CS-4053 Recommender System

Fall 2023

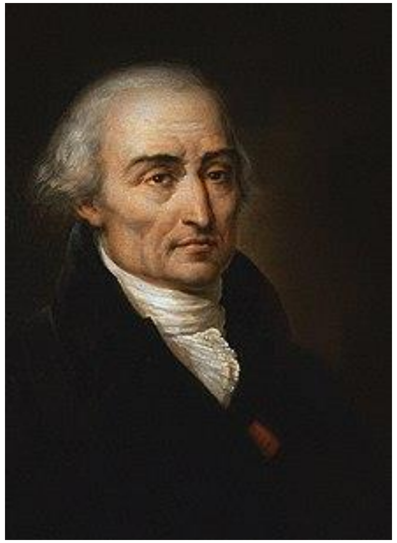
Lecture 8: Neural Networks

Course Instructor: Syed Zain Ul Hassan

National University of Computer and Emerging Sciences, Karachi

Email: zain.hassan@nu.edu.pk

Derivatives: Recap



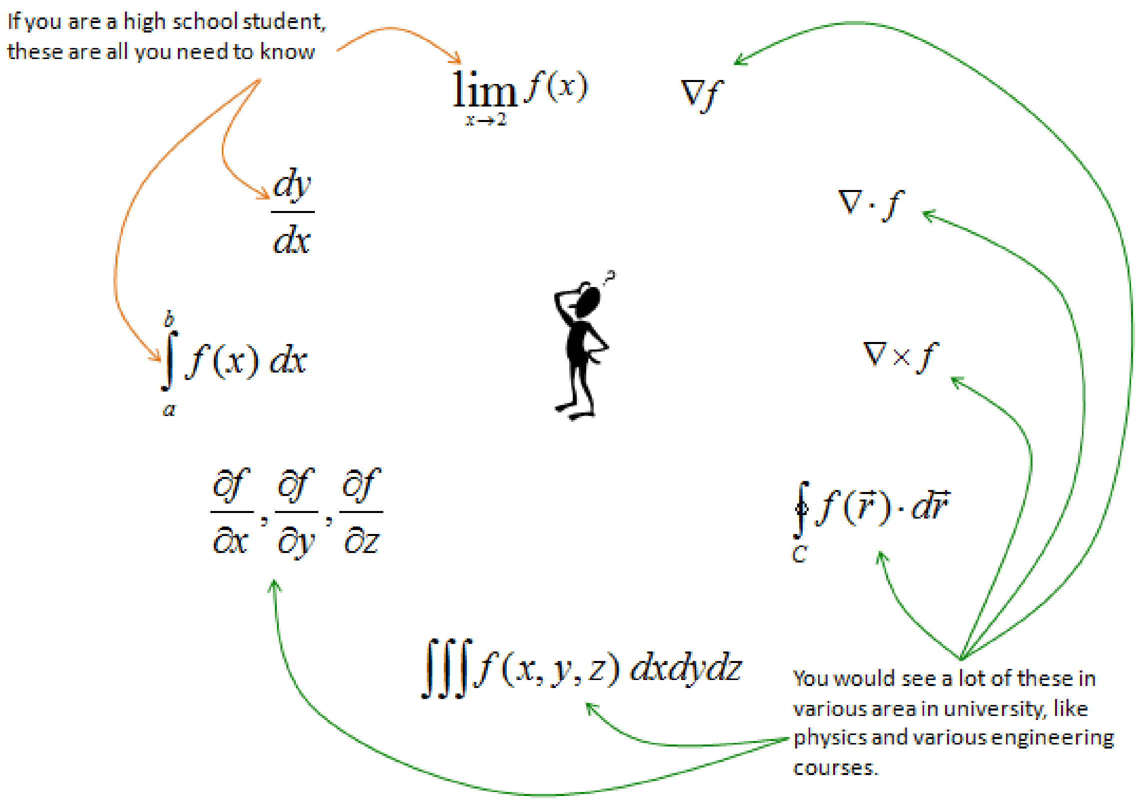
Joseph Lagrange

Gottfried Leibniz

Isaac Newton

Derivatives: Recap

- **Idea:** Intuition of derivatives will be enough



Derivatives: Recap

- Imagine having an arcade machine
- There is a certain game *G1* that you like
- To play this game you need coins
- For every ***x*** coin, you get to play ***2x*** minutes
- So for 1 coin you play 1 minute
- For 2 coins, you play 4 minutes and so on…

Derivatives: Recap

- **For coin x:** 1, 2, 3, 4, 5
- **Play time:** 2, 4, 6, 8, 10

- For every change in coin quantity x, how much more play time we get? ***2 more minutes***

Derivatives: Recap

- **x:** 1, 2, 3, 4, 5
- **f(x) = 2x :** 2, 4, 6, 8, 10

- **Derivative of f(x) :** $\frac{df(x)}{dx} = 2$

*"jump" in the value of **y** w.r.t change in value of **x***

Partial Derivatives: Recap

- Let us assume we have **3** games on this machine
- For every x coin, you get to play $2x$ minutes of **G1**
- For every y coin, you get to play $3y$ minutes of **G2**
- For every z coin, you get to play $5z$ minutes of **G3**

- If we only have coins for **G1**, then **G2** and **G3** don’t matter to us
- If we only have coins for **G2**, then **G1** and **G3** don’t matter to us
- If we only have coins for **G3**, then **G1** and **G2** don’t matter to us

Partial Derivatives: Recap

- A derivative taken w.r.t only one variable while treating the remaining variables as constants

Derivatives: Recap

- **x**: 1, 2, 3, 4, 5
- **y**: 1, 2, 3, 4, 5
- **z**: 1, 2, 3, 4, 5
- $\boldsymbol{f(x, y, z) = 2x + 3y + 5z}$

- $\frac{\partial f(x,y,z)}{\partial x} = 2$
- $\frac{\partial f(x,y,z)}{\partial y} = 3$
- $\frac{\partial f(x,y,z)}{\partial z} = 5$

Gradient: Recap

- Let us assume we have **3** games on this machine
- For every ***x*** coin, you get to play ***2x*** minutes of **G1**
- For every ***y*** coin, you get to play ***3y*** minutes of **G2**
- For every ***z*** coin, you get to play ***5z*** minutes of **G3**

- To "***optimize***" your gaming time, you need to decide coins for which games to use more i.e., you need to find the optimal values of **x**, **y** and **z** and see how much "***change***" that brings to your "***overall***" gaming time/ experience

Gradient: Recap

- Gradient yields a vector whose components are partial derivatives of the function with respect to its variables. You can think of gradient as the overall change i.e., *"change"* w.r.t every variable

$$\nabla_\theta(f) = \begin{bmatrix} \frac{\partial f(\theta)}{\partial x} \\ \frac{\partial f(\theta)}{\partial y} \\ \frac{\partial f(\theta)}{\partial z} \end{bmatrix}$$

Linear Regression

- *(Informally) "Give me a bunch of numbers as input and I will give you a number in return."*

$$\boldsymbol{y} = \boldsymbol{mx} + \boldsymbol{b}$$

- *(Formally)* It allows us to model relationship between a scalar value and one or more variables

Linear Regression

- Consider the following equation for **m = 2** and **b = 3**:

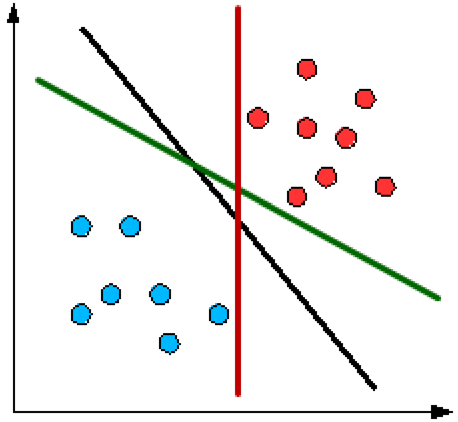
$$\boldsymbol{y} = mx + b = \mathbf{2}\boldsymbol{x} + \mathbf{3}$$

