


# Outline

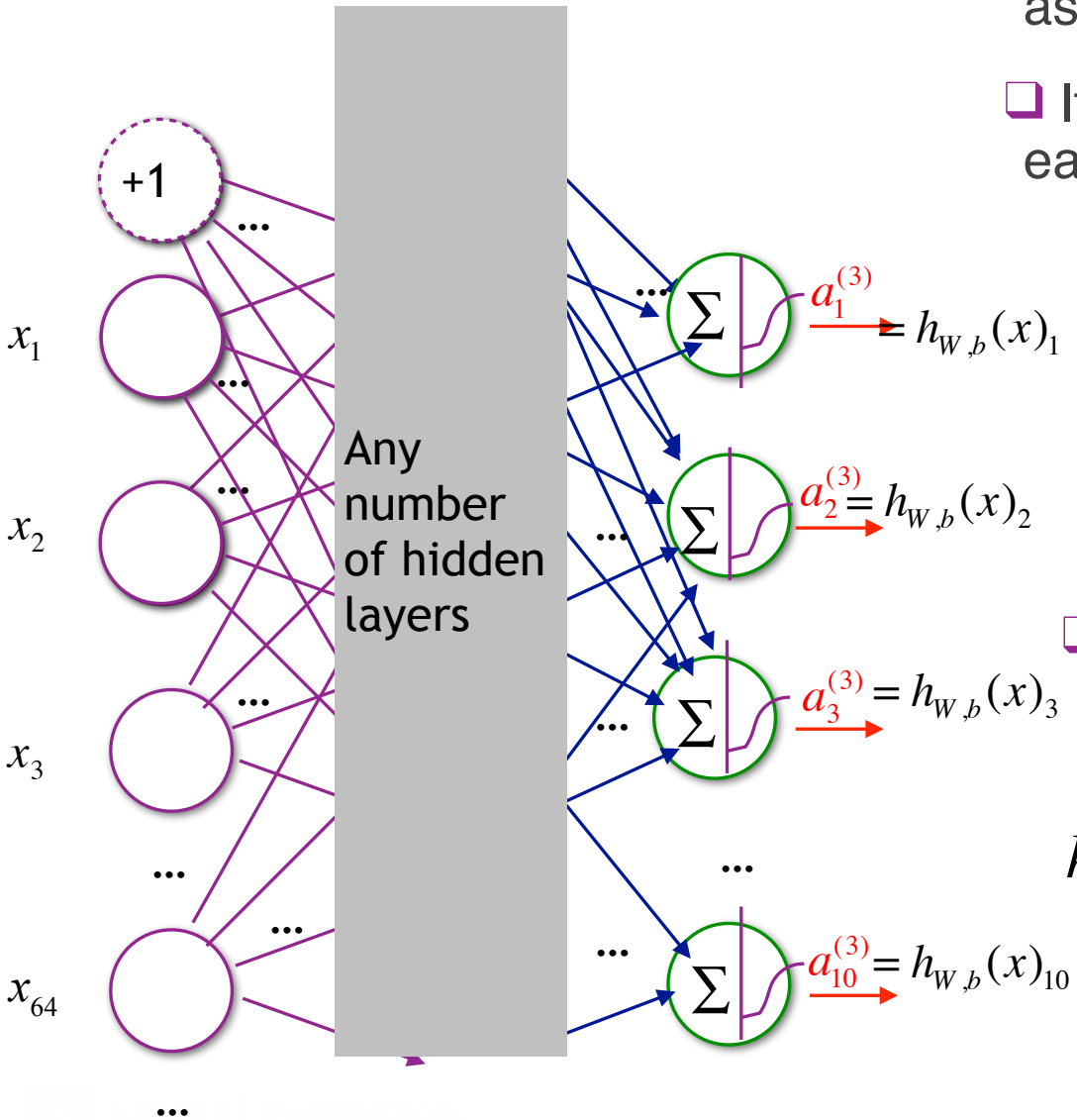
---

- ❑ Introduction to neurons
- ❑ Nonlinear classifiers from linear features
- ❑ Neural networks notation
- ❑ Pseudocode for prediction
- ❑ Training a neural network
- ❑ Implementing gradient descent for neural networks
  - Vectorization
- ❑ Preprocessing
- ❑ Initialization
- ❑ Activations

