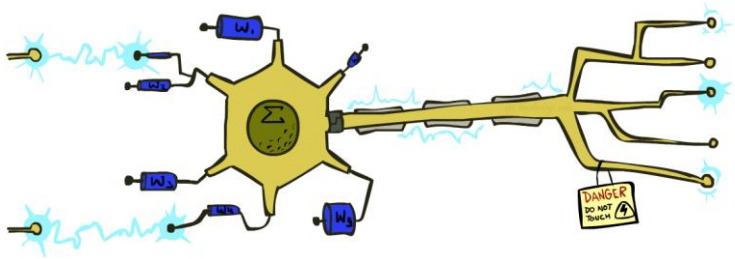
Neural Networks

CSE 473: Introduction to Artificial Intelligence



Megan Hazen (mh75@cs)
cs.uw.edu/473
acknowledgements

Welcome!
What does the term 'neuron' mean to you?

Administrivia

ANNOUNCEMENTS

- **Test 2** scores will be released Wednesday

ASSIGNMENTS

- **Project 4** published
- **Homework 4** due after break



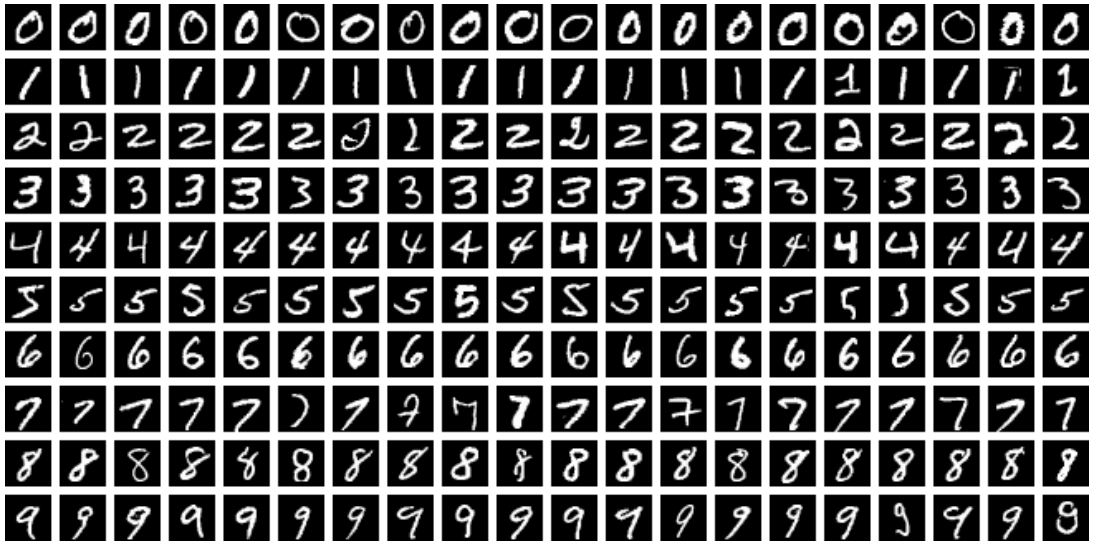
Taking Stock

CONTEXT

How do these models compare?

Model	Test Acc.	Train Time	Test Time
Logistic Reg.	92.55%	176.6s	0.01s
SVM (RBF)	96.85%	2.9s	7.74s
Decision Tree	87.58%	8.2s	0.01s
kNN (k=5)	96.88%	0.1s	3.58s
Neural Net.	?	?	?

One-shot benchmark using standard sklearn implementations (no hyperparameter tuning). Python notebook generated by Claude Code.

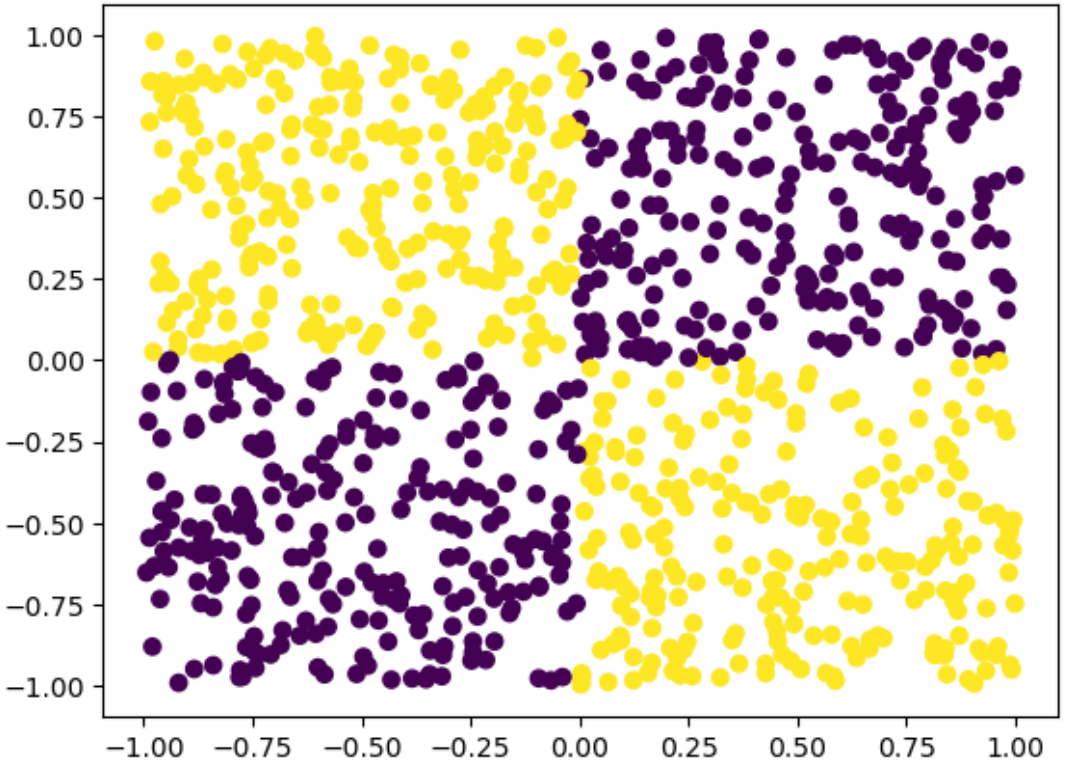


Benchmark experiment Inspired by Michael Nielsen

Motivation

CONTEXT

Separable with $h_{\boldsymbol{w}}(\boldsymbol{x}) = w_0 + w_1 x_1 x_2$

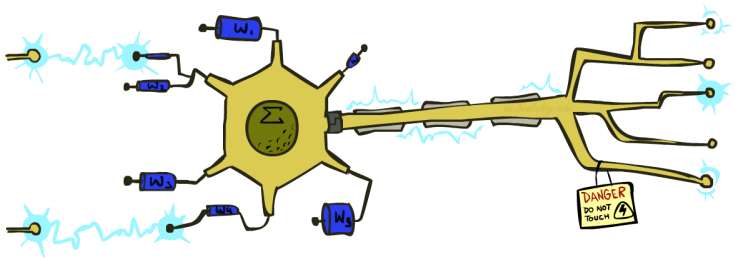


XOR

Shouldn't the model do the feature engineering?

