CEE 616: Probabilistic Machine Learning
M3 Deep Neural Networks: Neural Networks for Structured Data I

Jimi Oke

UMass Amherst
College of Engineering

Thu, Oct 16, 2025

Outline

Neural networks

Consider the linear model:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{b} \tag{1}$$

We can increase the flexibility of the model via a basis function expansion (feature extractor) $\phi(\boldsymbol{x})$:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{W}\phi(\boldsymbol{x}) + \boldsymbol{b} \tag{2}$$

If further parameterize $\phi(\boldsymbol{x})$ by $\boldsymbol{\theta}_2$ for better fitting, we have:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{W}\phi(\boldsymbol{x};\boldsymbol{\theta}_2) + \boldsymbol{b} \tag{3}$$

To even further increase complexity, we can recursively fit more feature extractors $f_\ell(\boldsymbol{x};\boldsymbol{\theta}_\ell)$:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = f_L(f_{L-1}(\cdots f_1(\boldsymbol{x};\boldsymbol{\theta}_1))\cdots)) \tag{4}$$

Each ℓ can be considered a layer in a **feedforward neural network** (FFNN) of L layers.

- Also known as a **multilayer perception** (MLP)
- When L is large, this is termed a **deep neural network** (DNN)

Biological neuron

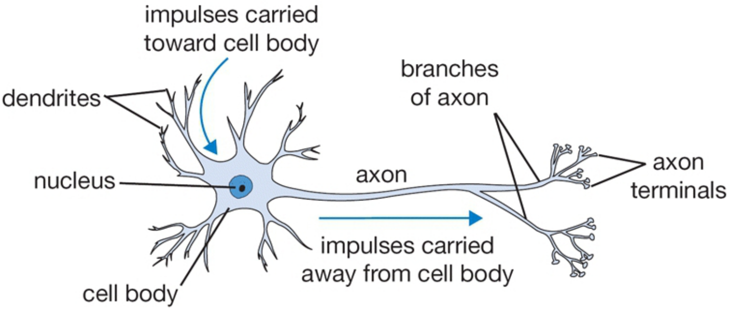


Figure: Biological neuron (Source: `https://cs231n.github.io/neural-networks-1/`)

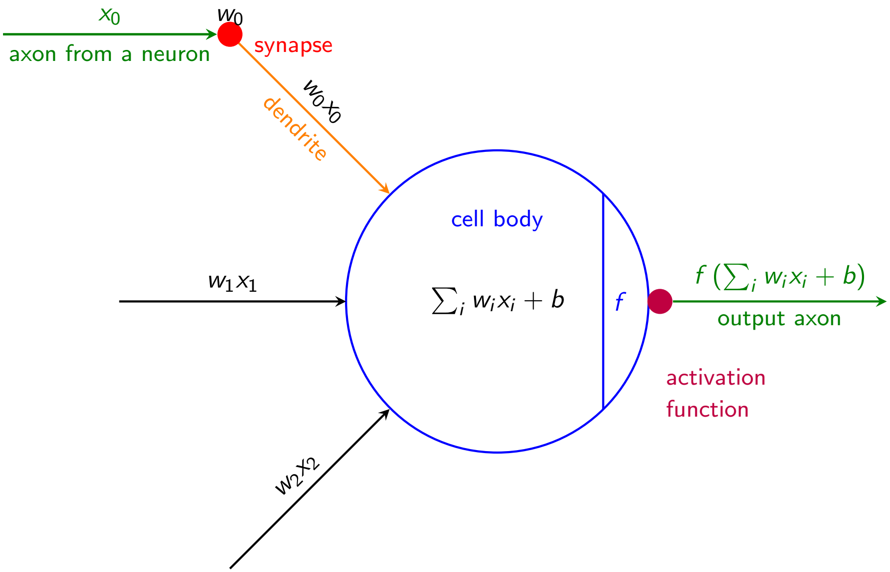
- ∼86 billion neurons are found in the human nervous system
- These neurons are connected by 10^{14} to 10^{15} synapses
- Each neuron receives input signals from its dendrites and outputs signals along a single axon
- The axon in turn connects to other neurons via synapses

Artificial neural networks

[Artificial] Neural networks (ANNs) are modeled as connected layers of neuron in an acyclic graph (no loops).