# Multiple Output Units: One-vs-All

□ Instead of just computing one output, we can compute as many as we would like

□ If we were classifying digits, we could have ten outputs, one for each digit - using a 1 of k representation (aka one-hot encoding)



$$y = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \text{If } y = 0 \quad y = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad \text{If } y = 1 \quad y = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \quad \text{If } y = 9$$

□ For our hypothesis  $h_{W,b}(x) \in \mathbb{R}^K$  we want:

$$h_{W,b}(x) = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad h_{W,b}(x) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad \dots \quad h_{W,b}(x) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

# Pre-processing steps

## One-vs-all

```
def convert_y_to_vect(y):
    y_vect = np.zeros((len(y), 10))
    for i in range(len(y)):
        y_vect[i, y[i]] = 1
    return y_vect
```

```
[9
 9
 4
 3
 ...]
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 ...]]
```

For each feature.  
Find the mean and standard deviation of the feature and then subtract the mean and divide by the standard deviation.  
 $x' = (x - \text{mean}) / \text{standard\_deviation}$

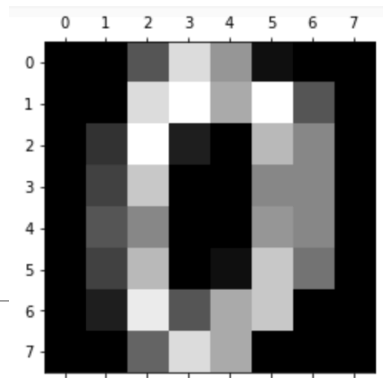
Mean of feature 1 is 0  
Mean of feature 2 is 3

## Scale the data set

```
X_scale = StandardScaler()
X = X_scale.fit_transform(digits.data)
```

```
0.  0.  5. 13.  9.  1.  0.  0.
0.  0. 13. 15. 10. 15.  5.  0.
0.  3. 15.  2.  0. 11.  8.  0.
0.  4. 12.  0.  0.  8.  8.  0.
0.  5.  8.  0.  0.  9.  8.  0.
0.  4. 11.  0.  1. 12.  7.  0.
0.  2. 14.  5. 10. 12.  0.  0.
0.  0.  6. 13. 10.  0.  0.  0.
```

```
0.      -0.335 -0.043  0.274 -0.664 -0.844 -0.41  -0.125
-0.059 -0.624  0.483  0.76  -0.058  1.128  0.88  -0.13
-0.045  0.111  0.896 -0.861 -1.15   0.515  1.906 -0.114
-0.033  0.486  0.47  -1.5   -1.614  0.076  1.542 -0.047
0.      0.765  0.053 -1.448 -1.737  0.044  1.44   0.
-0.061  0.811  0.63  -1.122 -1.066  0.661  0.818 -0.089
-0.035  0.742  1.151 -0.869  0.11   0.538 -0.757 -0.21
-0.024 -0.299  0.087  0.208 -0.367 -1.147 -0.506 -0.196
```



# Preprocessing

To speed up learning

- **Mean subtraction:** For every feature subtract the mean, thus zero centering the data

$$X = \begin{bmatrix} 2 & 2 & 3 \\ -1 & 4 & 6 \\ 2 & 0 & 9 \end{bmatrix}$$

- $X -= \text{np.mean}(X, \text{axis} = 0)$

$$\begin{bmatrix} 1. & 0. & -3. \\ -2. & 2. & 0. \\ 1. & -2. & 3. \end{bmatrix}$$

