University of Colorado **Boulder**

Department of Computer Science

# CSCI 5622: Machine Learning

Chris Ketelsen

## Lecture 17:
## Neural Networks Part 1

# Roadmap

**Today**:

- The Perceptron
- Simple Feed-Forward Networks

**Next Time**:

- Back Propagation
- Implementation Details

**Next Next Time**:

- Alternate Network Architectures
- Deep Learning

## History Lesson

- **1962**: Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*
    - First neuron-based learning algorithm
    - "Could learn anything that you could program"

## History Lesson

- **1962**: Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*
    - First neuron-based learning algorithm
    - "Could learn anything that you could program"
- **1969**: Minsky & Papert, *Perceptron: An Introduction to Computational Geometry*
    - First real complexity analysis
    - Showed, in principle, many things that perceptrons can't learn to do
    - Shut down any interest in neural networks

# History Lesson

- **1986**: Rumelhart, **Hinton** & Williams, *Back Propagation*
  - Overcame many difficulties raised by Minsky, et al
  - Neural Networks wildly popular again (for a while)

# History Lesson

- **1999-2005**
  - Shift to Bayesian Methods
    - Best ideas from neural networks
    - Direct statistical computing
  - Support Vector Machines
    - Nice mathematical properties
    - Global optima vs Local optima
- A few people still playing with NNs
  - Hinton (Univ. Toronto and Google)
  - LeCun (NYU and FaceBook)
  - Bengio (Univ. Montreal and IBM)

# History Lesson

- **2005-2010**
  - Core group continues to make improvements
  - Various tricks to make NNs practical
- **2010-present**
  - Went back to methods of 1980s
  - Were wildly successful

**Question**: Why? What made a 1980s algorithm suddenly amazing 30 years later?

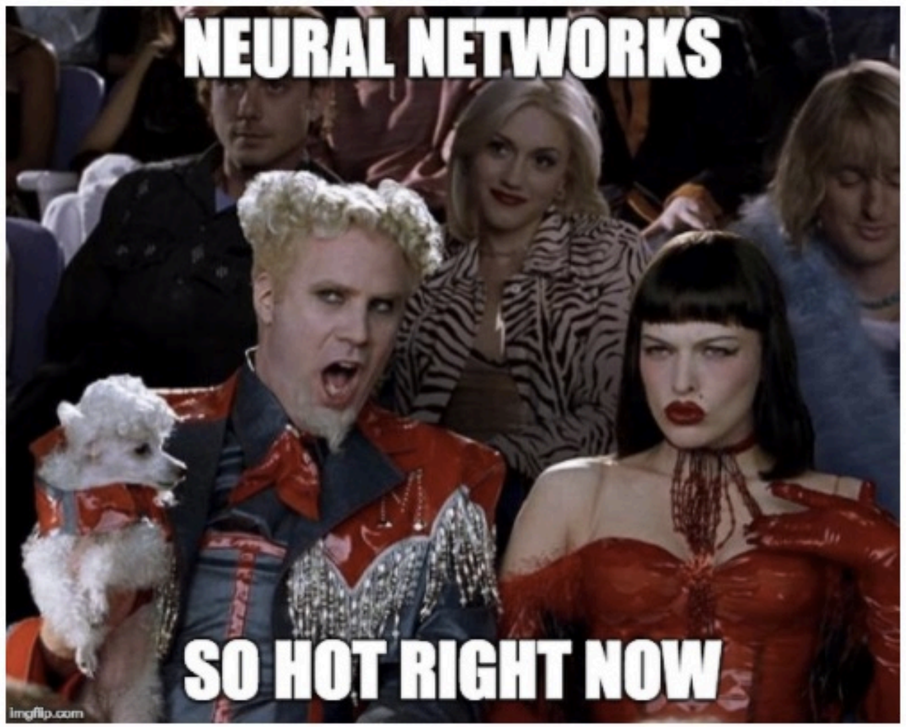
# History Lesson

**Question**: Why? What made a 1980s algorithm suddenly amazing 30 years later?

- Efficient algorithms (Back Propagation)
- Raw Computing Power
- Massive amounts of training data
    - Deep NNs in particular have tons of parameters
    - Need tons of data to effectively train

# History Lesson

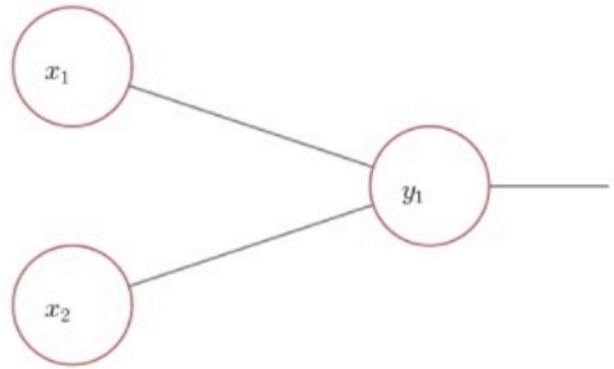
The history of Machine Learning has been very cyclic




But will they be in 20 years? Who knows?

# The Perceptron

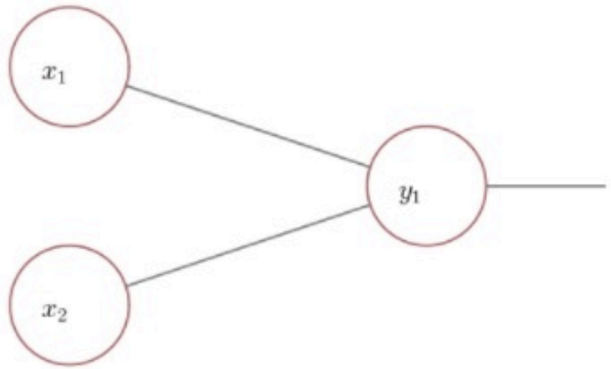
In simplest form, a perceptron takes some binary inputs, and predicts a binary output




Given some training data, learn weights associated with edges that make prediction accurate

# The Perceptron

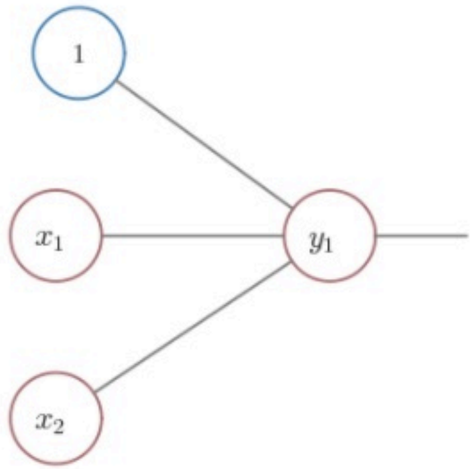
In simplest form, a perceptron takes some binary inputs, and predicts a binary output




$$\hat{y} = \begin{cases} 0 & \text{if } \mathbf{w}^T\mathbf{x} \leq \text{ threshold} \\ 1 & \text{if } \mathbf{w}^T\mathbf{x} > \text{ threshold} \end{cases}$$

