Slides adapted from Prof Carpuat and Duraiswami

CMSC 422 Introduction to Machine Learning

Lecture 19 Neural Networks II

Furong Huang / furongh@umd.edu

UNIVERSITY OF MARYLAND

Neural Networks

Last time

What are Neural Networks? (key components)

How to make a prediction given an input?

Why are neural networks powerful?

Neural Networks

Last time

- What are Neural Networks?
  - Multilayer perceptron
- How to make a prediction given an input?
  - Simple matrix operations + nonlinearities
- Why are neural networks powerful?
  - Universal function approximators!

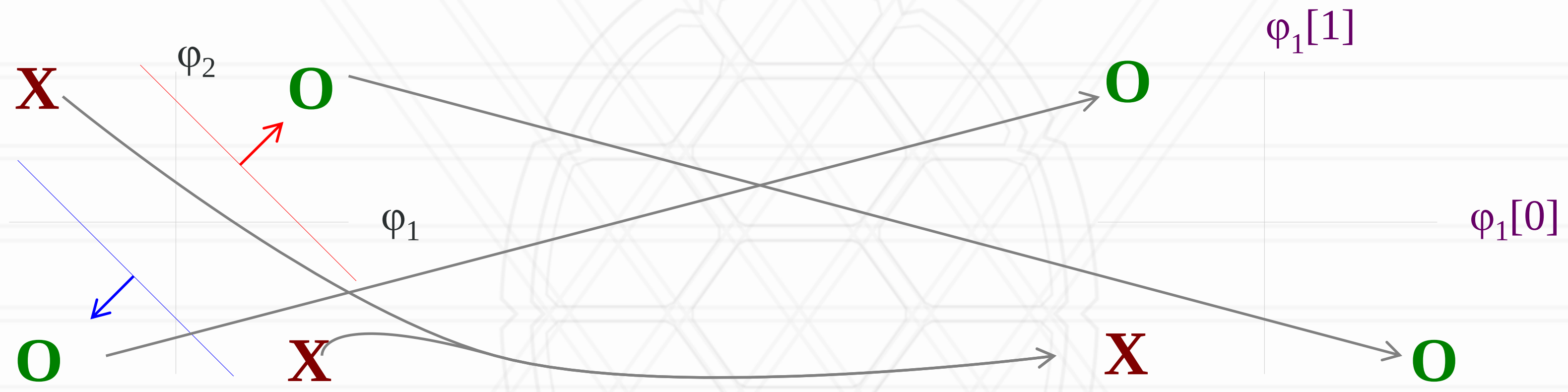
Today

- how to train neural networks?

