School of COMPUTING
Clemson University

Introduction to Artificial Neural Networks

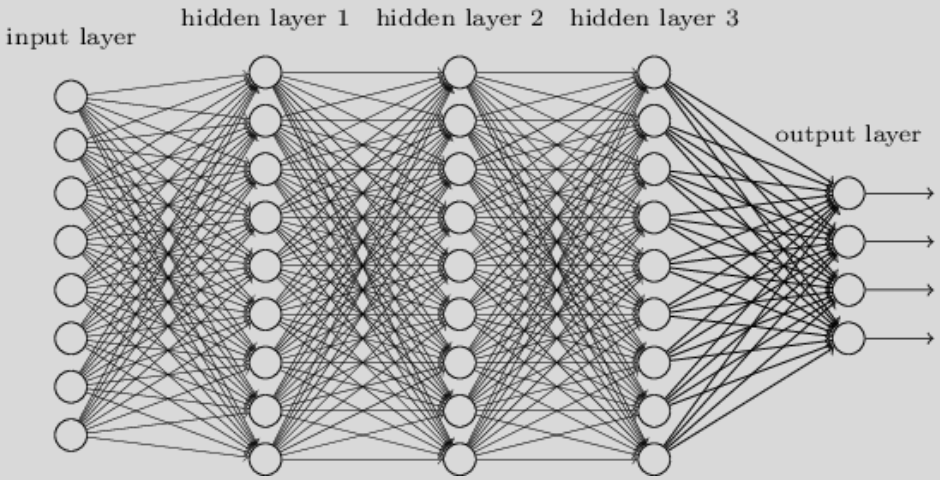
Aaron J. Masino, PhD
Associate Professor, School of Computing

Outline

- Overview and history of artificial neural networks (ANNs or NNets)
- Motivation for ANNs
- Terminology and structure of an ANNs
- Design choices
- Parameter learning with backpropagation
- PyTorch introduction

School of COMPUTING
Clemson University

Overview and history

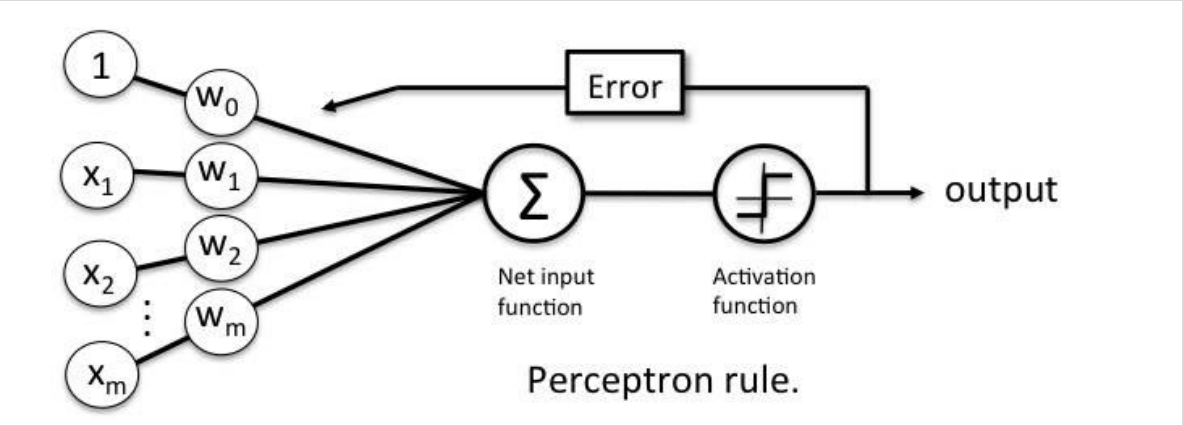


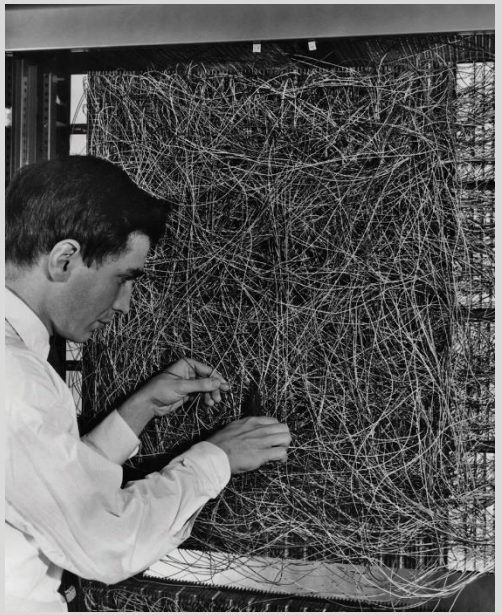
Overview

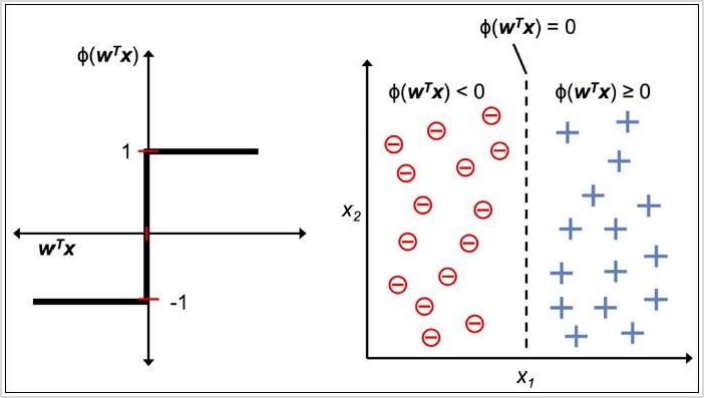
- ANNs are collections of computation nodes arranged in specified architecture
- Each computation node performs a mathematical operation – usually analogous to a logistic regression
- In principle, ANNs can model any continuous function to any degree of accuracy
- Have proven to be very powerful for complex AI problems – particularly classification and generative AI

Perceptron

- **Frank Rosenblatt** – Cornell Aeronautical Laboratory - (1958):
  - Mark I Perceptron (1960)
  - First computer → learn new skills by trial and error

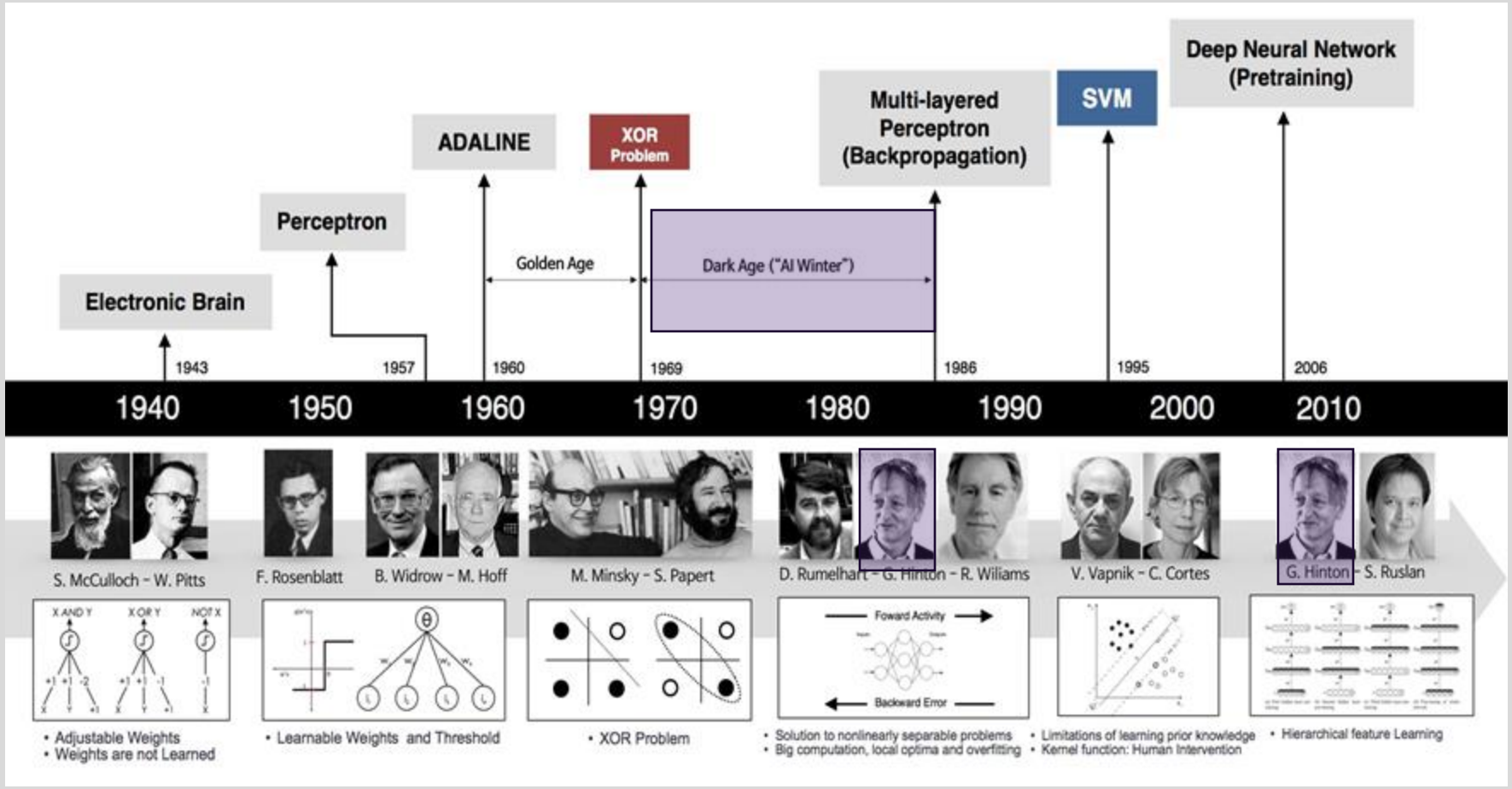




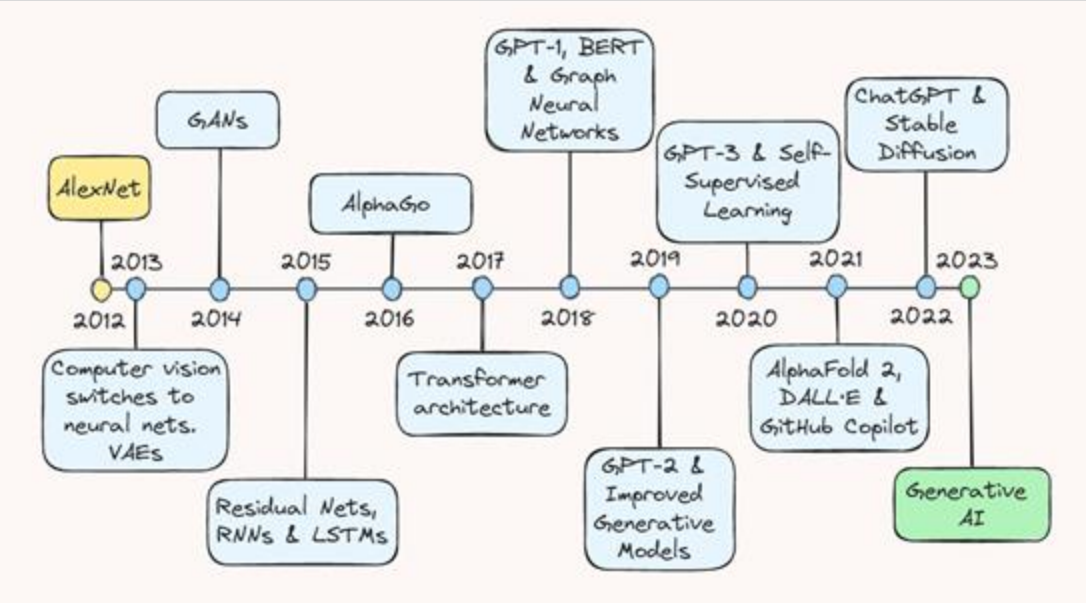


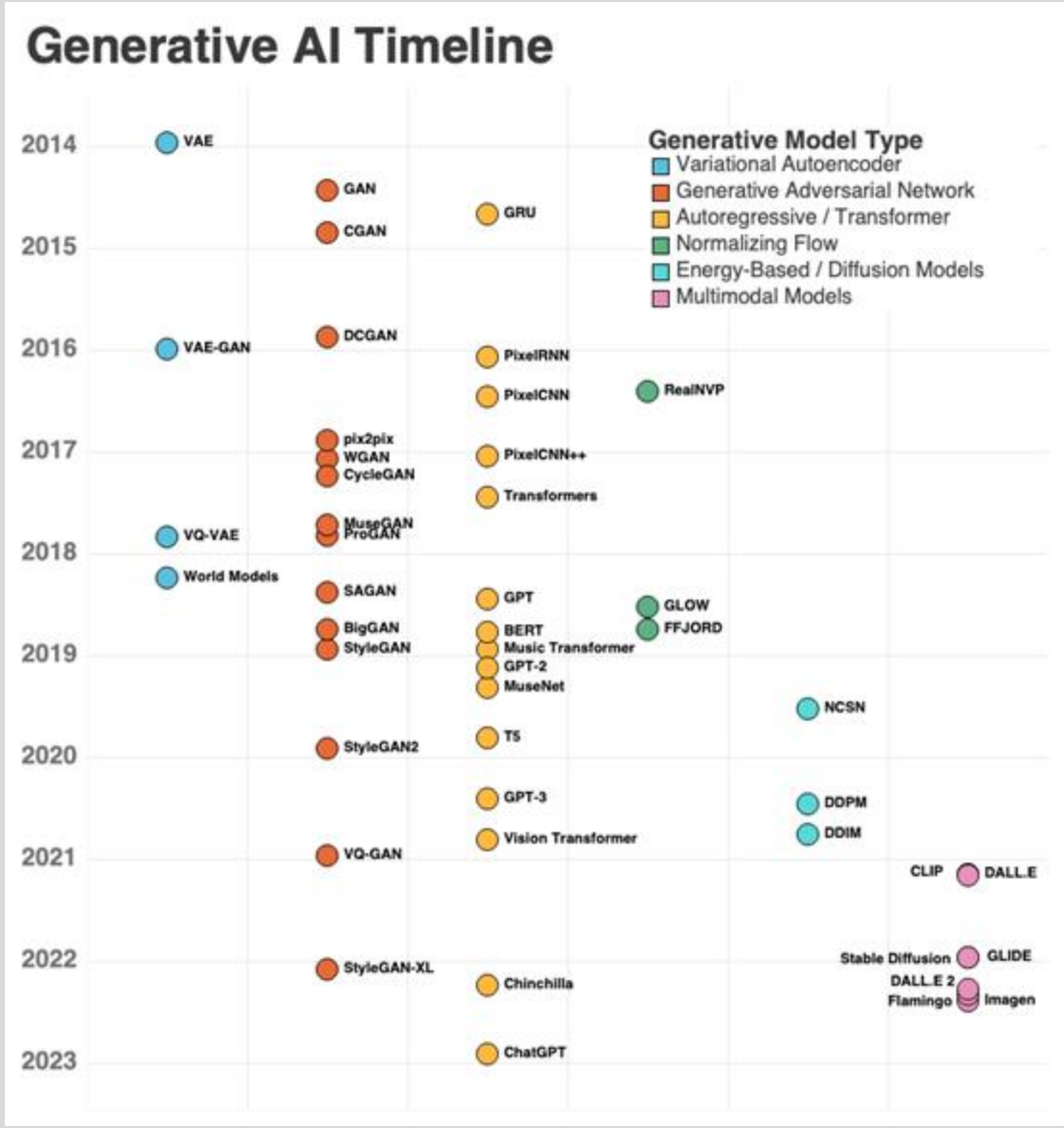
$$0 = \begin{cases} 1 \text{ if } \sum_i wiIi + \theta > 0 \\ 0 \text{ otherwise} \end{cases}$$

Historical Trends



Deep Learning Timeline





https://twitter.com/davidADSP/status/1609350313097695235
https://blog.pulze.ai/2023-in-generative-ai-comprehensive-recap/

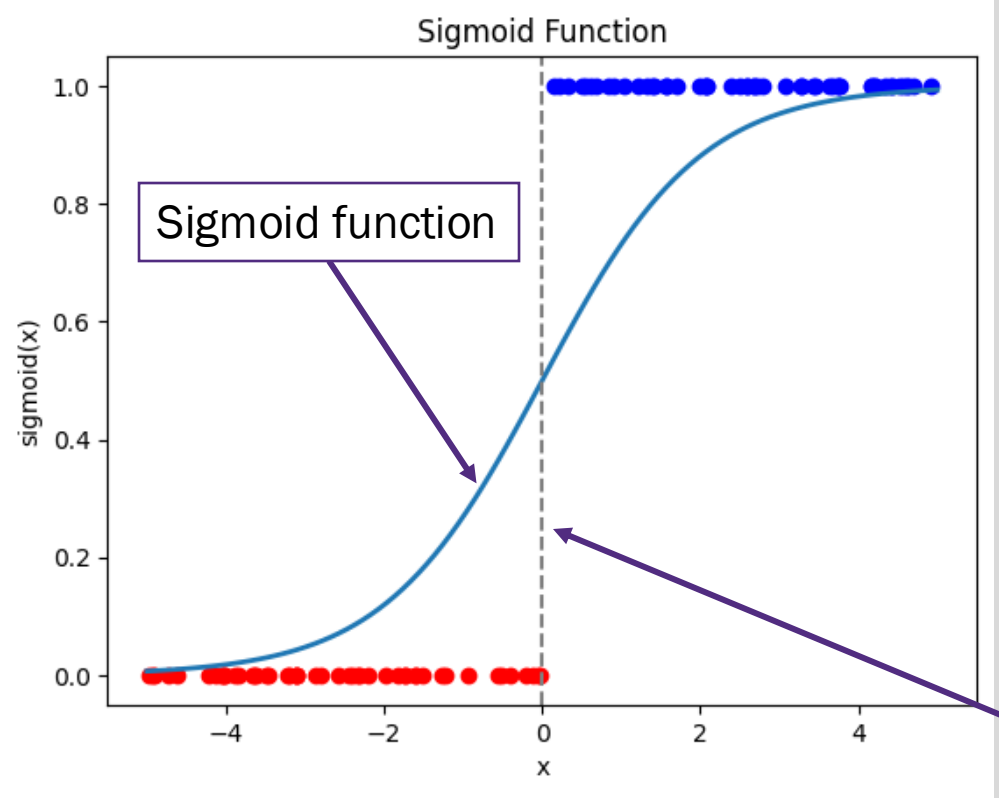
School of COMPUTING
Clemson University

Motivation for ANNs

Why do we need ANNs?