- For **x = 1, y = 5**
- For **x = 2, y = 7**
- For **x = 3, y = 9**

and so on…

Issue: Linearly Separable Data

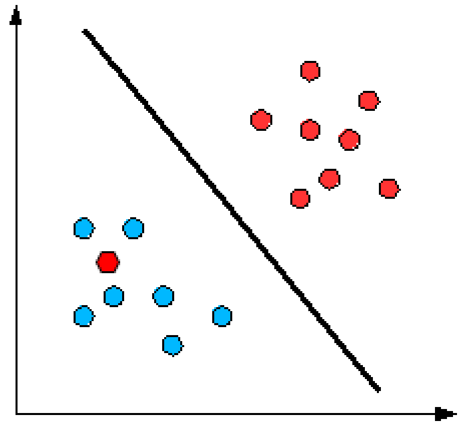
- Let us assume that we have to "separate" the red and blue data points



- We can essentially separate the given data into red and blue sets using a linear equation

Issue: Linearly Separable Data

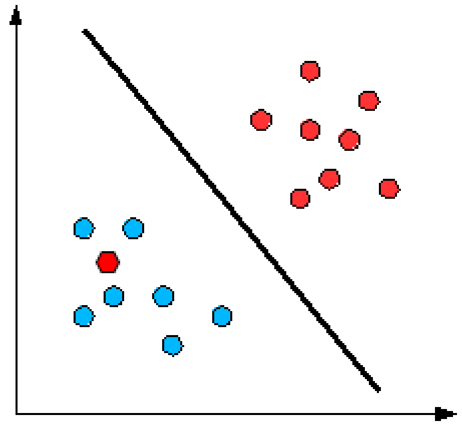
- Unfortunately, we are not so lucky in reality



- The data we have is rarely ever linearly separable hence a line is not enough to “separate” the given data points

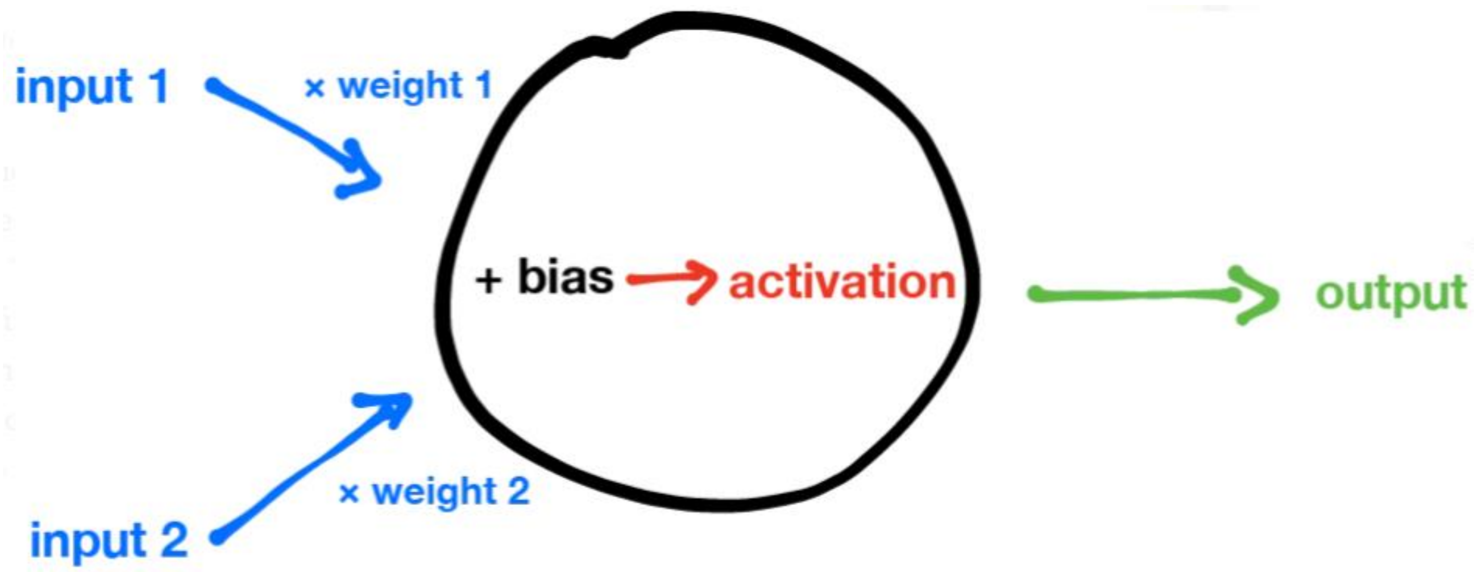
Issue: Linearly Separable Data

- We need non-linearity to separate these data points



Neural Network: What is a Neuron?

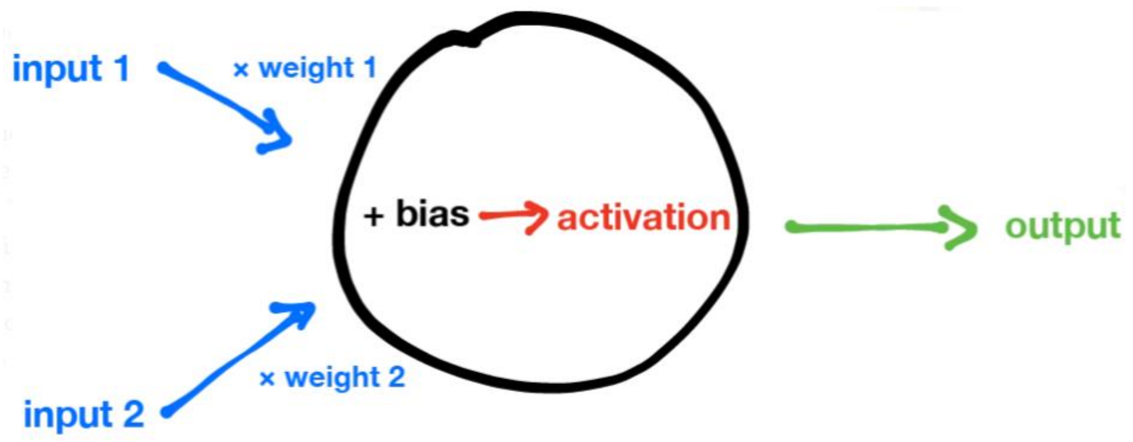
- A **neuron** takes any number of inputs and spits out an output



Neural Network: What is a Neuron?

- If we ignore the activation, this neuron can be expressed as:

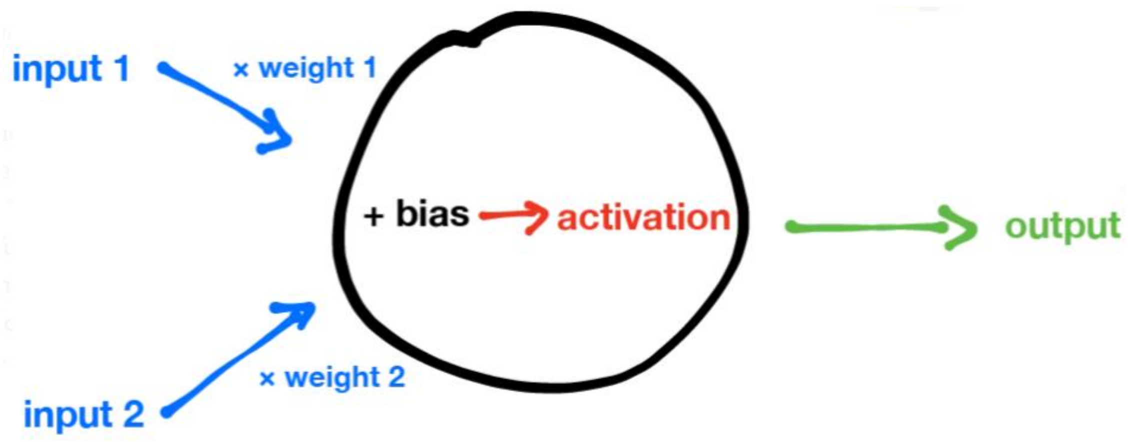
$$y = \boldsymbol{weight_1} * \boldsymbol{input1} + \boldsymbol{weight_2} * \boldsymbol{input2} + \boldsymbol{bias}$$



Neural Network: What is a Neuron?