Recap: Perceptrons

FOUNDATIONS



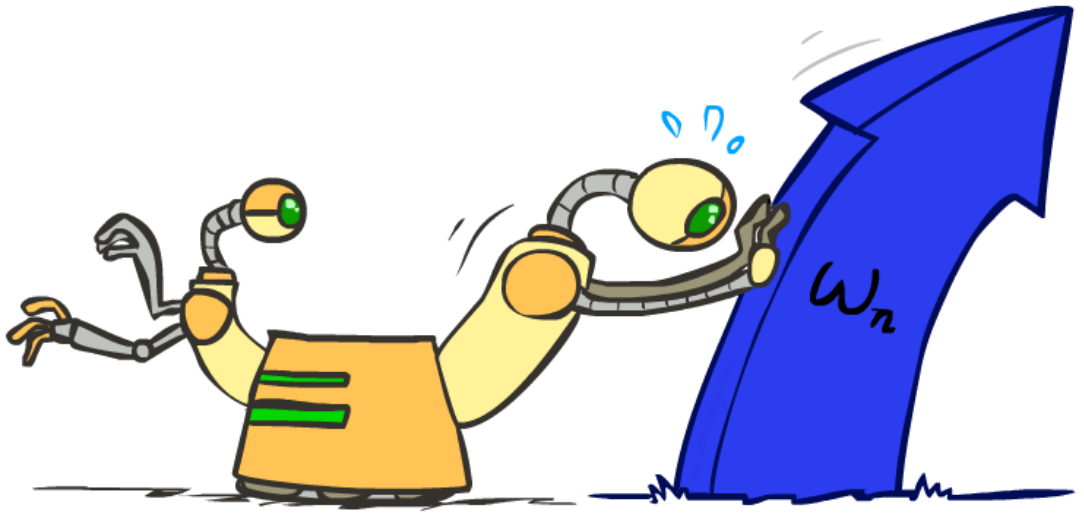
Perceptrons output $h_{\boldsymbol{w}}(x) = \boldsymbol{w} \cdot \boldsymbol{x}$

Repeat while not converged ($\exists i \;\; y_i \neq h_{\boldsymbol{w}}(\boldsymbol{x}_i)$):

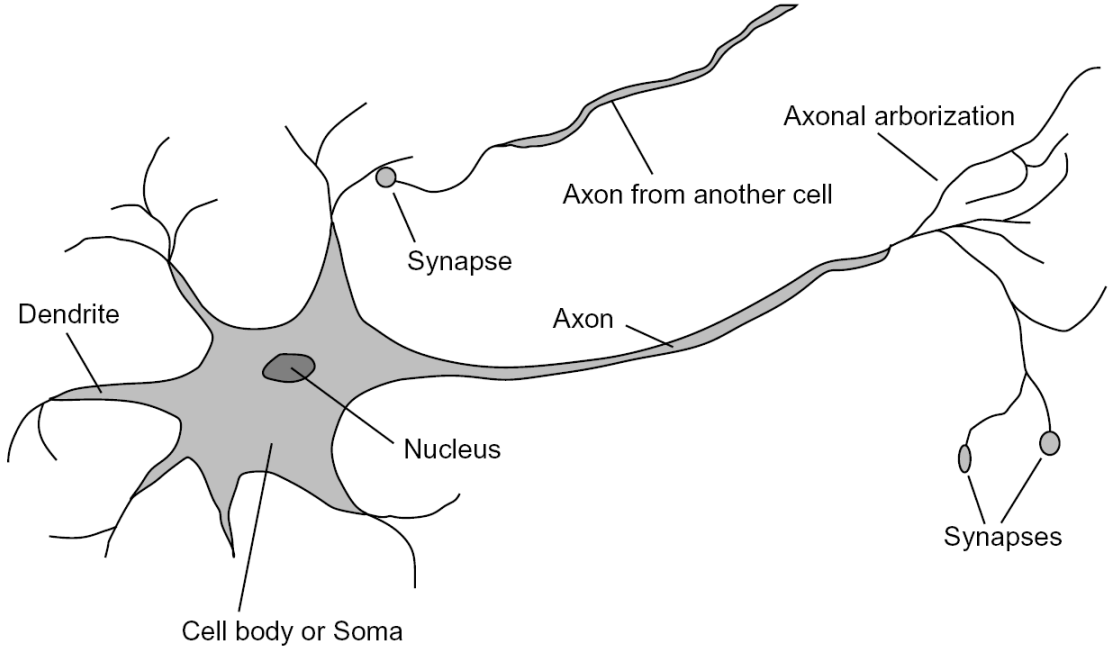
- For each data point i do:
  - $w \leftarrow w - \eta(y_i - h_{\boldsymbol{w}}(\boldsymbol{x}_i))\boldsymbol{x}_i$

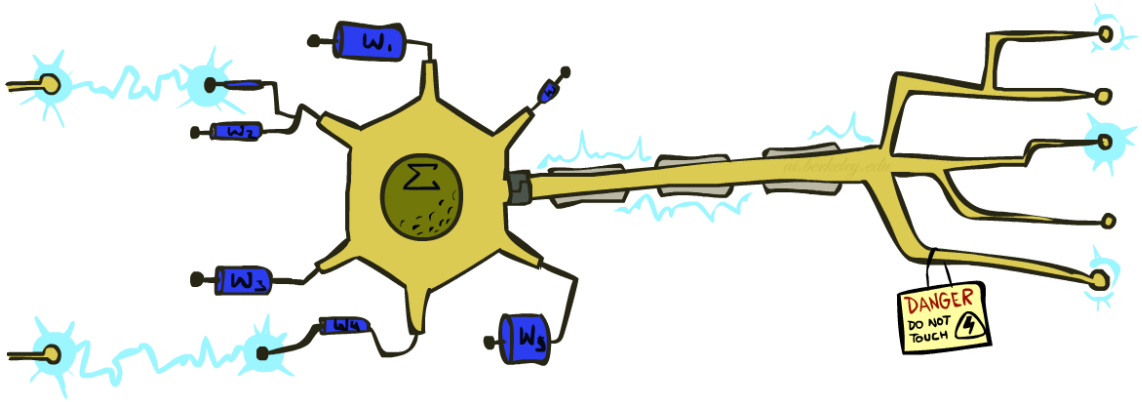
Gradient

Equivalent to least squares regression with $\mathcal{L} = \sum_i |y_i - h_{\boldsymbol{w}}(\boldsymbol{x}_i)|$



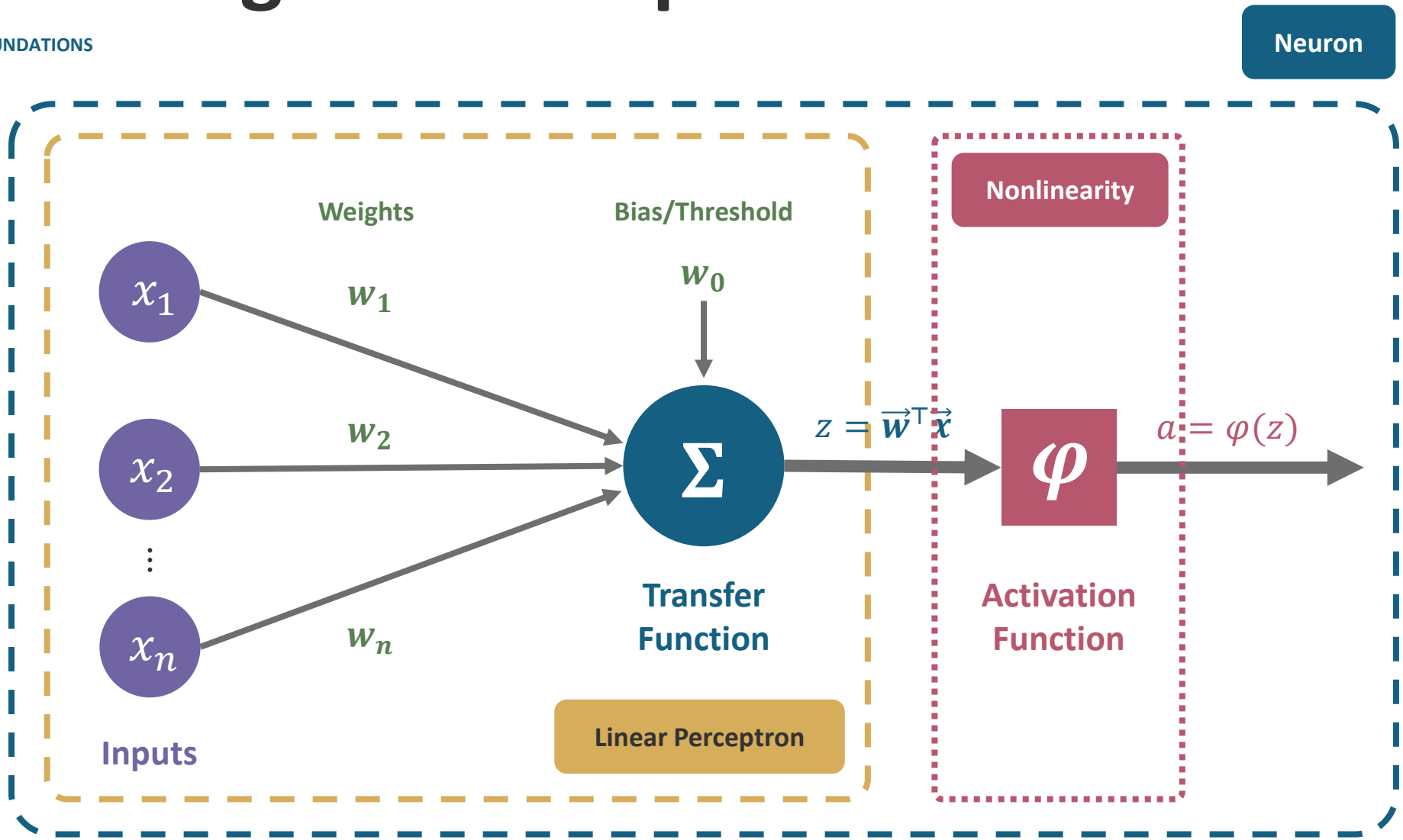
How does the human brain learn?





