Neural Networks

Astrini Sie

asie@seattleu.edu

Week 7

ECEGR4750 - Introduction to Machine Learning
Seattle University

November 2, 2023

Overview

Additional Reading

- Neural Networks 1 from CS231n: Convolutional Neural Networks for Visual Recognition at Stanford University
- http://experiments.mostafa.io/public/ffbpann/

Linear Classifiers

All the stuff that we learned so far (Linear and Logistic Regression) are linear classifiers

- Make predictions based on linear combinations of input features
- Decision boundaries are linear in the feature space (hyperplane in the n-dimensional feature space)

Linear Classifiers

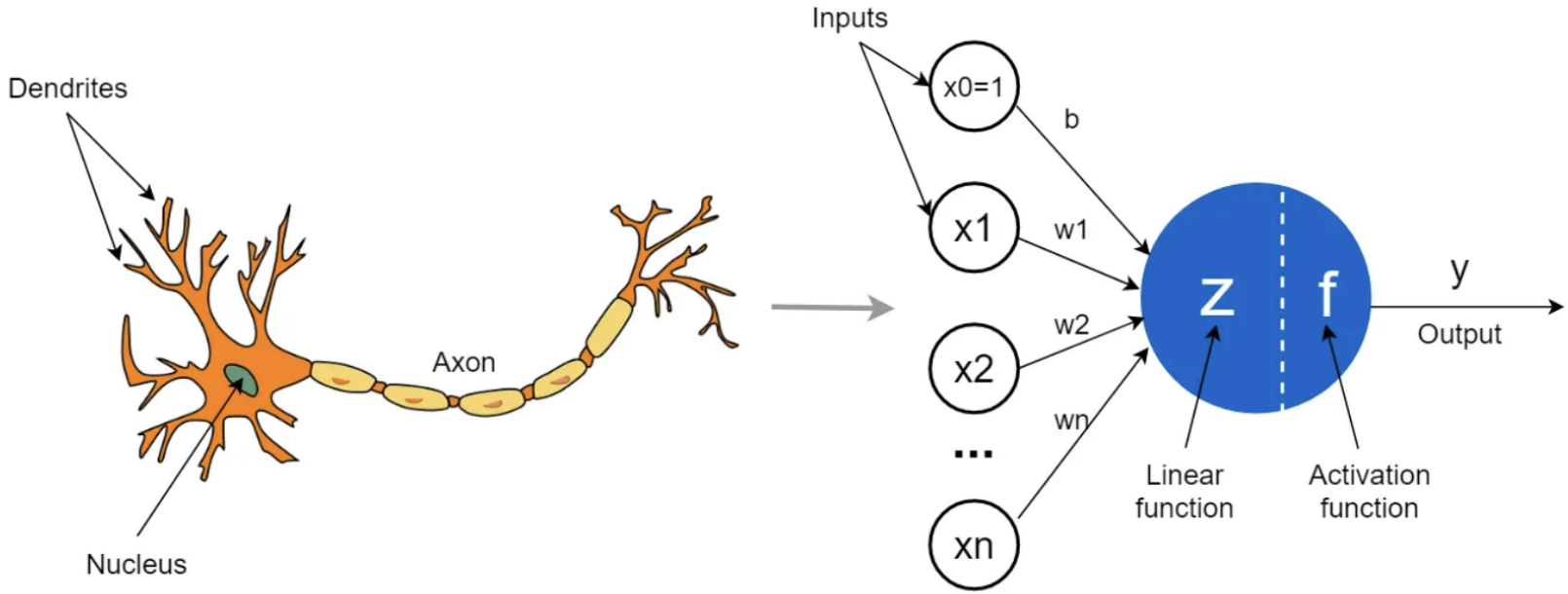
I thought all real world stuff is non-linear?

Linear Classifiers

I thought all real world stuff is non-linear?

Neural networks are excellent for capturing complex, non-linear relationships in data!

Perceptron

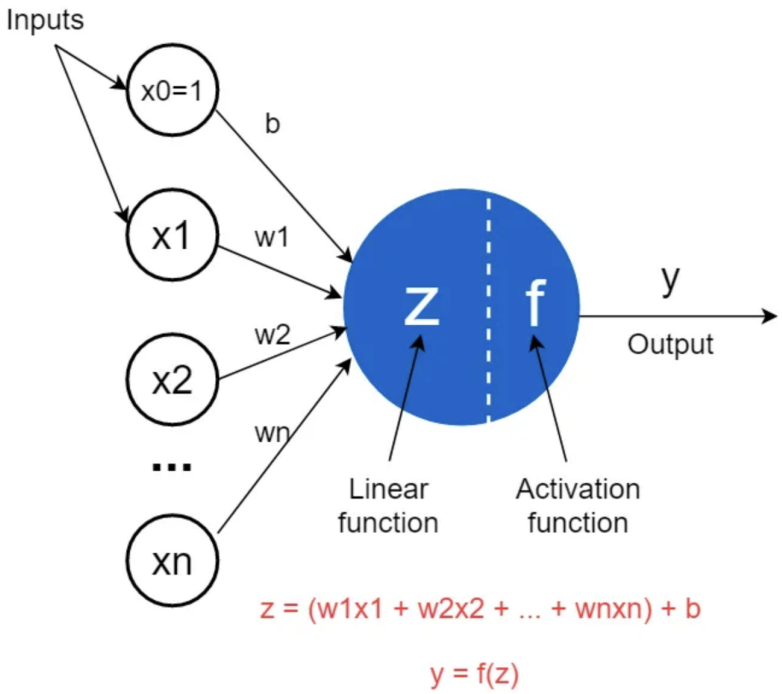


Left source and Right source

- Mimics biological neurons in the brain.
- Concept started in the 1940s, and got popularized in the 90s, and repopularized in the 2010s after GPU.