# The Perceptron

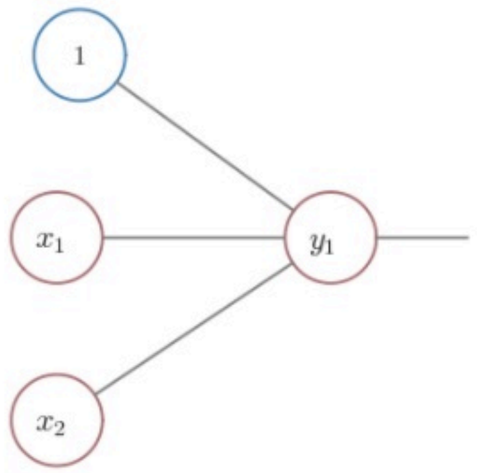
In simplest form, a perceptron takes some binary inputs, and predicts a binary output




$$\hat{y} = \begin{cases} 0 & \text{if } \mathbf{w}^T\mathbf{x} + b \leq 0 \\ 1 & \text{if } \mathbf{w}^T\mathbf{x} + b > 0 \end{cases}$$

# The Perceptron

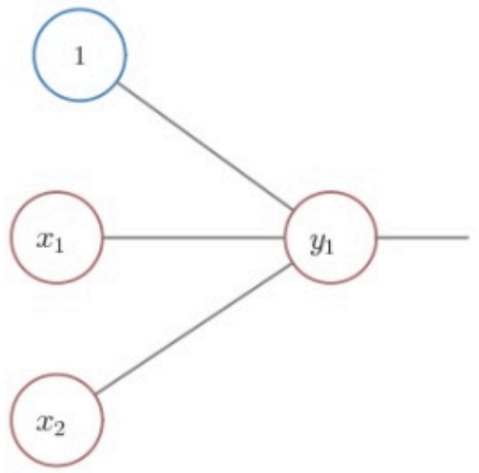
**Simple Example**: Can we learn OR?




| $x_1$ | 0 | 1 | 0 | 1 |
|---|---|---|---|---|
| $x_2$ | 0 | 0 | 1 | 1 |
| $x_1$ OR $x_2$ | 0 | 1 | 1 | 1 |

# The Perceptron

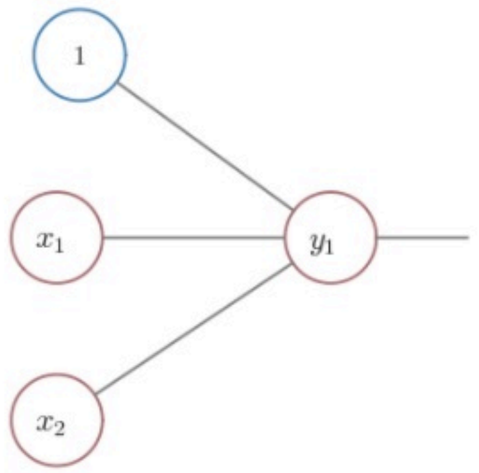
**Simple Example**: Can we learn `AND`?




| $x_1$ | 0 | 1 | 0 | 1 |
|---|---|---|---|---|
| $x_2$ | 0 | 0 | 1 | 1 |
| $x_1$ AND $x_2$ | 0 | 0 | 0 | 1 |

# The Perceptron

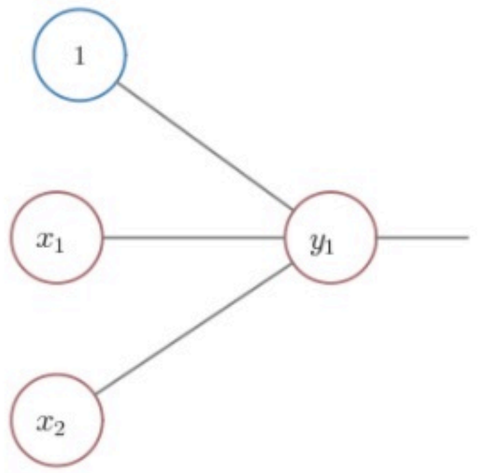
**Simple Example**: Can we learn `NAND`?




| $x_1$ | 0 | 1 | 0 | 1 |
|---|---|---|---|---|
| $x_2$ | 0 | 0 | 1 | 1 |
| $x_1$ NAND $x_2$ | 1 | 1 | 1 | 0 |

# The Perceptron

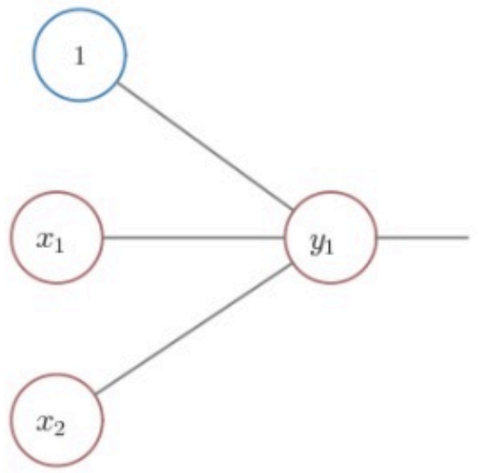
**Simple Example**: Can we learn XOR?




| | | | | |
|---|---|---|---|---|
| $x_1$ | | 0 | 1 | 0 | 1 |
| $x_2$ | | 0 | 0 | 1 | 1 |
| $x_1$ | XOR $x_2$ | 0 | 1 | 1 | 0 |

# The Perceptron

**Simple Example**: Can we learn XOR? **NOPE**!

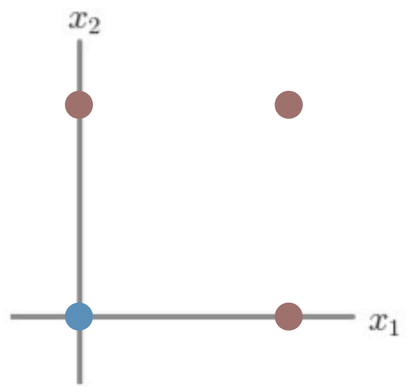



| | | | | |
|---|---|---|---|---|
| $x_1$ | | 0 | 1 | 0 | 1 |
| $x_2$ | | 0 | 0 | 1 | 1 |
| $x_1$ | XOR $x_2$ | 0 | 1 | 1 | 0 |