Let's revisit logistic regression classification with one predictor variable

$$P(y = 1|X) = \sigma(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$



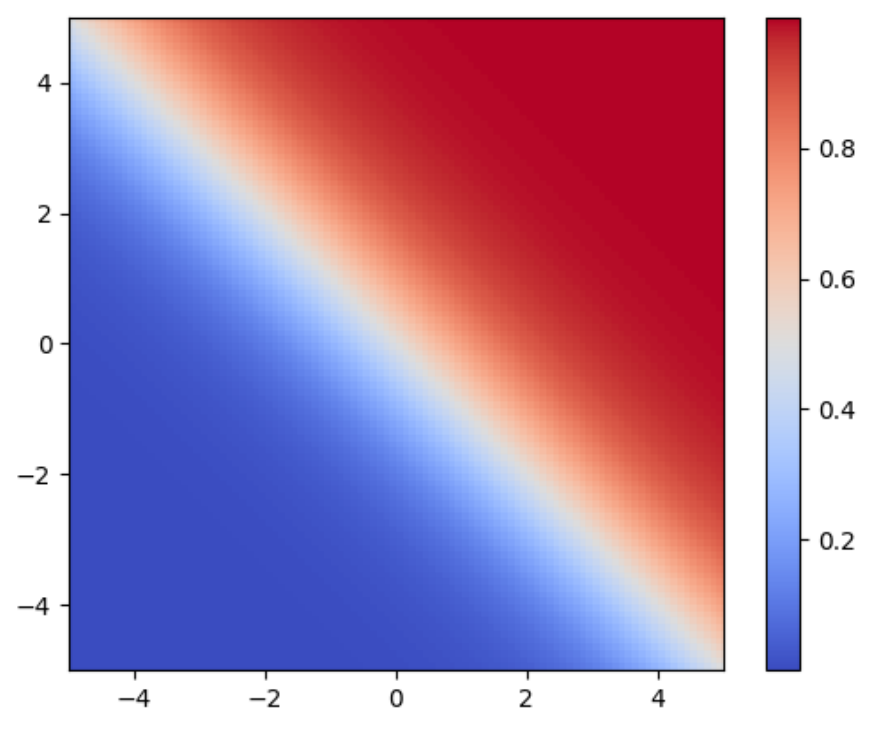
- To perform classification, we pick a threshold, γ (typically 0.5)
- For a given sample, $\tilde{x}$, we *decide*
  - $y = 1$ (positive class) if $\sigma(\tilde{x}) \geq \gamma$
  - $y = 0$ (negative class) if $\sigma(\tilde{x}) < \gamma$
- Without applying any transformation to predictor variables, this produces a *linear* decision boundary

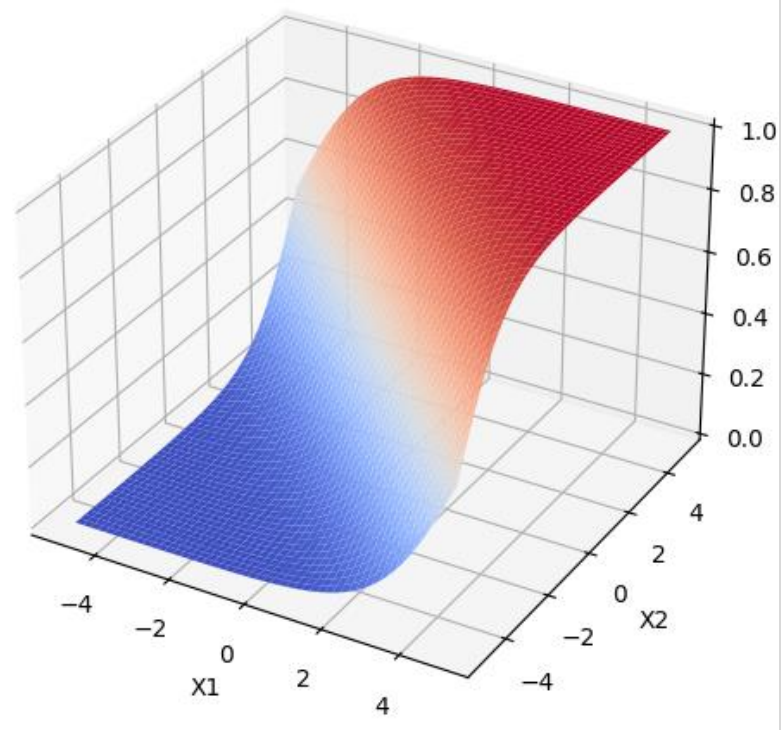
Decision boundary

Why do we need ANNs?

What about logistic regression classification with two predictors?

$$P(y = 1|X) = \sigma(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}$$



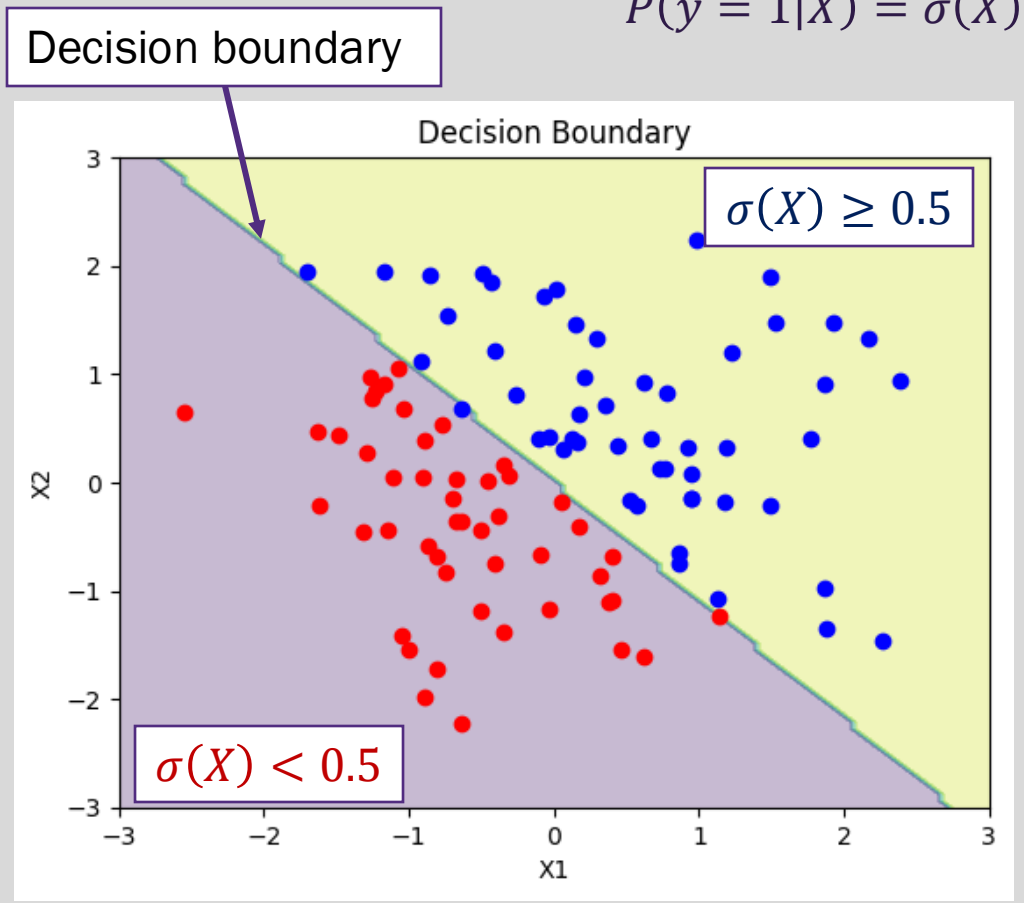


The sigmoid function of two input variables x_1 and x_2 forms a surface in 3D space

Why do we need ANNs?

What about logistic regression classification with two predictors?

$$P(y = 1|X) = \sigma(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}$$

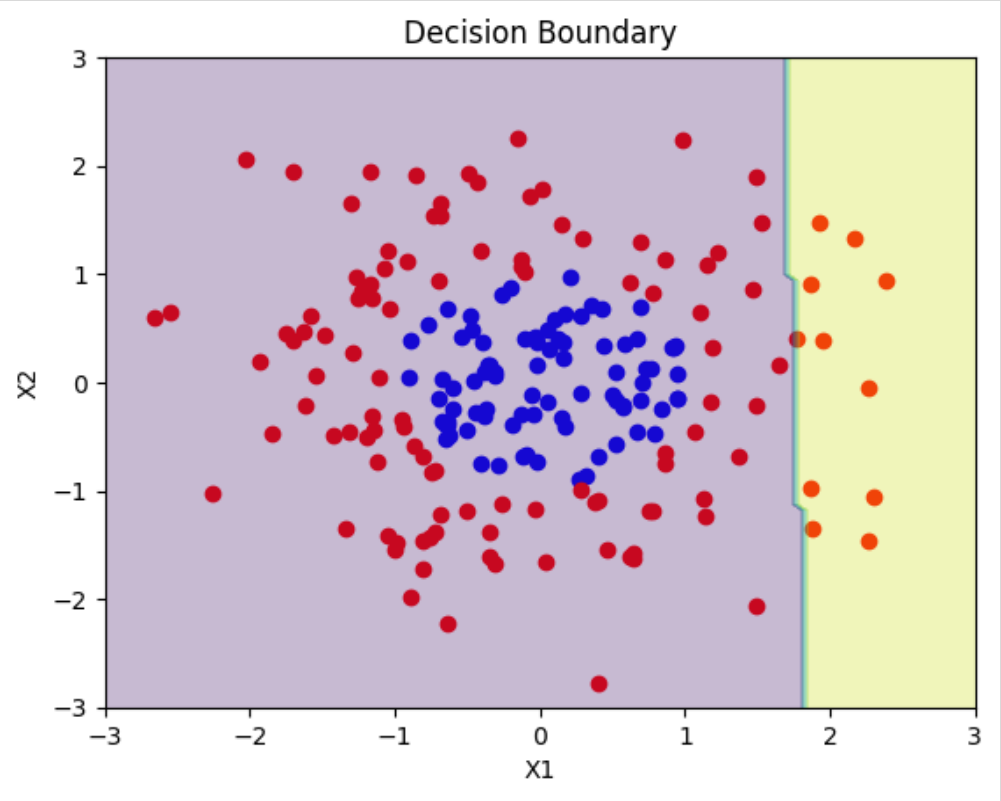


- The classification procedure doesn't change
- We still pick decision threshold, γ
- For a given sample, $\tilde{x}$, we *decide*
  - $y = 1$ (positive class) if $\sigma(\tilde{x}) \geq \gamma$
  - $y = 0$ (negative class) if $\sigma(\tilde{x}) < \gamma$
- Again, if we do not transform the predictor variables, decision boundary is linear

Why do we need ANNs?

What about logistic regression classification with two predictors?

$$P(y = 1|X) = \sigma(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}$$



- What if our classes are **not** linearly separable?
- Are we stuck?
- What if we add polynomial transformations?

$$x_3 \rightarrow x_1^2 \text{ and } x_4 \rightarrow x_2^2$$

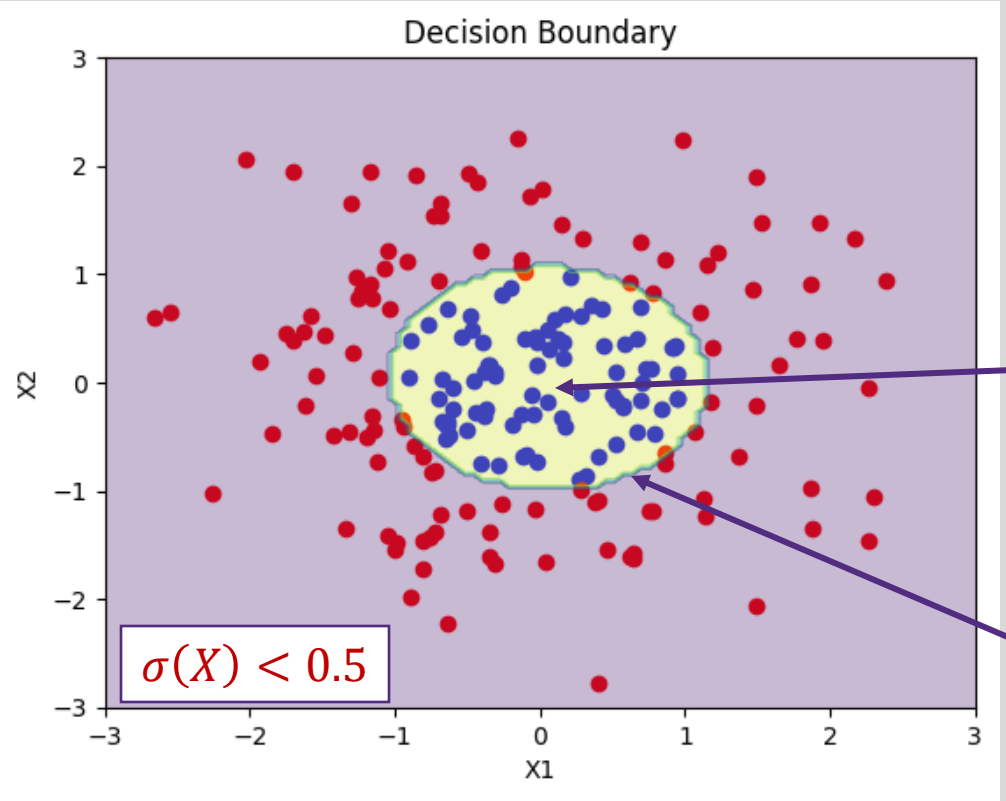
And and interaction term

$$x_5 \rightarrow x_1 x_2$$

Why do we need ANNs?

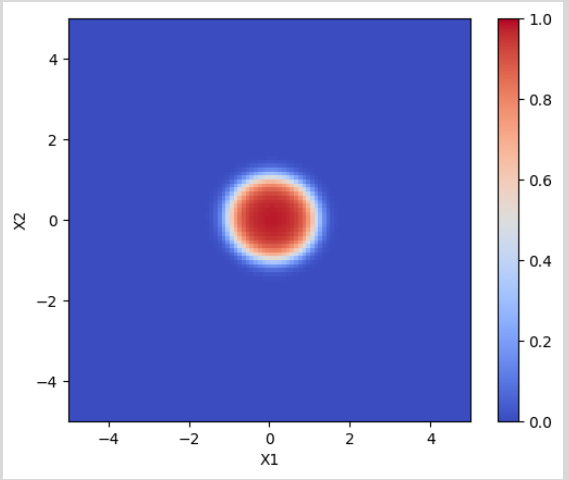
What about logistic regression classification with two predictors?

$$P(y=1|X) = \sigma(X) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\beta_3 x_1^2+\beta_4 x_2^2+\beta_5 x_1 x_2)}}$$



- Adding nonlinear feature transformations and interaction terms allows logistic regression to form a nonlinear decision boundaries

$$\sigma\left(\beta_0 + \sum_{k=1}^{5} \beta_k x_k\right)$$



Why do we need ANNs?

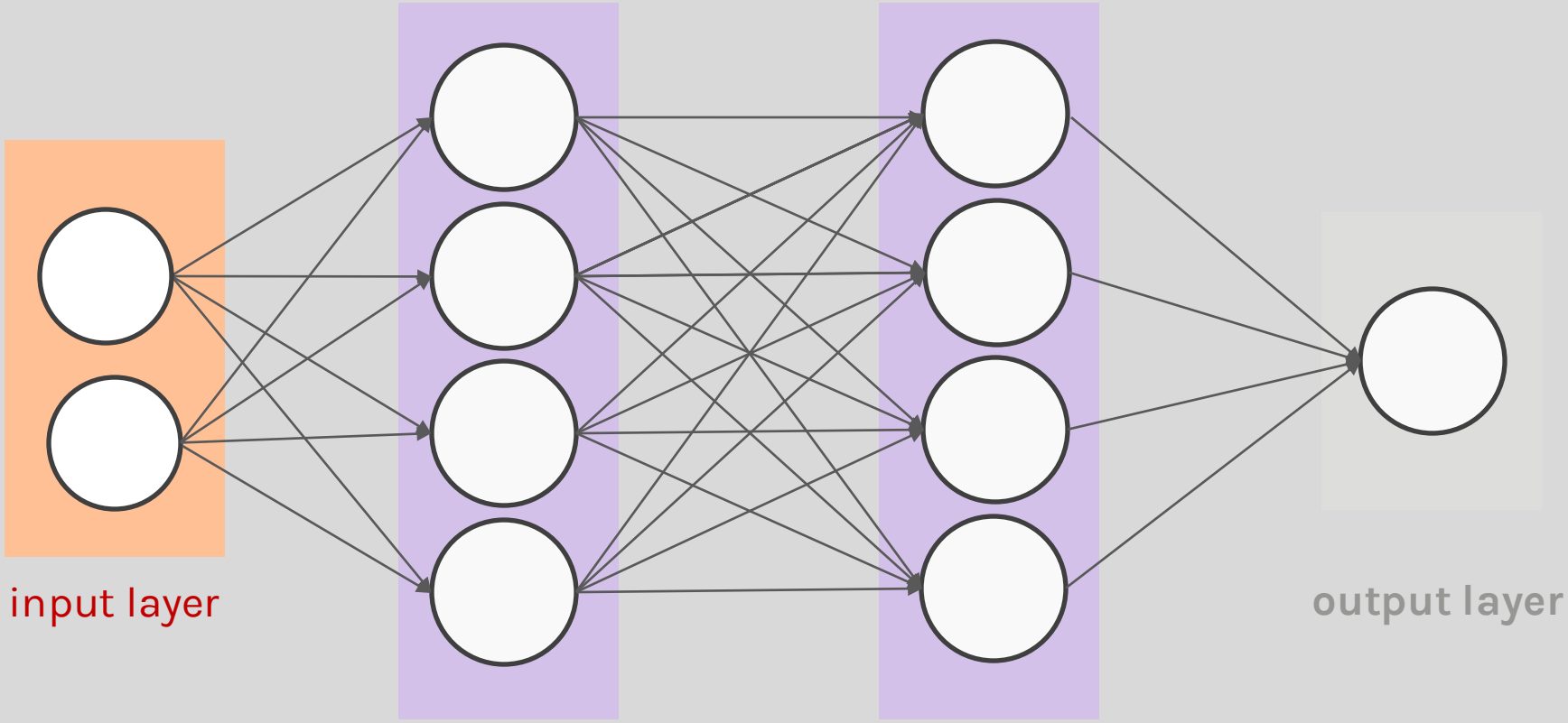
- Adding nonlinear feature transformations and interaction terms allows logistic regression to form a nonlinear decision boundaries
- Can we just add these whenever we need them?
- What if we have many features?
  - How should we select the transformations?
  - How should we select the interactions?
  - Can these be learned from the data? Yes, with ANNs.

