Introduction to Deep Learning

Bartek Wilczyński
University of Warsaw
bartek@mimuw.edu.pl
http://regulomics.mimuw.edu.pl

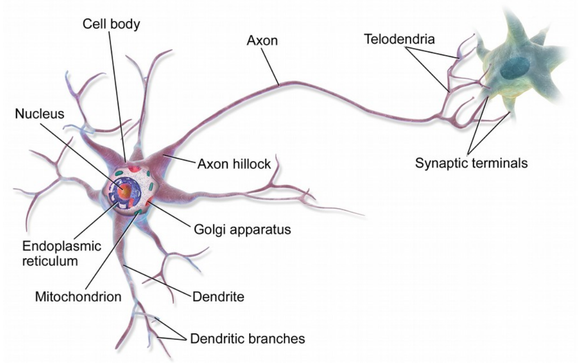
Overview of this lecture

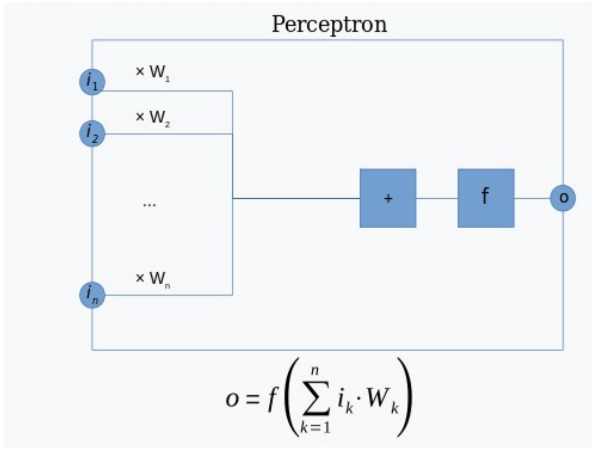
- Background on Artificial Neural Networks
  (why, what, when and how of ANNs)
- Deep learning system basics
  (how are modern ANN models built and trained)
- Important ANN arechitectures
  (typical ANN architectures)

Section I – ANN backstory

ANN backstory: McCulloch and Pitts Perceptron (1943)

- At the very beginning of research on computing machines, people had ideas of following the architecture of the biological neuron
- A computing machine would be built from a lot of artificial neurons (called perceptrons by McCulloch and Pitts)
- Each perceptron would aggregate many signals from other neurons, by computing a weighted sum of the inputs (weights w_i are parts of the Θ, as well as its bias b)
- Then apply a monotonic activation function (originally a step function) f to this sum, and calculate its ouput that is then connected as input to other perceptrons
- Perceptron machines were built in 1950's by Rosenblatt and were able to perform some computational tasks, however, their limitations, i.e. their inability to handle non-monotionic functions, such as XOR, made them at the time inferior to computers with Von Neuman architecture

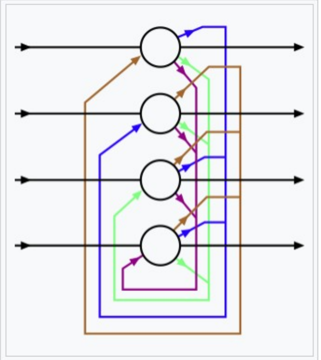


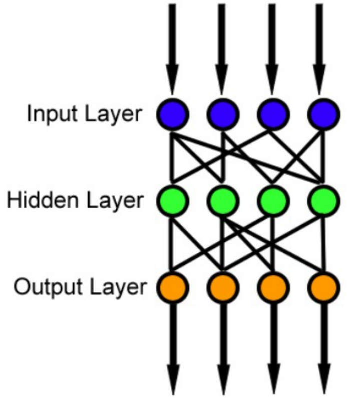


$$o = f\left(\sum_{k=1}^{n} i_k \cdot W_k\right)$$

ANN backstory – 1980's: Hopfield nets and Back-propagation in Feed-forward networks

