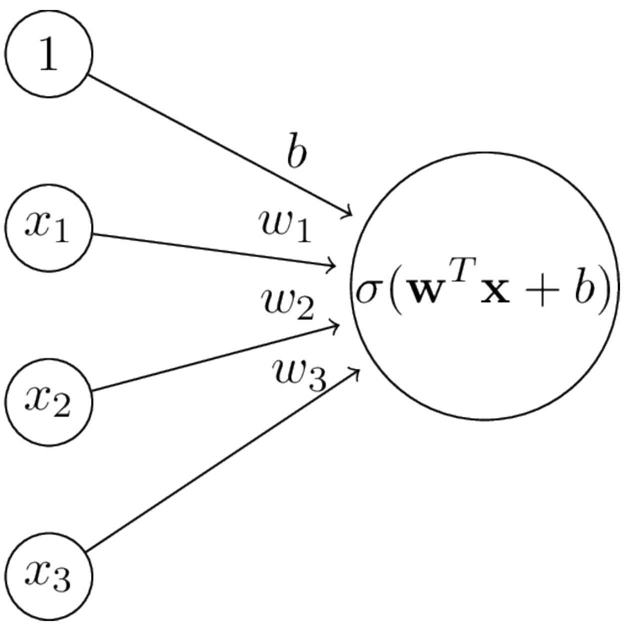
Multi-Layer Neural Networks

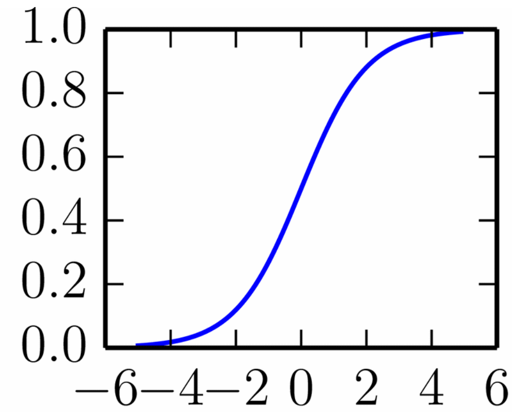
Review: Logistic Regression

<u>"Neuron"</u>



<u>Non-linearity</u>

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



Softmax Activation

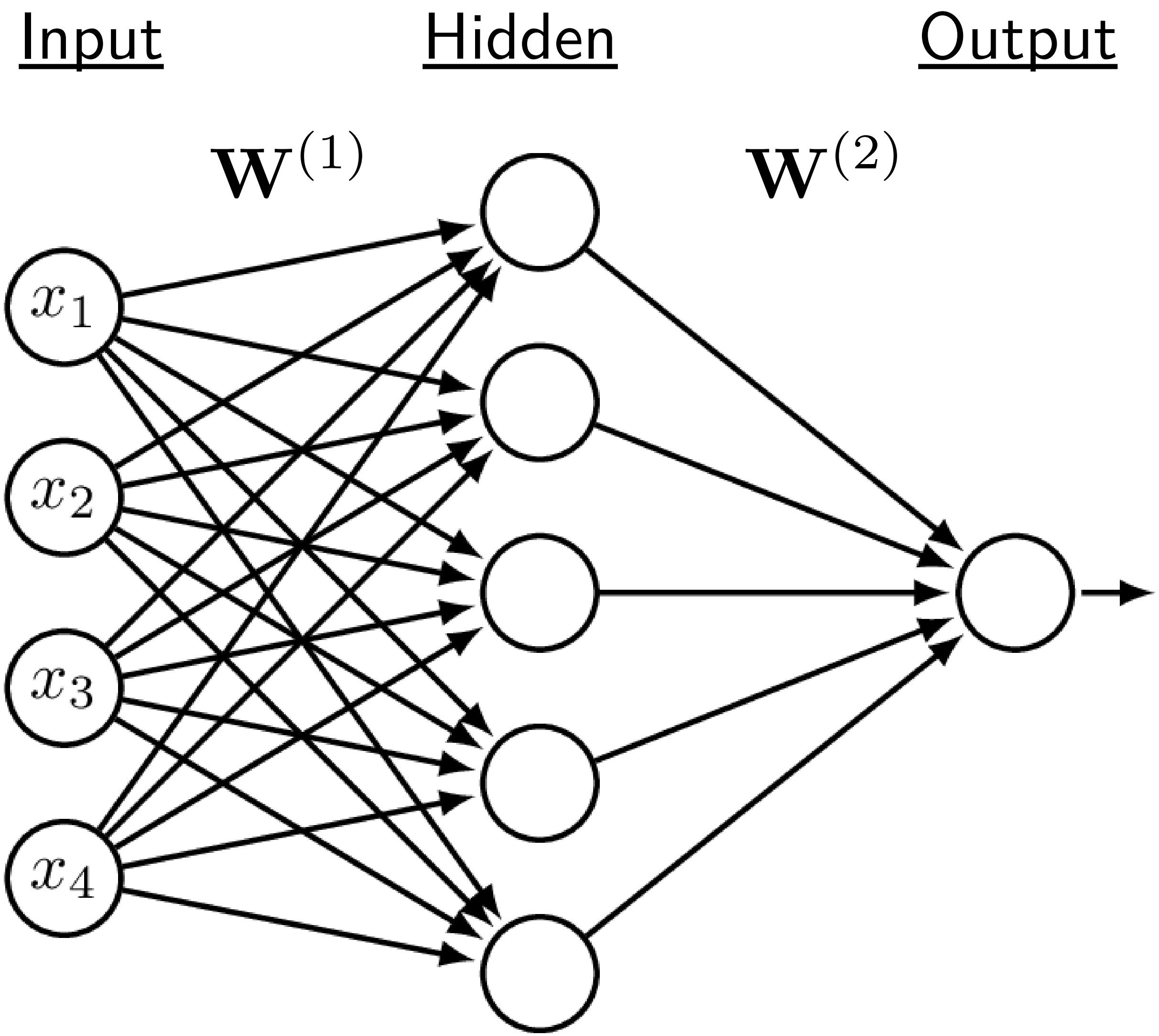
- What if we have a multi-class problem?
- **Softmax** for K classes:

$$\sigma(\mathbf{a})_i = \frac{e^{a_i}}{\sum_{j=1}^{K} e^{a_j}}$$

- The activations a_i are called **logits**.
- Cross-entropy loss function. The target vector $\boldsymbol{y}$ is "one-hot encoded".