Generalizing the Perceptron

FOUNDATIONS



First detailed by Rosenblatt (1958)

Activation Functions

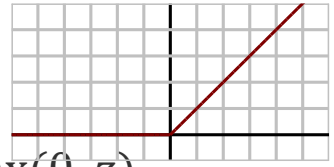
DEFINITION

Activation functions transform the perceptron output, introducing *nonlinearities*

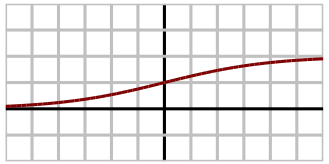
- Networks with linear activation functions are reducible to a single perceptron. Why?

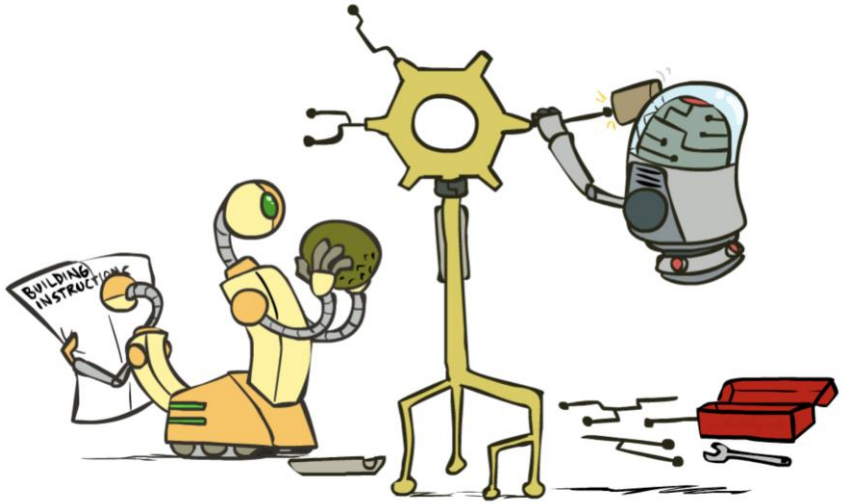
- Desirable Properties:
  - Nonlinear
  - Differentiable
  - *Continuously* differentiable?

- Candidates:
  - Rectified Linear Unit (ReLU) $\mathrm{ReLU}(z) = \max(0, z)$



  - Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$

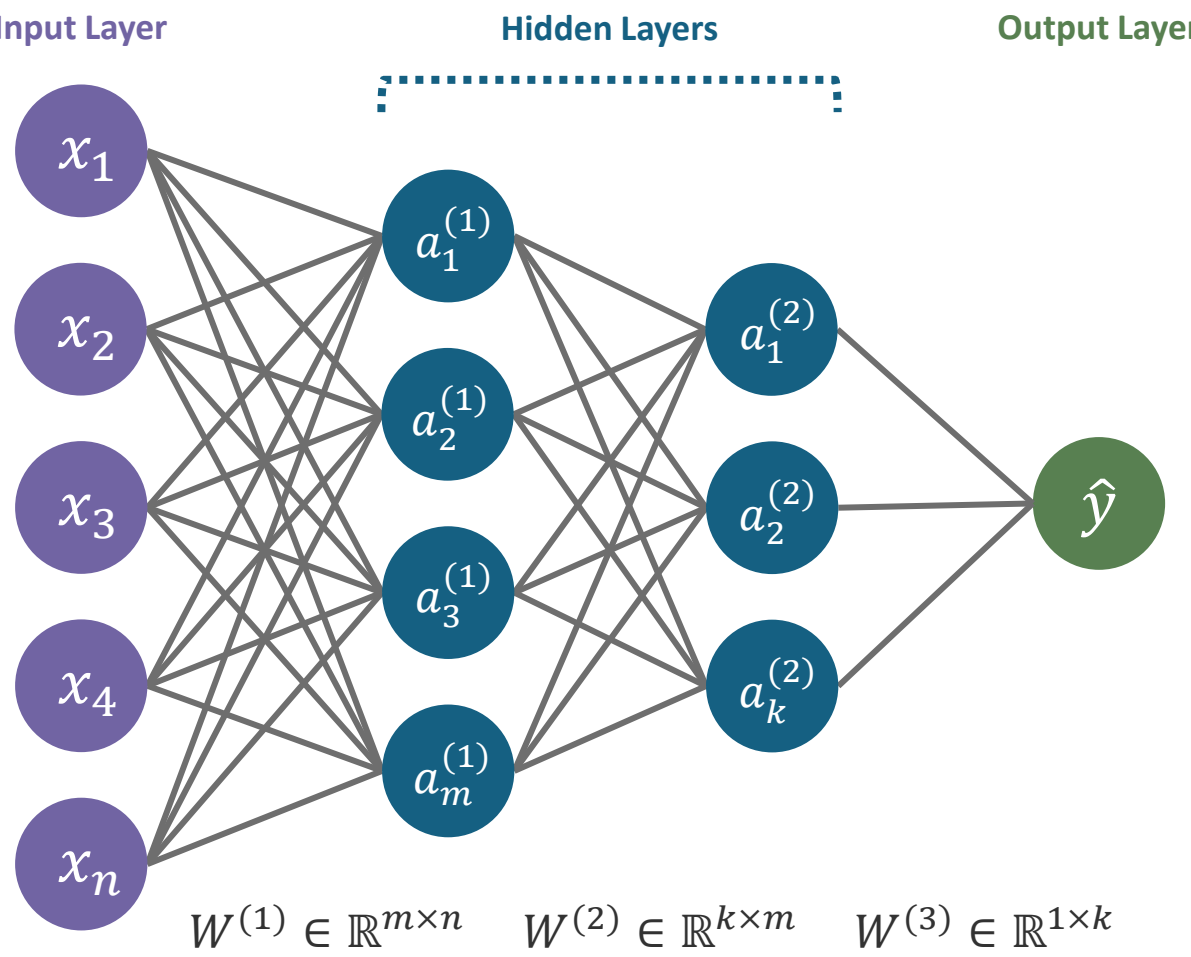




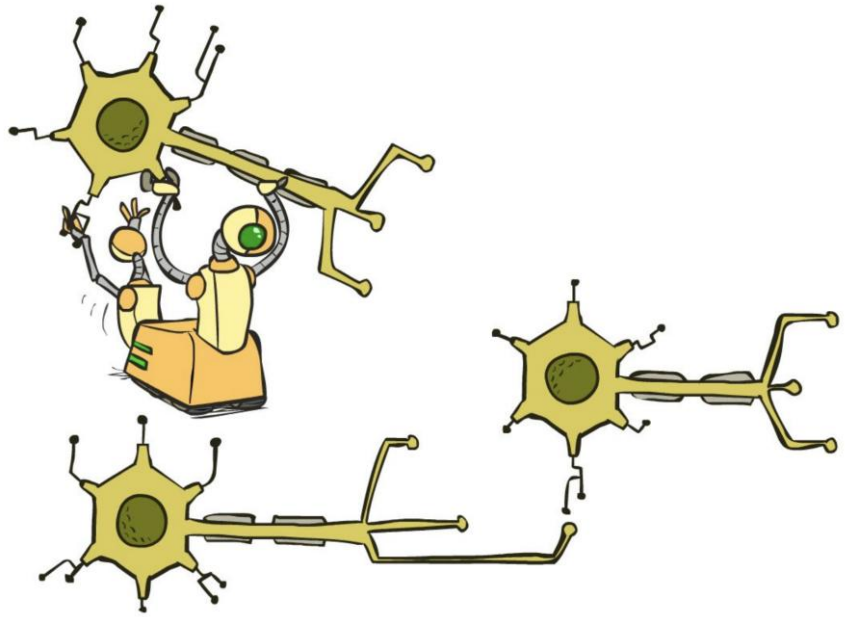
Questions?

