

Title: PSI 2017/2018 Machine Learning for Many Body Physics - Lecture 2

Date: Apr 10, 2018 09:00 AM

URL: <http://pirsa.org/18040055>

Abstract:

Reminder:

ML: machine learning

SL: supervised learning

UL: unsupervised learning

NN: neural network

Today: Intro to SL with feedforward NNs

↳ activation functions

↳ cost functions

↳ algorithms for optimizing the NN's weights and biases
(eg. gradient descent)

↳ backpropagation

SL with NNs

Recall the goal of SL : given

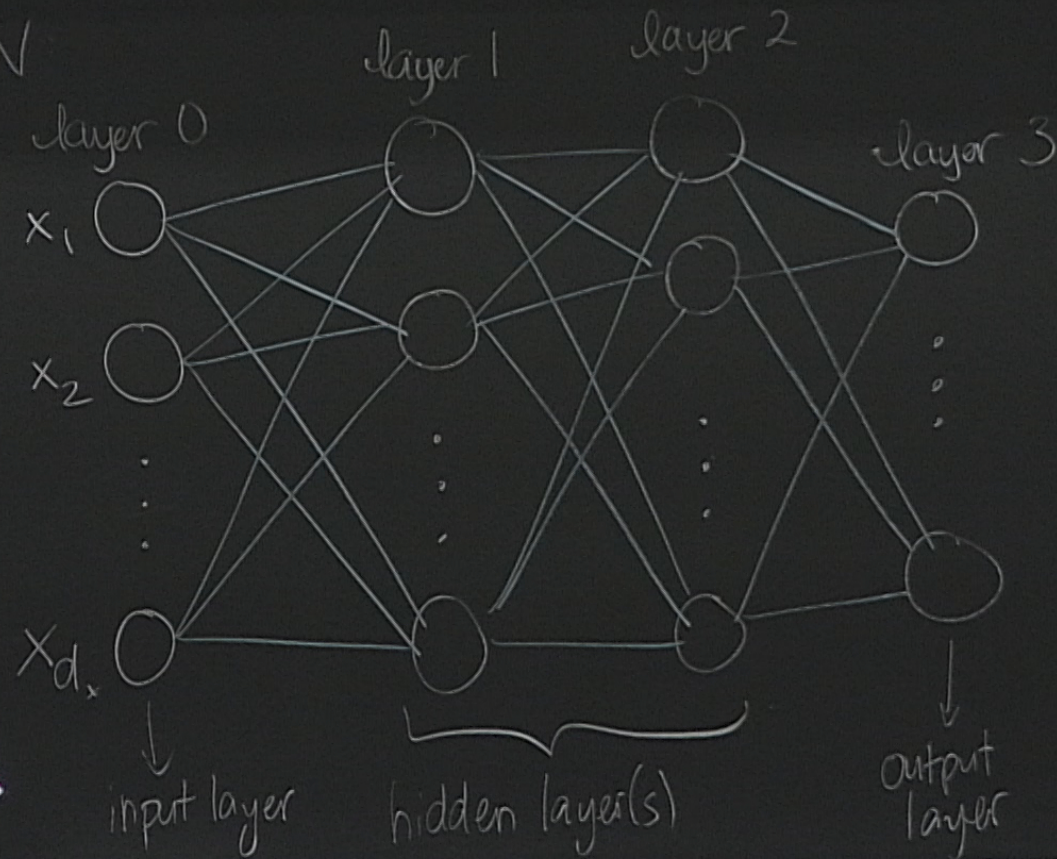
$$\mathcal{D} = \{(\vec{x}, \vec{y})\}, \text{ fit } \vec{f}(\vec{x}) \text{ to } \vec{y}$$