The structure of a perceptron

Inputs, Outputs, and Parameters



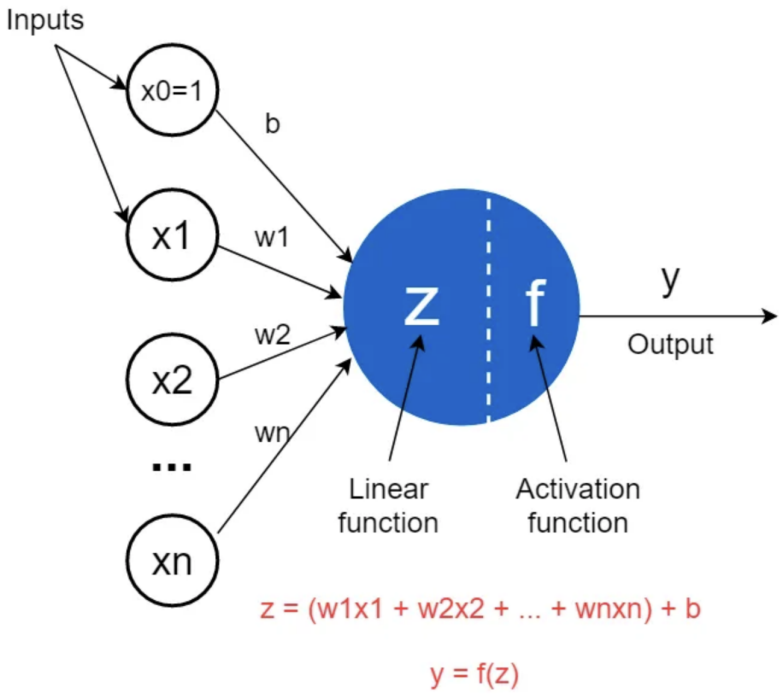
source

Inputs, Outputs, and Parameters

1. **Inputs**: $x_0, x_1, \ldots, n$
   Raw data, or output from other neurons

The structure of a perceptron

Inputs, Outputs, and Parameters



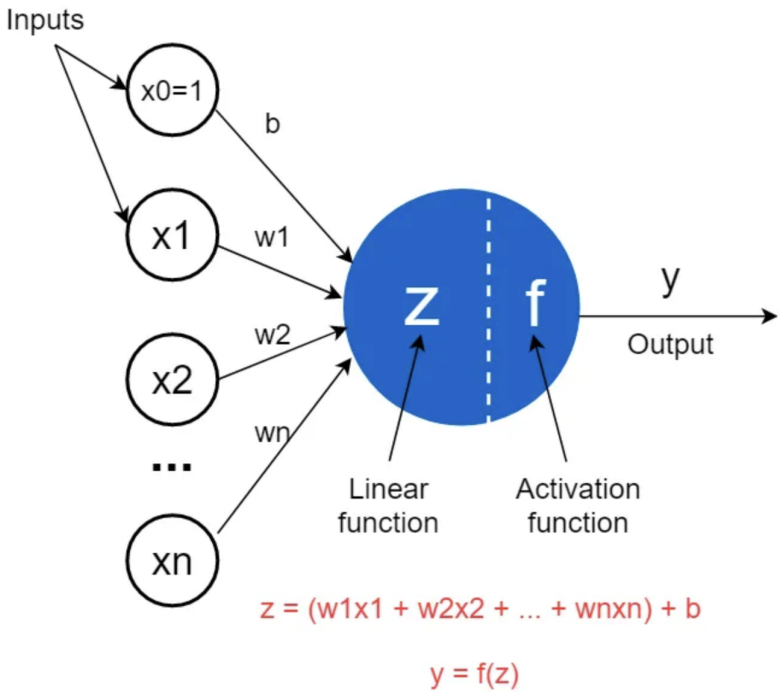
source

Inputs, Outputs, and Parameters

1. **Inputs**: $x_0, x_1, \ldots, n$
   Raw data, or output from other neurons
2. **Weights**: $w_1, w_2, \ldots, w_n$
   Control the level of importance of each input -¿ higher the weight, more important the input.

The structure of a perceptron

Inputs, Outputs, and Parameters



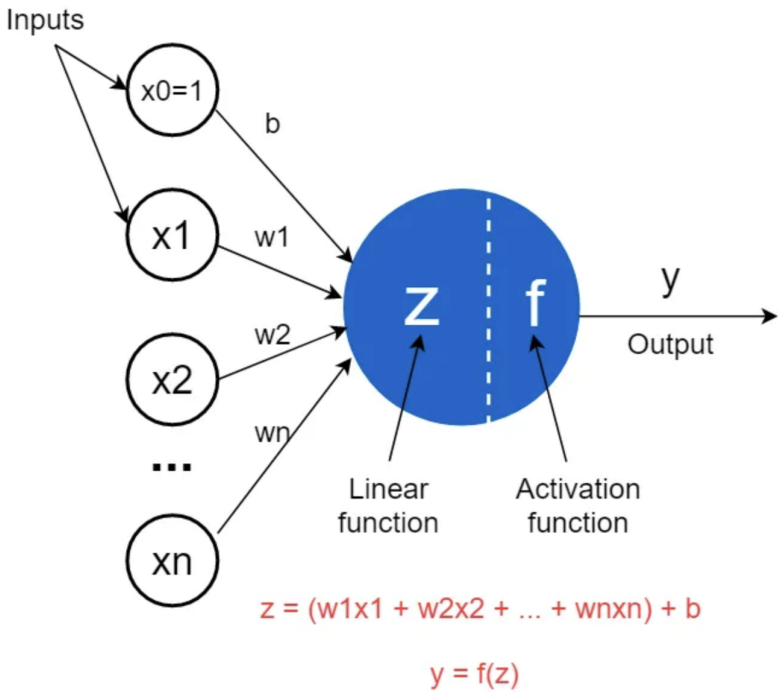
source

Inputs, Outputs, and Parameters

1. **Inputs**: $x_0, x_1, \ldots, n$
   Raw data, or output from other neurons
2. Weights: $w_1, w_2, \ldots, w_n$
   Control the level of importance of each input -¿ higher the weight, more important the input.
3. **Bias**: b
   To ensure that the activation function does not get a zero value in case all $x1, \ldots, x_n$ are zero.