Multilayer Perceptron

FOUNDATIONS

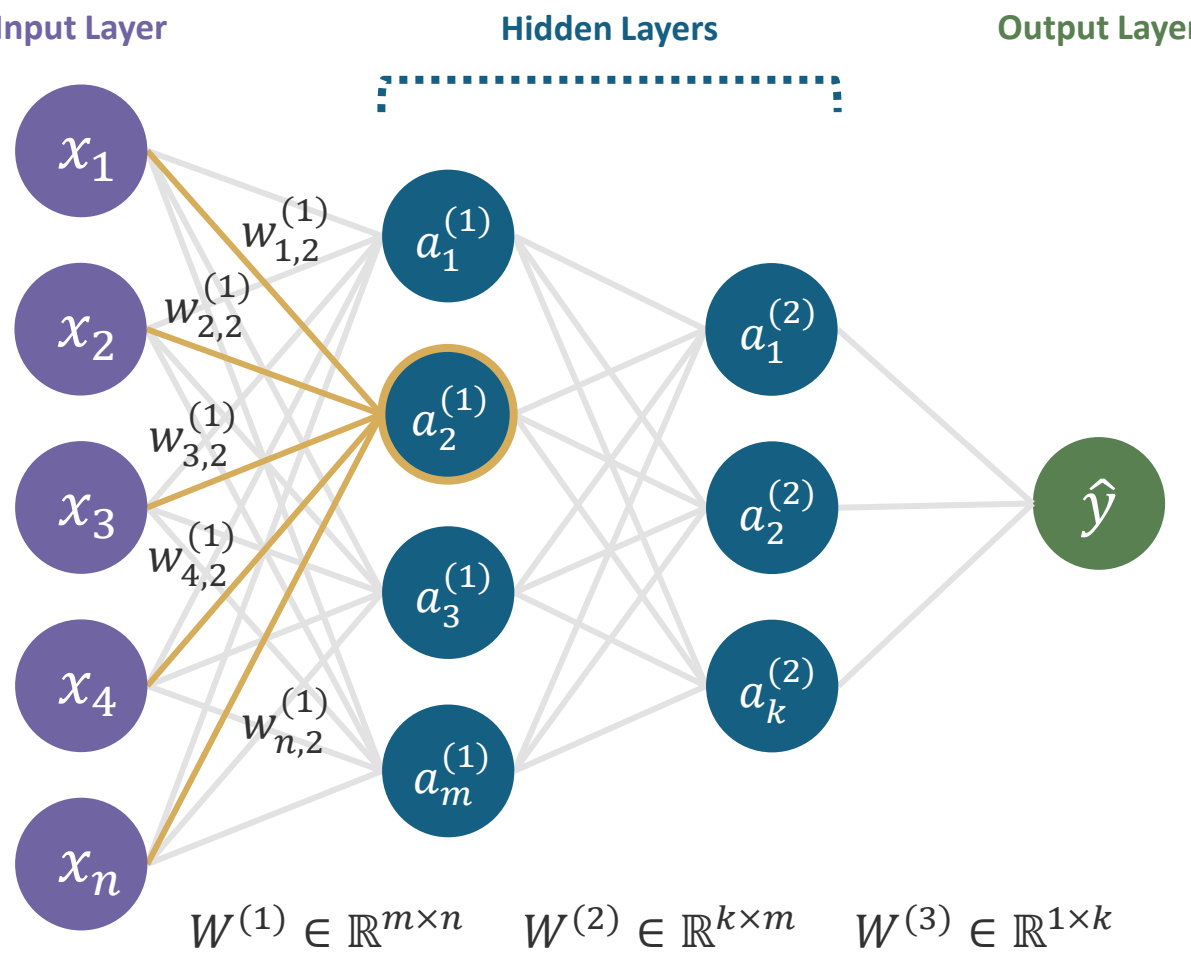


$$\hat{\boldsymbol{y}} = W^{(3)} \cdot \varphi(W^{(2)} \cdot \varphi(\boldsymbol{W}^{(1)}\boldsymbol{x}))$$



Forward Propagation

FOUNDATIONS



$$\hat{\boldsymbol{y}} = W^{(3)} \cdot \varphi(W^{(2)} \cdot \varphi(\boldsymbol{W}^{(1)}\boldsymbol{x}))$$

$$a_2^{(1)} = \varphi\left(\begin{bmatrix} \vdots \\ w_{1,2}^{(1)} \quad \cdots \quad w_{n,2}^{(l)} \\ \vdots \end{bmatrix}\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}\right)$$

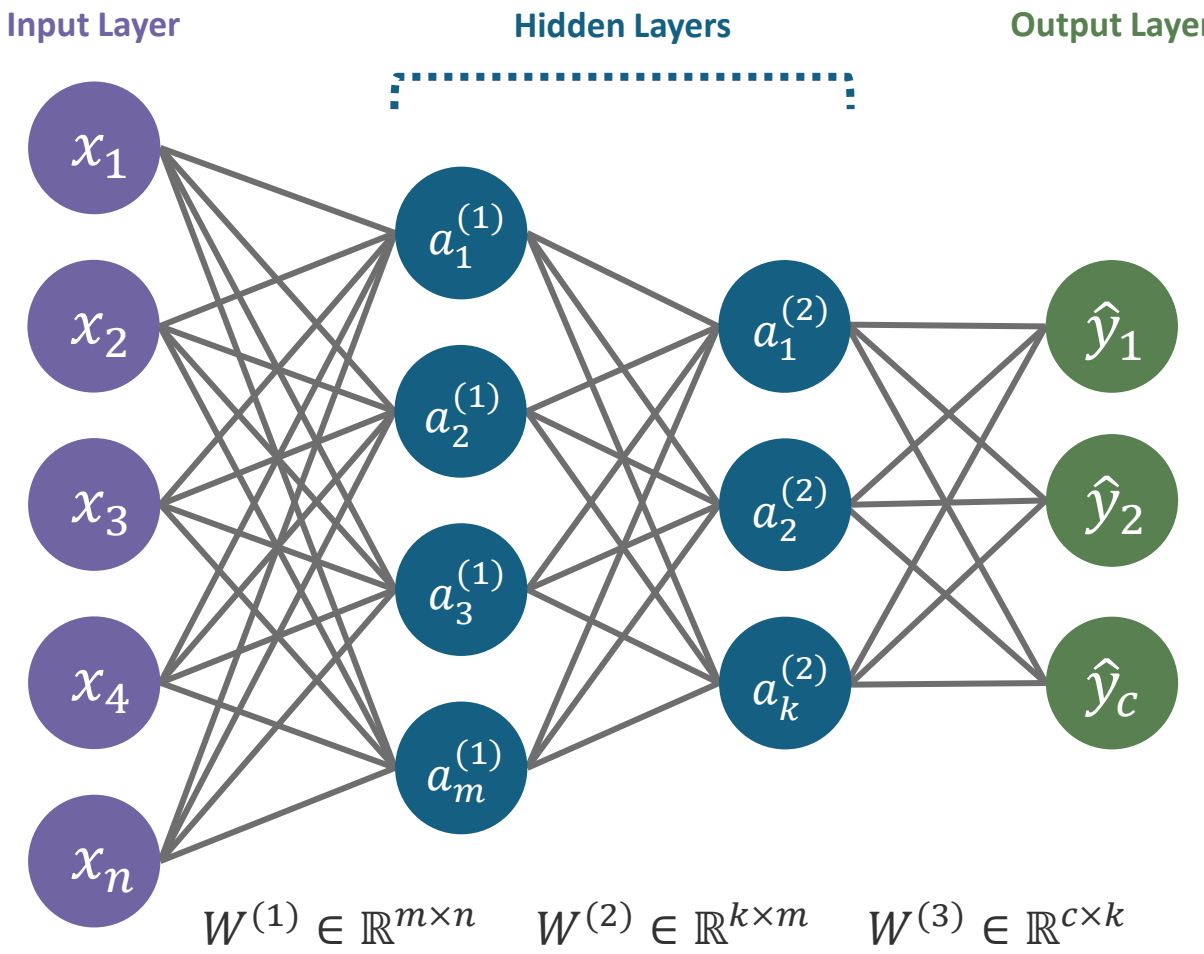
$$a_2^{(1)} = \varphi(\vec{\boldsymbol{w}}_2^{(1)}\vec{\boldsymbol{x}}) = \varphi(\sum_i w_{i,2}^{(1)} \cdot x_i)$$

