Learning From Data
Lecture 5: Deep Neural Networks

Yang Li yangli@sz.tsinghua.edu.cn

TBSI

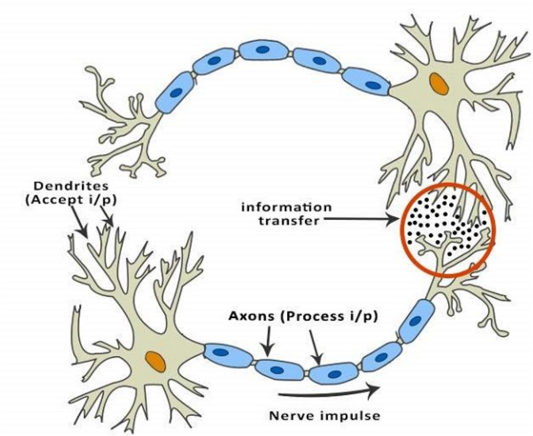
October 18, 2024

Today's Lecture

- Introduction to neural networks
  - Biological motivations
  - A case study
- Training a deep feedforward neural network
  - Forward pass
  - Backward propagation

Biological motivation

Schematic of biological neurons:



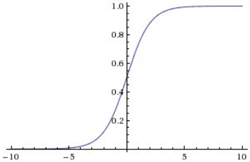
Each neuron takes information from other neurons, processes them, and then produces an output.

Biological motivation

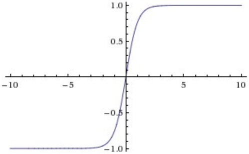
How does a neuron process its input? (a *coarse* model)

- Takes the weighted average of I inputs, e.g. $z = \sum_{i=0}^{I} w_i(x_i)$
- Neuron fires if z is above some threshold

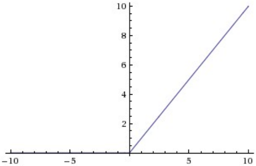
We call the threshold function **activation function**.



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



$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
$= 2(sigmoid(2z)) - 1$

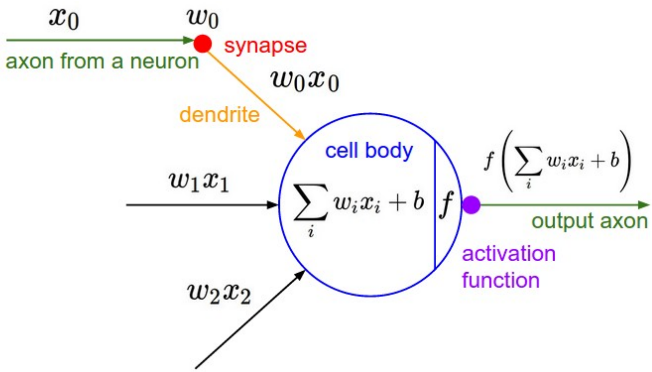


$ReLu(z) = max\{0, z\}$

Rectifying linear unit

Biological motivation

An artificial neuron with inputs x_1, x_2 and activation function f



A single neuron is a (linear) binary classifier:

- When f is the sigmoid function, equivalent to binary softmax
- When f is the sign function, equivalent to the perceptron

Neural networks

- The goal of a neural network is to approximate some function f^* such that $y = f^*(x)$.
- The neural network defines a mapping $y = f(x; \theta)$ and learns the value of parameters θ through training.
- Define **error function** that measures prediction error of f: e.g. a common error function used in classification is the **logarithmic loss** a.k.a. **cross-entropy loss**:

$$L = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

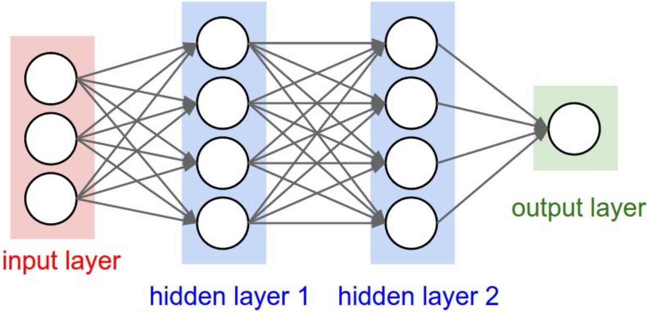
  - $\hat{y} = f(x; \theta)$ is the predicted output
  - y is the true output

A single layer of neurons are unable to approximate complex functions.

A feed forward neural network

In a **feed-forward neural network** (a.k.a. **multi-layer perceptron**), all units of one layer is connected to all of the next layer.