Example: a neural network to solve the XOR problem

$\varphi_0(x_1) = \{-1, 1\}$ $\varphi_0(x_2) = \{1, 1\}$

$\varphi_1(x_3) = \{-1, 1\}$



$\varphi_1(x_1) = \{-1, -1\}$ $\varphi_1(x_2) = \{1, -1\}$

$\varphi_0(x_3) = \{-1, -1\}$ $\varphi_0(x_4) = \{1, -1\}$

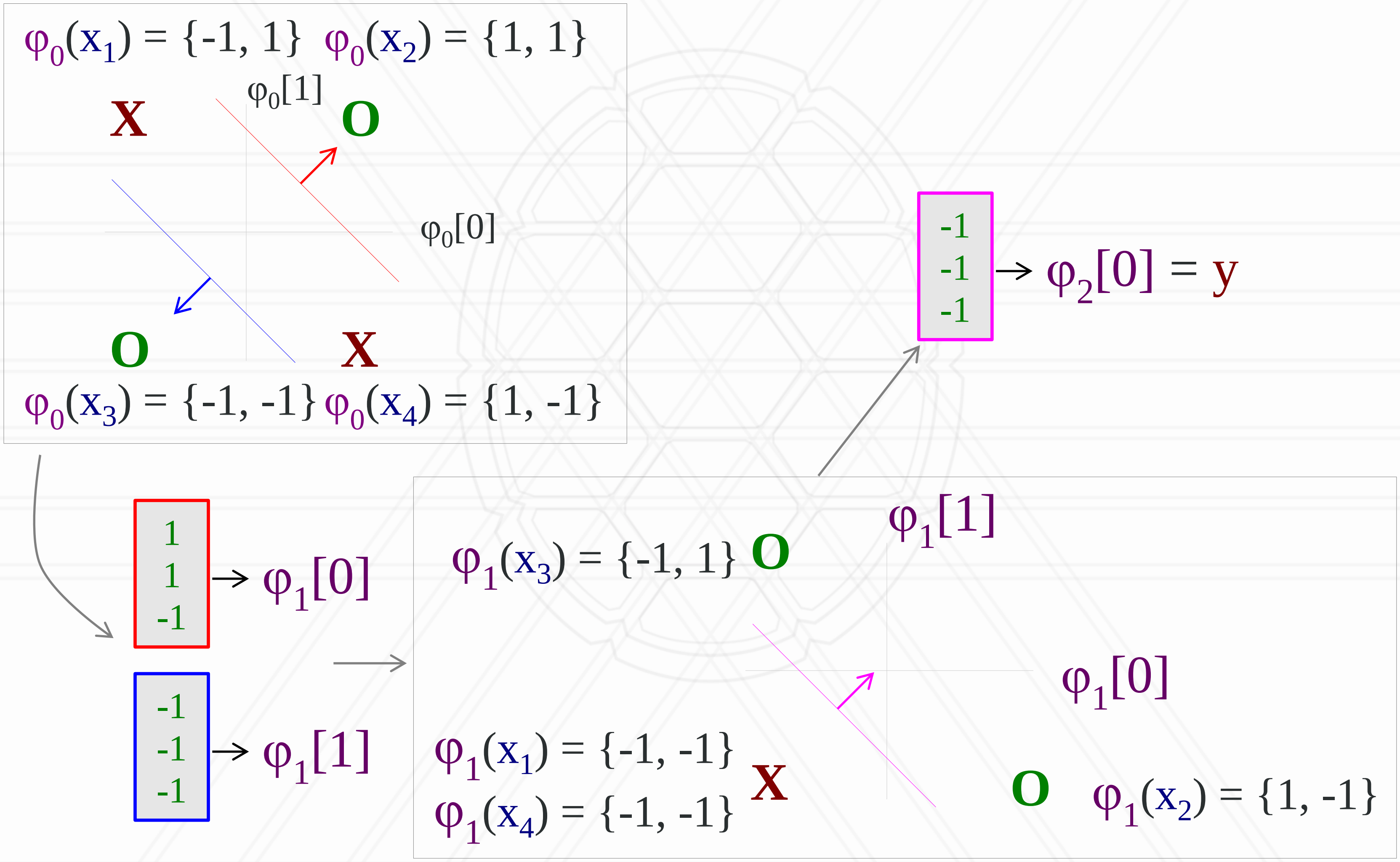
$\varphi_1(x_4) = \{-1, -1\}$

$\boxed{\begin{matrix} 1 \\ 1 \\ -1 \end{matrix}} \rightarrow \varphi_1[0]$