$$a_j^{(l)} = \varphi(\vec{\boldsymbol{w}}_j^{(l)}\vec{\boldsymbol{a}}^{(l-1)})$$

$$= \varphi(\sum_i w_{i,j}^{(l)} \cdot a_i^{(l-1)})$$

Classifier Output

FOUNDATIONS



$$\hat{y} = \text{argmax}_c \, \hat{y}_c$$

- $\hat{y}_1, \ldots, \hat{y}_c$ are called **logits**
- Normalize by sum?

$$\frac{\hat{y}_i}{\sum_c \hat{y}_c}$$

- Problem: logits can be negative…
- Solution: **Softmax** function

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_c e^{z_c}}$$

Questions?

Learning XOR

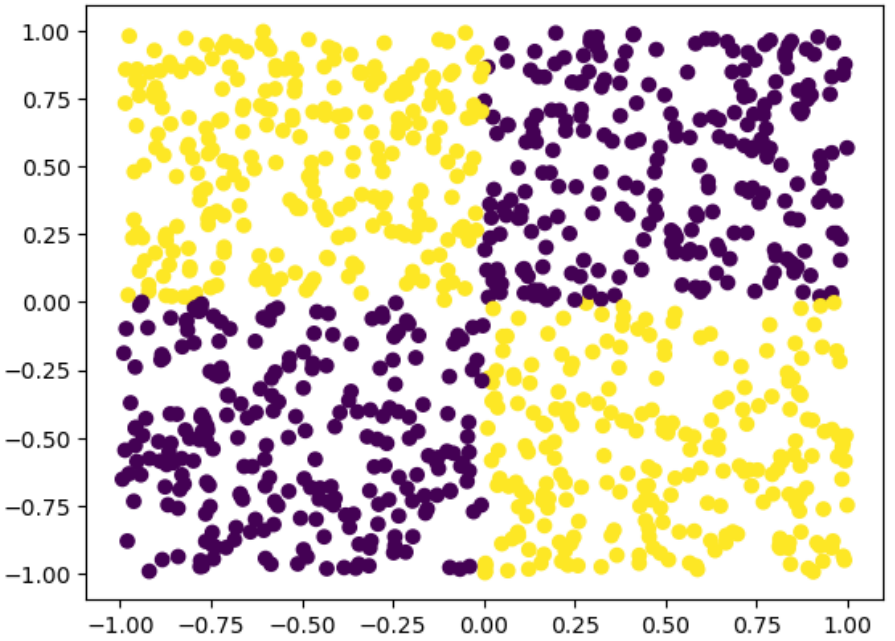
EXAMPLE

Can we learn XOR?

$$\text{XOR}(x_1, x_2) = \text{OR}(\text{AND}(x_1, \text{NOT}(x_2)), \text{AND}(\text{NOT}(x_1), x_2))$$

- Need to learn:
  - $\text{AND}(x_1, \text{NOT}(x_2))$
  - $\text{AND}(\text{NOT}(x_1), x_2)$
  - $\text{OR}(x_1, x_2)$

What perceptron weights w_0, w_1, w_2 approximate these functions?

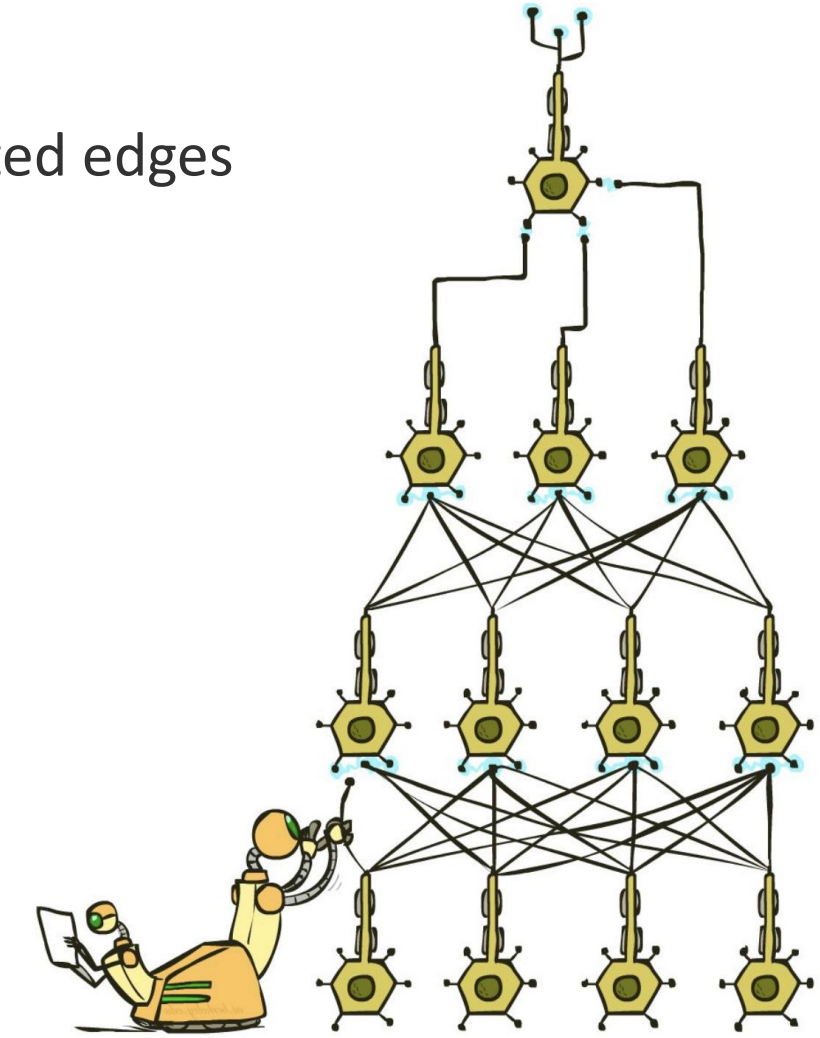


Neural Network

FOUNDATIONS

Neural networks consist of neurons connected by weighted edges in some architecture

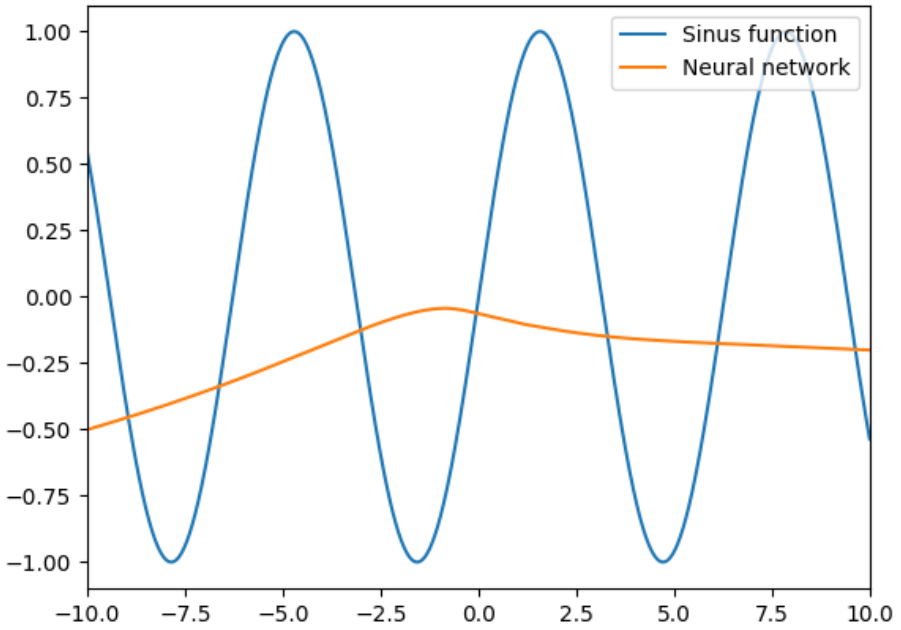
- Multilayer perceptrons are the classic neural networks
  - MLPs are an instance of a type of NN called **feed-forward networks**
- Hyperparameters:
  - Network Structure:
    - Number of hidden layers
    - Hidden layer sizes
    - Connections between neurons
  - Activation Function
  - …



