Introduction to Neural Networks

Section 10.1

Cathy Poliak, Ph.D.
cpoliak@central.uh.edu

Department of Mathematics
University of Houston

Neural Networks: Introduction

We live in the era of technology with computers taking over vast majority of the operation.

Computers have by far surpassed humans in the domains of

- numerical computations, and
- symbol manipulation

Remember such things as

- travel agencies?
- newspapers (made of.. paper)?
- those things for cashiers to count?

Neural Networks: Foreword

Nonetheless, there are still domains where humans reign supreme:

- pattern recognition,
- noise reduction,
- certain optimization tasks.

Example. A toddler can recognize his/her mom in a huge crowd, but a computer with a centralized architecture wouldn't be able to do the same.

Neural Networks: Foreword

Nonetheless, there are still domains where humans reign supreme:

- pattern recognition,
- noise reduction,
- certain optimization tasks.

Example. A toddler can recognize his/her mom in a huge crowd, but a computer with a centralized architecture wouldn't be able to do the same.

Example. Just check this priceless video of robots playing soccer:

`https://www.youtube.com/watch?v=qpwI2PoAckY`.

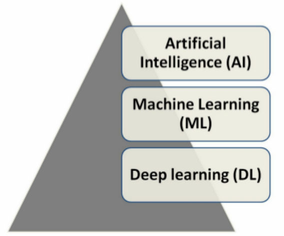
Improving on computers' capability in dealing with those tasks is the field of Artificial Intelligence (AI).

Artificial Intelligence (AI) and Machine Learning (ML)

Artificial Intelligence (AI) systems attempt creating machines that imitate parts of human intelligence mechanisms.

Machine learning (ML) is a branch of AI which helps computers to program themselves based on the input data.

The hierarchy of those critical concepts looks like this:



Artificial Neural Networks (ANN)

A decisive step in the improvement of AI machines came from the use of so-called Artificial Neural Networks (ANNs).

ANNs are an example of machine learning algorithms that try to

- emulate the neuronal structure of human brain, and
- reproduce the human thinking process for certain tasks.

Those tasks include

- object recognition,
- image classification,
- fraud detection,
- hand writing identification,
- video analysis.

Inspiration for Neural Networks

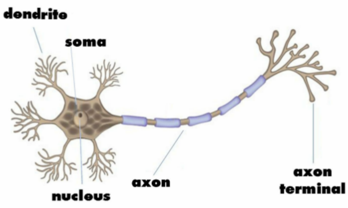
- The **human brain** is the central processing unit for all the functions performed by us as humans.
- It constitutes a complex network of neurons, most of which process and transmit information they obtain either from:
  - sensors (e.g. eyes or ears),
  - or other neurons.
- Weighing only 1.5 kilos, it has around 86 billion neurons.

Biological Structure of a Neuron

Neurons are the nodes of the brain network that receive, process and transmit signals.

The major components of each neuron are:

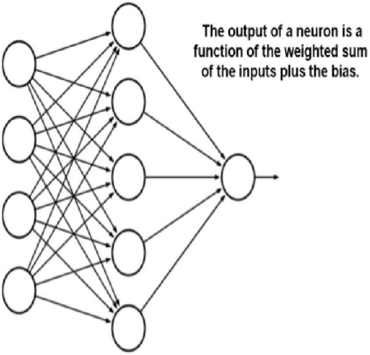
- **Dendrites**: Entry points in each neuron. They take input from other neurons/sensors of that network in form of electrical impulses.
- **Cell Body**: Calculates a function of dendrite inputs (signals).
- **Axons**: Transmit calculated outputs as signals to next neuron.



How do neural netWORKs WORK?

Similar to the biological neuron structure, ANNs define the neuron as a:

"Central processing unit that performs a mathematical operation to generate output from a set of inputs."



Essentially, ANN is a set of mathematical function approximations.

Neural Network Terminology

We would now be introducing new terminology associated with ANNs:

- Input layer
- Hidden layer
- Output layer
- Weights
- Bias
- Activation functions

Single Layer Neural Network

- A neural network takes an input vector of p variables $X = (X_1, X_2, \dots, X_p)$
- Builds a nonlinear function $f(X)$ to predict the response Y.
- The neural network model has the form

$$
\begin{aligned}
f(x) &= \beta_0 + \sum_{k=1}^{K} \beta_k H_k(X) \\
&= \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)
\end{aligned}
$$

Details

A single layer neural network is built in two steps.