- For image data, we can subtract the average pixel intensity

- $X -= \text{np.mean}(X)$

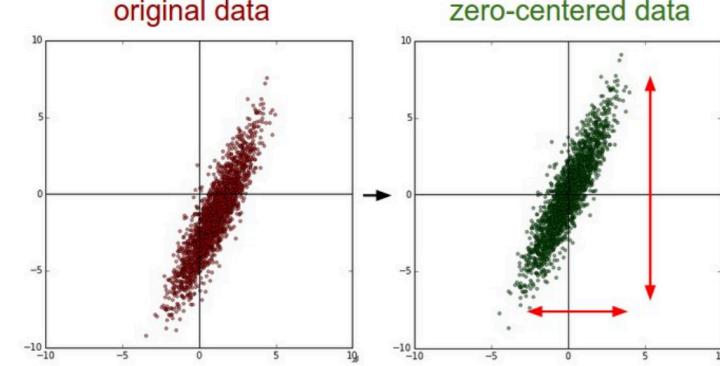
$$\begin{bmatrix} -1. & -1. & 0. \\ -4. & 1. & 3. \\ -1. & -3. & 6. \end{bmatrix}$$

- **Normalization:** make all the features are roughly the same scale

- $X /= \text{np.std}(X, \text{axis}=0)$

$$\begin{bmatrix} 0.7 & 0. & -1.2 \\ -1.4 & 1.2 & 0. \\ 0.7 & -1.2 & 1.2 \end{bmatrix}$$

- min/max scaling Or make the min and max for each feature -1 and 1 (only do this if each feature should be equally important)



<https://cs231n.github.io/neural-networks-2/>


$$X = \begin{bmatrix} 2 & 2 & 3 \\ -1 & 4 & 6 \\ 2 & 0 & 9 \end{bmatrix}$$

$$\begin{bmatrix} 1. & 0. & -3. \\ -2. & 2. & 0. \\ 1. & -2. & 3. \end{bmatrix}$$

Often only zero center for image data (and don't normalize)

# Outline

---

- ❑ Introduction to neurons
- ❑ Nonlinear classifiers from linear features
- ❑ Neural networks notation
- ❑ Pseudocode for prediction
- ❑ Training a neural network
- ❑ Implementing gradient descent for neural networks
  - Vectorization
- ❑ Preprocessing
-  ❑ Initialization
- ❑ Activations

# Initialization

in back propagation we propagate the error gradient from the output layer to the input layer.

Vanishing gradient problem: the gradients can get smaller as they propagate towards the front of the network - and thus they don't get change

Exploding gradients problem: the gradients can get larger - too large (mostly a problem in recurrent neural networks)

Another problem is layers learn at different speeds


Scale and center the data first

Activation function	Uniform distribution $[-r, r]$	Normal distribution
Logistic	$r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
Hyperbolic tangent	$r = 4\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = 4\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
ReLU (and its variants)	$r = \sqrt{2}\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

<https://www.deeplearning.ai/ai-notes/initialization/>

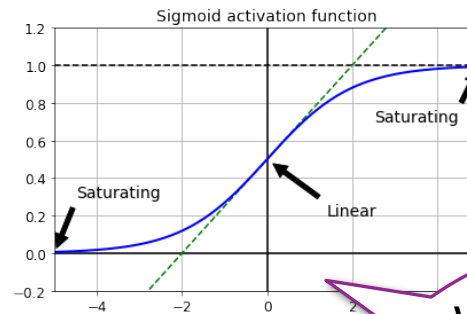
# Outline

---

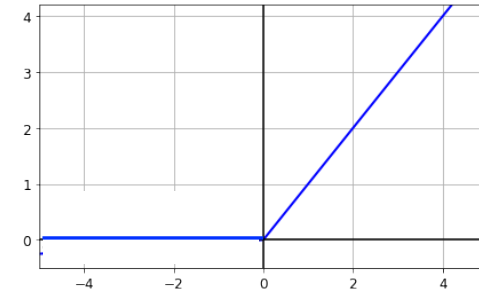
- ❑ Introduction to neurons
- ❑ Nonlinear classifiers from linear features
- ❑ Neural networks notation
- ❑ Pseudocode for prediction
- ❑ Training a neural network
- ❑ Implementing gradient descent for neural networks
  - Vectorization
- ❑ Preprocessing
- ❑ Initialization
-  ❑ Activations