$$Loss = -\sum_{i=1}^{K} y_i \log(\sigma(\mathbf{a})_i)$$

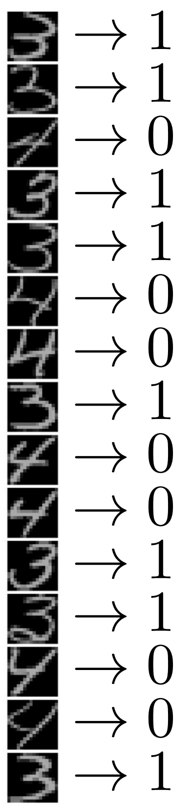
Multi-Layer Networks



Neural Network Example

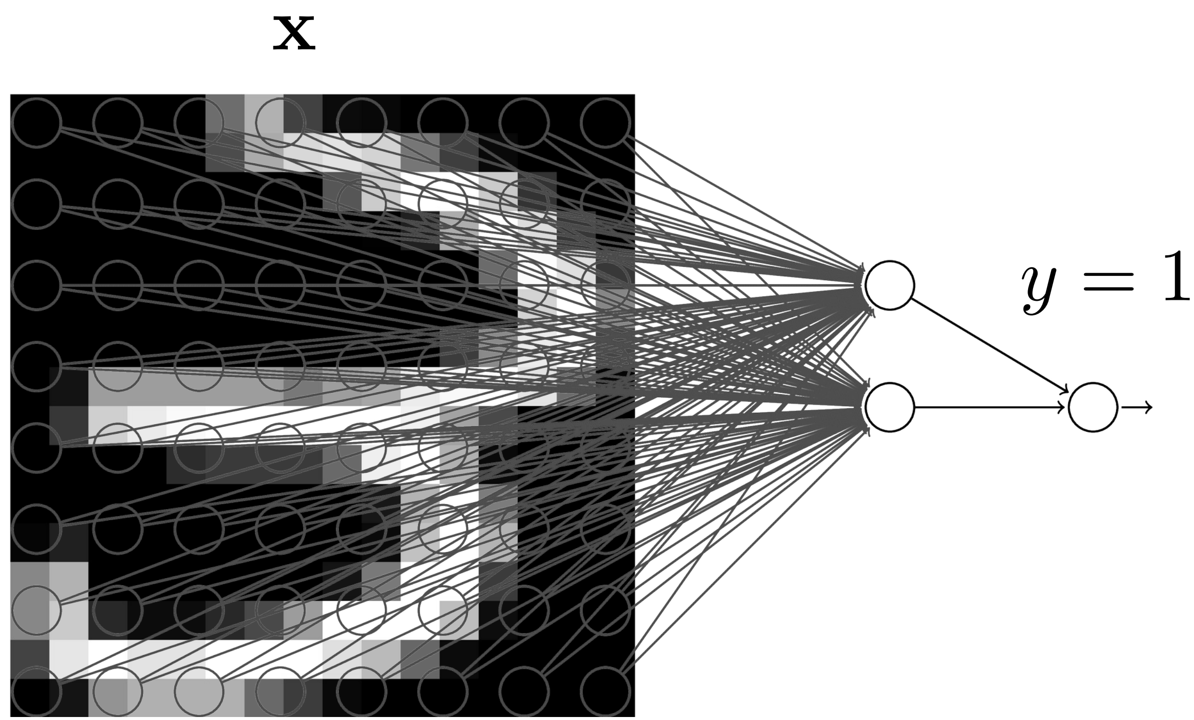
Training Data

$\mathbf{x}$ $\quad y$

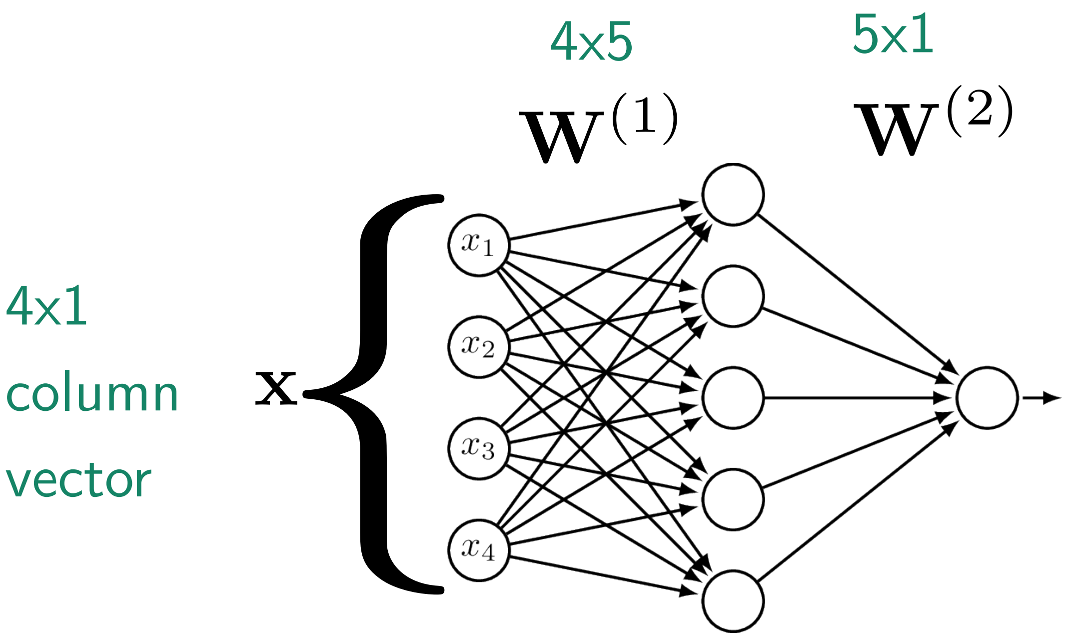


⋮

Network



Computation Example