1. K **activations** A_k, $K = 1, \ldots K$ in the hidden layer are computed as functions of the input features

$$A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj}X_j)$$

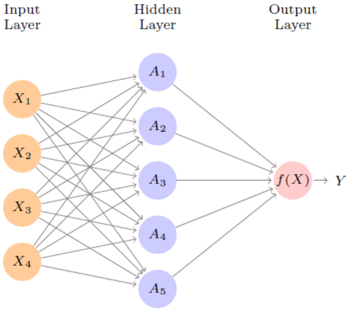
   where $G(Z)$ is a nonlinear activation function.

2. The K activations from the hidden layer then feed into the output layer, resulting in

$$f(x) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k$$

Layered Structure

Most neural network processing frameworks look as follows:

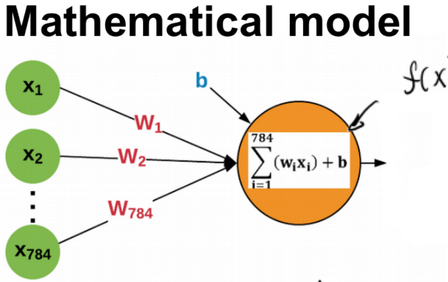


It consists of

- Input layer - the data inputs,
- Hidden layer(s) - processing "unit",
- Output layer - the neural network output.

Mathematical Model of a (Linear) Neuron

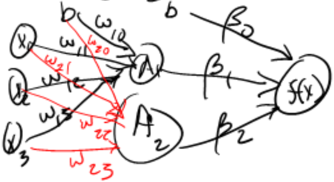
For a linear neuron, mathematical model would look like:



$f(x) = b + w_1 x_1 + w_2 x_2 + \cdots + w_{784} x_{784}$

X_1, X_2, X_3

Two activations A_1, A_2



$f(x) = \beta_0 + \beta_1 [g(w_{10} + w_{11} x_1 + w_{12} x_2 + w_{13} x_3$

$+ \beta_2 [g(w_{20} + w_{21} x_1 + w_{22} x_2 + w_{23} x_3$

Neuron Model: Weights and Bias

Let y - neuron output, then

$$y = b + \sum_{i=1}^{b} w_i x_i$$

Remotely reminds you of something? Recall multiple linear regression:

Neuron Model: Weights and Bias

Let y - neuron output, then

$$y = b + \sum_{i=1}^{b} w_i x_i$$

Remotely reminds you of something? Recall multiple linear regression:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

Only now the parameters we need to learn are called

- **weights** (instead of coefficients), and
- **bias** (instead of intercept)

and denoted as

- w_i (instead of β_i), $i = 1, \ldots, p$,
- b (instead of β_0)

Neuron Model: Weights and Bias

Let y - neuron output, then

$$y = b + \sum_{i=1}^{b} w_i x_i$$

Remotely reminds you of something? Recall multiple linear regression:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

Only now the parameters we need to learn are called

- **weights** (instead of coefficients), and
- **bias** (instead of intercept)

and denoted as

- w_i (instead of β_i), $i = 1, \ldots, p$,
- b (instead of β_0)

Our task is to estimate weights w_i and bias b, where

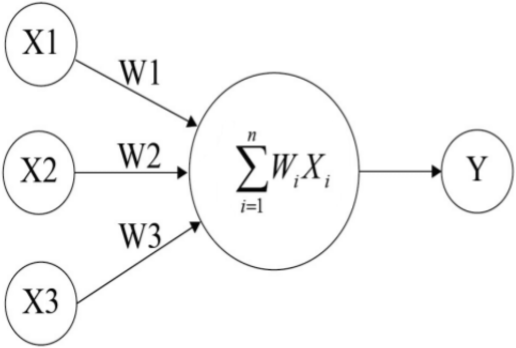
- weights determine how strongly one neuron affects the other,
- bias off-sets some of the effects.

Weights and Biases: Example.

