Deep Feedforward Networks and MLPs

Tirtharaj Dash

Dept. of CS & IS and APPCAIR
BITS Pilani, Goa Campus

September 2, 2021

MLPs I

- Other names: Feedforward Neural Networks, Deep Feedforward Neural Networks
- Special class of models: Deep Fully-Connected Feedforward Neural Networks, Multilayer Perceptrons.
- Goal: To approximate some function f^*.
- For example, for a classifier, $y = f^*(\mathbf{x})$ maps an input $\mathbf{x}$ to a category y.
- MLP defines a mapping $\mathbf{y} = f(\mathbf{x}; \mathbf{w})$ and learns the value of the parameters $\mathbf{w}$ that results in the best function approximation.

Note: $f(\mathbf{x}, \mathbf{w})$ and $f(\mathbf{x}; \mathbf{w})$ are often used interchangeably. Usually, $f(\mathbf{x}; \mathbf{w})$ means that the function f is parameterised by $\mathbf{w}$ and $\mathbf{x}$ are input variables. That is, in genreal, $f(vars; params)$. By supplying the values of *params* we create a new function f.

MLPs II

- Feedforward:
  - Information flows through the function being evaluated from $\mathbf{x}$
  - Intermediate computations used to define f
  - To the output $\mathbf{y}$
- Networks:
  - Represented by composing together many different functions
  - The model is associated with a directed acyclic graph (DAG) describing how the functions are composed.
  - Example: Chain structure

$$f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$$

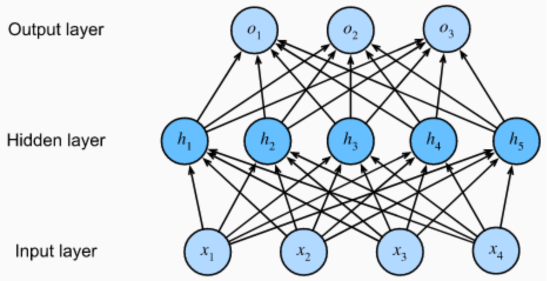
  - $f^{(i)}$: ith layer of the network
  - Length of the chain = *depth* of the model
  - Final layer: Output layer

MLPs III

- Computation:
  - Training data: Provides noisy, approximate examples of $f^*(\mathbf{x})$ evaluated at each data point. Each $\mathbf{x}$ is paired with a label $y \approx f^*(\mathbf{x})$.
  - Training: derive $f(\mathbf{x})$ to match $f^*(\mathbf{x})$.
  - Output layer must produce a value close to y (this is specified by training data)
  - The behaviour of other layers is not specified by the training data. (Tha is why they are called: *Hidden layers*)
  - Learning algorithm: decides how to use the intermediate layers and what each layer should do to best approximate f^*

Incorporating Hidden Layers I

- An MLP with 1 hidden layer of 5 hidden units



- depth(=number of layers): 2
- width: 5
- Both the layers are fully-connected.

Incorporating Hidden Layers II

Some notations:

- x: a scalar
- $\mathbf{x}$: a vector
- $\mathbf{X}$: a matrix
- X: a general tensor
- x_i: the ith element of vector $\mathbf{x}$
- x_{ij}: the element at $[i, j]$ position of matrix $\mathbf{X}$

(We will be subscribing to Textbook 1 for these notations.)

Incorporating Hidden Layers III

Now:

- Let $\mathbf{x} \in \mathbb{R}^d$ denote an example; d is the number of features (inputs)
- Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote a minibatch of n examples
- For our one-hidden-layer MLP with h hidden units: Let $\mathbf{H} \in \mathbb{R}^{n \times h}$ denote the outputs of the hidden layer. These are called *hidden representations*.
- Since both the layers are fully-connected:

$$\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h} \text{ and } \mathbf{b}^{(1)} \in \mathbb{R}^{1 \times h}$$

- Similarly, weights in the output layer (with q units):

$$\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q} \text{ and } \mathbf{b}^{(2)} \in \mathbb{R}^{1 \times q}$$

Incorporating Hidden Layers IV

- The forward computations are then:

$$\mathbf{H} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}$$

and

$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

- So, what happened here? – Write **O** directly in terms of **X** and see what happened.