School of COMPUTING
Clemson University

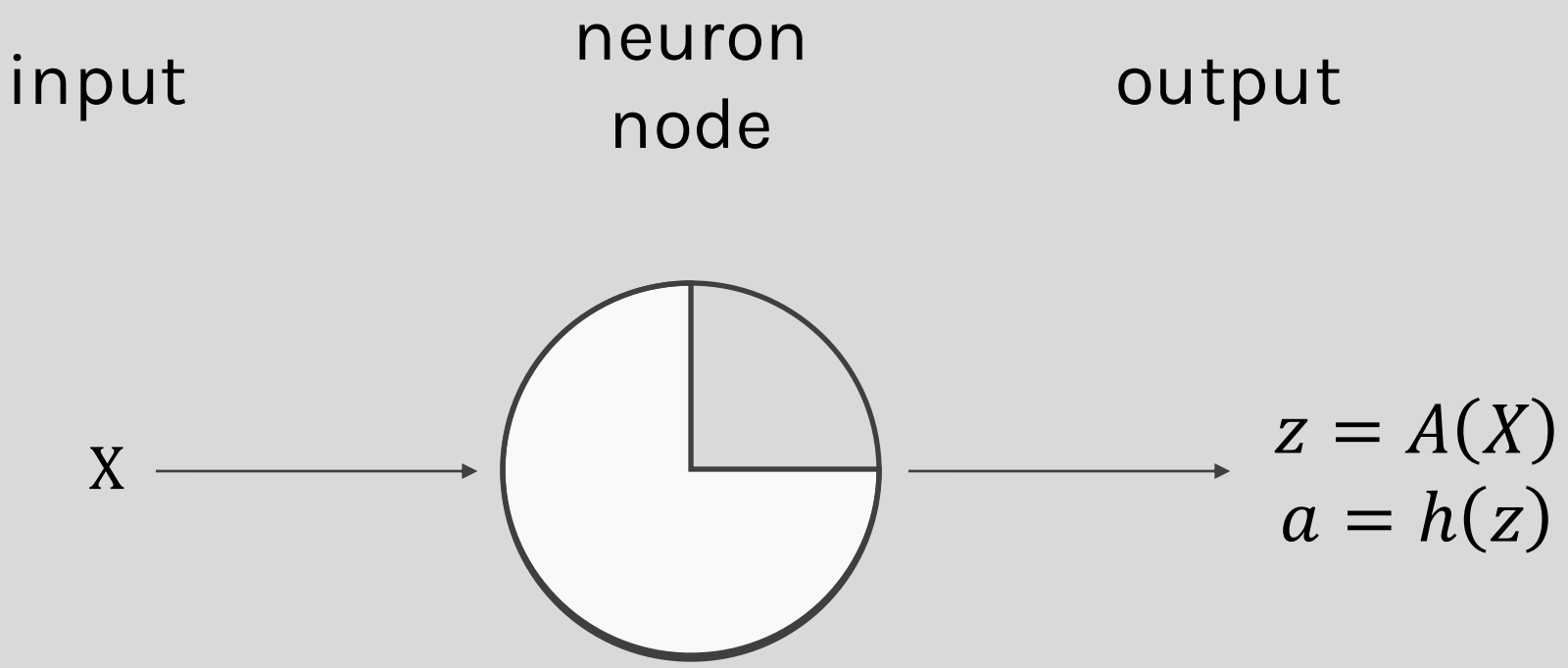
ANN Terminology & Structure

School of COMPUTING Clemson University

Anatomy of artificial neural network (ANN)



Anatomy of artificial neural network (ANN)

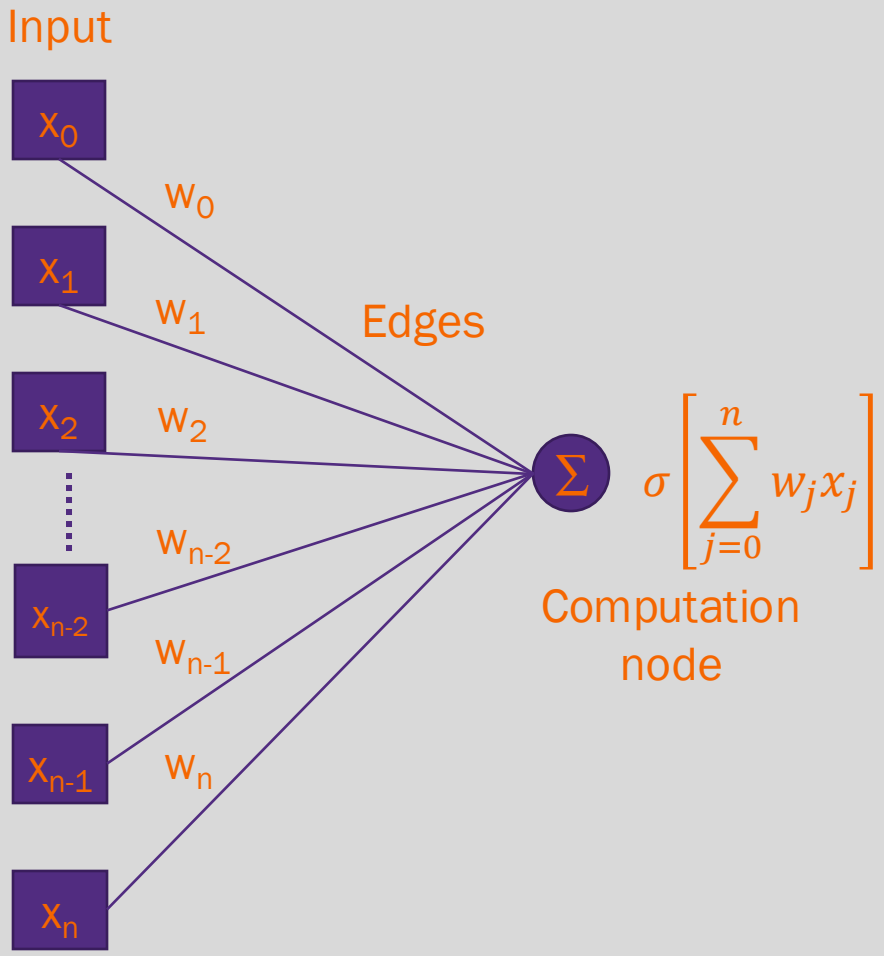


A is an affine transformation (typically, buy not always, a weighted sum)

h is a non-linear activation function (e.g., sigmoid)

Node computation and visual representation

- Neural networks are typically visualized as a computation graph
- Edges connect inputs to computation nodes
- Edges usually denote a weight parameter
- Computation nodes apply the affine transform and activation



One hidden layer

- This layer is fully connected (not all are)
- Hidden layer of multiple computation nodes

$$z_j^{(1)} = \left[\sum_{j=0}^{n} w_{j,k}^{(1)} x_j\right]$$

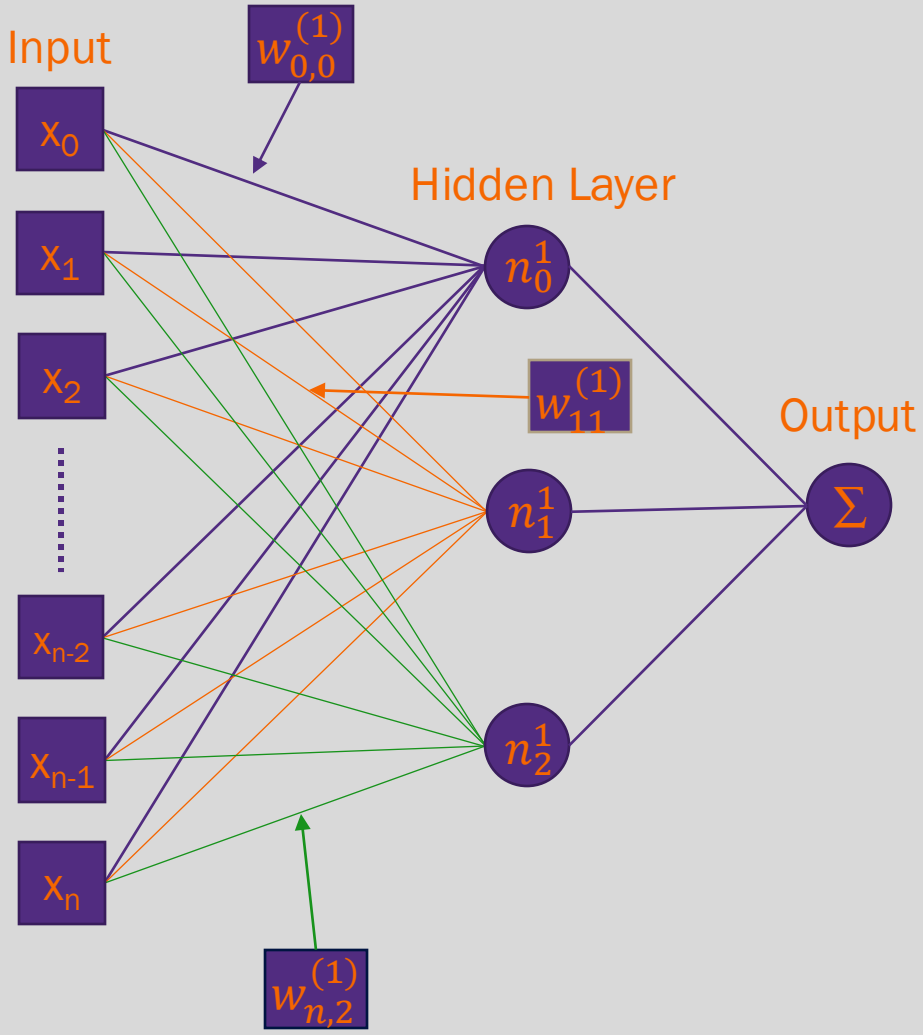
$$a_j^{(1)} = h\left(z_j^1\right)$$

- Output layer is a weighted linear sum

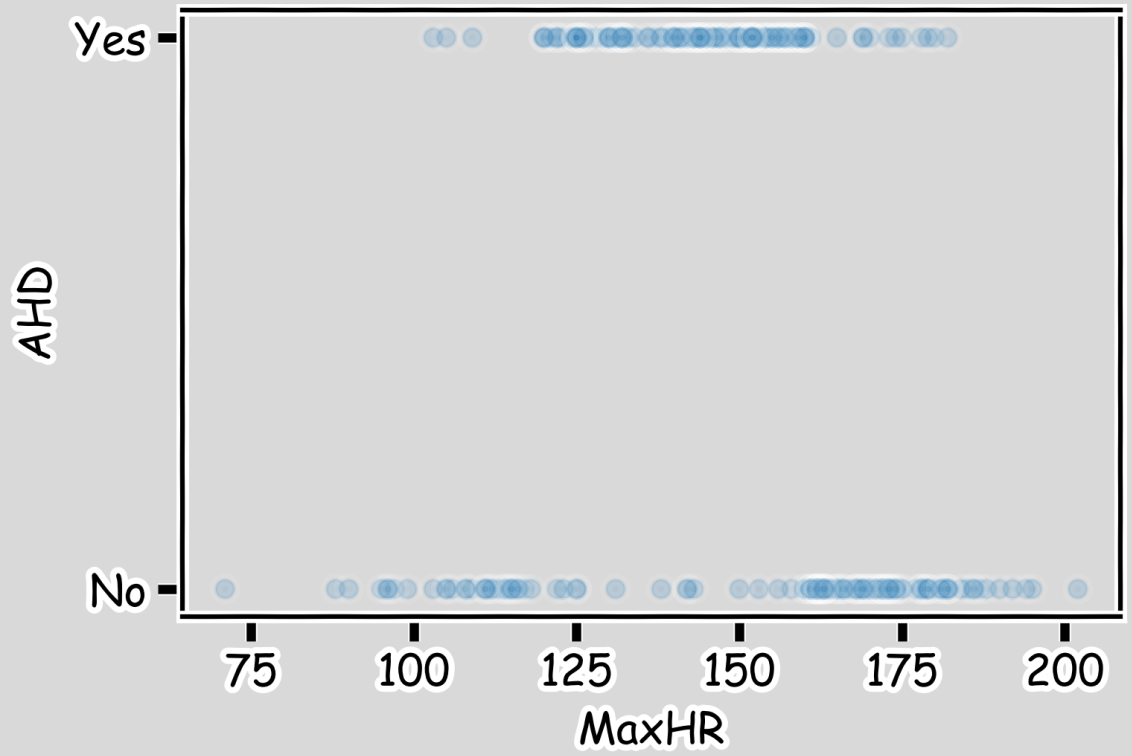
$$z^{(out)} = \left[\sum_{j=0}^{k} w_j^{(2)} a_j^{(1)}\right]$$

- In classification, sigmoid (binary) or softmax (multiclass) function is applied to output

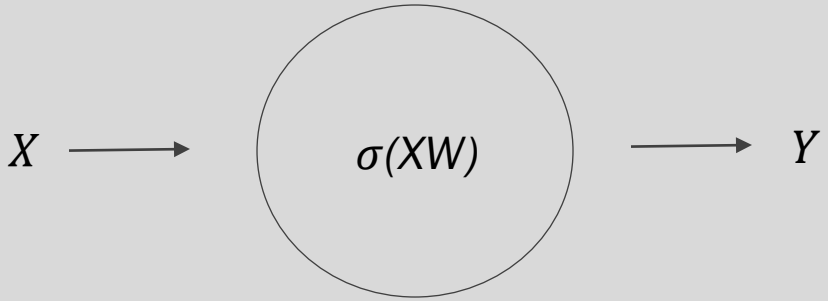
$$\sigma(z) = \frac{1}{1+e^{-z}} \quad \sigma_{SM}(z_k, \boldsymbol{z}) = \frac{e^{z_k}}{\sum_{j=1}^{M} e^{z_j}}$$

