

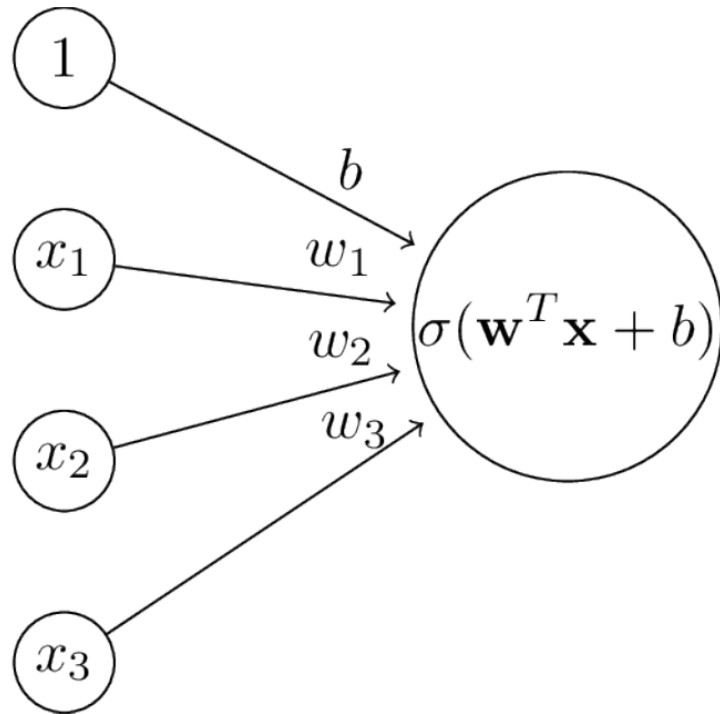
Multi-Layer Neural Networks

CS 445 – Machine Learning

Neural networks · activations · backpropagation

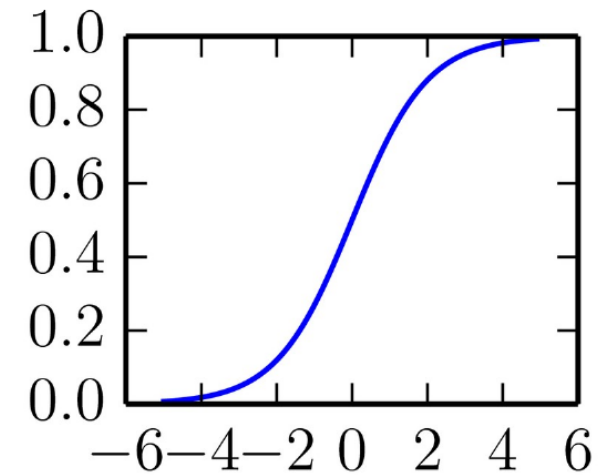
Review: Logistic Regression

“Neuron”

