CSE7850/CX4803 Machine Learning in Computational Biology

Lecture 7: Neural Networks

Yunan Luo

Acknowledgements: Some of the slides were adapted from lectures of Matt Gormley and Volodymyr Kuleshov

Recap: “Recipe” of (supervised) machine learning

- **Dataset**

$$\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid i = 1, 2, \ldots, n\}$$

- **Model**

$$\hat{y} = f_\theta(x)$$

- **Optimization**
  - Fit the model parameters on the dataset

- **Predictions**
  - Apply the model to predict for new data samples

$$\hat{y}_{\text{new}} = f_\theta(x_{\text{new}})$$

Recap: “Recipe” of (supervised) machine learning

- **Dataset**

$$\mathcal{D} = \{(x^{(i)}, y^{(i)}) \mid i = 1, 2, \ldots, n\}$$

- **Model**

$$\boxed{\hat{y} = f_\theta(x)}$$

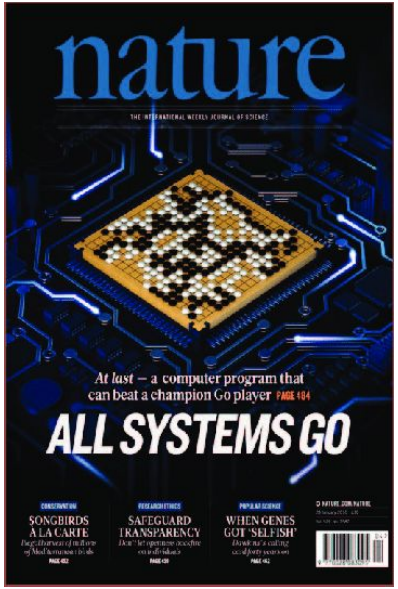
- **Optimization**
  - Fit the model parameters on the dataset

- **Predictions**
  - Apply the model to predict for new data samples

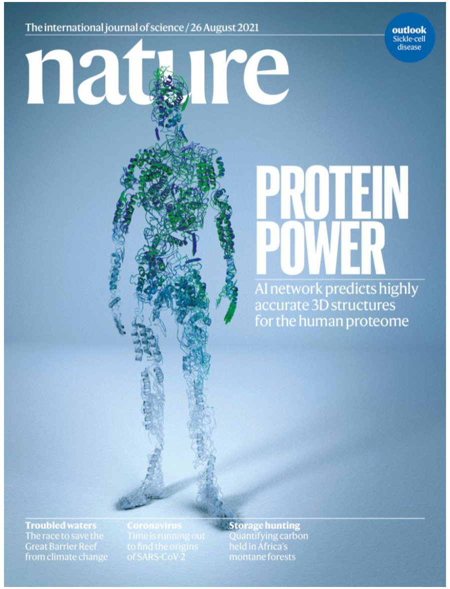
$$\hat{y}_{\text{new}} = f_\theta(x_{\text{new}})$$

- So far: linear regression, logistic regression, SVM, decision tree
- Today: a new class of models (Neural Networks)

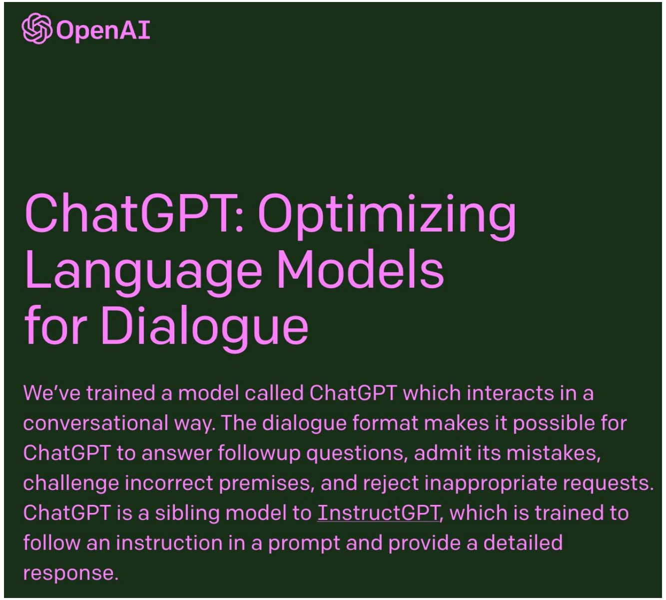
Neural networks are building blocks for complex AI algorithms



Alpha Go (2016)

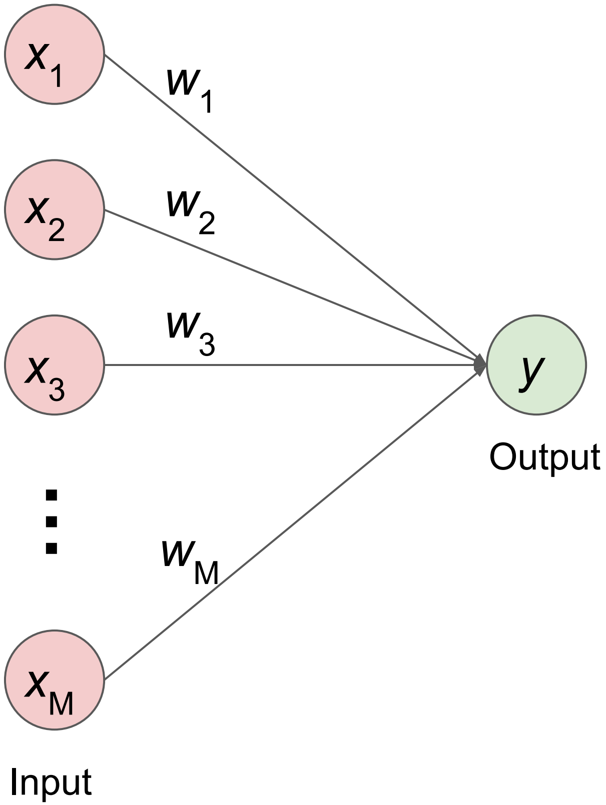


Alpha Fold (2018-2021)



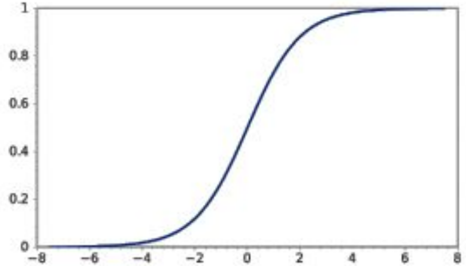
ChatGPT (2022)

Logistic regression

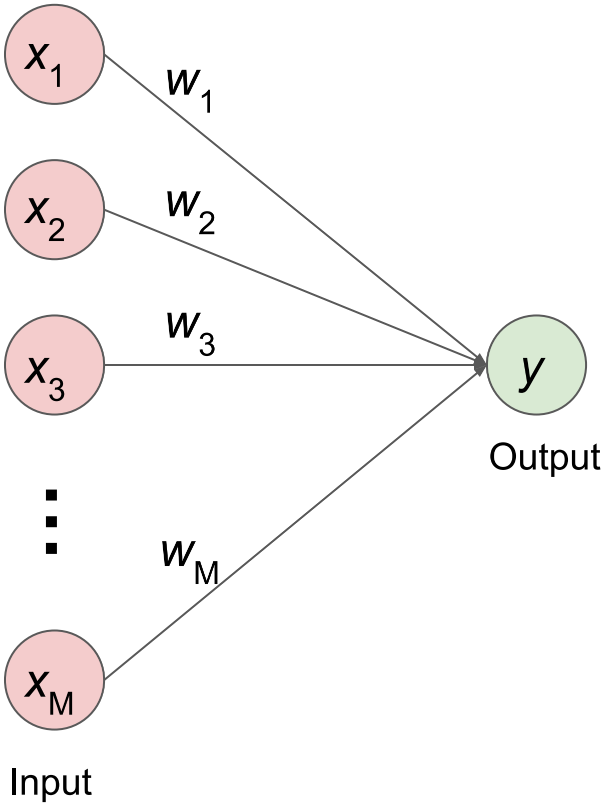


$$y = f_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$$

$$\text{where } \sigma(a) = \frac{1}{1 + \exp(-a)}$$



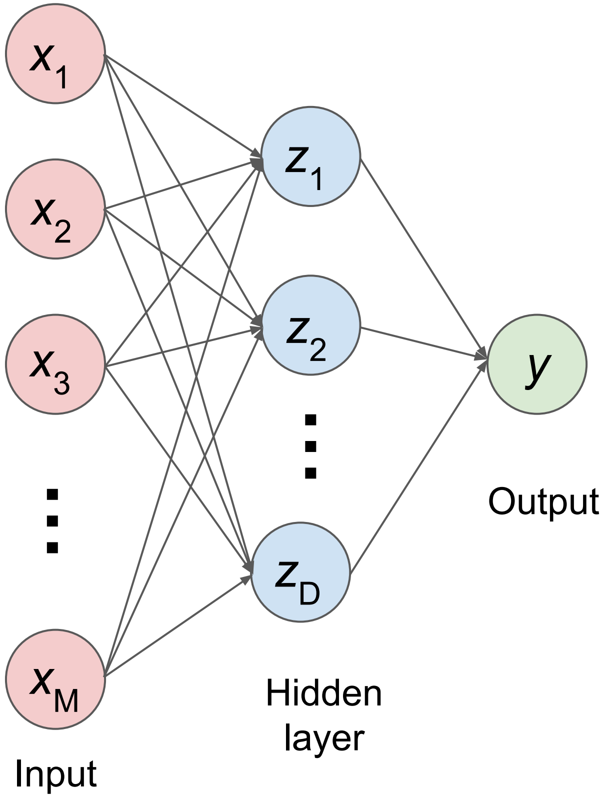
Linear regression



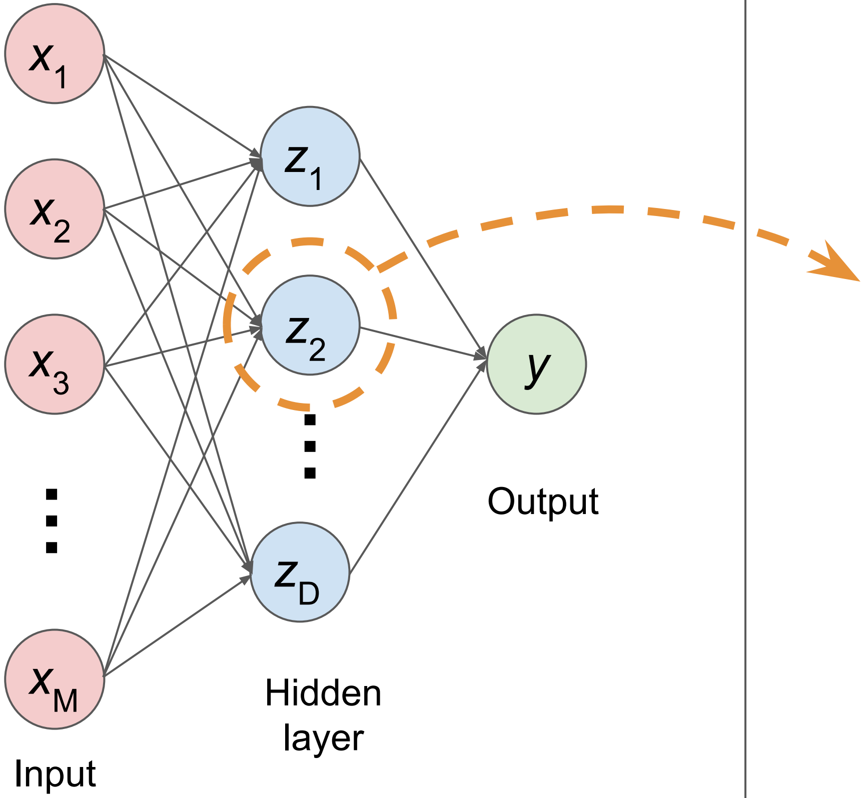
$$y = f_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$$

where $\sigma(a) = a$

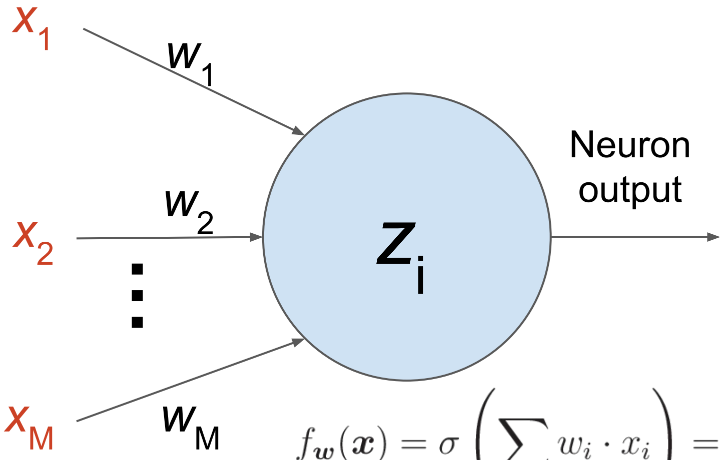
Neural network



Neural network



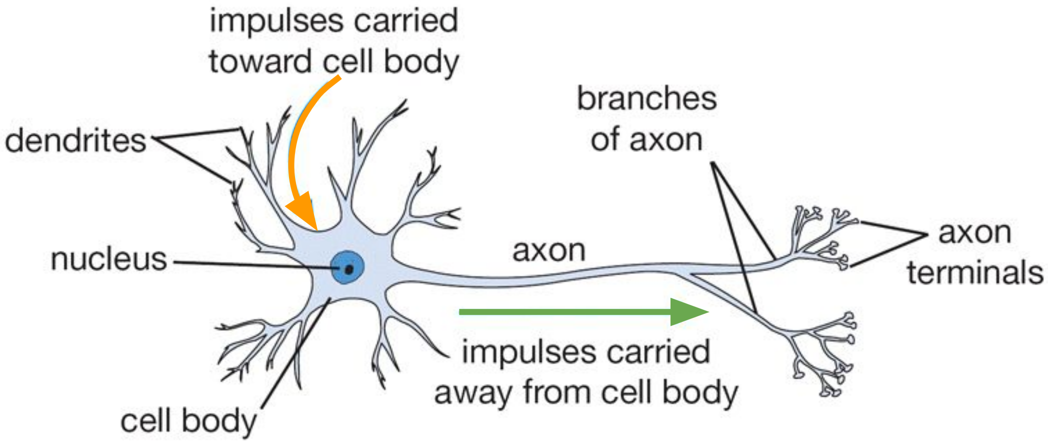
(Artificial) Neuron



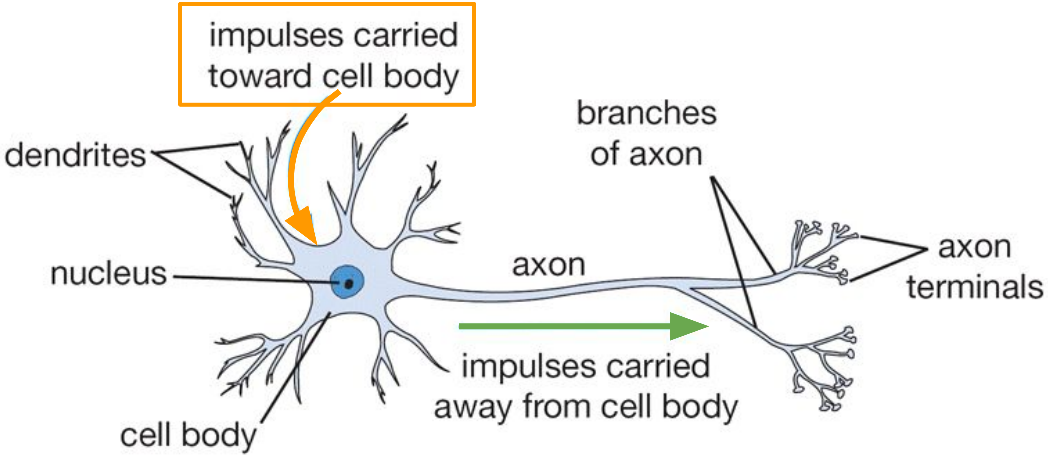
$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma\left(\sum_i w_i \cdot x_i\right) = \sigma(\boldsymbol{w}^T\boldsymbol{x})$$