If a neuron has three inputs x_1, x_2, x_3, then

- the weights applied to those inputs are denoted w_1, w_2, w_3,
- the output is

$$y = f(x_1, x_2, x_3) = \sum_{i=1}^{3} w_i x_i$$



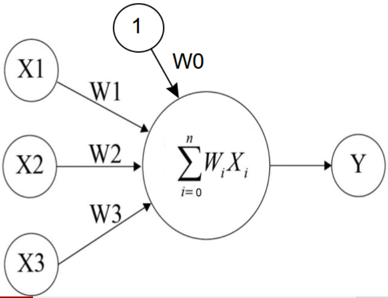
Weights and Biases: Example.

In most scenarios we need bias term as well, which is added by introducing

- an "artificial input" $x_0 \equiv 1$, and
- its corresponding weight w_0

For our previous three input example, the output now is

$$y = f(1, x_1, x_2, x_3) = \sum_{i=0}^{3} w_i x_i$$

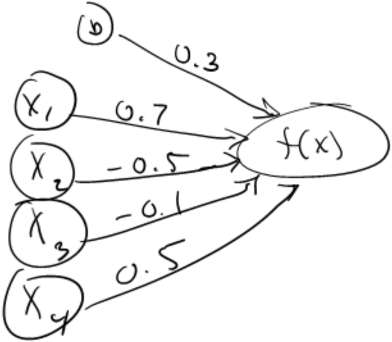


Example

Assume we have a linear neuron with

- inputs x_1, x_2, x_3, x_4
- weights $w_1 = 0.7, w_2 = -0.5, w_3 = -0.1, w_4 = 0.5$
- for bias node, $b = 0.3$.

Draw the corresponding neuron structure.



$$f(x) = 0.3 + 0.7x_1 - 0.5x_2 - 0.1x_3 + 0.5x_4$$

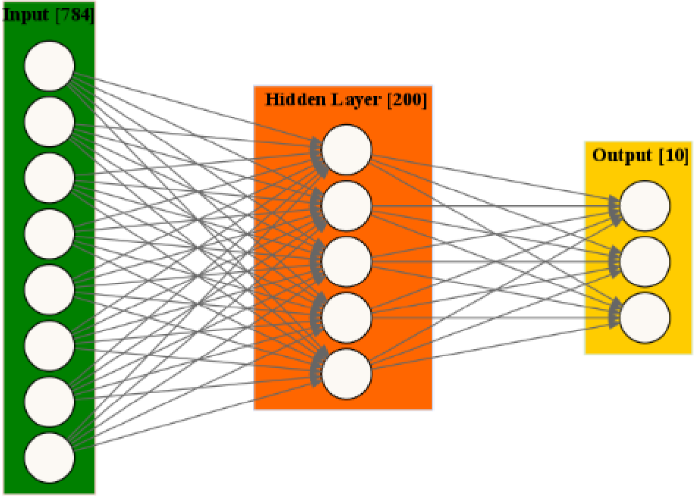
Estimate the Output

Calculate the output of the previous linear neuron for $x_1 = 3$, $x_2 = 7$, $x_3 = -3$, and $x_4 = 10$.

$$\hat{y} = \tilde{f}(x) = 0.3 + 0.7(3) - 0.5(7) - 0.1(-3) + 0.5(10)$$
$$= 0.3 + 2.1 - 3.5 + 0.3 + 5$$
$$= 4.2$$

Artificial Neural Networks

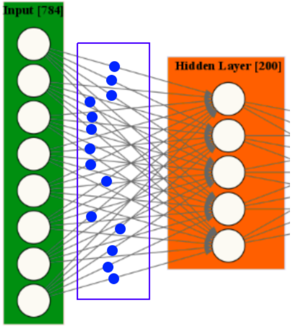
Easy so far? Well, typically we don't deal with just a single neuron, but rather with a network of neurons:

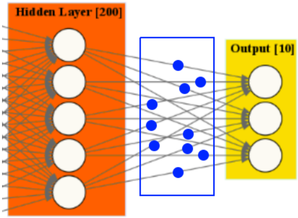


Artificial Neural Networks: Layered Structure

Each neuron in the hidden layer brings about:

- weight parameters, that determine either
  - how strongly it is affected by other neurons (e.g. inputs),
  - or how strongly it affects other neurons (e.g. outputs).





- and a bias term to offset some of the effects.

Training ANN. Universal Function Approximator.