# Activations

deep neural nets can have saturation problems with sigmoid and tan



$$\max(0, z)$$



ReLU - doesn't saturate in +region, suffers from dying ReLU problem

Leaky ReLU - typically hyperparameter  $\alpha = 0.2$ . This prevents leaky ReLU from dying. (Some results show  $\alpha = 0.2$  is better than  $\alpha = 0.01$ )

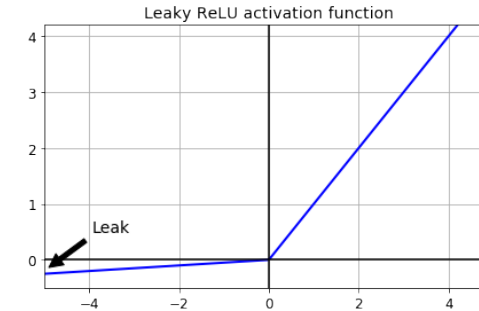
ELU (exponential linear unit) - typically hyperparameter  $\alpha = 1$ . It takes on negative values to prevent vanishing gradients and non-zero gradients, so don't have a dying unit problem; smooth everywhere. However, it is slower than ReLU and leaky ReLU due to the exponential function but often has faster convergence

SELU (scaled exponential unit)

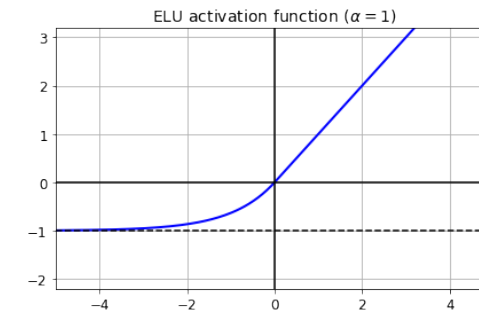
What happens if  $z=10$ , or  $-10$ ? What is the

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial \sigma} \frac{\partial \sigma}{\partial W_{ij}}$$

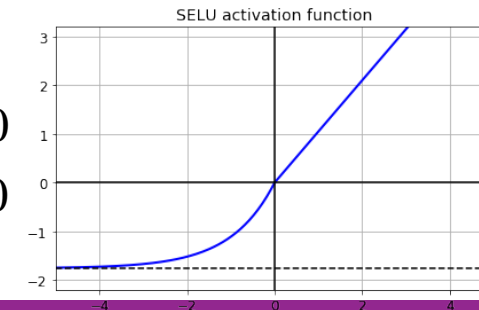
$$\sigma(z, z)$$



$$\text{ELU}_{\alpha}(z) = \begin{cases} \alpha(\exp(z) - 1) & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$



$$\text{SELU}_{\alpha}(z) = \lambda \begin{cases} \alpha(\exp(z) - 1) & \text{if } z > 0 \\ z & \text{if } z \leq 0 \end{cases}$$





# Hyperparameter testing

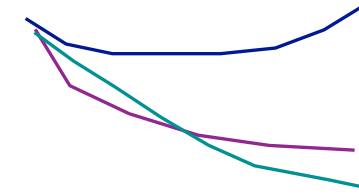
In batch  
gradient descent  
cost function  
should go down

Babysit models (if you don't have the ability to try too many parameters)

Sample at random the many different hyper-parameters:

uniformly at random examples:

- # hidden units e.g. 20-100
- # layers 2-6

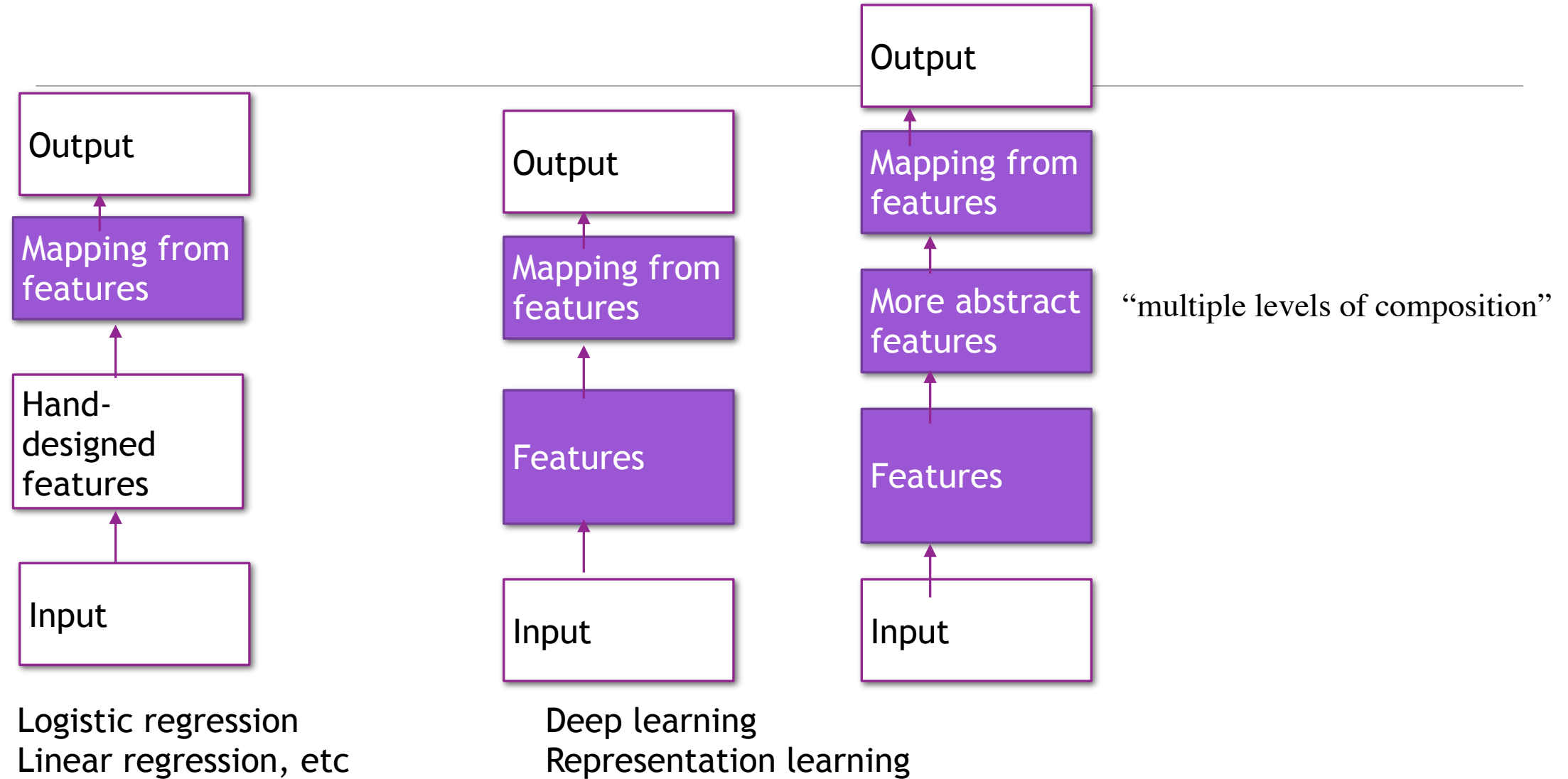


Log scale example:

- If you want  $\alpha \in [0.00001, 10]$  then choose uniformly at random in the range  $r \in [\log_{10} 0.00001, \log_{10} 10]$  then  $\alpha = 10^r$

