Outline for April 5

• Neural network architectures
• Choice of loss function
• Choice of non-linearity (activation function)
• Choice of weight initialization

• Lab 6 due TONIGHT
• Lab 7 released TODAY (last chance for partner forms!)
• Office hours TODAY 12:30-2:30pm
Outline for April 5

• Neural network architectures
  • Choice of loss function
  • Choice of non-linearity (activation function)
  • Choice of weight initialization
Neural Network

Architecture

3 layer, fully connected

P features

bias

Exercise

# params to optimize?
P_1, P_2, P_3

apply loss function

multi-class: P_2 x C

(prediction)
Exercise: 
optimize? 
apply loss function 
activation function 
how to find weights? 
back propagation!

\[ H^{(0)} = a \left( X W^{(0)} + b^{(0)} \right) \]

Idea: learn a function from input \((\vec{x})\) to output \((y)\) 

\[ H^{(2)} = a \left( H^{(1)} W^{(2)} + b^{(2)} \right) \]

\[ H^{(3)} = a \left( H^{(2)} W^{(3)} + b^{(3)} \right) \]

\[ \text{last layer: } 1 \times 1 \]

\[ \text{broadcast into } n \times p_2 \]

\[ \text{[aug]} b, b_2, b_3 \]

\[ \Rightarrow \min \text{ loss!} \]
Outline for April 5

• Neural network architectures
• Choice of loss function
• Choice of non-linearity (activation function)
• Choice of weight initialization
Loss Functions

for classification: Cross entropy

\[ H(p, q) = -\sum_{x \in X} p(x) \log q(x) \]

\( x = \{0, 1, 3\} \)

2 classes \( \{0, 1\} \)

\[ H(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log (1 - \hat{y}) \]

same as logistic regression

\( \hat{y} = \text{prob(output} = 1) \)
2 classes $\beta_0, \beta_1$

$$H(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

same as logistic regression loss!

$\hat{y} = \text{prob(output = 1)}$

$\hat{y}$ true

$\hat{y}$ pred

1 2 3 4 5
Multiclass

$C$ classes

$H(y, \hat{y}) = -\sum_{k=1}^{C} y_k \log \hat{y}_k$

make $y = [0, 0, 0, 1, 0]$

1-hot

true class is 4

$\sum_{k=1}^{C} \hat{y}_k = 1$

prob dist
Outline for April 5

- Neural network architectures
- Choice of loss function
- Choice of non-linearity (activation function)
- Choice of weight initialization
Option 1: sigmoid function

- Input: all real numbers, output: $[0, 1]$

\[
\sigma(x) = \frac{1}{1 + e^{-x}}
\]

- Derivative is convenient

\[
\sigma'(x) = \sigma(x)(1 - \sigma(x))
\]
Option 2: hyperbolic tangent

• Input: all real numbers, output: [-1, 1]

\[ \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \]
Option 3: Rectified Linear Unit (ReLU)

- Return $x$ if $x$ is positive (i.e. threshold at 0)

\[ f(x) = \max(0, x) \]
Discussion

• In light of backpropagation, what are the pros and cons of each activation function?
  
  – Sigmoid
  – Hyperbolic tangent
  – ReLU
Pros and Cons of Activation Functions

1) Sigmoid

- (-) When input becomes very positive or very negative, gradient approaches 0 (saturates and stops gradient descent)
- (-) Not zero-centered, so gradient on weights can end up all positive or all negative (zig-zag in gradient descent)
- (+) Derivative is easy to compute given function value!
Pros and Cons of Activation Functions

1) Sigmoid
   - (-) When input becomes very positive or very negative, gradient approaches 0 (saturates and stops gradient descent)
   - (-) Not zero-centered, so gradient on weights can end up all positive or all negative (zig-zag in gradient descent)
   - (+) Derivative is easy to compute given function value!

2) Tanh
   - (-) Still has a tendency to prematurely kill the gradient
   - (+) Zero-centered so we get a range of gradients
   - (+) Rescaling of sigmoid function so derivative is also not too difficult

Pros and Cons of Activation Functions

1) Sigmoid
   • (-) When input becomes very positive or very negative, gradient approaches 0 (saturates and stops gradient descent)
   • (-) Not zero-centered, so gradient on weights can end up all positive or all negative (zig-zag in gradient descent)
   • (+) Derivative is easy to compute given function value!

2) Tanh
   • (-) Still has a tendency to prematurely kill the gradient
   • (+) Zero-centered so we get a range of gradients
   • (+) Rescaling of sigmoid function so derivative is also not too difficult

3) ReLU
   • (+) Works well in practice (accelerates convergence)
   • (+) Function value very easy to compute! (no exponentials)
   • (-) Units can “die” (no signal) if input becomes too negative throughout gradient descent

Outline for April 5

- Neural network architectures
- Choice of loss function
- Choice of non-linearity (activation function)
- Choice of weight initialization
Weight initialization

• All 0’s initialization is bad! Causes nodes to compute the same outputs, so then the weights go through the same updates during gradient descent

• Need asymmetry! => usually use small random values
Weight initialization

- **Issue**: nodes with more randomly initialized inputs will have a higher variance in their output

- **Solution**: divide by the sqrt($n$) where $n$ is the “fan-in” (number of inputs)

Lab 7 getting started

• It is helpful to have our data be zero-centered, so we will subtract off the mean.

• It is also helpful to have the features be on the same scale, so we will divide by the standard deviation.

• We will compute the mean and std with respect to the *training data*, then apply the same transformation to all datasets.
Lab 7 getting started

• So far in this class, we have considered *stochastic gradient descent*, where one data point is used to compute the gradient and update the weights

• On the flipside is *batch gradient descent*, where we compute the gradient with respect to all the data, and then update the weights

• A middle ground uses *mini-batches* of examples before updating the weights. This is the approach we will use in Lab 7.
More hidden units can contribute to overfitting


Larger Neural Networks can represent more complicated functions. The data are shown as circles colored by their class, and the decision regions by a trained neural network are shown underneath. You can play with these examples in this ConvNetsJS demo.
However! It is always better to use a larger network and regularize in other ways.

The effects of regularization strength: Each neural network above has 20 hidden neurons, but changing the regularization strength makes its final decision regions smoother with a higher regularization. You can play with these examples in this ConvNetsJS demo.