Neural Networks -2

CMSC 422

Slides adapted from Prof. CARPUAT, and Prof. Rosenberg, HKUST

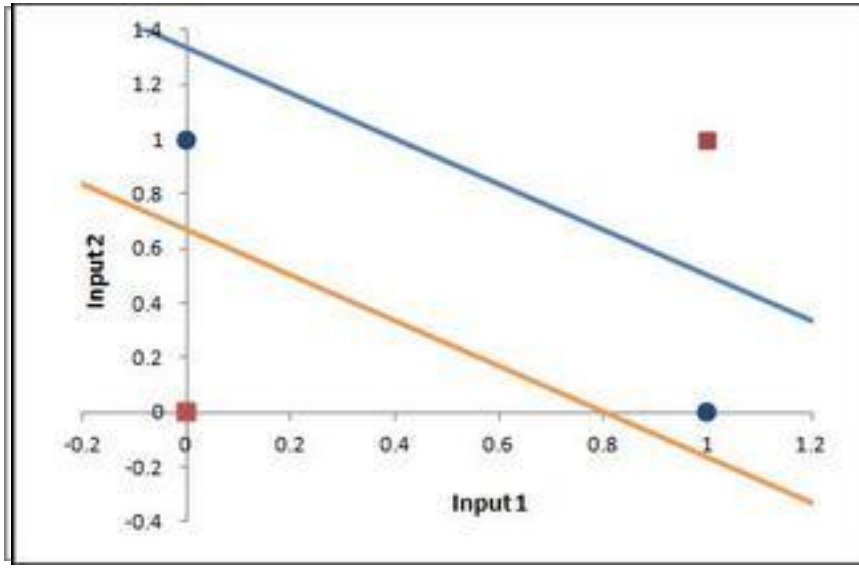
Neural Networks

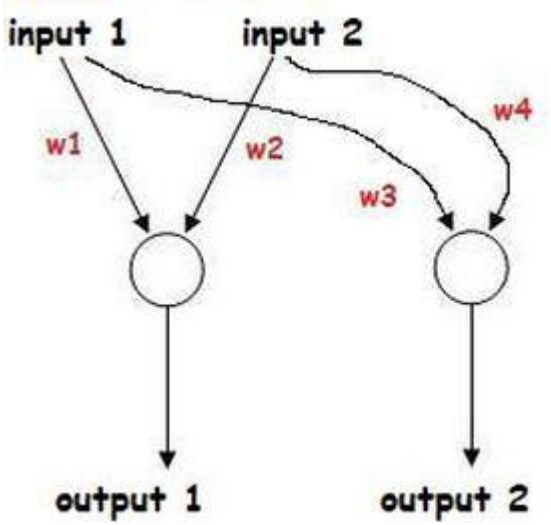
- Last time
  - What are Neural Networks?
  - How to make a prediction given an input?
  - Why are neural networks powerful?