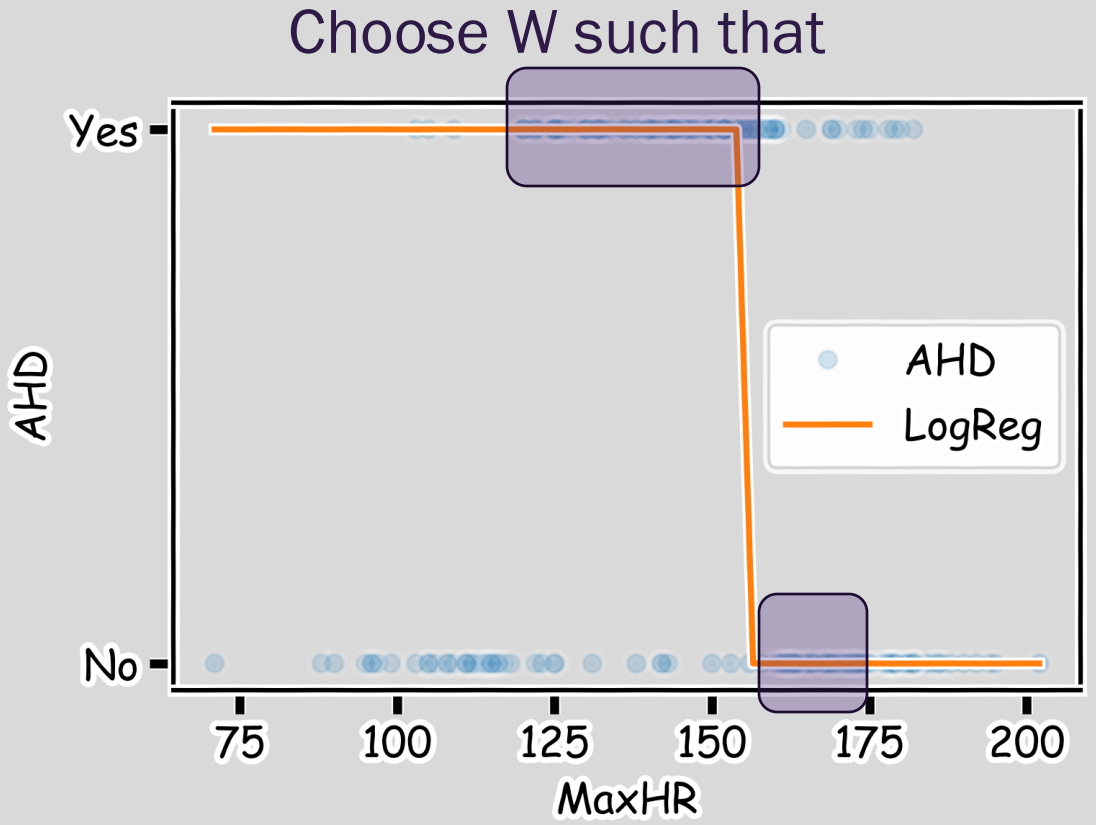


Example Using Heart Data

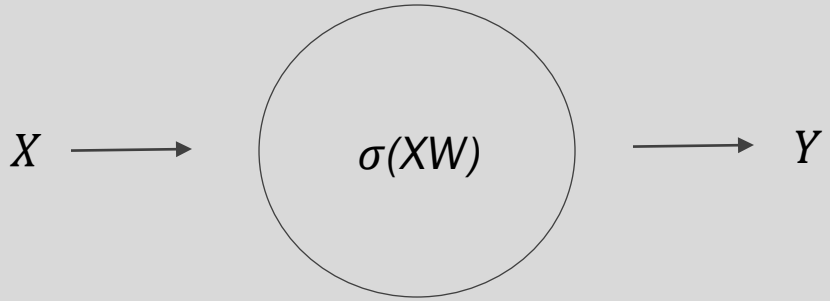


Example Using Heart Data

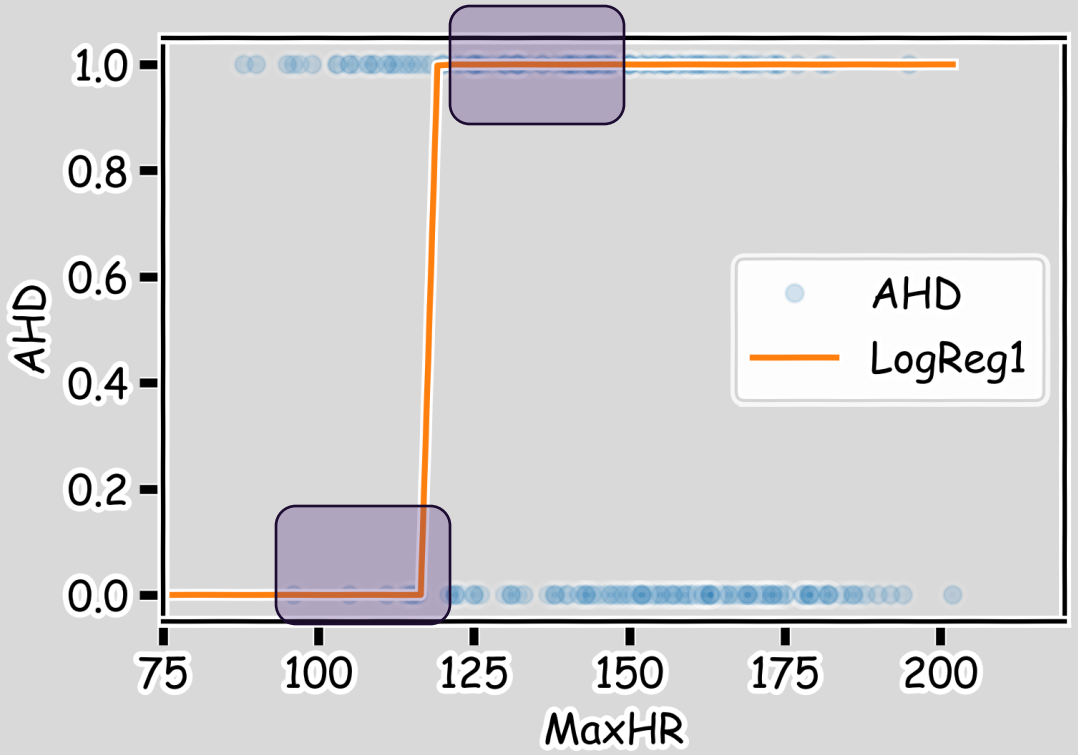




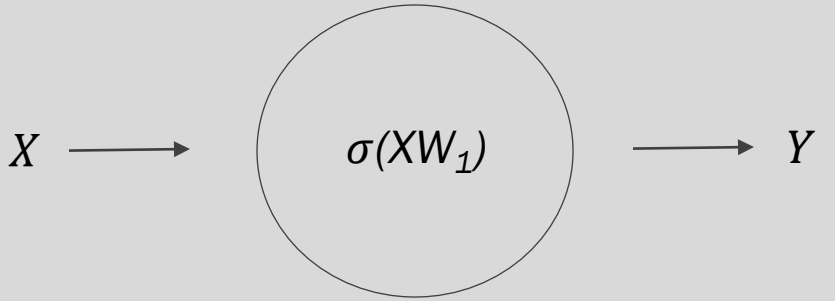
Example Using Heart Data

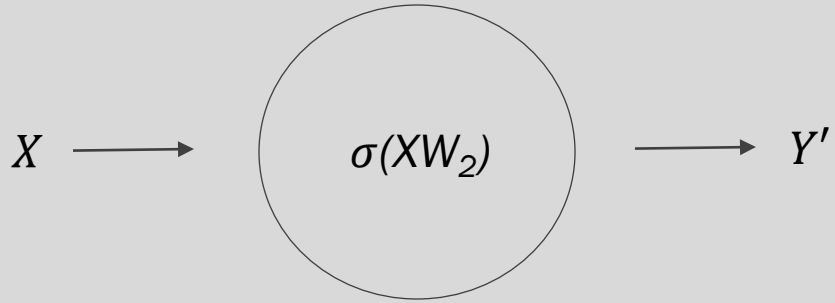


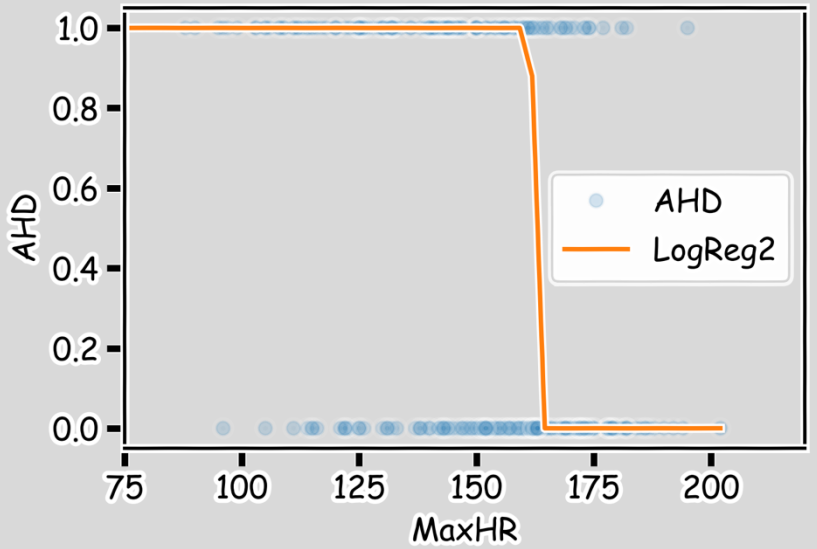
Choose W such that

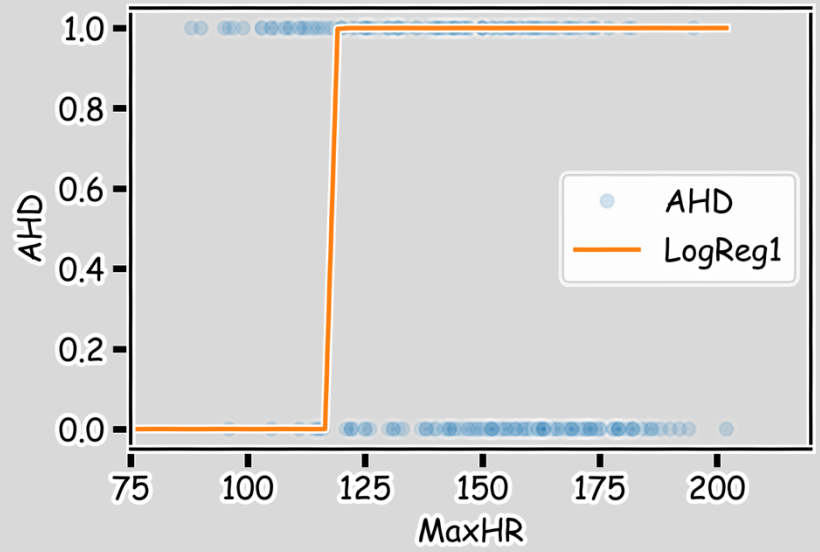


Example Using Heart Data

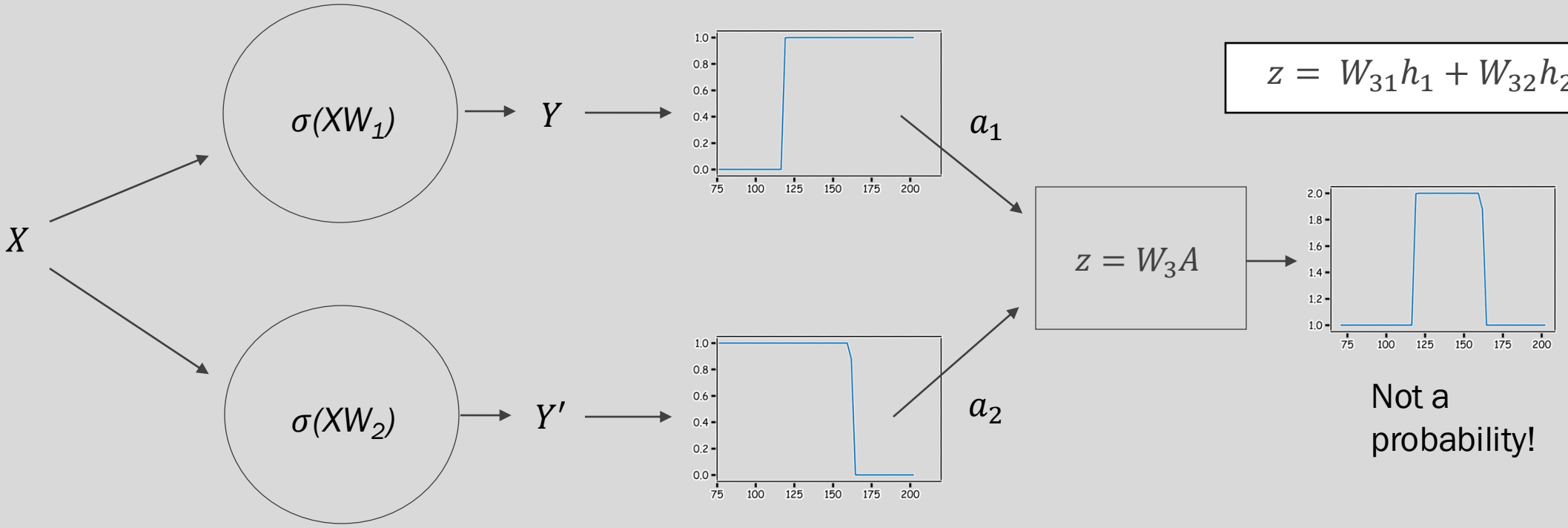




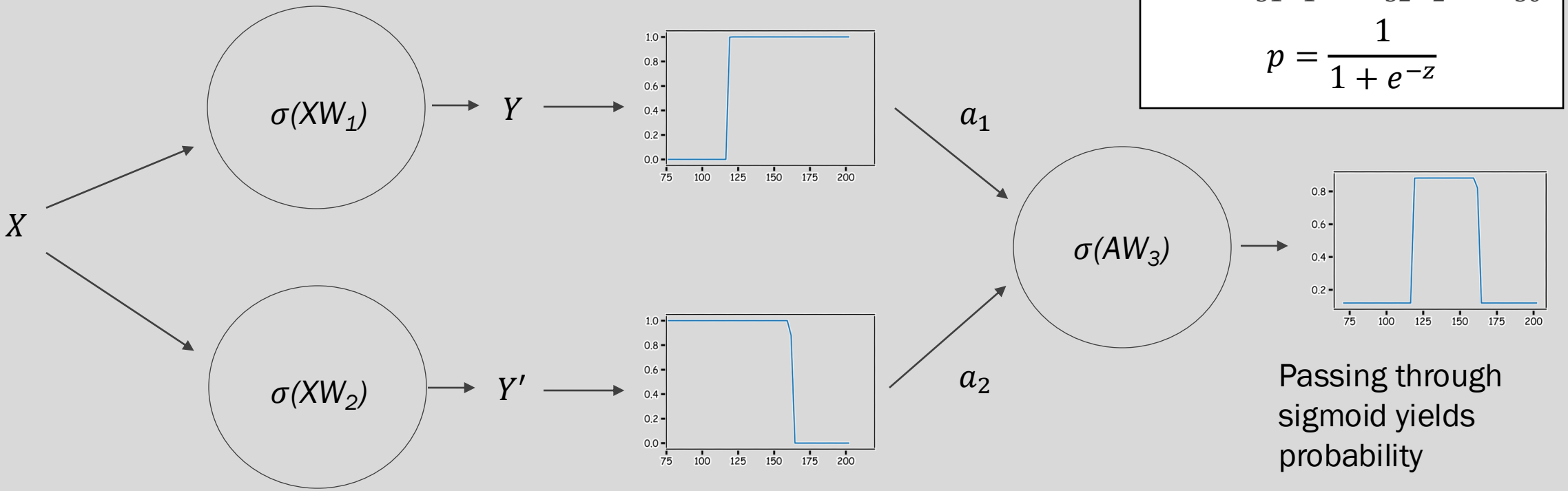




Combining Neurons



Combining Neurons



$$z = W_{31}h_1 + W_{32}h_2 + W_{30}$$

$$p = \frac{1}{1 + e^{-z}}$$

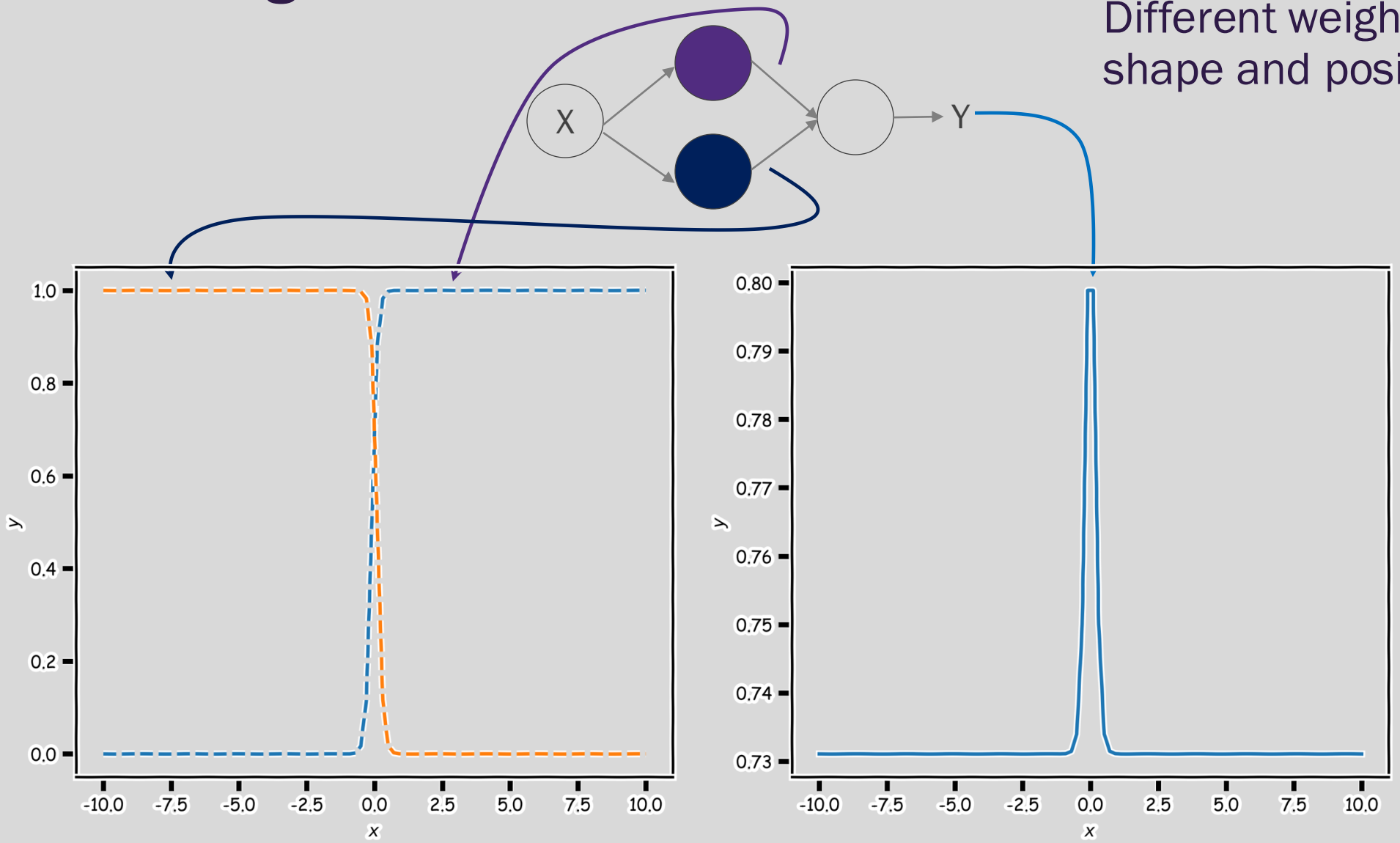
Passing through sigmoid yields probability

$$L = -y\ln(\mathrm{p}) - (1 - \mathrm{y})\ln(1 - \mathrm{p})$$

Need to learn W1, W2 and W3.