$\boxed{\begin{matrix} -1 \\ -1 \\ -1 \end{matrix}} \rightarrow \varphi_1[1]$

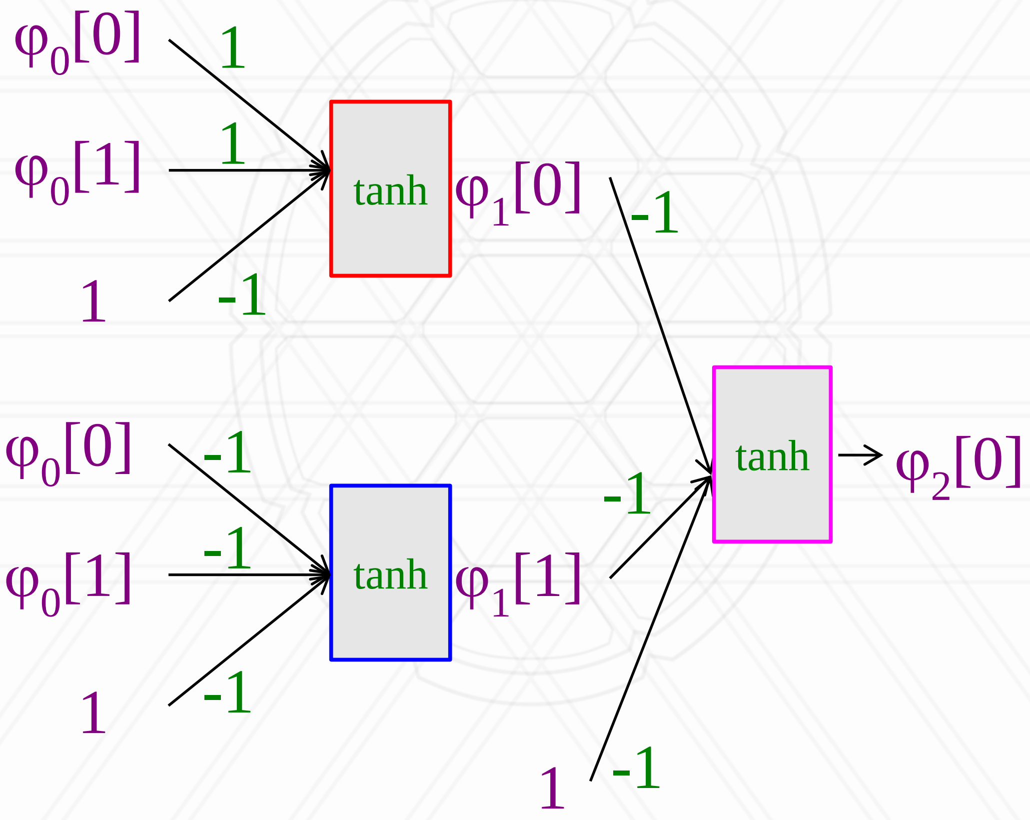
Example

- In new space, the examples are linearly separable!



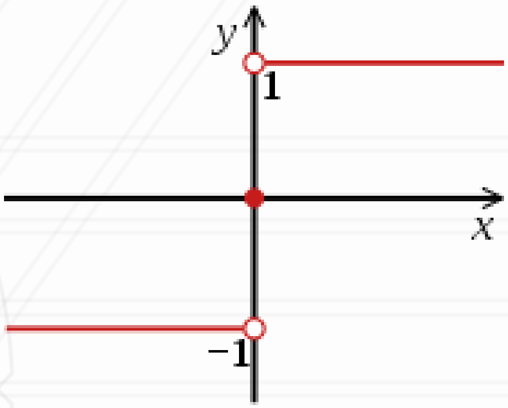
Example

- The final net

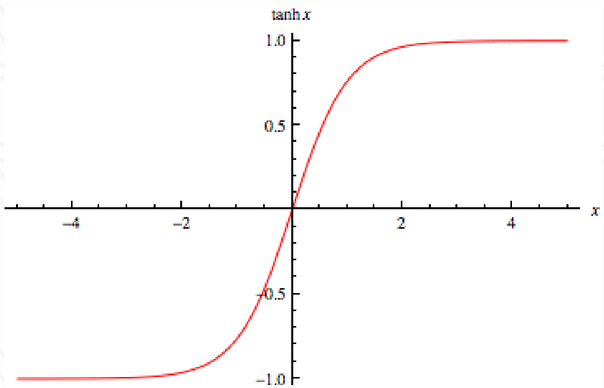


Activation Functions

- Examples:
  - Sign: $sign(x) = \begin{cases} -1, x < 0 \\ 0, x = 0 \\ 1, x > 0 \end{cases}$