- ANNs are organized into layers of neurons (or "units")
- Fully-connected layers are common
- The basic ANN architecture with multiple hidden layers is called the **multilayer perceptron (MLP)**
  - An ANN with only one hidden layer is called the **single layer perceptron**
  - N-layer neural network (number of hidden layers + output layer)
- The output neurons have no activation function. Instead, they perform a final transformation of outputs from the penultimate layer

Computational neuron model

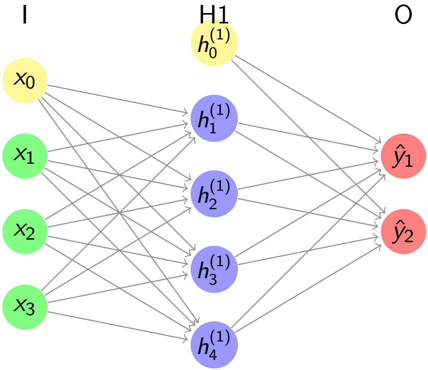


Computational neuron model (cont.)

- x_i: signals traveling along axons (inputs)
- w_i: measure of synaptic strength, which is learned;
  - $w_i > 0 \rightarrow$ excitory influence
  - $w_i < 0 \rightarrow$ inhibitory influence
- Dendrites carry signals $w_i x_i$ to the cell body, where they are summed.
- If the final sum $w_i x_i + b > t$ where t is a threshold[1], the neuron sends a spike along its axon (i.e. fires)
- Computationally, the firing rate of a neuron is represented by an **activation function** f
- The output of a neuron is also called the *activation*

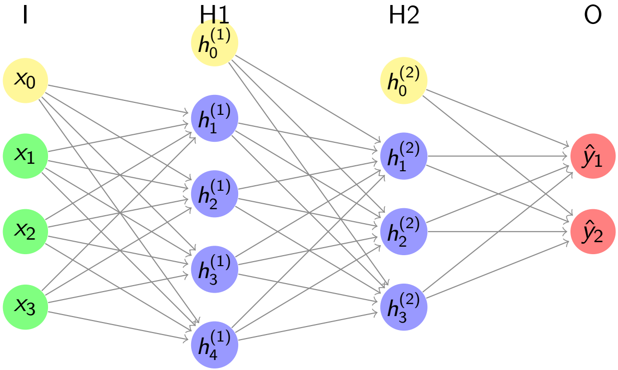
[1] intercept b is referred to as the "bias" in ML literature

Two-layer neural network (with bias neurons)



- **Layers**: 2 (input layer not counted); **Hidden layers:** 1
- **Neurons**: 7 (inputs not counted)
- **Learnable parameters:** $(4 \times 4) + (5 \times 2)$; total $= 26$

Three-layer neural network (with bias neurons)



- **Layers**: 3; **Hidden layers:** 2
- **Neurons**: 9
- **Learnable parameters:** $(4 \times 4) + (5 \times 3) + (4 \times 2) = 39$ weights; total $= 39$

Activation functions

In an ANN, the activation function f_ℓ modulates determines whether a certain neuron "fires" or passes information (hidden units $\boldsymbol{z}_\ell$ at layer ℓ) to the subsequent layer $\ell + 1$.

$$\boldsymbol{z}_\ell = f_\ell(\boldsymbol{z}_{\ell-1}) = \varphi_\ell(\boldsymbol{b}_\ell + \boldsymbol{W}_\ell \boldsymbol{z}_{\ell-1}) \tag{5}$$

- The input to the activation function $\boldsymbol{b}_\ell + \boldsymbol{W}_\ell \boldsymbol{z}_{\ell-1}$ is termed the **pre-activations**:

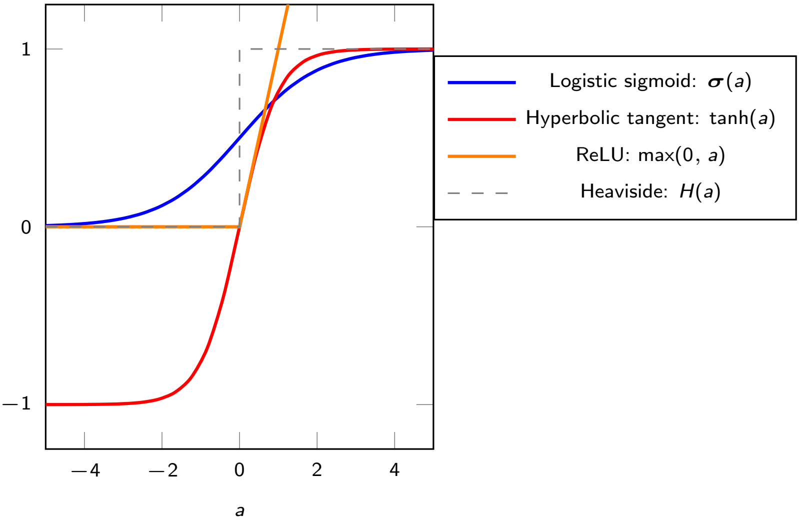
  $$\boldsymbol{a}_\ell = \boldsymbol{b}_\ell + \boldsymbol{W}_\ell \boldsymbol{z}_{\ell-1} \tag{6}$$

  Thus

  $$\boldsymbol{z}_\ell = \varphi_\ell(\boldsymbol{a}_\ell) \tag{7}$$

- In the historic MLP, the activation function was the non-differentiable Heaviside function (difficult to train)
- Later on, the sigmoid was introduced (smooth, trainable/differentiable)

Examples of activation functions

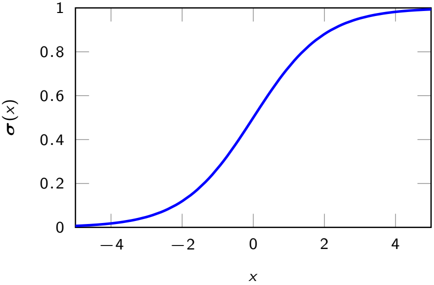


Logistic sigmoid function

- The form of the logistic sigmoid function is given by:

$$\boldsymbol{\sigma}(x) = \frac{1}{1 + e^{-x}} \tag{8}$$

- It transforms a real-valued input in the interval $[0, 1]$.



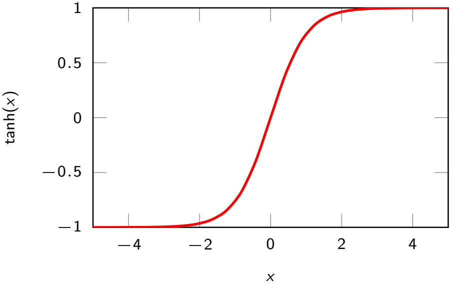
- Historically, it was used as it nicely represents the firing rate
- Recently, it has been superseded by the hyperbolic tangent due to its (a) gradient saturation and (b) non-zero-centeredness.

Hyperbolic tangent (tanh)

- The hyperbolic tangent function is given by:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1 \tag{9}$$

- It transforms a real-valued input in the interval $[-1, 1]$.



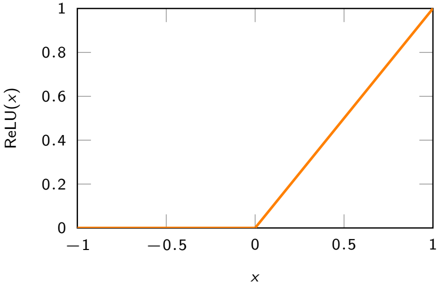
- Preferred to sigmoid activation function due to its zero-centeredness.

Rectified linear unit (ReLU)

- The ReLU is given by

$$\mathrm{ReLU}(x) = \max(0, x) \tag{10}$$

- Performs a simple thresholding of input at 0.