The XOR example

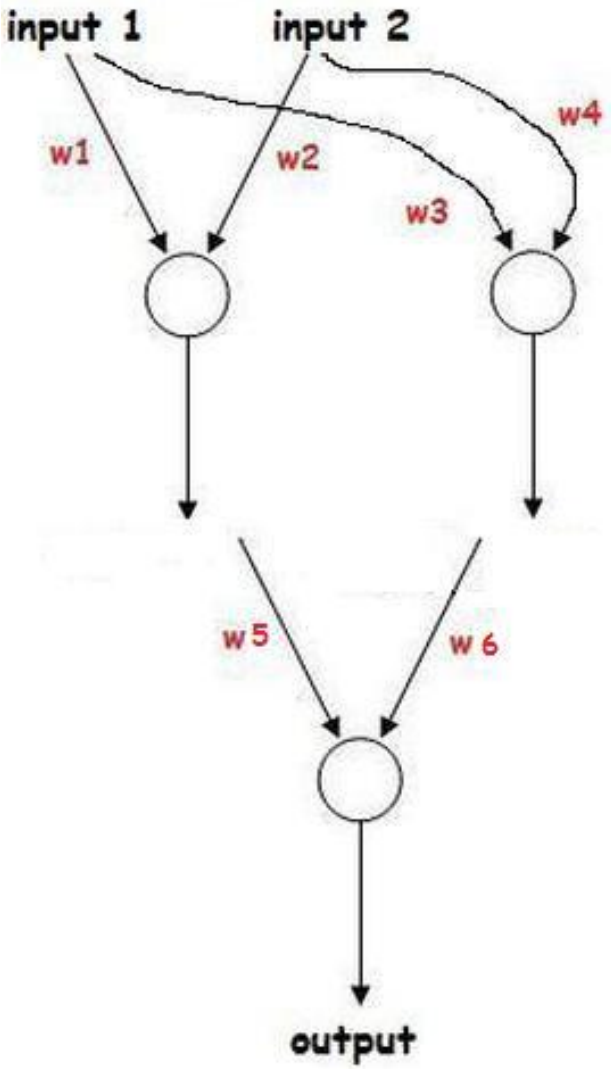
The example is not linearly separable, but two lines can separate it

	input 1	input 2	desired output
object 1	0	0	0
object 2	0	1	1
object 3	1	0	1
object 4	1	1	0



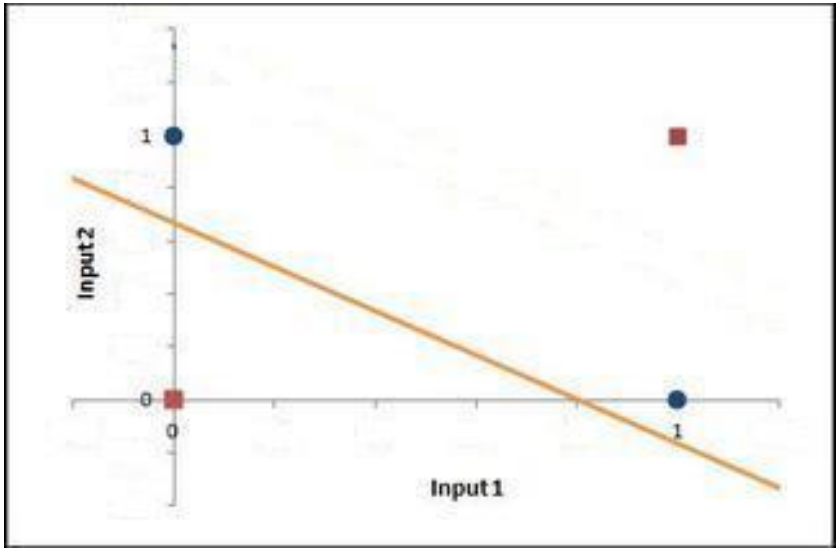


Note that both neurons are connected to THE SAME input sensors and both drawing lines on the same input space to make decisions about the same inputs. They need to work together. We also have two outputs , which is a problem. We want the neural network to categorize the objects it sees into just two groups, but having two outputs that can each be either 1 or 0 gives us four combinations: [00, 01, 10, 11]