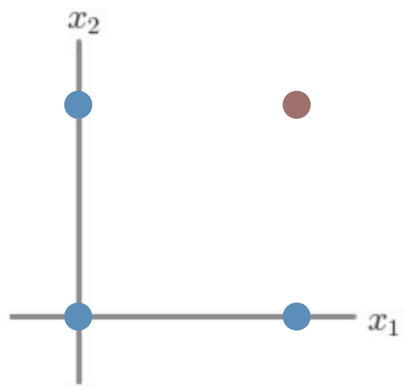
# The Perceptron

**Q**: Why can't we get XOR with this single perceptron model?

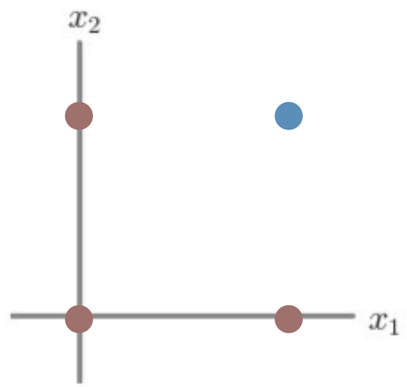
OR



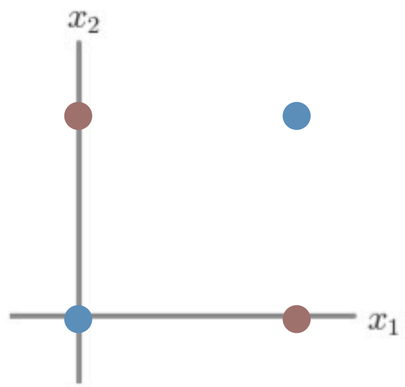

AND




NAND




XOR



# The Perceptron

**Q**: Why can't we get `XOR` with this single perceptron model?

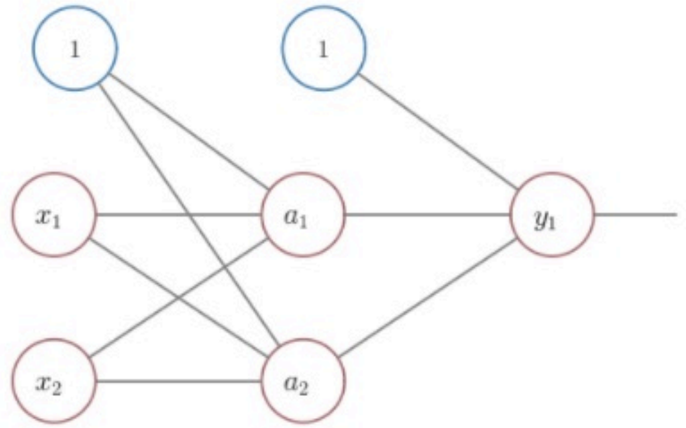
**A**: The perceptron is really just a linear classifier. Can only learn things that are linearly separable

**Q**: How can we fix this?

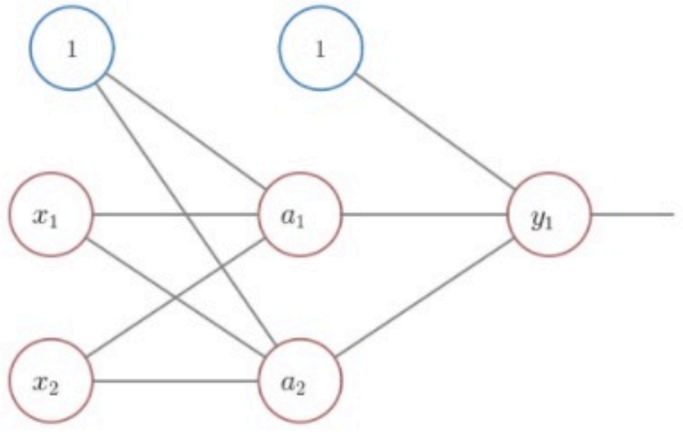
# The Perceptron

**Q**: How can we fix this?

**A**: The **Multi-Layer Perceptron**!



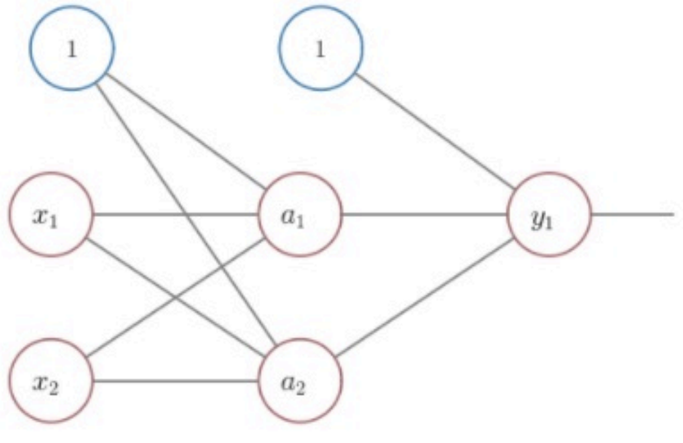
# The Perceptron




**Anatomy**:

- Input Layer
- Hidden Layer
- Output Layer

# The Perceptron




**Anatomy**: Layer 1 to Layer 2

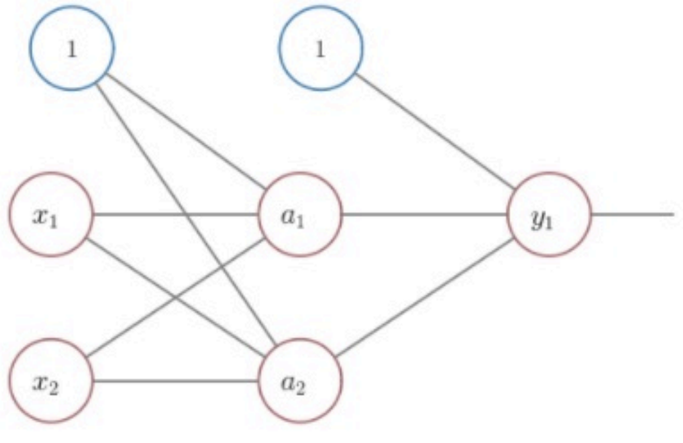
Define intermediate activities $z$, and activations $a$

$$z_1 = w^1_{11}x_1 + w^1_{12}x_2 + b^1_1, \quad a_1 = \mathrm{I}(z_1 > 0)$$

$$z_2 = w^1_{21}x_1 + w^1_{22}x_2 + b^1_2, \quad a_2 = \mathrm{I}(z_2 > 0)$$

The indicator function $I$ here is the **activation function**

# The Perceptron

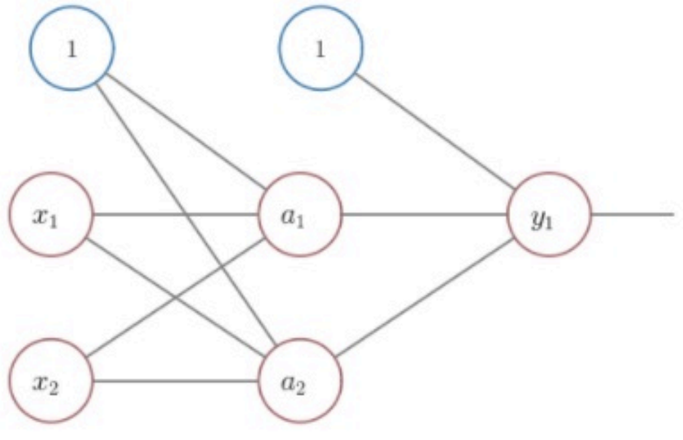



**Vectorized Anatomy**: Layer 1 to Layer 2

$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix}, \quad \mathbf{b}^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \end{bmatrix}$$

$$\mathbf{z}^2 = W^1\mathbf{x}^1 + \mathbf{b}^1, \qquad \mathbf{a}^2 = \mathrm{I}(W^1\mathbf{x}^1 + \mathbf{b}^1 > 0) = \mathrm{I}(\mathbf{z}^2 > 0)$$