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



- number of layers are called **depth** of the neural network
- number of units in a layer is called **width** of a layer

The XOR problem

XOR : the exclusive or

x_1	x_2	$y = x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

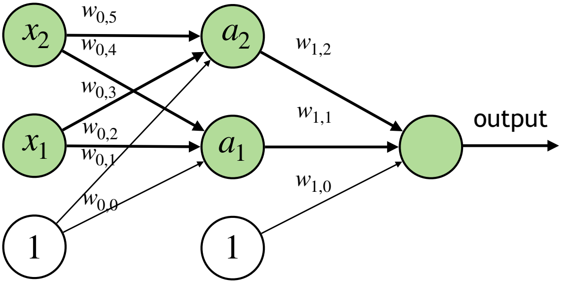
$h(x) = f_2(w_2^T f_1(W_1 x + b_1) + b_2)$

activition function: $f_1(\mathbf{z}), f_2(z)$

network weights: $W_1 = \begin{bmatrix} w_{0,2} & w_{0,4} \\ w_{0,3} & w_{0,5} \end{bmatrix}$, $b_1 = \begin{bmatrix} w_{0,0} \\ w_{0,1} \end{bmatrix}$,

$w_2 = \begin{bmatrix} w_{1,2} \\ w_{1,1} \end{bmatrix}$, $b_2 = w_{1,0}$

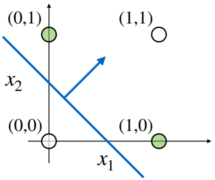
input layer | hidden layer | output layer

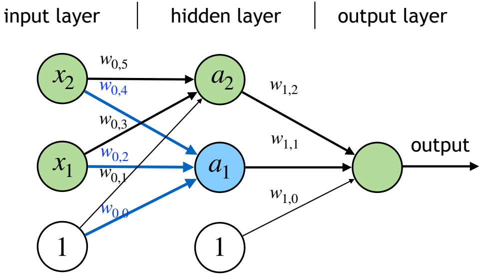


The XOR problem

$$h(x; W_1, b_1, w_2, b_2) = f_2(w_2^T f_1(W_1 x + b_1) + b_2)$$

Suppose $f_1(\mathbf{z}) = \begin{bmatrix} \mathbf{1}\{z_1 \geq 0\} \\ \mathbf{1}\{z_2 \geq 0\} \end{bmatrix}$, $f_2(z) = \mathbf{1}\{z \geq 0\}$. One solution:



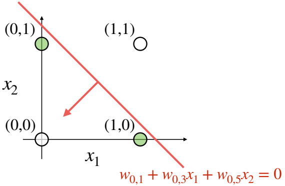


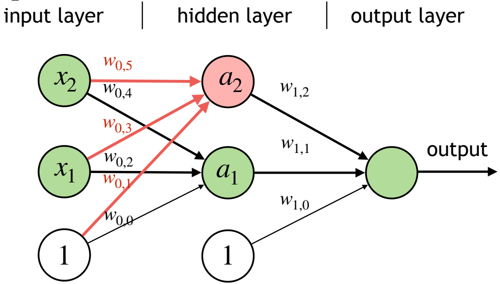
x_1	x_2	a_1
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

The XOR problem

$$h(x; W_1, b_1, w_2, b_2) = f_2(w_2^T f_1(W_1 x + b_1) + b_2)$$

Suppose $f_1(\mathbf{z}) = \begin{bmatrix} \mathbf{1}\{z_1 \geq 0\} \\ \mathbf{1}\{z_2 \geq 0\} \end{bmatrix}$, $f_2(z) = \mathbf{1}\{z \geq 0\}$. One solution:



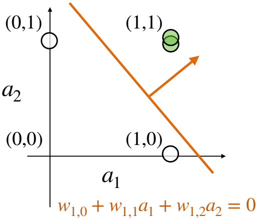


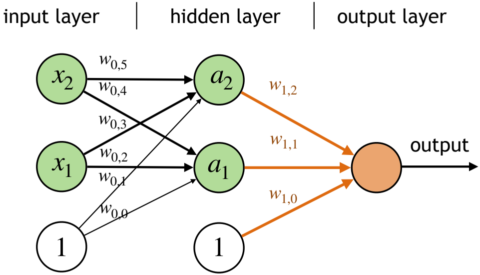
x_1	x_2	a_1	a_2
0	0	0	**1**
0	1	1	**1**
1	0	1	**1**
1	1	1	**0**