The structure of a perceptron

Inputs, Outputs, and Parameters



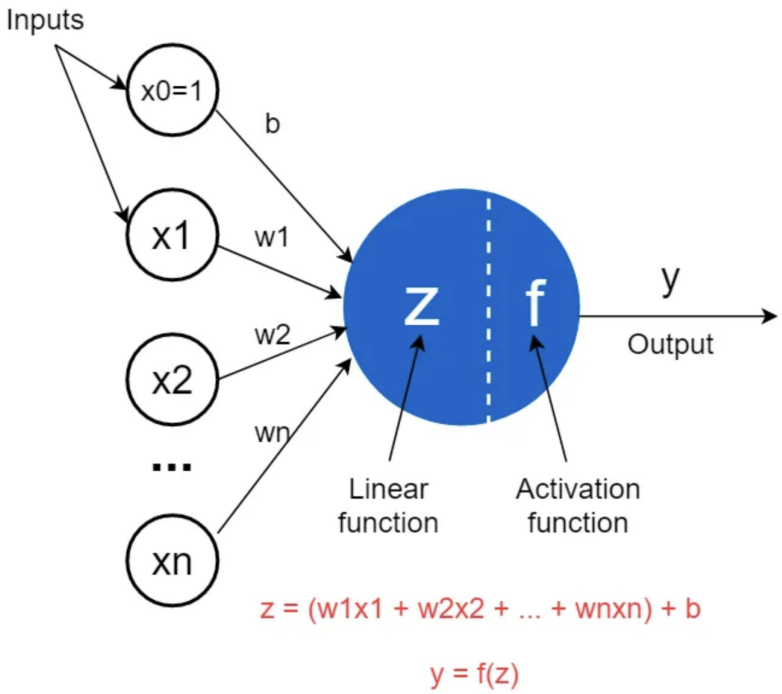
source

Inputs, Outputs, and Parameters

1. **Inputs**: $x_0, x_1, \ldots, n$
   Raw data, or output from other neurons
2. **Weights**: $w_1, w_2, \ldots, w_n$
   Control the level of importance of each input -¿ higher the weight, more important the input.
3. **Bias**: b
   To ensure that the activation function does not get a zero value in case all $x1, \ldots, x_n$ are zero.
4. **Output**: $y = f(z)$

The structure of a perceptron

Functions



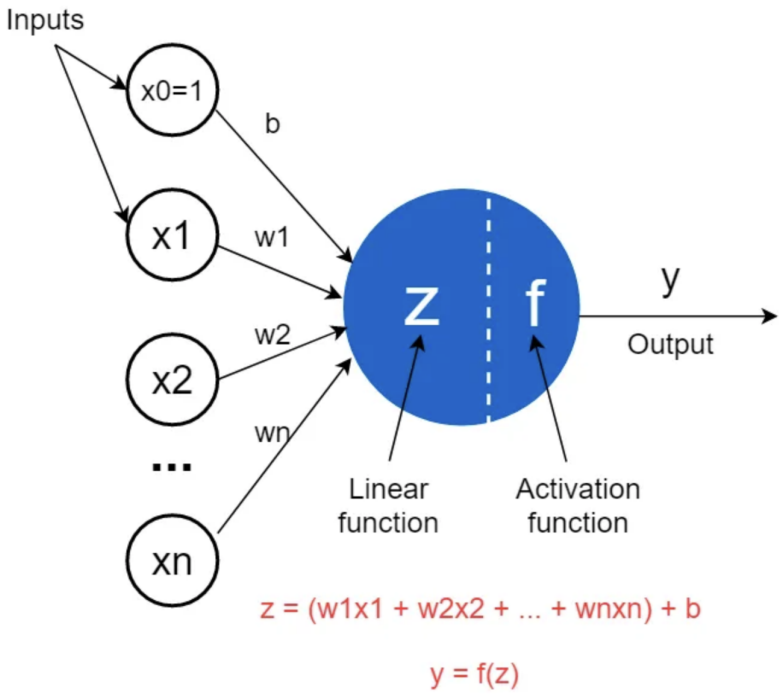
source

Functions

1. Linear Function:
   $z = w_1x_1 + \ldots + w_nx_n$
   Weighted sum of inputs (linear combination).

The structure of a perceptron

Functions



source

Functions

1. Linear Function:
   $z = w_1x_1 + \ldots + w_nx_n$
   Weighted sum of inputs (linear combination).
2. Activation Function: f
   Non-linear component of a perceptron.

Linear Function

Linear Function of a Perceptron

$$z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

where the weights w and bias b are parameters of the perceptron.

This is just like the model in a linear regression case!

Activation Function

The linear function of a perceptron is fed into a non linear function, called the activation function f, yielding the output y.

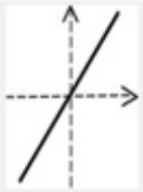
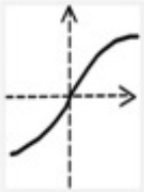
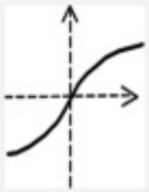
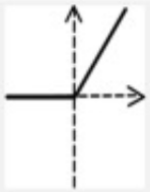
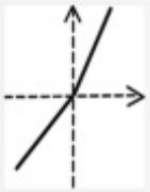
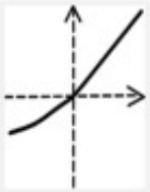
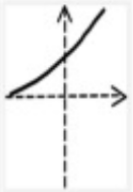
Activation Function	Identity	Binary Step	Logistic	TanH	ArcTan	ReLU	PreLU	ELU	SoftPlus
PLOT									
EQUATION	$f(x) = x$	$f(x) = \begin{cases} 0 & for\ x<0 \\ 1 & for\ x\geq 0 \end{cases}$	$f(x) = \frac{1}{1+e^{-x}}$	$f(x) = tangh(x) = \frac{2}{1+e^{-2x}} - 1$	$f(x) = tan^{-1}(x)$	$f(x) = \begin{cases} 0 & for\ x<0 \\ x & for\ x\geq 0 \end{cases}$	$f(x) = \begin{cases} \propto x & for\ x<0 \\ x & for\ x\geq 0 \end{cases}$	$f(x) = \begin{cases} \propto(e^{-x}-1) & for\ x<0 \\ x & for\ x\geq 0 \end{cases}$	$f(x) = log_e(1+e^x)$
DERIVATE	$f'(x) = 1$	$f'(x) = \begin{cases} 0 & for\ x\neq 0 \\ ? & for\ x=0 \end{cases}$	$f'(x) = f(x)(1-f(x))$	$f'(x) = 1-f(x)^2$	$f'(x) = \frac{1}{x^2+1}$	$f'(x) = \begin{cases} 0 & for\ x<0 \\ 1 & for\ x\geq 0 \end{cases}$	$f'(x) = \begin{cases} \propto & for\ x<0 \\ 1 & for\ x\geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x)+\propto & for\ x<0 \\ 1 & for\ x\geq 0 \end{cases}$	$f'(x) = \frac{1}{1+e^{-x}}$

Image from ScienceDirect

Activation Function

1.

