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($\mathbf{W}$, $\boldsymbol{v}$, $\hat{\boldsymbol{x}}$)

---

for $i = 1$ **to** *number of hidden units* **do**

  $h_i \leftarrow \tanh(\boldsymbol{w}_i \cdot \hat{\boldsymbol{x}})$ // compute activation of hidden unit i

end for

return $\boldsymbol{v} \cdot \boldsymbol{h}$ // compute output unit

---

Neural Network Training

Backpropagation algorithm

=

Gradient descent + Chain rule

What's our Training Objective?

- We'll consider the following objective

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

  - 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, $MaxIter$)

```
W ← D×K matrix of small random values        // initialize input layer weights
v ← K-vector of small random values          // initialize output layer weights
for iter = 1 ... MaxIter do
    G ← D×K matrix of zeros                   // initialize input layer gradient
    g ← K-vector of zeros                     // initialize output layer gradient

        Compute Gradient G and g

    W ← W − ηG                                // update input layer weights
    v ← v − ηg                                // update output layer weights
end for
return W, v
```

Recall: gradient descent for linear classifiers

Objective function to minimize

Number of steps

Step size

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

$z^{(0)} \leftarrow \langle 0, 0, \dots, 0 \rangle$ // initialize variable we are optimizing

for $k = 1 \dots K$ **do**

  $g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$ // compute gradient at current location

  $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // take a step down the gradient

end for

return $z^{(K)}$

What's our Training Objective?

- We'll consider the following objective

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

  – 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 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, $MaxIter$)

$\mathbf{W} \leftarrow D{\times}K$ matrix of small random values // initialize input layer weights
$\boldsymbol{v} \leftarrow K$-vector of small random values // initialize output layer weights
for $iter = 1 \ldots MaxIter$ **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**
      $\mathbf{G}_i \leftarrow \mathbf{G}_i - ev_i(1 - \tanh^2(a_i))\boldsymbol{x}$ // 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 + non-linearities
- Why are neural networks powerful?
  - Universal function approximators!
- How to train neural networks?
  - The backpropagation algorithm