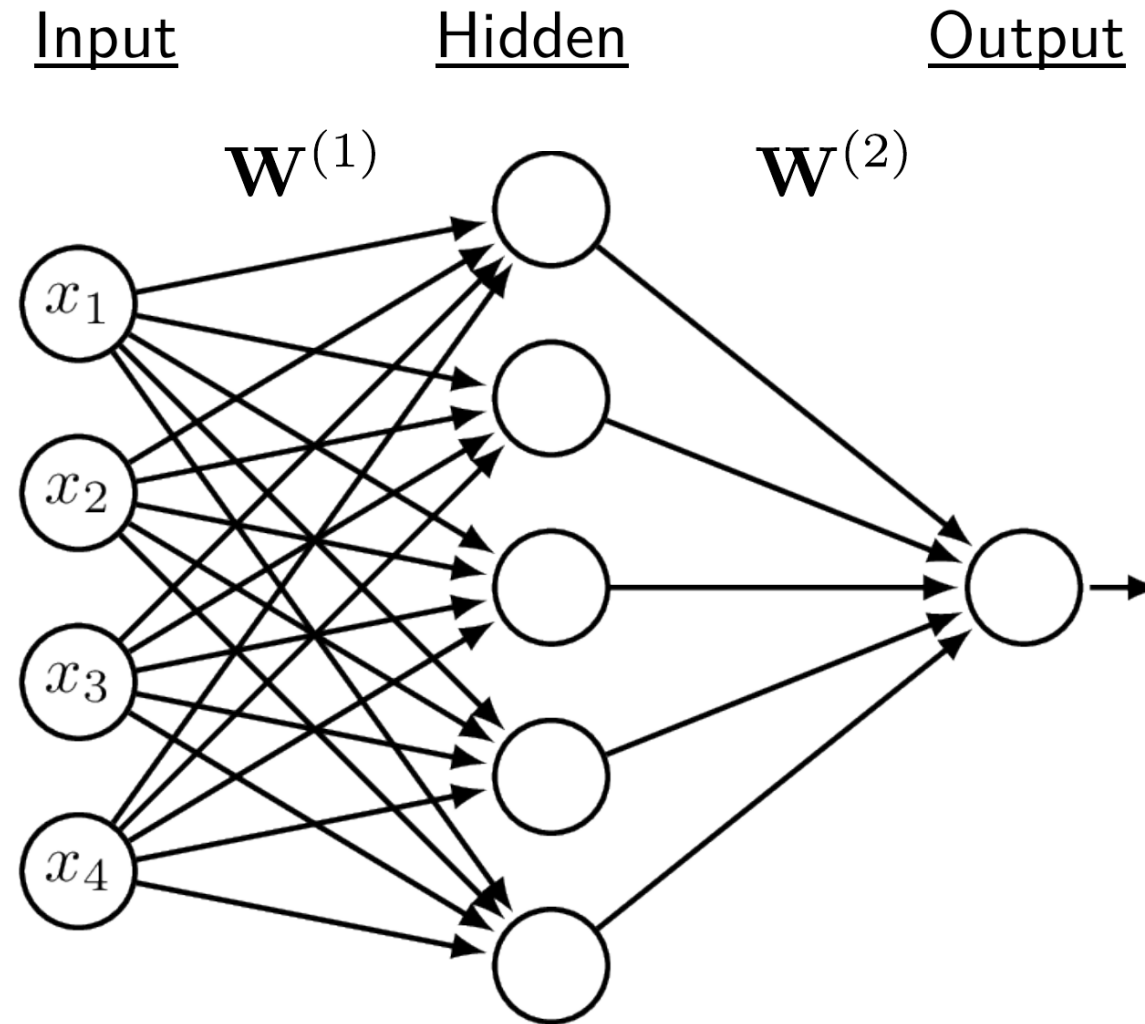


Non-linearity

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



Multi-Layer Networks



Neural Network Example

Training Data

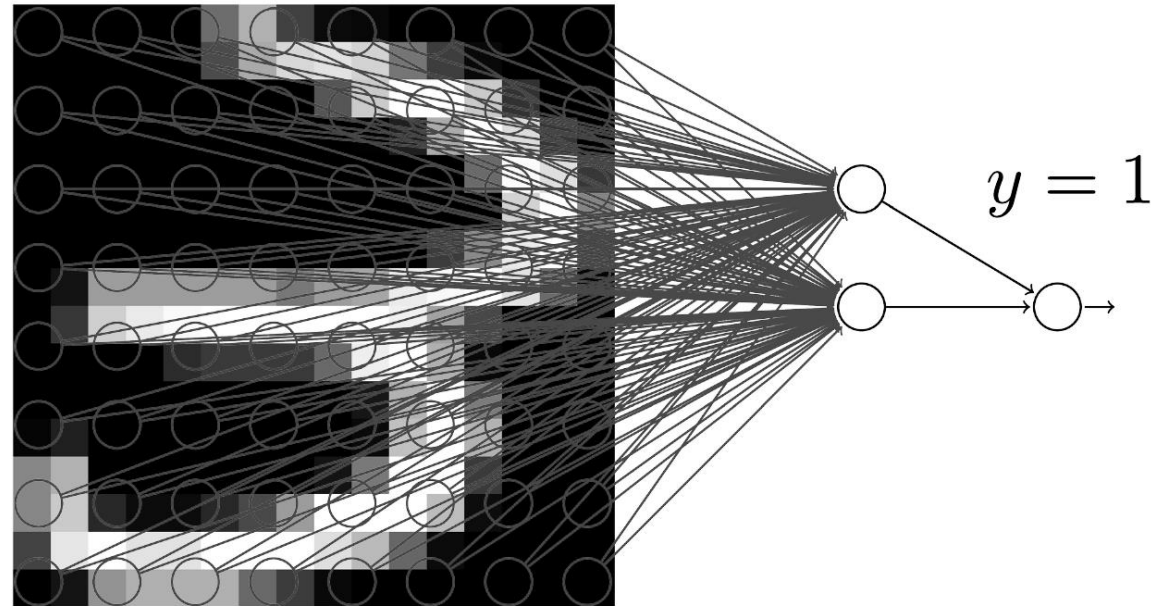
$\mathbf{x}$ y

 → 1
