NTNU | Norwegian University of Science and Technology

DEEP LEARNING
DEEP NEURAL NETWORKS

Tor Andre Myrvoll

2020-10-05

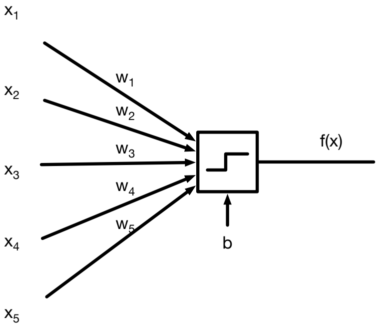
Introduction - Outline

- Overview of the coming lectures
- What are deep neural networks (DNN) ?
- Examples of DNN applications
- Learning DNN parameters

Introduction - Overview

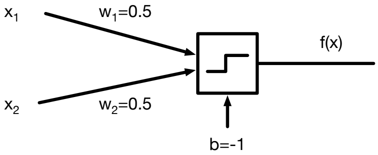
- Lecture 1: Multilayer perceptrons and back-propagation
- Lecture 2: Training DNNs - best practice
- Lecture 3: Convolutional Neural Networks (CNN)
- Lecture 4: Generative Adversariel Networks (GAN). Recursive Neural Networks (RNN)

Introduction - First there was the perceptron...



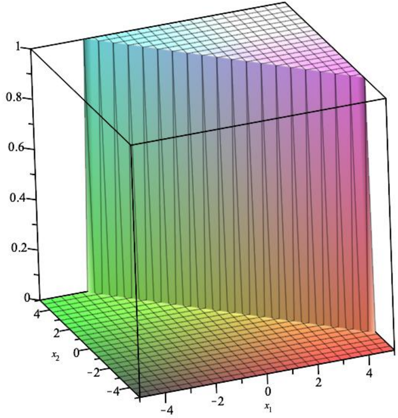
$$f(x) = \begin{cases} 1 & \sum_i w_i x_i + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Introduction - First there was the perceptron...



$$f(x) = \begin{cases} 1 & 0.5(x_1 + x_2) - 1 > 0 \\ 0 & \text{otherwise} \end{cases}$$

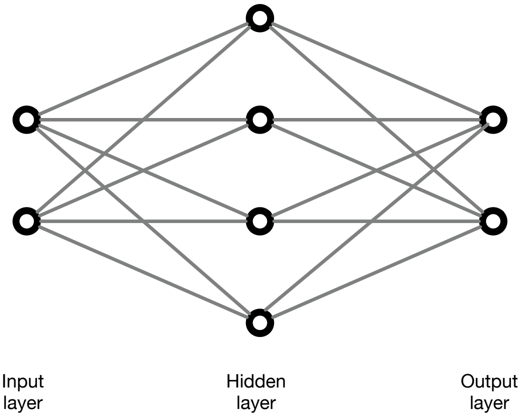
Introduction - First there was the perceptron...



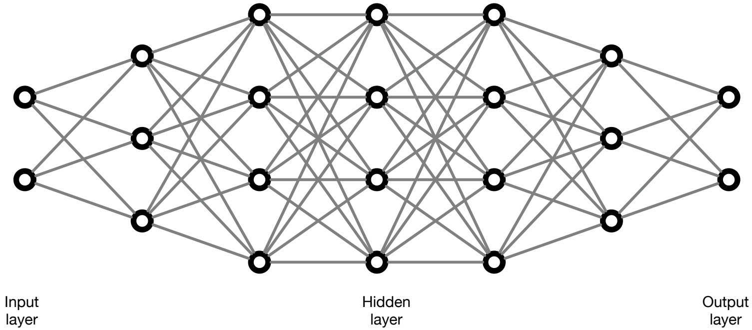
Introduction - First there was the perceptron...

