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** TwoLayerNetworkPredict$(W, v, \hat{x})$

1. **for** $i = 1$ **to** number of hidden units **do**
2. \( h_i \leftarrow \tanh(w_i \cdot \hat{x}) \) \hfill // compute activation of hidden unit $i$
3. **end for**
4. **return** $v \cdot h$ \hfill // 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** \texttt{TWOLAYERNETWORKTRAIN}(D, \eta, K, \text{MaxIter})

```plaintext
1: \textbf{W} \leftarrow D \times K \text{ matrix of small random values} \quad \text{// initialize input layer weights}
2: \textbf{v} \leftarrow K\text{-vector of small random values} \quad \text{// initialize output layer weights}
3: \textbf{for} \ iter = 1 \ldots \text{MaxIter} \quad \textbf{do}
4: \quad \textbf{G} \leftarrow D \times K \text{ matrix of zeros} \quad \text{// initialize input layer gradient}
5: \quad \textbf{g} \leftarrow K\text{-vector of zeros} \quad \text{// initialize output layer gradient}
6: \quad \textbf{end for}
7: \textbf{return} \ \textbf{W}, \textbf{v}
```

Compute Gradient G and g
Recall: gradient descent for linear classifiers

**Algorithm 22**  
\texttt{GradientDescent}(\mathcal{F}, K, \eta_1, \ldots)

1. \(z^{(0)} \leftarrow \langle 0, 0, \ldots, 0 \rangle\)  
   // initialize variable we are optimizing
2. \textbf{for} \(k = 1 \ldots K \) \textbf{do}
3. \hspace{1em} \(g^{(k)} \leftarrow \nabla_z \mathcal{F} |_{z^{(k-1)}}\)  
   // compute gradient at current location
4. \hspace{1em} \(z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}\)  
   // take a step down the gradient
5. \textbf{end for}
6. \textbf{return} \(z^{(K)}\)
What’s our Training Objective?

• We’ll consider the following objective

\[
\min_{\mathbf{W}, \mathbf{v}} \sum_n \frac{1}{2} \left( y_n - \sum_i \mathbf{v}_i f(\mathbf{w}_i \cdot x_n) \right)^2
\]

– 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 $n$:
$$y_n - \hat{y}_n$$

Vector of activations of hidden units for example $n$
Gradient of objective w.r.t. hidden unit weights $w_i$

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

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

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

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

$$
\nabla w_i = -ev_i f'(w_i \cdot x)x
$$

(This is on one example only)
Backprop in a 2-layer network

**Algorithm 25** TwoLayerNetworkTrain(D, η, K, MaxIter)

1. \( W \leftarrow D \times K \) 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 \ldots MaxIter do
   4. \( G \leftarrow D \times K \) matrix of zeros // initialize input layer gradient
   5. \( g \leftarrow K \) vector of zeros // initialize output layer gradient
   6. for all \((x, y) \in D\) do
      7. for \(i = 1 \) to \(K\) do
         8. \( a_i \leftarrow w_i \cdot \hat{x} \)
         9. \( h_i \leftarrow \tanh(a_i) \) // compute activation of hidden unit \(i\)
      end for
      10. \( \hat{y} \leftarrow v \cdot h \) // compute output unit
      11. \( e \leftarrow y - \hat{y} \) // compute error
      12. \( g \leftarrow g - eh \) // update gradient for output layer
      for \(i = 1 \) to \(K\) do
         13. \( G_i \leftarrow G_i - ev_i(1 - \tanh^2(a_i))x \) // update gradient for input layer
      end for
   end for
13. \( W \leftarrow W - \eta G \) // update input layer weights
14. \( v \leftarrow v - \eta g \) // update output layer weights
15. return \( W, v \)
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