- Demonstrates faster convergence than $\sigma(x)$ and $\tanh(x)$
- Popular for deep convolutional networks (several hidden layers)
- Neurons can be fragile, however, requiring care in selection of learning rate

Neural network notation

The sigmoid **activation (output)** of a neuron is denoted:

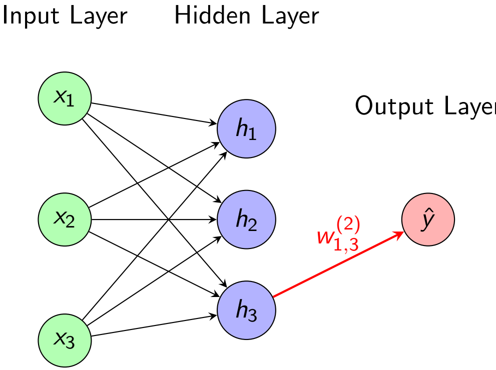
$$\varphi(w_0 z_1 + w_1 z_2 + \cdots + w_{m-1} z_{m-1} + b) = \varphi\left(\sum w_i z_i + b\right) = \text{new neuron} \quad (11)$$

Further, we denote each hidden unit as $z_{neuron}^{(layer)}$, e.g.

- $z_4^{(1)}$: fourth neuron in first layer (layers are counted from first hidden layer)

Weights are denoted as $w_{to,from}$, e.g.

- $w_{2,3}^2$: from the third neuron in the layer 1 to the second neuron in layer 2
- The superscript is not often used, as it is clear from the context which layer we are dealing with



Matrix operations in neural networks

Given the activation vector (D neurons) in the zeroth (input) layer:

$$\boldsymbol{x} \in \mathbb{R}^D = \boldsymbol{z}^{(0)} = \begin{bmatrix} z_1^0 \\ z_2^0 \\ \vdots \\ z_D^0 \end{bmatrix} \tag{12}$$

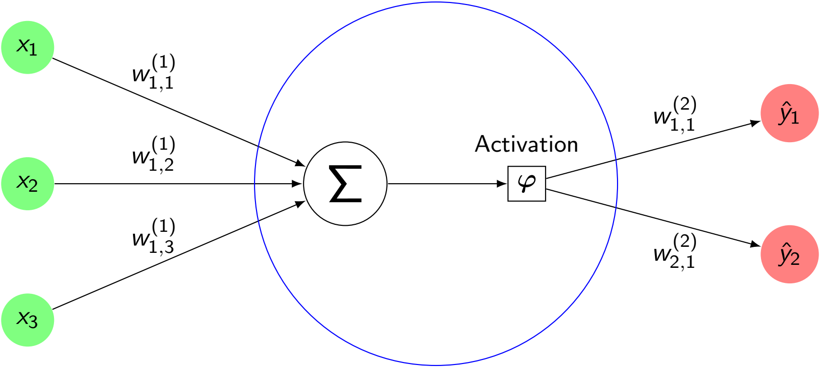
Then the activations in the next layer (M neurons) are given by:

$$\boldsymbol{z}^{(1)} = \varphi\left(\boldsymbol{W}\boldsymbol{z}^{(0)} + \boldsymbol{b}\right) = \varphi\left(\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,D} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M,1} & w_{M,2} & \cdots & w_{M,D} \end{bmatrix} \begin{bmatrix} z_1^0 \\ z_2^0 \\ \vdots \\ z_D^0 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_M \end{bmatrix}\right) \tag{13}$$

Example: If Layer 1 had only two neurons, then the weight matrix $\boldsymbol{W}$ would have only 2 rows.

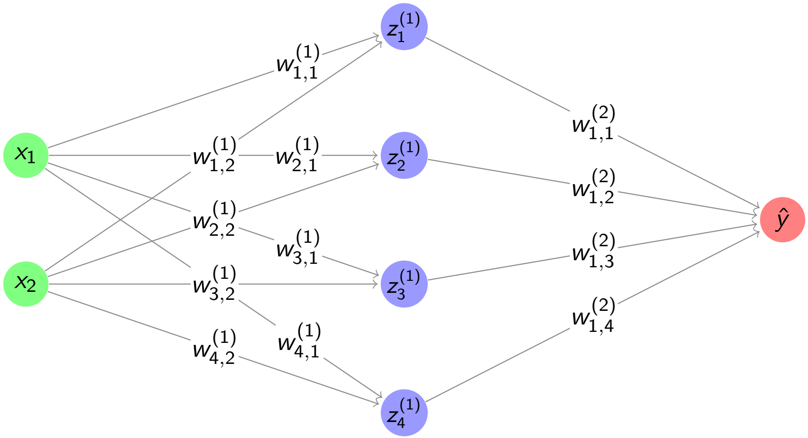
Example: MLP with two outputs

This simple MLP has 2 layers (1 hidden, one outer), and



Example: 2-layer regression MLP

Two-layer MLP for regression



Example: 2-layer regression MLP: scalar form equations

Given an observation x_{nd} with $d = 1, \ldots, D$ features, these equations describe the output from a 2-layer network:

$$
\begin{aligned}
z_m^{(1)} &= \varphi\left(\sum_{d=1}^{D} w_{1,d}^{(1)} x_{nd} + b_d^{(1)}\right) \\
y_i(x_i) &= \sum_{m=1}^{M} w_{1,m}^{(2)} z_m^{(1)} + b^{(2)}
\end{aligned}
$$

- D is number of input neurons
- M is number of hidden neurons
- Total number of learnable parameters: $M(D+1)$ weights and $(D+1)$ biases
- Linear/identity activation is used in output

Neural network loss function

Given K output neurons and N observations (where f_k is the output), we can compute the loss (cost) functions C as follows.

For regression:

$$\mathcal{L} = \sum_{k=1}^{K} \sum_{n=1}^{N} (y_{nk} - f_k(x_n))^2 \tag{14}$$

Thus, we can write, where $K = 1$ (univariate output):

$$\mathcal{L} = \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 \tag{15}$$

For classification, we use the cross-entropy (deviance) given K classes:

$$\mathcal{L} = -\sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \log f_k(x_n)) \tag{16}$$

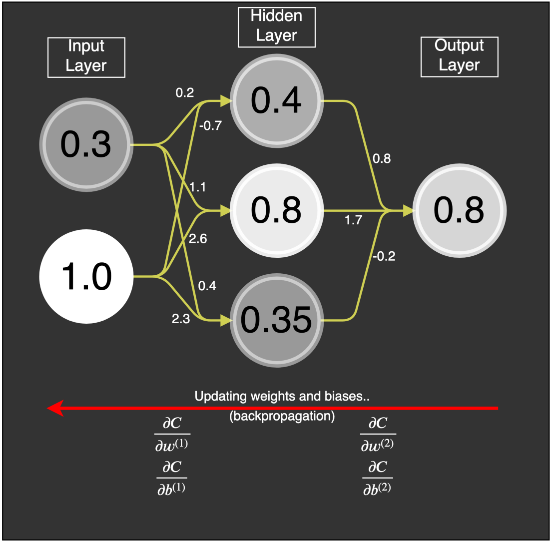
Training a neural network

- A neural network is trained or fitted by *learning* the optimal values of the weights (and biases).
- This learning is done via optimization (e.g. gradient descent)
- Gradient descent update:

$$w^{\text{new}} = w^{\text{old}} - \eta \frac{\partial \mathcal{L}}{\partial w^{\text{old}}} \tag{17}$$

  where:
  - η is the learning rate
  - $\mathcal{L}$ is the cost function (e.g. residual sum of squares)
  - w the weight
- In neural networks, the gradients are computed via **backpropagation**

Backpropagation overview



Training procedure

- Fix initial weights and perform a **forward sweep/pass** through the network computing the activations a (outputs) of each layer l as:

$$\begin{aligned} \boldsymbol{z}_\ell &= \varphi(\boldsymbol{W}_\ell \boldsymbol{z}_{\ell-1} + \boldsymbol{b}_\ell) && (18)\\ \boldsymbol{a}_\ell &= \boldsymbol{W}_\ell \boldsymbol{z}_{\ell-1} + \boldsymbol{b}_\ell && (19)\\ \boldsymbol{z}_\ell &= \varphi(\boldsymbol{a}_\ell) && (20) \end{aligned}$$

- At the output layer, we compute the cost (loss) function $\mathcal{L}$ (what we want to minimize)
- Then, we **backpropagate** the errors through each layer in order to compute the gradients for the weight updates:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_L} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_L} \frac{\partial \boldsymbol{z}_L}{\partial \boldsymbol{a}_L} \frac{\partial \boldsymbol{a}_L}{\partial \boldsymbol{W}_L} \tag{21}$$

  where L is the last layer.
- Repeat the forward and backward passes until cost is sufficiently minimized