Training is the act of

- presenting the ANN with some sample data, and
- modifying the weights to better approximate the desired function.

Training ANN. Universal Function Approximator.

Training is the act of

- presenting the ANN with some sample data, and
- modifying the weights to better approximate the desired function.

In general, ANN is a universal function approximator:

- given input data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^p$, and
- corresponding labels $y_1, \ldots, y_n \in \mathbb{R}$ of those inputs,
- it approximates function $f(.)$ such that

$$f(\mathbf{x}_i) = y_i, \; i = 1, \ldots, n$$

Training Methods

There are two main types of training:

- Supervised learning:
  1. ANN is supplied with both inputs and desired outputs
  2. Weights are modified to reduce the difference between the predicted and desired outputs
- Unsupervised learning:
  1. ANN is only supplied with inputs
  2. Weights are adjusted for similar inputs to generate similar outputs
  3. ANN identifies the patterns and differences in the inputs

Training ANN: Forward Propagation

To obtain a neural network output, we :

$$\text{input layer} \rightarrow \text{hidden layer(s)} \rightarrow \text{output layer}$$

which is called forward propagation.

At each neuron of a hidden layer,

1. $\sum_i weight_i \times input_i + bias$ is applied, where *input* comes from previous layer (could be an input or previous hidden layer).

Training ANN: Forward Propagation

To obtain a neural network output, we :

input layer $\rightarrow$ hidden layer(s) $\rightarrow$ output layer

which is called forward propagation.

At each neuron of a hidden layer,

1. $\sum_i weight_i \times input_i + bias$ is applied, where *input* comes from previous layer (could be an input or previous hidden layer).
2. Activation function f is applied to $\sum_i weight_i \times input_i + bias$ (more on activation function later, but for linear neuron it's identity)

Training ANN: Forward Propagation

To obtain a neural network output, we :

input layer $\rightarrow$ hidden layer(s) $\rightarrow$ output layer

which is called forward propagation.

At each neuron of a hidden layer,

1. $\sum_i weight_i \times input_i + bias$ is applied, where *input* comes from previous layer (could be an input or previous hidden layer).
2. Activation function f is applied to $\sum_i weight_i \times input_i + bias$ (more on activation function later, but for linear neuron it's identity)
3. Value $f(\sum_i weight_i \times input_i + bias)$ is propagated to the next layer (could be the next hidden layer or the output layer)

Backpropagation.

Once the output is arrived at after completion of forward propagation:

- we compute the error $\hat{y} - y$ (predicted output − true output),
- use this error to correct the weights and biases used in forward propagation.

How exactly is the error used?

1. Derivative of error is taken.
2. Amount of weight that has to be changed is determined by gradient descent.
3. The gradient suggests how steeply the error will be reduced or increased for a change in the weight.
4. The backpropagation keeps changing the weights until there is greatest reduction in errors by an amount known as the learning rate.

This process of

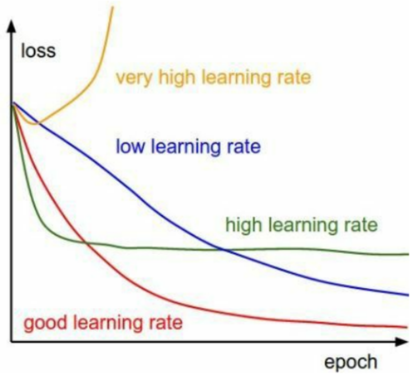
compute error $\rightarrow$ take derivative $\rightarrow$ change weights/bias via gradient descent

is known as backpropagation.

Backpropagation: Learning Rate.

The backpropagation keeps changing the weights until there is greatest reduction in errors by an amount known as the learning rate.

Learning rate is a scalar parameter, analogous to step size in numerical integration, used to set the rate of adjustments to reduce the errors faster.



Iterative Approach: Example

An example to illustrate the iterative method of online gradient descent:

- Each day you get lunch at the cafeteria.
  - Your diet consists of fish, chips, and ketchup.
  - You get several portions (inputs x) of each $\Rightarrow x_{fish}, x_{chips}, x_{ketchup}$.