- After a decade of lowered interest in neural networks (1970's – the first AI "winter") two ideas brought the neural networks back into the forefront of the computing research
- **Hopfield networks** were shown to converge to non-linear functions by John Hopfield
- Multi-layered feed forward networks were shown not only to be able to represent non-linear functions, but they were relatively easily trainable using the **back-propagation algorithm**
- The **back-propagation algorithm** is simply an application of a chain-rule to compute the gradient of the loss function with respect to all the weights in the network. Then, we can use **gradient descent** methods to identify how to change the weight values to lower the Loss function.
- Even though these were essentially all the necessary theoretical tools for modern ANNs, the computing infrastructure was deeply insufficient for effective ANN learning, so the field went back into the "second AI winter" of the 1990's


A Hopfield net with four nodes



ANN backstory: image recognition and modern deep learning breakthrough

- In the late 1990s, the availability of the large training datasets and vast improvements to the computing infrastructure allowed for a *renaissance* of Neural Network research
- In the 1990s pioneering work of Yan Le Cun on image processing using convolutional neural networks lead to the development of working ANNs for handwritten letter recognition in 1998 (LeNet5)
- Around 2006, availability of GPU computing platform such as CUDA provided a leap in ANN learning performance
- In 2012, improved CNN model AlexNet developed by Knizhevsky et al outperformed all other approaches by 10% in the ImageNet image classification challlenge. AlexNet, among other improvements pioneered usage of the simplified **RelU activation unit**
- In 2014 recurrent neural network seq2seq was succesfully used for machine translation
- Since 2017, transformer networks were used to the language processing leading to the current explosive growth of Large Language Models such as ChatGPT

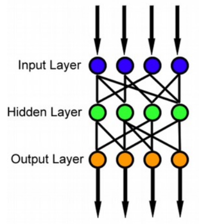
Section II – Modern deep learning

Overview of section II

- Multi-layered feed-forward networks and back-propagation
- Types of activation units: Sigmoid, step function and RelU
- Different optimization methods: SGD, momentum and ADAM
- Dropout as an example of regularization in ANNs

Multi-layered (deep) feed-forward neural networks

- Technically, any network consisting of at least three layers (including input and output), i.e. **containing at least 2 stacked layers of perceptrons** can be considered a “deep” neural network
- In such networks, the middle, **hidden layers**, are taking the outputs of the previous layers as inputs.
- Today, typical model (such as ImageNet → fig.) can have dozens or even hundreds of such layers
- If we allow connections between all neurons in consecutive layers, we call them **fully connected layers**
- The output of these network can be computed very quickly by feeding forward the values from inputs through all layers to the output.
- Current CPU/GPU hardware can do it very fast, especially if the activation function is as simple as ReLU



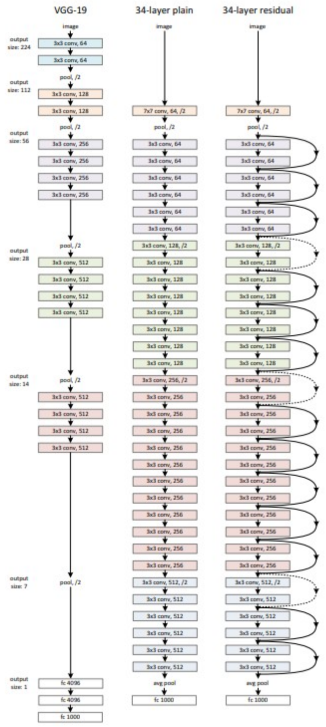


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

Backpropagation algorithm

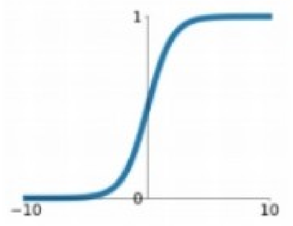
- Assuming that we have a set of training examples, where both inputs $X=\{x_i\}$ and outputs $Y=\{y_i\}$ are given, for any network computing a function $f(x_i,\Theta)$ we can compute the Loss function $L(f(x_i,\Theta),y_i)$
- The Loss function is basically giving us some well behaved difference between the response our network gives us with the weights Θ and the desired output
- Since our function f is just a nested application of the perceptron activation function, and the Loss function is just applied to f and y, the gradient (a matrix of partial derivatives with respect to all weights and biases) of the loss function can be computed efficiently for any pair $<x_i, y_i>$
- The gradient value gives us for each parameter (weight or bias) the direction it should be changed to make the loss function lower for a given example $<x_i, y_i>$
- In practice, this is done layer by layer, starting from output to the first perceptron layer to make the computation more efficient