- The perceptron was very popular in the 1950s, but problems with generalization to more complex problems made it fall out of vogue
- Over the years the two major ideas made the use of neural networks feasible:
  - Multiple layers of perceptrons made the networks more powerful
  - Differentiable processing functions made optimization easier
- Today one of the most commonly used neural networks is the *multilayer perceptron*

Introduction - What are deep neural networks



Introduction - What are deep neural networks



Introduction - What are deep neural networks

- Deep Neural Networks are in essence functions/mappings
- The mappings are defined by the parameters of the DNN
- Parameters are learned from data using a training algorithm
- DNNs can express a wide variety of mappings
  - Regression problems: $f : \mathbb{R}^m \to \mathbb{R}^n$
  - Posterior distributions: $f : \mathbb{R}^m \to \{p_1, p_2, \ldots, p_n\}$, $p_i > 0 \forall i$, $\sum p_i = 1$
  - Multilabels: $f : \mathbb{R}^m \to [0, 1]^n$, where $p \in [0, 1]$ indicates to what extend a label is present

Simple network structures like the one shown in the previous slide are often called feed-forward networks or multilayer perceptrons

Introduction - What are deep neural networks

Multilayer perceptrons with one or more layers are *universal approximators*:

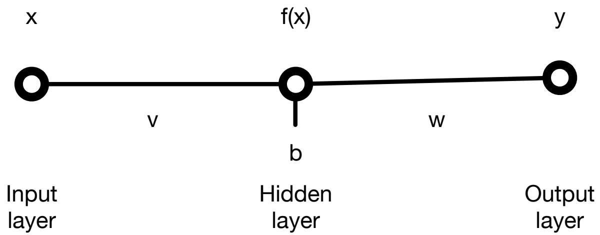
- Assume $x \in E \subset \mathbb{R}^m$, where E is a closed and bounded subset.
- Let $f : E \to \mathbb{R}^n$ be a smooth function defined on E.
- It can then shown that for every $\varepsilon > 0$ there always exists a neural network i with N parameters implementing a function $g : E \to \mathbb{R}^n$, so that $|f(x) - g(x)| < \varepsilon$.

This means that we can be sure that whatever relationship our data represents, our neural network can model it given enough parameters.

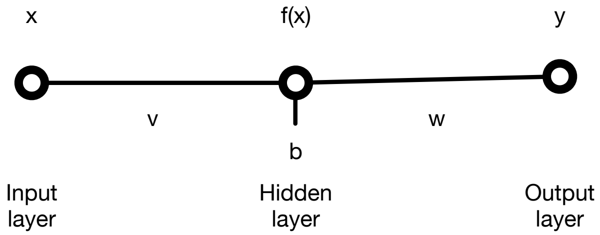
Introduction - What are deep neural networks

- In practice however, this theorem *seems* of little use, as a single layer network will grow exponentially when we increase the dimensionality of the domain, $\mathbb{R}^m$.
- In other words, going from $\mathbb{R}^m$ to $\mathbb{R}^{m+1}$, will on average increase the number of parameters to obtain an accuracy within $\varepsilon > 0$ by a factor $C > 1$.
- This may not sound bad if $C = 1 + \delta$, but consider classifying a 512×512 pixel image?! ($(C + \delta)^{512 \times 512} = (C + \delta)^{262144}$ will be a large number unless $C \approx 1$)
- This is where the power of *deep neural networks* enters. Experiments and theoretical analysis shows that incresing the *depth* of a neural network, will at some point dramatically decrease the number of parameters needed to approxiate a given function.

Multilayer perceptrons - Simplest MLP ever



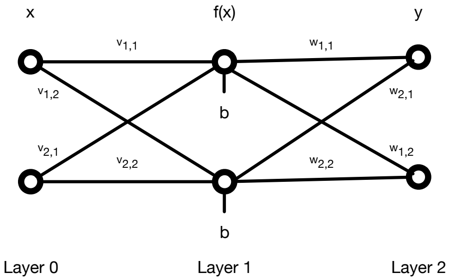
Multilayer perceptrons - Simplest MLP ever



The input-output relationship of this network is:

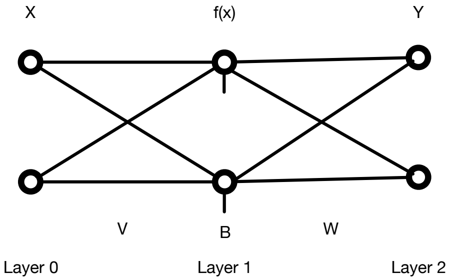
$$y = w \cdot f(v \cdot x + b)$$

Multilayer perceptrons - Slightly more complicated MLP



The input-output relationship of this network is already much more complicated.