# The Perceptron



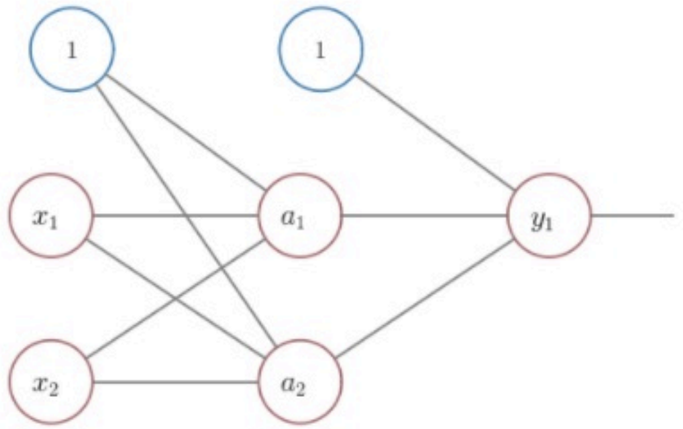

**Vectorized Anatomy**: Layer 2 to Layer 3

$$W^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 \end{bmatrix}, \quad \mathbf{b}^2 = \begin{bmatrix} b_1^2 \end{bmatrix}$$

$$\mathbf{z}^3 = W^2\mathbf{a}^2 + \mathbf{b}^2, \qquad \hat{y} = \mathrm{I}(W^2\mathbf{a}^2 + \mathbf{b}^2 > 0) = \mathrm{I}(\mathbf{z}^3 > 0)$$

# The Perceptron

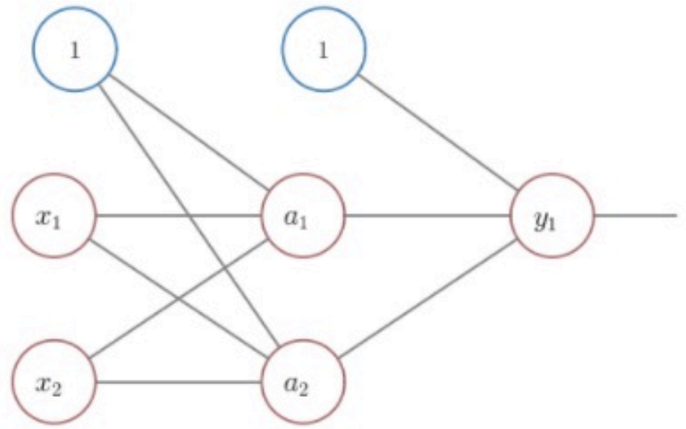
**Simple Example**: Can we learn XOR?




| | | | | |
|---|---|---|---|---|
| $x_1$ | | 0 | 1 | 0 | 1 |
| $x_2$ | | 0 | 0 | 1 | 1 |
| $x_1$ | XOR $x_2$ | 0 | 1 | 1 | 0 |

# The Perceptron

**Simple Example**: Can we learn XOR?




| | | | | |
|---|---|---|---|---|
| $x_1$ | 0 | 1 | 0 | 1 |
| $x_2$ | 0 | 0 | 1 | 1 |
| ($x_1$ OR $x_2$) AND ($x_1$ NAND $x_2$) | 0 | 1 | 1 | 0 |

# The Perceptron

## **Simple Example**: XOR Check

$$W^1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \qquad \mathbf{b}^1 = \begin{bmatrix} 0 \\ 1.5 \end{bmatrix}, \qquad W^2 = \begin{bmatrix} 1 & 1 \end{bmatrix}, \qquad \mathbf{b}^2 = \begin{bmatrix} -1 \end{bmatrix}$$

| $x_1$ | **0** | 1 | 0 | 1 |
|---|---|---|---|---|
| $x_2$ | **0** | 0 | 1 | 1 |
| $(x_1$ OR $x_2)$ AND $(x_1$ NAND $x_2)$ | **0** | 1 | 1 | 0 |

$$\mathbf{z}^2 = W^1\mathbf{x} + \mathbf{b}^1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 1.5 \end{bmatrix}, \qquad \mathbf{a}^2 = I(\mathbf{z}^2 > 0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\mathbf{z}^3 = W^2\mathbf{a}^2 + \mathbf{b}^2 = \begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix}, \qquad \hat{y} = I(\mathbf{z}^3 > 0) = \begin{bmatrix} 0 \end{bmatrix} \quad \checkmark$$

# The Perceptron

## **Simple Example**: XOR Check

$$W^1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \quad \mathbf{b}^1 = \begin{bmatrix} 0 \\ 1.5 \end{bmatrix}, \quad W^2 = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad \mathbf{b}^2 = \begin{bmatrix} -1 \end{bmatrix}$$

| $x_1$ | 0 | **1** | 0 | 1 |
|---|---|---|---|---|
| $x_2$ | 0 | **0** | 1 | 1 |
| $(x_1$ OR $x_2)$ AND $(x_1$ NAND $x_2)$ | 0 | **1** | 1 | 0 |

$$\mathbf{z}^2 = W^1\mathbf{x} + \mathbf{b}^1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}, \quad \mathbf{a}^2 = I(\mathbf{z}^2 > 0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\mathbf{z}^3 = W^2\mathbf{a}^2 + \mathbf{b}^2 = \begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix}, \quad \hat{y} = I(\mathbf{z}^3 > 0) = \begin{bmatrix} 1 \end{bmatrix} \quad \checkmark$$

# The Perceptron

## **Simple Example**: XOR Check

$$W^1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \qquad \mathbf{b}^1 = \begin{bmatrix} 0 \\ 1.5 \end{bmatrix}, \qquad W^2 = \begin{bmatrix} 1 & 1 \end{bmatrix}, \qquad \mathbf{b}^2 = \begin{bmatrix} -1 \end{bmatrix}$$

| $x_1$ | 0 | 1 | **0** | 1 |
|---|---|---|---|---|
| $x_2$ | 0 | 0 | **1** | 1 |
| $(x_1$ OR $x_2)$ AND $(x_1$ NAND $x_2)$ | 0 | 1 | **1** | 0 |

$$\mathbf{z}^2 = W^1\mathbf{x} + \mathbf{b}^1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}, \qquad \mathbf{a}^2 = I(\mathbf{z}^2 > 0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\mathbf{z}^3 = W^2\mathbf{a}^2 + \mathbf{b}^2 = \begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix}, \qquad \hat{y} = I(\mathbf{z}^3 > 0) = \begin{bmatrix} 1 \end{bmatrix} \quad \checkmark$$

# The Perceptron

## **Simple Example**: XOR Check

$$W^1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \quad \mathbf{b}^1 = \begin{bmatrix} 0 \\ 1.5 \end{bmatrix}, \quad W^2 = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad \mathbf{b}^2 = \begin{bmatrix} -1 \end{bmatrix}$$

| $x_1$ | 0 | 1 | 0 | **1** |
|---|---|---|---|---|
| $x_2$ | 0 | 0 | 1 | **1** |
| $(x_1$ OR $x_2)$ AND $(x_1$ NAND $x_2)$ | 0 | 1 | 1 | **0** |

$$\mathbf{z}^2 = W^1\mathbf{x} + \mathbf{b}^1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 2 \\ -0.5 \end{bmatrix}, \quad \mathbf{a}^2 = I(\mathbf{z}^2 > 0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\mathbf{z}^3 = W^2\mathbf{a}^2 + \mathbf{b}^2 = \begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix}, \quad \hat{y} = I(\mathbf{z}^3 > 0) = \begin{bmatrix} 0 \end{bmatrix} \quad \checkmark$$

