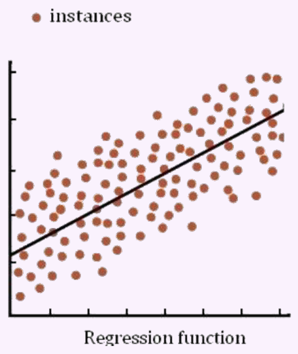
1.2.4 Supervised Learning

Given: Dataset $(x_1, y_1), \dots, (x_n, y_n)$

Goal: Find a function h such that $h(x_i) \approx y_i$

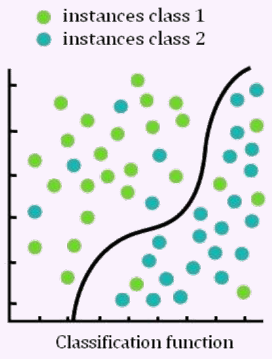
Regression Task

- y is a continous value
- Example: Temperature tomorrow



Classification Task

- y is a discrete class label
- Algorithm tries to predict a continous value describing the label probability
- Example: Will it be above 0°C tomorrow?



1.2.5 Representation

Machine learning algorithms need to handle a lot of different data (e.g. images, audio, etc.).

To make this as easy as possible we represent out input as an **input vector** (in $\mathbb{R}^n$). Vectors are a great representation as we can transform the data using linear algebra.

Representation

Mapping input to another space, that is **easier** to manipulate.

Feature

Features are the **independent variables** in machine learning models.

Model

The representation of what our algorithm has learned from the data it used in training. The model is the **output representation** of the learned ”rule set”.

1.2.6 Feature Engineering

Feature engineering is the process of selecting, manipulating and transforming raw data into features that can be used for machine learning.

For this it is important to know your data:

- Data distribution
- Outliers
- Data reflective of reality
- Biased data

- ...

On the basis of these factors we can choose approches that best fit our needs.

Regression:
- Linear Regression
- Multiple Linear Regression
- Regression Trees
- Non-linear Regression
- Polynomial Regression
- ...

Classification:
- Random Forest
- Decision Trees
- Logistic Regression
- Naïve Bayes
- Support Vector Machines
- ...

1.3 Evaluating a Model

To evaluate a model we need to know the goal we are trying to achieve.

Based on the goal we can use different evaluation metrics. These are the most common ones:

- Accuracy
- Precision
- Recall
- Mean Squared Error
- ...

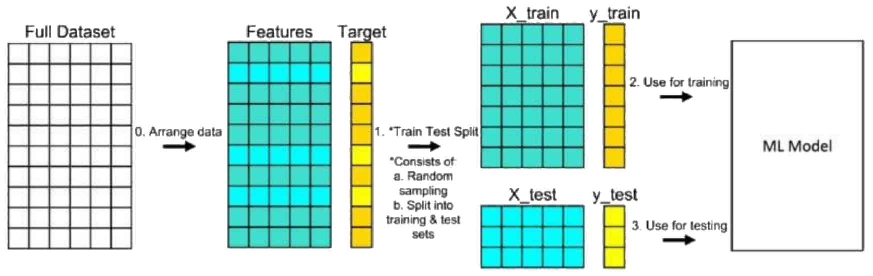
Which metric is best depends on the problem we are trying to solve.

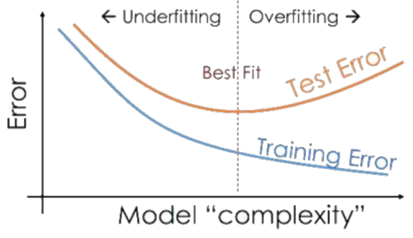
1.3.1 Overfitting

Overfitting describes the problem, that a model is trained ”too well” so that its closer to memorization than generalization. This means that the model is not able to generalize to new data and therefore performs worse on new data.

To deal with overfitting we can:

- Splitting data
  - Split data into training and test data





- Regularization
- Use more data, augement data (adding noise)
- Select different features
- Cross validation
- Ensemble methods