Equation summary: outer layer (regression case

At the outer layer L (without indexing by neuron):

$$a_L = \boldsymbol{w}_L^\top \boldsymbol{z}_{L-1} + b_L \tag{22}$$

$$o = a_L \quad \text{(linear activation or \textit{no} activation)} \tag{23}$$

$$\mathcal{L} = (o - y)^2 \tag{24}$$

The gradient of the cost function with respect to $\boldsymbol{w}_L$ is:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_L} = \frac{\partial \mathcal{L}}{\partial o} \frac{\partial o}{\partial a_L} \frac{\partial a_L}{\partial \boldsymbol{w}_L} = 2\left(a_L - y\right) \boldsymbol{z}_{L-1} \tag{25}$$

Thus, we see that this gradient depends on the activation from the previous layer a_{L-1}. Also wrt to the bias:

$$\frac{\partial \mathcal{L}}{\partial b_L} = \frac{\partial \mathcal{L}}{\partial o} \frac{\partial o}{\partial a_L} \frac{\partial a_L}{\partial b_L} = 2\left(a_L - y\right)(1) \tag{26}$$

Updating weights

We can then update the weights for the last layer for the next iteration $t+1$:

$$w_{L,t+1} = w_{L,t} - \rho \frac{\partial \mathcal{L}}{\partial w_L} \tag{27}$$

$$b_{L,t+1} = b_{L,t} - \rho \frac{\partial \mathcal{L}}{\partial b_L} \tag{28}$$

where ρ is the learning rate. To update the weights for layer $L-1$, we need to find the gradients $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{L-1}}$, where $\boldsymbol{\theta} = (\boldsymbol{W}, \boldsymbol{b})$.

Using the chain rule again, we write:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{L-1}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_{L-1}} \frac{\partial \boldsymbol{z}_{L-1}}{\partial \boldsymbol{a}_{L-1}} \frac{\partial \boldsymbol{a}_{L-1}}{\partial \boldsymbol{\theta}_{L-1}} \tag{29}$$

Backward pass

But we recall that $\mathcal{L}$ is not *explicitly* dependent on $\boldsymbol{z}_{L-1}$ as $\mathcal{L} = (o - y)^2$. However, it is *implicitly* dependent, since

$$\mathcal{L} \propto o, \tag{30}$$

$$o \propto a_L \tag{31}$$

and

$$a_L \propto z_{L-1} \tag{32}$$

So, we use the chain rule to expand $\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_{L-1}}$ as follows:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_{L-1}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_L} \frac{\partial \boldsymbol{z}_L}{\partial \boldsymbol{a}_L} \frac{\partial \boldsymbol{a}_L}{\partial \boldsymbol{z}_{L-1}} \tag{33}$$

Backward pass (cont.

We can then expand the cost function gradient wrt to weights for layer $L-1$ as:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{L-1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}_{L-1}} \frac{\partial \mathbf{z}_{L-1}}{\partial \mathbf{a}_{L-1}} \frac{\partial \mathbf{a}_{L-1}}{\partial \boldsymbol{\theta}_{L-1}} \tag{34}$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{z}_{L}} \frac{\partial \mathbf{z}_{L}}{\partial \mathbf{a}_{L}} \frac{\partial \mathbf{a}_{L}}{\partial \mathbf{z}_{L-1}} \frac{\partial \mathbf{z}_{L-1}}{\partial \mathbf{a}_{L-1}} \frac{\partial \mathbf{a}_{L-1}}{\partial \boldsymbol{\theta}_{L-1}} \tag{35}$$