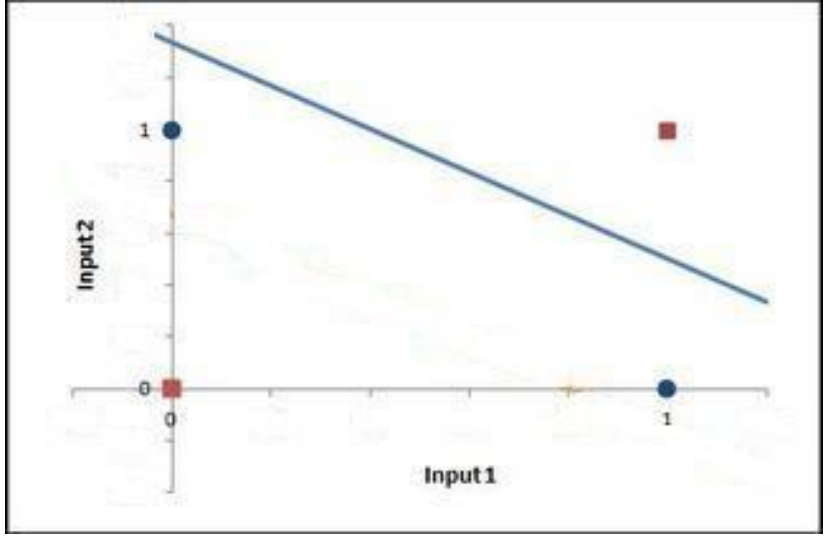


The first layer (the two neurons) draws two lines through input space, while the second layer (the single neuron) draws one line through a new space defined by output of the previous layer

First Layer



	input 1	input 2	output
object 1	0	0	0
object 2	0	1	1
object 3	1	0	1
object 4	1	1	1



	input 1	input 2	output
object 1	0	0	0
object 2	0	1	0
object 3	1	0	0
object 4	1	1	1

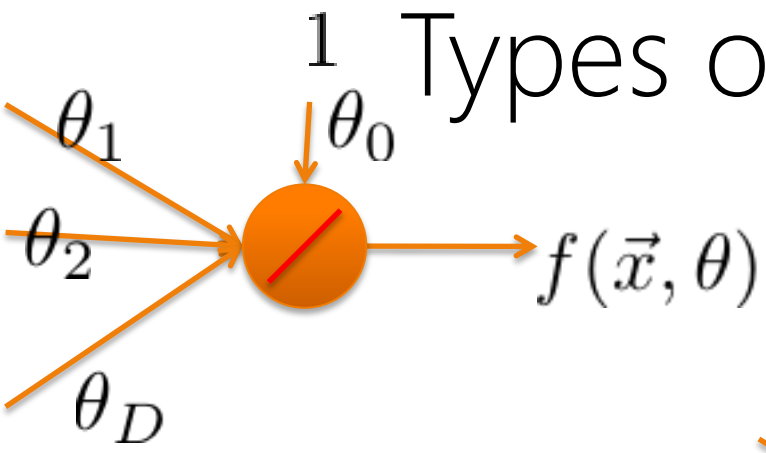
Putting it together

	input 1	input 2	output
object 1	0	0	0
object 2	0	1	1
object 3	1	0	1
object 4	1	1	1

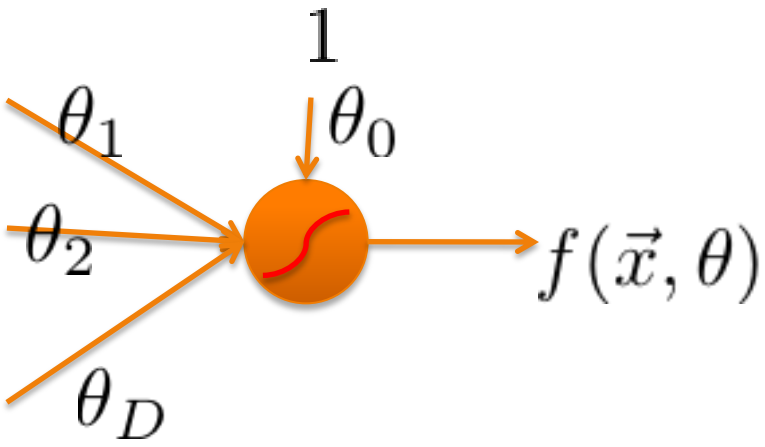
	input 1	input 2	output
object 1	0	0	0
object 2	0	1	0
object 3	1	0	0
object 4	1	1	1

	input 1	input 2	output
object 1	0	0	0
object 2	1	0	1
object 3	1	0	1
object 4	1	1	0