A neuron computes an output using the inputs and a set of functions

Inspired by biological neural networks

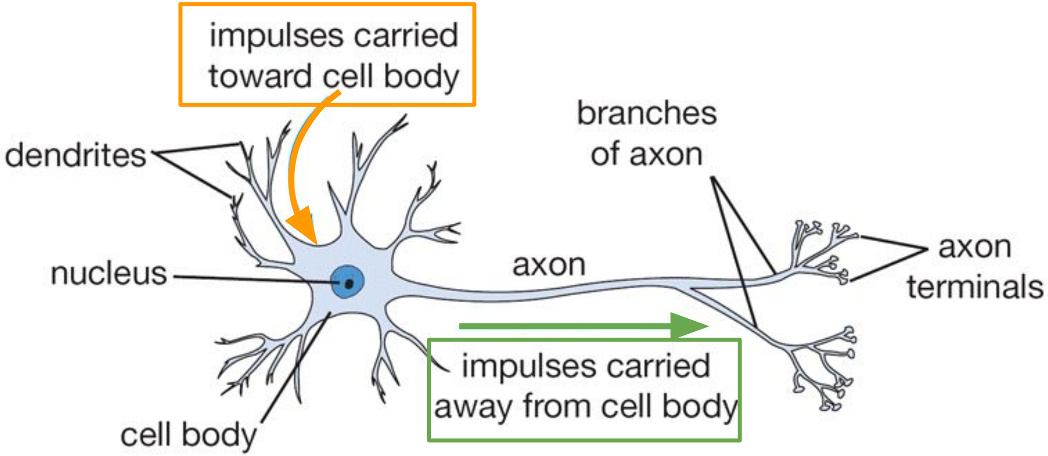


Inspired by biological neural networks



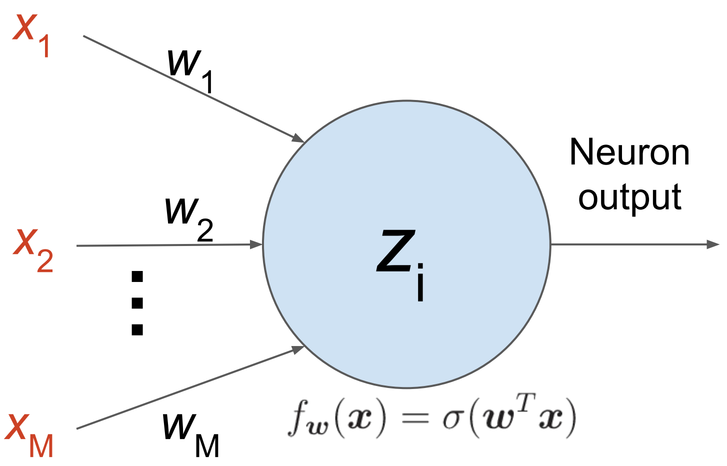
- Each neuron receives **input signals** from its dendrites

Inspired by biological neural networks



- Each neuron receives **input signals** from its dendrites
- If input signals are ***strong enough***, neuron **fires output** along its axon, which connects to the dendrites of other neurons.

Computation in an Artificial Neuron



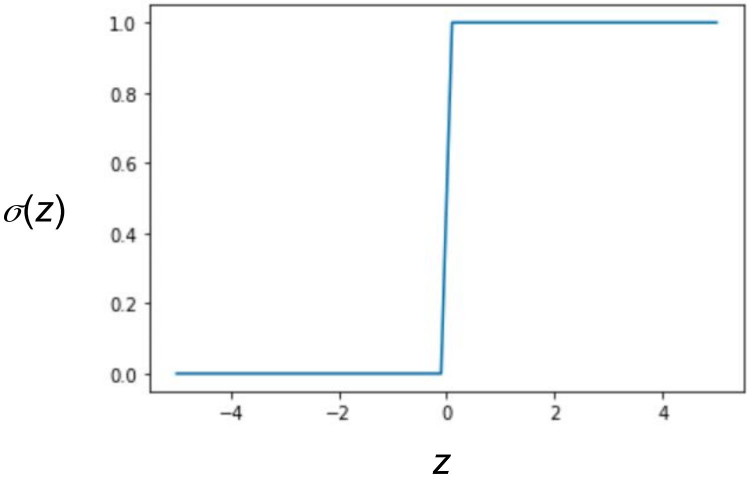
More formally, we say that a neuron is a model $f : \mathbb{R}^d \to [0, 1]$, with the following components:

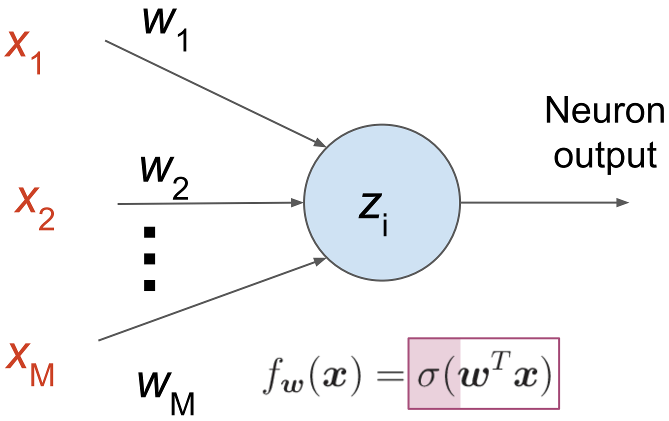
- Inputs $x_1, x_2, \ldots, x_d$, denoted by a vector x.
- Weight vector $w \in \mathbb{R}^d$ that modulates input x as $w^\top x$.
- An activation function $\sigma : \mathbb{R} \to \mathbb{R}$ that computes the output $\sigma(w^\top x)$ of the neuron based on the sum of modulated features $w^\top x$.

Activation function

Example 1: Step function

$$\sigma(z) = \begin{cases} 1 & z > 0 \\ 0 & \text{otherwise} \end{cases}$$





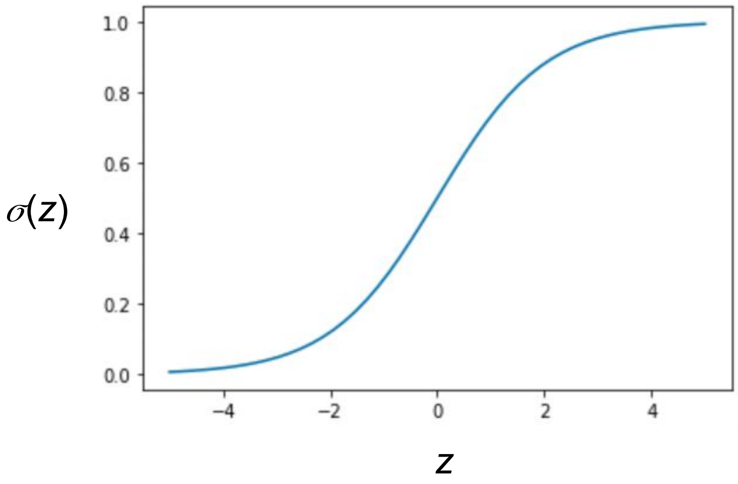
$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boxed{\sigma(\boldsymbol{w}^T \boldsymbol{x})}$$

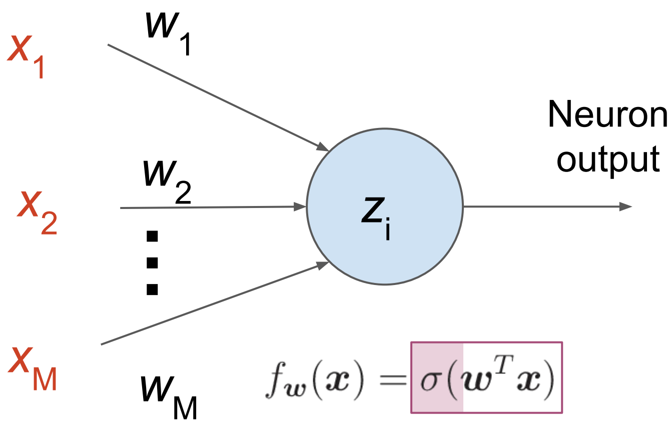
Recall in bio neurons: If input signals are ***strong enough***, neuron fires an output

Activation function

Example 2: Logistic function

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



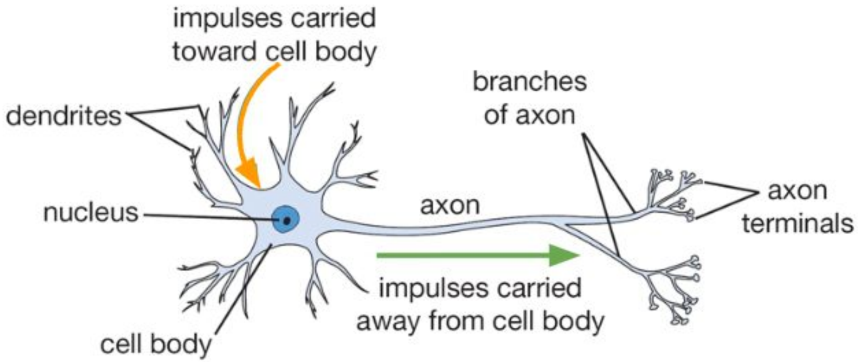


$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boxed{\sigma(\boldsymbol{w}^T\boldsymbol{x})}$$

Recall in bio neurons: If input signals are ***strong enough***, neuron fires an output

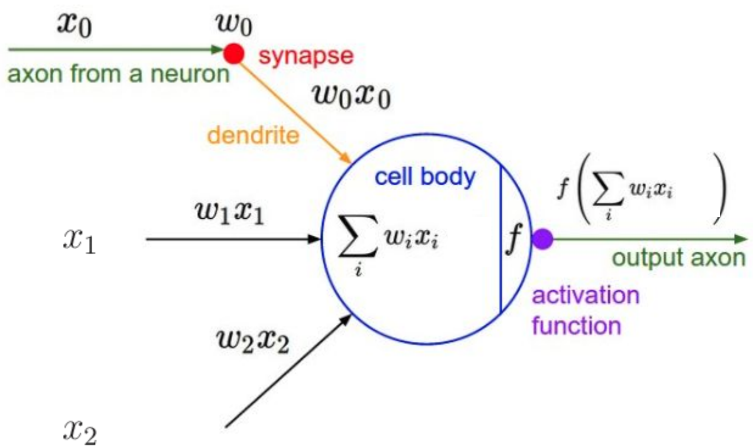
Biological and Artificial Neurons

Biological neuron



- Each neuron receives input signals from its dendrites
- If input signals are strong enough, neuron fires output along its axon, which connects to the dendrites of other neurons.

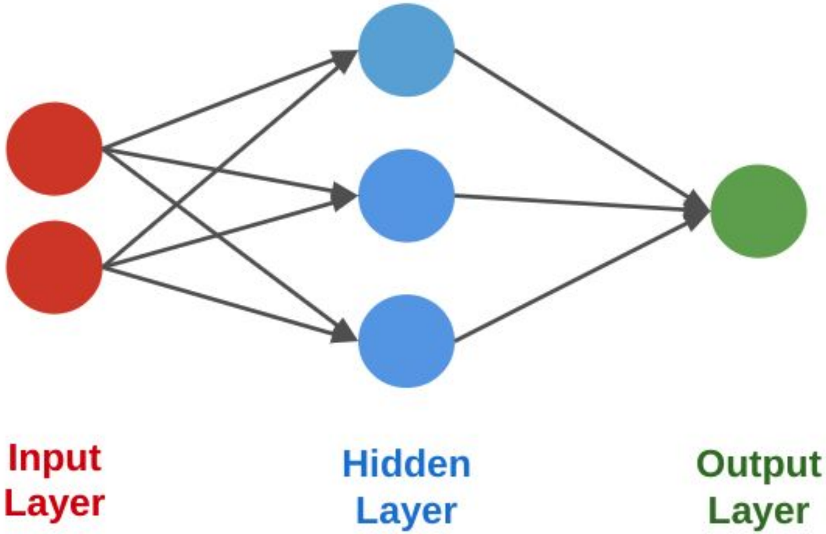
Artificial neuron



- Dendrite j gets signal x_j; modulates multiplicatively to $w_j \cdot x_j$.
- The body of the neuron sums the modulated inputs: $\sum_{j=1}^{d} w_j \cdot x_j$.
- These go into the activation function that produces an ouput.