Learn more at: <https://cs231n.github.io/neural-networks-3/>

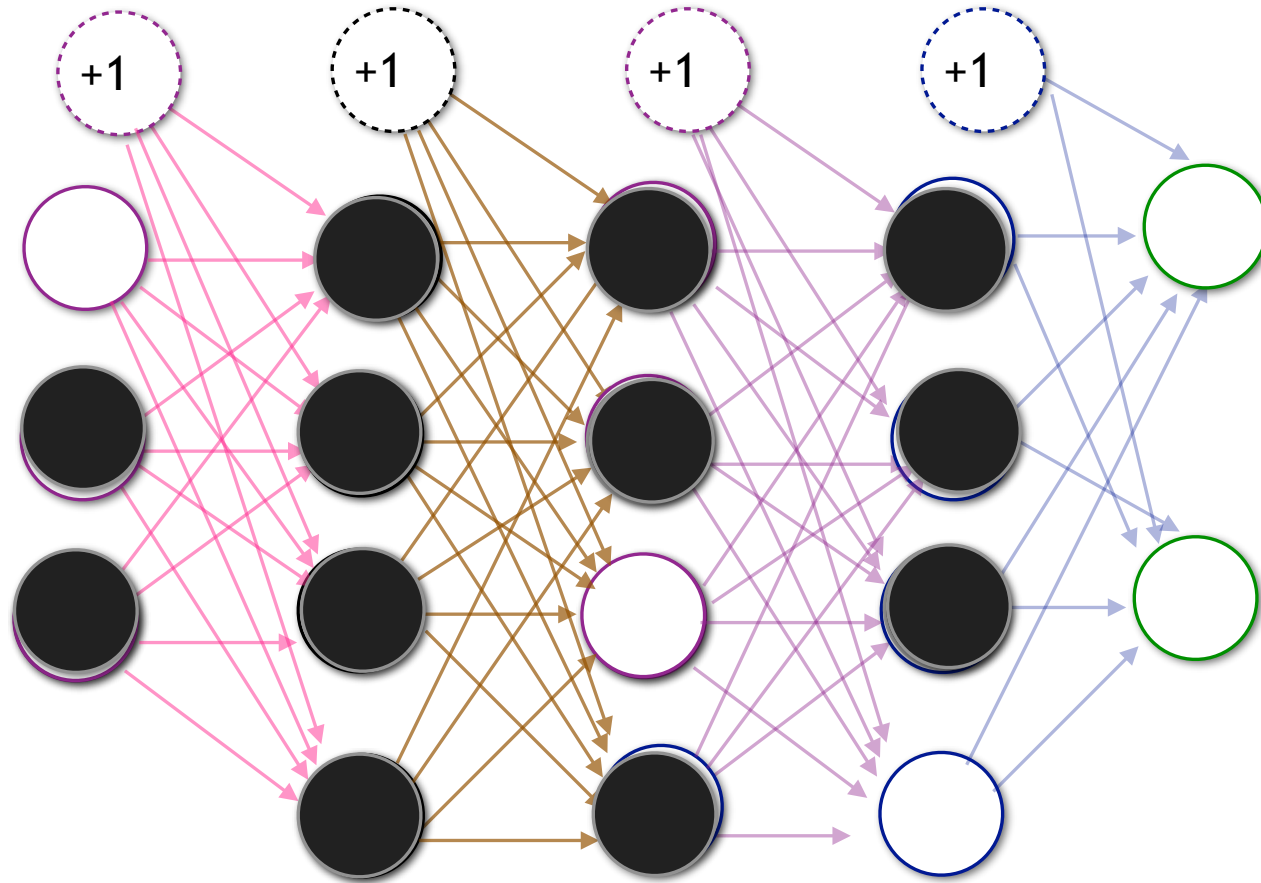
“As of 2016, a rough rule of thumb is that a supervised deep learning algorithm will generally achieve acceptable performance with around 5,000 labeled examples per category, and will match or exceed human performance when trained with a dataset containing at least 10 million labeled examples.” Ian Goodfellow, Yoshua Bengio and Aaron Courville



# Regularization: Dropout

Each iteration, randomly set some activations of hidden neurons to 0 (typically 10% to 50%)

Require the network to have multiple ways to predict (i.e. cannot rely on any 1 node or path through the network)



# Regularization: Early stopping

Stop training before it is overfitting

Typically wait until it is above the min for some time.