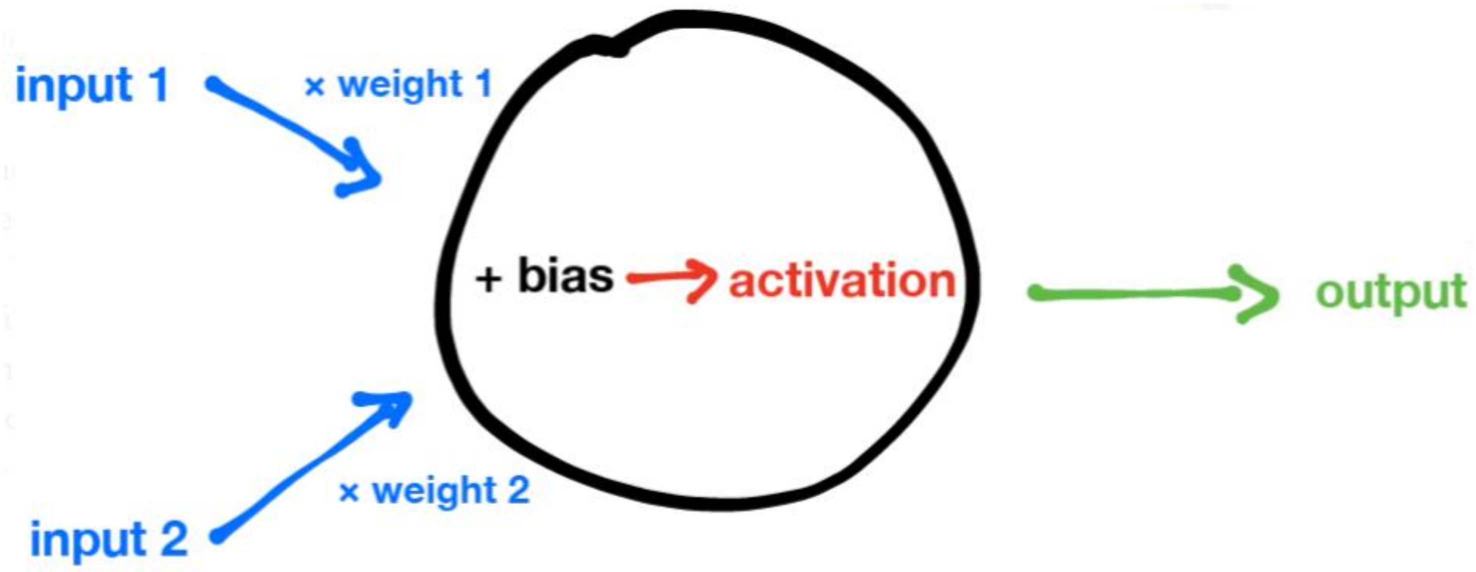
- Without activation, we are performing **regression** with this neuron:

$$y = weight_1 * input1 + weight_2 \ * input2 + bias$$



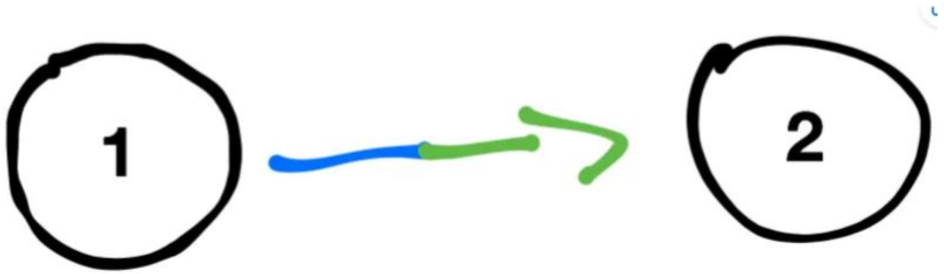
Neural Network: What is an Activation?

- A **activation function** adds non-linearity to the output of the neuron and helps decide whether the neuron should be "*activated*" or not



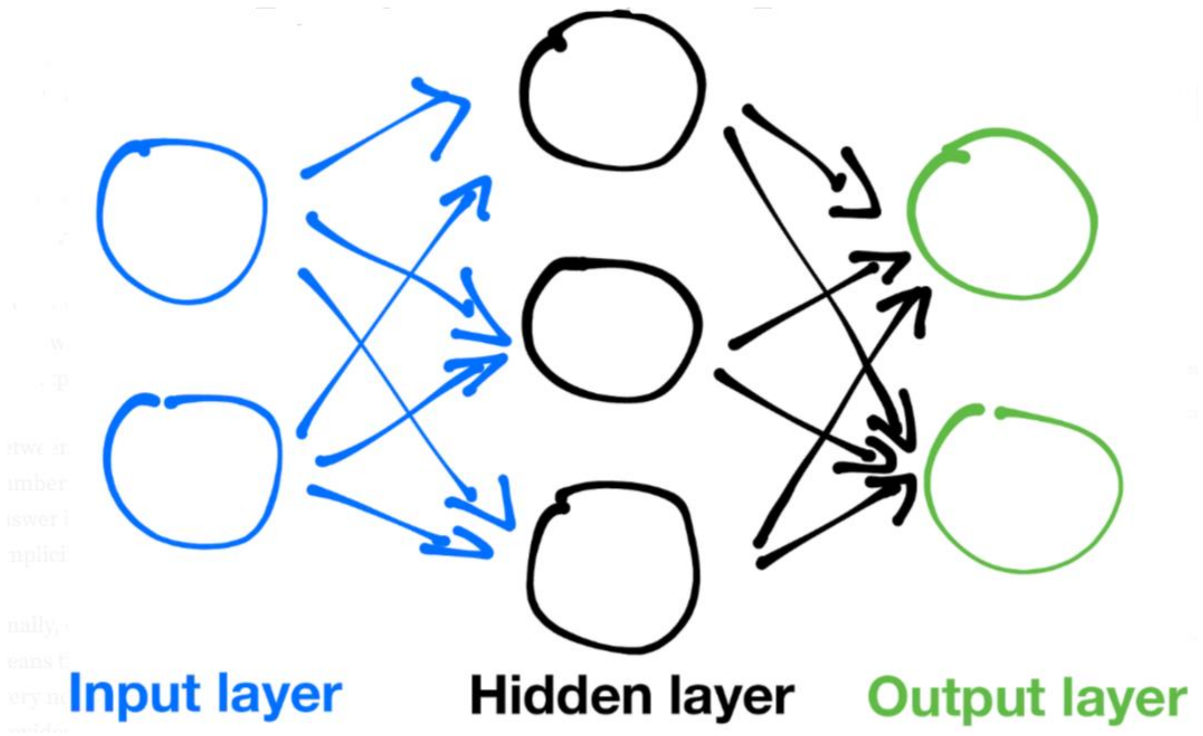
Neural Network: Why is it called a Network?