$$\bullet \vec{x} = (x_1, x_2, \dots, x_{d_x})$$

$$\bullet \vec{y} = (y_1, y_2, \dots, y_{d_y})$$

$$\bullet \vec{f} : \mathbb{R}^{d_x} \rightarrow \mathbb{R}^{d_y}$$

Feedforward NN

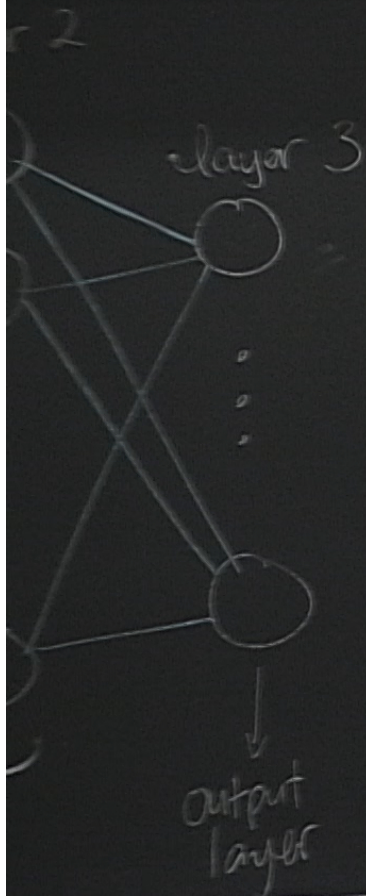


Each $\bigcirc$ is a "neuron"

n_l = # of neurons in layer l

L = largest value of l
($L=3$ here)

$n_0 = dx$, $n_L = dy$



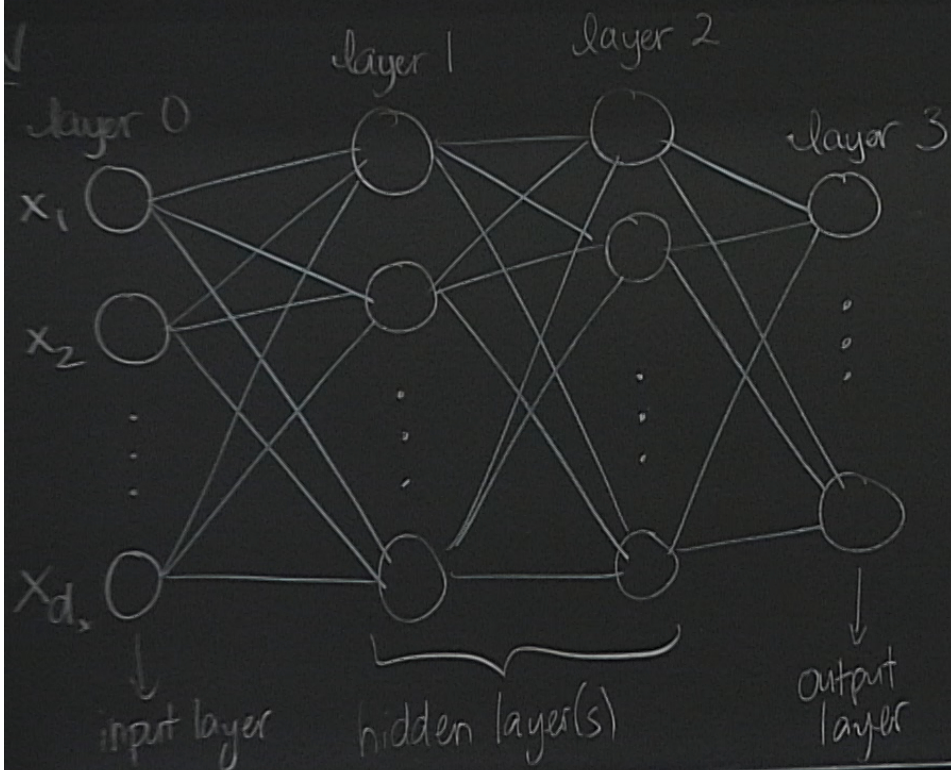
Each $\bigcirc$ is a "neuron"

$n_l = \#$ of neurons in layer l

$L = \text{largest value of } l$
($L=3$ here)

$n_0 = d_x$, $n_L = d_y$

Define $a_i^{(0)} = x_i$ ($1 \leq i \leq d_x$)



Each $\bigcirc$ is a "neuron"

