# Methods for AI

Notes by Bence Csabai, taught by Lehel Csató

September 9 2021

## 1 Disclaimer

These are **not the official class notes**, these were created by a student, but I try to be as exact as possible. Some parts of these notes are copied from the lecture slides by Lehel Csató. If you have any questions you can find me via email: csabai.bence at gmail.com

## 2 Literature

Recommended literature can be found in the slides on page 15 of lesson 1 with link to free PDFs/interactive jupyter notebooks.

- **Bishop: Pattern Recognition and Machine Learning** (this is the most highly recommended book)
- **Deisenroth et. al.: Mathematics for Machine Learning**
- **Smola et. al.: Dive Into Deep Learning** (interactive notebooks)

## 3 Lecture 1

### 3.1 Introduction

Let's make a rough description for how we think about large deep learning/machine learning models. First we need some data to work with and learn from. Then we need a computer. At times needs to be huge and have for example strong GPUs to be able to process all the computations needed for a complex deep learning task. We also need a neural network that will be running on the computer and will train for the specific task. The way a neural network's training can be described - with a "biological" approach - is that it observes its surroundings and learns from trial and error.

Let us think about a question. How would we train a computer to recognize faces, or just to decide if there is a person in the picture or not? One approach would be to write a software to segment the images into eyes, nose, ears but this is not necessarily the best approach. A neural network could train to find

features on faces on its own, and if it can see eyes, for example, the likelihood that there is a face in the picture will be pretty high.

It is very important to know what features our model learns. Sometimes the results of training can give us a false sense of success. For example if trying to decide if an image contains cows or not, you may find that the network can correctly identify the images in the training dataset. But it is quite possible that the model did not in fact learn to detect cows, but rather, based on training data, recognized that if there is green grass in the picture, there is a high likeliness that there are also cows! This could prove to be a problem later if you try to use this model to find cow on images that do not contain grass, or on images that have grass but no cows.

Therefore it is one of the most important things not just to write the model but also think about and test what is happening on the inside. Try to use as robust of a dataset for training as possible. E.g. for images/videos, use many types of backgrounds, and if needed, add noise manually.

## 3.2 Neural Networks

Neural networks (mathematically) are basically functions $f(\mathbf{x}) = \mathbf{y}$ that take some kind of $\mathbf{x}$ observation and map it to a $\mathbf{y}$ prediction space. !DATA!

In the real world there **"is"** a function $y = f(x)$ but when collecting the data, $\mathbf{x}$ is almost always **corrupted** by some noise (e.g. lights, background sounds, electrical noise, etc.). The noise can take on multiple forms:

$$t_n = y_n + \epsilon \qquad \text{additive noise} \tag{1}$$

$$t_n = h(y_n, \epsilon) \qquad h \text{ distortion function} \tag{2}$$

where $\epsilon$ is the noise that is corrupting the data.

Our goal - the goal of the training - is to find the best $f$ function. For this we:

- Collect a **dataset** (pairs of $x_i$ inputs and $y_i$ outputs)
- Assume a function class that we find $f$ in, like polynomials, Fourier-bases, wavelets, or sigmoids.
- Infer the optimal function from the class using the dataset and the parameters of the function.

The parameters of the function that identify the optimal function based on the components above are called **latent variables**. (Also in literature these are usually called **learnable parameters**) We use the $\theta$ notation for these varibles which contain most importantly the weights and the biases. !UNSURE!

## 3.3 Linear Fitting, Perceptron

One of the simplest types of function fittings is linear regression. When our data is one dimensional (we have predictions for one data point) we can fit a

line

$$f(\mathbf{x}) = \theta_0 + \theta_1 x_1 \tag{3}$$

If we have a two dimensional space, our linear prediction will be a plane in a 3-D space

$$f(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \tag{4}$$

Usually our data in machine learning problems is high dimensional. Let us mark our data as a $D$ dimensional column vector. This can be data that was originally a vector (e.g. single-channel time series data) but it could be a matrix/tensor converted to a vector (e.g. grayscale image that has its columns placed after each other). The equation for this linear fitting can be written as follows:

$$f(\mathbf{x}) = \theta_0 + \theta_1 x_1 + ... + \theta_D x_D \tag{5}$$

$$f(\mathbf{x}) = \theta_0 + \boldsymbol{\theta} \cdot \mathbf{x} \tag{6}$$

This is the idea behind the most basic and oldest neural networks, the single layer perceptrons where it could be used for classification as follows:

$$f(x) = \begin{cases} 1, & \text{if } \theta_0 + \boldsymbol{\theta} \cdot \mathbf{x} > 1 \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Often, $\boldsymbol{\theta}$ is called the weight vector $\mathbf{w}$ and $\theta_0$ is the bias $b$

$$f(x) = \begin{cases} 1, & \text{if } \boldsymbol{w} \cdot \mathbf{x} + b > 1 \\ 0, & \text{otherwise} \end{cases} \tag{8}$$