Incorporating Hidden Layers V

- Here:

$$\begin{aligned}\mathbf{O} &= (\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)} \\ &= \mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)} \\ &= \mathbf{X}\mathbf{W} + \mathbf{b}\end{aligned}$$

- This turned out to be an affine transformation of inputs ($\mathbf{X}$).
- This collapsed our MLP into a linear model.

Note: An affine function is a composition of a linear function followed by a translation. For $\mathbf{x} \in \mathbb{R}^d$, $a\mathbf{x}$ is linear; $(\mathbf{x} + b) \circ a\mathbf{x}$ is affine.

Incorporating Hidden Layers VI

Adding non-linearity to hidden layers:

- Let σ denote some non-linear *activation* function:

$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$$

  and

$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

- In general:

$$\mathbf{H}^{(\ell)} = \sigma(\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)})$$

  where, $\mathbf{H}^{(0)} = \mathbf{X}$, and ℓ denotes a layer index.

Note: σ is applied element-wise.

Activation Functions I

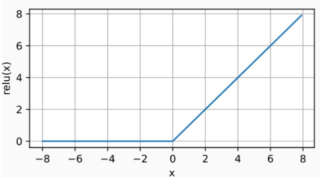
- An activation function decides whether a neuron should be activated based on the net-input a neuron receives.
- They are differentiable operators to *transform* input signals to outputs, while most of them add non-linearity.
- We have already discussed some activation functions in our previous tutorials.

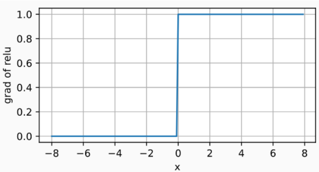
Rectified Linear Unit (ReLU):

- Probably, the most popular choice: simple to implement, reasonably good performance on a variety of predictive tasks
- Given an element x, ReLU is defined as:

$$ReLU(x) = \max(0, x)$$

- ReLU and its gradient:





Note: ReLU is not differentiable at $x = 0$. Hence it needs approximation as follows.

$$\frac{\partial ReLU(x)}{d\partial x} = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$

Activation Functions III

- Benefits:
  - Its derivatives are particularly well behaved: either they vanish or they just let the argument through.
  - It mitigates vanishing-gradient problem (more on this later).
- Other variants: Leaky-ReLU, Parametric-ReLU (refer to some research papers and tutorials for more details).

Activation Functions IV

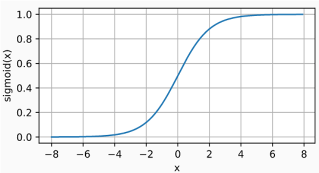
Sigmoid:

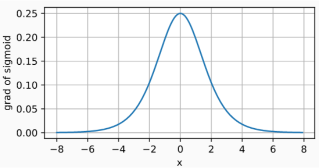
- Popular choice for neural network implementations in pre-deep learning era.
- Smooth, nice gradient
- Defined as (also called *squashing function, as its range is* $[0, 1]$):

$$sigmoid(x) = \frac{1}{1 + \exp(-\lambda x)}$$

usually, $\lambda = 1$; it determines the slant.