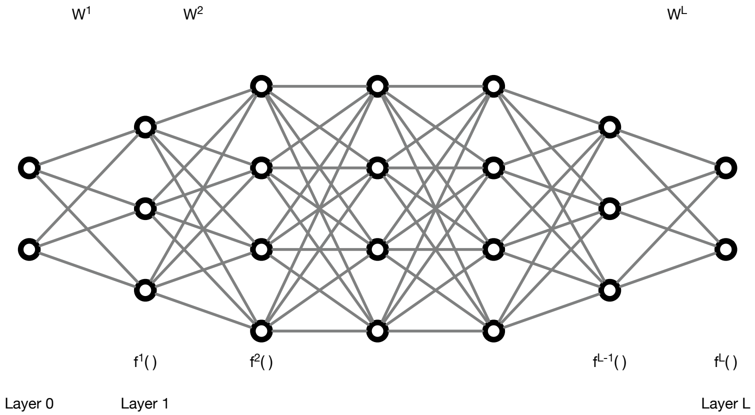
Multilayer perceptrons - Slightly more complicated MLP



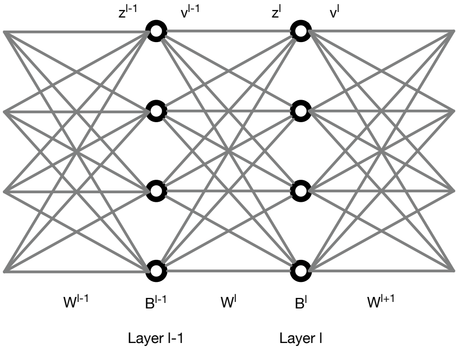
Using matrix notation:

$$Y = W^T \cdot f(V^T \cdot X + B)$$

Multilayer perceptrons - General MLP



Multilayer perceptrons - General MLP closeup



- Weight matrix W^l
- Bias vector B^l
- Excitation vector
  $z^l = W^{l,T} \cdot v^{l-1} + B^l$
- Activation vector $v^l = f(z^l)$

Activation functions - Hidden layer functions

Activation functions:

- The sigmoid:

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

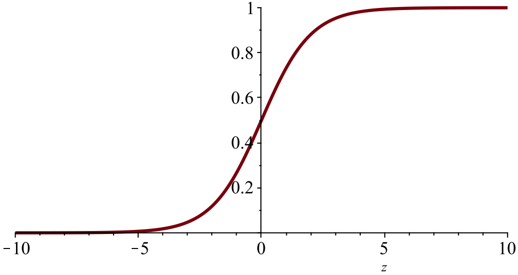
- The hyperbolic tangent:

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

- The rectified linear unit:

$$\mathrm{ReLU}(z) = \begin{cases} 0, & z < 0 \\ z, & 0 \leq z \end{cases}$$

Activation functions - Sigmoid

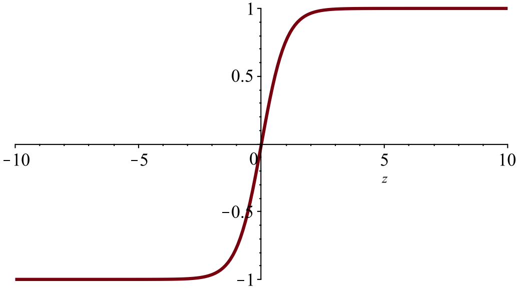


- Function: $\sigma(z) = \frac{1}{1+e^{-z}}$
- Continuous approximation to the classical perceptron function

$$\text{per}(z) = \begin{cases} 0, & z < 0 \\ 1, & 0 \leq z \end{cases}$$

- Used to be the most popular activation function for MLPs

Activation functions - Hyperbolic tangent

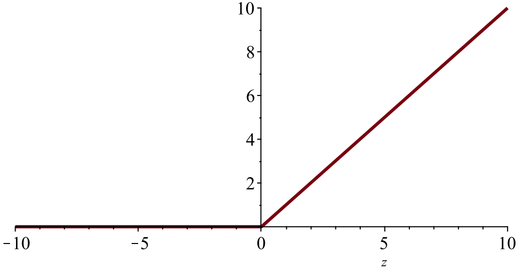


- Function: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- Symmetric version of the sigmoid

$$\text{per}(z) = \begin{cases} 0, & z < 0 \\ 1, & 0 \leq z \end{cases}$$

- Popular alternative to the sigmoid

Activation functions - Rectified Linear Unit



- Function: $\mathrm{ReLU}(z) = \begin{cases} 0, & z < 0 \\ z, & 0 \leq z \end{cases}$
- The most common activation function today
- Training ReLU based networks faster and easier