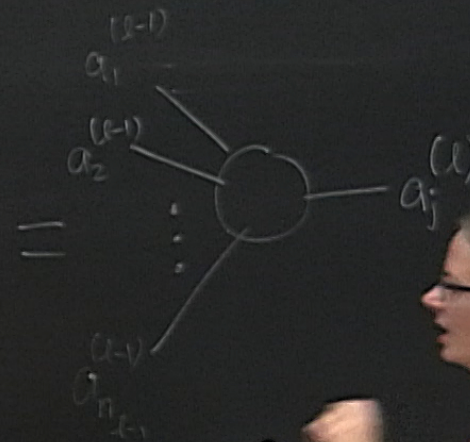
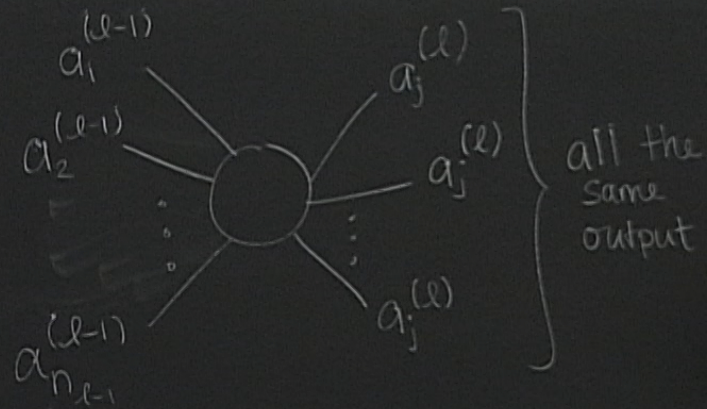
$n_l = \#$ of neurons in layer l

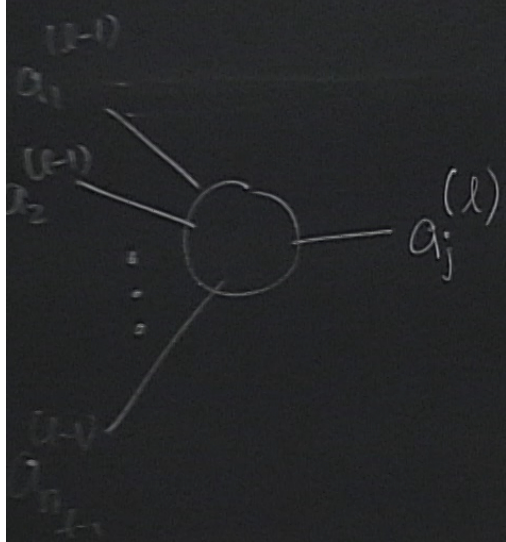
$L = \text{largest value of } l$
($L=3$ here)

$n_0 = d_x, n_L = d_y$

Define $a_i^{(0)} = x_i$ ($1 \leq i \leq d_x$)

Zoom in on the j^{th} neuron in layer $l > 0$:





$$a_j^{(l)} = g_l \left(\underbrace{\sum_{i=1}^{n_{l-1}} a_i^{(l-1)} W_{ij}^{(l)} + b_j^{(l)}}_{z_j^{(l)}} \right)$$

Each g_l is a non-linear "activation function" that we choose

$$a_j^{(l)} = g_l \left(\underbrace{\sum_{i=1}^{n_{l-1}} a_i^{(l-1)} W_{ij}^{(l)} + b_j^{(l)}}_{z_j^{(l)}} \right)$$

Each g_l is a non-linear "activation function" that we choose

$W_{ij}^{(l)}$: weights
 $b_j^{(l)}$: biases

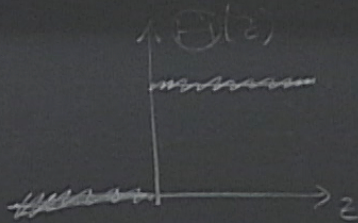
} the network will
adjust these
as it "learns"