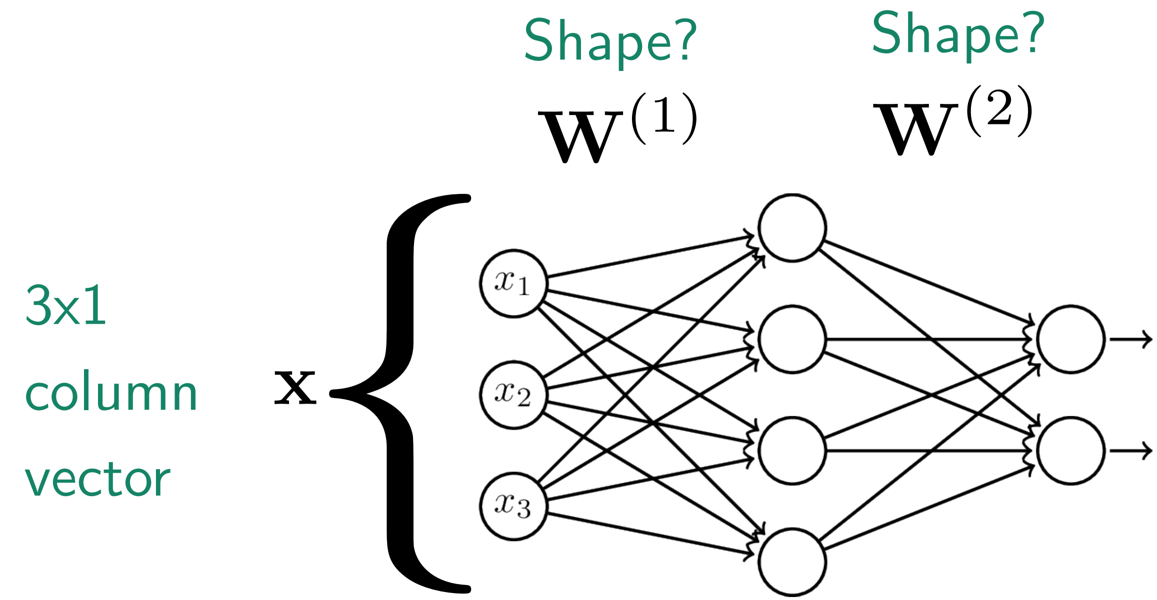


Hidden activation: $h(\mathbf{x}^\mathsf{T} W^{(1)})$

Output activation: $\sigma\left(h(\mathbf{x}^\mathsf{T} W^{(1)}) W^{(2)}\right)$

(h is the non-linearity at the hidden layer. *σ* is the non-linearity at the output. Applied element-wise.)