Activation Function

- Sigmoid and tanh are less and less used these days partly because their are bounded from both sides and the gradient of them vanishes as z goes to both positive and negative infinity.
- ReLU is the most commonly used activation function because calculation of its derivative is efficient, yielding to a much faster training time.
- Softplus is not used very often either in practice and can be viewed as a smoothing of the ReLU so that it has a proper second order derivative.
- GELU and leaky ReLU are both variants of ReLU but they have some non-zero gradient even when the input is negative.

Activation Function

What about a linear activation function? Why can't we choose a linear activation function?

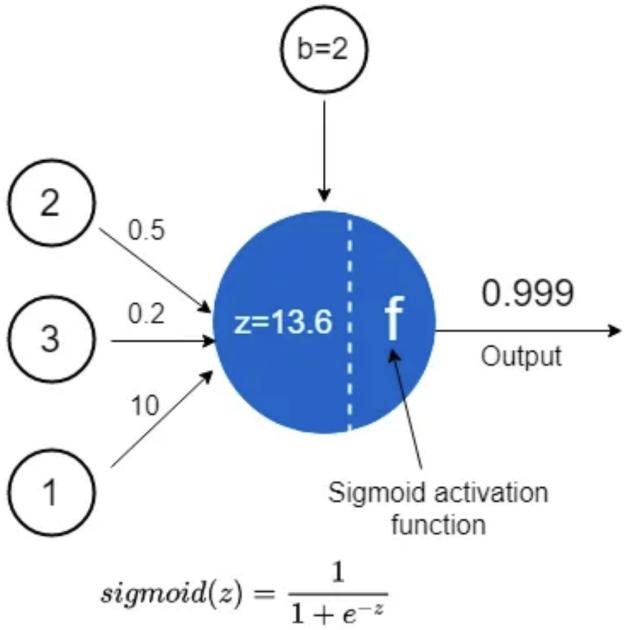
Activation Function

What about a linear activation function? Why can't we choose a linear activation function?

Applying a linear function to another linear function will result in a linear function over the original input. This loses much of the representational power of the neural network as often times the output we are trying to predict has a non-linear relationship with the inputs. Without non-linear activation functions, the neural network will simply perform linear regression.

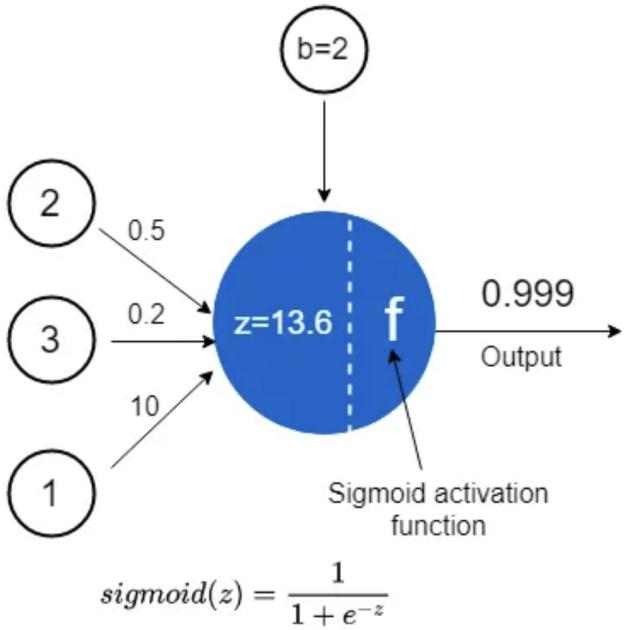
Forward-pass in a single neuron

Let's look at this example and solve the forward pass from input to output for this single neuron:



source

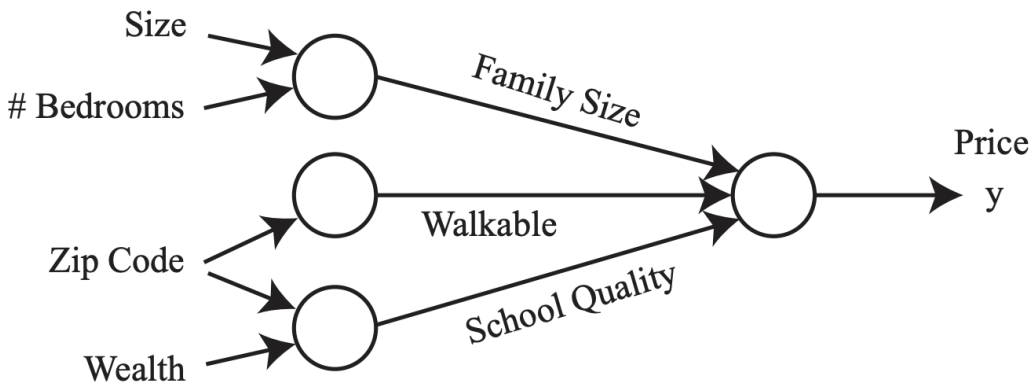
Forward-pass in a single neuron



Inputs: $x_1 = 2, x_2 = 3, x_3 = 1$
Weights: $w_1 = 0.5, w_2 = 0.2, w_3 = 10$
Bias: $b = 2$
Linear Function: $w_1x_1 + w_2x_2 + w_3x_3 + b = 13.6$
Activation Function: $f(z) = \frac{1}{1+e^{-z}}$
Output: $y = f(z) = 0.999$

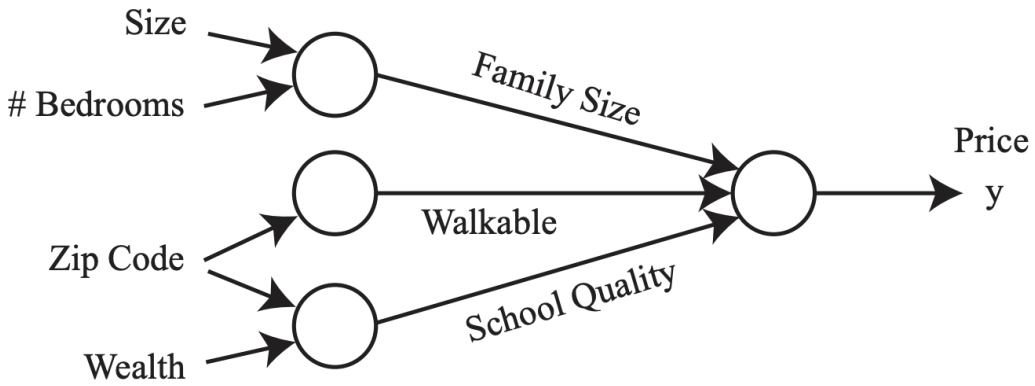
Two-layer Neural Network

We can stack neurons, taking the output of one neuron and passing it as input to another neuron, resulting in a more complex function.



