Today

- Supervised Learning
  - Linear regression (18.6)
  - Neural Networks (18.7)
- Making complex decision (17.1, 17.2, 17.3)
- Background for Reinforcement Learning

Univariate linear regression

- Regression on a single variable
  - Input $x$, and output $y$; learn weights $w$
  - $y = w_0 + w_1x$
- Hypothesis becomes
  - $h_w(x) = w_0 + w_1x$
- Finding best weights is linear regression

Example

![Example graph showing house price vs. size in square feet]
Learning

• Find the weights which minimize the loss
  • How do we define loss?
  • $L_2$ is the squared distance (to the line)
    • Also called the $L_2$ norm
      \[ Loss(h_w) = \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) = \sum_{j=1}^{N} (y_j - h_w(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_0 + w_1 x_j))^2 \]

Graph of loss function

Learning

• In univariate case, can solve exactly
  • In general, use gradient descent to improve values
• Simple algorithm:
  • Initialize $w$ randomly
  • Loop until converged:
    • For each $w_i$ in $w$
      * $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(w)$

Gradient

\[ Loss(h_w) = \sum_{j=1}^{N} (y_j - (w_0 + w_1 x_j))^2 \]

• What is the gradient/slope with respect to $w_0$?
  • Just use a single training example, not all $N$
  • $\frac{\partial}{\partial w} (y - h(x))^2 = 2(y - h(x)) \cdot \frac{\partial}{\partial w} (y - h(x))$
  • $\frac{\partial}{\partial w_0} (y - h(x)) = \frac{\partial}{\partial w_0} (y - (w_0 + w_1 x)) = -1$
    * $\frac{\partial}{\partial w} (y - h(x))^2 = -2 \cdot (y - h(x))$
  • $\frac{\partial}{\partial w_1} (y - h(x)) = \frac{\partial}{\partial w_1} (y - (w_0 + w_1 x)) = -x$
    * $\frac{\partial}{\partial w} (y - h(x))^2 = -2 \cdot (y - h(x)) \cdot x$
Final algorithm

- Weight update was:
  - \( w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \text{Loss}(w) \)
  - \( w_0 \leftarrow w_0 + \alpha (y - h_w(x)) \)
  - \( w_1 \leftarrow w_1 + \alpha (y - h_w(x)) \cdot x \)
- For more variables:
  - \( w_i \leftarrow w_i + \alpha x_i (y - h(x)) \)
- Incremental update version
- Batch update version in book

Function vs classifier

- So far we’ve learned a function approximator
  - Can also be used to learn a classifier
  - For instance, classify as true if \( h(x) \geq 0 \)
    - Or \( h_w(x) = 1 \) if \( w \cdot x \geq 0 \)
    - Or \( h_w(x) = \text{Threshold}(w \cdot x) \)
- Can visualize performance with a training curve
Threshold function

- What if we use a different threshold function
- Get a different type of regression
- $Logistic(z) = \frac{1}{1 + e^{-z}}$
- $h_{w}(x) = Logistic(w \cdot x)$
- Derivative of logistic:
  - $g'(w \cdot x) = g(w \cdot x) (1 - g(w \cdot x))$
  - $w_i \leftarrow w_i + \alpha (y - h(x)) \cdot (h(x)(1-h(x))x_i$

Notes

- Linear regression is limited to linearly separable classes
  - BUT, can use more complex inputs to make the classes linearly separable [kernel]
- Regression can easily learn majority function
  - Recall, this was hard for decision trees

Extending to multiple layers

- What we have so far is called a perceptron
  - Single-layer neural network
- Can extend to multiple layers
  - Usually assume that all outputs from previous layer fully connected to next layer

\[
\alpha = \frac{1000}{1000 + t}
\]
Neural Network training

- Use same rule as before to train the output layer
  - Let $\text{Err}_k$ be the error in the $k$th output
    - Or assume a single output unit and ignore $k$
  - Let $\Delta_k = \text{Err}_k \cdot g'(in_k)$
  - Update rule becomes:
    * $w_{j,k} \leftarrow w_{j,k} + \alpha \cdot a_j \cdot \Delta_k$

Neural Network training

- Input rule more tricky
  - Where do the errors come from?
    - Let $\Delta_i = g'(in_i) \sum_k w_{i,k} \Delta_k$
    * $w_{i,j} \leftarrow w_{i,j} + \alpha \cdot a_i \cdot \Delta_j$
  - Full derivation in book