Activation functions - Final layer functions

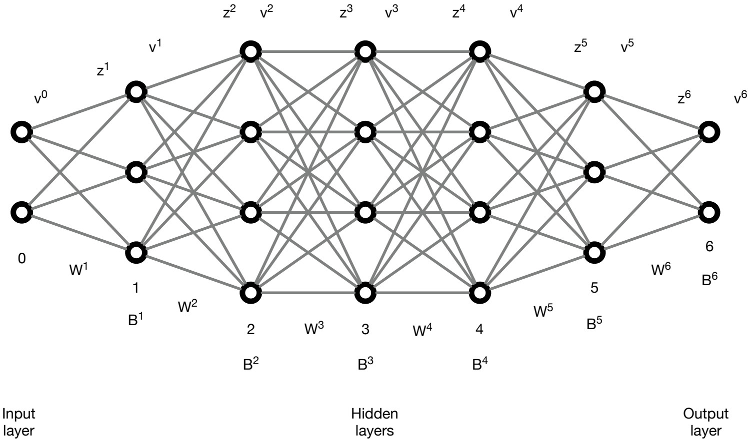
The functions used in the final layer, the output, of the network, varies from the functions used in the hidden layers. Examples:

- Linear output: $f(z) = z$. Mostly used in regression where one typically wants unbounded output.
- Softmax: Let $\{z_i\}$ be the elements of the exitation vector z. Then the ith output from the softmax function is

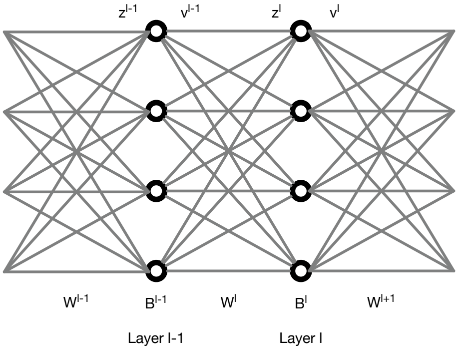
$$f_i(z) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

  We see that the elements of the softmax function are positive and sums to one. Used for classification problems.
- Sigmoid: Same as for hidden layers. Used in multi-label classification.

Training the network - The forward computation



Training the network - The forward computation



- Given an input $x \in \mathbb{R}^m$, we want to compute the output, $f(x)$ of the neural network.
- This is done using a simple forward loop over the layers of the networks
- As we we do the forward computation we will store the intermediate results for use with the *backpropagation algorithm* that we will be discussing after this

Training the network - The forward computation

$v^0 \leftarrow x$
for $l \leftarrow 1, L$ **do**
    $z^l \leftarrow W^l v^{l-1} + B^l$
    $v^l \leftarrow f^l(z^l)$
end for
if Regression **then**
    $v^L \leftarrow z^L$
else
    $v^L \leftarrow \text{softmax}\left(z^L\right)$
end if

Training the network - The objective function

The MLP is determined by the following:

- The network topology
  - Number of layers
  - Number of nodes per layer
- The choice of nonlinearities $f(x)$
- The weights $\{W^l\}$ and biases $\{B^l\}$

The network topology and nonlinearity is usually chosen in advance and held fixed while we optimize the network parameters.

Training the network - The objective function

Given a training set $X = \{x_n\}$, $Y = \{y_n\}$. To train an MLP we need to define an *objective function* that measures the discrepancy, or error, between the output of an MLP, $\hat{y}_n = f(x_n)$, and the *true value* from the training set, y_n.

- We define a *loss function* $l(y, \hat{y})$, that measures the loss when we predict $\hat{y}$ as opposed to y.
- Examples of loss functions are:
  - The squared error: $l(y, \hat{y}) = \|y - \hat{y}\|^2$. This is a good choice for regression problems
  - The cross entropy loss: $l(y, \hat{y}) = \sum_i -y_i \log \hat{y}_i$. This is the most common choice for classification problems where y is a vector of $0 - 1$ labels.