1.4 Neural Networks

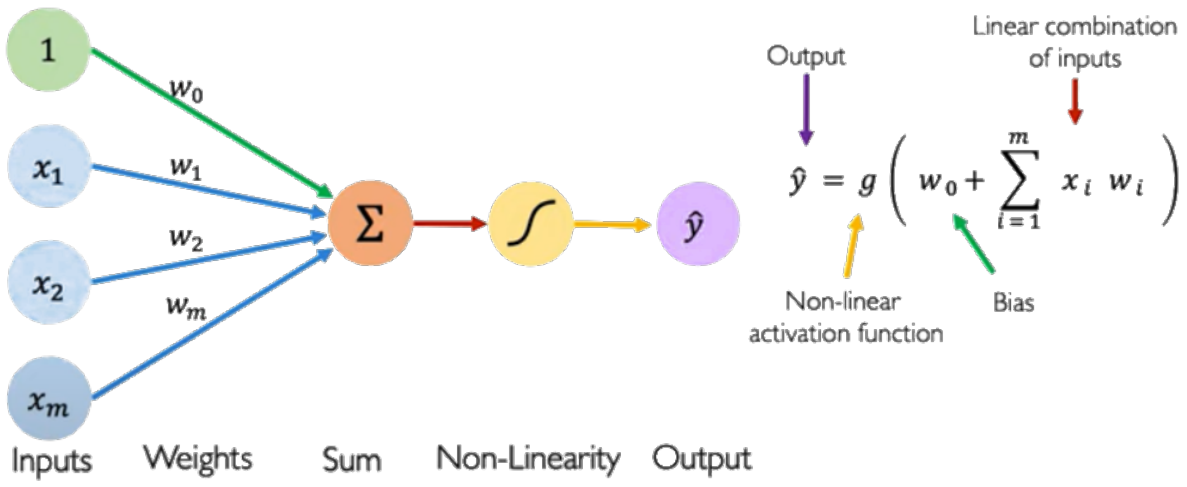
1.4.1 Deep Learning

Up until now we always had to supervise learning in some way, like feature extraction.

But with deep learning we can **learn the underlying features** directly from data without specifying them.

1.4.2 Perceptron

The perceptron can be seen as an **artificial neuron**. It takes multiple inputs, that are weighed individually and summed up to give a single output.

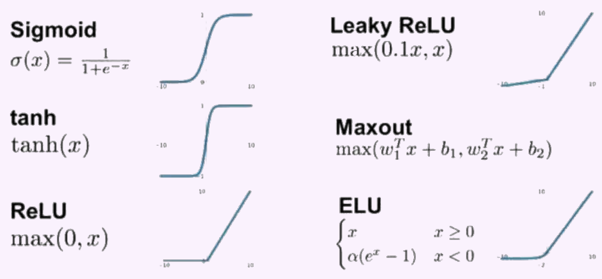


1
x_1
x_2
x_m
w_0
w_1
w_2
w_m
Σ
$\hat{y}$
Inputs
Weights
Sum
Non-Linearity
Output
Output
Linear combination of inputs
$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$
Non-linear activation function
Bias

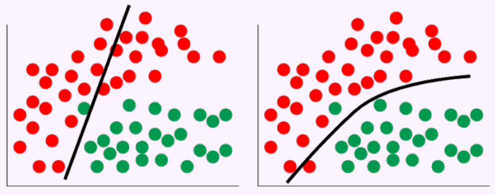
- Neurons correspond to nodes or units
- A link from unit j to uni i propagates activation y from j to i
- The weight $w_{i,j}$ of the link determines the strength and sign of the connection
- All weights together are called W or θ and describe the model
- The **total input activation** is the sum of the input activations
- The **output activation** is determined by the activation function g

Activation Function

A activation function decides if a neuron should be active. Nowadays they're mostly non-linear.

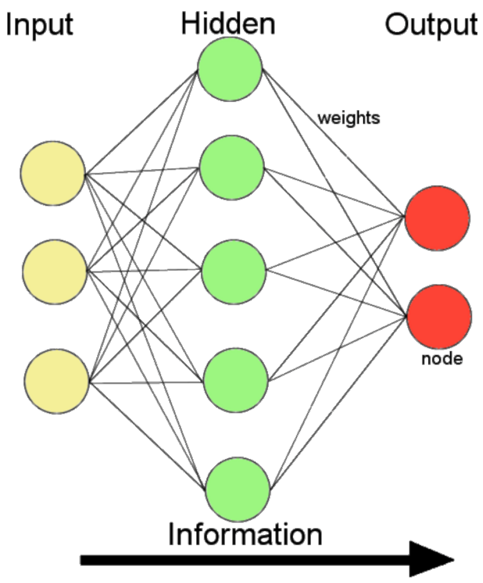


Why do we need an activation function?
- It adds non-linearity to a neural network
- Without it we would have a linear regression model
- Allows for backpropagation



1.4.3 Perceptron to Neural Network

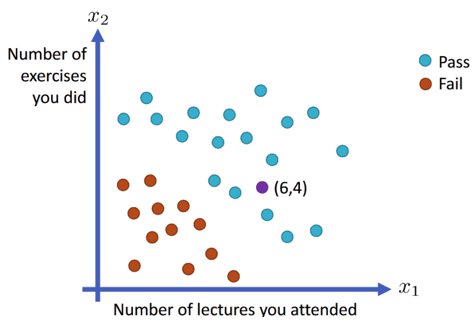
- Perceptrons may have multiple output nodes which can be combined to other perceptrons
- We can build networks of these node: **Multilayer Perceptron (MLP)**
- In a MLP information flow is **unidirectional**
- Information is distributed and processed in parallel
- For each node:
$$\boxed{z_{k,i} = \sigma(w^k_{0,i} + \sum_{j=1}^{n_{k-1}} z_{k-1,j} \cdot w^k_{j,i})}$$