Iterative Approach: Example

An example to illustrate the iterative method of online gradient descent:

- Each day you get lunch at the cafeteria.
  - Your diet consists of fish, chips, and ketchup.
  - You get several portions (inputs x) of each $\Rightarrow x_{fish}, x_{chips}, x_{ketchup}$.
- Cashier only tells you the total price of the meal (response y).
  - After several days, you should be able to figure out the price (weight w) of each portion $\Rightarrow w_{fish}, w_{chips}, w_{ketchup}$

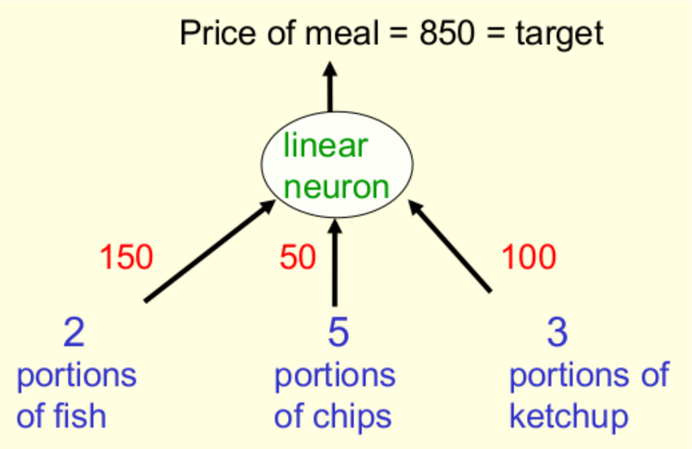
Iterative Approach: Example

An example to illustrate the iterative method of online gradient descent:

- Each day you get lunch at the cafeteria.
  - Your diet consists of fish, chips, and ketchup.
  - You get several portions (inputs x) of each $\Rightarrow x_{fish}, x_{chips}, x_{ketchup}$.
- Cashier only tells you the total price of the meal (response y).
  - After several days, you should be able to figure out the price (weight w) of each portion $\Rightarrow w_{fish}, w_{chips}, w_{ketchup}$
- The iterative approach:
  1. Start with random guesses for the prices ($\Leftrightarrow$ weights), and then
  2. Adjust them to get a better fit to the observed prices of whole meals (y_i's).

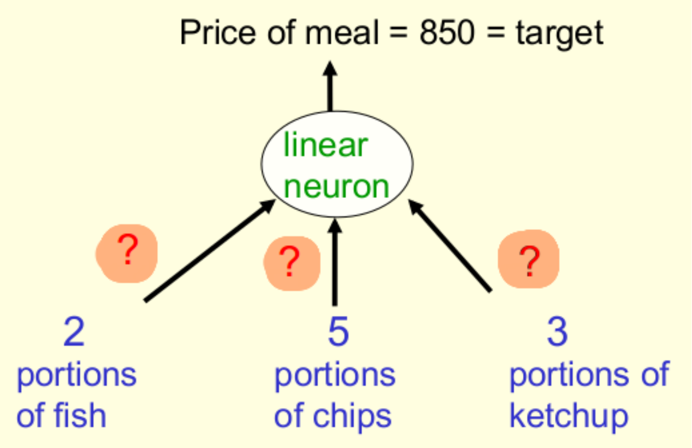
Iterative Approach: Example

The true weights used by the cashier



Iterative Approach: Example

In reality? True weights are unknown.



Iterative Approach: Example

- The portion prices are weights of our linear neuron.

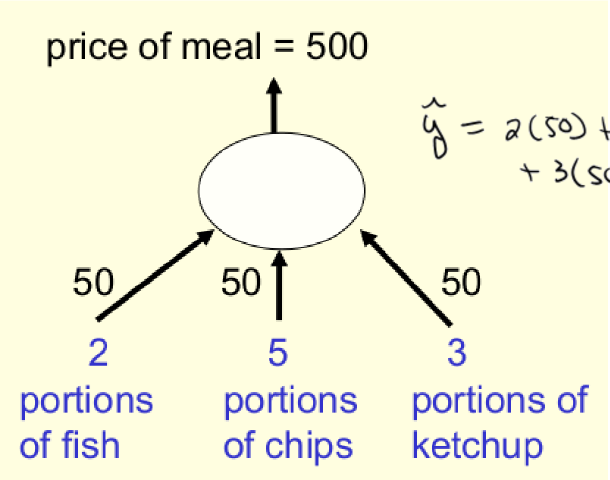
$$\mathbf{w} = (w_{fish}, w_{chips}, w_{ketchup})$$

- Each meal price is our response y, and it is calculated via linear combination of portion prices (weights w) multiplied by # of portions (inputs x):

$$price = x_{fish} w_{fish} + x_{chips} w_{chips} + x_{ketchup} w_{ketchup}$$

- We start with random guesses for the weights, and then adjust the guesses slightly to give a better fit to the prices given by the cashier.