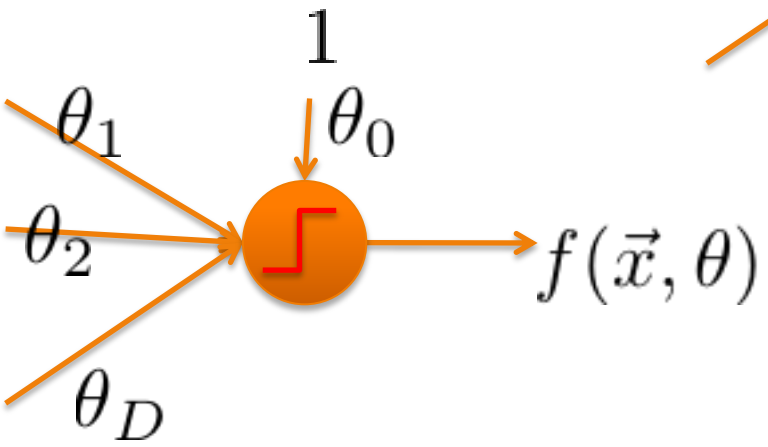
Types of Neurons



Linear Neuron



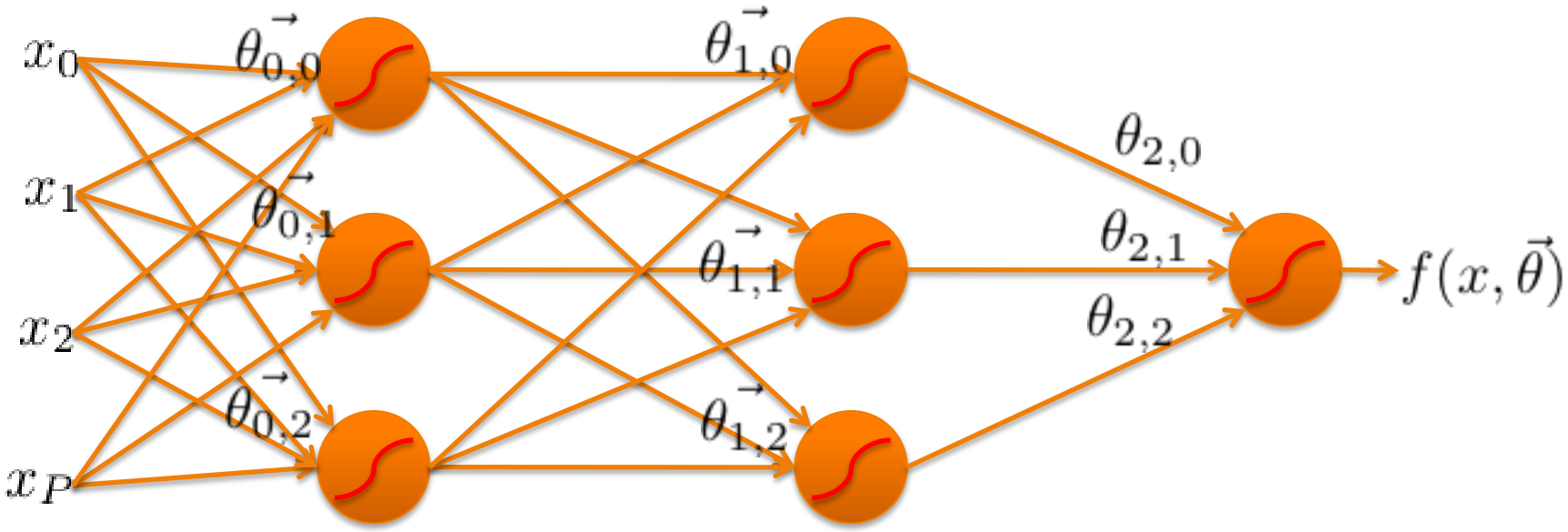
Sigmoid Neuron



Perceptron

Multilayer Networks

- Cascade Neurons together
- The output from one layer is the input to the next
- Each Layer has its own sets of weights