- A single neuron doesn't really help us find complex patterns *(and is not cool enough!)*
- We need a network of inter-connected neurons to make complex decisions
- The output of one neuron becomes the input of another neuron



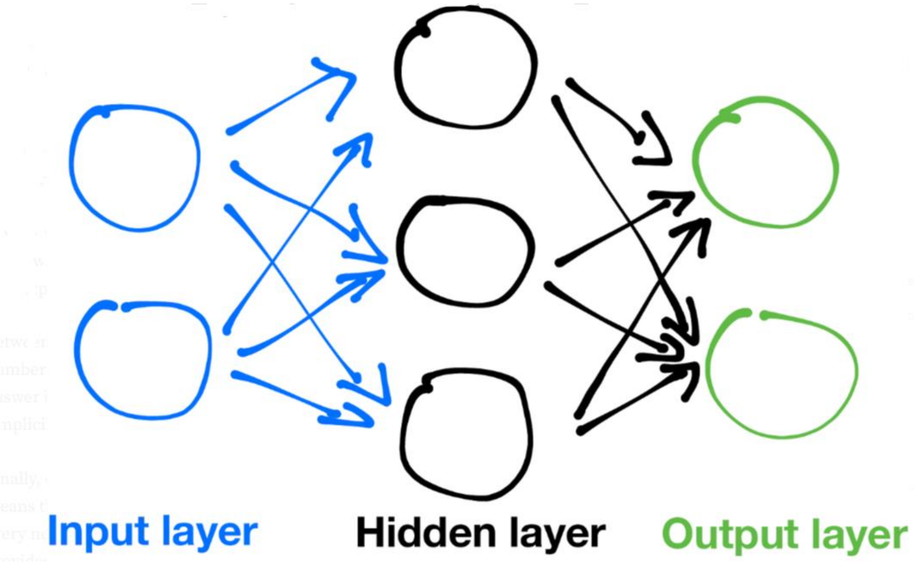
Neural Network: Architecture

- A neural network is composed of layers of neurons



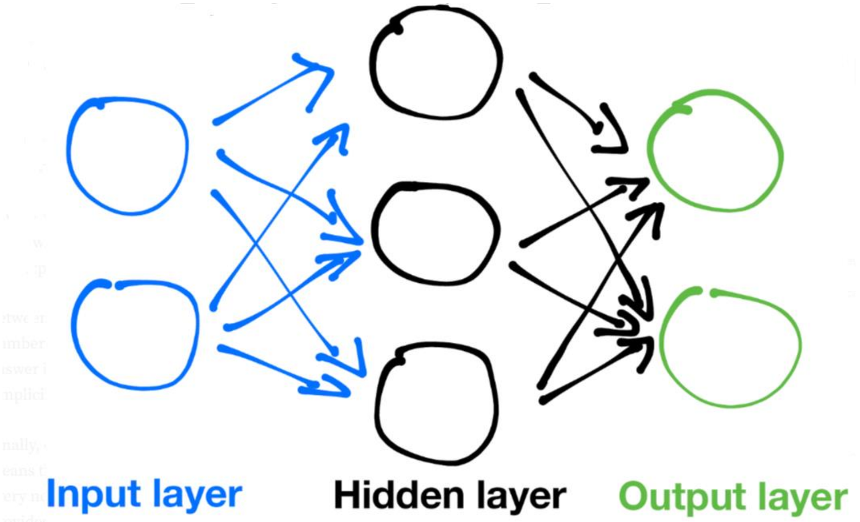
Neural Network: Architecture

- A neural network is composed of fully-connected layers of neurons
- In general, there are three types of layers: an input layer, one or more hidden layers, and an output layer



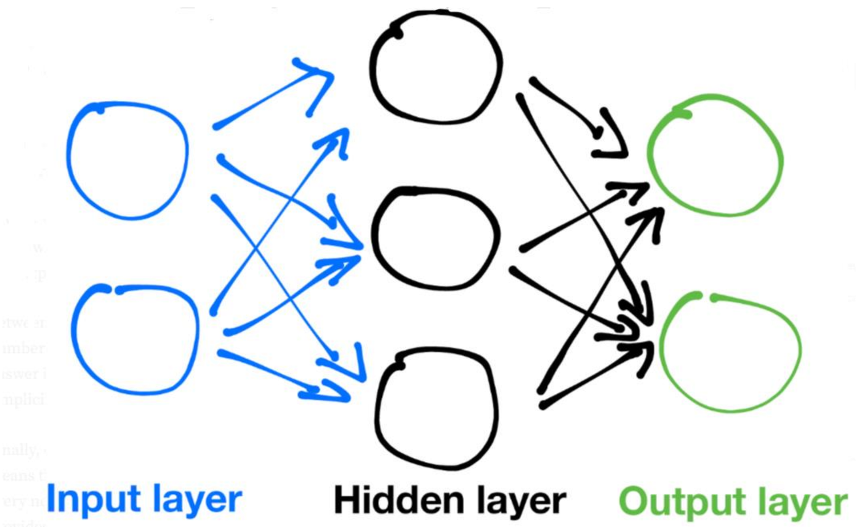
Neural Network: Architecture

- The **input layer** will take on values of whatever the input is to the neural network
- We can have our network take any number of inputs by changing the number of neurons in the input layer



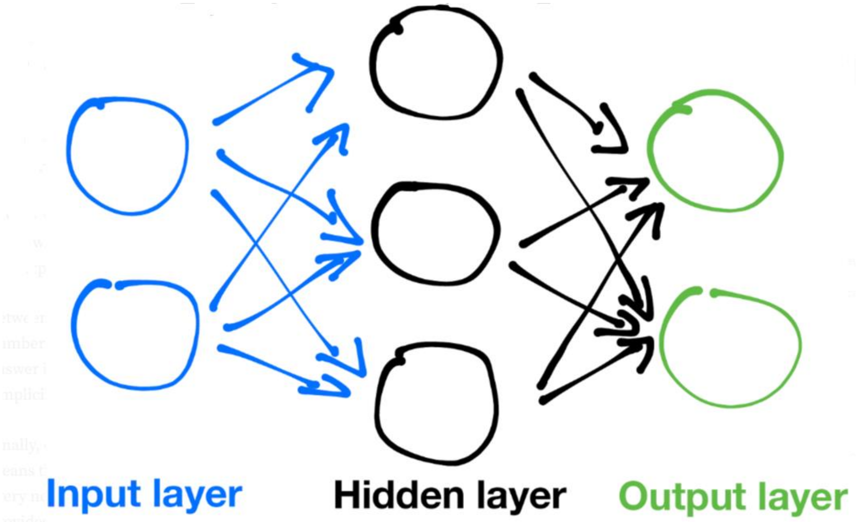
Neural Network: Architecture

- The output of the **output layer** will be the output of the whole neural network
- We can change the number of neurons in the output layer to match the number of outputs we want



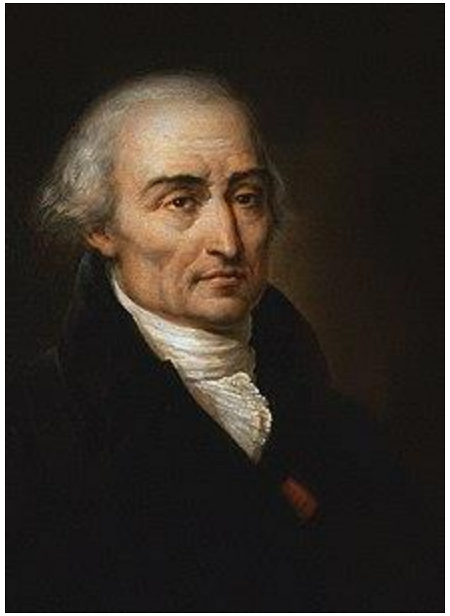
Neural Network: Architecture

- Between the input layer and the output layer are **hidden layers**
- We cannot generally know the number of hidden layers we should use *(nice!)*

