Linear Classifiers (Part 3)

CS114B Lab 4

Kenneth Lai

February 17, 2022

Multi-Class Classification

- Two-class (binary) classification
  - Compute "score" $z = \theta \cdot \mathbf{x}$ (or $\mathbf{w} \cdot \mathbf{x} + b$)
  - Compute decision $\hat{y}$ as a function of z

Multi-Class Classification

- Two-class (binary) classification
  - Compute "score" $z = \theta \cdot \mathbf{x}$ (or $\mathbf{w} \cdot \mathbf{x} + b$)
  - Compute decision $\hat{y}$ as a function of z
    - If $\hat{y}$ is interpreted as the probability of (or indicator for) one class, $1 - \hat{y}$ is the probability of (indicator for) the other class

Multi-Class Classification

- Two-class (binary) classification
  - Compute "score" $z = \theta \cdot \mathbf{x}$ (or $\mathbf{w} \cdot \mathbf{x} + b$)
  - Compute decision $\hat{y}$ as a function of z
    - If $\hat{y}$ is interpreted as the probability of (or indicator for) one class, $1 - \hat{y}$ is the probability of (indicator for) the other class
- Multi-class (multinomial) classification
  - Compute a vector of scores $\mathbf{z} = \Theta \cdot \mathbf{x}$ (or $\mathbf{W} \cdot \mathbf{x} + \mathbf{b}$)
  - Compute decision $\hat{y}$ as a function of $\mathbf{z}$

Linear Maps and Matrices

- Given a feature vector $\mathbf{x}$, we want a score vector $\mathbf{z}$

Linear Maps and Matrices

- Given a feature vector $\mathbf{x}$, we want a score vector $\mathbf{z}$
- More generally, we want to define a linear map between the feature vector space and the score vector space

Linear Maps and Matrices

- Given a feature vector $\mathbf{x}$, we want a score vector $\mathbf{z}$
- More generally, we want to define a linear map between the feature vector space and the score vector space
- A matrix is a rectangular array of scalars, that can be used to define a linear map

Linear Maps and Matrices

- Warning! A note on notation:

Linear Maps and Matrices

- Warning! A note on notation:
  - Math convention: A p-by-n matrix defines a linear map from $\mathbb{R}^n$ to $\mathbb{R}^p$

Linear Maps and Matrices

- Warning! A note on notation:
  - Math convention: A p-by-n matrix defines a linear map from $\mathbb{R}^n$ to $\mathbb{R}^p$
    - Matrices have shapes (output dimension, input dimension)
    - Let $\Theta \in \mathbb{R}^{p \times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{z} \in \mathbb{R}^p$
    - $\mathbf{z} = \Theta \cdot \mathbf{x}$ (or $\mathbf{W} \cdot \mathbf{x} + \mathbf{b}$)

Linear Maps and Matrices

- Warning! A note on notation:
  - Math convention: A p-by-n matrix defines a linear map from $\mathbb{R}^n$ to $\mathbb{R}^p$
    - Matrices have shapes (output dimension, input dimension)
    - Let $\Theta \in \mathbb{R}^{p \times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{z} \in \mathbb{R}^p$
    - $\mathbf{z} = \Theta \cdot \mathbf{x}$ (or $\mathbf{W} \cdot \mathbf{x} + \mathbf{b}$)
  - Computer science convention (mostly): An n-by-p matrix defines a linear map from $\mathbb{R}^n$ to $\mathbb{R}^p$ (technically $\mathbb{R}^{1 \times n}$ to $\mathbb{R}^{1 \times p}$)

Linear Maps and Matrices

- Warning! A note on notation:
  - Math convention: A p-by-n matrix defines a linear map from $\mathbb{R}^n$ to $\mathbb{R}^p$
    - Matrices have shapes (output dimension, input dimension)
    - Let $\Theta \in \mathbb{R}^{p \times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{z} \in \mathbb{R}^p$
    - $\mathbf{z} = \Theta \cdot \mathbf{x}$ (or $\mathbf{W} \cdot \mathbf{x} + \mathbf{b}$)
  - Computer science convention (mostly): An n-by-p matrix defines a linear map from $\mathbb{R}^n$ to $\mathbb{R}^p$ (technically $\mathbb{R}^{1 \times n}$ to $\mathbb{R}^{1 \times p}$)
    - Matrices have shapes (input dimension, output dimension)
    - Let $\Theta \in \mathbb{R}^{n \times p}$, $\mathbf{x} \in \mathbb{R}^{1 \times n}$, and $\mathbf{z} \in \mathbb{R}^{1 \times p}$
    - $\mathbf{z} = \mathbf{x} \cdot \Theta$ (or $\mathbf{x} \cdot \mathbf{W} + \mathbf{b}$)