Initializing the weights with random values, e.g $50, 50, 50$**.**



$$\hat{y} = 2(50) + 5(50) + 3(50) = 500$$

Iterative Approach: Example

- True value $y = 850$; predicted value $\hat{y} = 500 \implies$

$$\text{Residual error} = 850 - 500 = 350$$

Iterative Approach: Example

- True value $y = 850$; predicted value $\hat{y} = 500 \implies$

$$\text{Residual error} = 850 - 500 = 350$$

- Below we provide a **delta-rule** update for gradient descent algorithm given a learning rate α:

$$\Delta w_i = \alpha \times \frac{\partial}{\partial w_i}\left[\frac{1}{2}(\hat{y} - y)^2\right], \; i = 1, 2, 3$$

$$\frac{\partial}{\partial w_1}\left[\frac{1}{2}\left(x_1 w_1 + x_2 w_2 + x_3 w_3 - y\right)^2\right]$$

$$= x_1 \left(x_1 w_1 + x_2 w_2 + x_3 w_3 - y\right)$$

$$= x_1 \left(\hat{y} - y\right)$$

Iterative Approach: Example

- True value $y = 850$; predicted value $\hat{y} = 500 \implies$

$$\text{Residual error} = 850 - 500 = 350$$

- Below we provide a **delta-rule** update for gradient descent algorithm given a learning rate α:

$$\Delta w_i = \alpha \times \frac{\partial}{\partial w_i}\left[\frac{1}{2}(\hat{y} - y)^2\right],\ i = 1, 2, 3$$

where

$$\frac{\partial}{\partial w_i}\left[\frac{1}{2}(y - \hat{y})^2\right] = \frac{\partial}{\partial w_i}\left[\frac{1}{2}(y - \sum_i w_i x_i)^2\right] = \cdots = -x_i(y - \hat{y})$$

Iterative Approach: Example

- True value $y = 850$; predicted value $\hat{y} = 500 \implies$

$$\text{Residual error} = 850 - 500 = 350$$

- Below we provide a **delta-rule** update for gradient descent algorithm given a learning rate α:

$$\Delta w_i = \alpha \times \frac{\partial}{\partial w_i}\left[\frac{1}{2}(\hat{y} - y)^2\right], \ i = 1, 2, 3$$

where

$$\frac{\partial}{\partial w_i}\left[\frac{1}{2}(y - \hat{y})^2\right] = \frac{\partial}{\partial w_i}\left[\frac{1}{2}(y - \sum_i w_i x_i)^2\right] = \cdots = -x_i(y - \hat{y})$$

- Hence, the eventual weight updates are:

$$w_i^{upd} = w_i - \Delta w_i = w_i + \alpha \times x_i(y - \hat{y}), \quad i = 1, 2, 3$$

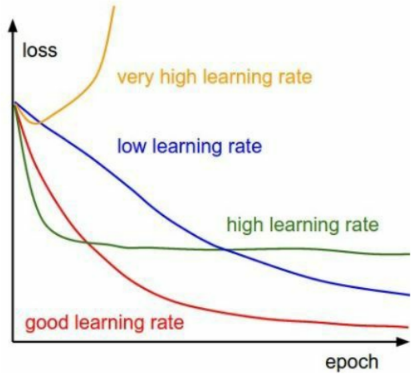
Learning Rate α.

Learning rate α is a scalar parameter used to set the rate of adjustments/updates in order to reduce the training errors faster.

Picking a learning rate value is an art, which can lead to your model:

- training & learning fast,
- training & learning slow,
- not training & learning at all.

Unfortunately, we won't be mastering that art in this course.