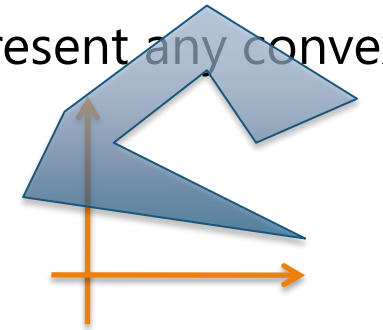


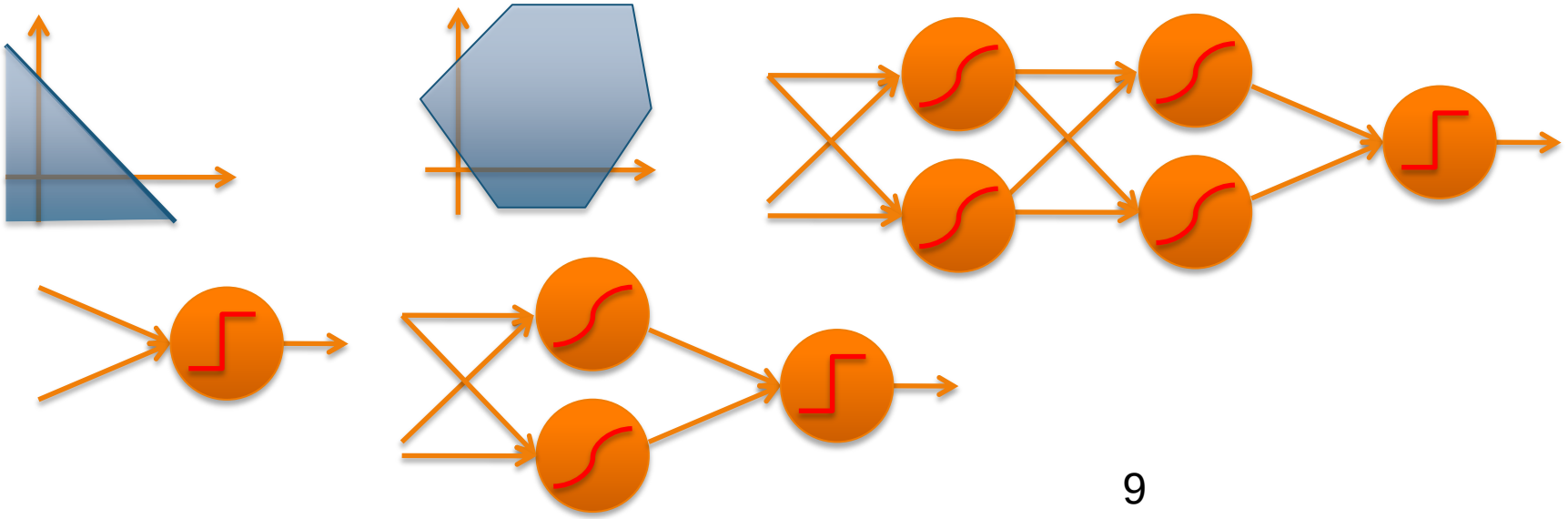
Non-Linearity is Important!

- Cascade of linear networks is a linear classifier
- Cannot find decision boundaries which are of different shapes
- Cascade of nonlinear networks can learn nonlinear boundaries

Linear Separability

- 1-layer cannot handle XOR
- More layers can handle more complicated spaces – but require more parameters
- Each node splits the feature space with a hyperplane
- If the second layer is AND a 2-layer network can represent any convex hull.
- Restatement of theorem from last time





Neural Networks

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

- Today
  - how to train neural networks?