The XOR problem

$$h(x; W_1, b_1, w_2, b_2) = f_2(w_2^T f_1(W_1 x + b_1) + b_2)$$

Suppose $f_1(\mathbf{z}) = \begin{bmatrix} \mathbf{1}\{z_1 \geq 0\} \\ \mathbf{1}\{z_2 \geq 0\} \end{bmatrix}$, $f_2(z) = \mathbf{1}\{z \geq 0\}$. One solution:





x_1	x_2	a_1	a_2	y
0	0	0	1	**0**
0	1	1	1	**1**
1	0	1	1	**1**
1	1	1	0	**0**

Universal approximation theorem

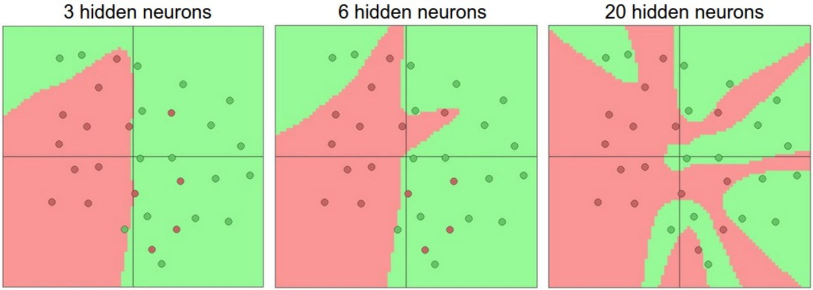
Universal approximation theorem (Cybenko,1989; Hornik et al., 1991) A feed-forward network with a single hidden layer containing a finite number of neurons can approximate any continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function.

- First proved by George Cybenko in 1989 for sigmoid activation function;
- With one hidden layer, layer width of an *universal approximator* has to be exponentially large $\leftarrow$ *More effective to increase the* ***depth*** *of neural networks*
- ReLU networks with width n+1 is sufficient to approximate any continuous function of n-dimensional input variables if depth is allowed to grow. (Lu et. al, 2017; Hanin 2018)

Overfitting

Increase the size and number of layers in a neural network,

- the **capacity** , i.e. representation power of the network increases.
- but overfitting can occur: fits the noise in the data instead of the (assumed) underlying relationship.

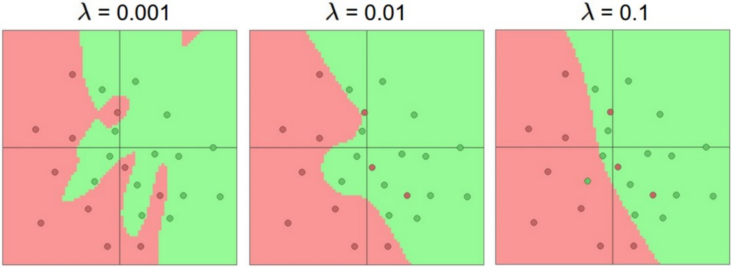


Regularization

One way to control overfitting in training neural networks
A common regularization approach is **parameter norm penalties**

$$\tilde{L}(w; X, y) = L(w; X, y) + \lambda\Omega(w)$$

- L2 parameter regularization: $\Omega(w) = \frac{1}{2}||w||_2^2 = \frac{1}{2}w^T w$ drives the weights closer to the origin



- L1 parameter regularization: $\Omega(w) = ||w||_1 = \sum_{i=1}^{k} |w_i|$ drives solutions more sparse.

Forward pass and Backpropagation

See Powerpoint slides.

Practical issues

Which activation function to use?

- *sigmoid* function $\sigma(z)$: gradient $\nabla f(z)$ **saturates** when z is highly positive or highly negative. Not suitable for hidden unit activation.
- $tanh(z)$: similar to identity function near 0 , resembles a linear model when activation is small, performs better than sigmoid. ($tanh(0) = 0$, $\sigma(0) = \frac{1}{2}$).
- $ReLu(z)$: easy to optimize (6 times faster than sigmoid), often used with affine transformation $g(W^T x + b)$. *Derivative is 1 whenever the unit is active.*

Sigmoidal activation functions *are often preferred than* **piecewise linear activation functions** *in non-feed forward networks. e.g. probabilistic models, RNNs etc*

Additional resources

Deep neural network is a relative young field with lots of empirical results. Read more on the practical things to do for building and training neural networks:

- Stanford Class on Convolutional Neural Networks: `http://cs231n.github.io`
- Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning*, MIT Press, 2016

Demos:

- `http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/`
- `https://playground.tensorflow.org/`