Universal Approximation Theorem

DEFINITION

Theorem: A one-layer feed-forward neural network of sufficient width and a nonlinear activation function can model an arbitrary continuous function

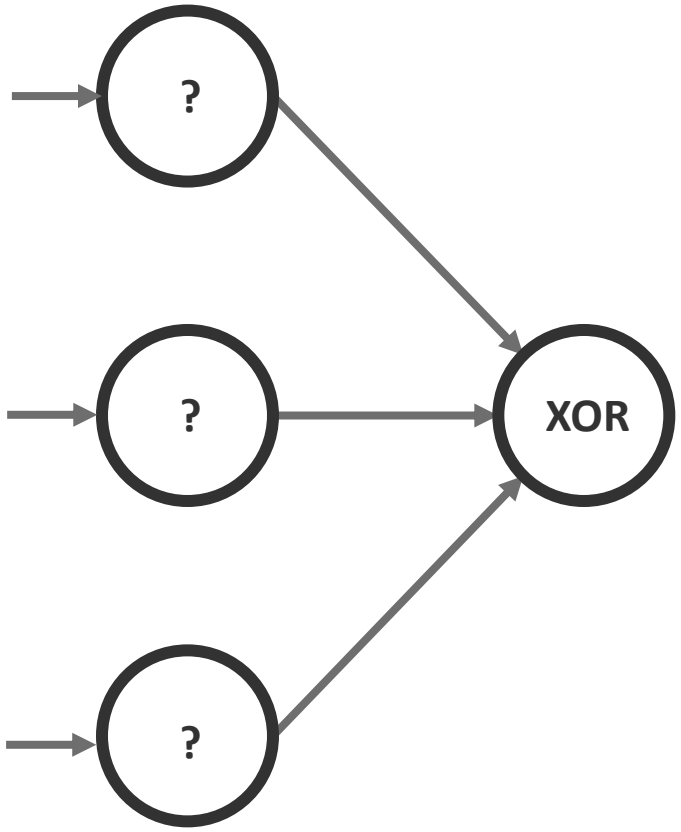


Learning XOR: Revisited

EXAMPLE

Can we learn XOR with a shallow network??

$$\text{XOR}(x_1, x_2) = ?$$



Yes, but no longer as a composition of functions

TensorFlow Playground Example

Questions?

Gradient Descent

DEFINITION

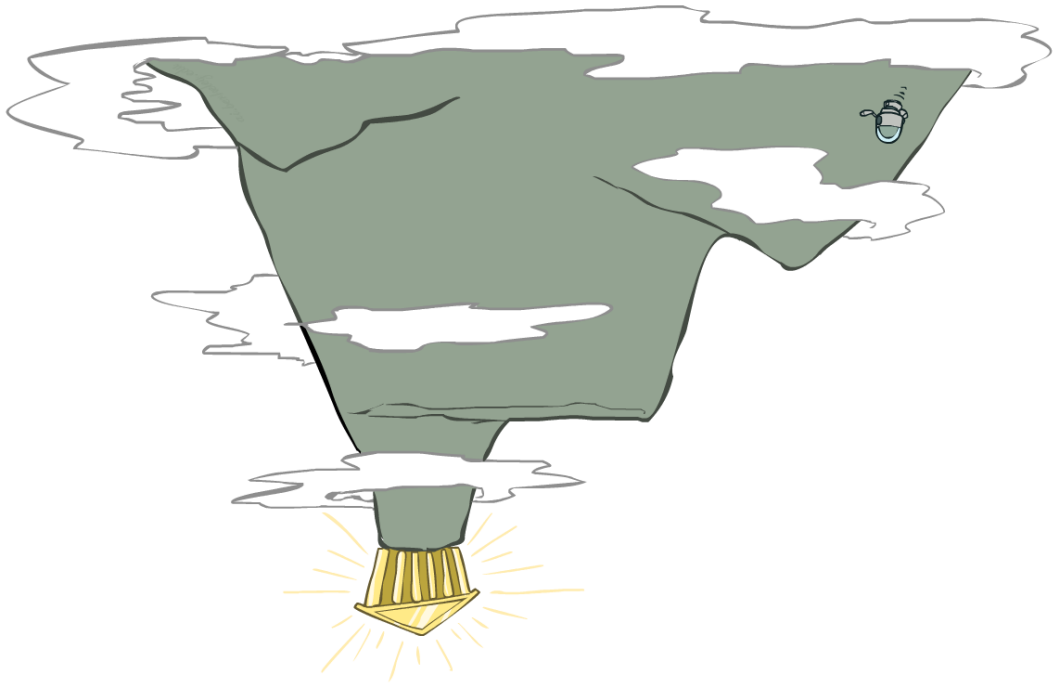
Gradient descent is a *greedy* optimization approach

- Take small steps in the direction of the steepest descent (negative gradient)

$$w \leftarrow w - \eta \frac{\partial}{\partial w} \mathcal{L}$$

  - Stop when updates become too small

- Considerations
  - How to find, calculate gradient?
  - Step size η matters a lot!
  - When to stop?



Neural Network Learning

FOUNDATIONS

Goal: Train the network to find weights that model $h \approx f$

Approach:

- Define loss function $\mathcal{L}(x, y)$ as error between $\hat{y}$ (prediction) and y (actual)
- Perform gradient descent with $\frac{\partial}{\partial W}\mathcal{L}(x, y)$

Complication:

- Gradient $\frac{\partial}{\partial W^{(l)}}\mathcal{L}(x, y)$ depends on $\frac{\partial}{\partial \vec{\boldsymbol{a}}^{(l)}}\mathcal{L}(x, y)$, depends on $\frac{\partial}{\partial \vec{\boldsymbol{z}}^{(l+1)}}\mathcal{L}(x, y)$, …

Chain Rule

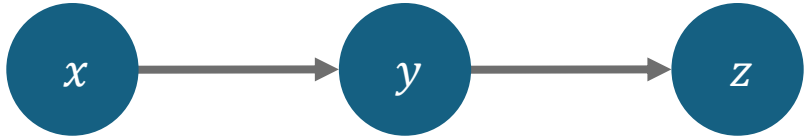
DEFINITION

- Standard formulation:

$$\frac{\partial}{\partial x} g\big(f(x)\big) = g'\big(f(x)\big) \cdot f'(x)$$

- Equivalent formulation:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$



Training a Single Neuron

FOUNDATIONS

$$a_j^{(l)} = \varphi(\vec{w}_j^{(l)}\vec{a}^{(l-1)}) = \varphi(\sum_i w_{i,j}^{(l)} \cdot a_i^{(l-1)})$$

Given $\vec{x}, y,\ \mathcal{L}(\vec{x}, y) = \left(y - \varphi(\vec{w}^\top \vec{x})\right)^2$ with activation function φ:

$$\frac{\partial}{\partial \vec{w}} \mathcal{L}(\vec{x}, y) = \frac{\partial}{\partial \vec{w}} \left(y - \varphi(\vec{w}^\top \vec{x})\right)^2$$

