STATS / DATA SCI 315

Regression
Loss functions and gradient descent

Training Dataset

- Need a dataset where we know the sale price, area, and age for each home
- This is called a *training dataset* or *training set*
- Put one sale info on each row
- Each row is called an *example* (or *data point*, *data instance*, *sample*)
- Each example has
    - A label (price)
    - Features (area, age)

Choosing weights and bias based on training data

- Choose the weights and the bias such that our model predictions best fit the true prices observed in the data
- Long form of our linear model:

$$\text{price} = w_{\text{area}} \cdot \text{area} + w_{\text{age}} \cdot \text{age} + b$$

- If we had *d* features instead of just two:

$$\hat{y} = w_1 x_1 + \ldots + w_d x_d + b$$

- The “hat” on top of y denotes that it is an estimate

More compact notation

- Collect all features into a vector $\mathbf{x} \in \mathbb{R}^d$ and all weights into a vector $\mathbf{w} \in \mathbb{R}^d$
- Use dot product to express model compactly:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b$$

- Entire dataset of n examples is referred to as the *design matrix* $\mathbf{X} \in \mathbb{R}^{n \times d}$
- $\mathbf{X}$ contains one row for every example and one column for every feature
- Prediction vector $\hat{\mathbf{y}} \in \mathbb{R}^n$ can be expressed via the matrix-vector product:

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

Loss functions and gradient descent

Loss function

- We need a way to measure how good a prediction $\hat{y}^{(i)}$ is when the true label is $y^{(i)}$
- Most popular regression loss is the *squared error*:

  $l^{(i)}(\mathbf{w},b) = ½ (\hat{y}(i)−y(i))^2$
- The ½ above is just for convenience
- Quality of model on entire dataset is assessed by:

$$L(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^{n} l^{(i)}(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right)^2.$$

Visualizing the fit with 1-dimensional x

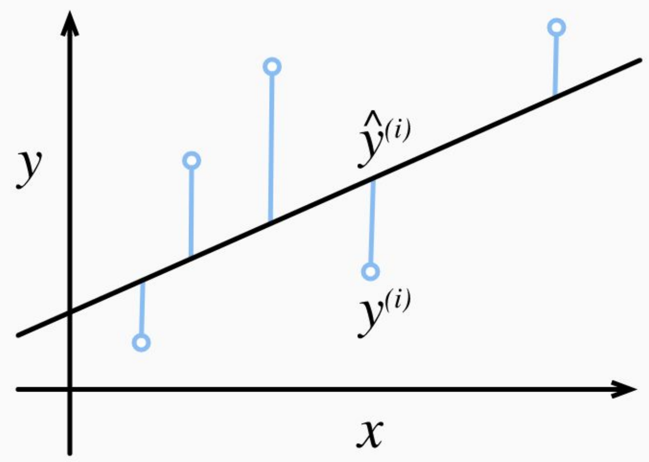

Fig. 3.1.1 Fit data with a linear model.

Minimizing the loss

- Bias can be absorbed into weights by using a “dummy” feature which is always 1:

  $$\mathbf{w}^\top \mathbf{x} + b = (\mathbf{w}, b)^\top (\mathbf{x}, 1)$$

- Best choice for $\mathbf{w}$ is given by:

  $$\mathbf{w}^* = \mathrm{argmin}_{\mathbf{w}}\, L(\mathbf{w}) = \mathrm{argmin}_{\mathbf{w}}\, \tfrac{1}{2} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

- Note that, in the last step, we removed *n* and used the Euclidean norm
- This *optimization problem* turns out to have a closed form solution