1.5 Forward Propagation

1.5.1 Applying a NN

Assume a distribution of an exam:

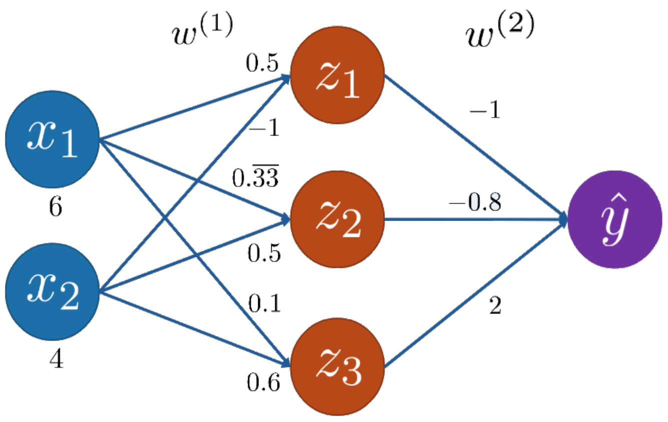


Assume the purple point is $x_1 = 6, x_2 = 4$ and the weights of our NN are:

$$\boxed{\begin{array}{lll} w^1_{1,1} = 0.5 & w^1_{2,1} = -1 & w^2_{1,1} = -1 \\ w^1_{1,2} = 0.\overline{33} & w^1_{2,2} = 0.5 & w^2_{2,1} = -0.8 \\ w^1_{1,3} = 0.1 & w^1_{2,3} = 0.6 & w^2_{3,1} = 2 \end{array}}$$

Further assumme a bias $w^k_{0,i} = 0$.

This yields a NN model:



From this we can calculate z:

$$\boxed{\begin{aligned}&z_1 = g\left(w^1_{0,1} + \sum_{j=1}^{2} x_j \cdot w^1_{j,1}\right) = g\left(x_1 w^1_{1,1} + x_2 w^1_{2,1}\right) = g\left(3 + (-4)\right) = g(-1)\\ &z_2 = g\left(2+2\right) = g(4)\\ &z_3 = g\left(0.6 + 2.4\right) = g(3)\end{aligned}}$$

For the activation function g we use the sigmoid function: $\boxed{g(x) = \frac{1}{1+e^{-x}}}$

This yields $z_1 = 0.26, z_2 = 0.98, z_3 = 0.95$.

To get the prediction $\hat{y}$ we need to calculate the output activation:

$$\boxed{\begin{aligned}\hat{y} &= \sigma\left(w^2_{0,1} + \sum_{j=1}^{3} z_j \cdot w^2_{j,i}\right)\\ &= \sigma\left(w^2_{0,1} + z_1 w^2_{1,1} + z_2 w^2_{2,1} + z_3 w^2_{3,1}\right)\\ &= \sigma\left((-1)\cdot 0.26 + (-0.8)\cdot 0.98 + 2 \cdot 0.95\right)\\ &= \sigma(0.856) = 0.7\end{aligned}}$$

With this we can conclude, that according to our model, the student has a 70% chance of passing the exam.

1.6 Backpropagation

Assume different weight than in the example above so that $\hat{y} = 0.14$.

Quantifying Loss

$$\boxed{\mathcal{L}\left(f(x^i;\theta), y^i\right)}$$

Describes the cost of incorrect predictions.

Empirical Loss

$$\boxed{J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f(x^i;\theta), y^i\right)}$$

Measures the total loss over the dataset. Also known as **objective function, cost function or emprical risk**.

Our goal is to **minimize the empirical loss**:

$$\boxed{\theta^* = \arg\min_{\theta} J(\theta)}$$

θ^* is the weight setting where the empirical loss is minimized.

This can be done using **Gradient descent**, as the **Weight space** is a N-dimensional space where N ist the total number of weights.