Forward Propagation:
given input x, compute network output

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

1: **for** $i = 1$ **to** *number of hidden units* **do**
2: $\quad h_i \leftarrow \tanh(w_i \cdot \hat{x})$ // compute activation of hidden unit i
3: **end for**
4: **return** $v \cdot h$ // compute output unit

Neural Network Training

Backpropagation algorithm

=

Gradient descent + Chain rule

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

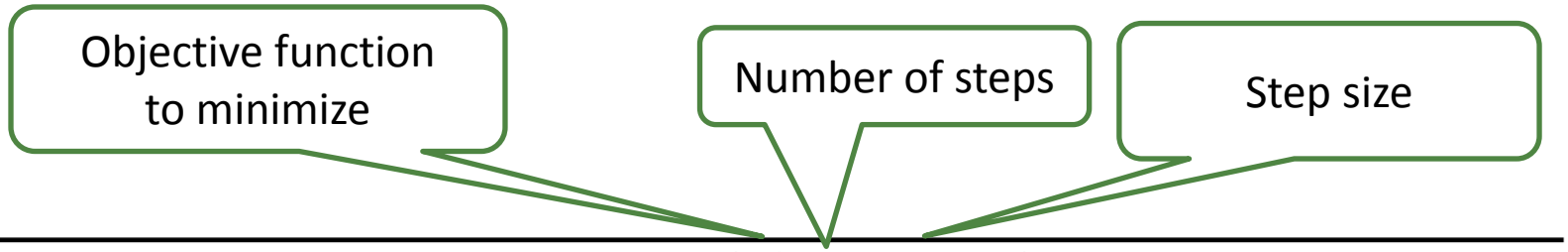
- Other objectives are possible (e.g., other loss functions, add regularizer)

Backprop in a 2-layer network

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

1: $\mathbf{W} \leftarrow D \times K$ matrix of small random values // initialize input layer weights
2: $v \leftarrow K$-vector of small random values // initialize output layer weights
3: **for** *iter* = 1 ... *MaxIter* **do**
4: $\mathbf{G} \leftarrow D \times K$ matrix of zeros // initialize input layer gradient
5: $g \leftarrow K$-vector of zeros // initialize output layer gradient
6:
7:
8:
9:
10:
11: Compute Gradient G and g
12:
13:
14:
15:
16:
17:
18: $\mathbf{W} \leftarrow \mathbf{W} - \eta \mathbf{G}$ // update input layer weights
19: $v \leftarrow v - \eta g$ // update output layer weights
20: **end for**
21: **return** $\mathbf{W}$, v

Recall: gradient descent for linear classifiers



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

1: $z^{(0)} \leftarrow \langle 0, 0, \ldots, 0 \rangle$ // initialize variable we are optimizing
2: **for** $k = 1 \ldots K$ **do**
3: $g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$ // compute gradient at current location
4: $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // 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)

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

$$\nabla_{\boldsymbol{v}} = -\sum_{n} e_n \boldsymbol{h}_n$$

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 $\boldsymbol{w_i}$

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

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

Chain rule

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

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

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

(This is on one example only)

Backprop in a 2-layer network

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

```
1:  W ← D×K matrix of small random values          // initialize input layer weights
2:  v ← K-vector of small random values            // initialize output layer weights
3:  for iter = 1 ... MaxIter do
4:      G ← D×K matrix of zeros                    // initialize input layer gradient
5:      g ← K-vector of zeros                      // initialize output layer gradient
6:      for all (x,y) ∈ D do
7:          for i = 1 to K do
8:              a_i ← w_i · x̂
9:              h_i ← tanh(a_i)                    // compute activation of hidden unit i
10:         end for
11:         ŷ ← v · h                              // compute output unit
12:         e ← y − ŷ                              // compute error
13:         g ← g − eh                             // update gradient for output layer
14:         for i = 1 to K do
15:             G_i ← G_i − ev_i(1 − tanh²(a_i))x  // update gradient for input layer
16:         end for
17:     end for
18:     W ← W − ηG                                 // update input layer weights
19:     v ← v − ηg                                 // update output layer weights
20: end for
21: return W, 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 + non-linearities
- Why are neural networks powerful?
  - Universal function approximators!
- How to train neural networks?
  - The backpropagation algorithm