$$\vec{a}^{(l)} = (a_1^{(l)}, a_2^{(l)}, \dots, a_{n_l}^{(l)})$$

Note: $\vec{f}(\vec{x}) = \vec{a}^{(L)}$

Activation Functions

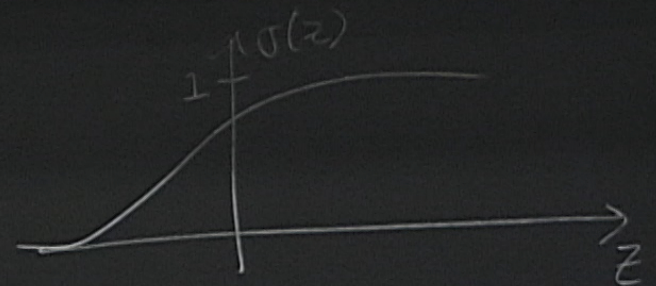
① Perceptron $\Theta(z)$



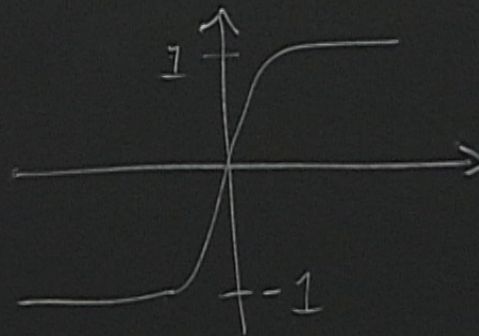
↳ A small change to one of the $W_{ij}^{(l)}$ or $b_j^{(l)}$ → small change in $z_j^{(l)}$
→ no change OR a large discontinuous change in $\Theta(z_j^{(l)})$

Learning is difficult!

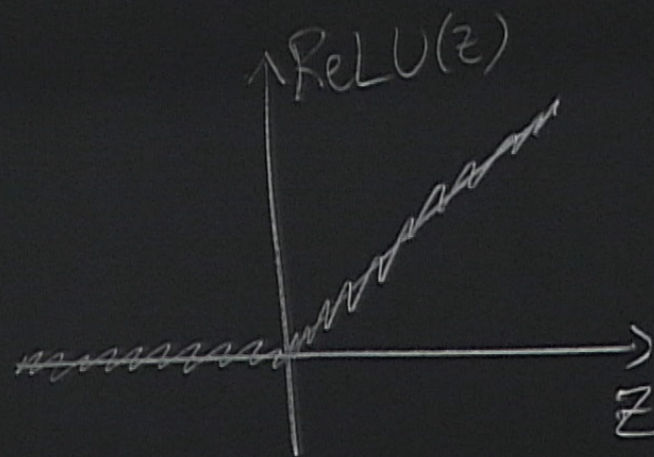
② Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$



③ $\tanh(z)$



④ Rectified linear unit (ReLU)



$$\text{ReLU}(z) = \max(0, z)$$

Cost functions

- We want to find weights and biases such that the network's output $\vec{a}^{(L)}(\vec{x})$ is a good approx. to $\vec{y}(\vec{x})$
- Use a "cost function" to measure how well the network is approximating the labels

input layer

hidden layer(s)

output layer

define $a_i = x_0$ $(1 \leq i \leq n)$

① Mean-squared error (MSE)

$$C_{MSE} = \frac{1}{2|D|} \sum_{\vec{x} \in D} \sum_{i=1}^{n_L} \left[a_i^{(L)}(\vec{x}) - y_i(\vec{x}) \right]^2$$

② Cross-entropy (CE)

$$C_{CE} = -\frac{1}{|D|} \sum_{\vec{x} \in D} \sum_{i=1}^{n_L} \left[y_i(\vec{x}) \log a_i^{(L)}(\vec{x}) + (1 - y_i(\vec{x})) \log (1 - a_i^{(L)}(\vec{x})) \right]$$

We would like to minimize C with respect to the weights & biases in each layer l :

$$\left. \begin{aligned} \frac{\partial C}{\partial W_{ij}^{(l)}} &= 0 \\ \frac{\partial C}{\partial b_j^{(l)}} &= 0 \end{aligned} \right\} \text{for all } i, j, l$$

Algorithms to optimize the weights and biases

C is a function of N parameters

$p_1, p_2, \dots, p_N$

weights & biases

We want to solve $\frac{\partial C}{\partial p_k} = 0 \quad (1 \leq k \leq N)$

Simple algorithm: gradient descent

$$\vec{p} \rightarrow \vec{p} - \eta \vec{\nabla}_{\vec{p}} C$$

η : a parameter we choose called the "learning rate"
"hyper parameter"

We need to calculate $\frac{\partial C}{\partial W_{ij}^{(e)}}$ and $\frac{\partial C}{\partial b_j^{(e)}}$

C is written in terms of output $\vec{a}^{(L)}$ of the last layer

Idea:

- start at last layer
- compute gradients at layer l from layer $l+1$

 } "backpropagation"