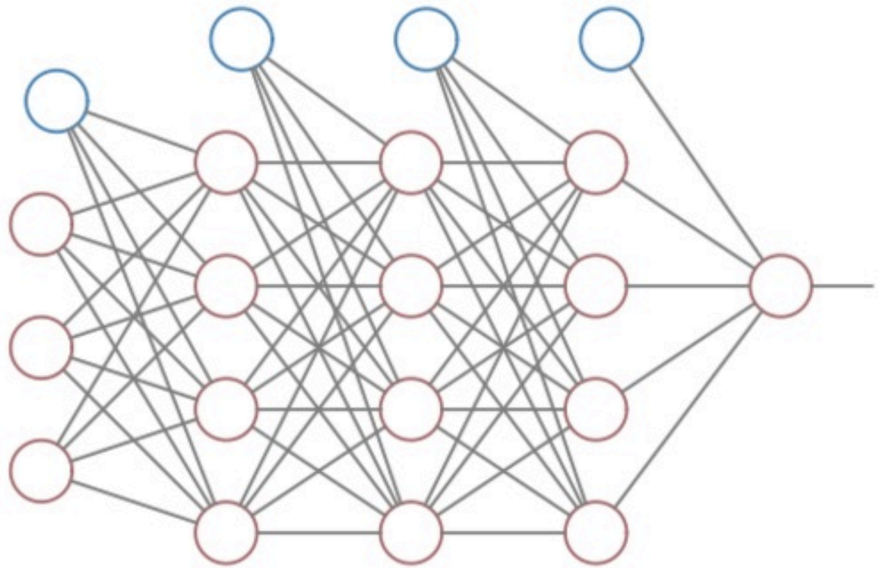
# Feed-Forward Neural Networks

OK, so Perceptron's are cool and all, but when do we get to *actual* Neural Networks?!

# Feed-Forward Neural Networks

OK, so Perceptron's are cool and all, but when do we get to *actual* Neural Networks?!

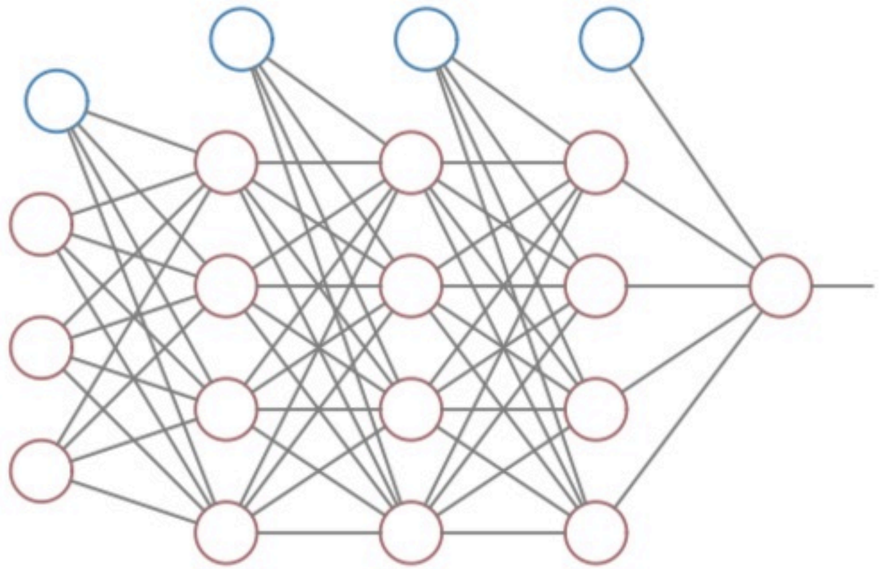
We're pretty much already there...



# Feed-Forward Neural Networks

OK, so Perceptron's are cool and all, but when do we get to *actual* Neural Networks?!

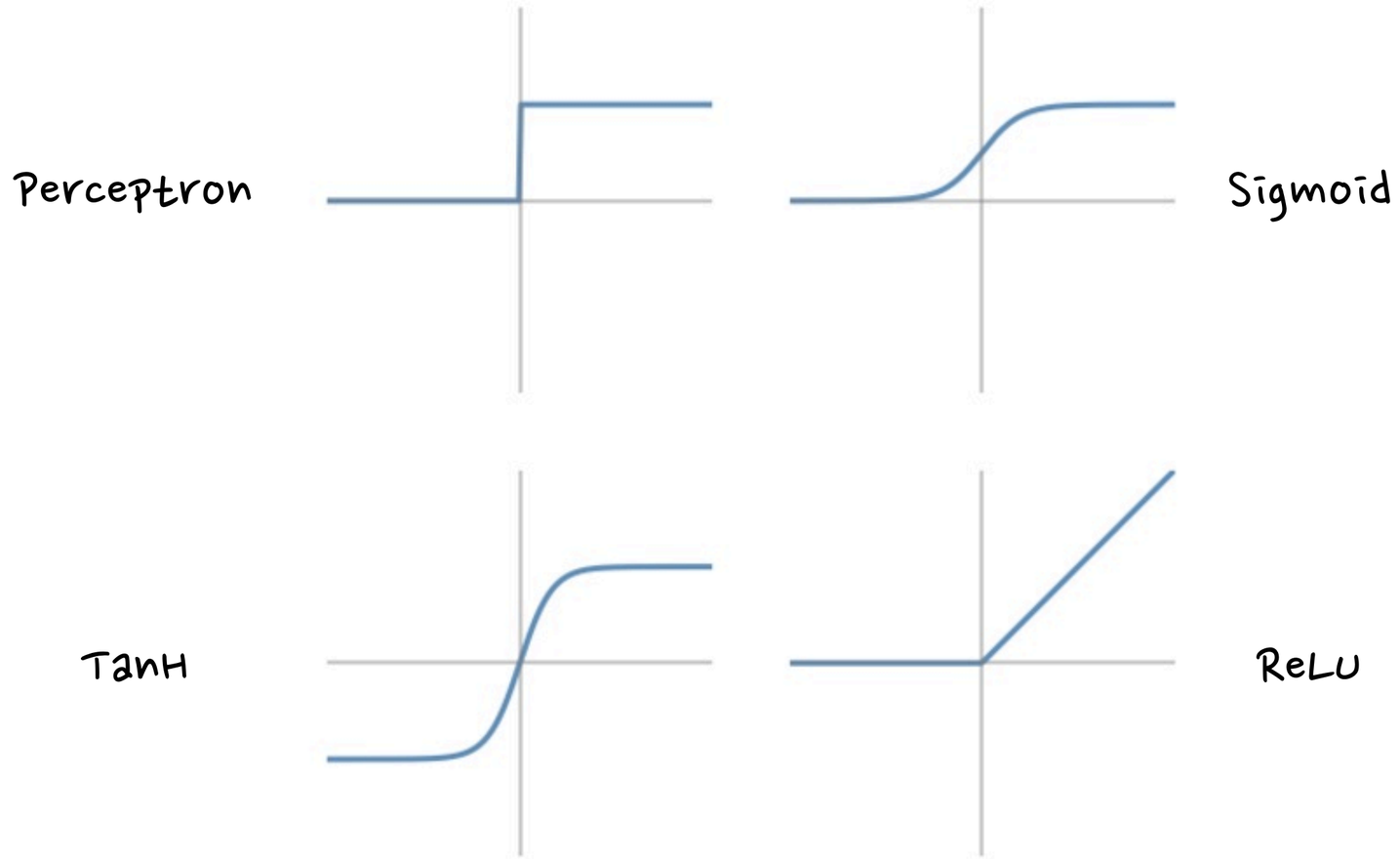
We're pretty much already there...