- Sigmoid and its gradient:





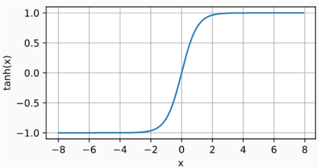
Note: Grad of $sigmoid(x) = sigmoid(x)(1 - sigmoid(x))$

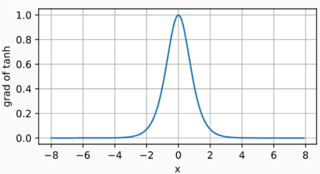
Tanh:

- Another popular choice (even now) in many studies.
- It also squashes the input by transforming it into range $[-1, 1]$.
- Defined as:

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

- Tanh and its gradient:





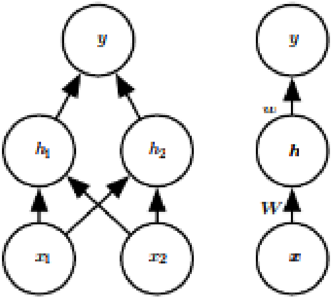
Note: Grad of $\tanh(x) = 1 - \tanh^2(x)$

Re-doing XOR I

- 2-bit XOR ("exclusive or"): $x_{1,2} \in \{0, 1\}$. When *exactly* one of x_1, x_2 is equal to 1, the XOR function returns 1; otherwise, it returns 0.
- In this simple example, we are not concerned with statistical generalisation, rather with closely approximating the XOR function $f^*(\cdot)$.
- That is, our model must perform correctly on 4 data points.

Re-doing XOR II

- We construct the following network:



(Source: Textbook 2)

- This network can be represented as:

$$f(\mathbf{x}; \mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) = \max(0, \mathbf{x}\mathbf{W}^{(1)} + \mathbf{b})\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

Re-doing XOR III

- The matrix and vector dimensions are as follows (based on Textbook 1):
  - $\mathbf{x} \in \{0, 1\}^{1\times 2}$
  - $\mathbf{W}^{(1)} \in \mathbb{R}^{2\times 2}$
  - $\mathbf{b}^{(1)} \in \mathbb{R}^{1\times 2}$
  - $\mathbf{W}^{(2)} \in \mathbb{R}^{2\times 1}$
  - $\mathbf{b}^{(2)} \in \mathbb{R}^{1\times 1}$

I don't like these notations: Here vectors are denoted in $\mathbb{R}^{1\times d}$; whereas, these should really be $\mathbb{R}^{d\times 1}$.

Re-doing XOR IV

- Now, let's provide a solution to these parameter values (based on Textbook 2):
  - $\mathbf{W}^{(1)} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$
  - $\mathbf{b}^{(1)} = \begin{bmatrix} 0 & -1 \end{bmatrix}$
  - $\mathbf{W}^{(2)} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$
  - $\mathbf{b}^{(2)} = \begin{bmatrix} 0 \end{bmatrix}$

  Along with these, let's write our inputs in batch form $\mathbf{X}$:
  - $\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$

Re-doing XOR V

- Hidden representation can be obtained by

$$\mathbf{H} = ReLU(\mathbf{XW}^{(1)} + \mathbf{b}^1) = \begin{bmatrix} \max(0,0) & \max(0,-1) \\ \max(0,1) & \max(0,0) \\ \max(0,1) & \max(0,0) \\ \max(0,2) & \max(0,1) \end{bmatrix}$$

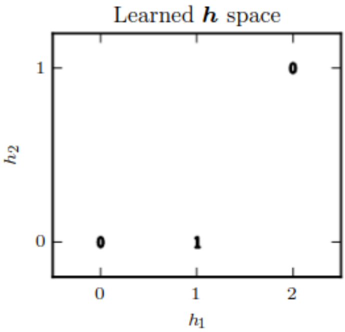
Note that we are writing $ReLU(z)$ as $max(0,z)$.

- This turns out to be:

$$\mathbf{H} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

Re-doing XOR VI

- This hidden-layer transformation changed the representation of the examples. Visualising the *h*-space, we see:



(Source: Textbook 2)

The examples now lie on a space where a linear model can solve the problem. Let us see this next:

Re-doing XOR VII

- Output layer computation is:

$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

  Nice. Isn't it?

- But, here we already provided our solutions to the model parameters. How did we get these?

  (Next lecture: We shall find out.)