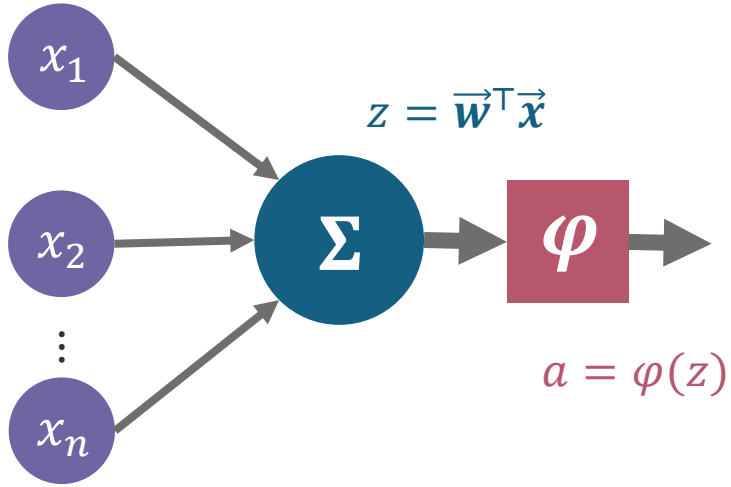
By the chain rule:

$$\frac{\partial}{\partial \vec{w}} \mathcal{L}(\vec{x}, y) = 2\left(y - \varphi(\vec{w}^\top \vec{x})\right) \cdot \frac{\partial}{\partial \vec{w}} \left(y - \varphi(\vec{w}^\top \vec{x})\right)$$

$$= -2\left(y - \varphi(\vec{w}^\top \vec{x})\right) \cdot \varphi'(\vec{w}^\top \vec{x}) \cdot \frac{\partial}{\partial \vec{w}} \vec{w}^\top \vec{x}$$

$$= -2\left(y - \varphi(\vec{w}^\top \vec{x})\right) \cdot \varphi'(\vec{w}^\top \vec{x}) \cdot \vec{x}$$

$\frac{\partial L}{\partial a}$ $\frac{\partial a}{\partial z}$ $\frac{\partial z}{\partial w}$

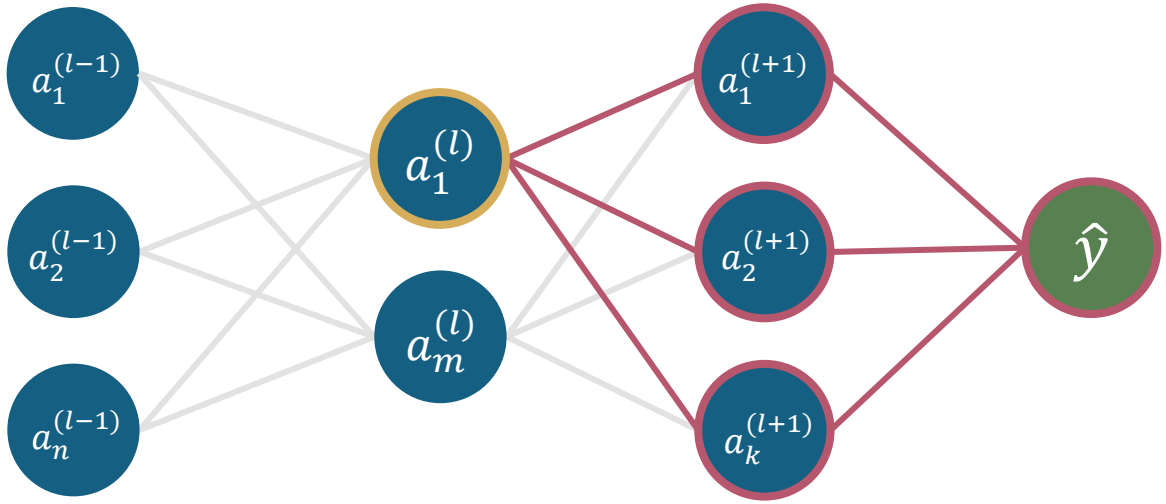


Backpropagation

DEFINITION

Compute the effect of a weight w on the loss function (i.e. $\frac{\partial \mathcal{L}}{\partial w}$) by **propagating the error backwards** through the network, combining with chain rule

$$\frac{\partial}{\partial z_i^{(l)}} \mathcal{L}(\vec{x}, y) = \frac{\partial \mathcal{L}}{\partial a_i^{(l)}} \cdot \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} = \frac{\partial \mathcal{L}}{\partial a_i^{(l)}} \cdot \varphi'(z_i^{(l)})$$



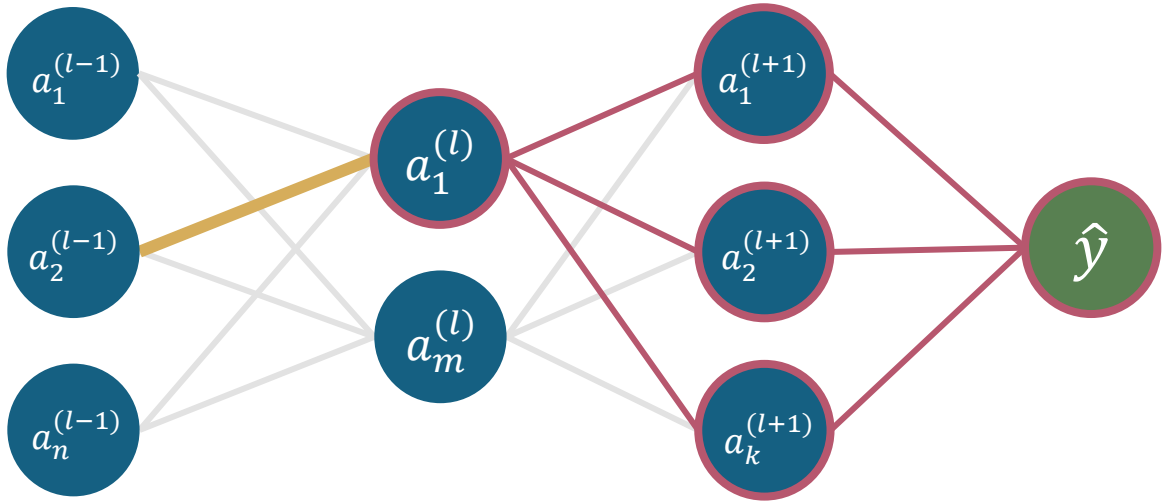
Rumelhart, Hinton, Williams in 1986

Backpropagation (2)

DEFINITION

Compute the effect of a weight w on the loss function (i.e. $\frac{\partial \mathcal{L}}{\partial w}$) by **propagating the error backwards** through the network, combining with chain rule

$$\frac{\partial}{\partial w_{i,j}^{(l)}} \mathcal{L}(\vec{x}, y) = \frac{\partial \mathcal{L}}{\partial z_i^{(l)}} \cdot \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial \mathcal{L}}{\partial z_i^{(l)}} \cdot a_j^{(l-1)}$$



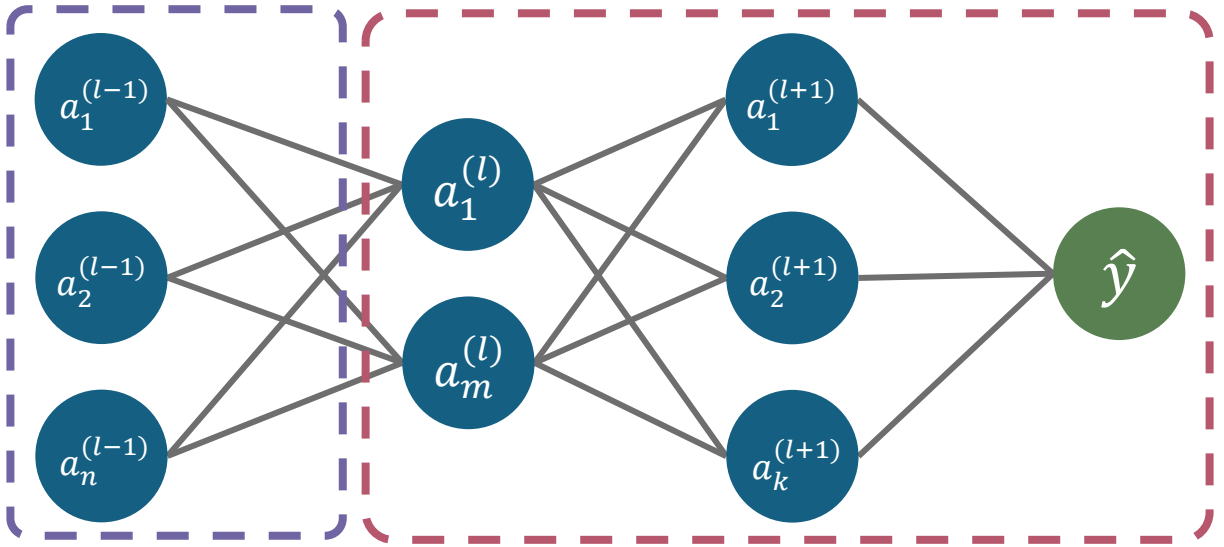
Weight Gradients

DEFINITION

Compute the effect of a weight w on the loss function (i.e. $\frac{\partial \mathcal{L}}{\partial w}$) by **propagating the error backwards** through the network, combining with chain rule