The main difference between an MLP and a general Feed-Forward Neural Network is the choice of activation function

# Feed-Forward Neural Networks

Here are some common activation functions:




Different activation functions lead to different behavior during the training process (tanh and ReLU currently in favor)

# Feed-Forward Neural Networks

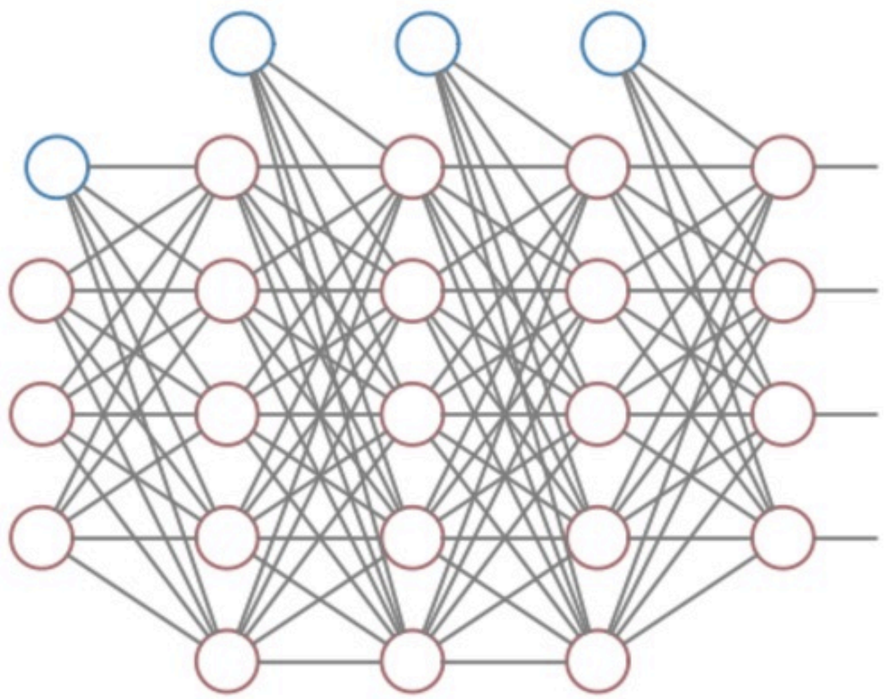
## Beyond Binary Classification

Pedestrian

Car

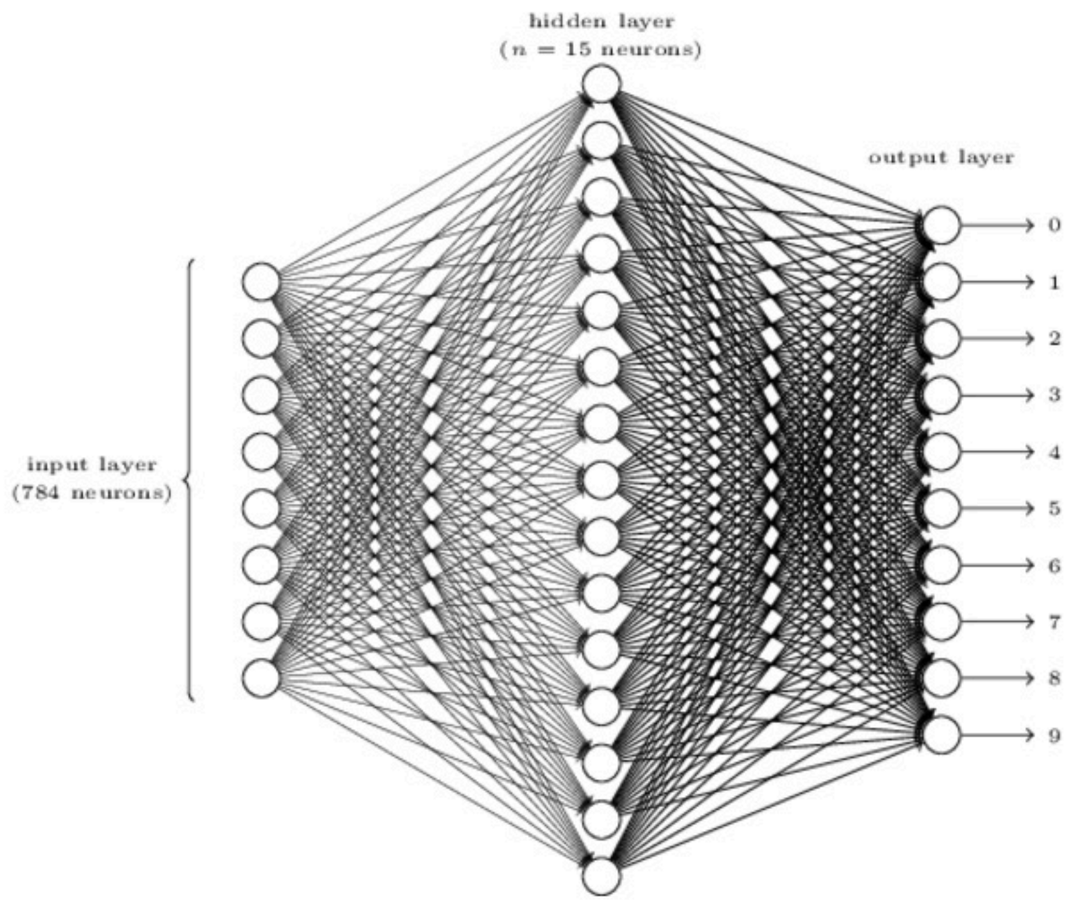
Motorcycle

Truck



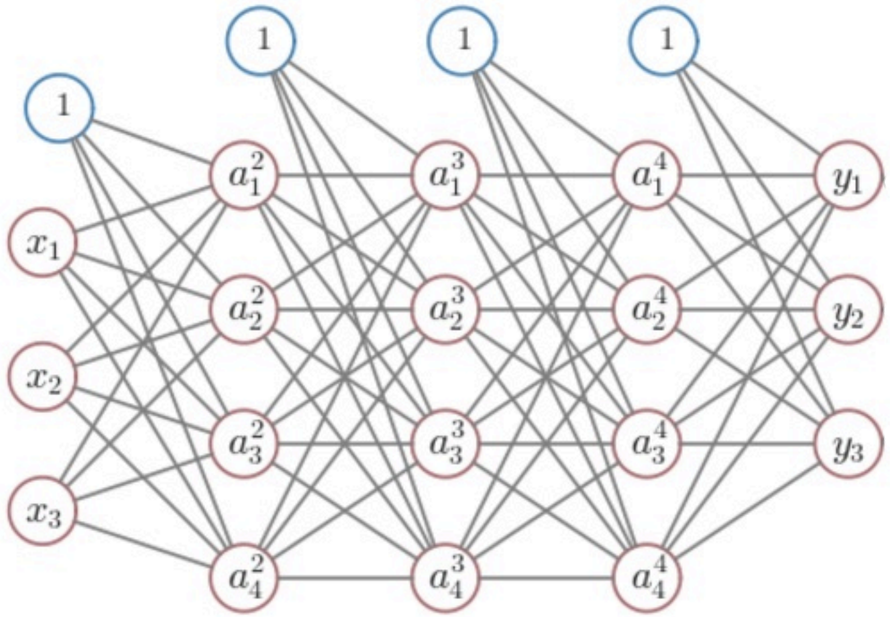
# Feed-Forward Neural Networks

## Beyond Binary Classification



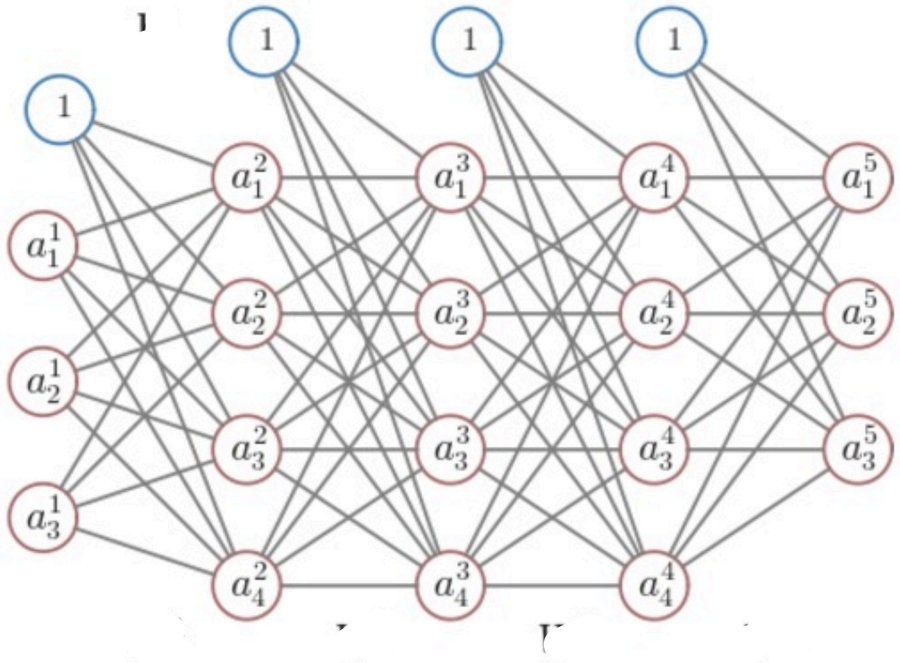