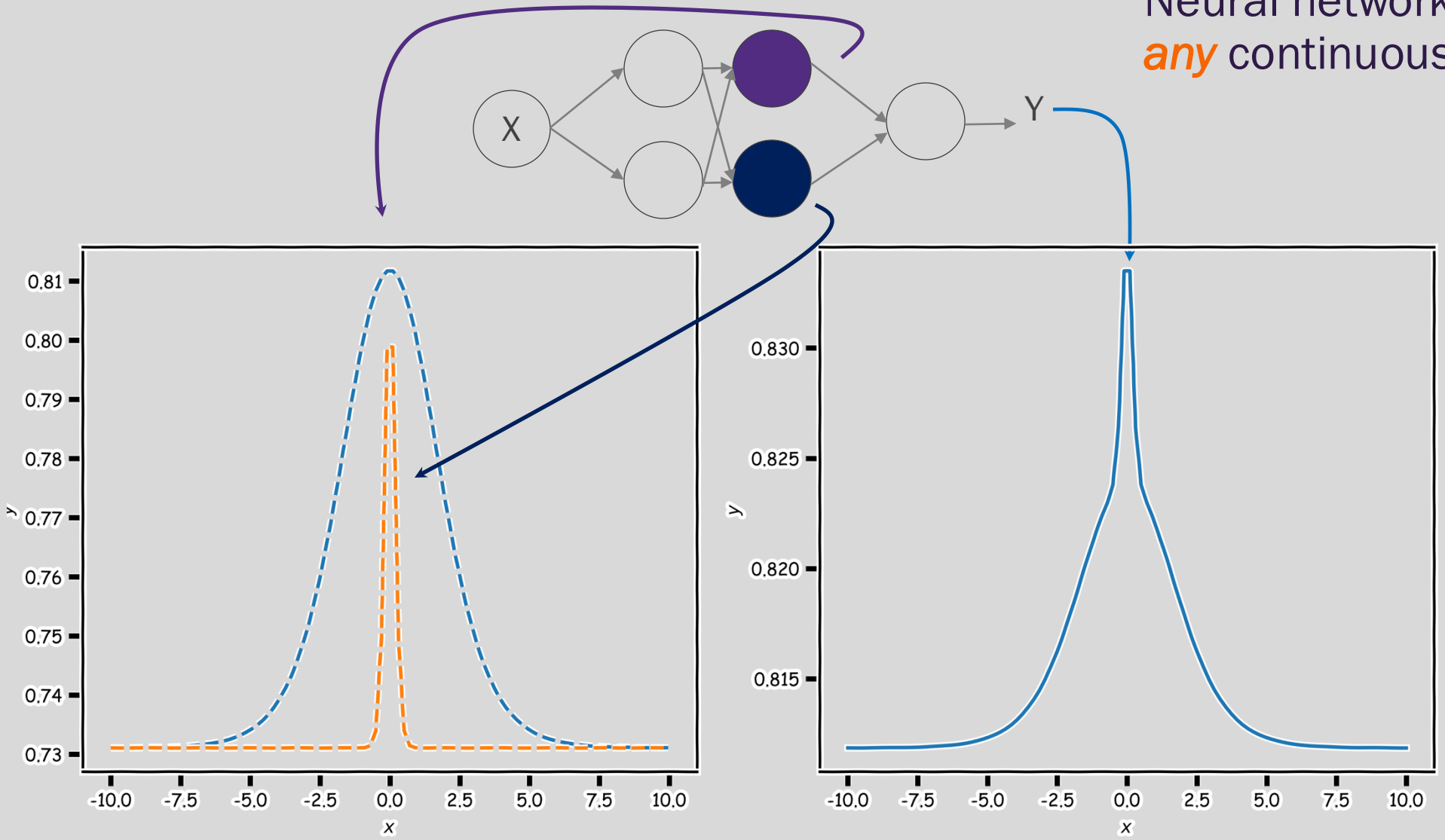
Combining Neurons

Different weights change the shape and position



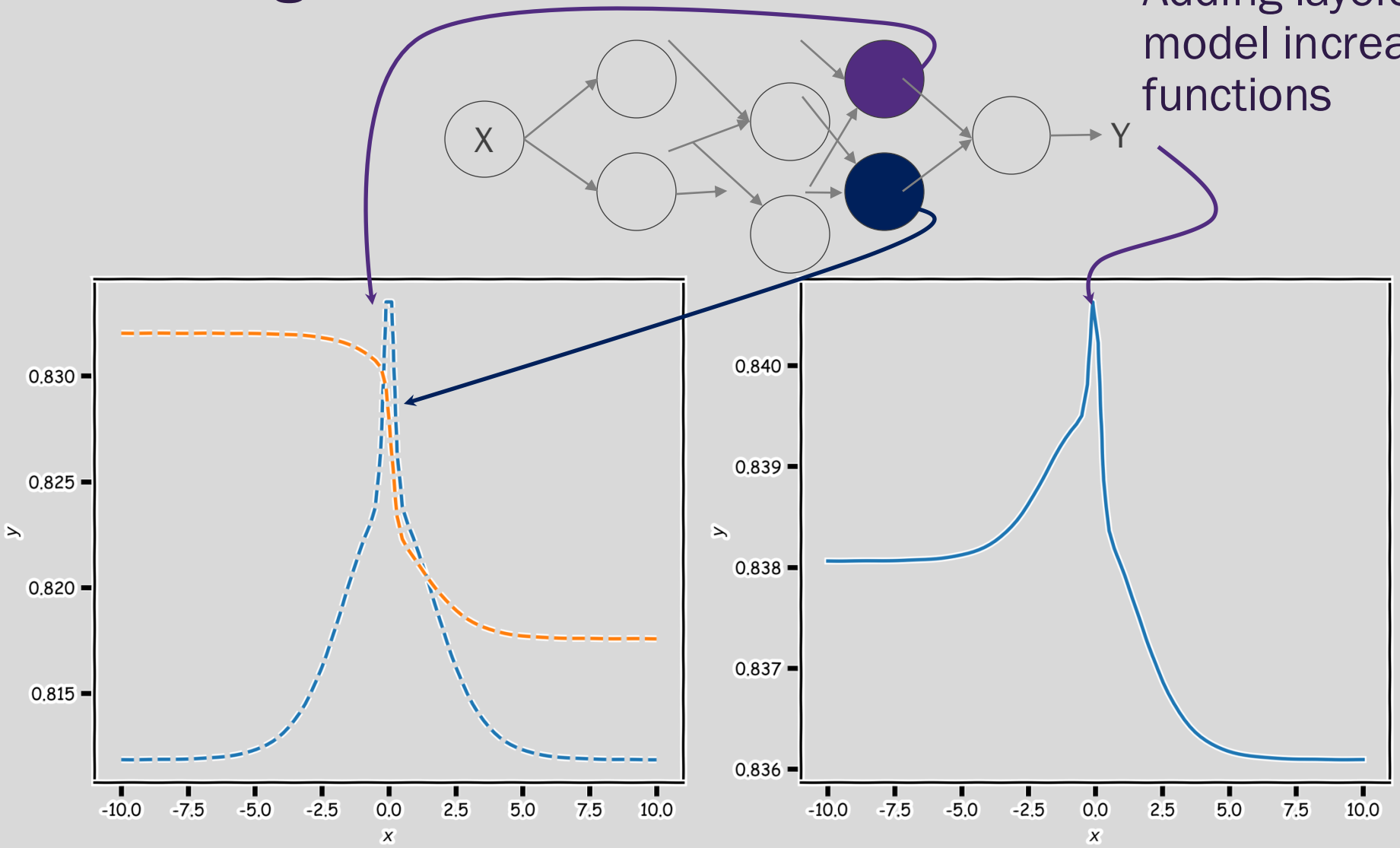
Combining Neurons

Neural networks can model *any* continuous function!



Combining Neurons

Adding layers allows us to model increasingly complex functions



School of COMPUTING
Clemson University

Design Choices – loss function

Loss Function

Likelihood for a given point:

$$p(y_i|W;x_i)$$

Assume independency, likelihood for all measurements:

$$L(W;X,Y) = p(Y|W;X) = \prod_i p(y_i|W;x_i)$$

Maximize the likelihood, or equivalently maximize the log-likelihood:

$$\log L(W;X,Y) = \sum_i \log p(y_i|W;x_i)$$

Turn this into a loss function:

$$\mathcal{L}(W;X,Y) = -\log L(W;X,Y)$$

Loss Function

Examples:

- Distribution is **Normal** then likelihood is:

$$p(y_i|W;x_i) = \frac{1}{\sqrt{\{2\pi^2\sigma\}}} e^{-\frac{(y_i-\hat{y}_i)^2}{2\sigma\text{^}2}}$$

Regression $\mathcal{L}(W;X,Y) = \sum_i (y_i - \hat{y}_i)^2$

- Distribution is **Bernouli** then likelihood is:

$$p(y_i|W;x_i) = p_i^{y_i}(1-p_i)^{1-y_i}$$

Binary $\mathcal{L}(W;X,Y) = -\sum_i [y_i \log p_i + (1-y_i)\log(1-p_i)]$

Categorical

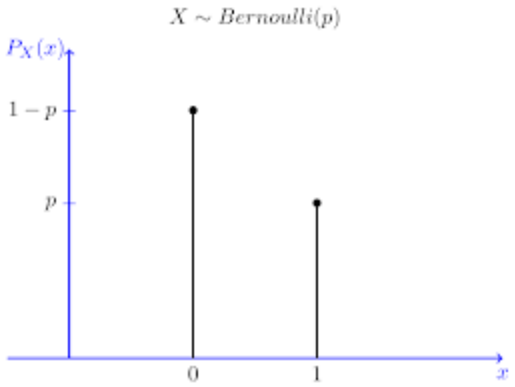
$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i$$

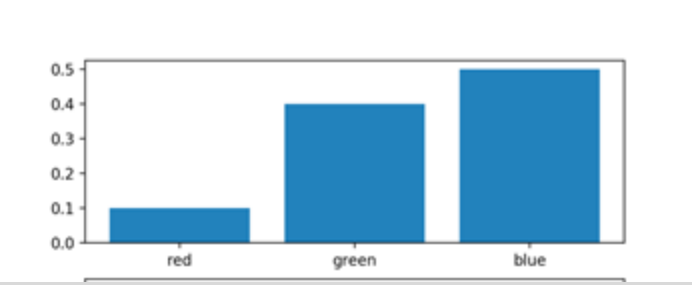
School of COMPUTING
Clemson University

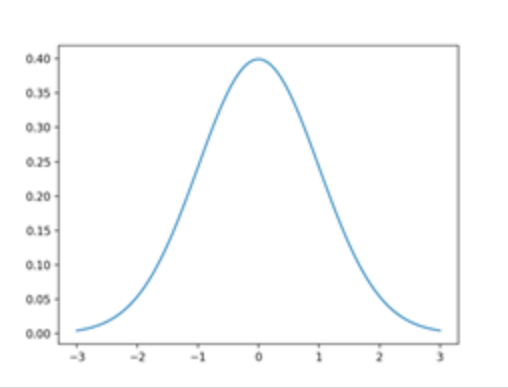
Design Choices – network output

Output Units

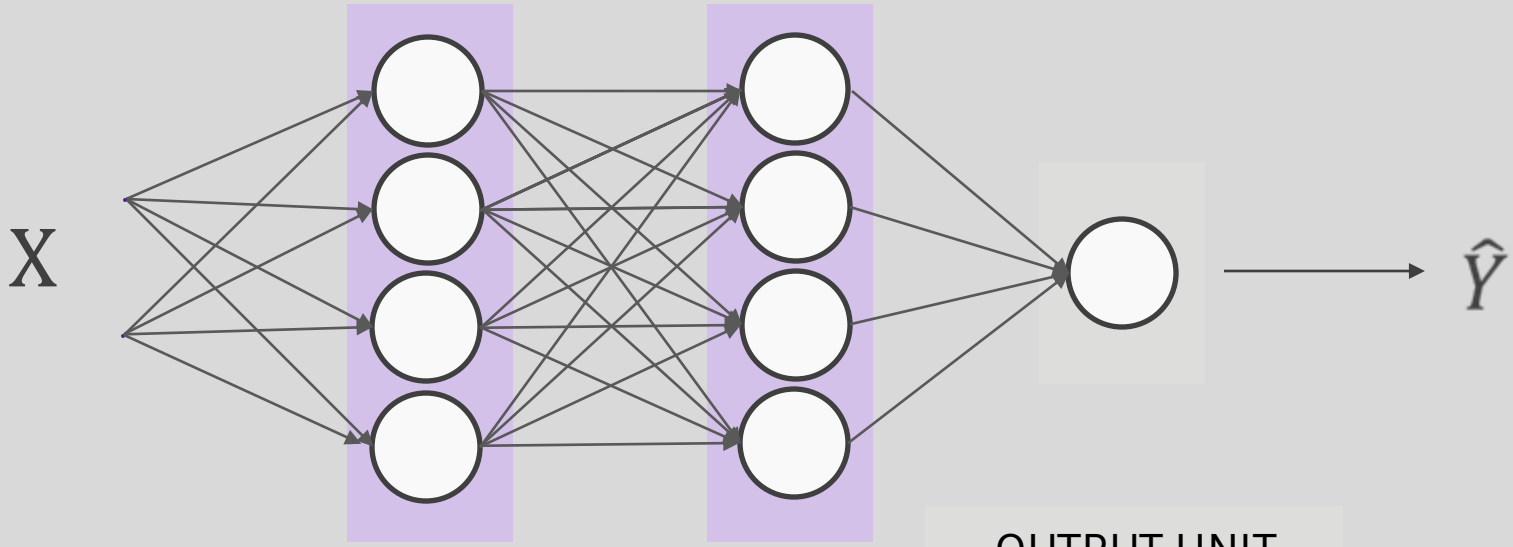
Output Type	Output Distribution	Output layer	Loss Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	Linear	MSE

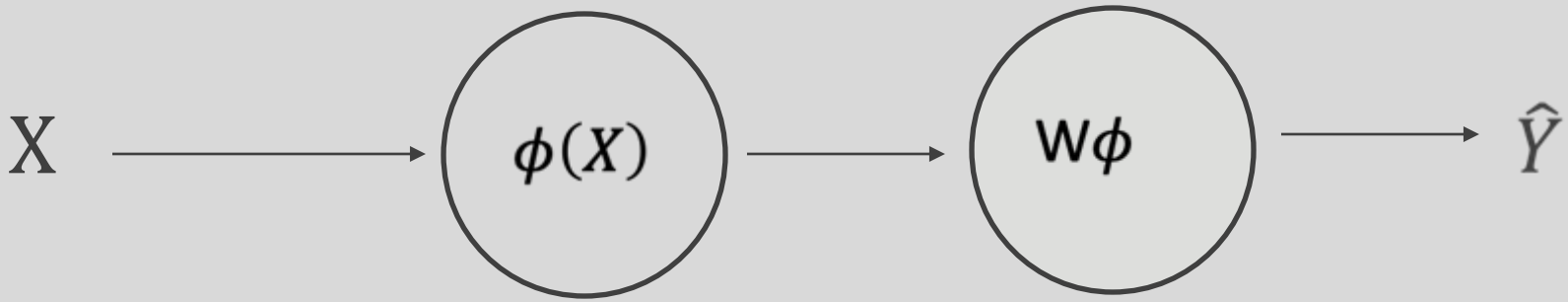






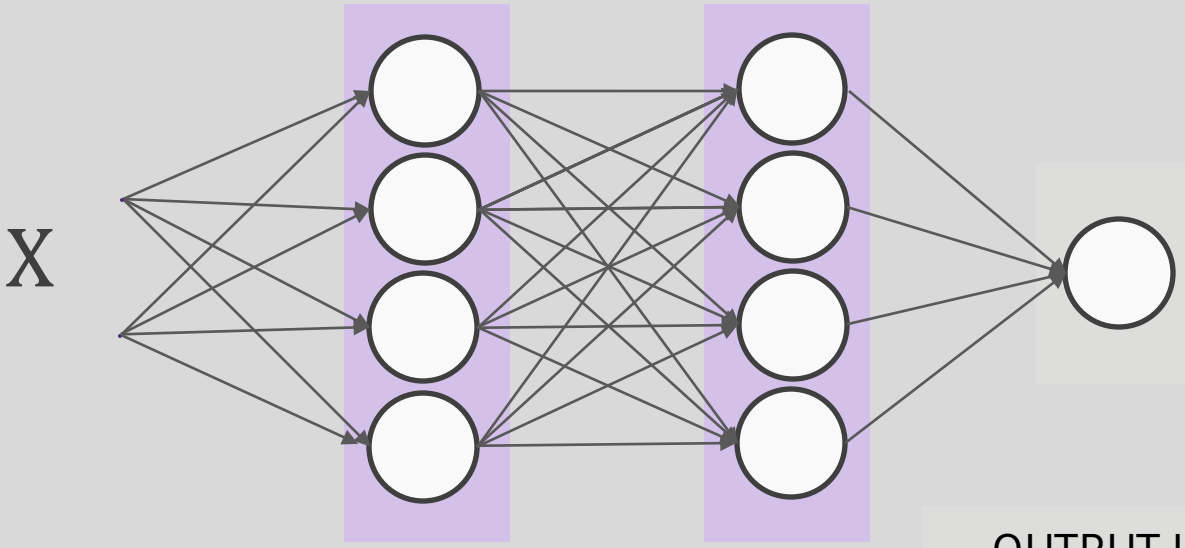
Output unit for regression





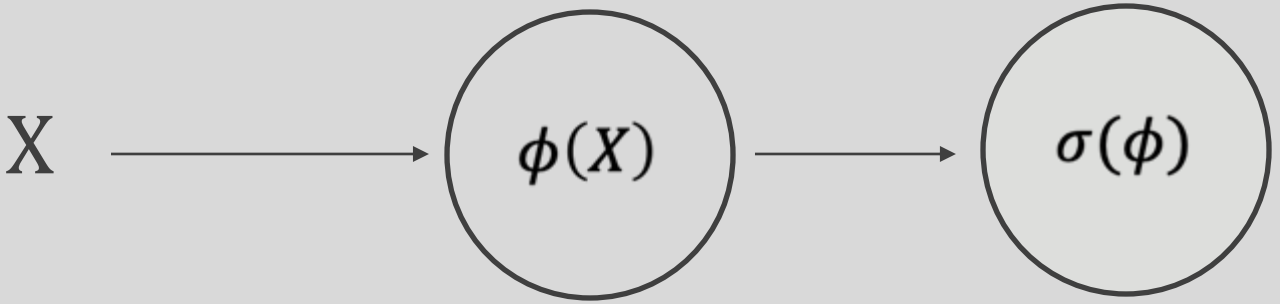
$$X \Longrightarrow \phi(X) \Longrightarrow \hat{Y} = W\phi(X)$$