Taking derivatives

- How do we take the derivative of $L(\mathbf{w}) = ½ \|\mathbf{y}-\mathbf{X}\mathbf{w}\|^2$ w.r.t. $\mathbf{w}$?
- Expanding gives us $L(\mathbf{w}) = ½\, \mathbf{y}^T\mathbf{y} - \mathbf{w}^T\mathbf{X}^T\mathbf{y} + ½\, \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}$
- The first term is independent of $\mathbf{w}$
- For second term we use: if $F(\mathbf{w}) = \mathbf{w}^T\mathbf{x}$ then $\nabla F(\mathbf{w}) = \mathbf{x}$
- For the third term we use: if $H(\mathbf{w}) = ½\, \mathbf{w}^T\mathbf{A}\, \mathbf{w}$ then $\nabla H(\mathbf{w}) = ½\, (\mathbf{A} + \mathbf{A}^T)\mathbf{w}$
  - If $\mathbf{A}$ is symmetric, i.e., $\mathbf{A} = \mathbf{A}^T$, this simplifies to $\mathbf{A}\,\mathbf{w}$
- Therefore, the derivative w.r.t. $\mathbf{w}$ is:

$$-\mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\mathbf{w}$$

Set gradient to zero

- Note that the derivative w.r.t. **w** has *d* components
- It is often called a *gradient*

$$\nabla L(\mathbf{w}) = -\mathbf{X}^{\top}\mathbf{y} + \mathbf{X}^{\top}\mathbf{X}\mathbf{w}$$

- Setting it to zero gives the closed-form solution

$$\mathbf{w}^* = (\mathbf{X}^{\top}\mathbf{X})^{-1}\mathbf{X}^{\top}\mathbf{y}$$

- We're assuming here that $\mathbf{X}^{\top}\mathbf{X}$ is invertible

Closed-form solutions are rare

- We got lucky in our simple setting (linear regression with squared loss)
- Usually we're not so lucky
- Even simple changes can destroy this property (like using absolute error)
- If we insisted on closed form solutions, almost *all* of DL will be excluded
- Key technique for incrementally lowering the loss function: *gradient descent*
- Iteratively reduces the loss by updating the parameters in the direction of the *negative gradient*

Gradient Descent

- For any objective function $J(\mathbf{w})$, GD update takes the form:
  $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J(\mathbf{w})$
- The gradient gives you the direction of fastest local *increase* in J
- Since we're looking to minimize J, we move in the direction of *negative* gradient
- The step size (aka learning rate) η controls how much we move

Gradient Descent on the Loss

- Takes the form:
  $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L(\mathbf{w})$
- Here, the gradient has the form
  $\nabla L(\mathbf{w}) = 1/n \sum_i ½ \nabla (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2 = 1/n \sum_i \mathbf{x}^{(i)} (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})$
- So GD update becomes
  $\mathbf{w} \leftarrow \mathbf{w} - \eta/n \sum_i \mathbf{x}^{(i)} (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})$
- Requires one full pass through the entire data set

Minibatch Stochastic Gradient Descent

- In each iteration, we first randomly sample a minibatch **B** consisting of a fixed number of training examples
- Then we update
  $\mathbf{w} \leftarrow \mathbf{w} - \eta/|\mathbf{B}| \sum_{i \in \mathbf{B}} \mathbf{x}^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})$
- Note that the set **B** is random and changes from iteration to iteration
- η and |**B**| (the batch size) are *hyperparameters:* they're kept fixed during training
- However, an outer loop might optimize them by tracking performance on a *validation set*

Updates with bias kept separately

- Update **w**:
  $\mathbf{w} \leftarrow \mathbf{w} - \eta/|\mathbf{B}| \sum_{i \in \mathbf{B}} \mathbf{x}^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)})$
- Update bias:
  $b \leftarrow b - \eta/|\mathbf{B}| \sum_{i \in \mathbf{B}} (\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)})$

Making predictions with learned model

- Train for some predetermined number of iterations (or until some other stopping criteria are met)
- Record the estimated model parameters, denoted $\hat{\mathbf{w}}, \hat{b}$
- Given a new house with area x_1 and age x_2, we predict its price as
  $\hat{\mathbf{w}}^\top \mathbf{x} + \hat{b}$ where $\mathbf{x} = (x_1, x_2)$
- Estimating targets given features is commonly called *prediction*
- It's also (misleadingly) called *inference* in deep learning jargon