Linear Maps and Matrices

- Warning! A note on notation:
  - Math convention: A p-by-n matrix defines a linear map from $\mathbb{R}^n$ to $\mathbb{R}^p$
    - Matrices have shapes (output dimension, input dimension)
    - Let $\Theta \in \mathbb{R}^{p \times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{z} \in \mathbb{R}^p$
    - $\mathbf{z} = \Theta \cdot \mathbf{x}$ (or $\mathbf{W} \cdot \mathbf{x} + \mathbf{b}$)
  - Computer science convention (mostly): An n-by-p matrix defines a linear map from $\mathbb{R}^n$ to $\mathbb{R}^p$ (technically $\mathbb{R}^{1 \times n}$ to $\mathbb{R}^{1 \times p}$)
    - Matrices have shapes (input dimension, output dimension)
    - Let $\Theta \in \mathbb{R}^{n \times p}$, $\mathbf{x} \in \mathbb{R}^{1 \times n}$, and $\mathbf{z} \in \mathbb{R}^{1 \times p}$
    - $\mathbf{z} = \mathbf{x} \cdot \Theta$ (or $\mathbf{x} \cdot \mathbf{W} + \mathbf{b}$)
    - More intuitive (input $\rightarrow$ output)

Linear Maps and Matrices

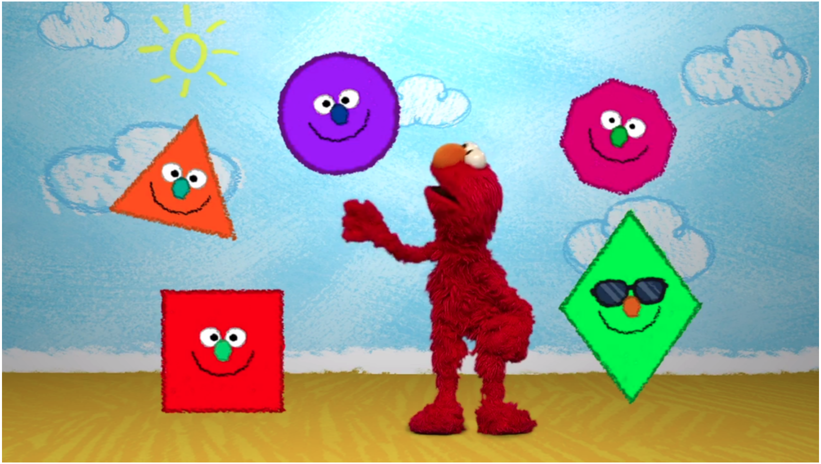
- Warning! A note on notation:
  - Math convention: A p-by-n matrix defines a linear map from $\mathbb{R}^n$ to $\mathbb{R}^p$
    - Matrices have shapes (output dimension, input dimension)
    - Let $\Theta \in \mathbb{R}^{p \times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{z} \in \mathbb{R}^p$
    - $\mathbf{z} = \Theta \cdot \mathbf{x}$ (or $\mathbf{W} \cdot \mathbf{x} + \mathbf{b}$)
  - Computer science convention (mostly): An n-by-p matrix defines a linear map from $\mathbb{R}^n$ to $\mathbb{R}^p$ (technically $\mathbb{R}^{1 \times n}$ to $\mathbb{R}^{1 \times p}$)
    - Matrices have shapes (input dimension, output dimension)
    - Let $\Theta \in \mathbb{R}^{n \times p}$, $\mathbf{x} \in \mathbb{R}^{1 \times n}$, and $\mathbf{z} \in \mathbb{R}^{1 \times p}$
    - $\mathbf{z} = \mathbf{x} \cdot \Theta$ (or $\mathbf{x} \cdot \mathbf{W} + \mathbf{b}$)
    - More intuitive (input $\rightarrow$ output)
    - Aligns with the convention in (mini)batch training that the first dimension is the batch size ("feature vectors are stacked row-wise")

General Advice

- Know your shapes!

General Advice

- Know your shapes!



Source

Multi-Class Classification

- What is the activation function g?