Output unit for binary classification



$$\hat{Y} = \mathrm{P}(\mathrm{y} = 1)$$

OUTPUT UNIT



$$\hat{Y} = \mathrm{P}(\mathrm{y} = 1)$$

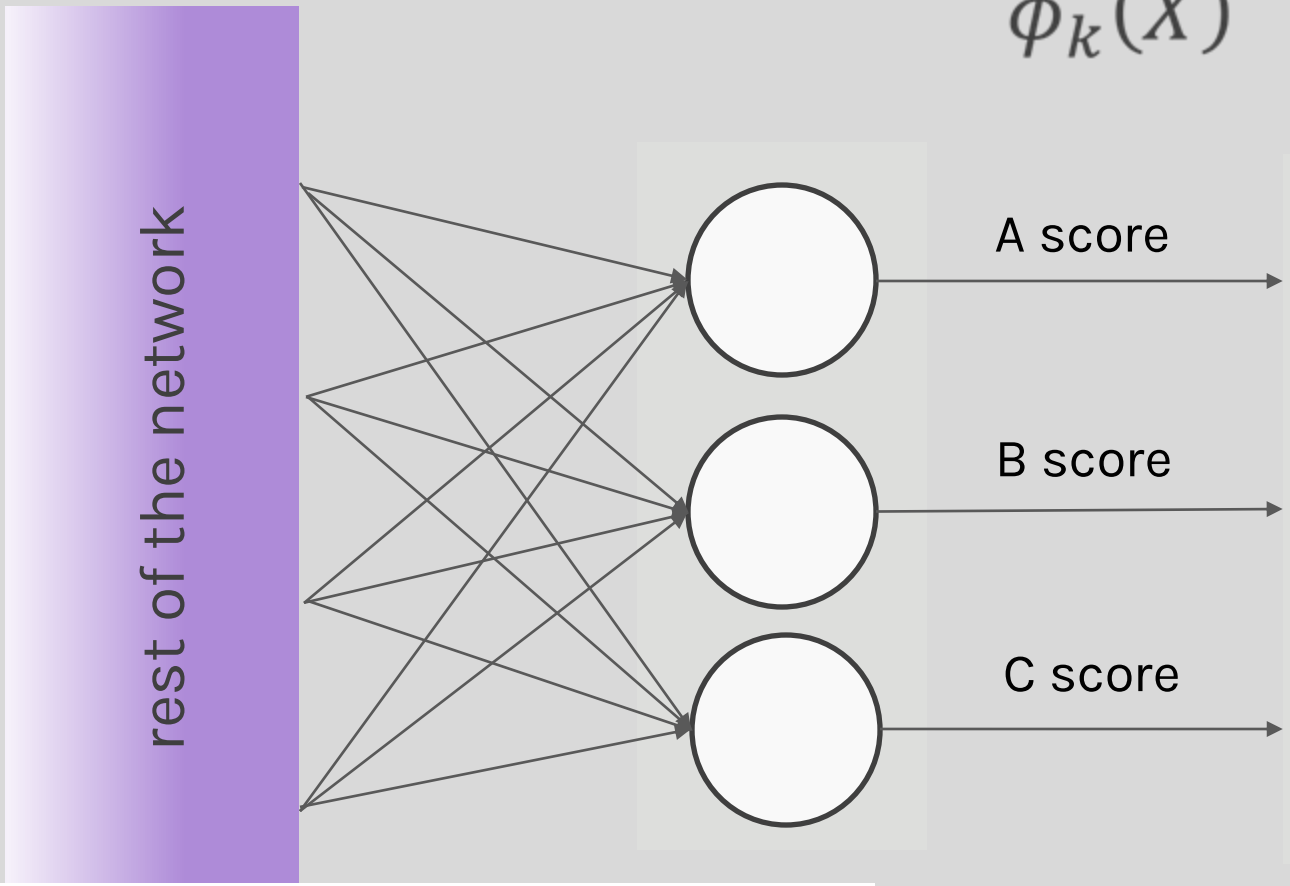
$$X \Longrightarrow \phi(X) \Longrightarrow P(y = 1) = \frac{1}{1 + e^{-\phi(X)}}$$

School of COMPUTING Clemson University

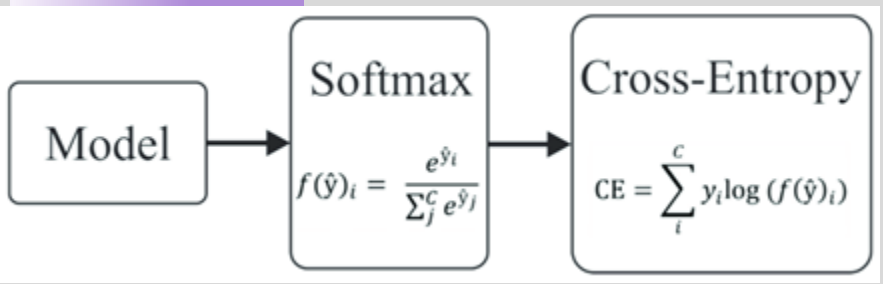
SoftMax

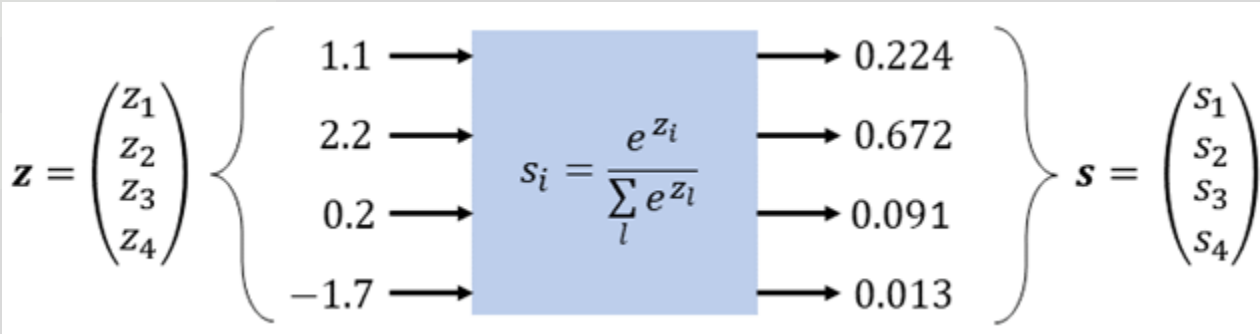
$$\phi_k(X)$$

$$\hat{Y} = \frac{e^{\phi_k(X)}}{\sum_{k=1}^{K} e^{\phi_k(X)}}$$



OUTPUT UNIT





School of COMPUTING
Clemson University

Design Choices – activation function

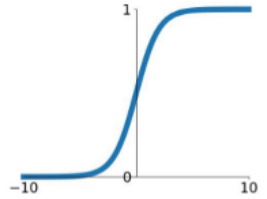
Activation functions

- Nonlinearity necessary to enable modeling nonlinear transformations
- Desirable to have a continuous, locally differentiable function to support model training (more to come)
- sigmoid was dominant in early research but suffers from saturation
- ReLU most popular currently

Sample Activation Functions

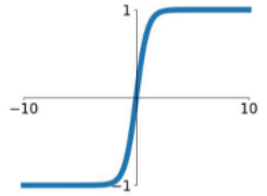
Sigmoid

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



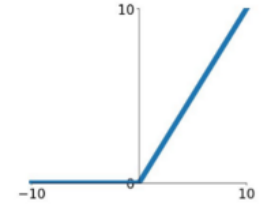
tanh

$$\tanh(x)$$



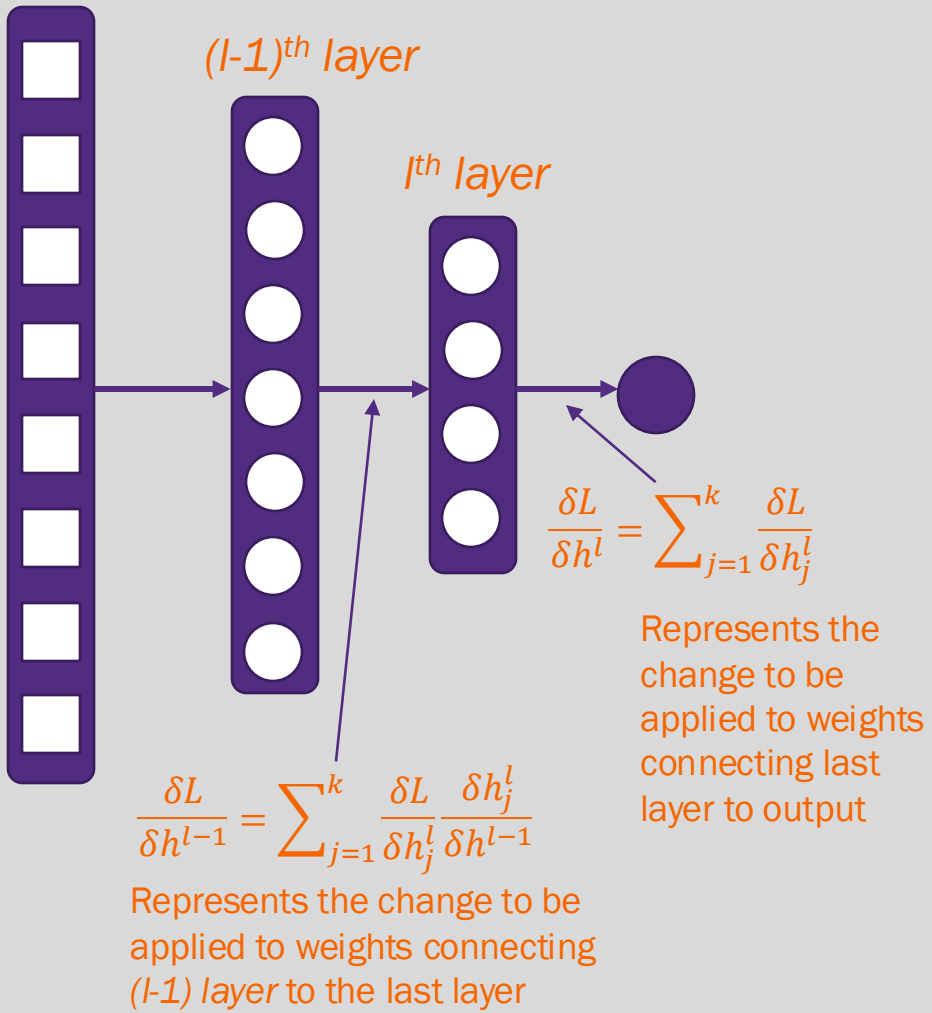
ReLU

$$\max(0, x)$$



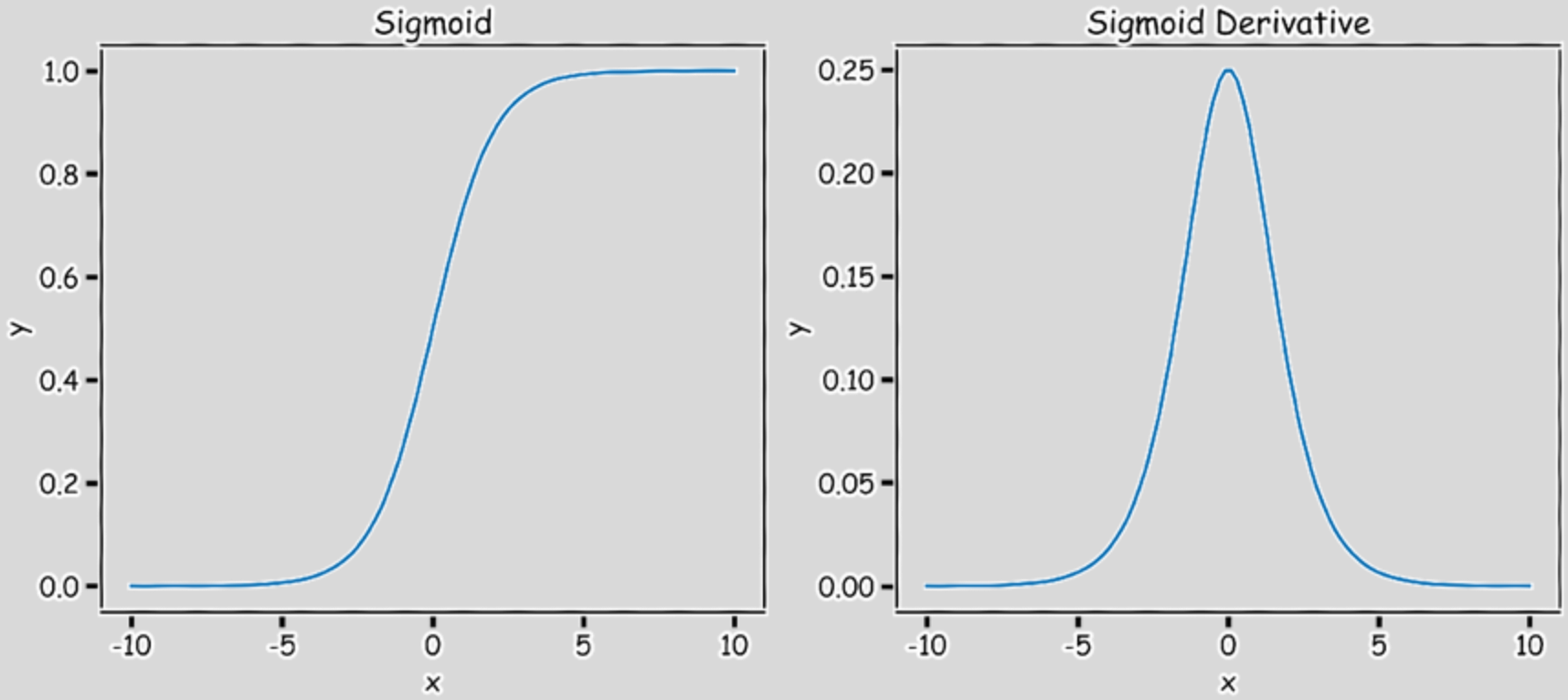
Backpropagation

- Goal: Compute the gradient (partial derivatives) of the loss function with respect to model weights in an efficient manner
- Start by calculating loss gradient WRT to final hidden layer
- Apply the Chain Rule to “propagate” error backwards to previous layer



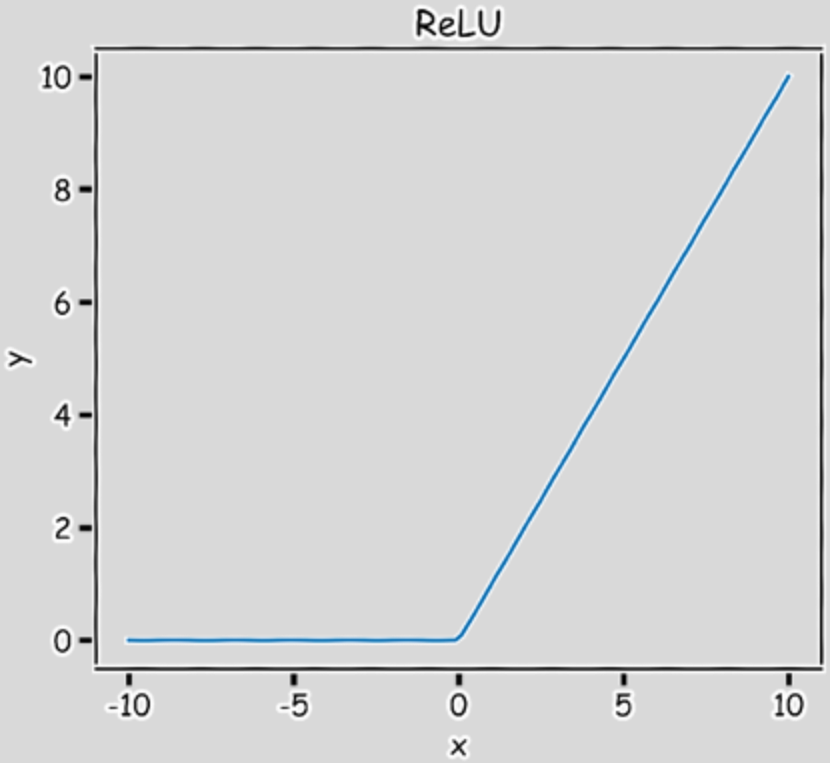
Sigmoid (aka Logistic)

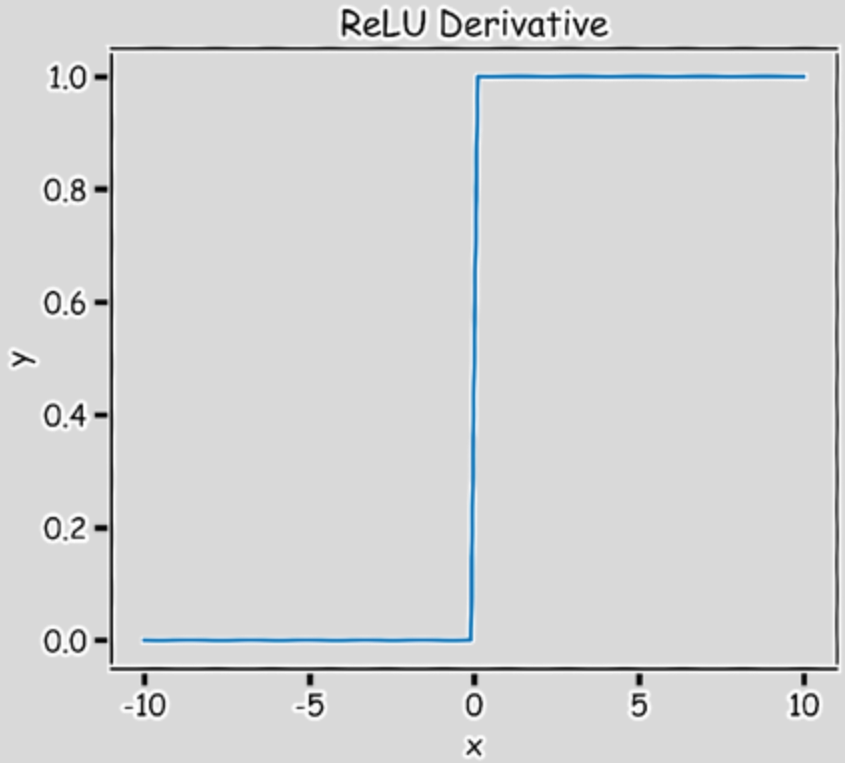
$$y = \frac{1}{1 + e^{-x}}$$



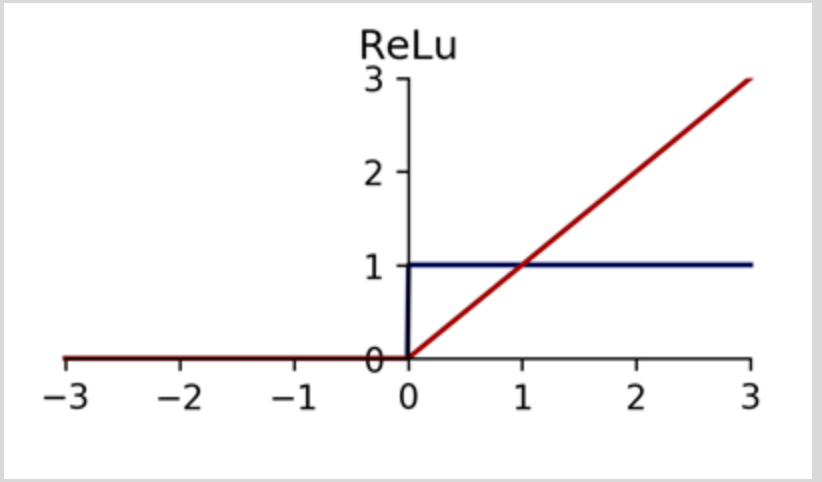
Derivative is **zero** for much of the domain. This leads to “vanishing gradients” in backpropagation -> no learning parameter update!