 → 1
 → 0
 → 1
 → 1
 → 0
 → 0
 → 1
 → 0
 → 0
 → 1
 → 1
 → 0
 → 0
 → 1

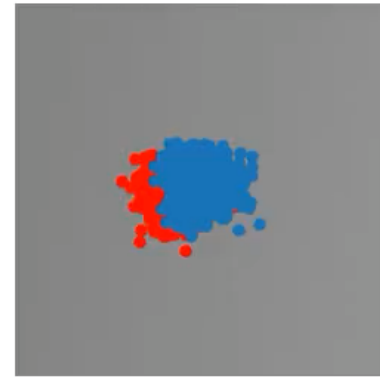
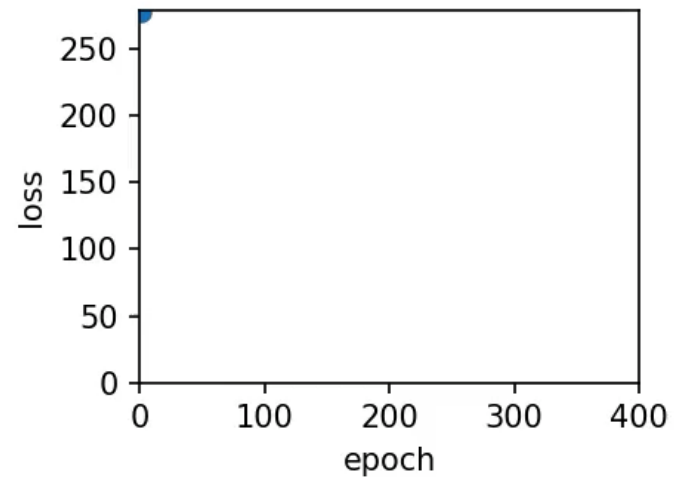
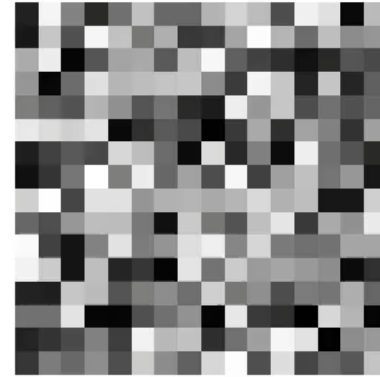
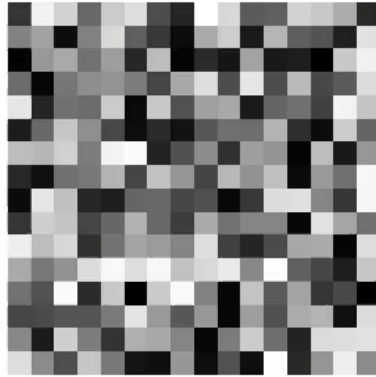
⋮