Once these gradients are computed, we update the weights for the $(t+1)$th iteration using:

$$\boldsymbol{\theta}_{L-1,t+1} = \boldsymbol{\theta}_{L-1,t} - \rho \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{L-1}} \tag{36}$$

$$\tag{37}$$

Summary: forward pass

1. $(t = 0)$: Initialize weights and biases: $\boldsymbol{\theta}$
2. Perform forward pass to compute activations:

$$\boldsymbol{a}_{\ell,0} = \boldsymbol{W}_{\ell,0} \times \boldsymbol{z}_{\ell-1,0} + \boldsymbol{b}_{\ell,0} \tag{38}$$

$$\boldsymbol{z}_{\ell} = \varphi(\boldsymbol{a}_{\ell,0}) \tag{39}$$

At output layer:

$$a_{L,0} = \boldsymbol{w}_{L,0}^{\top} \boldsymbol{Z}_{L-1,0} + b_{L,0} \tag{40}$$

$$o = a_{L,0} \tag{41}$$

$$\mathcal{L} = (o - y)^2 \tag{42}$$

Summary: backward pass—outer layer

3. Backward pass, outer layer L:
   a. Compute gradients:

$$\frac{\partial \mathcal{L}}{\partial b_L} = \frac{\partial \mathcal{L}}{\partial o} \frac{\partial o}{\partial a_L} \frac{\partial a_L}{\partial b_L} \tag{43}$$

   b. Update weights:

$$\boldsymbol{\theta}_{L,1} = \boldsymbol{\theta}_{L,0} - \rho \frac{\partial \mathcal{L}_0}{\partial \boldsymbol{\theta}_L} \tag{44}$$

Summary: backward pass—last hidden layer

Recall:

$$\boldsymbol{z}_{L-1} = \varphi(\boldsymbol{a}_{L-1}) \tag{45}$$

$$\boldsymbol{a}_{L-1} = \boldsymbol{W}_{L-1}\boldsymbol{z}_{L-2} + \boldsymbol{b}_{L-1} \tag{46}$$

4. Backward pass, layer $L-1$:
   a. Compute gradients:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{L-1}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_L}\frac{\partial \boldsymbol{z}_L}{\partial \boldsymbol{a}_L}\frac{\partial \boldsymbol{a}_L}{\partial \boldsymbol{z}_{L-1}}\frac{\partial \boldsymbol{z}_{L-1}}{\partial \boldsymbol{a}_{L-1}}\frac{\partial \boldsymbol{a}_{L-1}}{\partial \boldsymbol{\theta}_{L-1}} \tag{47}$$

(48)

   b. Update weights:

$$\boldsymbol{\theta}_{L-1,1} = \boldsymbol{\theta}_{L-1,0} - \rho\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{L-1}} \tag{49}$$

Summary: backward pass—second-to-last hidden layer

Recall

$$\boldsymbol{a}_{L-1} = \boldsymbol{W}_{L-1}\boldsymbol{z}_{L-2} + \boldsymbol{b}_{L-1} \tag{50}$$

$$\boldsymbol{z}_{L-2} = \varphi(\boldsymbol{a}_{L-2}) \tag{51}$$

$$\boldsymbol{a}_{L-2} = \boldsymbol{W}_{L-2}\boldsymbol{z}_{L-3} + \boldsymbol{b}_{L-2} \tag{52}$$

5. Backward pass, layer $L-2$:
   a. Compute gradients:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{L-2}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_L}\frac{\partial \boldsymbol{z}_L}{\partial \boldsymbol{a}_L}\frac{\partial \boldsymbol{a}_L}{\partial \boldsymbol{z}_{L-1}}\frac{\partial \boldsymbol{z}_{L-1}}{\partial \boldsymbol{a}_{L-1}}\frac{\partial \boldsymbol{a}_{L-1}}{\partial \boldsymbol{z}_{L-2}}\frac{\partial \boldsymbol{z}_{L-2}}{\partial \boldsymbol{a}_{L-2}}\frac{\partial \boldsymbol{a}_{L-2}}{\partial \boldsymbol{\theta}_{L-2}} \tag{53}$$

   b. Update weights:

$$\boldsymbol{\theta}_{L-2,1} = \boldsymbol{\theta}_{L-2,0} - \rho\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_{L-2}} \tag{54}$$

$$\tag{55}$$

Summary: backward pass—first hidden layer

Recall

$$\boldsymbol{a}_\ell = \boldsymbol{W}_\ell \boldsymbol{z}_{\ell-1} + \boldsymbol{b}_\ell \tag{56}$$

$$\boldsymbol{z}_\ell = \varphi(\boldsymbol{a}_\ell) \tag{57}$$

$$\tag{58}$$

3. Backward pass, layer (1):
   5. Compute gradients:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_1} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_L}\frac{\partial \boldsymbol{z}_L}{\partial \boldsymbol{a}_L}\frac{\partial \boldsymbol{a}_L}{\partial \boldsymbol{z}_{L-1}}\frac{\partial \boldsymbol{z}_{L-1}}{\partial \boldsymbol{a}_{L-1}}\cdots\frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_1}\frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{a}_1}\frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{\theta}_1} \tag{59}$$