Types of activation functions

- Originally proposed activation function for perceptrons was the step function
- At the inception of back-propagation algorithm the most widely used were the sigmoid and tanh functions due to their differentiability
- Currently, the ReLU (rectified linear unit) function is the most popular due to its simplicity and ease of computation – its values (for propagation) are just x or 0 and its derivatives (for backpropagation) are just 0 or 1

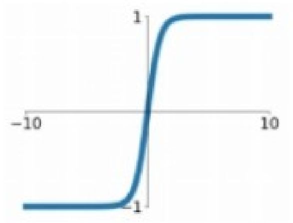
Sigmoid

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



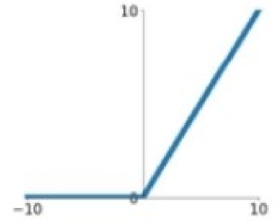
tanh

$\tanh(x)$



ReLU

$\max(0, x)$



Stochastic gradient descent algorithm

- Now that we have the gradient of the loss function for any observation, we need to find an efficient way to create a learning loop
- The simplest method is to take the loop over all observations (an epoch) and adapt the weights according to the gradient function with some learning rate ($\eta \ll 1$, e.g. 10^{-5})
- If we do this in random order in each epoch, this is a variant of a method called Stochastic Gradient Descent (SGD)
- Usually we need multiple learning epochs to reach convergence

```
Choose Initial weights and biases $\Theta$
until change in Loss is low enough:
   #new epoch
   For each training pair $<x_i, y_i>$:
       Calculate the gradient $\nabla L(f(x_i, \Theta), y_i)$
       for each parameter $w$ in $\Theta$:
            w = w - $\eta \nabla L(f(x_i, \Theta), y_i)$ (w)
```

- For technical reasons (memory size in GPU) usually epochs will be divided further into batches, so that the gradient is averaged over multiple samples that are loaded into GPU all at once

Adaptive optimizers momentum and ADAM

- SGD is applicable practically to any data, but it cannot adapt to the fact that some weights have similar gradients for many observations, while others are oscillating
- The **momentum** algorithm simply adds a rescaled gradient from the previous step to the current gradient – this leads to faster learning of weights with consistent gradients and slower learning for weights with oscillating gradients
- ADAM (ADAptive Moment estimation) takes this idea further, by looking at first and second moments of the gradient and adding two “forgetting” factors β_1 and β_2 to the estimation process.

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)}$$
$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)\left(\nabla_w L^{(t)}\right)^2$$

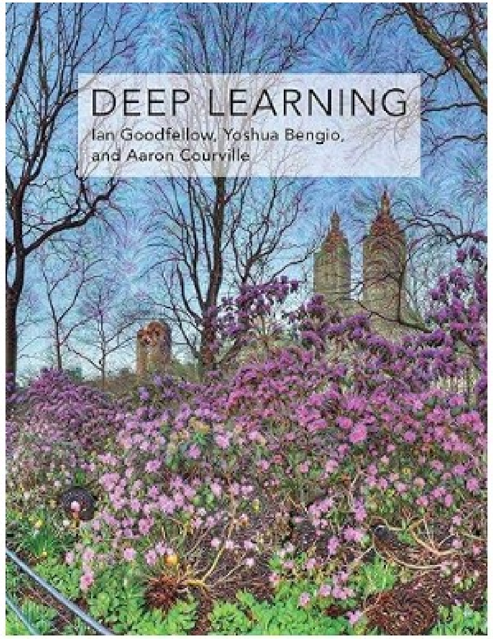
- This method is very effective at differentiating effective learning rates for weights with consistent and inconsistent gradients, while remaining relatively effective