Network

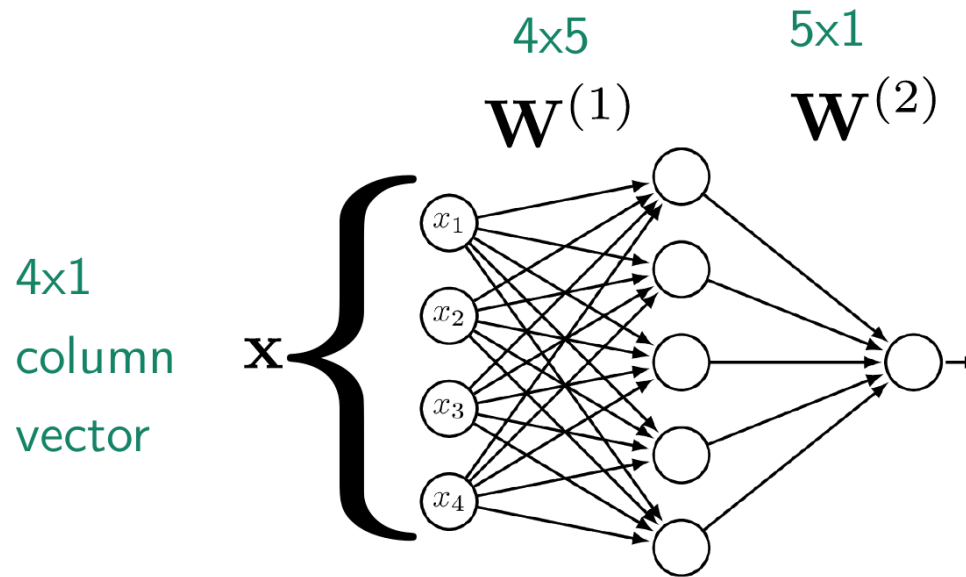
$\mathbf{x}$



Training Example



Computation Example

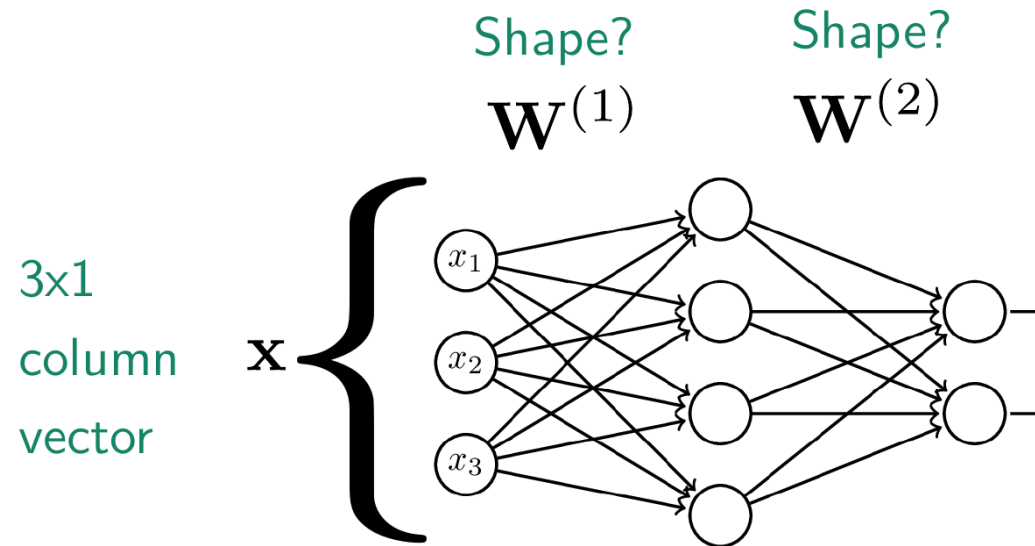


Hidden activation: $h(\mathbf{x}^\top \mathbf{W}^{(1)})$

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

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

Quiz

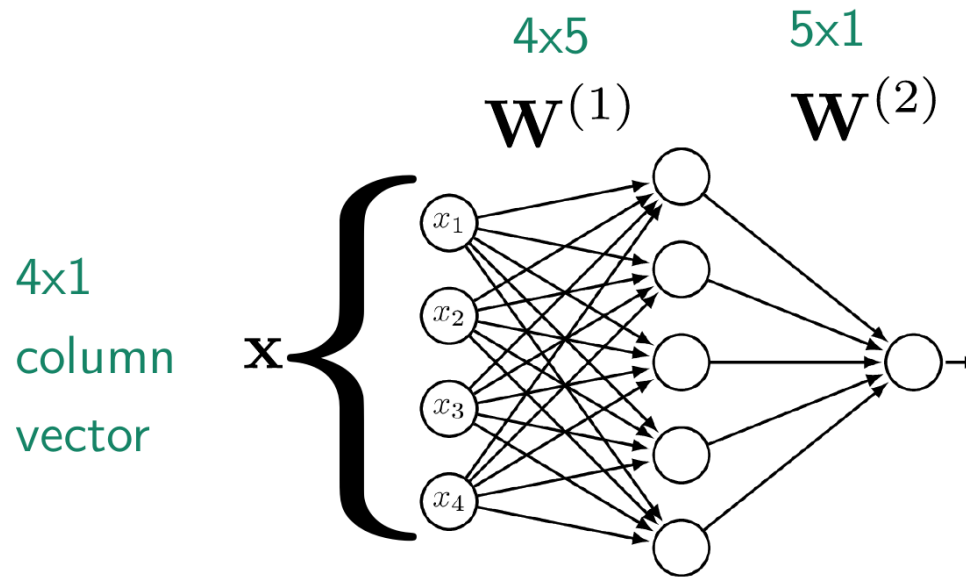


Hidden activation: $h(\mathbf{x}^T \mathbf{W}^{(1)})$

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

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

Bias Weights



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

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

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

Softmax Activation

What if we have a multi-class problem?

Softmax for K classes:

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

The non-squashed activations a_i are called logits.