source

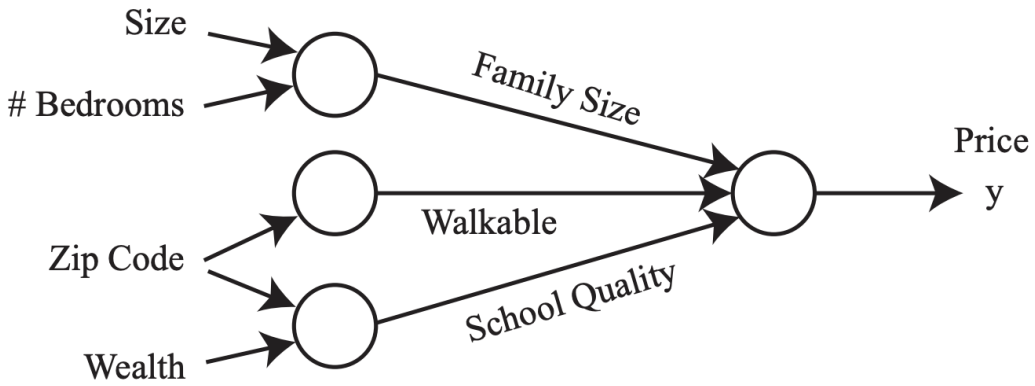
Forward-pass in a two-layer neural network



Regression problem:

1. **Input Features:** x_1, x_2, x_3, x_4 = Size, Bedrooms, Zip Code, Wealth

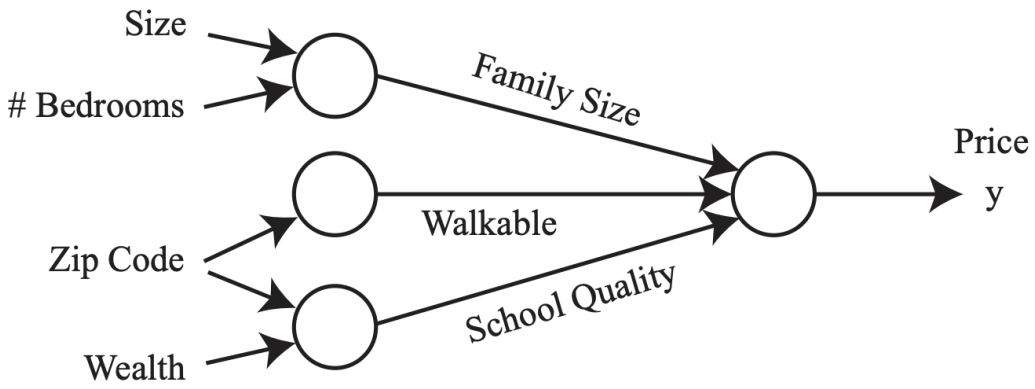
Forward-pass in a two-layer neural network



2. **First Layer (Hidden Layer):**
   Linear Function:

$$z_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)}$$
$$z_2^{(1)} = w_{23}^{(1)} x_3 + b_2^{(1)}$$
$$z_3^{(1)} = w_{33}^{(1)} x_3 + w_{34}^{(1)} x_4 + b_3^{(1)}$$

Forward-pass in a two-layer neural network



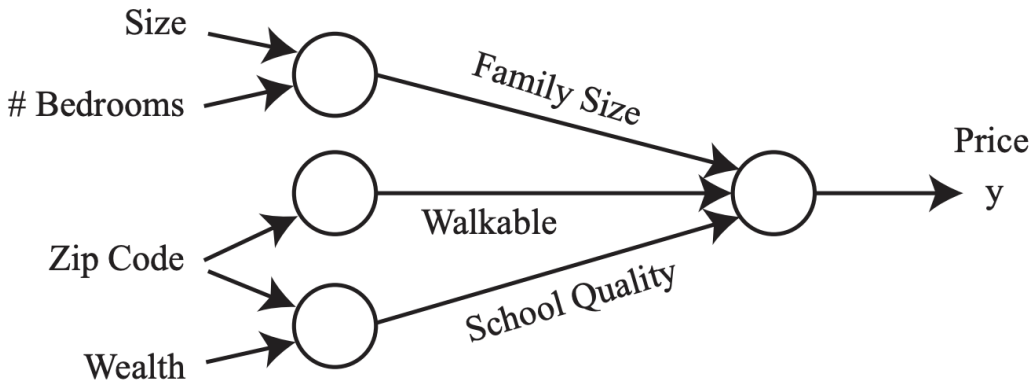
2. **First Layer (Hidden Layer):**
   Activation Function: a_1, a_2, a_3 = Family Size, Walkable, School Quality
   Assuming ReLU activation function:

$$a_1^{(1)} = \text{ReLU}(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)})$$

$$a_2^{(1)} = \text{ReLU}(w_{23}^{(1)} x_3 + b_2^{(1)})$$

$$a_3^{(1)} = \text{ReLU}(w_{33}^{(1)} x_3 + w_{34}^{(1)} x_4 + b_3^{(1)})$$

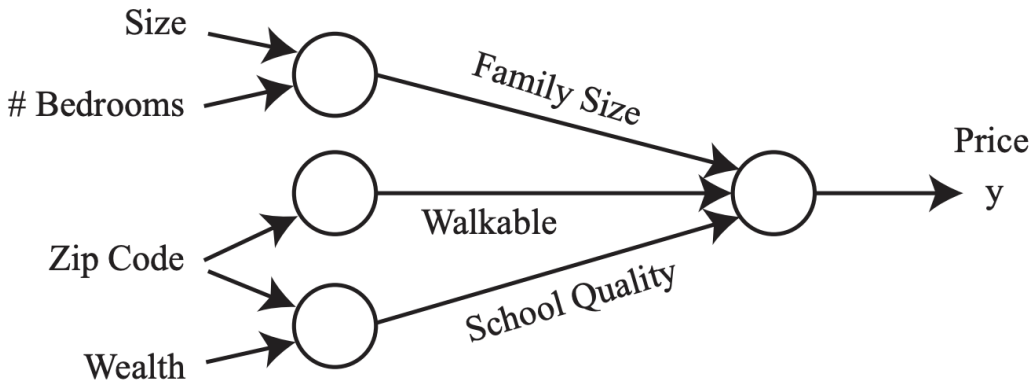
Forward-pass in a two-layer neural network