Regularization in deep learning: Dilution&dropout to prevent overfitting

- Typically, deep learning models have very large capacity, and as such are prone to over-fitting,
- In general, as the network weights are parameters in deep learning, we can use many regularization techniques that are used in standard ML (e.g. L_1, L_2)
- However, arguably the most popular regularization techniques for deep learning are based on randomized removal of subsets of weights during training
  - In case of dilution – during each training epoch (or batch) we select a random subset (the size of this subset is a learning parameter) of weights that will not get updated
  - In case of dropout – we select all weights associated with a certain subset of neurons that will not be used at all
- Dropout is a very robust technique that prevents overfitting by requiring the flow of information to be distributed through multiple nodes.
- It slows down the process of training, but usually it is worth to wait longer to have a model with much better generalization

More on modern deep learning theory

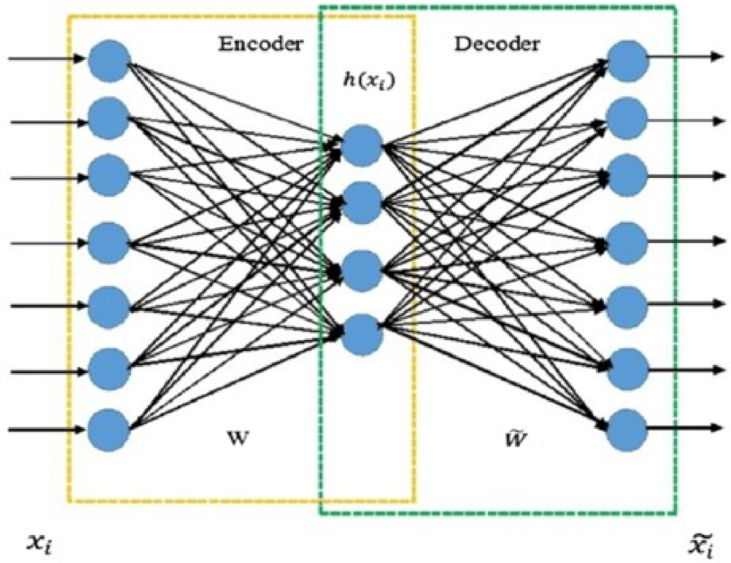
- The “Deep learning” book by Goodfellow et al. is very good for getting more details on the theory of deep learning
- You can buy the printed book from MIT Press or bookstores
- There is a free online version at: https://www.deeplearningbook.org/



Section III –
Popular network architectures

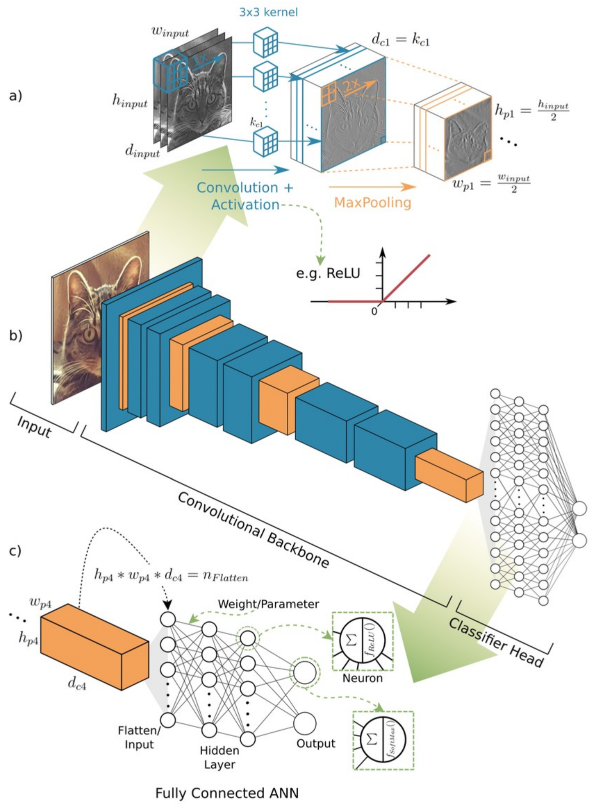
Autoencoders

- Autoencoders are essentially ANNs for dimensionality reduction
- They crucially contain one layer that is lower dimensionality than the input
- They are trained by assessing if the decoder network can reconstruct the input from the reduced latent representation
- Their architecture (of the decoder and encoder) can be quite diverse
- They are very useful as a part of larger ANNs