Cross-entropy loss uses a one-hot encoded target vector y .

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

Gradient Descent Reminder

Define a loss function:

$$L(w, D) = - \sum_{(x_i, y_i) \in D} \sum_{j=1}^K y_{i,j} \log(\sigma(a)_j)$$

Find the gradient of the error function with respect to the weights:

$$\nabla_w L(w, D)$$

Take small steps in the direction of the gradient:

$$w \leftarrow w - \alpha \nabla_w L(w, D)$$

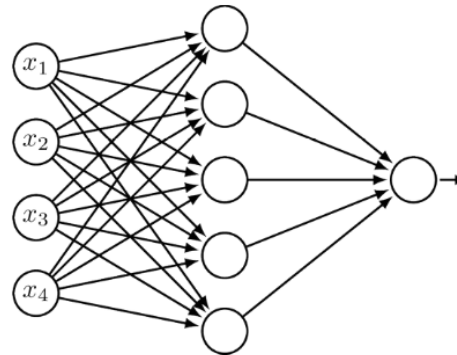
Backpropagation / Reverse Mode Automatic Differentiation

Efficient algorithm for implementing gradient descent in neural networks.

Activation

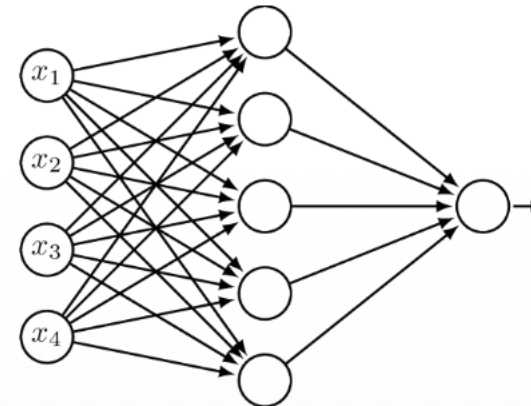


Forward pass:



Error Signal

Backward pass:



Backpropagation / Reverse Mode Automatic Differentiation

Invented:

Linnainmaa (1970): cumulative rounding error as a Taylor expansion.

