CS 342

- Homework 1
  - Due 9/6, can still turn it in before 9/8! (with penalties)
  - Bonus point
- Homework 2
  - Posted, due next Thursday
  - Two parts
    - SuperTux trace collection
    - Train classifiers
- Quiz 1
  - Grade posted

Agenda

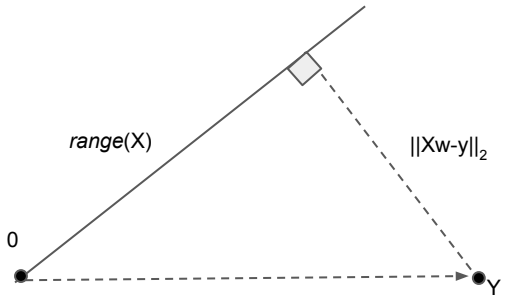
1. Linear Regression
2. Logistic Regression
3. Multi-layer Perceptron
4. PyTorch code walk through

Linear Regression

- Problem: Given $\{x_i\}$ and $\{y_i\}$, where $x_i \in R^d$ can be used to predict $y \in R$, find $\hat{y}=h(x)$ such that $\hat{y} \approx y$.
- Linear Regression
  - Model assumes $h(x) = w^T x+b$
  - For simplicity, we write $h(x)=w^T x$ (add 1 to the entry of each x_i)
  - Find model parameter w such that squared error is minimized: $\min_w ||Xw-y||_2$

Linear Regression

- Interpretation
  - $\min_w ||Xw-y||_2 = \min_z ||z-y||_2$, where $z \in$ *range*(X). It is minimized when z-y $\perp$ z. We are finding the projection of Y onto the subspace spanned by X.



Linear Regression

- Solution
  - Since $X^T(y-Xw^*)=0$, i.e. we get $w^*=(X^TX)^{-1}X^Ty$

Linear Regression

- Linear Regression in PyTorch
  - Model:
    - __init__()
      - self.linear = nn.Linear(input_dim, output_dim)
    - forward()
      - return self.linear(x)
  - Training:
    - criterion = nn.MSELoss()
    - loss = criterion(outputs, targets)
  - Note:
    - Define model parameters in __init__(), write logic in forward()

Linear Regression

- Linear Regression in PyTorch
  - Testing:
    - outputs = m(inputs)

Logistic Regression

- Problem: Given $\{x_i\}$ and $\{y_i\}$, where $x_i \in R^d$ can be used to predict $y \in \{0,1\}$, find $\hat{y}=h(x)$ such that $\hat{y} \approx y$.
- Logistic Regression
  - Model assumes log odds (log p/(1-p)) is affine of x, i.e. $\log P(\hat{y}_i=1|x_i)/(1-P(\hat{y}_i=1|x_i))=w^T x_i$
    - Rearrange, we get $P(\hat{y}_i=1|x_i)=1/(1+e^{w\hat{}Tx})$

Logistic Regression

- Logistic Regression in PyTorch
  - Model:
    - self.linear = nn.Linear(input_dim, output_dim)
    - return self.linear(x)
  - Training:
    - criterion = nn.CrossEntropyLoss()
    - loss = criterion(outputs, targets)
    - # outputs, targets are nxk tensors
  - Note:
    - Exactly same as linear regression, except with a different loss functions

Logistic Regression

- Logistic Regression in PyTorch
  - Testing
    - outputs = m(inputs)
    - predictions = argmax(outputs, dim=1)
    - softmax = nn.Softmax(dim=1)(outputs)

Multi-layer Perceptron

- MLP
  - Models
    - Stacked linear layers with non-linear activation functions in between
    - Refer to lecture slides for details
  - Training
    - Train with GD/SGD because linear and the activation functions are differentiable
    - However, loss is non-convex

Multi-layer Perceptron

- MLP in PyTorch
  - Model
    - __init__()
      - self.fc1 = nn.Linear(inputs_dim, h1_dim)
      - self.fc2 = nn.Linear(h1_dim, outputs_dim)
      - self.relu1 = nn.ReLU()
    - forward()
      - x = self.fc1(x)
      - x = self.relu(x)
      - x = self.fc2(x)
      - return x

Multi-layer Perceptron

- MLP in PyTorch
  - Training
    - criterion = nn.CrossEntropyLoss() # Take classification as our example
    - loss = criterion(outputs, targets)
  - Testing
    - outputs = m(inputs)
    - predictions = argmax(outputs, dim=1)
    - softmax = nn.Softmax(dim=1)(outputs)

PyTorch Tutorial

- Link: philkr.net/cs342/section/section2.ipynb