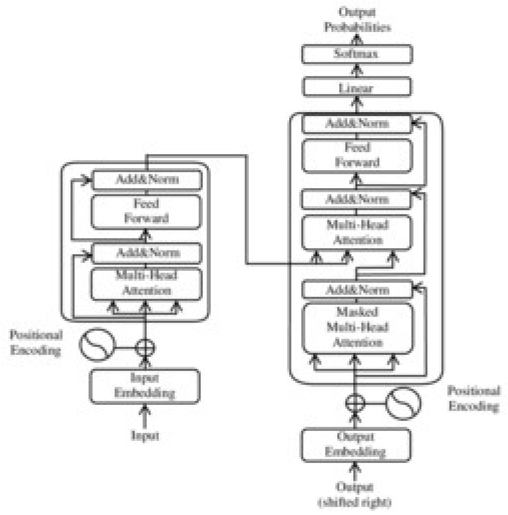
Convolutional Neural Networks

- Convolutional neural networks are very useful for image analysis
- They use the notion of a filter – i.e. a linear transformation that is applied to all positions in an input image in parallel
- While filters have been used for a long time in image analysis, the notion of training filter weights in a CNN has proven to be crucial



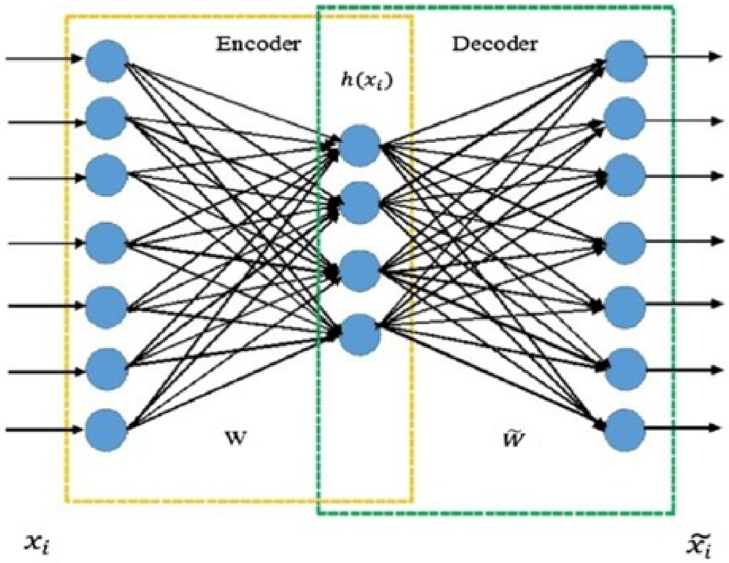
Tranformers

- Transformer networks are complex models that were proposed in 2017 in a seminal paper by Vaswani et al “Attention is all you need”
- They employ a multi-head attention mechanism that allows for contextualised latent state of a model based on a sequence of input tokens
- Large Language models (like ChatGPT) are based on transformers and are trained by feeding them with large bodies of sequences with the objective of filling in the gaps



Autoencoders

- Autoencoders are essentially ANNs for dimensionality reduction
- They crucially contain one layer that is lower dimensionality than the input
- They are trained by assessing if the decoder network can reconstruct the input from the reduced latent representation
- Their architecture (of the decoder and encoder) can be quite diverse
- They are very useful as a part of larger ANNs


Encoder
Decoder
$h(x_i)$
W
$\tilde{W}$
x_i
$\tilde{x}_i$