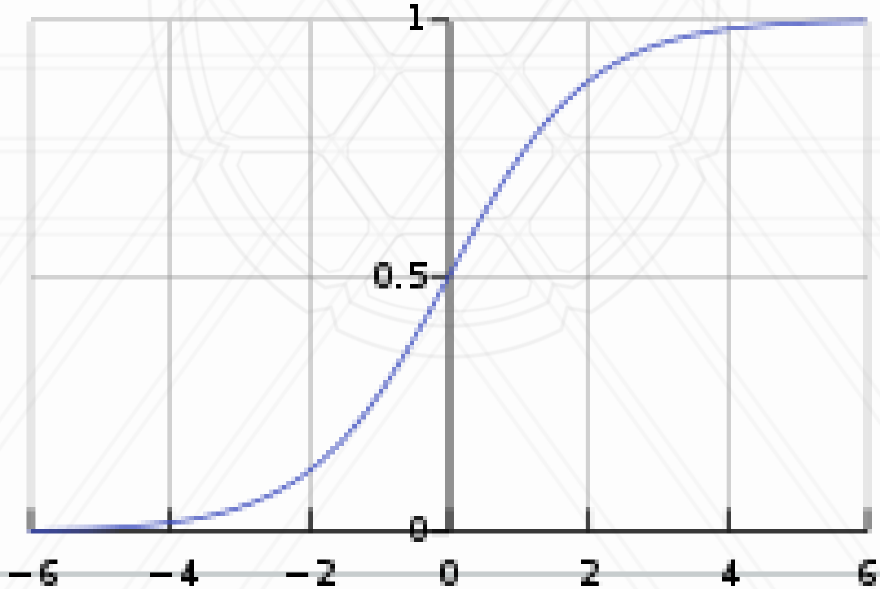


  - Tanh: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$
    - Derivative: $\frac{\mathrm{d}}{dx}\tanh(x) = 1 - \tanh^2(x)$



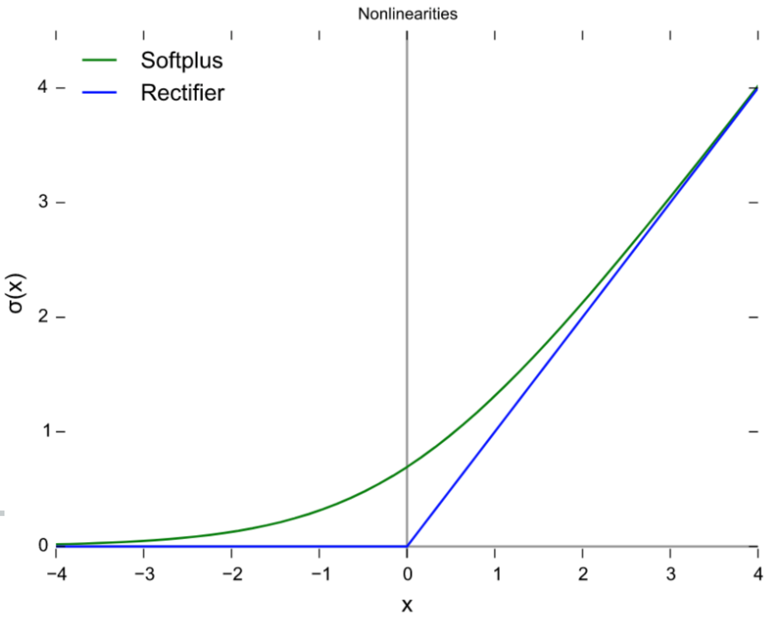
Activation Functions

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$, logistic function
  - Derivative: $\frac{d}{dx}\sigma(x) = \frac{e^x}{1+e^x} = \sigma(x)\,(1 - \sigma(x))$



Activation Functions

- Relu: $relu(x) = x^+ = \max(0, x)$ rectified linear unit
  - A smooth approximation: softplus
    $$f(x) = \log(1 + \exp x)$$
  - The derivative of softplus is logistic function $f'(x) = \frac{e^x}{1+e^x}$



Exercise

- Learn softmax function https://en.wikipedia.org/wiki/Softmax_function

Forward Propagation: given input x, compute network output

Algorithm 24 TwoLayerNetworkPredict($\mathbf{W}, v, \hat{x}$)

```
for i = 1 to number of hidden units do
    h_i ← tanh(w_i · x̂)          // compute activation of hidden unit i
end for
return v · h                      // compute output unit
```

Neural Network Training

Backpropagation algorithm

=

Gradient descent + Chain rule

Recall: gradient descent for linear classifiers

Objective function to minimize

Number of steps

Step size

Algorithm 22 GradientDescent($\mathcal{F}$, K, η_1, ...)