$$\frac{\partial}{\partial W^{(l)}} \mathcal{L}(\vec{\boldsymbol{x}}, y) = \frac{\partial \mathcal{L}}{\partial \vec{\boldsymbol{z}}^{(l)}} \cdot \frac{\partial \vec{\boldsymbol{z}}^{(l)}}{\partial W^{(l)}} = \frac{\partial \mathcal{L}}{\partial \vec{\boldsymbol{z}}^{(l)}} \cdot (\vec{a}^{(l-1)})^{\mathsf{T}}$$

$$\frac{\partial}{\partial A} A\vec{\boldsymbol{x}} = \vec{\boldsymbol{x}}^{\mathsf{T}}$$



Backpropagation: Recursive Formulation

DEFINITION

Compute the effect of a weight w on the loss function (i.e. $\frac{\partial \mathcal{L}}{\partial w}$) by **propagating the error backwards** through the network, combining with chain rule

$$\vec{\boldsymbol{\delta}}^{(l)} := \frac{\partial \mathcal{L}}{\partial \vec{\boldsymbol{z}}^{(l)}} = \left(W^{(l+1)} \vec{\boldsymbol{\delta}}^{(l+1)}\right) \cdot \varphi'(\vec{\boldsymbol{z}}^{(l)})$$

Element-wise product

For linear output layer L, $\vec{\boldsymbol{\delta}}^{(\boldsymbol{L})} = \boldsymbol{y} - \hat{\boldsymbol{y}}$

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \vec{\delta}^{(l)} \left(a^{(l-1)}\right)^{\mathsf{T}}$$

$$\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \vec{\delta}^{(l)}$$

Backpropagation: Weight Updates

DEFINITION

Compute the effect of a weight w on the loss function (i.e. $\frac{\partial \mathcal{L}}{\partial w}$) by **propagating the error backwards** through the network, combining with chain rule

$$w \leftarrow w - \eta \frac{\partial \mathcal{L}}{\partial w}$$

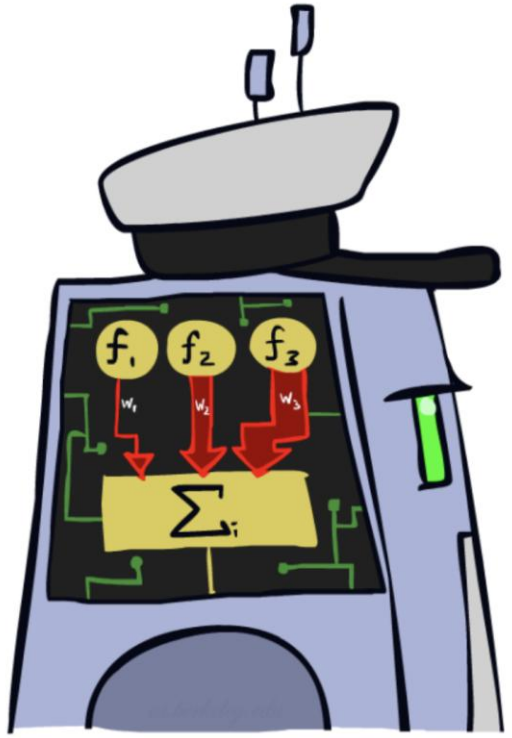
$$w \leftarrow w - \eta [\left(W^{(l+1)} \vec{\boldsymbol{\delta}}^{(l+1)} \right) \cdot \varphi'(\vec{\boldsymbol{z}}^{(l)}) \cdot (\vec{a}^{(l-1)})^{\mathsf{T}}]$$

Questions?

Practical Considerations

FOUNDATIONS

- Derivative Calculations
  - Need to be able to calculate $\frac{\partial}{\partial z}\varphi(z) = \varphi'(z)$
  - What happens when neuron activations have very small gradients?
- Parallelism?
  - Matrix multiplication is a good candidate for parallelization
  - Deep learning breakthrough arrived after using GPUs for training
- Batching
  - Batch Gradient Descent: Standard gradient descent with all training samples
  - Stochastic Gradient Descent: Pick random training sample to backpropagate
- Scale Requirements?
  - How much data do you need to build a network?
  - Can you use regularization to avoid over training?



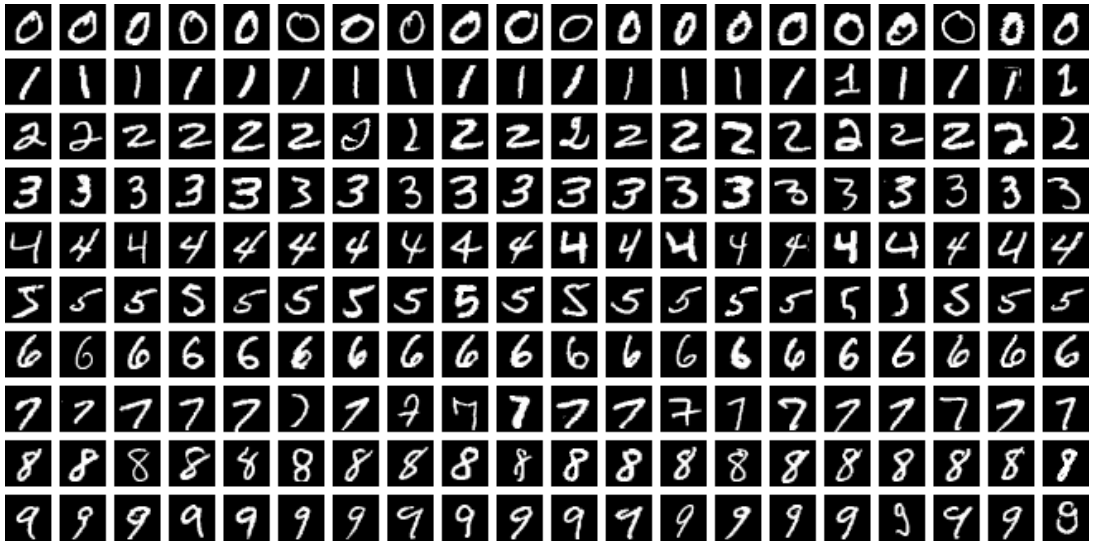
MLP Performance

CONTEXT

How do these models compare?

Model	Test Acc.	Train Time	Test Time
Logistic Reg.	92.55%	176.6s	0.01s
SVM (RBF)	96.85%	2.9s	7.74s
Decision Tree	87.58%	8.2s	0.01s
kNN (k=5)	96.88%	0.1s	3.58s
Neural Net.	**97.92%**	45.9s	0.06s

One-shot benchmark using standard sklearn implementations (no hyperparameter tuning). Python notebook generated by Claude Code.



Benchmark experiment Inspired by Michael Nielsen

Questions?

that's it for today!

SUMMARY

- Neural networks consist of layers of neurons (perceptrons with nonlinear activation functions)
- Neural networks are trained via gradient descent, using backpropagation (chain rule) to calculate gradients

UPCOMING

- Deep Learning
- Natural Language Processing
- Large Language Models

REMINDERS

- Do **Homework 4** (due 12.7)
- Do **Project 4** (due 12.11)