# Feed-Forward Neural Networks

OK, suppose that we're given a set of learned Weights and Biases. How do we make predictions?



# Feed-Forward Neural Networks

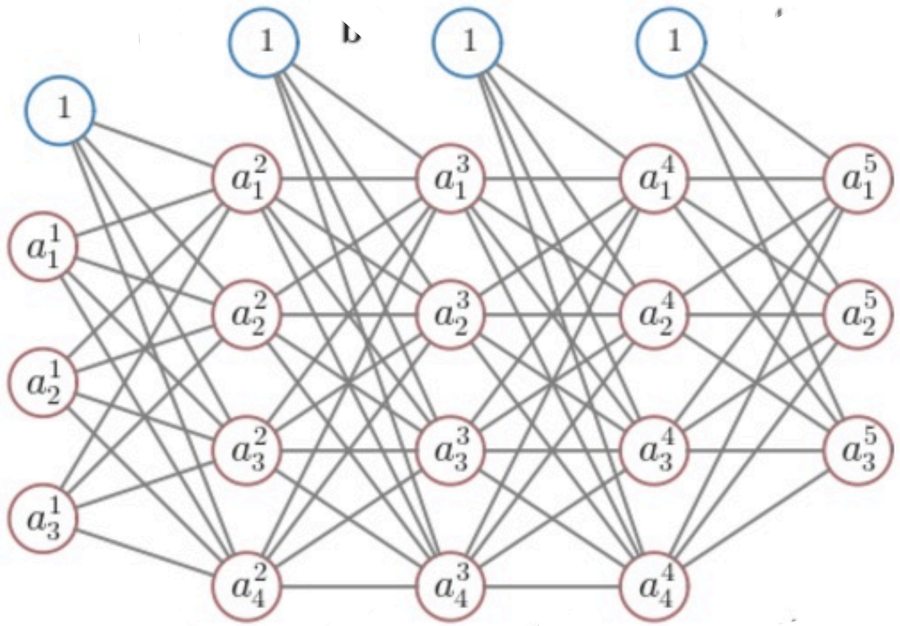
OK, suppose that we're given a set of learned Weights and Biases. How do we make predictions?




For simplicity, think of inputs and outputs as same as activations

# Feed-Forward Neural Networks

OK, suppose that we're given a set of learned Weights and Biases. How do we make predictions?

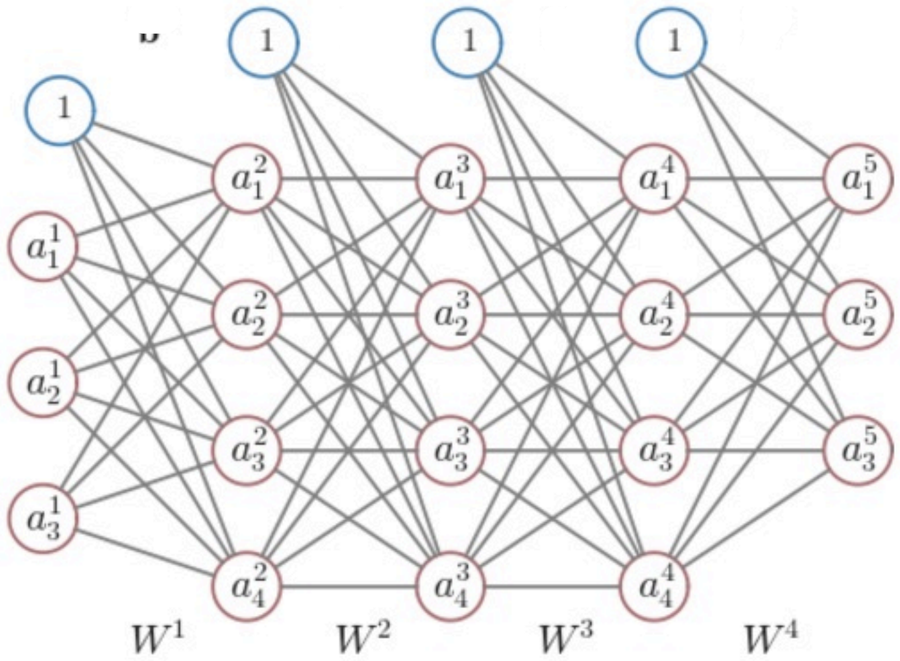



Inputs, outputs, intermediate activities, activations live on nodes

Weights and biases live on edges

# Feed-Forward Neural Networks

OK, suppose that we're given a set of learned Weights and Biases. How do we make predictions?