QUIZ

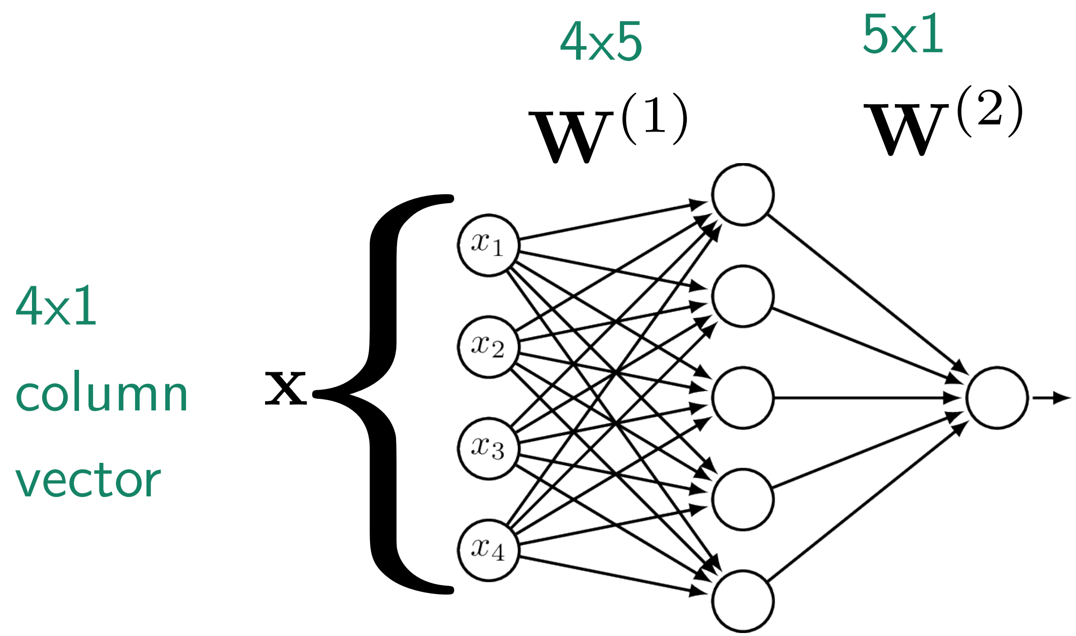


Hidden activation: $h(\mathbf{x}^\mathsf{T} W^{(1)})$

Output activation: $\sigma\left(h(\mathbf{x}^\mathsf{T} W^{(1)}) W^{(2)}\right)$

(h is the non-linearity at the hidden layer. *σ* is the non-linearity at the output. Applied element-wise.)

Bias Weights



Hidden activation: $h(\mathbf{x}^\mathsf{T} W^{(1)} + \mathbf{b}^{(1)})$

Output activation: $\sigma\left(h(\mathbf{x}^\mathsf{T} W^{(1)} + \mathbf{b}^{(1)}) W^{(2)} + b^{(2)}\right)$

(h is the non-linearity at the hidden layer. σ is the non-linearity at the output. Applied element-wise.)

Backpropagation

- Activation at the output layer:

$$a_k = \sigma\left(\sum_j w_{j,k}^{(2)} \, h\left(\sum_i w_{i,j}^{(1)} x_i\right)\right)$$

- Here σ is the activation function at the output layer. Units at the input layer are indexed with *i*, hidden with *j* and output with *k*.
- Error metric, assuming multiple output units:

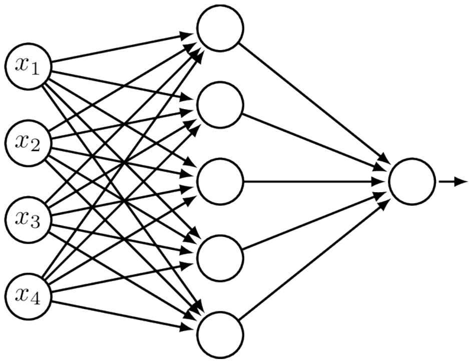
$$Loss = \frac{1}{k}\sum_k \left(y_k - a_k\right)^2 \qquad\qquad Loss = -\sum_{i=1}^{K} y_i \log(\sigma(\mathbf{a})_i)$$

- Now just compute $\dfrac{\partial Loss}{\partial w_{j,k}^{(2)}}$ and $\dfrac{\partial Loss}{\partial w_{i,j}^{(1)}}$.

Backpropagation Algorithm

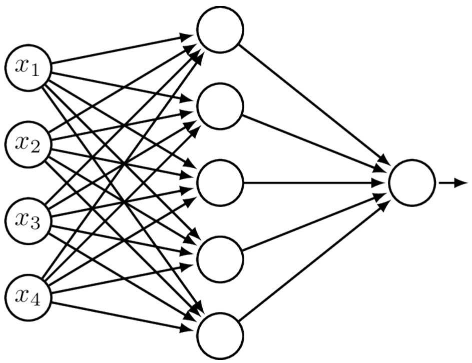
- Forward Pass:

Activation



- Backward Pass:

Error Signal



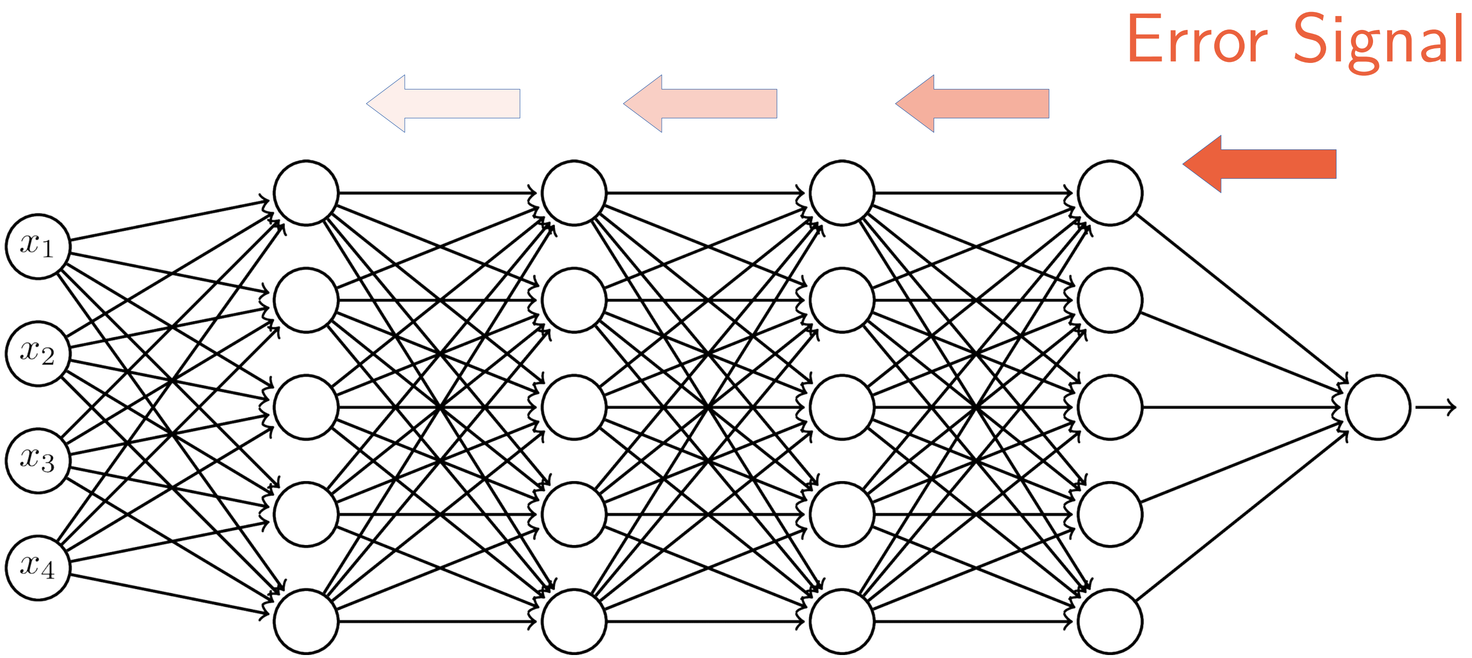
Backpropagation: Some Good News

- Calculating partial derivatives is tedious, but mechanical
- Modern neural network libraries perform automatic differentiation
  - Tensorflow
  - PyTorch
  - Etc.
- The programmer just needs to specify the network structure and the loss function – No need to explicitly write code for performing weight updates
- The computational cost for the backward pass is not much more than the cost for the forward pass

Deep vs. Shallow Networks

- How best to add capacity?
  - More units in a single hidden layer?
    - Three layer networks are universal approximators: with enough units any continuous function can be approximated
    - Adding layers makes the learning problem harder…

Vanishing Gradients



Advantages of Deep Architectures

- There are tasks that require exponentially many hidden units for a three-layer architecture, but only polynomially many with more hidden layers
- The best hand-coded image processing algorithms have deep structure
- The brain has a deep architecture
- MORE SOON.