Multi-Class Classification

- What is the activation function g?
- Let $\mathbf{z} = \begin{bmatrix} z_1 & \dots & z_p \end{bmatrix}$ be a vector of scores for each class

Multi-Class Classification

- What is the activation function g?
- Let $\mathbf{z} = \begin{bmatrix} z_1 & \dots & z_p \end{bmatrix}$ be a vector of scores for each class
- Logistic regression: softmax function
  - $\text{softmax}(z_c) = \dfrac{e^{z_c}}{\sum_{k=1}^{p} e^{z_k}} = P(y = c|\mathbf{x}) = \hat{y}_c$

Multi-Class Classification

- What is the activation function g?
- Let $\mathbf{z} = \begin{bmatrix} z_1 & \dots & z_p \end{bmatrix}$ be a vector of scores for each class
- Logistic regression: softmax function
  - $\text{softmax}(z_c) = \frac{e^{z_c}}{\sum_{k=1}^{p} e^{z_k}} = P(y = c|\mathbf{x}) = \hat{y}_c$
  - $\text{softmax}(\mathbf{z}) = \hat{\mathbf{y}}$ is a vector of probabilities for each class

Multi-Class Classification

- What is the activation function g?
- Let $\mathbf{z} = \begin{bmatrix} z_1 & \dots & z_p \end{bmatrix}$ be a vector of scores for each class
- Logistic regression: softmax function
  - $\text{softmax}(z_c) = \frac{e^{z_c}}{\sum_{k=1}^{p} e^{z_k}} = P(y = c|\mathbf{x}) = \hat{y}_c$
  - $\text{softmax}(\mathbf{z}) = \hat{\mathbf{y}}$ is a vector of probabilities for each class
- Perceptron: argmax function
  - $\underset{k=1}{\overset{p}{\text{argmax}}}(z_k) = \hat{y}$

Multi-Class Classification

- What is the activation function g?
- Let $\mathbf{z} = \begin{bmatrix} z_1 & \dots & z_p \end{bmatrix}$ be a vector of scores for each class
- Logistic regression: softmax function
  - $\text{softmax}(z_c) = \frac{e^{z_c}}{\sum_{k=1}^{p} e^{z_k}} = P(y = c|\mathbf{x}) = \hat{y}_c$
  - $\text{softmax}(\mathbf{z}) = \hat{\mathbf{y}}$ is a vector of probabilities for each class
- Perceptron: argmax function
  - $\underset{k=1}{\overset{p}{\text{argmax}}}(z_k) = \hat{y}$
  - What if there is a tie?
    - Do whatever `numpy.argmax` does

Naïve Bayes as a Linear Classifier

- Suppose we observe a document d. What is the most likely class $\hat{c}$?

Naïve Bayes as a Linear Classifier

- Suppose we observe a document d. What is the most likely class $\hat{c}$?
- $P(c|d) = \dfrac{P(d|c)P(c)}{P(d)}$
  - Bayes' Rule
- $\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(d|c)P(c)$
  - $P(d)$ is the same for each class
- $\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, P(c) \prod_{i \in \text{positions}} P(w_i|c)$
  - Bag of words assumption, Naïve Bayes assumption
- $\hat{c} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in \text{positions}} \log P(w_i|c)$
  - If $xy = z$, then $\log(x) + \log(y) = \log(z)$

Naïve Bayes as a Linear Classifier

- $\hat{c} = \operatorname*{argmax}_{c \in C} \sum_{w \in |V|} \left[(\log P(w|c))(\text{count}(w,d)) \right] + \log P(c)$
  - $\sum_{i \in \text{positions}} \log P(w_i|c) = \sum_{w \in |V|} \left[(\log P(w|c))(\text{count}(w,d)) \right]$

Naïve Bayes as a Linear Classifier

- $\hat{c} = \operatorname*{argmax}_{c \in C} \sum_{w \in |V|} \left[(\log P(w|c))(\text{count}(w, d)) \right] + \log P(c)$
  - $\sum_{i \in \text{positions}} \log P(w_i|c) = \sum_{w \in |V|} \left[(\log P(w|c))(\text{count}(w, d)) \right]$
- Let $\ell_{cw} = \log P(w|c)$, $x_w = \text{count}(w, d)$, and $p_c = \log P(c)$