Rectified Linear Unit (ReLU)





$$y = \max(0, x)$$



Two major advantages:

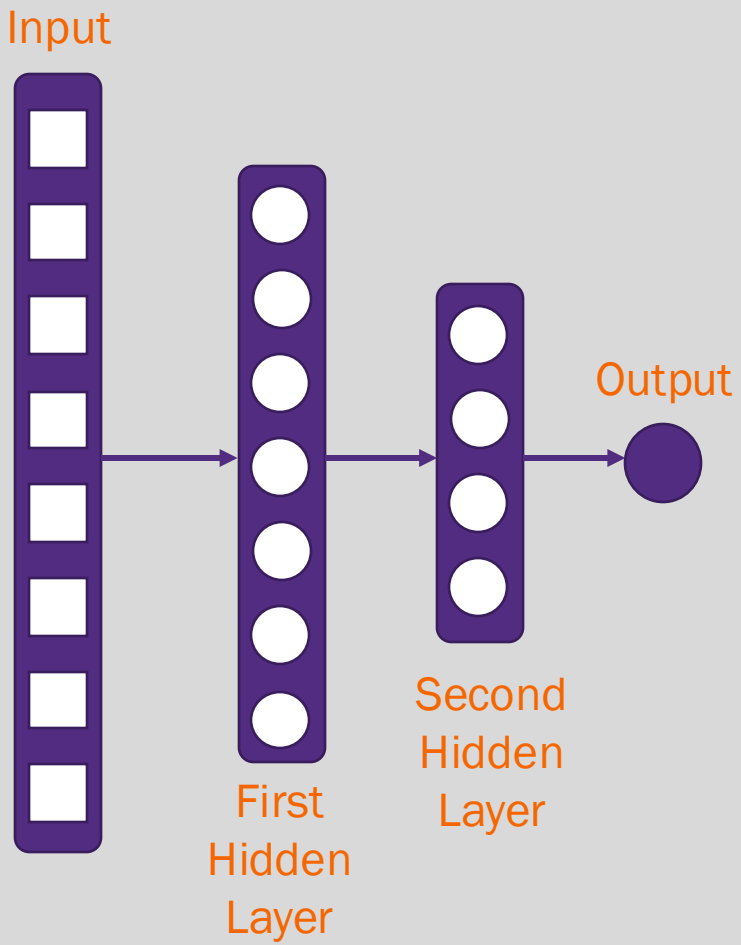
1. No vanishing gradient when x > 0
2. Provides sparsity (regularization) since y = 0 when x < 0

School of COMPUTING
Clemson University

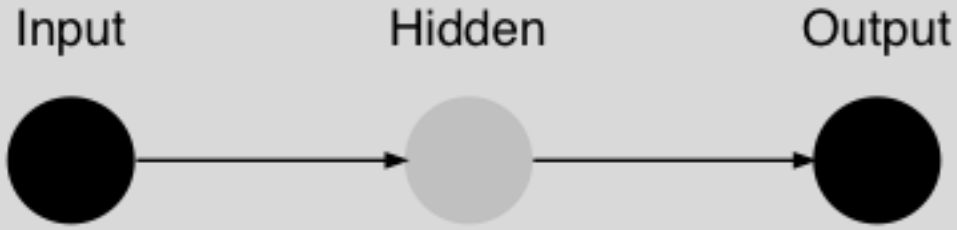
Design Choices – architecture

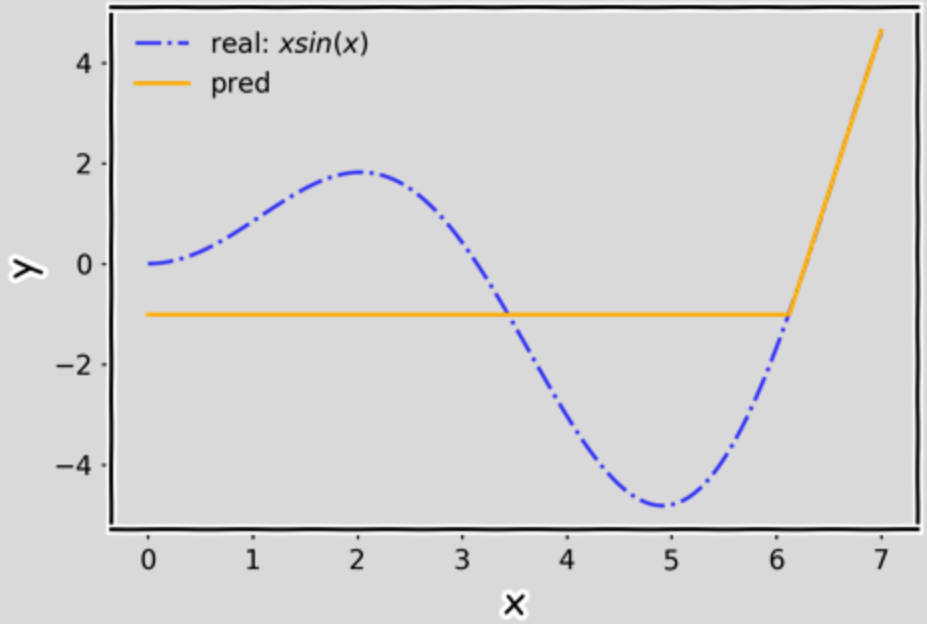
Deep learning

- Includes two or more hidden layers
- Each layer is fully connected to the next layer (feed forward networks, other architectures relax this constraint)
- Successive hidden layers use output of previous layer as input
- All computation nodes apply weighted summation and non-linear activation function to inputs
- Deep vs wide? Should we add more nodes to a single layer or add more layers?