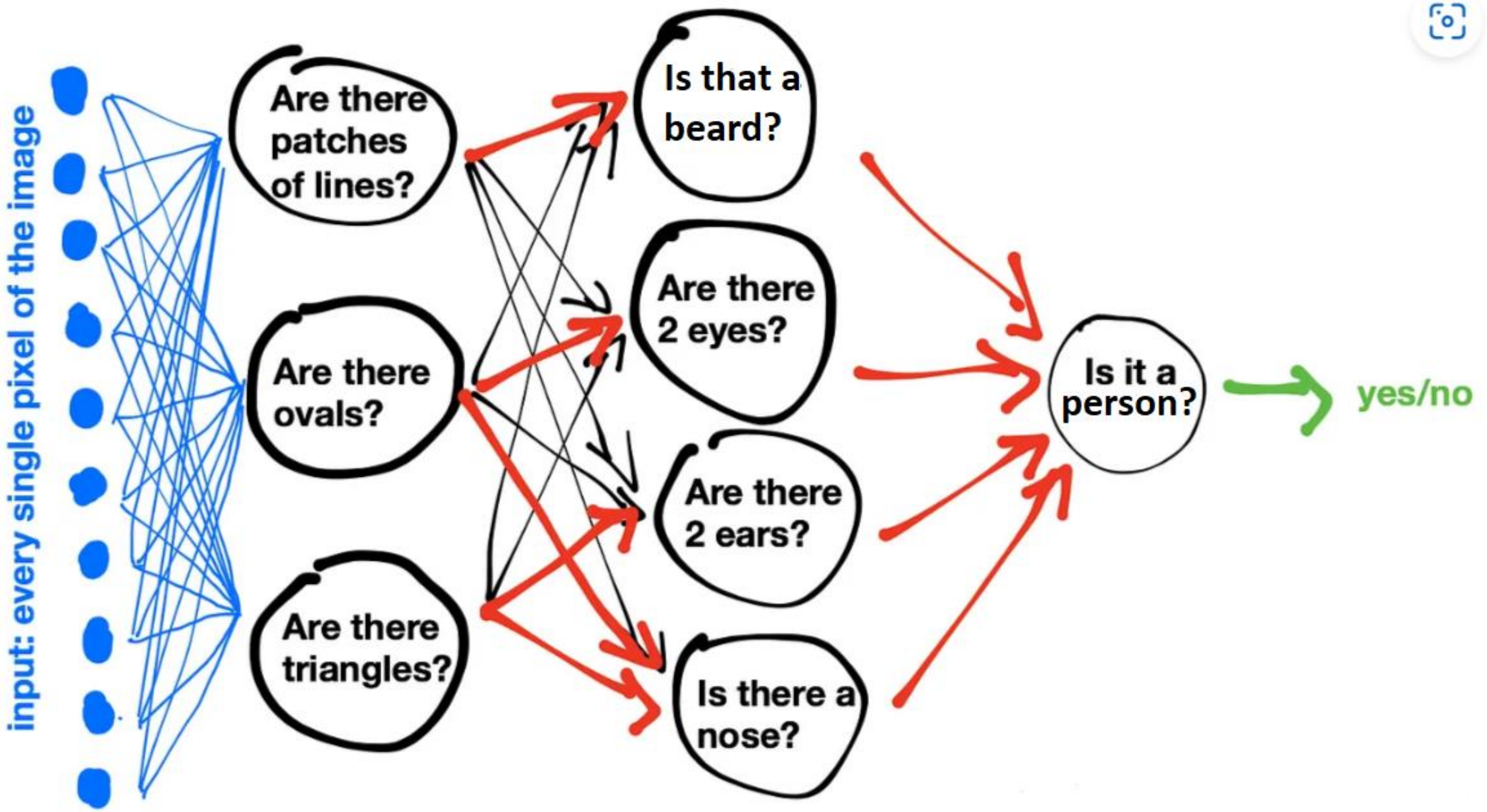


Neural Network: How does it work?

- If we want to identify a person we need to look for features
- Which features are relevant (discriminative)?

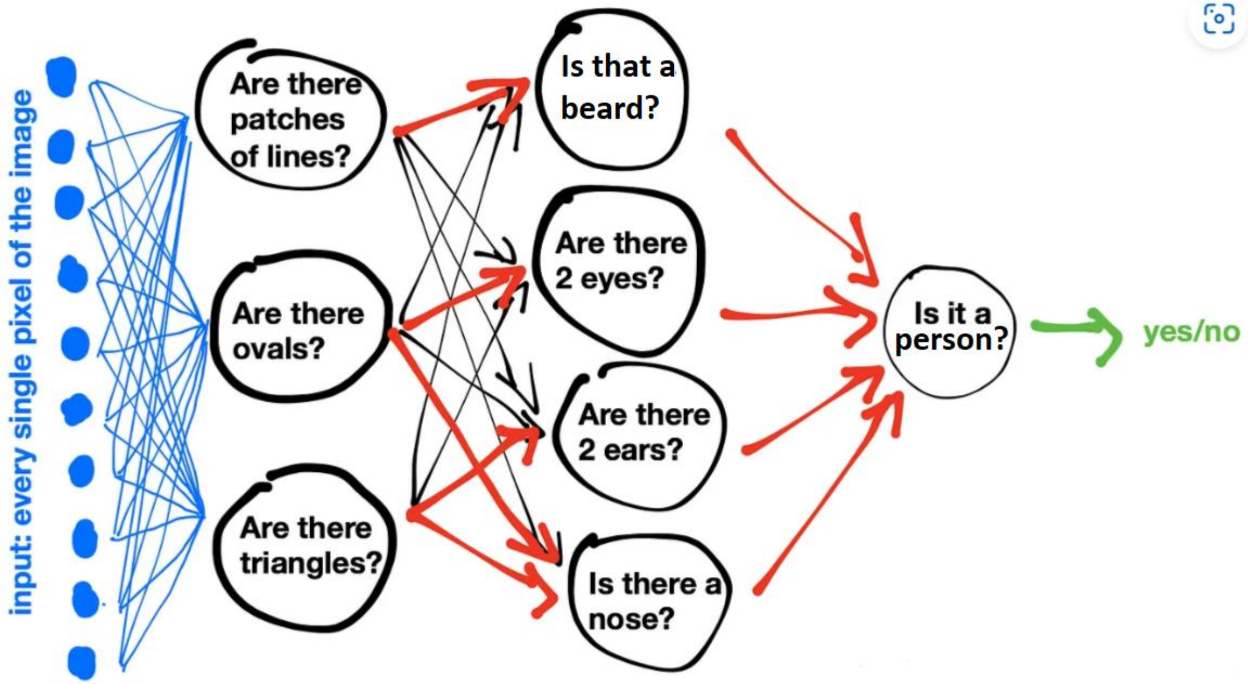


Neural Network: How does it work?