1: $z^{(0)} \leftarrow \langle 0, 0, \dots, 0 \rangle$ // initialize variable we are optimizing
2: **for** $k = 1 \dots K$ **do**
3: $\quad g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$ // compute gradient at current location
4: $\quad z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // take a step down the gradient
5: **end for**
6: **return** $z^{(K)}$

What's our Training Objective?

We'll consider the following objective

$$\min_{\mathbf{W},v} \sum_n \frac{1}{2}\left(y_n - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x}_n)\right)^2$$

i.e. our goal is to find parameters **W**, v that minimize squared error

Other objectives are possible (e.g., other loss functions, add regularizer)

Backprop in a 2-layer network

Algorithm 25 $\textsc{TwoLayerNetworkTrain}(\mathbf{D}, \eta, K, \mathit{MaxIter})$

$\mathbf{W} \leftarrow D{\times}K$ matrix of small random values // initialize input layer weights
$\boldsymbol{v} \leftarrow K$-vector of small random values // initialize output layer weights
for $\mathit{iter} = 1 \dots \mathit{MaxIter}$ **do**
    $\mathbf{G} \leftarrow D{\times}K$ matrix of zeros // initialize input layer gradient
    $\boldsymbol{g} \leftarrow K$-vector of zeros // initialize output layer gradient
    for all $(x,y) \in \mathbf{D}$ **do**
        for $i = 1$ **to** K **do**
            $a_i \leftarrow \boldsymbol{w}_i \cdot \hat{\boldsymbol{x}}$
            $h_i \leftarrow \tanh(a_i)$ // compute activation of hidden unit i
        end for
        $\hat{y} \leftarrow \boldsymbol{v} \cdot \boldsymbol{h}$ // compute output unit
        $e \leftarrow y - \hat{y}$ // compute error
        $\boldsymbol{g} \leftarrow \boldsymbol{g} - e\boldsymbol{h}$ // update gradient for output layer
        for $i = 1$ **to** K **do**
            $\mathbf{G}_i \leftarrow \mathbf{G}_i - e v_i (1 - \tanh^2(a_i)) \boldsymbol{x}$ // update gradient for input layer
        end for
    end for
    $\mathbf{W} \leftarrow \mathbf{W} - \eta \mathbf{G}$ // update input layer weights
    $\boldsymbol{v} \leftarrow \boldsymbol{v} - \eta \boldsymbol{g}$ // update output layer weights
end for
return $\mathbf{W}, \boldsymbol{v}$

Backprop in a 2-layer network

Algorithm 25 TwoLayerNetworkTrain($\mathbf{D}$, η, K, $MaxIter$)

$\mathbf{W} \leftarrow D{\times}K$ matrix of small random values // initialize input layer weights
$v \leftarrow K$-vector of small random values // initialize output layer weights
for $iter = 1 \ldots MaxIter$ **do**
  $\mathbf{G} \leftarrow D{\times}K$ matrix of zeros // initialize input layer gradient
  $g \leftarrow K$-vector of zeros // initialize output layer gradient

  Compute Gradient G and g

  $\mathbf{W} \leftarrow \mathbf{W} - \eta\mathbf{G}$ // update input layer weights
  $v \leftarrow v - \eta g$ // update output layer weights
end for
return $\mathbf{W}$, v

What's our Training Objective?