3. **Output Layer:**
   Linear Function:

$$z_1^{(2)} = w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + w_{13}^{(2)} a_3^{(1)} + b_1^{(2)}$$

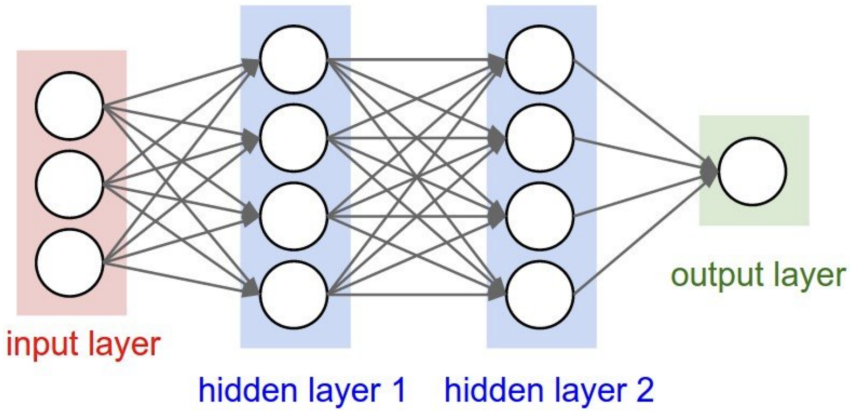
Forward-pass in a two-layer neural network



3. **Output Layer:**
   Activation Function:
   For regression tasks, the output layer doesn't have an activation function, or an identity activation function: $y = \text{Price}$

$$y = a_1^{(2)} = w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + w_{13}^{(2)} a_3^{(1)} + b_1^{(2)}$$

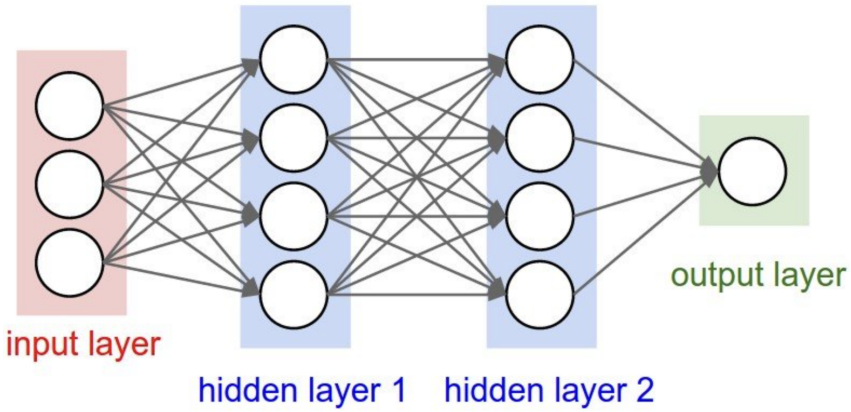
Multi-layer Neural Network



source

By expressing inputs x, weights w, hidden layers activation a, and outputs y as vectors, we can write down the equations for a forward-pass in a neural network.

Multi-layer Neural Network

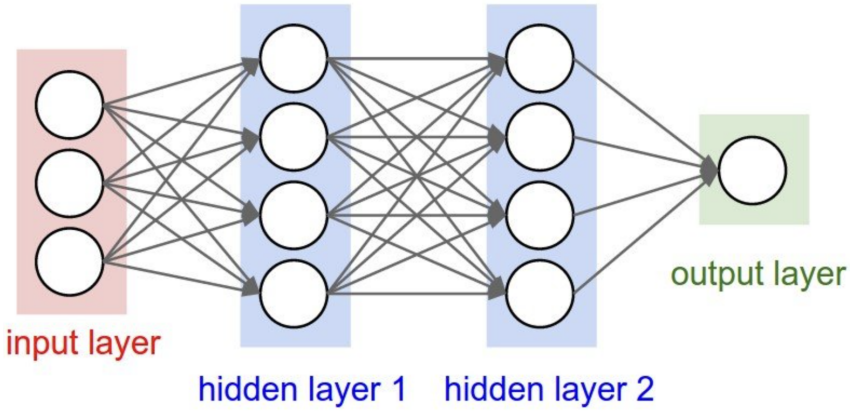


where:
$z_i^{(k)}$ = Output of unit i in layer k after going through the linear function prior to the activation function
$a_i^{(k)}$ = Output of the activation layer of unit i in layer k
$w_{ij}^{(k)}$ = Weight of connection from unit i in layer k to unit j in layer $k-1$

If in layer k there are m_k units of neurons, $a^{(k)} \in \mathbf{R}^{m_k}$, $w^{(k)} \in \mathbf{R}^{m_k \times m_{k-1}}$, $b^{(k)} \in \mathbf{R}^{m_k}$

Forward Pass

Summary



Goal: to calculate the output y prediction from inputs x based on the current values of weights w and biases b.

$$x \quad \text{(Input layer)}$$

$$z^{(1)} = w^{(1)} \cdot x + b^{(1)} \quad \text{(Hidden Layer 1, linear function)}$$

$$a^{(1)} = f(z^{(1)}) = f(w^{(1)} \cdot x + b^{(1)}) \quad \text{(Hidden Layer 1, activation function)}$$

$$z^{(2)} = w^{(2)} \cdot a^{(1)} + b^{(2)} \quad \text{(Hidden Layer 2, linear function)}$$

$$a^{(2)} = f(z^{(2)}) = f(w^{(2)} \cdot a^{(1)} + b^{(2)}) \quad \text{(Hidden Layer 2, activation function)}$$

and so on

Training a Neural Network

So far we have written the **"forward pass"** of the neural network: writing relationships between *input* features x and *output* y through a set of *parameters* w and b.

Now, we need to **"train"** the neural network: getting the optimal values of *parameters* weights w and biases b that minimizes some cost function (made prediction $\hat{y}$ as close to the real output y).

