

Neural Networks

CMSC 422

MARINE CARPUAT

marine@cs.umd.edu

Neural Networks

- Last time
 - What are Neural Networks?
 - 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 + 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 `TWO_LAYER_NETWORK_PREDICT`(\mathbf{W}, v, \hat{x})

```
1: for  $i = 1$  to number of hidden units do
2:    $h_i \leftarrow \tanh(\mathbf{w}_i \cdot \hat{x})$  // 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

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

- i.e. our goal is to find parameters \mathbf{W} , \mathbf{v} that minimize squared error
- Other objectives are possible (e.g., other loss functions, add regularizer)

Backprop in a 2-layer network

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

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

Recall: gradient descent for linear classifiers

Objective function
to minimize

Number of steps

Step size

Algorithm 22 GRADIENTDESCENT($\mathcal{F}, K, \eta_1, \dots$)

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

What's our Training Objective?

- We'll consider the following objective

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

- i.e. our goal is to find parameters \mathbf{W} , \mathbf{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

$$\nabla_v = - \sum_n e_n h_n$$

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

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

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial f_i} \frac{\partial f_i}{\partial w_i}$$

Chain rule

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

$$\frac{\partial f_i}{\partial w_i} = f'(w_i \cdot \mathbf{x}) \mathbf{x}$$

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

(This is on one example only)

Backprop in a 2-layer network

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

```
1:  $\mathbf{W} \leftarrow D \times K$  matrix of small random values // initialize input layer weights
2:  $\mathbf{v} \leftarrow K$ -vector of small random values // initialize output layer weights
3: for  $iter = 1 \dots MaxIter$  do
4:    $\mathbf{G} \leftarrow D \times K$  matrix of zeros // initialize input layer gradient
5:    $\mathbf{g} \leftarrow K$ -vector of zeros // initialize output layer gradient
6:   for all  $(x, y) \in \mathbf{D}$  do
7:     for  $i = 1$  to  $K$  do
8:        $a_i \leftarrow \mathbf{w}_i \cdot \hat{\mathbf{x}}$ 
9:        $h_i \leftarrow \tanh(a_i)$  // compute activation of hidden unit  $i$ 
10:    end for
11:     $\hat{y} \leftarrow \mathbf{v} \cdot \mathbf{h}$  // compute output unit
12:     $e \leftarrow y - \hat{y}$  // compute error
13:     $\mathbf{g} \leftarrow \mathbf{g} + e\mathbf{h}$  // update gradient for output layer
14:    for  $i = 1$  to  $K$  do
15:       $\mathbf{G}_i \leftarrow \mathbf{G}_i + e\mathbf{v}_i(1 - \tanh^2(a_i))\mathbf{x}$  // update gradient for input layer
16:    end for
17:  end for
18:   $\mathbf{W} \leftarrow \mathbf{W} - \eta\mathbf{G}$  // update input layer weights
19:   $\mathbf{v} \leftarrow \mathbf{v} - \eta\mathbf{g}$  // update output layer weights
20: end for
21: return  $\mathbf{W}, \mathbf{v}$ 
```

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

Inconveniences 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