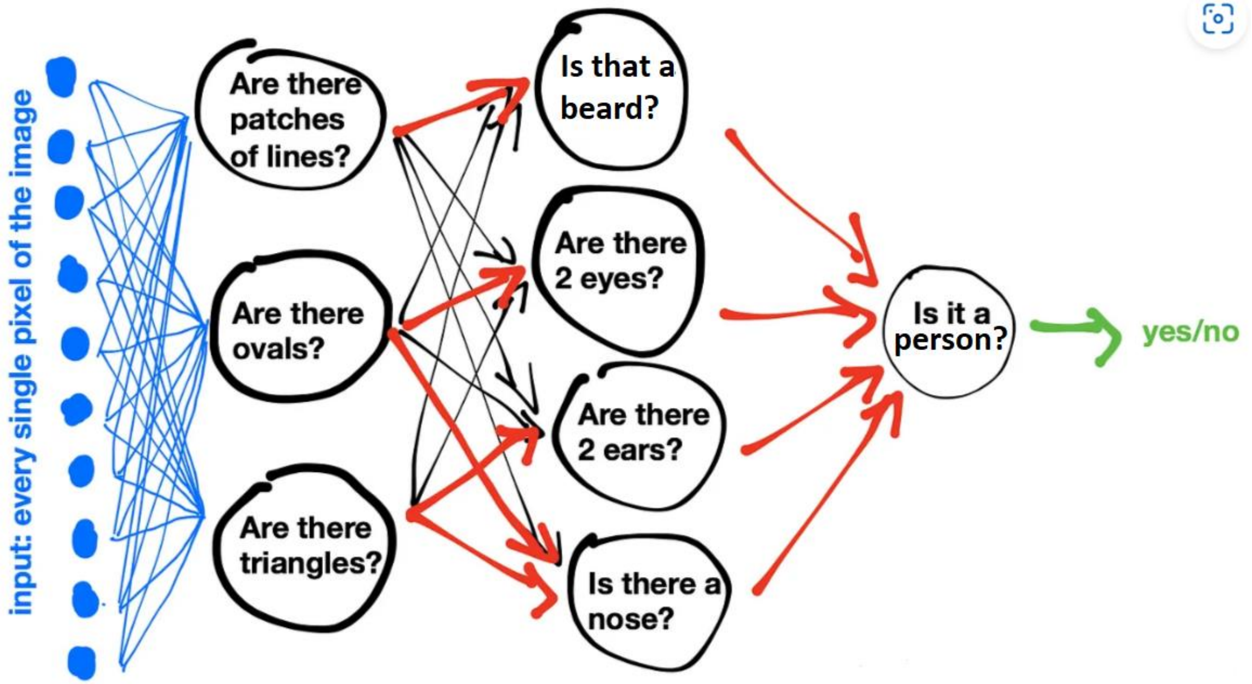
Neural Network: How does it work?

- Notice that the neurons that are deeper ask more abstract questions about the features



Neural Network: Training

- How do we train our neural network to identify a person?



Neural Network: Training

1. Feed raw input (features) to the input layer
2. Initialize all the weights and biases for hidden layers with random values
3. Test if the network can accurately produce the output
    I. If it does not produce accurate results then adjust the weights and biases. It simply means we want to use optimization (e.g. gradient descent) to minimize the value of our loss function. Repeat **Step 3**
    II. If it does produce accurate results then terminate training

*To "adjust" the parameters (weights and biases) we use **backpropagation***

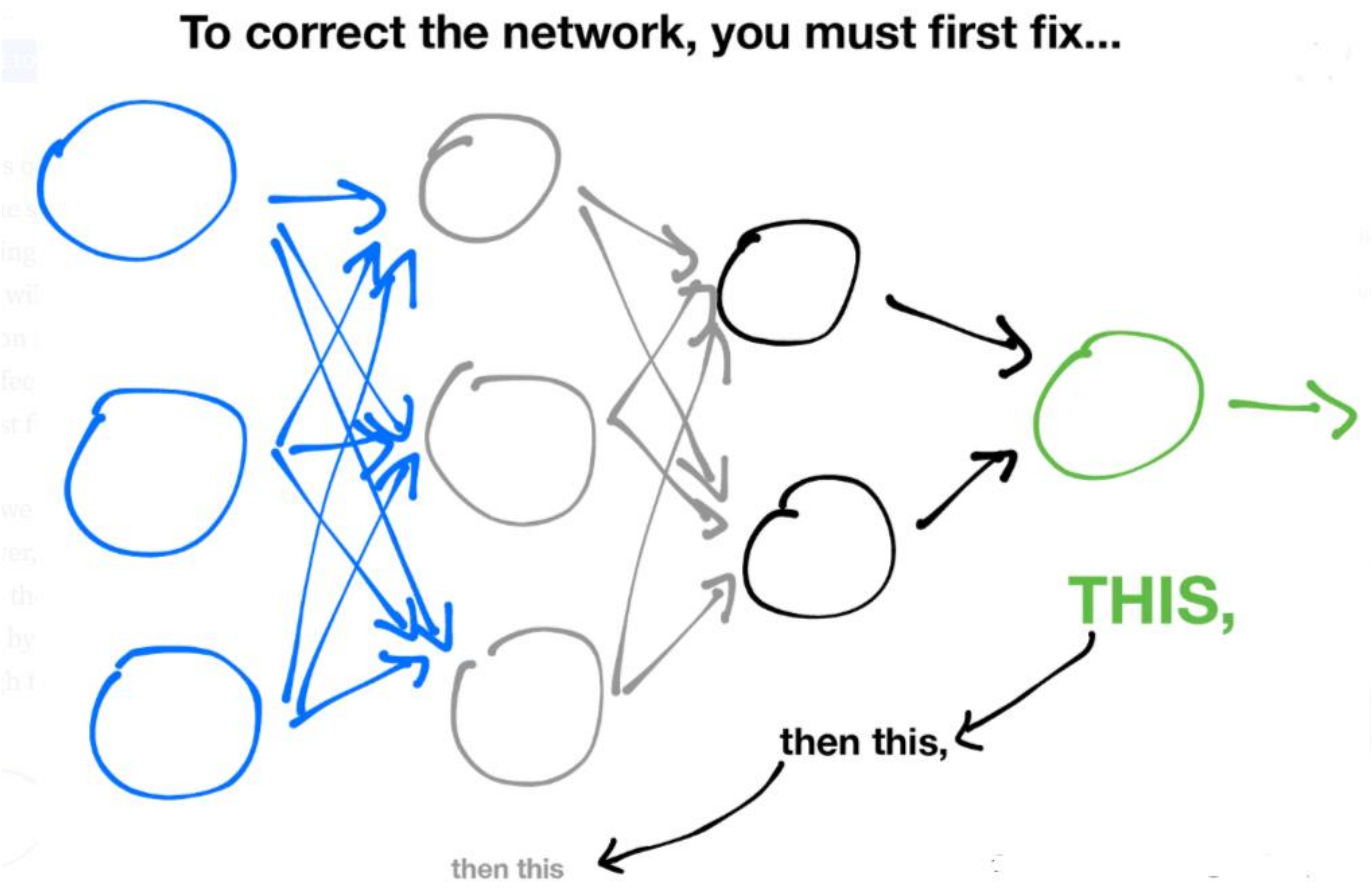
Neural Network: Training

- When the neural network outputs the wrong answer, you find the slopes (derivative) of the output layer first because it was the direct cause of the incorrect answer

- Since the output layer depends on the hidden layer, you'll have to fix that too by finding the slopes and using gradient descent