Training a Neural Network

We will use the Gradient Descent approach to update the weights and biases:

$$w^{(k)} := w^{(k)} + \Delta w^{(k)}$$

$$\Delta w^{(k)} = \alpha \frac{\partial J(w)}{\partial w^{(k)}}$$

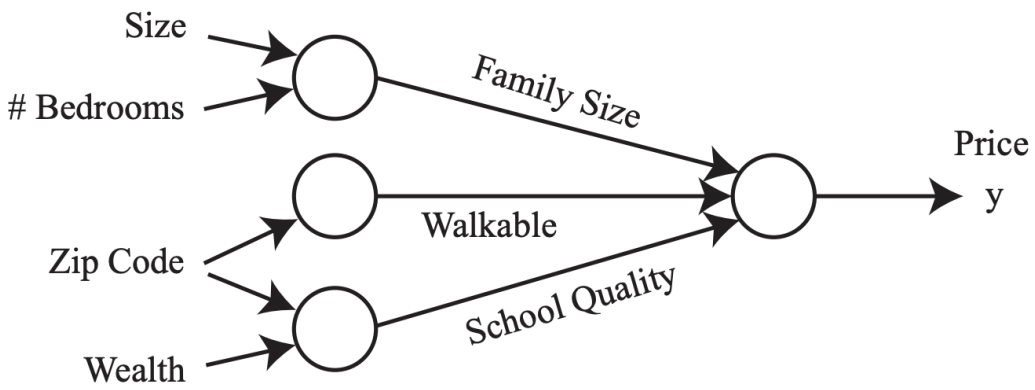
where $\Delta w^{(k)}$ is the tiny amount of update for the weights, and is physically characterized as the "step" walking along the gradient, scaled by the learning rate α.

The gradient term is called **"backpropagation"**, in which the error (in terms of cost function $J(w)$ is backpropagated through the layers.

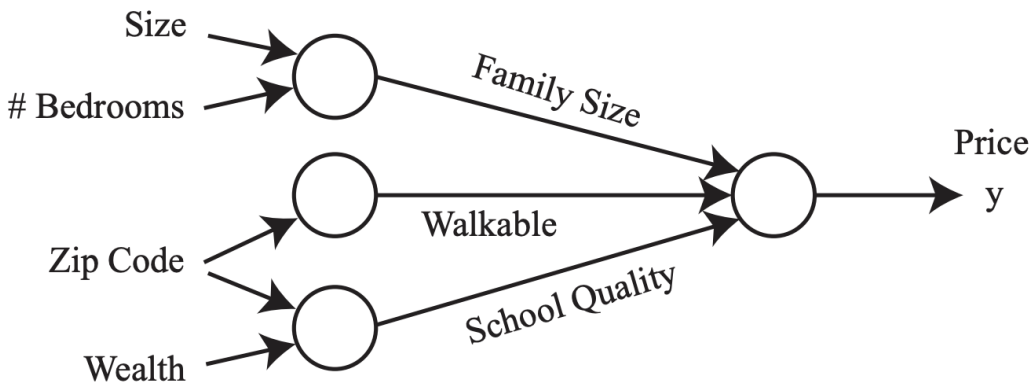
Backward pass in a two-layer neural network

Now that we've written the forward-pass of a two-layer neural network, we can derive the backward-pass in which the error is backpropagated for each neuron with respect to each weights. The proportion of these error is then added to the weights as update for that iteration.

Let's use the same example that we were working on earlier:



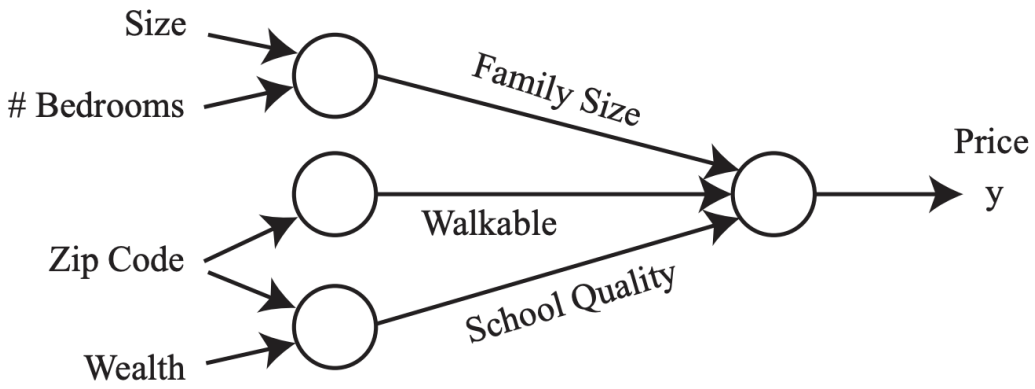
Backward pass in a two-layer neural network



1. **Cost Function:** $J(w)$. For regression tasks, the cost function can be MAE or MSE. For classification tasks, the cost function is binary cross-entropy for binary classification and categorical cross-entropy for multi-class classification. Let's use MSE for this task.

$$J(w) = (\hat{y} - y)^2$$

Backward pass in a two-layer neural network



2. **Weight Gradient:** For each weights and biases, calculate the gradient of the cost function with respect to each of them, $\frac{\partial J(w)}{\partial w_{ij}^{(k)}}$.

   Final layer:

$$\frac{\partial J(w)}{\partial w_{11}^{(2)}}, \frac{\partial J(w)}{\partial w_{12}^{(2)}}, \frac{\partial J(w)}{\partial w_{13}^{(2)}}, \frac{\partial J(w)}{\partial b_{1}^{(2)}}$$

   Hidden layer:

$$\frac{\partial J(w)}{\partial w_{11}^{(1)}}, \frac{\partial J(w)}{\partial w_{12}^{(1)}}, \frac{\partial J(w)}{\partial b_{1}^{(1)}}, \frac{\partial J(w)}{\partial w_{23}^{(1)}}, \frac{\partial J(w)}{\partial b_{2}^{(1)}}, \frac{\partial J(w)}{\partial w_{33}^{(1)}}, \frac{\partial J(w)}{\partial w_{34}^{(1)}}, \frac{\partial J(w)}{\partial b_{3}^{(1)}}$$

Backward pass in a two-layer neural network

Let's solve the weight gradient using chain rule:

Final Layer

1. $\frac{\partial J(w)}{\partial w_{11}^{(2)}}$

$$\frac{\partial J(w)}{\partial w_{11}^{(2)}} = \frac{\partial J(w)}{\partial a_1^{(2)}} \times \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \times \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}} \tag{1}$$

$$\frac{\partial J(w)}{\partial a_1^{(2)}} = \frac{\partial (a_1^{(2)} - y)^2}{\partial a_1^{(2)}} = 2(a_1^{(2)} - y) \tag{2}$$

$$\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = \frac{\partial z_1^{(2)}}{\partial z_1^{(2)}} = 1 \tag{3}$$

$$\frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}} = \frac{\partial (w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + w_{13}^{(2)} a_3^{(1)} + b_1^{(2)})}{\partial w_{11}^{(2)}} = a_1^{(1)} \tag{4}$$

Backward pass in a two-layer neural network

Plugging Equations 2, 3, and 4 into Equation 1:

$$\frac{\partial J(w)}{\partial w_{11}^{(2)}} = 2(a_1^{(2)} - y) \times 1 \times a_1^{(1)}$$

$$= 2a_1^{(1)}(a_1^{(2)} - y)$$

2. $\frac{\partial J(w)}{\partial w_{12}^{(2)}}$. Repeating the same process above, we get:

$$\frac{\partial J(w)}{\partial w_{12}^{(2)}} = 2a_2^{(1)}(a_1^{(2)} - y)$$

2. $\frac{\partial J(w)}{\partial w_{13}^{(2)}}$. Repeating the same process above, we get:

$$\frac{\partial J(w)}{\partial w_{13}^{(2)}} = 2a_3^{(1)}(a_1^{(2)} - y)$$

Backward pass in a two-layer neural network

Hidden Layer

1. $\frac{\partial J(w)}{\partial w_{11}^{(1)}}$

$$\frac{\partial J(w)}{\partial w_{11}^{(1)}} = \frac{\partial J(w)}{\partial a_1^{(2)}} \times \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \times \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} \times \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} \times \frac{\partial z_1^{(1)}}{\partial w_{11}^{(1)}} \tag{5}$$

$$\frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} = \frac{\partial (w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + w_{13}^{(2)} a_3^{(1)} + b_1^{(2)})}{\partial a_1^{(1)}} = w_{11}^{(2)} \tag{6}$$

$$\frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} = \frac{\partial \text{ReLU}(z_1^{(1)})}{\partial z_1^{(1)}} = 1 \quad \text{if} \quad z_1^{(1)} > 0 \tag{7}$$

$$\frac{\partial z_1^{(1)}}{\partial w_{11}^{(1)}} = \frac{\partial (w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + b_1^{(1)})}{\partial w_{11}^{(1)}} = x_1 \tag{8}$$

Backward pass in a two-layer neural network

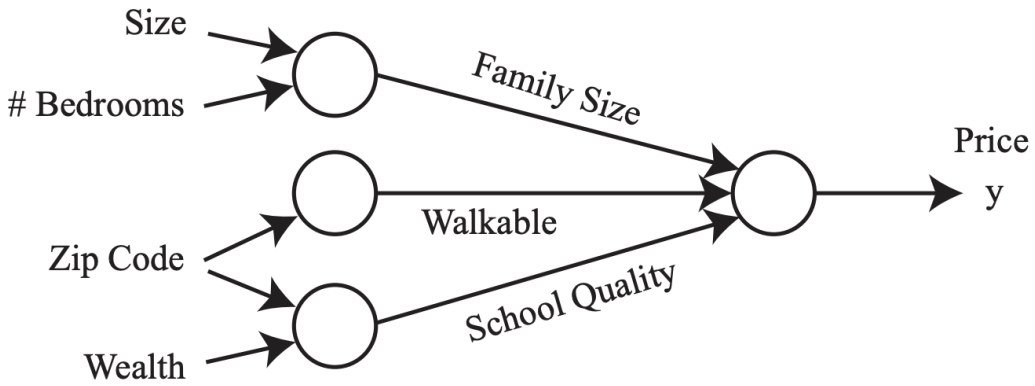
Plugging Equations 2, 3, 6, 7, and 8 into Equation 5:

$$\frac{\partial J(w)}{\partial w_{11}^{(1)}} = 2(a_1^{(2)} - y) \times 1 \times w_{11}^{(2)} \times 1 \times x_1$$

$$= 2w_{11}^{(2)} x_1 (a_1^{(2)} - y)$$

and so on... (try calculating the rest yourself)

Backward pass in a two-layer neural network



3. **Weight Update:** Update each weights $w_{ij}^{(k)}$ at layer k between neuron i at layer k and neuron j at layer $k-1$ using the update rule.

$$w_{ij}^{(k)} := w_{ij}^{(k)} + \alpha \frac{\partial J(w)}{\partial w_{ij}^{(k)}}$$

where α is the learning rate.

Backward Pass or Backpropagation

Summary

Goal: to update each weights and biases in the network so that the predicted output is closer to the actual output.

$$J(w) = \quad \text{MAE, MSE, or cross-entropy (Cost Function)}$$

$$\frac{\partial J(w)}{\partial w_{ij}^{(k)}} \quad \text{(Weight Gradient for each weights and biases)}$$

$$w_{ij}^{(k)} := w_{ij}^{(k)} + \alpha \frac{\partial J(w)}{\partial w_{ij}^{(k)}} \quad \text{(Weight Update)}$$

Training a Neural Network

Now that we have done **"backpropagation"** and update all the weights, we will iteratively repeat this process until the weight converges to some optimal values.

Number of Iterations. For the case of Stochastic Gradient Descent (SGD), where the gradient is calculated for each training sample, we will repeat the process for n iterations, where n is the number of training samples.

Number of Epochs. Once all the n training samples have undergone the forward-pass and backward-pass process, we call that one **"epoch"**. This means, for 1 epoch, there are n iterations. We will repeat the process for a certain number of epoch until convergence.

Note: In Batch Gradient Descent (BGD) where all the training samples are fed through at once as matrix multiplication through the forward and backward pass, we only have 1 iteration per 1 epoch.

Summary

For the case of Stochastic Gradient Descent (SGD), where the weights are updated for one training sample at a time:

Training a Neural Network

```
for a in range(epoch):
 for b in range(training samples):
```

1. **Forward Pass**. For each layer k of the neural network, compute:
   1. **Linear Function**: $z^{(k)} = w^{(k)} \cdot a^{(k-1)} + b^{(k)}$
   2. **Activation Function**: $a^{(k)} = f(z^{(k)})$
   3. (At the final layer) **Cost Function**: $J(w)$. This can be either an MSE for regression or a logistic function (cross-entropy) for classification.
2. **Backpropagation**.
3. **Weight Update**. For each weight at layer k between neuron i at layer k and neuron j at layer $k-1$:
   1. $w_{ij}^{(k)} := w_{ij}^{(k)} + \alpha \frac{\partial J(w)}{\partial w_{ij}^{(k)}}$