We'll consider the following objective

$$\min_{\mathbf{W},v} \sum_n \frac{1}{2}\left(y_n - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x}_n)\right)^2$$

i.e. our goal is to find parameters **W**, v that minimize squared error

Other objectives are possible (e.g., other loss functions, add regularizer)

Gradient of objective w.r.t. output layer weights v

$$\nabla_v = -\sum_n e_n \boldsymbol{h}_n$$

Error at example n:
$y_n - \widehat{y_n}$

Vector of activations of hidden units for example n

Gradient of objective w.r.t. hidden unit weights w_i

$$\mathcal{L}(\mathbf{W}) = \frac{1}{2}\left(y - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x})\right)^2$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_i} = \frac{\partial \mathcal{L}}{\partial f_i}\frac{\partial f_i}{\partial \boldsymbol{w}_i}$$

Chain rule

$$\frac{\partial \mathcal{L}}{\partial f_i} = -\left(y - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x})\right) v_i = -e v_i$$

$$\frac{\partial f_i}{\partial \boldsymbol{w}_i} = f'(\boldsymbol{w}_i \cdot \boldsymbol{x})\boldsymbol{x}$$

$$\nabla_{\boldsymbol{w}_i} = -e v_i f'(\boldsymbol{w}_i \cdot \boldsymbol{x})\boldsymbol{x}$$

 (This is on one example only)

Backprop in a 2-layer network

Algorithm 25 $\textsc{TwoLayerNetworkTrain}(\mathbf{D}, \eta, K, \mathit{MaxIter})$

$\mathbf{W} \leftarrow D{\times}K$ matrix of small random values // initialize input layer weights
$\boldsymbol{v} \leftarrow K$-vector of small random values // initialize output layer weights
for $\mathit{iter} = 1 \dots \mathit{MaxIter}$ **do**
  $\mathbf{G} \leftarrow D{\times}K$ matrix of zeros // initialize input layer gradient
  $\boldsymbol{g} \leftarrow K$-vector of zeros // initialize output layer gradient
  for all $(\boldsymbol{x},y) \in \mathbf{D}$ **do**
    for $i = 1$ **to** K **do**
      $a_i \leftarrow \boldsymbol{w}_i \cdot \hat{\boldsymbol{x}}$
      $h_i \leftarrow \tanh(a_i)$ // compute activation of hidden unit i
    end for
    $\hat{y} \leftarrow \boldsymbol{v} \cdot \boldsymbol{h}$ // compute output unit
    $e \leftarrow y - \hat{y}$ // compute error
    $\boldsymbol{g} \leftarrow \boldsymbol{g} - e\boldsymbol{h}$ // update gradient for output layer
    for $i = 1$ **to** K **do**
      $\mathbf{G}_i \leftarrow \mathbf{G}_i - e v_i (1 - \tanh^2(a_i)) \boldsymbol{x}$ // update gradient for input layer
    end for
  end for
  $\mathbf{W} \leftarrow \mathbf{W} - \eta \mathbf{G}$ // update input layer weights
  $\boldsymbol{v} \leftarrow \boldsymbol{v} - \eta \boldsymbol{g}$ // update output layer weights
end for
return $\mathbf{W}, \boldsymbol{v}$

Forward propagation

Update gradients

Update parameters

Tricky issues with neural network training

Sensitive to initialization

- Objective is non-convex, many local optima
- In practice: start with random values rather than zeros

Many other hyperparameters

- Number of hidden units (and potentially hidden layers)
- Gradient descent learning rate
- Stopping criterion

Neural networks vs. linear classifiers

Advantages of Neural Networks:

More expressive

Less feature engineering

Inconvenients of Neural Networks:

Harder to train

Harder to interpret

Neural Network Architectures

We focused on a **2-layer feedforward** network

Other architectures are possible

- More than 2 layers (aka deep learning)
- Recurrent network (i.e. network has cycles)
- Can still be trained with backpropagation
  - But more issues arise when networks get more complex (e.g., vanishing gradients)

Try different architectures and training parameters here:

http://playground.tensorflow.org

What you should know

What are Neural Networks?

- Multilayer perceptron

How to make a prediction given an input?

- Forward propagation: Simple matrix operations + nolinearities

Why are neural networks powerful?

- Universal function approximators!

How to train neural networks?

- The backpropagation algorithm

Furong Huang
4124 IRB, College Park, MD 20740
301.405.8010 / furongh@umd.edu