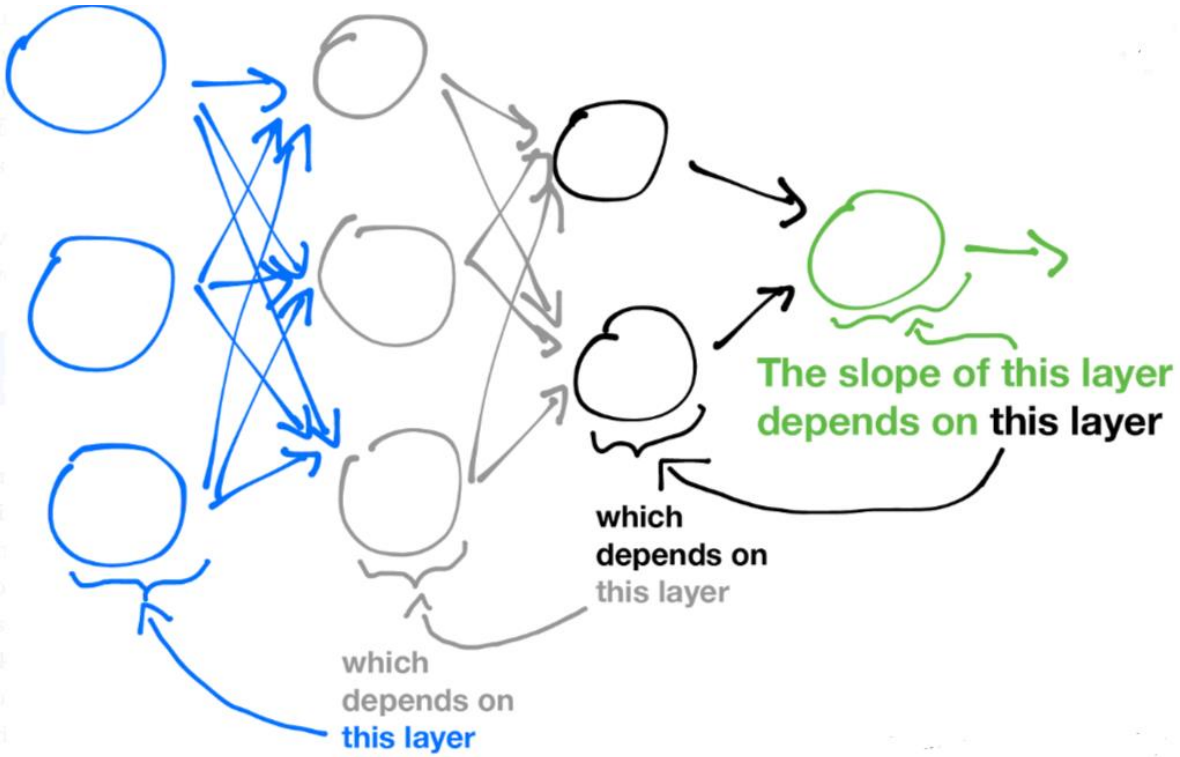
- Eventually you'll work your way back (backpropagate) to the first hidden layer

Neural Network: Backpropagation



Neural Network: Backpropagation

- We calculate slopes by starting from the back and moving backwards through the network until we get all the slopes for gradient descent

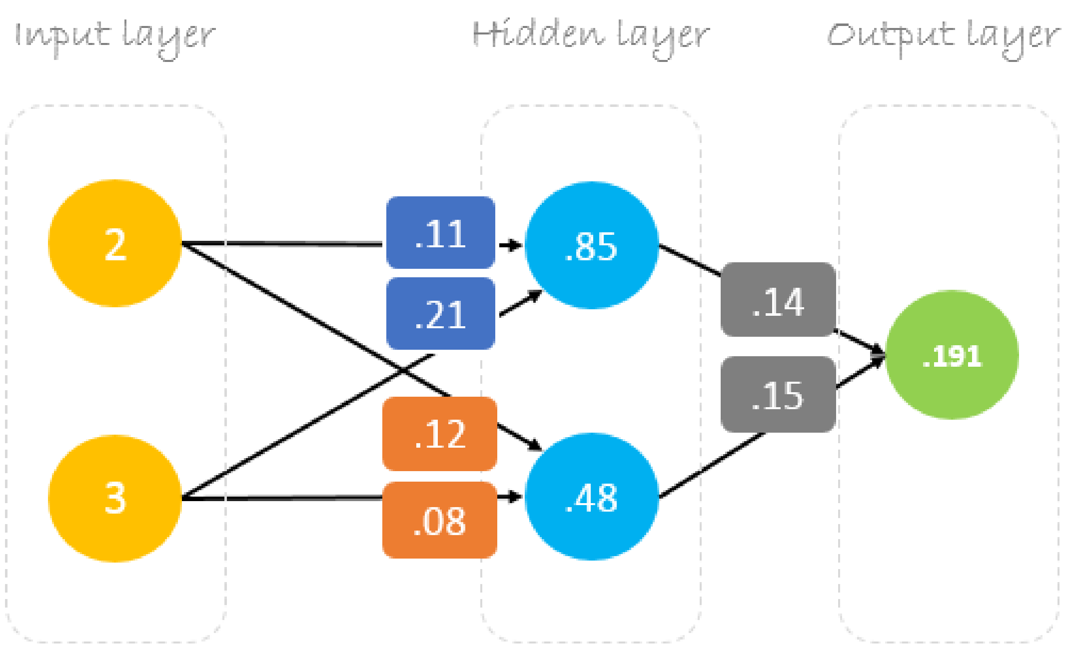


Neural Network Training: Example

- Let us consider the following ANN. Our target output is **1** and initial weights are:
  - **w1 = 0.11**
  - **w2 = 0.21**
  - **w3 = 0.12**
  - **w4 = 0.08**
  - **w5 = 0.14**
  - **w6 = 0.15**

Neural Network Training: Forward Pass

- Let us consider the following ANN. Our target output is **1**

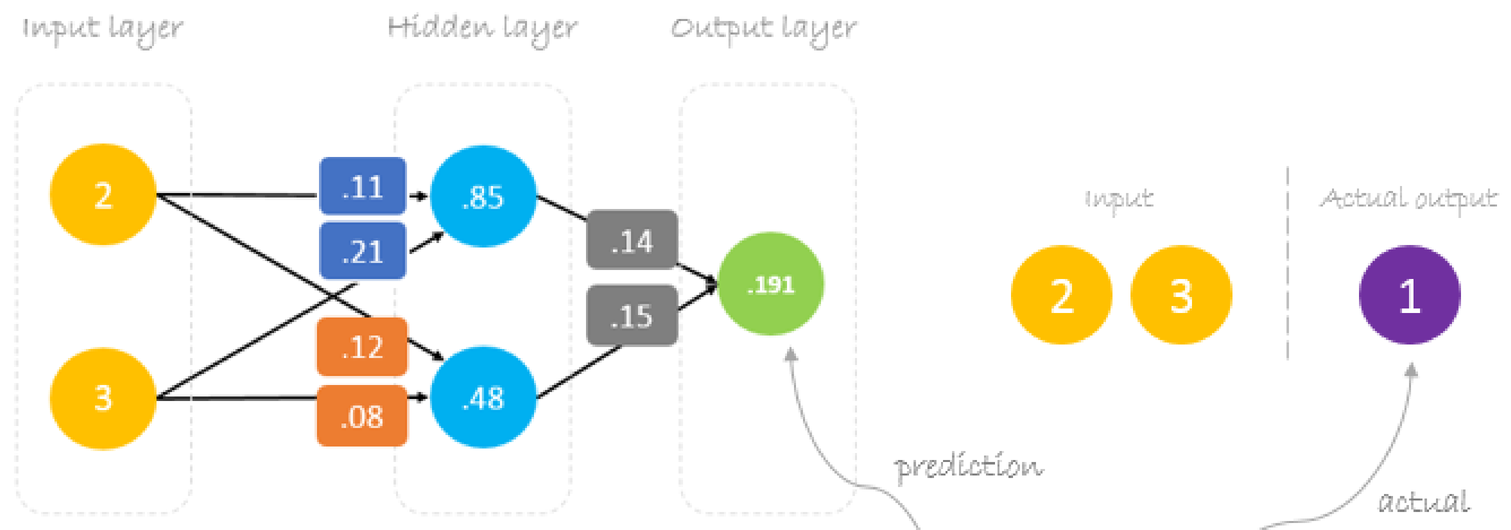


2 x .11 + 3 x .21 = .85

2 x .12 + 3 x .08 = .48

.85 x .14 + .48 x .15 = .191

Neural Network Training: Forward Pass



Error = 0 , if prediction = actual

$$\text{Error} = \frac{1}{2}(\text{prediction} - \text{actual})^2$$

Error is always positive because of the square

$\frac{1}{2}$ is added to ease the calculation of the derivative

$$\text{Error} = \frac{1}{2}(0.191 - 1.0)^2 = 0.327$$

Neural Network Training: Backward Pass

- Since the value of weights influence the error. We need to update the weights in order to reduce this error (loss) by using Gradient Descent

$$\underbrace{{}^{*}W_x}_{\text{New weight}} = \overbrace{W_x}^{\text{Old weight}} - \underbrace{a}_{\text{Learning rate}} \left(\overbrace{\frac{\partial Error}{\partial W_x}}^{\text{Derivative of Error with respect to weight}}\right)$$