Iterative Approach: Example

- If we select learning rate $\alpha = \frac{1}{35}$:
  - $w_1^{upd} = 50 + \frac{1}{35}2(350) = 70,$
  - $w_2^{upd} = 50 + \frac{1}{35}5(350) = 100,$
  - $w_3^{upd} = 50 + \frac{1}{35}3(350) = 80$

$$\hat{y} = 70(2) + 100(5) + 80(3) = 880$$

Iterative Approach: Example

- If we select learning rate $\alpha = \frac{1}{35}$:
  - $w_1^{upd} = 50 + \frac{1}{35}2(350) = 70,$
  - $w_2^{upd} = 50 + \frac{1}{35}5(350) = 100,$
  - $w_3^{upd} = 50 + \frac{1}{35}3(350) = 80$
- The updated predicted value $\hat{y}$:

$$\hat{y} = \sum_i w_i^{upd} x_i = 880,$$

which is much closer to the true value $y = 850$.

Iterative Approach: Example

- If we select learning rate $\alpha = \frac{1}{35}$:
  - $w_1^{upd} = 50 + \frac{1}{35}2(350) = 70$,
  - $w_2^{upd} = 50 + \frac{1}{35}5(350) = 100$,
  - $w_3^{upd} = 50 + \frac{1}{35}3(350) = 80$
- The updated predicted value $\hat{y}$:

$$\hat{y} = \sum_i w_i^{upd} x_i = 880,$$

  which is much closer to the true value $y = 850$.
- Notice that, while the weight for fish and ketchup got **better** (70 is closer to 150, 80 is closer to 100), the weight for chips got **worse** (100 instead of correct value 50)!
- We aren't guaranteed a weight improvement at each update, but over time, given sufficient data, weights typically converge to the correct values.

Linear ANN: Example

Example. Presume we have an ANN of linear neurons with

- Input layer of three neurons: x_1, x_2, x_3,
- Fully-connected hidden layer of two neurons: h_1, h_2,
- One output neuron y.

Linear ANN: Example

Example. Presume we have an ANN of linear neurons with

- Input layer of three neurons: x_1, x_2, x_3,
- Fully-connected hidden layer of two neurons: h_1, h_2,
- One output neuron y.

Presume the following **weight matrices** for

input & **hidden** layer:

Input \ Hidden	h_1	h_2
1 (bias)	0.5	0.3
x_1	0.2	0.5
x_2	0.5	-0.2
x_3	-0.4	0.7

Linear ANN: Example

Example. Presume we have an ANN of linear neurons with

- Input layer of three neurons: x_1, x_2, x_3,
- Fully-connected hidden layer of two neurons: h_1, h_2,
- One output neuron y.

Presume the following **weight matrices** for

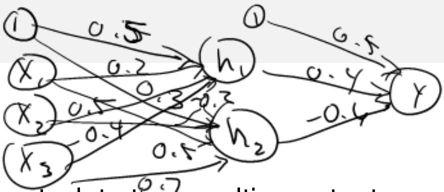
input & **hidden** layer:

Input \ Hidden	h_1	h_2
1 (bias)	0.5	0.3
x_1	0.2	0.5
x_2	0.5	-0.2
x_3	-0.4	0.7

hidden & **output** layer:

Hidden \ Output	y
1 (bias)	0.5
h_1	0.4
h_2	-0.6

Linear ANN: Example



Example (cont'd). Proceed to

- Draw this ANN, and
- Given $x_1 = 5, x_2 = 10, x_3 = -2$, calculate the resulting output
- What is the predicted output $\hat{y}$ for the previous example?

$$h_1 = 0.5 + 0.2X_1 + 0.5X_2 - 0.4X_3$$
$$= 0.5 + 0.2(5) + 0.5(10) - 0.4(-2)$$
$$= 7.3$$

$$h_2 = 0.3 + 0.5X_1 - 0.2X_2 + 0.7X_3$$
$$= 0.3 + 0.5(5) - 0.2(10) + 0.7(-2)$$
$$= -0.6$$

$$\hat{y} = 0.5 + 0.4h_1 - 0.6h_2$$
$$= 0.5 + 0.4(7.3) - 0.6(-0.6) = 3.78$$

Epoch

- One iteration or pass through the process of providing the network with an input and updating the network's weights is called an epoch.
- It is a full run of feed-forward and backpropagation for update of weights. It is also one full read through of the entire dataset.
- Typically, many epochs, in the order of tens of thousands at times, are required to train the neural network efficiently.