First applied to neural networks:

Werbos (1982): advances in nonlinear sensitivity analysis.

Shown to create useful representations in the hidden layer:

Rumelhart, Hinton, and Williams (1985): learning internal representations by error propagation.

Detailed history:

Schmidhuber (2015): Deep learning in neural networks: an overview.

Modern ML libraries automate the backward pass.

PyTorch and TensorFlow handle automatic differentiation for us.

Deep vs. Shallow Networks

How best to add capacity?

More units in a single hidden layer?

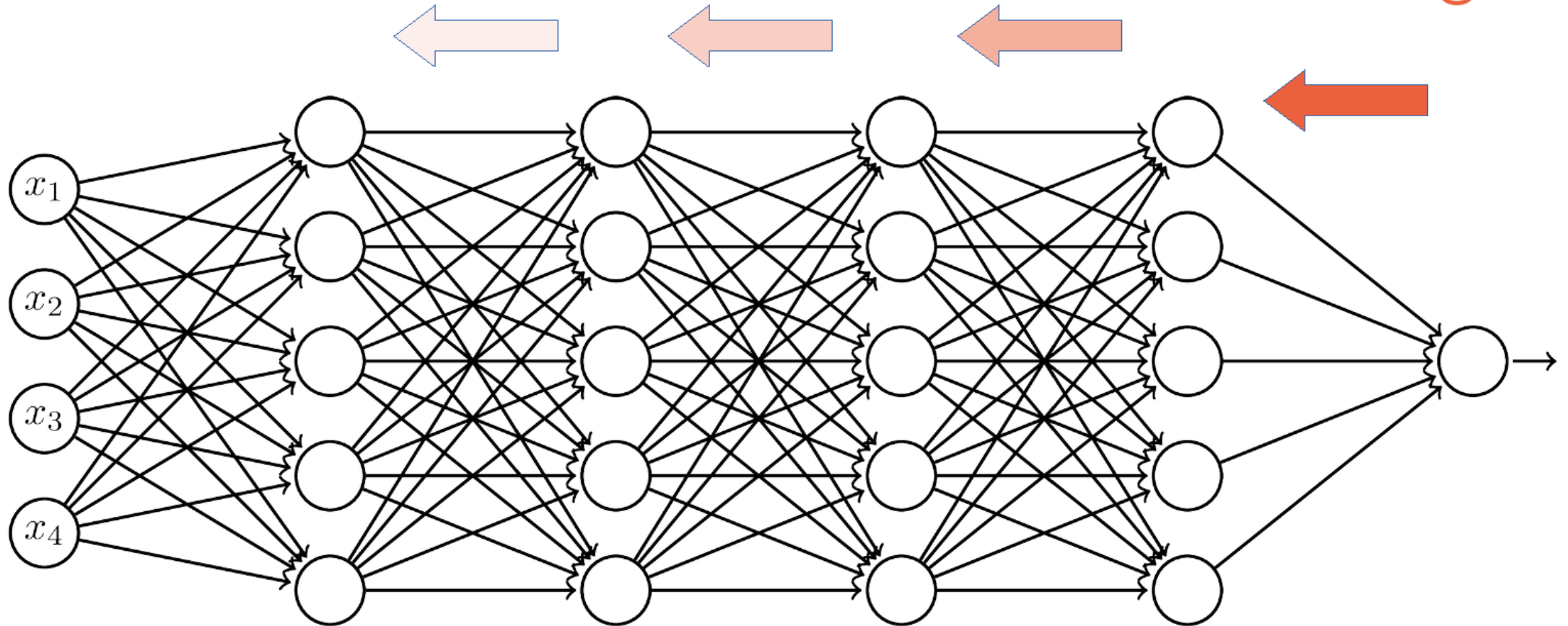
Three-layer networks are universal approximators: with enough units, any continuous function can be approximated.

Adding layers makes the learning problem harder...

Depth can add representational power while also making optimization harder.

Vanishing Gradients

Error Signal



Advantages of Deep Architectures

Some tasks 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.