Training the network - Learning the parameters

- Given a loss function we can define the *total loss* for the training problem:

$$L(W, B, X, Y) = \frac{1}{|X|} \sum_n l(y_n, f(x_n; W, B))$$

- Our goal is now to find the network parameters W, B so that

$$\hat{W}, \hat{B} = \arg\min_{W,B} L(W, B, X, Y)$$

- It is not possible to *solve* this in closed form, or even approximately.
- The solution is to use *numerical search*, or more specifically, *gradient descent*

Training the network - Gradient descent

Assume we have a function $f(x)$ that we want to minimize with respect to x. That is, we want to find

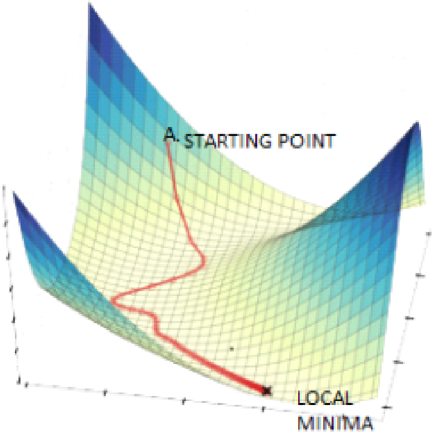
$$x^\star = \arg\min_x f(x)$$

The function is too complicated to solve directly, however, we can compute the gradient at any point x. Let $f(x^{(0)})$ be an initial guess of the minimum of the function. We know that the gradient $\nabla_x f(x)$ points towards larger $f(x)$, while the negative gradient points towards smaller $f(x)$. Let our next guess for $x^\star$ be

$$x^{(1)} = x^{(0)} - \epsilon \nabla_x f(x)$$

For small ϵ we are guaranteed that $f(x^{(1)} < f(x^{(0)})$, and so we can create a sequence of updates that may converge to the optimal value.

Training the network - Gradient descent



Training the network - Gradient descent

We plan to optmimize the MLP parameters in the following way:

1. Initialize $W^{(0)}, B^{(0)}$, eg. by using Gaussian noise
2. Iterate until convergence:

$$
\begin{aligned}
W^{(i)} &= W^{(i-1)} - \epsilon \nabla_W L(W, B, X, Y) \\
B^{(i)} &= B^{(i-1)} - \epsilon \nabla_B L(W, B, X, Y)
\end{aligned}
$$

To achieve this we need to be able to compute $\nabla_W L(W, B, X, Y)$ and $\nabla_B L(W, B, X, Y)$.

$$
\nabla_W L(W, B, X, Y) = \frac{1}{|X|} \sum_n \nabla_W l(y_n, f(x_n; W, B))
$$

(B is computed in a similar manner)

Training the network - The backward propagation algorithm

It turns out that the gradients can be computed in an efficient manner using *back propagation*

1. Initilize the network
2. For each iteration
   - **2.1** Do the forward computation
   - **2.2** Compute the gradient wrt. the last set of weights, W^L for each training example x_n, y_n
   - **2.3** It can now be shown that the gradient wrt. W^l depends on the gradient wrt. W^{l+1}. Hence, we can start at the end and propagate the gradient backward to the beginning of the network

Training the network - Back propagation

In a little more detail (discussing W only, B is similar):

- For each layer l we can compute

$$\nabla_{W_t^l} L(W, B, X, Y) = \left[f'(z_t^l) \cdot e_t^l \right] (v_t^{l-1})^T$$

  where

$$e_t^l = (W_t^l)^T (f'(z_t^l \cdot e_t^{l+1}))$$

  starting with

$$e_t^L = (V_t^L - y)$$

Summary - Wrapping up

- Neural networks are functions
- Universal approximation means they can model anything given by a mapping
- They can be efficiently trained using back propagation

Summary - Next time

- Best practices for neural network training
- Batch, mini-bacth, stochastic gradient descent
- Regularization, drop-out
- Normalization
- And more...

Thank you for your attention

NTNU | Norwegian University of Science and Technology