Algorithm:

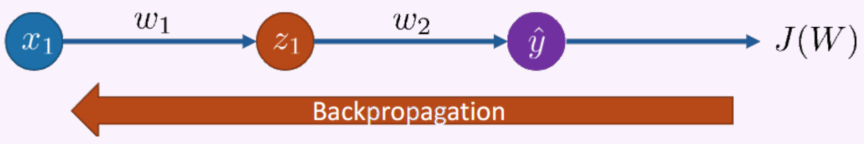
1. Initialize weights randomly
2. Loop until convergence
   - Compute gradient $\dfrac{\partial J(\theta)}{\partial \theta}$
   - Update weights $\theta \leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$ (with α learning rate)
3. Return weights

How to compute $\frac{\partial J(\theta)}{\partial \theta}$

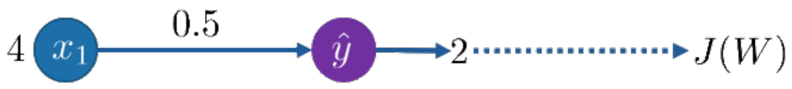
Main Question: How much do the weights affect the outcome i.e. the final loss?
Using the chain rule we can describe the problem as:

$$\boxed{\frac{\partial J(\theta)}{\partial w_2} = \frac{\partial J(\hat{y})}{\cdot} \frac{\partial \hat{y}}{\partial w_2}} \qquad \boxed{\frac{\partial J(\theta)}{\partial w_1} = \frac{\partial J(\theta)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}}$$

We can repeat this for every weight in the network using the gradient from later layers: Propogate the error back to all nodes, through the network.



Example



Given $g = ReLU(x) = \max(0, x), x_1 = 4, y = 1, \hat{y} = 2, \mathcal{L} = (\hat{y} - y)^2, J(\theta) = \mathcal{L}$

ReLU: Rectified Linear Unit

If x is greater than 0 return x else return 0.

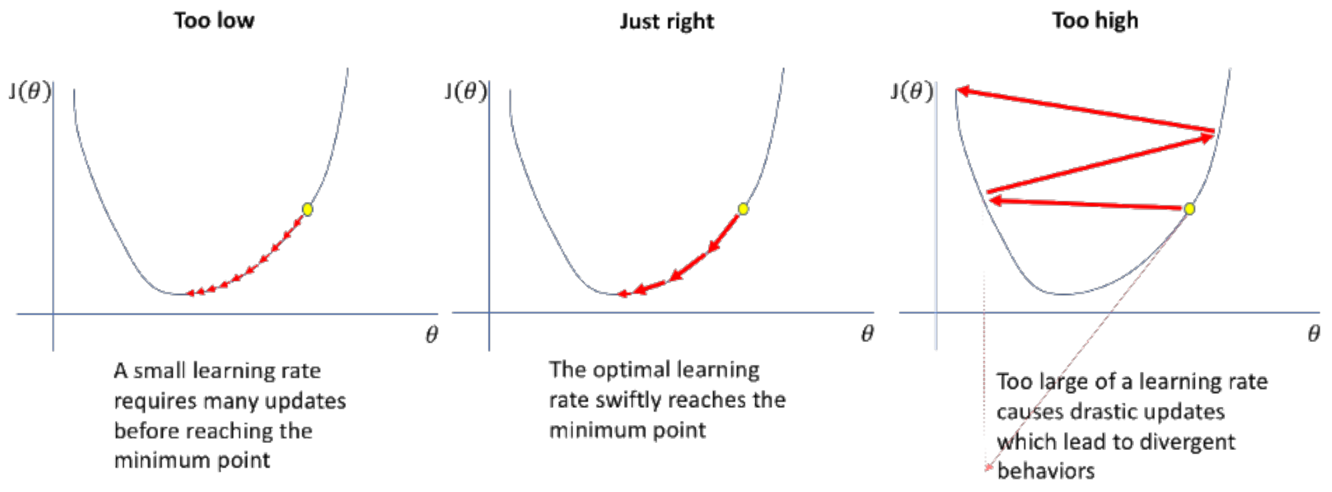
Goal: $\frac{\partial J(\theta)}{\partial w_1} = \frac{\partial J(\theta)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1}$

1. $\frac{\partial J(\theta)}{\partial \hat{y}} = 2(\hat{y} - y) = 2(2 - 1) = 2$ (Power rule)
2. $\frac{\partial \hat{y}}{\partial w_1} = ReLU'\left(w_0 + \sum_{i=1}^{n} w_i x_i\right) \cdot x_1 = 1 \cdot 4 = 4$
3. $\frac{\partial J(\theta)}{\partial w_1} = \frac{\partial J(\theta)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1} = 2 \cdot 4 = 8$

Now we also need to update the weights $\theta = \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$ (assume learning rate $\alpha = 0.05$):
$\theta_{\text{new}} = \theta_{\text{current}} - \alpha \frac{\partial J(\theta)}{\partial \theta}$
$\theta_{\text{new}} = 0.5 - 0.05 \cdot 8 = 0.1$

Loss function in general are hard to optimize as its difficult to find a global minimum.

Basic Idea:
Change weight into the direction of the steepest descent of the error function given a step size - This step size is called the **learning rate α**, but finding a good value is hard.


Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors