Gradient Descent

February 15, 2019

Logistic Regression

- For each feature i:
  - Value x_i
  - Weight w_i
- Bias term b

Logistic Regression

- For each feature i:
  - Value x_i
  - Weight w_i
- Bias term b

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b = \mathbf{w} \cdot \mathbf{x} + b$$

Logistic Regression

- Logistic function $\sigma(z) = \dfrac{1}{1 + e^{-z}}$

Logistic Regression

- Logistic function $\sigma(z) = \dfrac{1}{1 + e^{-z}}$
- $p(y = 1|\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \dfrac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$
- $p(y = 0|\mathbf{x}) = 1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \dfrac{e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$

Cross-entropy Loss

- Let $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$ be the classifier output for some $\mathbf{x}$
- What is the probability that the classifier is correct?

Cross-entropy Loss

- Let $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$ be the classifier output for some $\mathbf{x}$
- What is the probability that the classifier is correct?
  - If $y = 1$, then $p(y = 1|\mathbf{x}) = \hat{y}$
  - If $y = 0$, then $p(y = 0|\mathbf{x}) = 1 - \hat{y}$

Cross-entropy Loss

- In general, $p(y|\mathbf{x}) = \hat{y}^y(1-\hat{y})^{1-y}$

Cross-entropy Loss

- In general, $p(y|\mathbf{x}) = \hat{y}^y(1-\hat{y})^{1-y}$
- Take the log of both sides:
  $\log p(y|\mathbf{x}) = y \log \hat{y} + (1-y)\log(1-\hat{y})$

Cross-entropy Loss

- In general, $p(y|\mathbf{x}) = \hat{y}^y(1-\hat{y})^{1-y}$
- Take the log of both sides:
  $\log p(y|\mathbf{x}) = y \log \hat{y} + (1-y)\log(1-\hat{y})$
- Turn this into a loss function:
  $L(\hat{y}, y) = -\log p(y|\mathbf{x}) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$

Cross-entropy Loss

- In general, $p(y|\mathbf{x}) = \hat{y}^y(1-\hat{y})^{1-y}$
- Take the log of both sides:
  $\log p(y|\mathbf{x}) = y \log \hat{y} + (1-y)\log(1-\hat{y})$
- Turn this into a loss function:
  $L(\hat{y}, y) = -\log p(y|\mathbf{x}) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$
- Why is this called the cross-entropy loss?

Cross-entropy Loss

- Minimize average loss for each example j:

$$Cost(\mathbf{w}, b) = \frac{1}{m} \sum_{j=1}^{m} L(\hat{y}^{(j)}, y^{(j)})$$

Cross-entropy Loss

- Minimize average loss for each example j:

$$Cost(\mathbf{w}, b) = \frac{1}{m} \sum_{j=1}^{m} L(\hat{y}^{(j)}, y^{(j)})$$

- How?

Gradient Descent

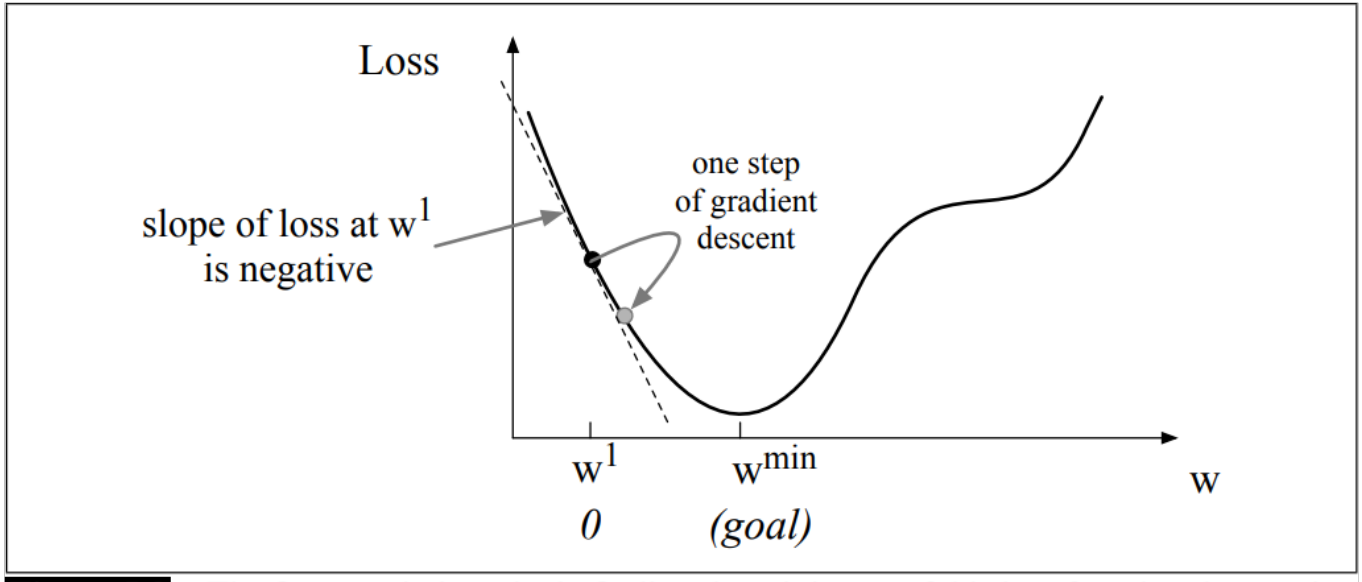

Figure 5.3 The first step in iteratively finding the minimum of this loss function, by moving w in the reverse direction from the slope of the function. Since the slope is negative, we need to move w in a positive direction, to the right. Here superscripts are used for learning steps, so w^1 means the initial value of w (which is 0), w^2 at the second step, and so on.

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ randomly

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ randomly
- At each time step t:
  - Compute gradient ∇L

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ randomly
- At each time step t:
  - Compute gradient ∇L
  - $\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_n} \\ \frac{\partial L}{\partial b} \end{bmatrix}$
  - $\approx$ slope of loss function L

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ randomly
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ randomly
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient

- $\theta_{t+1} = \theta_t - \eta \nabla L$
  - $\eta =$ learning rate

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ randomly
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient

- $\theta_{t+1} = \theta_t - \eta \nabla L$
  - $\eta =$ learning rate
    - Trade-off between speed of convergence and "zig-zag" behavior
    - Often a function of t

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ randomly
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient

- $\theta_{t+1} = \theta_t - \eta \nabla L$
  - $\eta =$ learning rate
    - Trade-off between speed of convergence and "zig-zag" behavior
    - Often a function of t
- Because L is convex, we eventually reach a global minimum

Gradient Descent

- $L(\mathbf{w}, b) = -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1 - y) \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b))]$

Gradient Descent

- $L(\mathbf{w}, b) = -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1 - y) \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b))]$
- ...
- (calculus)
- ...

Gradient Descent

- $L(\mathbf{w}, b) = -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1 - y) \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b))]$
- ...
- (calculus)
- ...
- $\frac{\partial L}{\partial w_i} = [\sigma(\mathbf{w} \cdot \mathbf{x} + b) - y]x_i$
- $\frac{\partial L}{\partial b} = \sigma(\mathbf{w} \cdot \mathbf{x} + b) - y$

Stochastic Gradient Descent

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

where: L is the loss function
f is a function parameterized by θ
x is the set of training inputs $x^{(1)}$, $x^{(2)}$,..., $x^{(n)}$
y is the set of training outputs (labels) $y^{(1)}$, $y^{(2)}$,..., $y^{(n)}$

$\theta \leftarrow 0$
repeat T times
  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
  Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?
  Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?
  $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss ?
  $\theta \leftarrow \theta - \eta\, g$ # go the other way instead
return θ

Figure 5.5 The stochastic gradient descent algorithm

Minibatch Gradient Descent

- Stochastic gradient descent can result in very choppy movements

Minibatch Gradient Descent

- Stochastic gradient descent can result in very choppy movements

$$\frac{\partial Cost}{\partial w_i} = \frac{1}{m}\sum_{j=1}^{m}[\sigma(\mathbf{w}\cdot\mathbf{x}^{(j)} + b) - y^{(j)}]x_i^{(j)}$$

Example

- Python time!

Regularization

- Problem of overfitting

Regularization

- Problem of overfitting
  - Models can fit the training data too well
    - Accidental correlations get high weights
    - Poor generalization performance

Regularization

$$Cost(\mathbf{w}, b) = \left(\frac{1}{m} \sum_{j=1}^{m} L(\hat{y}^{(j)}, y^{(j)}) \right) + \alpha R(\mathbf{w})$$

Regularization

$$Cost(\mathbf{w}, b) = \left(\frac{1}{m} \sum_{j=1}^{m} L(\hat{y}^{(j)}, y^{(j)}) \right) + \alpha R(\mathbf{w})$$

- $R(\mathbf{w}) =$ regularization term
- $\alpha =$ amount of regularization

L1 Regularization

$$R(\mathbf{w}) = ||\mathbf{w}||_1 = \sum_{i=1}^{n} |w_i|$$

L1 Regularization

$$R(\mathbf{w}) = ||\mathbf{w}||_1 = \sum_{i=1}^{n} |w_i|$$

- $=$ sum of absolute values of weights
- Manhattan distance
- Lasso regression
- Some large weights, many zero weights

L2 Regularization

$$R(\mathbf{w}) = ||\mathbf{w}||_2^2 = \sum_{i=1}^{n} w_i^2$$

L2 Regularization

$$R(\mathbf{w}) = ||\mathbf{w}||_2^2 = \sum_{i=1}^{n} w_i^2$$

- $=$ sum of squares of weights
- Euclidean distance
- Ridge regression
- Many small weights

Multinomial Logistic Regression

- (aka maximum entropy classifier)
- Logistic regression with more than two classes

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{i=1}^{n} w_{i,c} x_i \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{i=1}^{n} w_{i,c} x_i \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

- Softmax function $\sigma(z_c) = \dfrac{e^{z_c}}{\sum_{k \in C} e^{z_k}} = p(y = c|\mathbf{x})$