   6. Update weights:

$$\boldsymbol{\theta}_{1,1} = \boldsymbol{\theta}_{1,0} - \rho\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_1} \tag{60}$$

$$\tag{61}$$

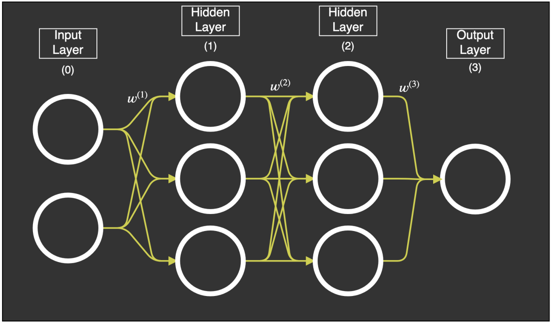
Summary of backpropagation

1. Fix initial weights $\boldsymbol{\theta}_{\ell,0} = (\boldsymbol{W}_{\ell,0}, \, \boldsymbol{b}_{\ell,0})$ and perform a forward sweep/pass through the network computing the activations a (outputs) of each layer ℓ as:

$$\boldsymbol{a}_\ell = \sigma(\boldsymbol{W}_\ell a_{l-1} + b_\ell) \tag{62}$$

2. At the output layer, we compute the loss/cost function $\mathcal{L}$ (what we want to minimize)
3. Then, we *backpropagate* the errors through each layer in order to compute the gradients $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_\ell}$ and weight updates $\boldsymbol{\theta}_{\ell,t+1}$
4. Repeat the forward and backward passes until cost is sufficiently minimized

Example: backpropagation for 3-layer network



$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_3} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_3}\frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{a}_3}\frac{\partial \boldsymbol{a}_3}{\partial \boldsymbol{\theta}_3} \quad (\boldsymbol{z}_3 \equiv o) \tag{63}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_2} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_3}\frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{a}_3}\frac{\partial \boldsymbol{a}_3}{\partial \boldsymbol{z}_2}\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{a}_2}\frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{\theta}_2} \tag{64}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_1} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_3}\frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{a}_3}\frac{\partial \boldsymbol{a}_3}{\partial \boldsymbol{z}_2}\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{a}_2}\frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_1}\frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{a}_1}\frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{\theta}_1} \tag{65}$$

Regression MLP architecture

Typical hyperparameter values are:

Hyperparameter	Value
# input neurons	1 per input feature
# hidden layers	Usually 1 – 5
# neurons per hidden layer	Usually 10 – 100
# output neurons	1 per prediction dimension
hidden layer activation	ReLU
output activation	None (if unbounded)
loss function	MSE or MAE/Huber

Classification MLP architecture

- For classifcation, input and hidden layers are chosen in similar fashion to the regression case
- However, the number of output neurons is given by the name of classes/labels
- The output layer activation is typically the softmax function:

$$\text{softmax}(z_k) = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}} \tag{66}$$

  where z_k is the unnormalized log probability of each class k
- The loss function is taken as the cross entropy

Other types of neural networks

The standard ANN architecture (MLP) we have studied is also called the feed-forward network.

Other architectures have been shown to give better performance for various applications:

- Recurrent neural networks (RNNs): time-series forecasting
- Convolutional neural networks (CNNs): image classification
- Long short-term memory networks (LSTMs): time-series, pattern identification, etc.

Reading

We will discuss the CNN on Wednesday, along with examples in Python.

- **PMLI**: 13.1-3
- **PML**: 8.3, 9.4
- **ESL**: 11
- **DL**: 6-8, 11, 12
- Experiment in this **playground**