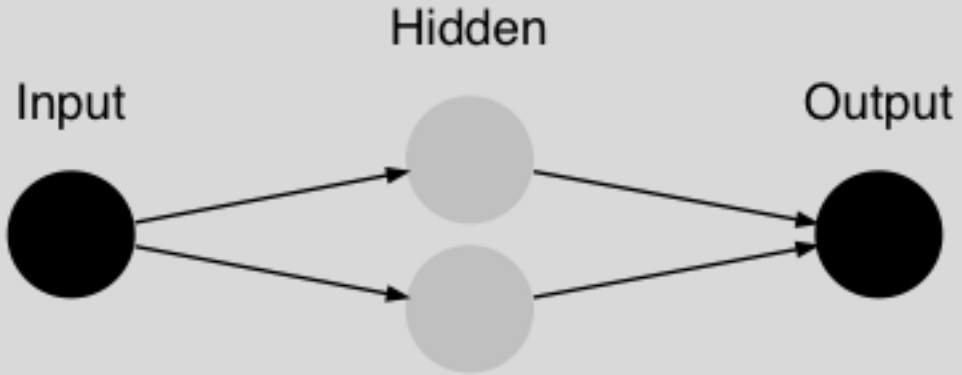


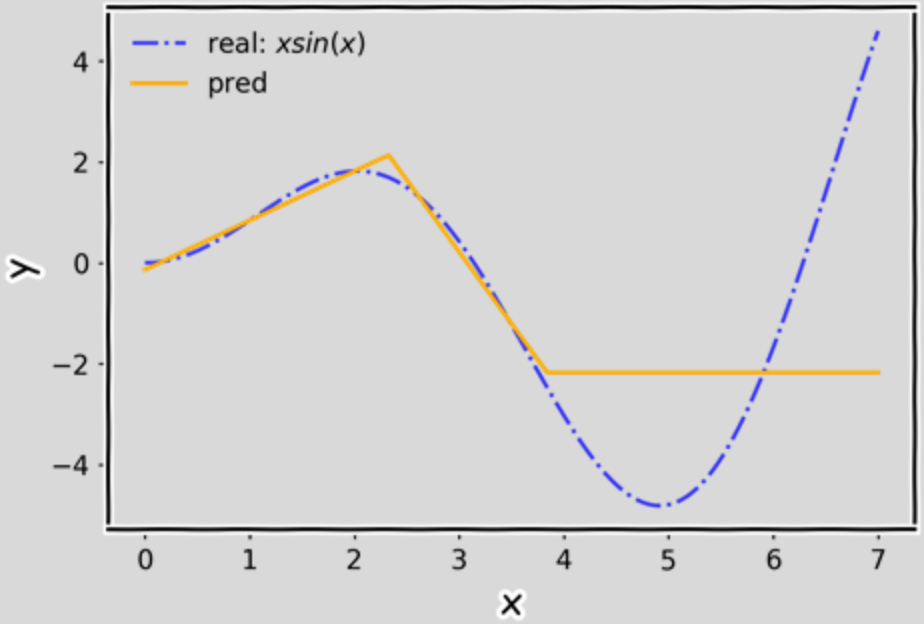
NN in action



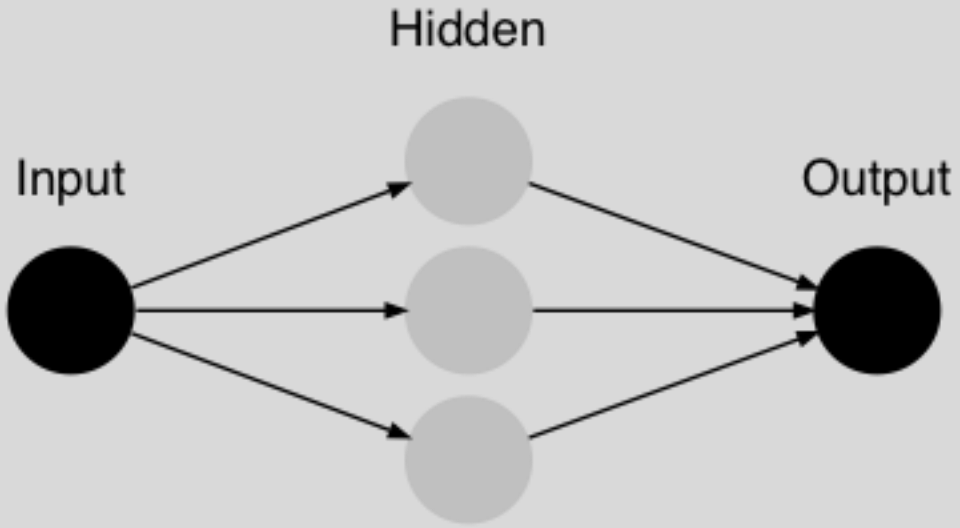


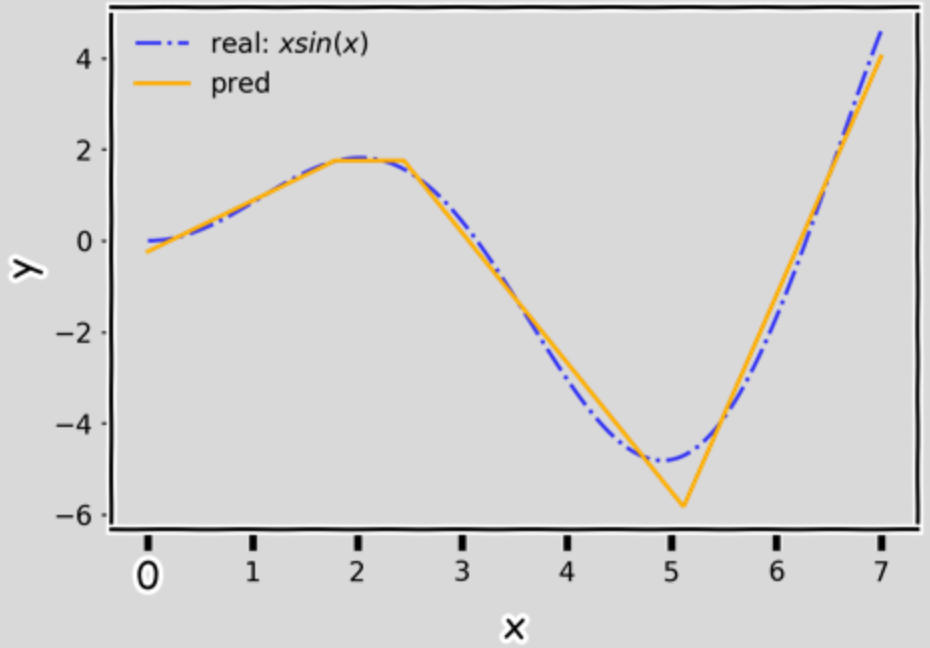
NN in action



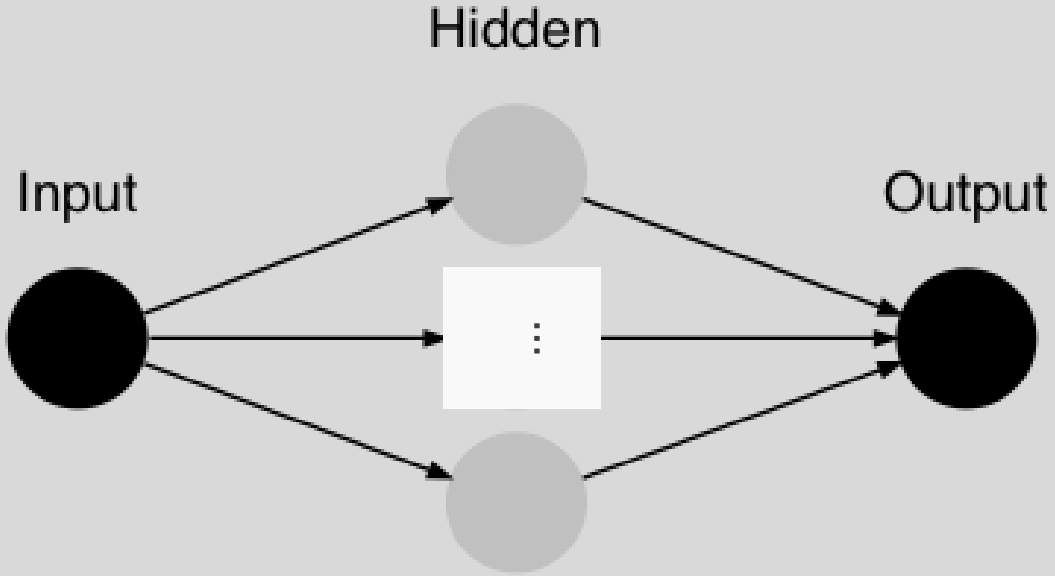


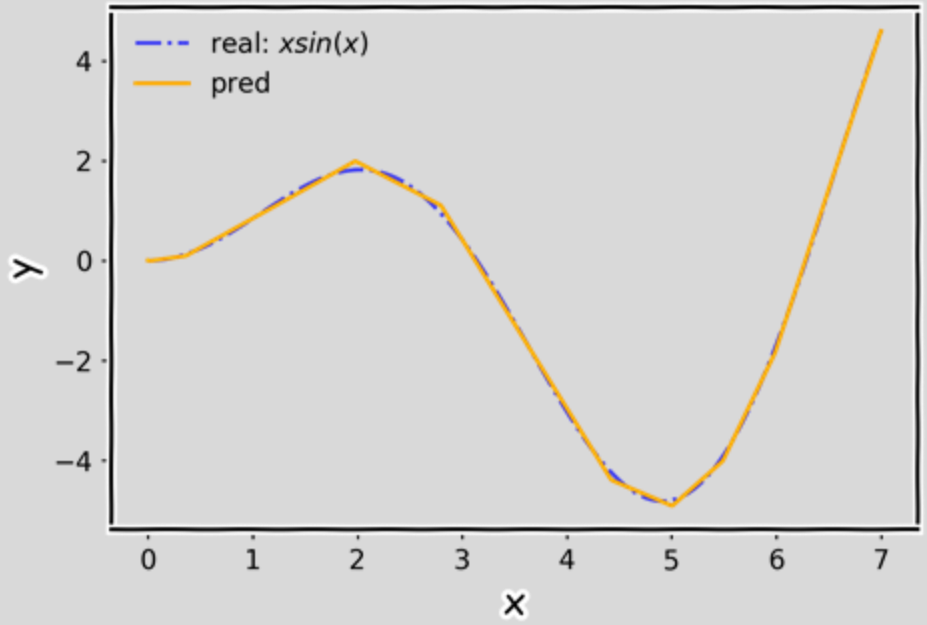
NN in action



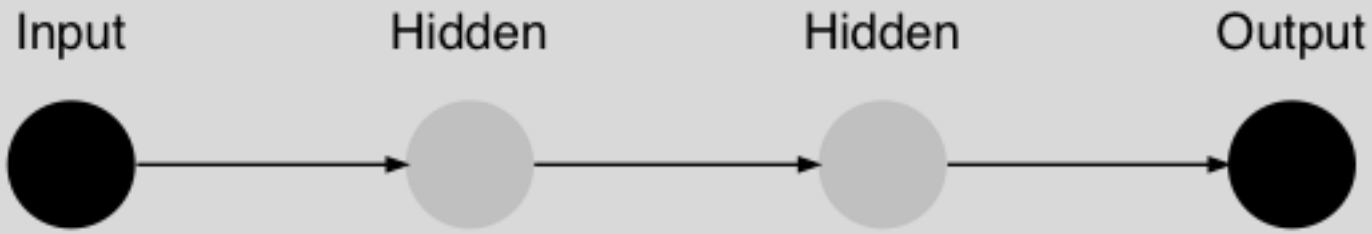


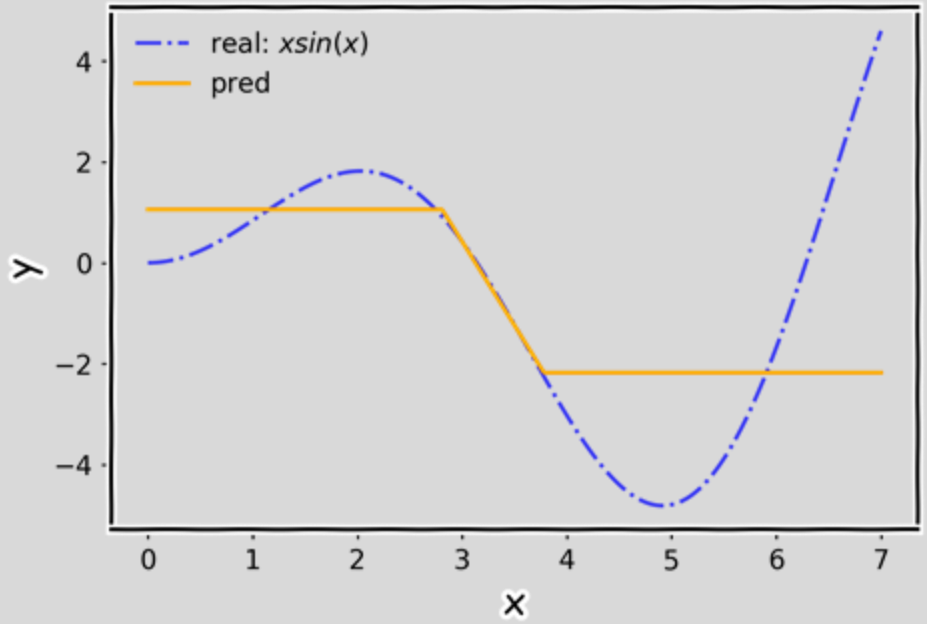
NN in action



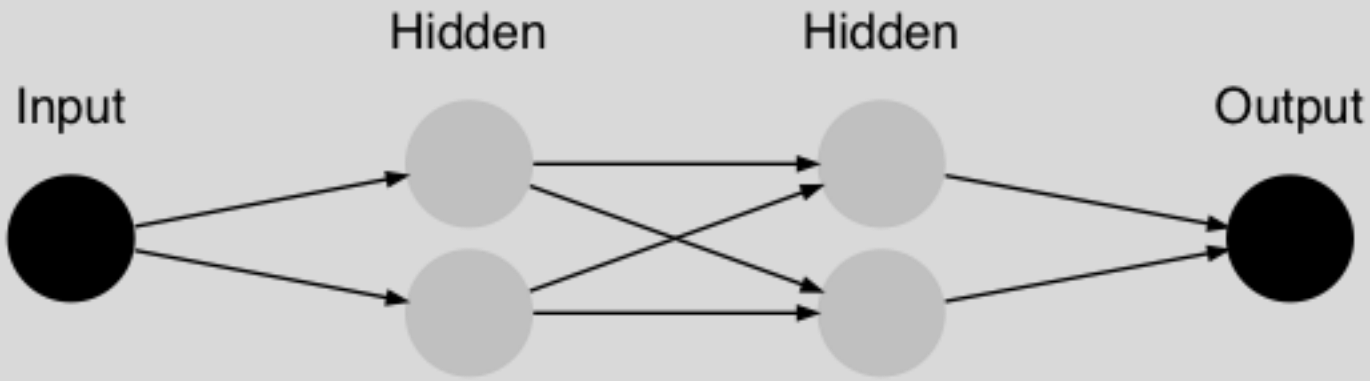


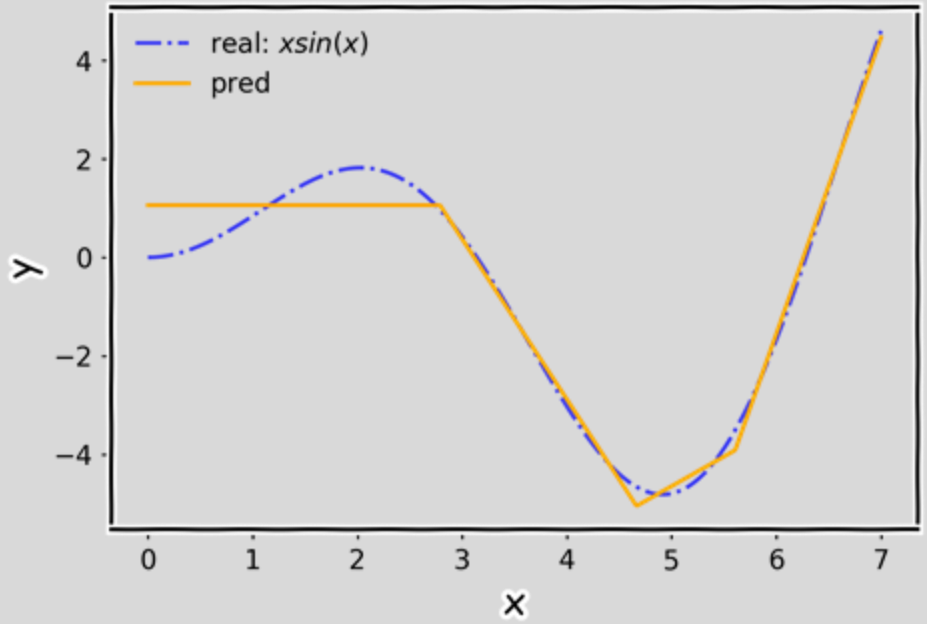
NN in action


Input
Hidden
Hidden
Output

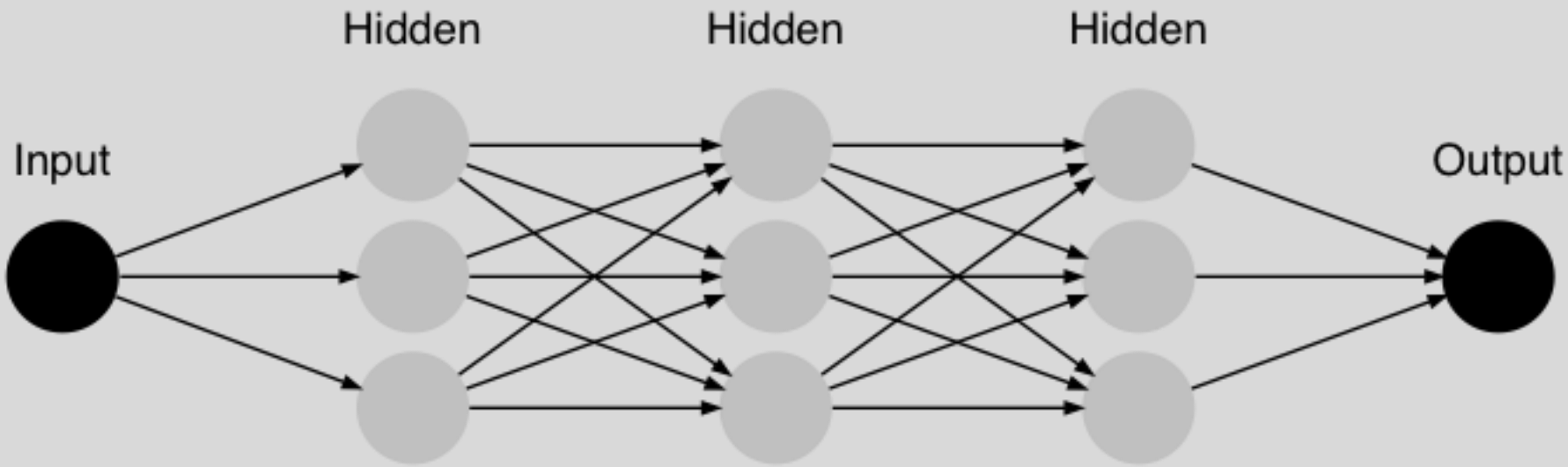

real: xsin(x)
pred
y
x
4
2
0
−2
−4
0
1
2
3
4
5
6
7

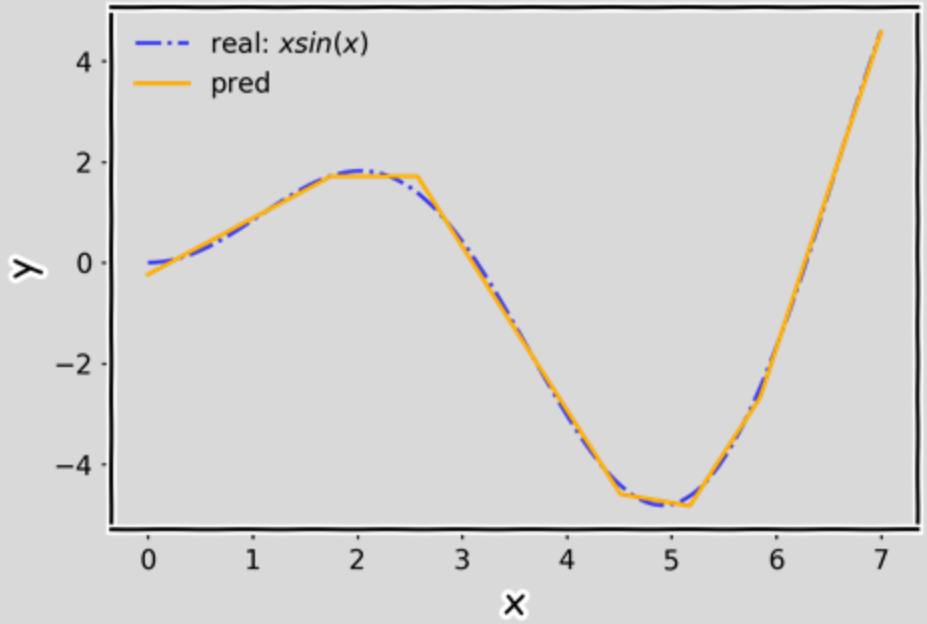
NN in action


Input
Hidden
Hidden
Output


real: xsin(x)
pred
y
x
4
2
0
−2
−4
0
1
2
3
4
5
6
7

NN in action


Hidden
Hidden
Hidden
Input
Output


real: xsin(x)
pred
4
2
0
−2
−4
y
0
1
2
3
4
5
6
7
x

Universal Approximation Theorem

Think of a Neural Network as function approximation.

$$Y = f(x) + \epsilon$$

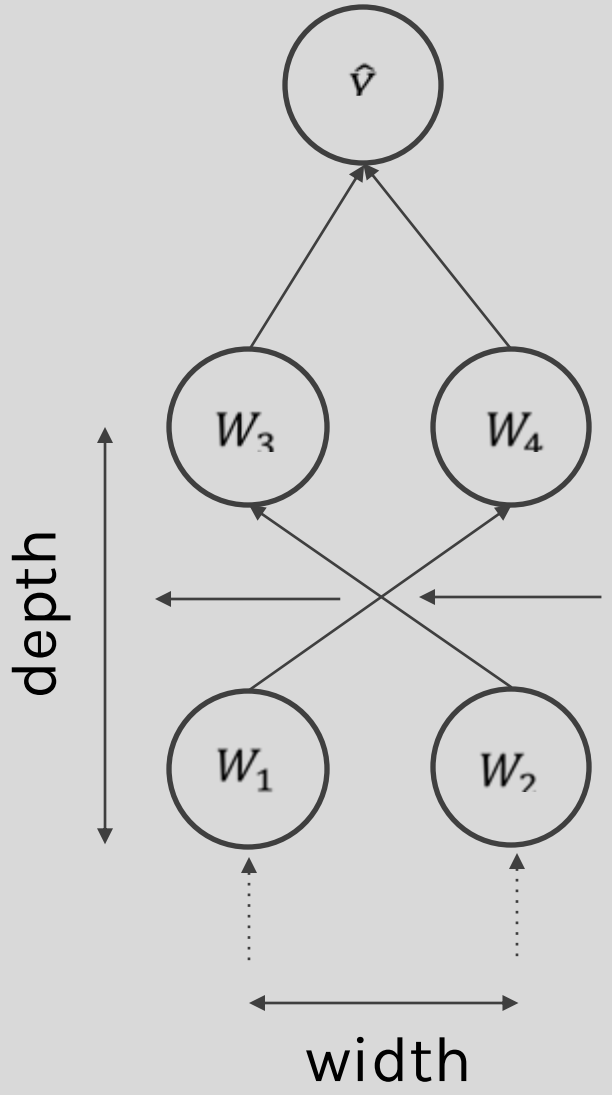
$$Y = \hat{f}(x) + \epsilon$$

$$\text{NN:} \Longrightarrow \hat{f}(x)$$

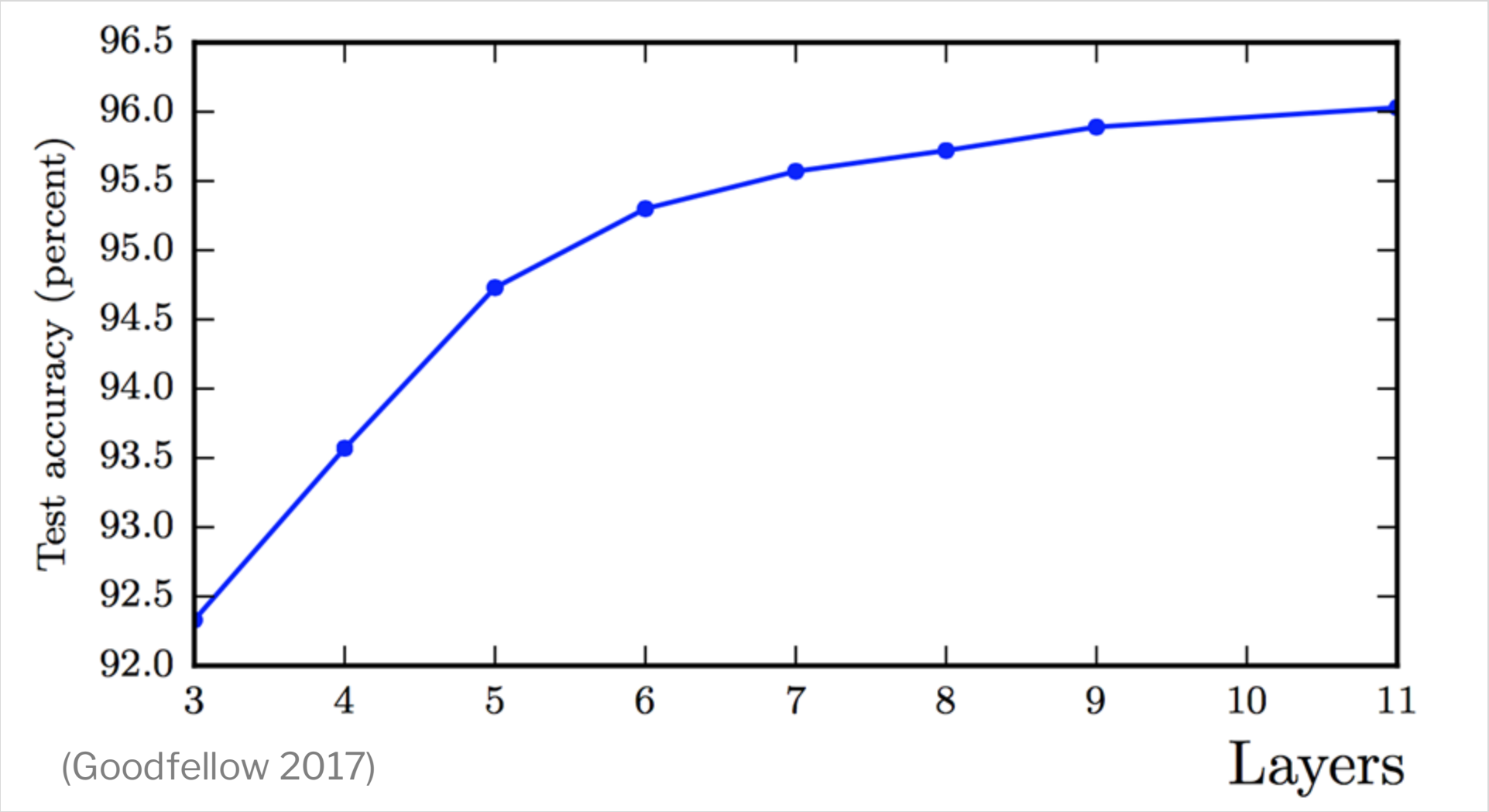
One hidden layer is enough to *represent* an approximation of any function to an arbitrary degree of accuracy

So why deeper?

- Shallow net may need (exponentially) more width



Better Generalization with Depth



(Goodfellow 2017)

Limitations

- Feed-forward (fully connected network)
  - Works well with structured data but tends to do poorly with unstructured data (e.g., images)
  - Requires geometrically large number of model parameters as number of layers and nodes increase
  - Performs poorly for certain problem classes
    - Classification of highly unstructured data (e.g., images)
    - Many natural language tasks (e.g., translation)
- Other architectures attempt to address these issues (we'll see these later)
  - Convolutional neural network (CNN)
  - Recurrent neural network (RNN)
  - Transformer networks

School of COMPUTING
Clemson University

PyTorch

PyTorch (pytorch.org)

- Ecosystem of libraries and trained models to support deep learning research and application development
- *PyTorch* library is the core library
  - Deep learning building blocks
  - Tensor operations are optimized for GPUs
  - Autograd support for automatic differentiation (backpropagation)
- Multiple sub-libraries for specialized support with datasets and models
  - torchvision – computer vision
  - torchtext – natural language

School of COMPUTING Clemson University

PyTorch Tutorials

https://pytorch.org/tutorials/