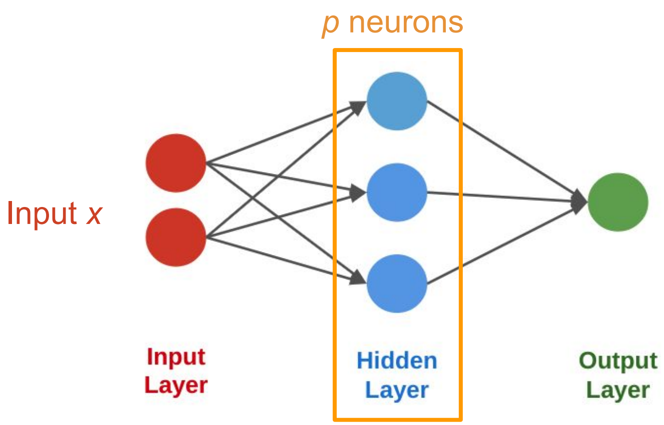
Neural Networks: Intuition

- A neural network is a directed graph in which a node is a neuron that takes as input the outputs of the neurons that are connected to it.



- Networks are typically organized in layers.

Neural Networks: Layers

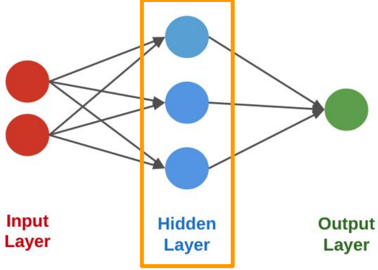


A neural network layer is a model $f : \mathbb{R}^d \to \mathbb{R}^p$ that applies p neurons in parallel to an input x.

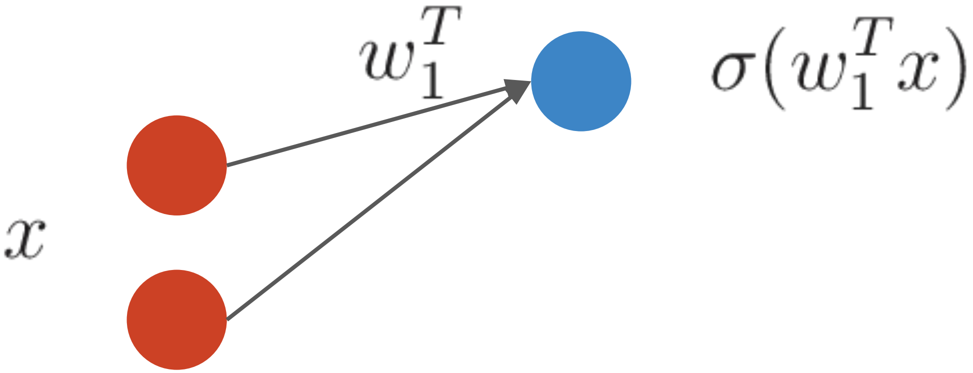
$$f(x) = \begin{bmatrix} \sigma(w_1^\top x) \\ \sigma(w_2^\top x) \\ \vdots \\ \sigma(w_p^\top x) \end{bmatrix}.$$

where each w_k is the vector of weights for the k-th neuron. We refer to p as the *size* of the layer.

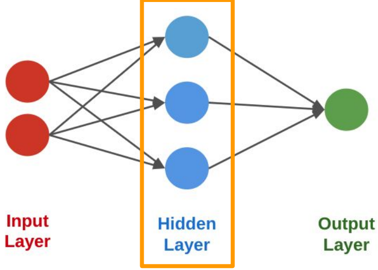
Neural Networks: Layers



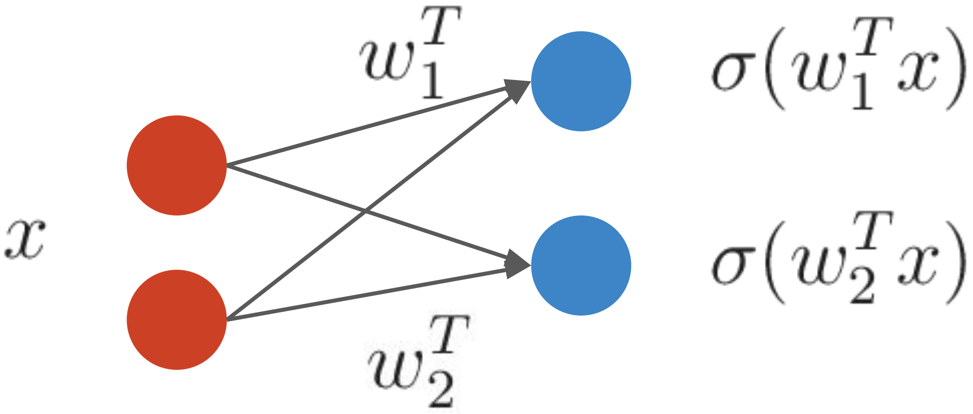
- The first output of the layer is a neuron with weights w_1



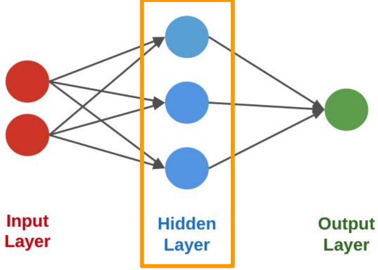
Neural Networks: Layers



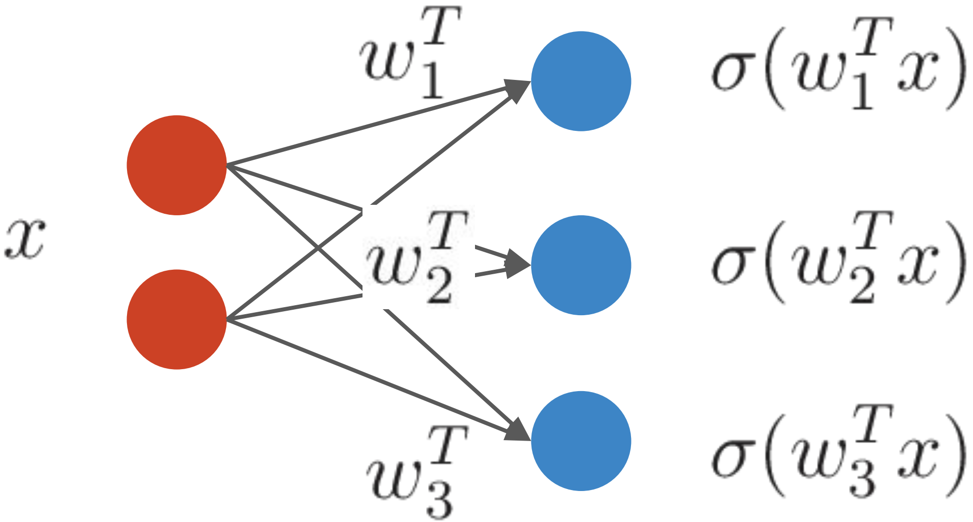
- The second neuron has weights w_2



Neural Networks: Layers



- The second neuron has weights w_3



- The parameters of the layer are w_1, w_2, and w_3

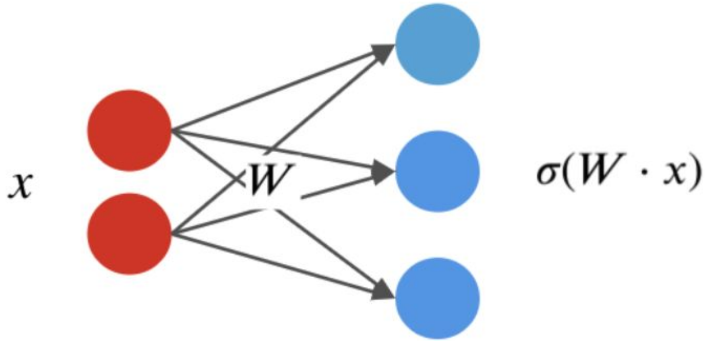
Neural Networks: Layers

By combining the w_k into one matrix W, we can write in a more succinct vectorized form:

$$f(x) = \sigma(W \cdot x) = \begin{bmatrix} \sigma(w_1^\top x) \\ \sigma(w_2^\top x) \\ \vdots \\ \sigma(w_p^\top x) \end{bmatrix},$$

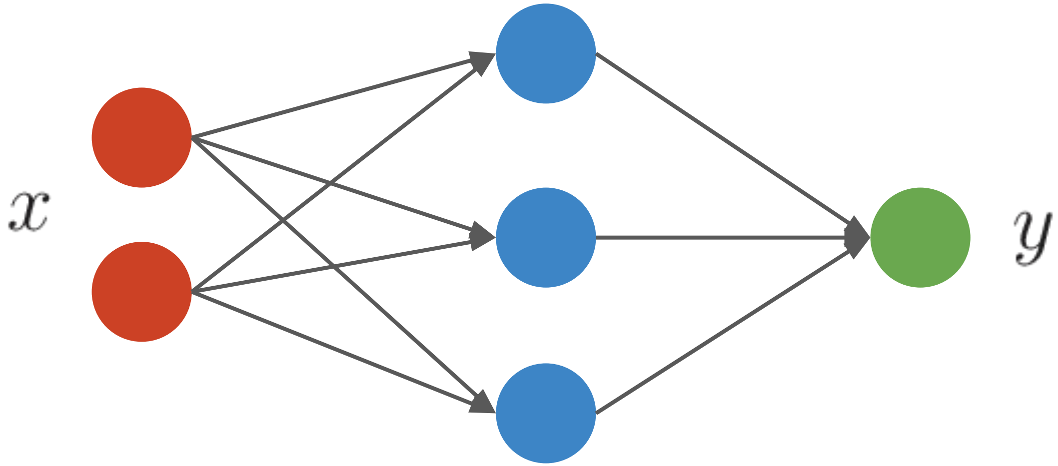
where $\sigma(W \cdot x)_k = \sigma(w_k^\top x)$ and $W_{kj} = (w_k)_j$.

Visually, we can represent this as follows:



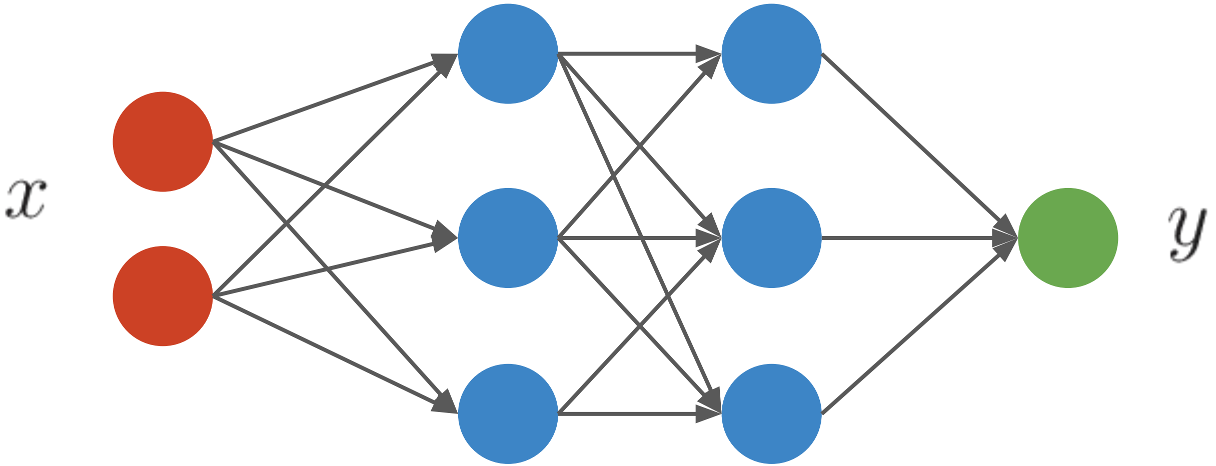
A neural network can have multiple hidden layers

- Two layers (one hidden layer + one output layer):



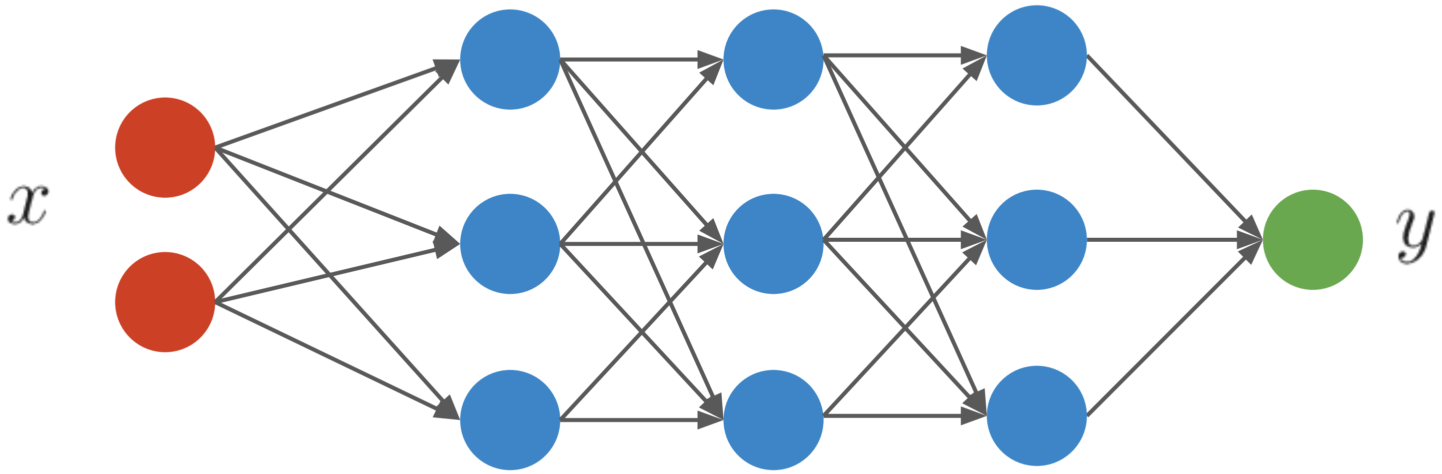
A neural network can have multiple hidden layers

- Three layers (two hidden layers + one output layer):



A neural network can have multiple hidden layers

- Four layers (three hidden layers + one output layer):



Neural Networks: Notation

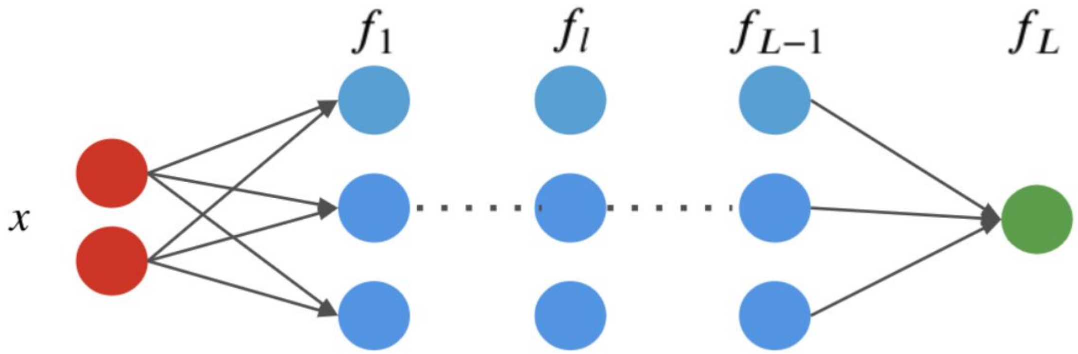
A neural network is a model $f : \mathbb{R}^d \to \mathbb{R}$ that consists of a composition of L neural network layers:

$$f(x) = f_L \circ f_{L-1} \circ \ldots f_l \circ \ldots f_1(x).$$

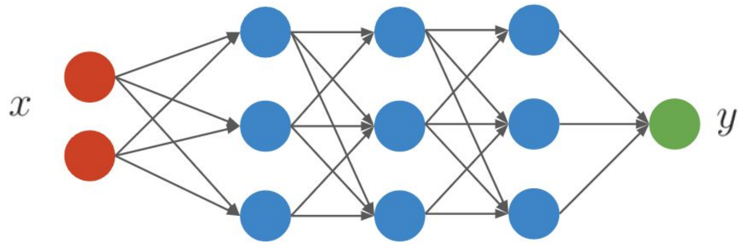
The final layer f_L has size one (assuming the neural net has one ouput); intermediary layers f_l can have any number of neurons.