Neural Network Training: Backward Pass

- The derivative of our loss can be evaluated by using Chain rule:

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$

chain rule

Error = $\frac{1}{2}(\text{prediction} - \text{actual})^2$

prediction = $(i_1\, w_1 + i_2\, w_2)\, w_5 + (i_1\, w_3 + i_2\, w_4)\, w_6$

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial\,(i_1\, w_1 + i_2\, w_2)\, w_5 + (i_1\, w_3 + i_2\, w_4)\, w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(\text{predictoin} - \text{actula})\frac{\partial(predictoin - actula)}{\partial prediciton} * (i_1\, w_3 + i_2\, w_4)$$

$h_2 = i_1 w_3 + i_2 w_4$

$$\frac{\partial Error}{\partial W_6} = (\text{predictoin} - \text{actula}) * (h_2)$$

Δ = prediction – actual

delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

Neural Network Training: Backward Pass

- Now to update **w6**, we can use the formula:

$$^{*}W_6 = W_6 - a\,\Delta h_2$$

$$*W_6 = 0.15 – (0.05) * (-0.809 * 0.48)$$

$$*W_6 = 0.169$$

Δ = 0.191 – 1 = -0.809 ← **Delta** = prediction - actual

a = 0.05 ← **Learning rate,** we smartly guess this number

Neural Network Training: Backward Pass

- We can compute and update **w5** in a similar manner

$$ {}^{*}W_6 = W_6 - a\,\Delta h_2 $$

$$ {}^{*}W_6 = 0.15 - (0.05) * (-0.809 * 0.48) $$

$$ {}^{*}W_6 = 0.169 $$

$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

$a = 0.05$ ← **Learning rate,** we smartly guess this number

Neural Network Training: Backward Pass

- In order to update **w1** (existing between input layer and hidden layer):

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial h_1} * \frac{\partial h_1}{\partial W_1}$$ chain rule

Error = $\frac{1}{2}$(prediction – actual)^2

prediction = $(h_1)\, w_5 + (h_2)\, w_6$

$h_1 = i_1 w_1 + i_2 w_2$

$$\frac{\partial Error}{\partial W_1} = \frac{\partial \frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial\, (h_1)\, w_5 + (h_2)\, w_6}{\partial h_1} * \frac{\partial\, i_1 w_1 + i_2 w_2}{\partial w_1}$$

$$\frac{\partial Error}{\partial W_1} = 2 * \frac{1}{2}(\mathbf{predictoin} - \mathbf{actula}) \frac{\partial(predictoin - actula)}{\partial prediciton} * (w_5) * (i_1)$$

$$\frac{\partial Error}{\partial W_1} = (\mathbf{predictoin} - \mathbf{actula}) * (w_5 i_1)$$

Δ = prediction – actual delta

$$\frac{\partial Error}{\partial W_1} = \Delta\, w_5 i_1$$

Neural Network Training: Backward Pass

- Now to update **w1**, we can use the formula:

$$*W_1 = W_1 - a\ (\Delta\ W_{5}\ .\ i_1)$$

$$*W_1 = 0.11 - (0.05) * (-0.809) * (0.28)$$

$$*W_1 = 0.121$$

$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

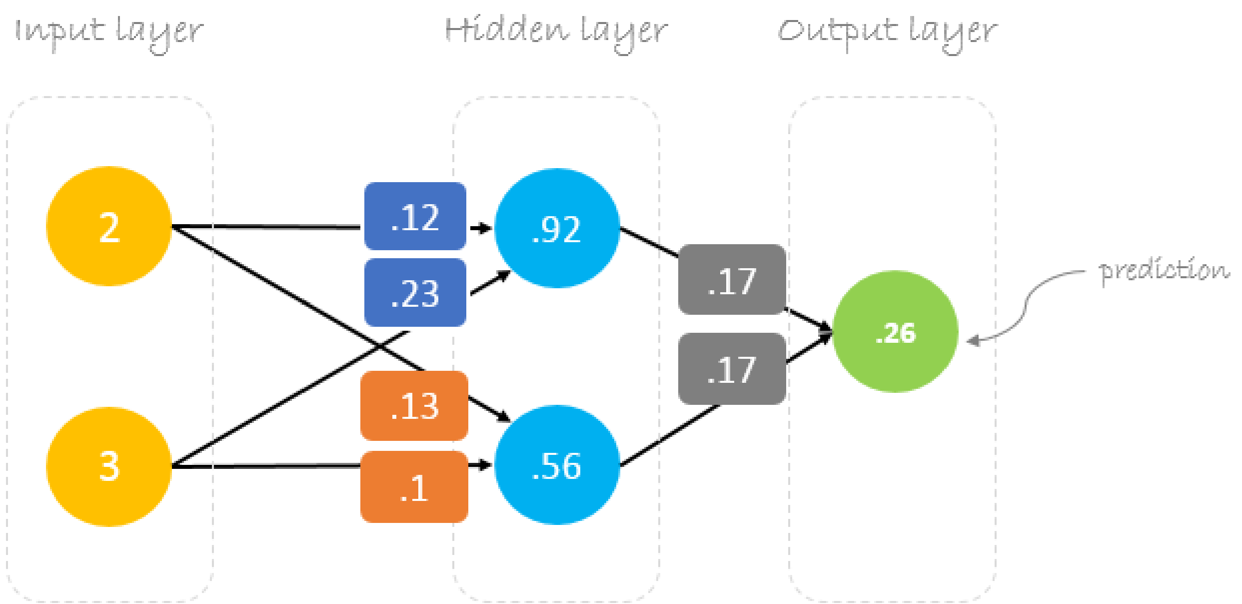
$a = 0.05$ ← **Learning rate,** we smartly guess this number

Neural Network Training: Updated Weights

- All the updated weights *(rounded off to 2 digits)* are as follows:
  - **w1 = 0.12**
  - **w2 = 0.23**
  - **w3 = 0.13**
  - **w4 = 0.10**
  - w5 = 0.17
  - w6 = 0.17
- **Task:** Re-calculate and verify the updated values of these weights

Neural Network Training: Forward Pass *(again)*

- Now, using the new weights we will repeat the forward pass



- Notice that our prediction is slightly better but still not perfect. Therefore, keep updating the weights and performing backward-forward passes until the error is minimized

Are we still missing something?

- Bias
- Activation
  - Sigmoid *(for binary classification)*
  - ReLU *(suffers from vanishing gradient problem)*
  - LeakyReLU *(improved ReLU)*
  - Softmax *(for multi-class classification)*
- Regularization
- Dropout
- Optimization
  - Stochastic Gradient Descent
  - Batch and Mini-batch Gradient Descent
  - Momentum
  - Adam