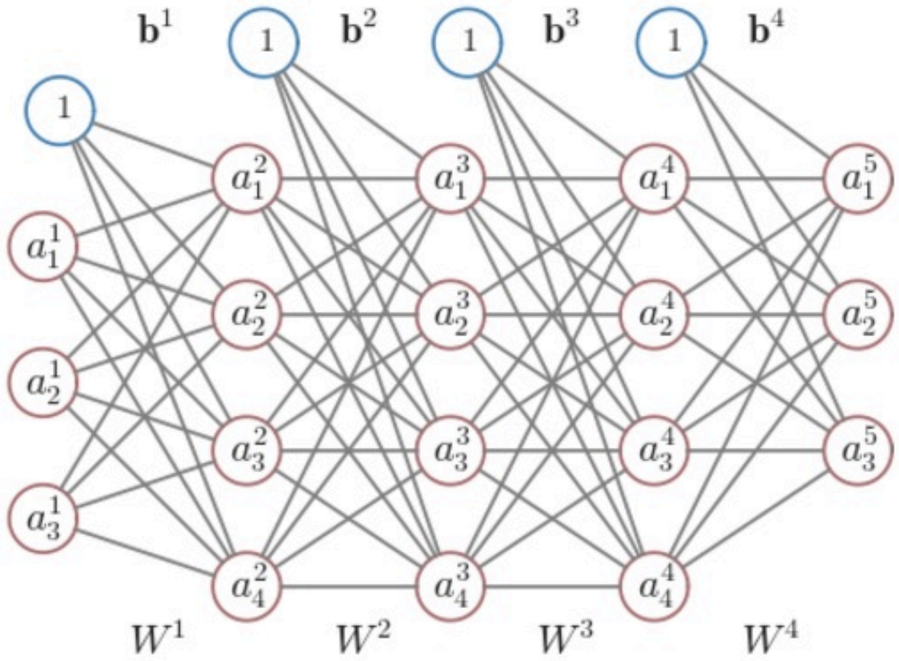



Store the weights connecting layer $\ell$ and $\ell + 1$ in $W^\ell$.

$w_{ij}^\ell$ is weight from node $j$ in layer $\ell$ to node $i$ in layer $\ell + 1$

# Feed-Forward Neural Networks

OK, suppose that we're given a set of learned Weights and Biases. How do we make predictions?

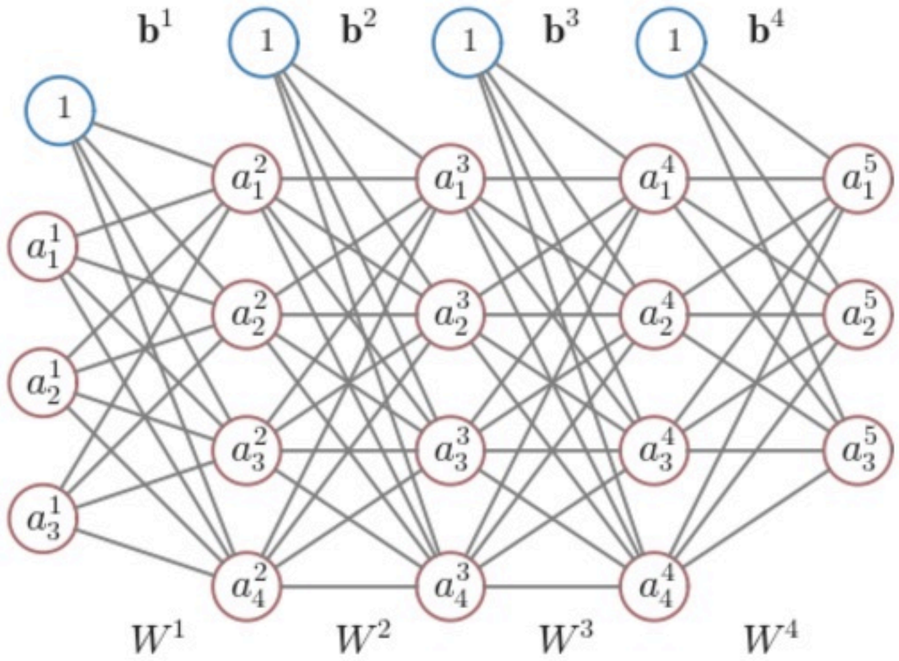



Store the biases for layer $\ell + 1$ in $\mathbf{b}^\ell$.

$b_i^\ell$ is bias on node $i$ in layer $\ell + 1$

# Feed-Forward Neural Networks

OK, suppose that we're given a set of learned Weights and Biases. How do we make predictions?




Activation of nodes in layer $\ell + 1$ given by

$$\mathbf{a}^{\ell+1} = g(\mathbf{z}^{\ell+1}) \quad \text{where} \quad \mathbf{z}^{\ell+1} = W^{\ell}\mathbf{a}^{\ell} + \mathbf{b}^{\ell}$$

# Feed-Forward Neural Networks

Suppose your network has $L$ layers

Make a prediction based on test point $\mathbf{x}$

Initialize $\mathbf{a}^1 = \mathbf{x}$

for $\ell = 1, \ldots, L-1$ :

$$\mathbf{z}^{\ell+1} = W^\ell \mathbf{a}^\ell + \mathbf{b}^\ell$$

$$\mathbf{a}^{\ell+1} = g(\mathbf{z}^{\ell+1})$$

The prediction $\hat{\mathbf{y}}$ is simply $\mathbf{a}^L$

If this is classification, predict label based on largest value in $\hat{\mathbf{y}}$

# Feed-Forward Neural Networks

OK, so how would we learn the weights for such a monstrosity?

Pretty much the usual way.

Define a loss function $C(\mathbf{x}, \mathbf{y}; \mathbf{W}, \mathbf{b})$

$$\text{Regression}: \quad \frac{1}{2}\sum_{k=1}^{m}(y_k - \hat{y}_k)^2$$

$$\text{Classification}: \quad -\sum_{k=1}^{m} y_k \log(\hat{y}_k) + (1 - y_k)\log(1 - \hat{y}_k)$$

Then what?

# Feed-Forward Neural Networks

Then we just do Stochastic Gradient Descent, as usual:

$$w_{ij}^{\ell} \leftarrow w_{ij}^{\ell} - \eta \frac{\partial C}{\partial w_{ij}^{\ell}} \qquad b_{i}^{\ell} \leftarrow b_{i}^{\ell} - \eta \frac{\partial C}{\partial b_{i}^{\ell}}$$

No problem right ...

Turns out that those partial derivatives are not that easy to compute

This is where Back Propagation comes in. It will require:

- Calculus
- Linear Algebra
- An Iron Stomach

We'll dive into Back Prop next time.

# Feed-Forward Neural Networks

Some things to think about before we get there:

- Is the problem convex?
- Do we need to worry about regularization?
- Can we do batch training?
- How do we choose good architectures?

# Acknowledgements

Many of the slides in this lecture were adopted from Mike Mozer

The material on the perceptron was adopted from Hal Daume's book: *A Course in Machine Learning*

# In Class

# In Class

# In Class

# In Class

# In Class