The notation $f \circ g(x)$ denotes the composition $f(g(x))$ of functions.

We can visualize this graphically as follows.



Terminology

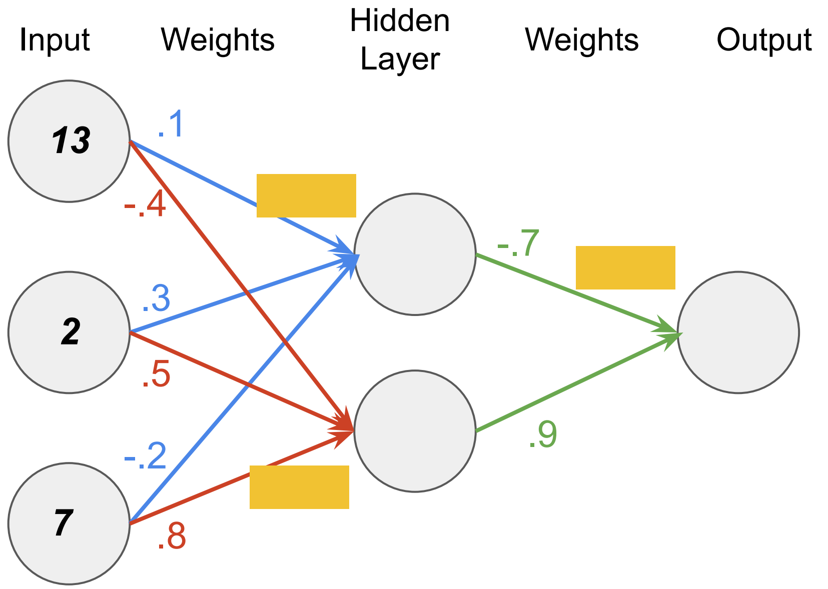


- Neural network is a broad concept, but this type of neural network is also known as
  - Fully connected neural networks
  - Multi-layer perceptron (MLP)
- The layers of MLP are often called “linear layers” or “dense layers” in popular deep learning packages
- We will cover other types of “neural networks” in this course, which all use MLP layer as building blocks:
  - Convolutional neural networks (CNN), Recurrent neural networks (RNN), Graph neural networks (GNN), Transformers

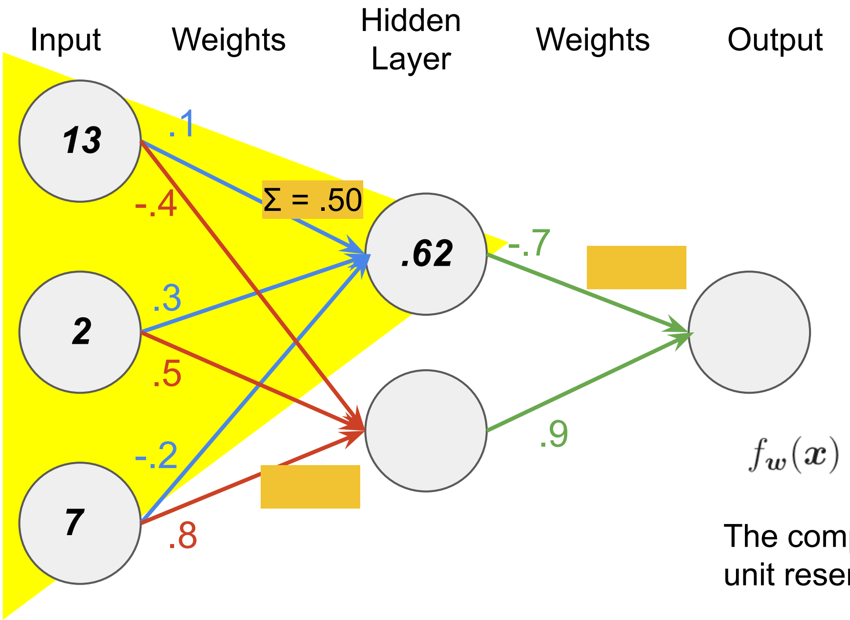
Example of Computation in Neural Networks

Suppose we **already learned the weights** of the neural network with sigmoid activation function.

To make a new prediction, we take in some new features (aka. the input layer) and perform the feed-forward computation.



Example of Computation in Neural Networks

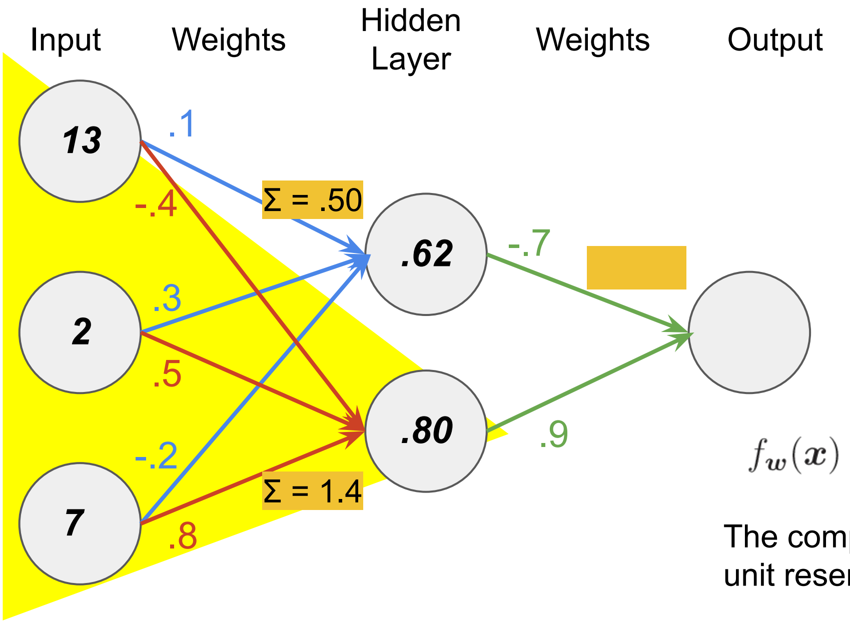


$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma\left(\sum_i w_i \cdot x_i\right) = \sigma(\boldsymbol{w}^T\boldsymbol{x})$$

The computation of each neural network unit resembles binary logistic regression.

.50 = 13(.1) + 2(.3) + 7(-.2)
.62 = σ(.50)

Example of Computation in Neural Networks

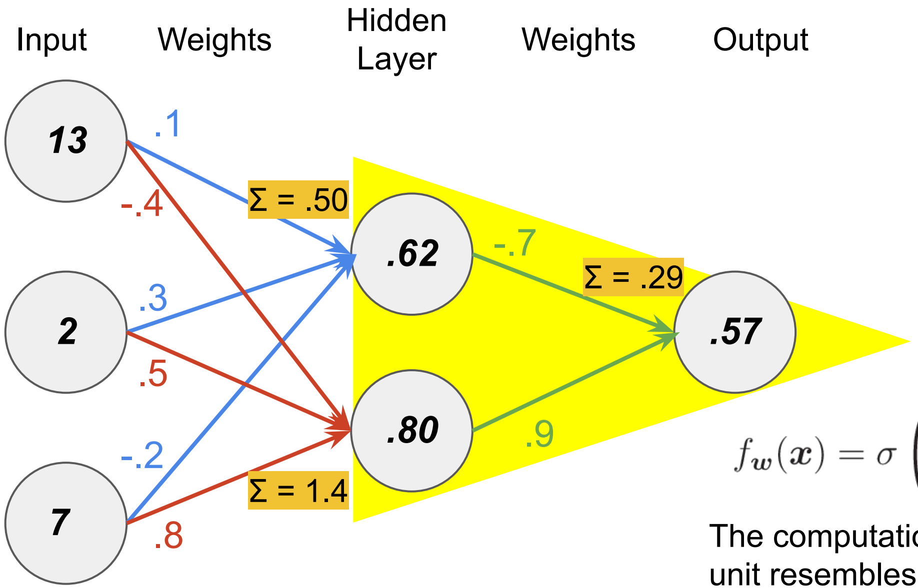


$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma\left(\sum_i w_i \cdot x_i\right) = \sigma(\boldsymbol{w}^T\boldsymbol{x})$$

The computation of each neural network unit resembles binary logistic regression.

1.4 = 13(-.4) + 2(.5) + 7(.8)
.80 = σ(1.4)

Example of Computation in Neural Networks

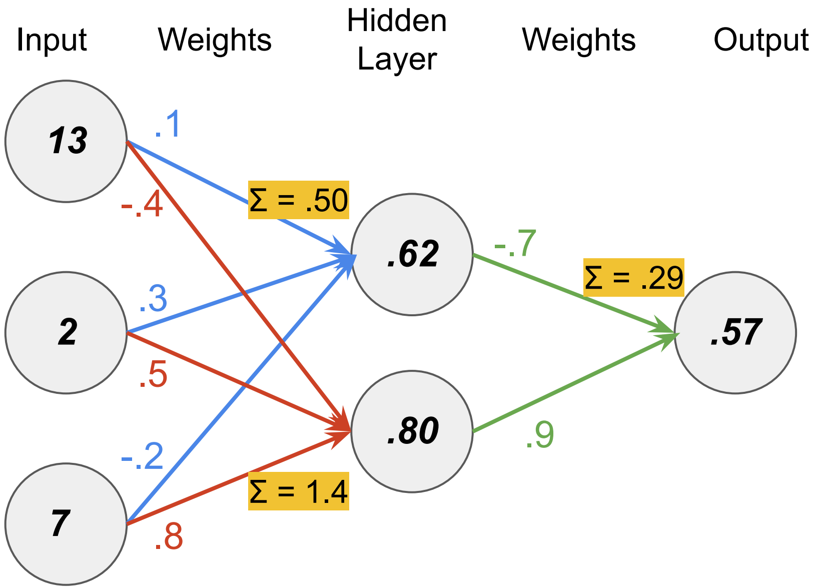


$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \sigma\left(\sum_i w_i \cdot x_i\right) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$$

The computation of each neural network unit resembles binary logistic regression.

.29 = .62(-.7) + .80(.9)
.57 = σ(.29)

Example of Computation in Neural Networks

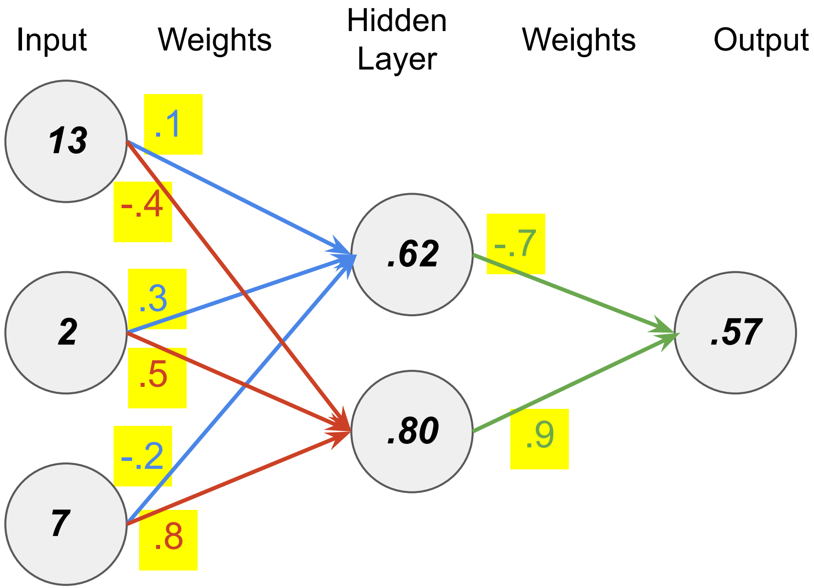


$.50 = 13(.1) + 2(.3) + 7(-.2)$
$.62 = \sigma(.50)$

$1.4 = 13(-.4) + 2(.5) + 7(.8)$
$.80 = \sigma(1.4)$

$.29 = .62(-.7) + .80(.9)$
$.57 = \sigma(.29)$

Example of Computation in Neural Networks



- In this example, we assumed that we already learned the weights of the neural network.
- In practice, we have to learn to assign “useful” values to those weights such that the output value is close to the desired target value

Design Space of Neural Networks

Design decisions to make for building an NN

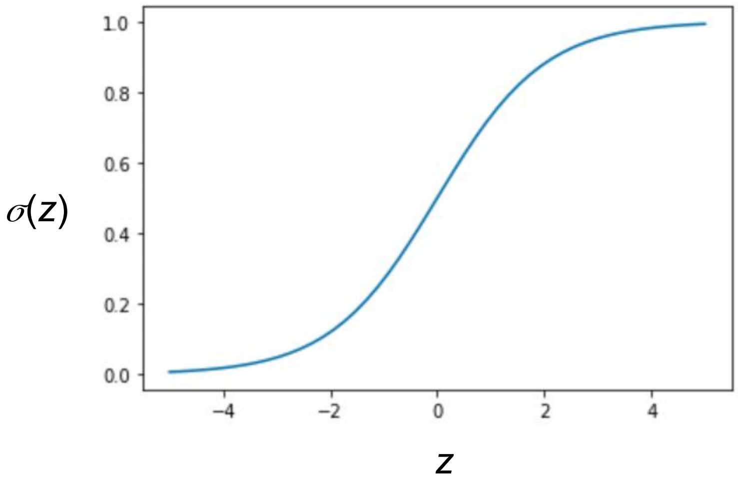
- # of hidden layers (depth)
- # of units per hidden layer (width)
- Type of activation function (nonlinearity)
- Form of objective function
- How to initialize the parameters

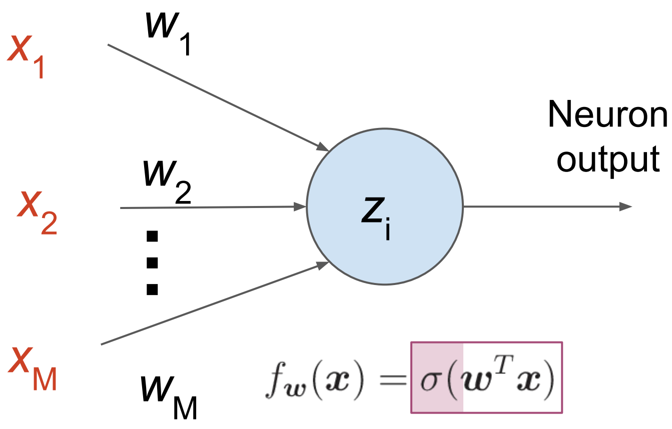
Activation functions

Activation function

Logistic function

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$





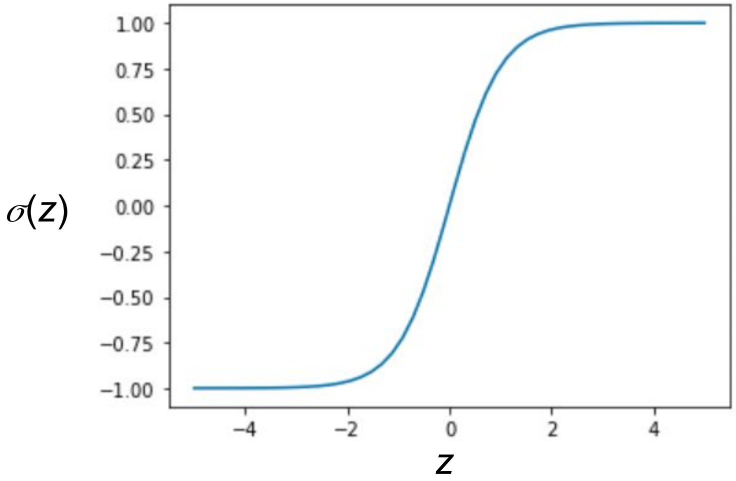
$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boxed{\sigma(\boldsymbol{w}^T \boldsymbol{x})}$$

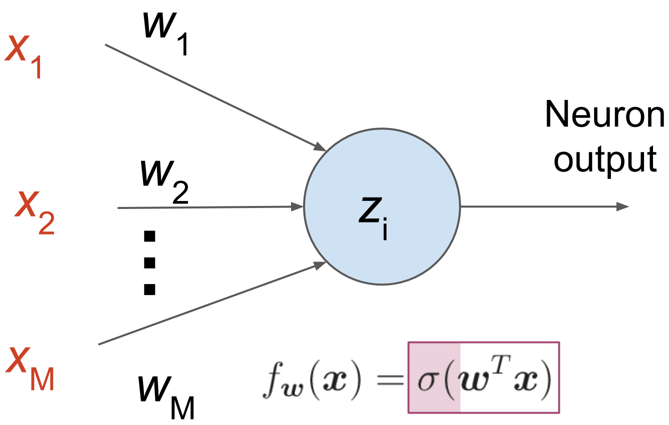
Recall in bio neurons: If input signals are ***strong enough***, neuron fires an output

Activation function

Hyperbolic tangent function (tanh)

$$\sigma(z) = \tanh(z)$$





$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boxed{\sigma(\boldsymbol{w}^T\boldsymbol{x})}$$

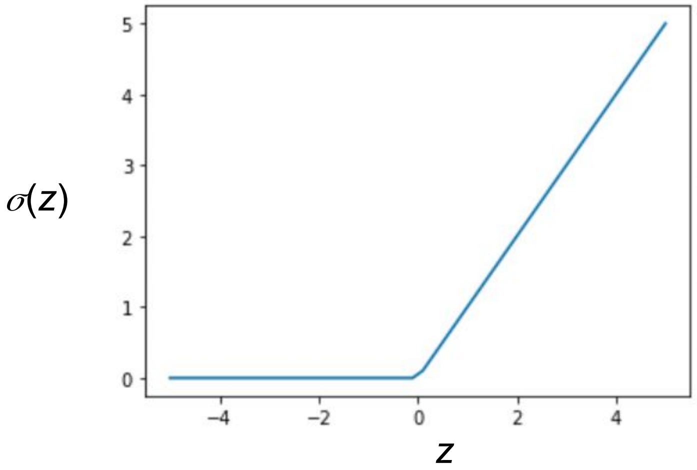
Recall in bio neurons: If input signals are ***strong enough***, neuron fires an output

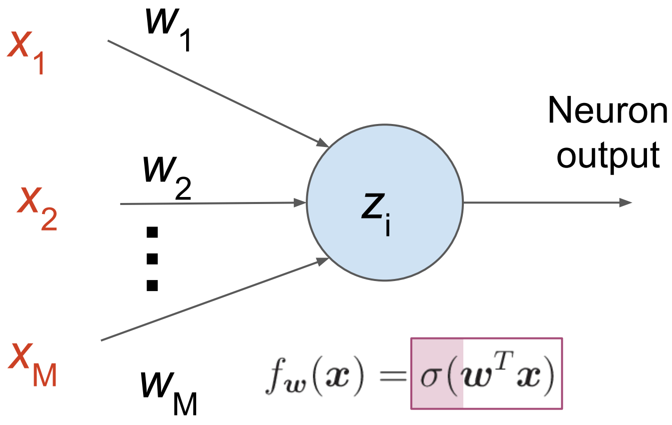
Activation function

Rectified linear unit (ReLU)

$$\sigma(z) = \max(0, z)$$

* ReLU and its variants are widely used in modern deep neural networks



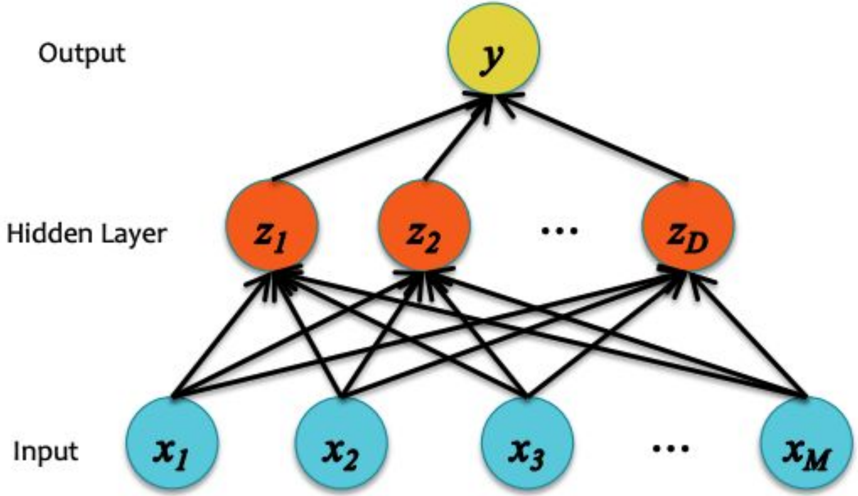


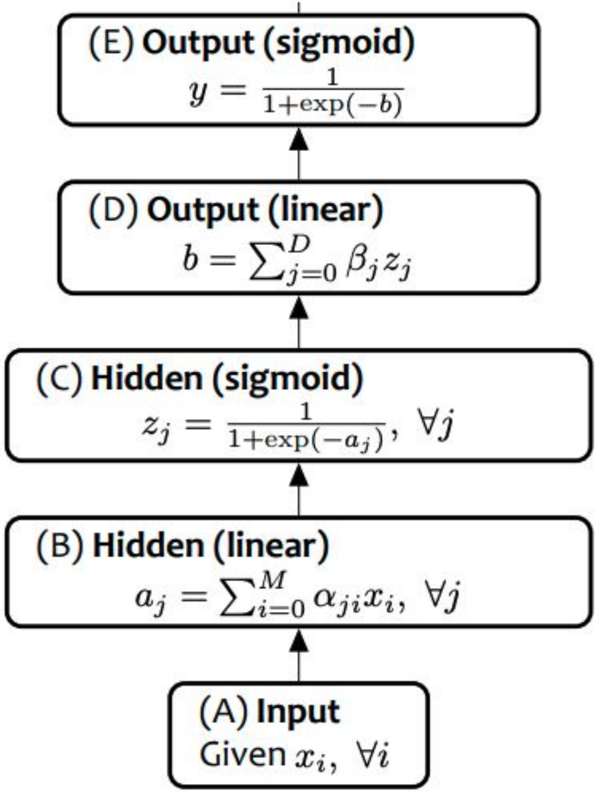
$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boxed{\sigma(\boldsymbol{w}^T\boldsymbol{x})}$$

Recall in bio neurons: If input signals are ***strong enough***, neuron fires an output

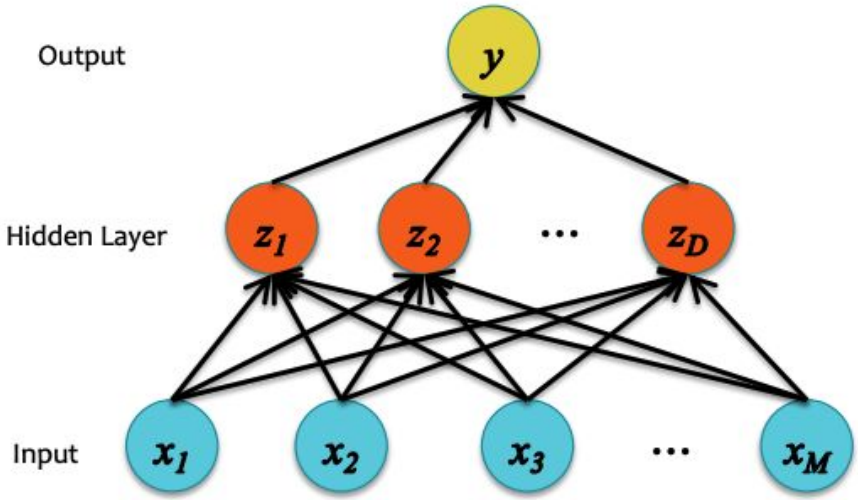
Loss functions & Output layers

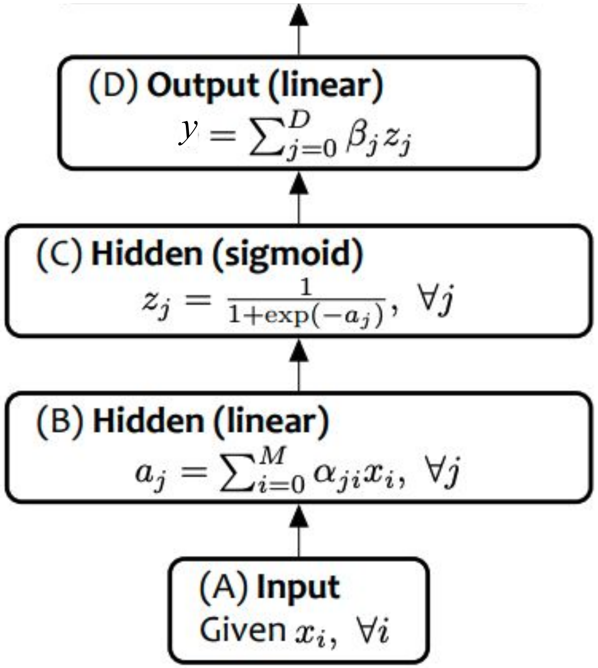
Neural Network for Classification





Neural Network for Regression





Objective functions

Regression loss:

- We can use the same objective as Linear Regression
- i.e., mean squared error (MSE)

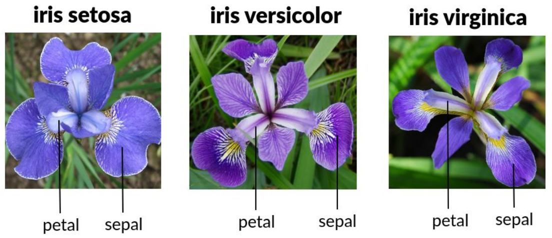
$$J = \ell_Q(y, y^{(i)}) = \frac{1}{2}(y - y^{(i)})^2$$

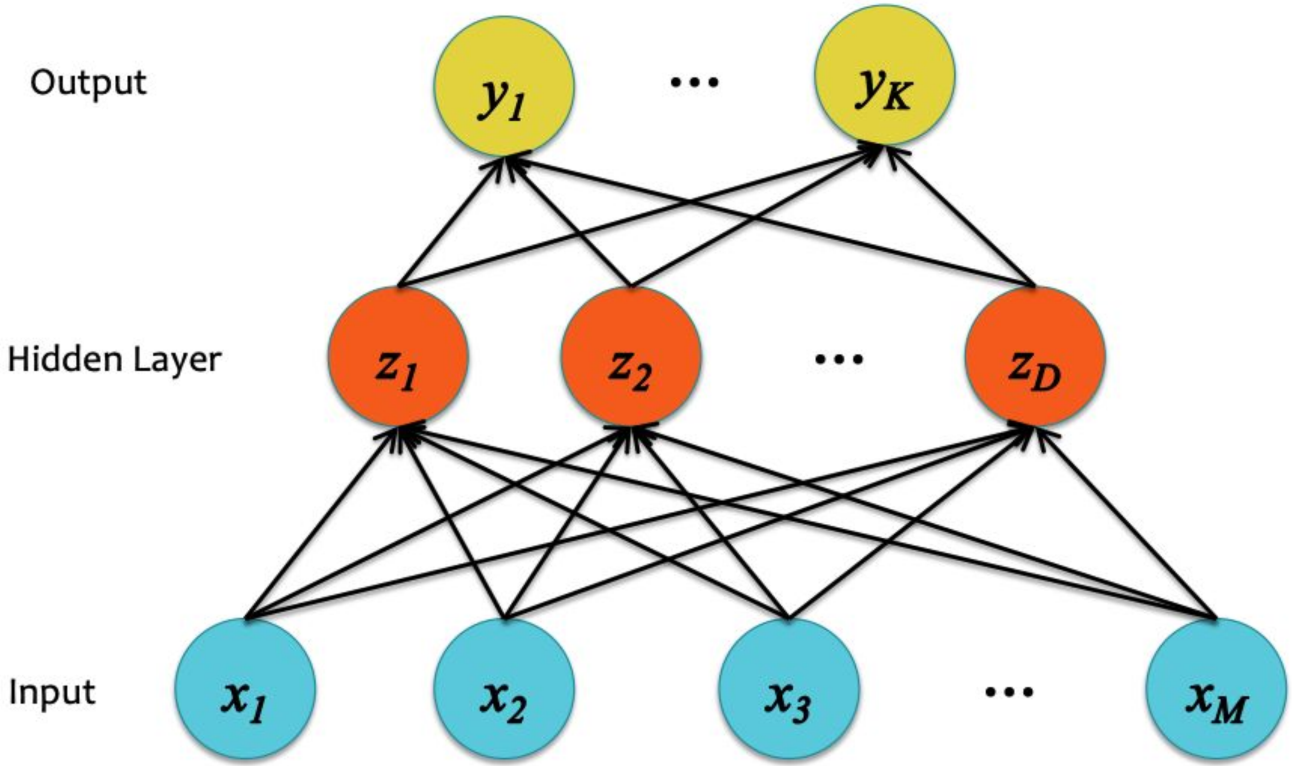
(Binary) Classification loss:

- We can use the same objective as Binary Logistic Regression
- a.k.a. Binary Cross-Entropy loss
- This requires our output y to be a probability in [0,1]

$$J = \ell_{CE}(y, y^{(i)}) = -(y^{(i)} \log(y) + (1 - y^{(i)}) \log(1 - y))$$

Multi-class classification

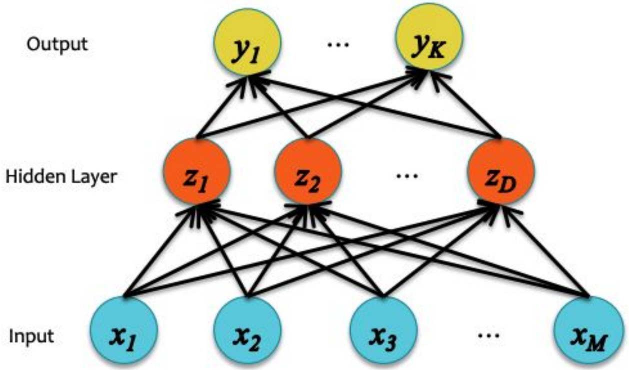


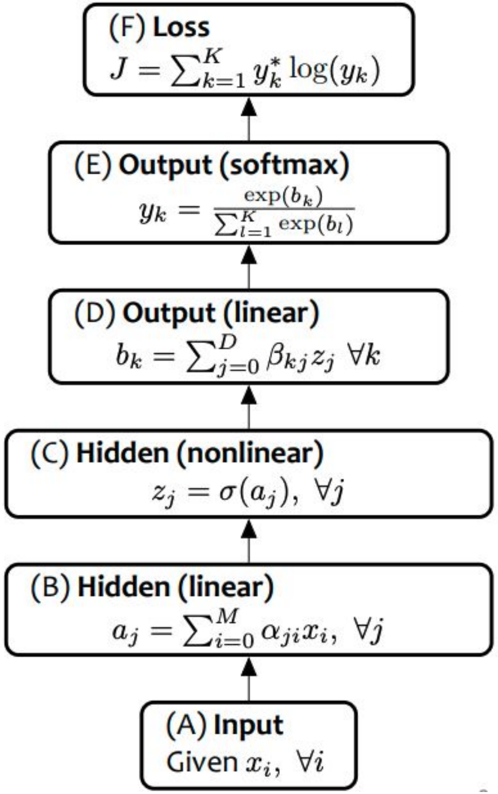


Multi-class classification

Softmax:

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$$





Objective functions

(Multi-class) Classification loss:

- Let $\mathbf{y}^{(i)}$ represent our true label as a one-hot vector:

$\mathbf{y}^{(i)} =$

1	2	3	4	5	6	...	K
0	0	0	1	0	0	...	0

- Assume our model outputs a length *K* vector of probabilities:

$$y = f_\theta(x)_k = \sigma(\theta_k^\top x)_k = \frac{\exp(\theta_k^\top x)}{\sum_{l=1}^{K} \exp(\theta_l^\top x)},$$

- The loss of a single training sample ($\mathbf{x}^{(i)}$, $\mathbf{y}^{(i)}$):

$$J = \ell_{CE}(\mathbf{y}, \mathbf{y}^{(i)}) = -\sum_{k=1}^{K} y_k^{(i)} \log(y_k)$$

How to optimize an NN?

Backpropagation

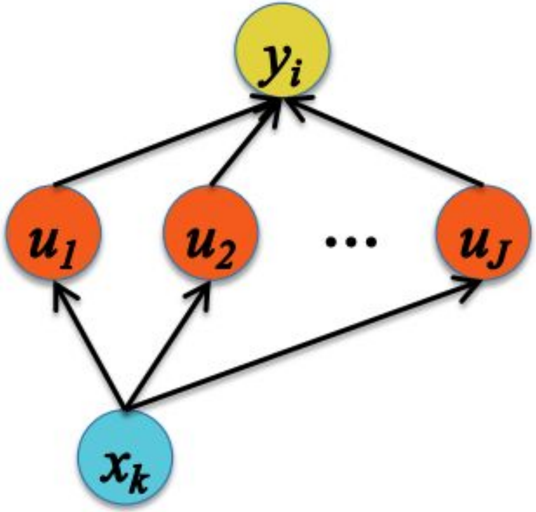
Calculus review: Chain Rule

Given:

$$\boldsymbol{y} = g(\boldsymbol{u}) \quad \text{and} \quad \boldsymbol{u} = h(\boldsymbol{x})$$

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j}\frac{du_j}{dx_k} \qquad \forall i, k$$



Calculus review: Chain Rule

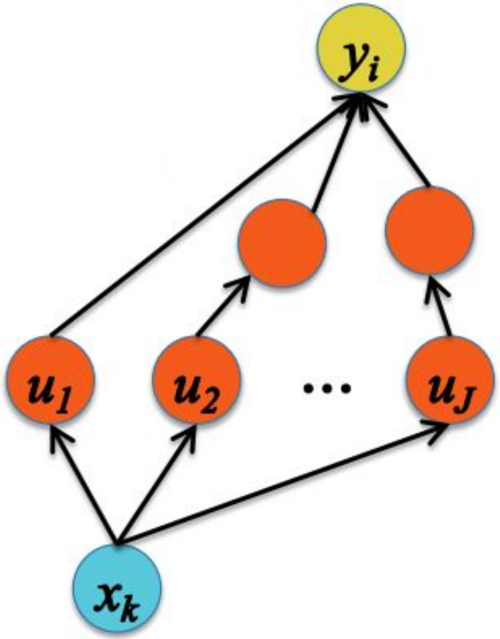
Given:

$$\boldsymbol{y} = g(\boldsymbol{u}) \quad \text{and} \quad \boldsymbol{u} = h(\boldsymbol{x})$$

Chain Rule:

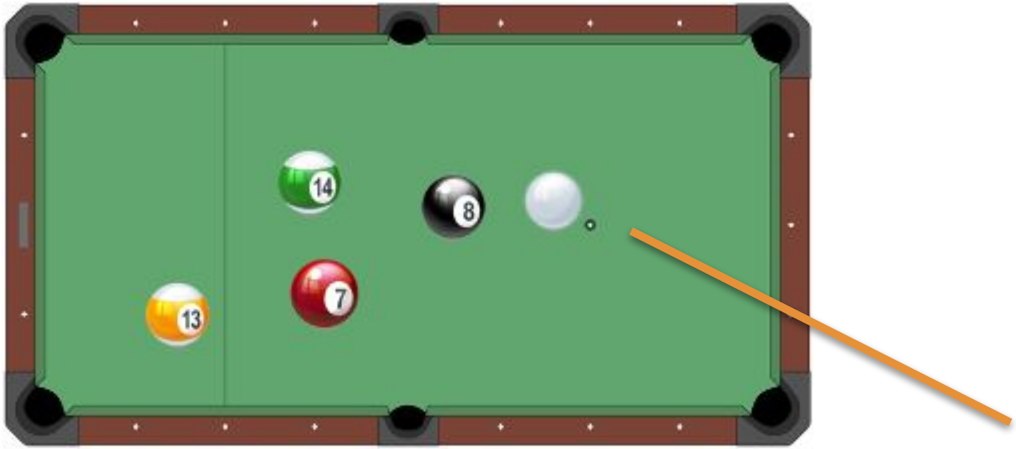
$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k} \qquad \forall i, k$$

Backpropagation is just repeated application of the **chain rule**

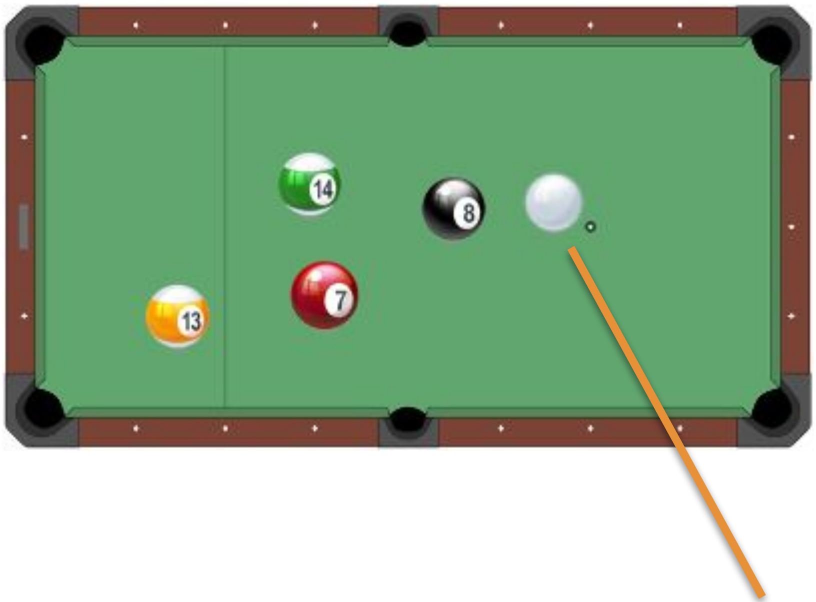


Intuition: Backpropagation of Errors

- Consider billiards as an example

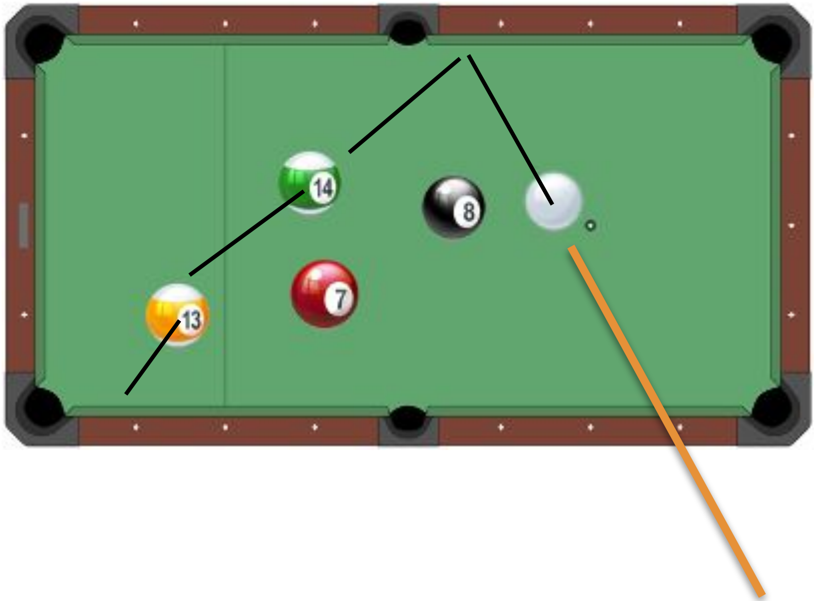


Intuition: Backpropagation of Errors



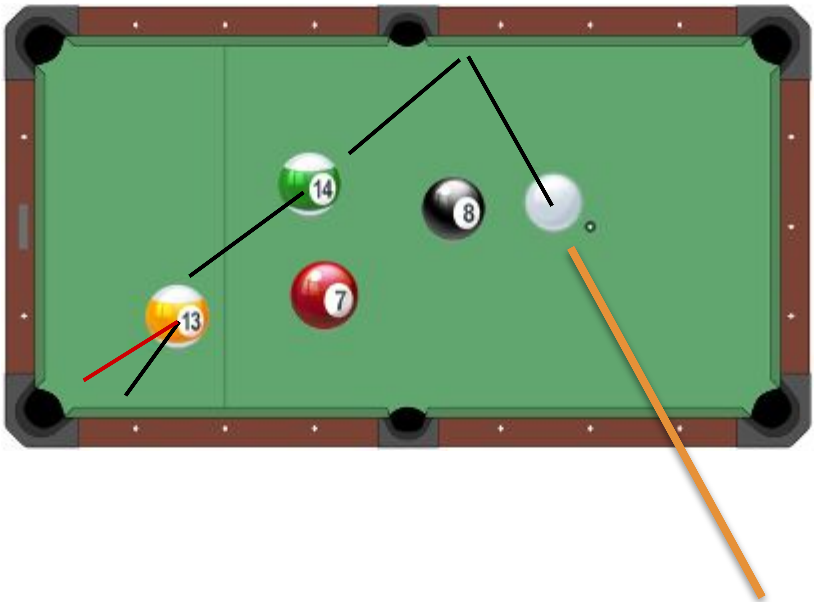
Slides credit: Stoyanov & Eisner, 2012

Intuition: Backpropagation of Errors



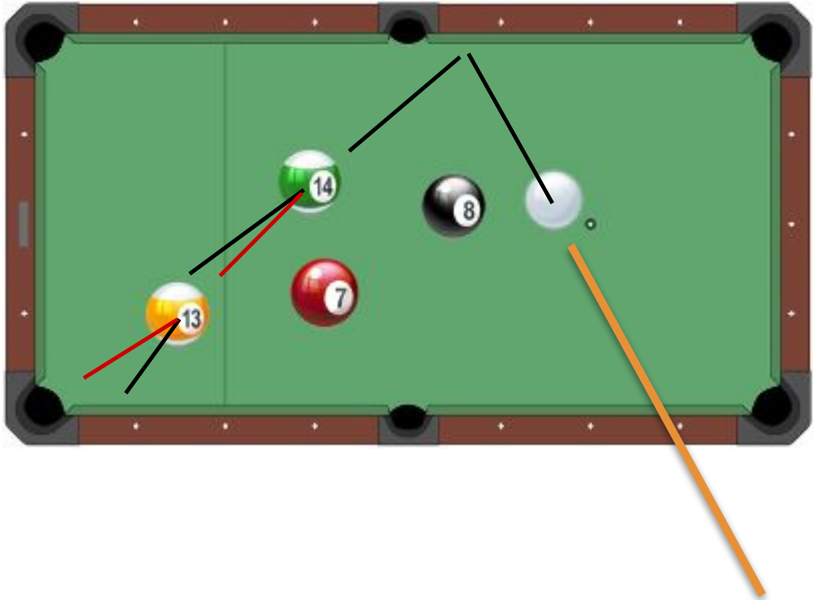
Slides credit: Stoyanov & Eisner, 2012

Intuition: Backpropagation of Errors



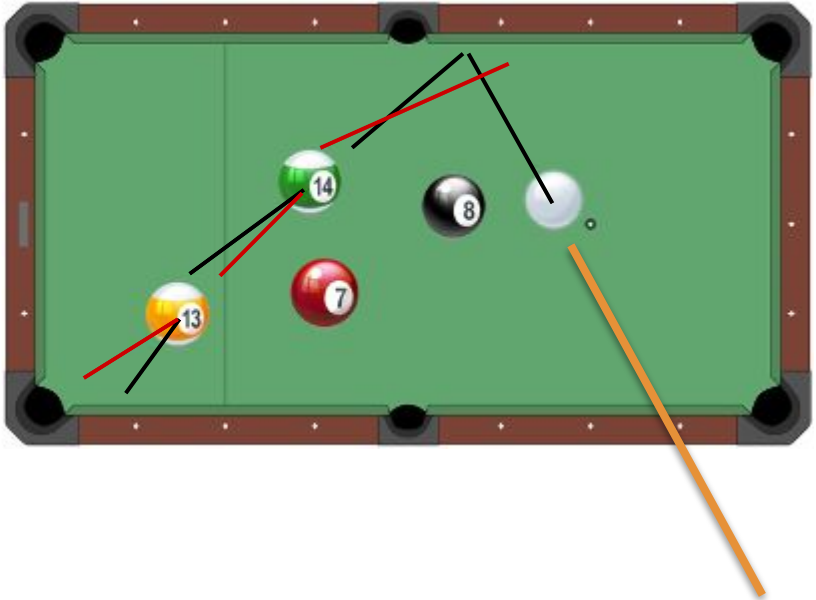
Slides credit: Stoyanov & Eisner, 2012

Intuition: Backpropagation of Errors



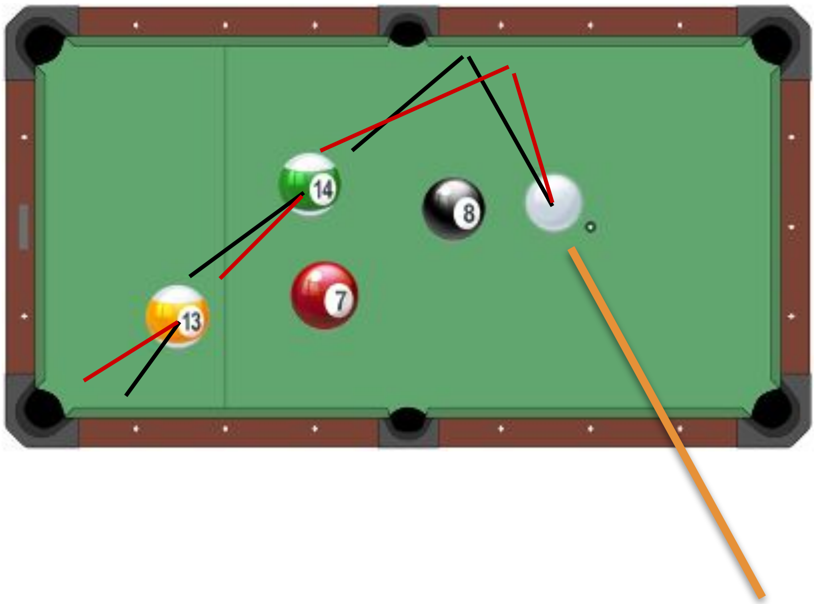
Slides credit: Stoyanov & Eisner, 2012

Intuition: Backpropagation of Errors



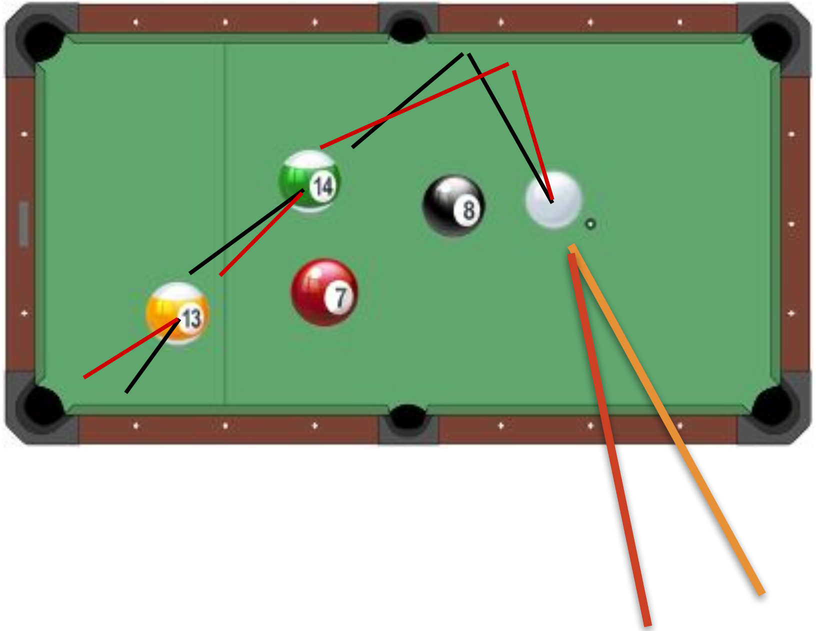
Slides credit: Stoyanov & Eisner, 2012

Intuition: Backpropagation of Errors



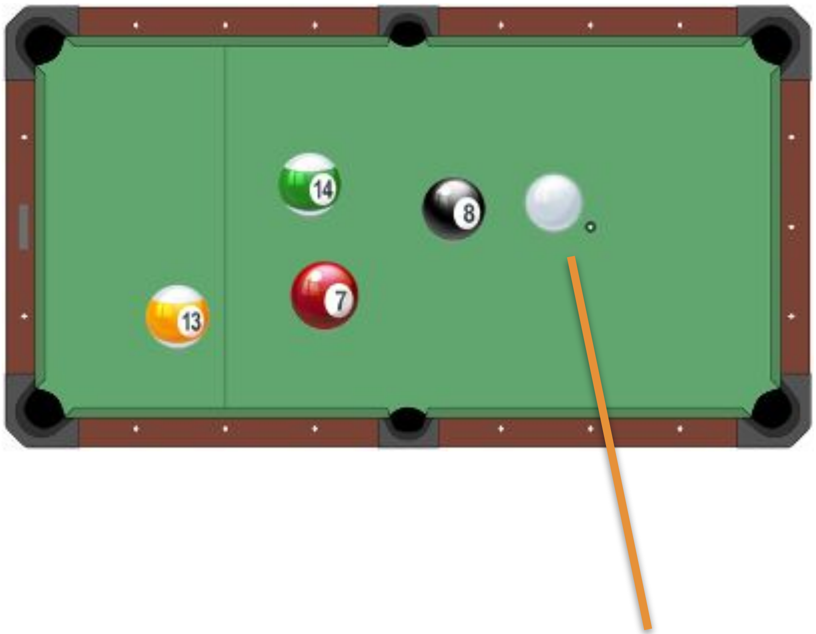
Slides credit: Stoyanov & Eisner, 2012

Intuition: Backpropagation of Errors



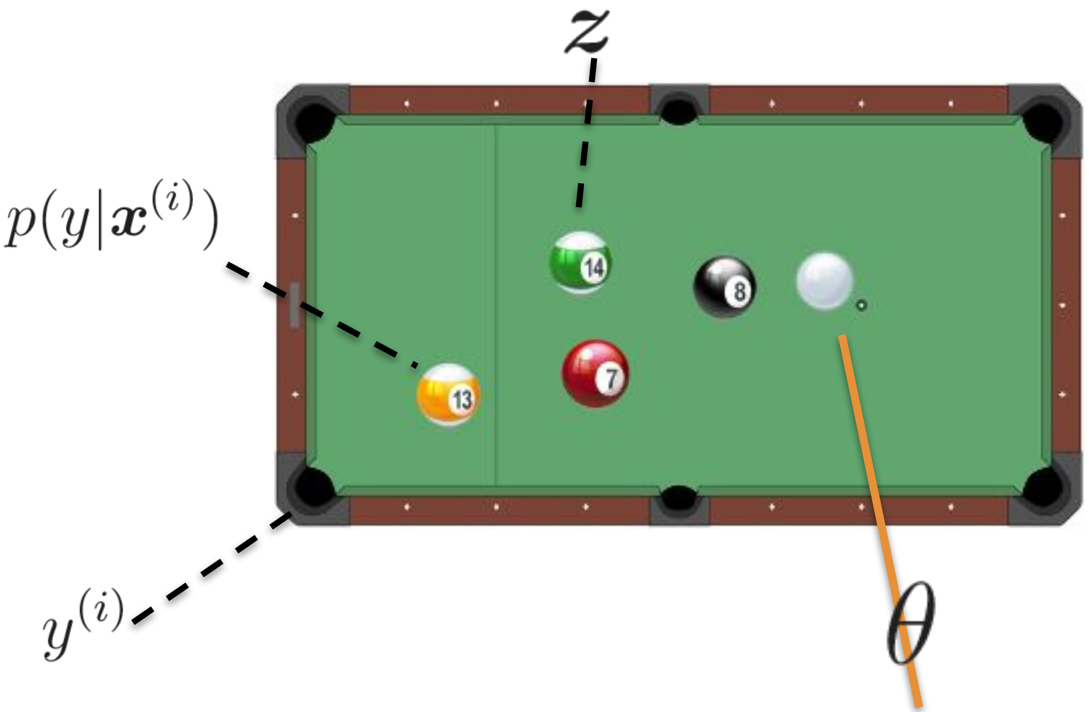
Slides credit: Stoyanov & Eisner, 2012

Intuition: Backpropagation of Errors



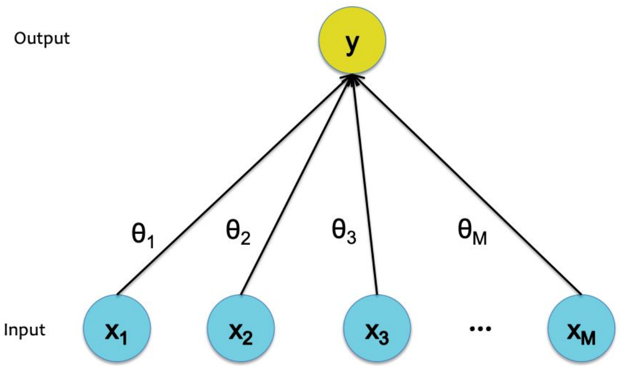
Slides credit: Stoyanov & Eisner, 2012

Intuition: Backpropagation of Errors



Slides credit: Stoyanov & Eisner, 2012

Backpropagation in Logistic Regression



In gradient descent, we need to compute

$$\frac{\partial J}{\partial \theta_j}$$

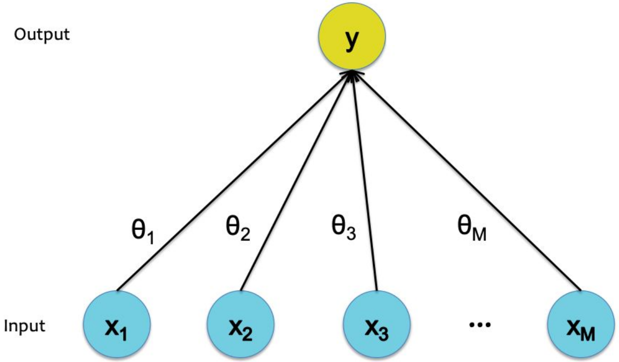
Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backpropagation in Logistic Regression



In gradient descent, we need to compute

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial a} \cdot \frac{\partial a}{\partial \theta_j}$$

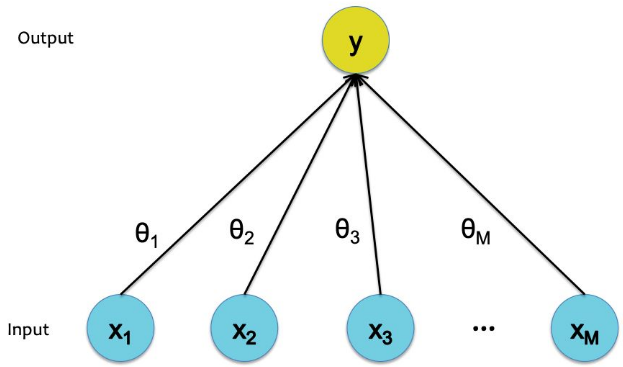
Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backpropagation in Logistic Regression

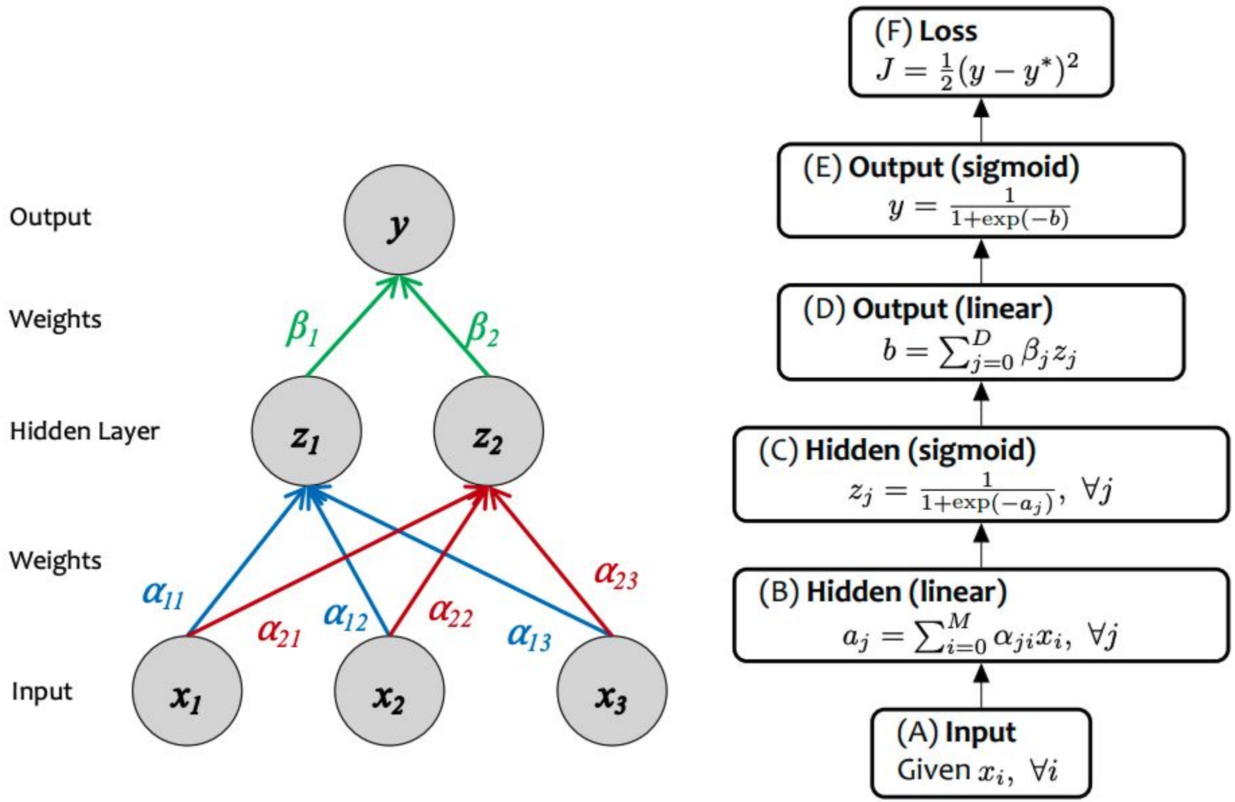


In gradient descent, we need to compute

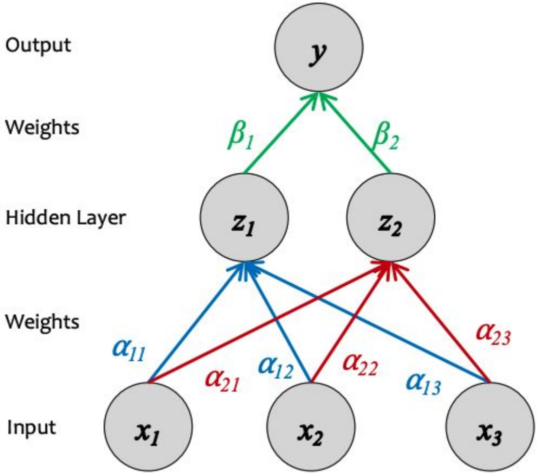
$$\frac{\partial J}{\partial \theta_j} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial a} \cdot \frac{\partial a}{\partial \theta_j}$$

Forward	Backward
$J = y^* \log y + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
$y = \frac{1}{1 + \exp(-a)}$	$\frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da}, \ \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$
$a = \sum_{j=0}^{D} \theta_j x_j$	$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \ \frac{da}{d\theta_j} = x_j$

Backpropagation in Neural Network



Backpropagation in Neural Network



In gradient descent, we need to compute

$$\frac{\partial J}{\partial \alpha_{ij}} \qquad \frac{\partial J}{\partial \beta_k}$$

Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

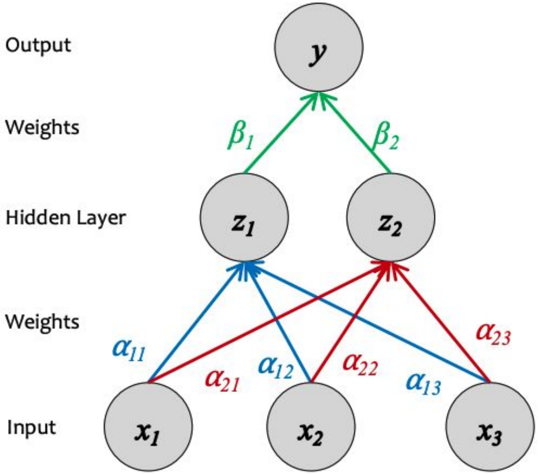
$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backpropagation in Neural Network



In gradient descent, we need to compute

$$\frac{\partial J}{\partial \alpha_{ij}} \qquad \frac{\partial J}{\partial \beta_k}$$

Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

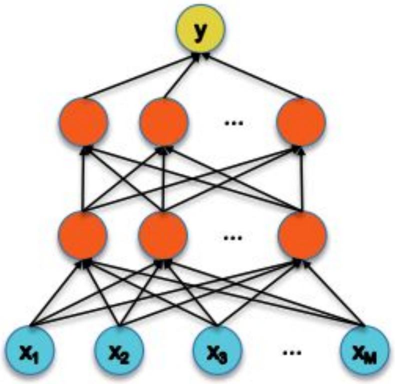
$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \sum_{j=0}^{D} \frac{dJ}{da_j}\frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \alpha_{ji}$$

Backpropagation in Neural Network

	Forward	Backward
Loss	$J = y^* \log y + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
Sigmoid	$y = \frac{1}{1 + \exp(-b)}$	$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$
Linear	$b = \sum_{j=0}^{D} \beta_j z_j$	$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$ $\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$
Sigmoid	$z_j = \frac{1}{1 + \exp(-a_j)}$	$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$
Linear	$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$ $\frac{dJ}{dx_i} = \sum_{j=0}^{D} \frac{dJ}{da_j}\frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \alpha_{ji}$

Gradient Descent



Now we know how to compute the gradient

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_d} \end{bmatrix}$$

We can run gradient descent to optimize the neural network

$$\theta^{(t)} = \theta^{(t-1)} - \alpha \cdot \nabla J(\theta^{(t-1)})$$

Revisit Gradient Descent

1. Start with an initial guess $\theta^{(0)}$ and set $t = 0$
2. While Termination Criterion is not satisfied
   a. Compute the gradient

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_d} \end{bmatrix} = \frac{1}{n} \sum_{i=1}^{n} \begin{bmatrix} \frac{\partial J^{(i)}(\theta)}{\partial \theta_0} \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_d} \end{bmatrix} = \frac{1}{n} \sum_{i=1}^{n} \left(f_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

   b. Update θ: $\theta^{(t)} = \theta^{(t-1)} - \alpha \cdot \nabla J(\theta^{(t-1)})$
   c. Increment t: $t = t + 1$
3. Output $\theta^{(t)}$

Revisit Gradient Descent

1. Start with an initial guess $\theta^{(0)}$ and set $t = 0$
2. While Termination Criterion is not satisfied
   a. Compute the gradient

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_d} \end{bmatrix} = \frac{1}{n}\sum_{i=1}^{n} \begin{bmatrix} \frac{\partial J^{(i)}(\theta)}{\partial \theta_0} \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_d} \end{bmatrix} = \boxed{\frac{1}{n}\sum_{i=1}^{n}} \left(f_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

   b. Update θ: $\theta^{(t)} = \theta^{(t-1)} - \alpha \cdot \nabla J(\theta^{(t-1)})$
   c. Increment t: $t = t + 1$
3. Output $\theta^{(t)}$

- Need to go over all data samples
- Inefficient: O(Nd) time, where d is #features
- What if the training set has 10^6 images?

Stochastic Gradient Descent

1. Start with an initial guess $\theta^{(0)}$ and set $t = 0$
2. While Termination Criterion is not satisfied
   a. Randomly sample a data sample $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$
   b. Compute the pointwise gradient

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_d} \end{bmatrix} = \cancel{\frac{1}{n}\sum_{i=1}^{n}} \begin{bmatrix} \frac{\partial J^{(i)}(\theta)}{\partial \theta_0} \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_d} \end{bmatrix} = \cancel{\frac{1}{n}\sum_{i=1}^{n}} \left(f_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

   c. Update θ: $\theta^{(t)} = \theta^{(t-1)} - \alpha \cdot \nabla J(\theta^{(t-1)})$
   d. Increment t: $t = t + 1$
3. Output $\theta^{(t)}$

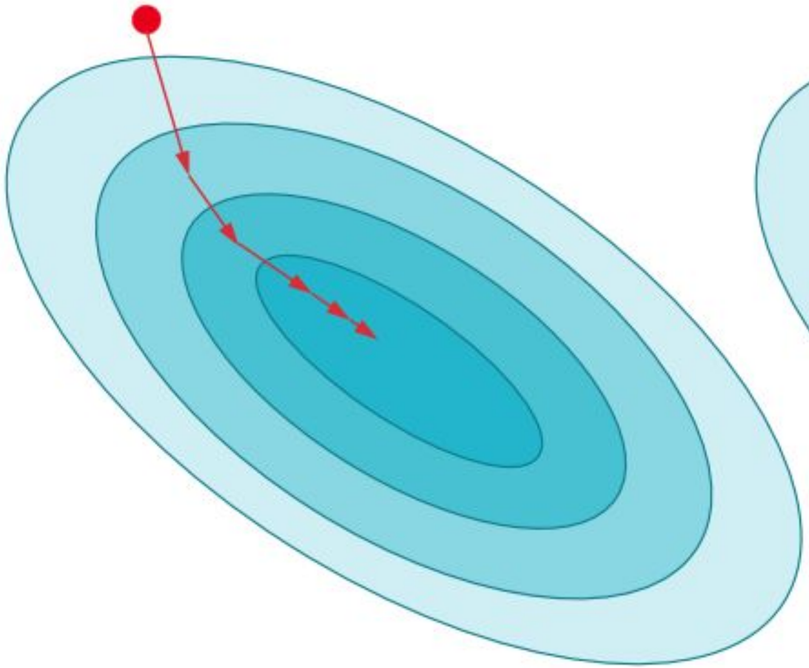
Stochastic Gradient Descent

1. Start with an initial guess $\theta^{(0)}$ and set $t = 0$
2. While Termination Criterion is not satisfied
   a. Randomly sample a data sample $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$
   b. Compute the pointwise gradient

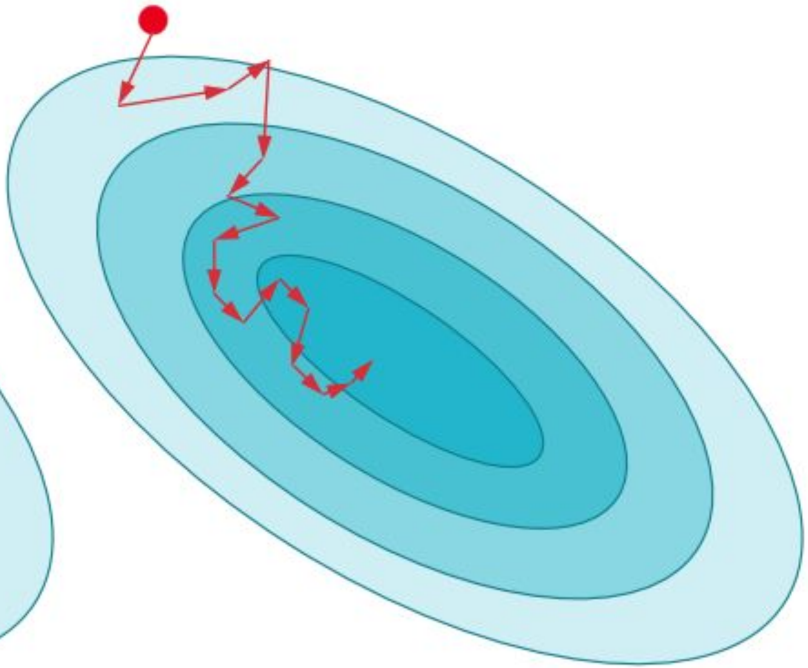
$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_d} \end{bmatrix} = \begin{bmatrix} \frac{\partial J^{(i)}(\theta)}{\partial \theta_0} \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_d} \end{bmatrix} = \left(f_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

   c. Update θ: $\theta^{(t)} = \theta^{(t-1)} - \alpha \cdot \nabla J(\theta^{(t-1)})$
   d. Increment t: $t = t + 1$
3. Output $\theta^{(t)}$

Stochastic Gradient Descent vs. Gradient Descent



Gradient Descent



Stochastic Gradient Descent

Mini-batch Stochastic Gradient Descent

A compromise between GD and SGD

1. Start with an initial guess $\theta^{(0)}$ and set $t = 0$
2. While Termination Criterion is not satisfied
   a. Randomly sample a batch of data samples (with size $m << n$), $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{m}$
   b. Compute the batch-wise gradient

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_d} \end{bmatrix} = \boxed{\frac{1}{m}\sum_{i=1}^{m}} \begin{bmatrix} \frac{\partial J^{(i)}(\theta)}{\partial \theta_0} \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J^{(i)}(\theta)}{\partial \theta_d} \end{bmatrix} = \boxed{\frac{1}{m}\sum_{i=1}^{m}} \left(f_\theta(\boldsymbol{x}^{(i)}) - y^{(i)}\right) \cdot \boldsymbol{x}^{(i)}$$

   c. Update θ: $\theta^{(t)} = \theta^{(t-1)} - \alpha \cdot \nabla J(\theta^{(t-1)})$
   d. Increment t: $t = t + 1$
3. Output $\theta^{(t)}$

- *m* is called "batch size"
- SGD and Mini-batch SGD are terms used interchangeably in the literature

Conclusions

- Neural Networks
  - Multi-layer architecture
  - Nonlinear transformation (activation function)
- Design space of NN
- Backpropagation
- Stochastic Gradient Descent