Naïve Bayes as a Linear Classifier

- $\hat{c} = \operatorname*{argmax}_{c \in C} \sum_{w \in |V|} \left[(\log P(w|c))(\text{count}(w, d)) \right] + \log P(c)$
  - $\sum_{i \in \text{positions}} \log P(w_i|c) = \sum_{w \in |V|} \left[(\log P(w|c))(\text{count}(w, d)) \right]$
- Let $\ell_{cw} = \log P(w|c)$, $x_w = \text{count}(w, d)$, and $p_c = \log P(c)$

$$\hat{c} = \operatorname*{argmax}_{c \in C} \sum_{w \in |V|} \ell_{cw} x_w + p_c$$

Naïve Bayes as a Linear Classifier

- $\hat{c} = \underset{c \in C}{\operatorname{argmax}} \sum_{w \in |V|} \left[(\log P(w|c))(\text{count}(w,d)) \right] + \log P(c)$
  - $\sum_{i \in \text{positions}} \log P(w_i|c) = \sum_{w \in |V|} \left[(\log P(w|c))(\text{count}(w,d)) \right]$
- Let $\ell_{cw} = \log P(w|c)$, $x_w = \text{count}(w,d)$, and $p_c = \log P(c)$

$$\begin{aligned} \hat{c} &= \underset{c \in C}{\operatorname{argmax}} \sum_{w \in |V|} \ell_{cw} x_w + p_c \\ &= \underset{c \in C}{\operatorname{argmax}} (\ell_c \cdot \mathbf{x} + p_c) \end{aligned}$$

Naïve Bayes as a Linear Classifier

- Let $\mathbf{x}$ be a feature vector, $\mathbf{p} = \texttt{self.prior}$, and $\mathcal{L} = \texttt{self.likelihood}$

Naïve Bayes as a Linear Classifier

- Let $\mathbf{x}$ be a feature vector, $\mathbf{p} = \texttt{self.prior}$, and $\mathcal{L} = \texttt{self.likelihood}$
- $\mathbf{z} = \mathcal{L} \cdot \mathbf{x} + \mathbf{p}$

Naïve Bayes as a Linear Classifier

- Let $\mathbf{x}$ be a feature vector, $\mathbf{p} = \texttt{self.prior}$, and $\mathcal{L} = \texttt{self.likelihood}$
- $\mathbf{z} = \mathcal{L} \cdot \mathbf{x} + \mathbf{p}$
- $\hat{c} = \operatorname*{argmax}_{c \in C}(z_c)$

Training Linear Classifiers

- Naïve Bayes: estimate parameters (log-prior, log-likelihood) directly from training data

Training Linear Classifiers

- Naïve Bayes: estimate parameters (log-prior, log-likelihood) directly from training data
- Logistic regression, perceptron:
  - Define a loss function

Training Linear Classifiers

- Naïve Bayes: estimate parameters (log-prior, log-likelihood) directly from training data
- Logistic regression, perceptron:
  - Define a loss function
  - Update the parameters using gradient descent

Loss Functions

- How wrong is your classifier?

Loss Functions

- How wrong is your classifier?
- Logistic regression: cross-entropy loss
  - $L(\hat{y}, y) = -\log P(y|\mathbf{x}) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$

Loss Functions

- How wrong is your classifier?
- Logistic regression: cross-entropy loss
  - $L(\hat{y}, y) = -\log P(y|\mathbf{x}) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$
    - Minimizing loss = maximizing the (log-)probability of the true y given $\mathbf{x}$

Loss Functions

- How wrong is your classifier?
- Logistic regression: cross-entropy loss
  - $L(\hat{y}, y) = -\log P(y|\mathbf{x}) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$
    - Minimizing loss = maximizing the (log-)probability of the true y given $\mathbf{x}$
- Perceptron: perceptron loss
  - $L(\hat{y}, y) = (\hat{y} - y)z$ (for $y \in \{0, 1\}$)

Loss Functions

- How wrong is your classifier?
- Logistic regression: cross-entropy loss
  - $L(\hat{y}, y) = -\log P(y|\mathbf{x}) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$
    - Minimizing loss = maximizing the (log-)probability of the true y given $\mathbf{x}$
- Perceptron: perceptron loss
  - $L(\hat{y}, y) = (\hat{y} - y)z$ (for $y \in \{0, 1\}$)
    - (You may also see $L = \max(0, -yz)$, for $y \in \{-1, 1\}$)

Loss Functions

- How wrong is your classifier?
- Logistic regression: cross-entropy loss
  - $L(\hat{y}, y) = -\log P(y|\mathbf{x}) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$
    - Minimizing loss = maximizing the (log-)probability of the true y given $\mathbf{x}$
- Perceptron: perceptron loss
  - $L(\hat{y}, y) = (\hat{y} - y)z$ (for $y \in \{0, 1\}$)
    - (You may also see $L = \max(0, -yz)$, for $y \in \{-1, 1\}$)
    - If $\hat{y} = y$, $L = 0$
    - If $\hat{y} \neq y$, $L > 0$

Derivatives

- The derivative of a function measures the instantaneous rate of change in a function's output with respect to a change in its input

Derivatives

- The derivative of a function measures the instantaneous rate of change in a function's output with respect to a change in its input
  - "Slope of a function's graph"

Derivatives

- The derivative of a function measures the instantaneous rate of change in a function's output with respect to a change in its input
  - "Slope of a function's graph"
- The derivative of f with respect to x is denoted $\dfrac{df}{dx}$, f', etc.

Derivatives

- Rules of differentiation
  - Constant rule: If $f(x)$ is constant, then $f'(x) = 0$
  - Sum rule: $(f+g)' = f' + g'$
  - Product rule: $(fg)' = f'g + fg'$
  - Power rule: If $f(x) = x^r$, then $f'(x) = rx^{r-1}$
  - Chain rule: If $h(x) = f(g(x))$, then $h'(x) = f'(g(x)) \cdot g'(x)$ (or $\frac{dh}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}$)

Derivatives

- Rules of differentiation
  - Constant rule: If $f(x)$ is constant, then $f'(x) = 0$
  - Sum rule: $(f+g)' = f' + g'$
  - Product rule: $(fg)' = f'g + fg'$
  - Power rule: If $f(x) = x^r$, then $f'(x) = rx^{r-1}$
  - Chain rule: If $h(x) = f(g(x))$, then $h'(x) = f'(g(x)) \cdot g'(x)$ (or $\frac{dh}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}$)
- Anything more complicated than this, we will tell you what the derivative is

Partial Derivatives

- The partial derivative of a function of several variables measures the instantaneous rate of change in a function's output with respect to a change in one of its inputs, with the others held constant

Partial Derivatives

- The partial derivative of a function of several variables measures the instantaneous rate of change in a function's output with respect to a change in one of its inputs, with the others held constant
- The partial derivative of f with respect to x is denoted $\dfrac{\partial f}{\partial x}$, f_x, etc.

Gradients

- The gradient of a function of several variables is a vector of partial derivatives

Gradients

- The gradient of a function of several variables is a vector of partial derivatives

$$\nabla F = \begin{bmatrix} \dfrac{\partial F}{\partial x_1} \\ \vdots \\ \dfrac{\partial F}{\partial x_n} \end{bmatrix}$$

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
    - $\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_n} \\ \frac{\partial L}{\partial b} \end{bmatrix}$

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- 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

Gradient Descent

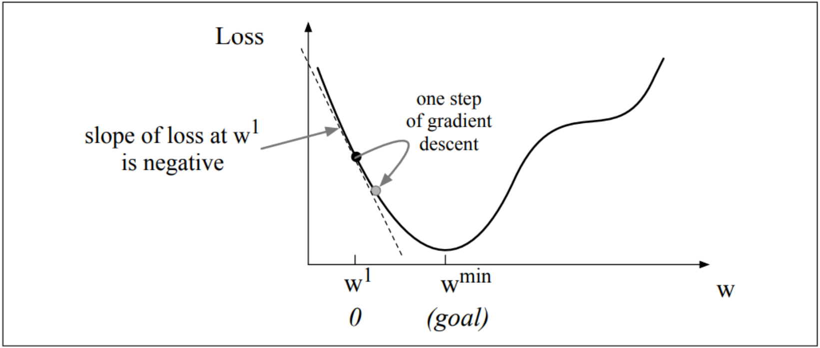


Figure 5.4 The first step in iteratively finding the minimum of this loss function, by moving w in the reverse direction from the slope of the function. Since the slope is negative, we need to move w in a positive direction, to the right. Here superscripts are used for learning steps, so w^1 means the 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 or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- At each time step t:
  - Compute gradient ∇L
  - Move in direction of negative gradient
- $\theta_{t+1} = \theta_t - \eta \nabla L$

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- 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 or $\mathbf{0}$)
- 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
    - "Hyperparameter": parameter set before training

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- 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
    - "Hyperparameter": parameter set before training
    - Trade-off between speed of convergence and "zig-zag" behavior

Gradient Descent

- Initialize parameters $\theta = \mathbf{w}, b$ (randomly or $\mathbf{0}$)
- 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
    - "Hyperparameter": parameter set before training
    - Trade-off between speed of convergence and "zig-zag" behavior
    - Often a function of t

Gradients in Logistic Regression

- $L(\hat{y}, y) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})]$

Gradients in Logistic Regression

- $L(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$
- ...
- (calculus–see supplement slides)
- ...

Gradients in Logistic Regression

- $L(\hat{y}, y) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$
- ...
- (calculus–see supplement slides)
- ...
- $\dfrac{\partial L}{\partial w_j} = (\hat{y} - y)x_j$
- $\dfrac{\partial L}{\partial b} = \hat{y} - y$

Gradients in Logistic Regression

- $L(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$
- ...
- (calculus–see supplement slides)
- ...
- $\dfrac{\partial L}{\partial w_j} = (\hat{y} - y)x_j$
- $\dfrac{\partial L}{\partial b} = \hat{y} - y$

- $\nabla L = (\hat{y} - y)\mathbf{x}$

Gradients in Perceptrons

- $L(\hat{y}, y) = (\hat{y} - y)z$

Gradients in Perceptrons

- $L(\hat{y}, y) = (\hat{y} - y)z$
- ...
- (calculus–see supplement slides)
- ...

Gradients in Perceptrons

- $L(\hat{y}, y) = (\hat{y} - y)z$
- ...
- (calculus–see supplement slides)
- ...
- $\dfrac{\partial L}{\partial w_j} = (\hat{y} - y)x_j$
- $\dfrac{\partial L}{\partial b} = \hat{y} - y$

Gradients in Perceptrons

- $L(\hat{y}, y) = (\hat{y} - y)z$
- ...
- (calculus–see supplement slides)
- ...
- $\frac{\partial L}{\partial w_j} = (\hat{y} - y)x_j$
- $\frac{\partial L}{\partial b} = \hat{y} - y$

- $\nabla L = (\hat{y} - y)\mathbf{x}$

Gradients in Perceptrons

- $L(\hat{y}, y) = (\hat{y} - y)z$
- ...
- (calculus–see supplement slides)
- ...
- $\dfrac{\partial L}{\partial w_j} = (\hat{y} - y)x_j$
- $\dfrac{\partial L}{\partial b} = \hat{y} - y$
- $\nabla L = (\hat{y} - y)\mathbf{x}$
- Does this look familiar?

Perceptron Learning Algorithm

- If $\hat{y} = y$, then $\hat{y} - y = 0$
  - Do nothing

Perceptron Learning Algorithm

- If $\hat{y} = y$, then $\hat{y} - y = 0$
  - Do nothing
- If $\hat{y} = 0$ and $y = 1$, then $\hat{y} - y = -1$
  - $\nabla L = -\mathbf{x}$
  - $\theta_{t+1} = \theta_t + \eta \mathbf{x}$
  - Increment weights

Perceptron Learning Algorithm

- If $\hat{y} = y$, then $\hat{y} - y = 0$
  - Do nothing
- If $\hat{y} = 0$ and $y = 1$, then $\hat{y} - y = -1$
  - $\nabla L = -\mathbf{x}$
  - $\theta_{t+1} = \theta_t + \eta \mathbf{x}$
  - Increment weights
- If $\hat{y} = 1$ and $y = 0$, then $\hat{y} - y = 1$
  - $\nabla L = \mathbf{x}$
  - $\theta_{t+1} = \theta_t - \eta \mathbf{x}$
  - Decrement weights

Gradients in Multinomial Logistic Regression

- Cross-entropy loss $L(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{p} y_k \log \hat{y}_k$

Gradients in Multinomial Logistic Regression

- Cross-entropy loss $L(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{p} y_k \log \hat{y}_k$
- Gradient ∇L becomes a matrix, where
  - $\dfrac{\partial L}{\partial w_{jk}} = (\hat{y}_k - y_k)x_j$
  - $\dfrac{\partial L}{\partial b_k} = \hat{y}_k - y_k$

Gradients in Multinomial Logistic Regression

- Cross-entropy loss $L(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{p} y_k \log \hat{y}_k$
- Gradient ∇L becomes a matrix, where
  - $\dfrac{\partial L}{\partial w_{jk}} = (\hat{y}_k - y_k)x_j$
  - $\dfrac{\partial L}{\partial b_k} = \hat{y}_k - y_k$
- $\nabla L = \mathbf{x} \otimes (\hat{\mathbf{y}} - \mathbf{y})$, where
  - $\otimes$ denotes the outer product

Multi-class Perceptron Learning Algorithm

- Multi-class perceptron loss $L(\hat{y}, y) = z_{\hat{y}} - z_y$

Multi-class Perceptron Learning Algorithm

- Multi-class perceptron loss $L(\hat{y}, y) = z_{\hat{y}} - z_y$
- If $\hat{y} = y$, then do nothing
- Else:

Multi-class Perceptron Learning Algorithm

- Multi-class perceptron loss $L(\hat{y}, y) = z_{\hat{y}} - z_y$
- If $\hat{y} = y$, then do nothing
- Else:
  - For the correct class y, $\dfrac{\partial L}{\partial z_y} = -1$
    - $(\nabla L)_y = -\mathbf{x}$
    - $(\theta_y)_{t+1} = (\theta_y)_t + \eta \mathbf{x}$
    - Increment weights

Multi-class Perceptron Learning Algorithm

- Multi-class perceptron loss $L(\hat{y}, y) = z_{\hat{y}} - z_y$
- If $\hat{y} = y$, then do nothing
- Else:
  - For the correct class y, $\dfrac{\partial L}{\partial z_y} = -1$
    - $(\nabla L)_y = -\mathbf{x}$
    - $(\theta_y)_{t+1} = (\theta_y)_t + \eta \mathbf{x}$
    - Increment weights
  - For the predicted class $\hat{y}$, $\dfrac{\partial L}{\partial z_{\hat{y}}} = 1$
    - $(\nabla L)_{\hat{y}} = \mathbf{x}$
    - $(\theta_{\hat{y}})_{t+1} = (\theta_{\hat{y}})_t - \eta \mathbf{x}$
    - Decrement weights

Multi-class Perceptron Learning Algorithm

- Multi-class perceptron loss $L(\hat{y}, y) = z_{\hat{y}} - z_y$
- If $\hat{y} = y$, then do nothing
- Else:
  - For the correct class y, $\dfrac{\partial L}{\partial z_y} = -1$
    - $(\nabla L)_y = -\mathbf{x}$
    - $(\theta_y)_{t+1} = (\theta_y)_t + \eta \mathbf{x}$
    - Increment weights
  - For the predicted class $\hat{y}$, $\dfrac{\partial L}{\partial z_{\hat{y}}} = 1$
    - $(\nabla L)_{\hat{y}} = \mathbf{x}$
    - $(\theta_{\hat{y}})_{t+1} = (\theta_{\hat{y}})_t - \eta \mathbf{x}$
    - Decrement weights
  - For other classes, do nothing

(Mini-)Batch Training

- What is a time step?

(Mini-)Batch Training

- What is a time step?
  - Stochastic gradient descent: update θ after every training example

(Mini-)Batch Training

- What is a time step?
  - Stochastic gradient descent: update θ after every training example
    - Can result in very choppy movements

(Mini-)Batch Training

- What is a time step?
  - Stochastic gradient descent: update θ after every training example
    - Can result in very choppy movements
  - Batch gradient descent: update θ after processing the entire training set

(Mini-)Batch Training

- What is a time step?
  - Stochastic gradient descent: update θ after every training example
    - Can result in very choppy movements
  - Batch gradient descent: update θ after processing the entire training set
  - Minibatch gradient descent: update θ after m training examples

(Mini-)Batch Training

- What is a time step?
  - Stochastic gradient descent: update θ after every training example
    - Can result in very choppy movements
  - Batch gradient descent: update θ after processing the entire training set
  - Minibatch gradient descent: update θ after m training examples
    - Gradient = average of individual gradients

(Mini-)Batch Training

- Let $\mathbf{x}$ consist of the feature vectors $\mathbf{x}^{(i)}$ for each document i in the (mini-)batch of size m, stacked on top of each other

(Mini-)Batch Training

- Let $\mathbf{x}$ consist of the feature vectors $\mathbf{x}^{(i)}$ for each document i in the (mini-)batch of size m, stacked on top of each other
- $\mathbf{x} = \begin{bmatrix} x_1^{(1)} & \dots & x_n^{(1)} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_1^{(m)} & \dots & x_n^{(m)} & 1 \end{bmatrix}$

(Mini-)Batch Training

- Let $\mathbf{x}$ consist of the feature vectors $\mathbf{x}^{(i)}$ for each document i in the (mini-)batch of size m, stacked on top of each other
- $\mathbf{x} = \begin{bmatrix} x_1^{(1)} & \dots & x_n^{(1)} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_1^{(m)} & \dots & x_n^{(m)} & 1 \end{bmatrix}$
- $\mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$

(Mini-)Batch Training

- $\nabla L = \frac{1}{m}\left(\mathbf{x}^T \cdot (\hat{\mathbf{y}} - \mathbf{y})\right)$

(Mini-)Batch Training

- $\nabla L = \frac{1}{m}\left(\mathbf{x}^T \cdot (\hat{\mathbf{y}} - \mathbf{y})\right)$
  - What is $\mathbf{x}^T \cdot (\hat{\mathbf{y}} - \mathbf{y})$?

(Mini-)Batch Training

$$\begin{bmatrix} x_1^{(1)} & \dots & x_1^{(m)} \\ \vdots & \ddots & \vdots \\ x_n^{(1)} & \dots & x_n^{(m)} \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \vdots \\ \hat{y}^{(m)} - y^{(m)} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) x_n^{(i)} \\ \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) \end{bmatrix}$$

(Mini-)Batch Training

$$
\begin{bmatrix} x_1^{(1)} & \dots & x_1^{(m)} \\ \vdots & \ddots & \vdots \\ x_n^{(1)} & \dots & x_n^{(m)} \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \hat{y}^{(1)} - y^{(1)} \\ \vdots \\ \hat{y}^{(m)} - y^{(m)} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) x_n^{(i)} \\ \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) \end{bmatrix}
$$

$$
= \begin{bmatrix} \displaystyle\sum_{i=1}^{m} \left(\frac{\partial L}{\partial w_1} \right)^{(i)} \\ \vdots \\ \displaystyle\sum_{i=1}^{m} \left(\frac{\partial L}{\partial w_n} \right)^{(i)} \\ \displaystyle\sum_{i=1}^{m} \left(\frac{\partial L}{\partial b} \right)^{(i)} \end{bmatrix}
$$

(Mini-)Batch Training

$$
\begin{bmatrix}
x_1^{(1)} & \dots & x_1^{(m)} \\
\vdots & \ddots & \vdots \\
x_n^{(1)} & \dots & x_n^{(m)} \\
1 & 1 & 1
\end{bmatrix}
\cdot
\begin{bmatrix}
\hat{y}^{(1)} - y^{(1)} \\
\vdots \\
\hat{y}^{(m)} - y^{(m)}
\end{bmatrix}
=
\begin{bmatrix}
\sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) x_1^{(i)} \\
\vdots \\
\sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) x_n^{(i)} \\
\sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})
\end{bmatrix}
$$

$$
=
\begin{bmatrix}
\sum\limits_{i=1}^{m} \left(\frac{\partial L}{\partial w_1} \right)^{(i)} \\
\vdots \\
\sum\limits_{i=1}^{m} \left(\frac{\partial L}{\partial w_n} \right)^{(i)} \\
\sum\limits_{i=1}^{m} \left(\frac{\partial L}{\partial b} \right)^{(i)}
\end{bmatrix}
$$

$$
= \sum_{i=1}^{m} (\nabla L)^{(i)}
$$

(Mini-)Batch Training

- $\nabla L = \frac{1}{m}\left(\mathbf{x}^T \cdot (\hat{\mathbf{y}} - \mathbf{y})\right)$
  - What is $\mathbf{x}^T \cdot (\hat{\mathbf{y}} - \mathbf{y})$?
    - It computes the sum of the gradients for each document i in the mini-batch!

(Mini-)Batch Training

- $\nabla L = \frac{1}{m}\left(\mathbf{x}^T \cdot (\hat{\mathbf{y}} - \mathbf{y})\right)$
  - What is $\mathbf{x}^T \cdot (\hat{\mathbf{y}} - \mathbf{y})$?
    - It computes the sum of the gradients for each document i in the mini-batch!
    - Then to get the average gradient, we just divide by m