CS440/ECE448
Lecture 7:
Linear Regression

Mark Hasegawa-Johnson, 1/2022
License: CC-BY 4.0; redistribute at will, as long as you cite the source.
Outline

• Review: perceptron
• Linear regression: a neuron without the nonlinearity
• Mean-squared error
• Learning the solution: stochastic gradient descent
• Multiple linear regression
Review: Perceptron Learning Algorithm

For each training instance $\tilde{x}$ with ground truth label $y \in \{-1, 1\}$:

- Classify with current weights: $f(\tilde{x}) = \text{sign}(\vec{w}^T \tilde{x})$
- Update weights:
  - If $f(\tilde{x}) = y$ then do nothing
  - If $f(\tilde{x}) \neq y$ then
    $$\vec{w} = \vec{w} + y\tilde{x} = \begin{cases} 
    \vec{w} + \tilde{x} & y = +1 \\
    \vec{w} - \tilde{x} & y = -1 
    \end{cases}$$
Example

Now we have $\vec{w}^T = [0,2,1]$.

Suppose the next token is $\vec{x}^T = [-2,1,1]$, with the label $y = -1$. Since $f(x)$ is wrong, we update:

$$\vec{w} = \vec{w} - \vec{x}$$

$$\vec{w} = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix} - \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$$

$\vec{x} = \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}$, $y = -1$
Review: Multi-Class Perceptron

For each training instance $\tilde{x}$ with ground truth label $y \in \{-1, 1\}$:

- Classify with current weights: $f(\tilde{x}) = \arg\max_c (\mathbf{w}_c^T \tilde{x})$

- Update weights:
  - If $f(\tilde{x}) = y$ then do nothing
  - If $f(\tilde{x}) \neq y$ then
    $$\mathbf{w}_y = \mathbf{w}_y + \eta \tilde{x}$$
    $$\mathbf{w}_{f(\tilde{x})} = \mathbf{w}_{f(\tilde{x})} - \eta \tilde{x}$$
Notation: Vector dot product, with bias added

\[ f(\vec{x}) = \text{argmax}_c (\vec{w}_c^T \vec{x}) \]

\[ \vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_D \\ 1 \end{bmatrix}, \quad \vec{w}_c = \begin{bmatrix} w_{c,1} \\ \vdots \\ w_{c,D} \\ b_c \end{bmatrix} \]

\[ \vec{w}_c^T \vec{x} = b_c + \sum_{j=1}^{D} w_{c,j} x_j \]
Outline

• Review: perceptron
• Linear regression: a neuron without the nonlinearity
• Mean-squared error
• Learning the solution: stochastic gradient descent
• Multiple linear regression
Linear regression

Linear regression is used to estimate a real-valued target variable, $y$, using a linear combination of real-valued input variables:

$$f(x) = \mathbf{w}^T \mathbf{x} = b + \sum_{j=1}^{D} w_j x_j$$

... so that ...

$$f(x) \approx y$$
Linear regression is like a neuron without a nonlinearity

- The neuron’s **excitation** is
  \[
  \overrightarrow{w}^T \overrightarrow{x} = b + \sum_{j=1}^{D} w_j x_j
  \]

- The neuron’s **activation** is
  \[
  f(x) = \overrightarrow{w}^T \overrightarrow{x}
  \]
Polynomial regression = multivariate linear regression

We can use linear regression to solve nonlinear regression problems by simply augmenting the features. For example, suppose we start with just one input variable, $x$, but suppose we expand it to four variables like this:

$$\tilde{x} = \begin{bmatrix} x \\ x^2 \\ x^3 \\ 1 \end{bmatrix}$$

Then

$$f(x) = \tilde{w}^T \tilde{x} = b + w_1 x + w_2 x^2 + w_3 x^3$$
Multivariate linear regression in general

More generally, multivariate linear regression fits a D-dimensional hyperplane in the (D+1)-dimensional space \((x_1, \ldots, x_D, y)\):

\[
\begin{bmatrix}
    x_1 \\
    \vdots \\
    x_D \\
    1
\end{bmatrix}
\]

Then

\[
f(\hat{x}) = \vec{w}^T \hat{x} = b + \sum_{j=1}^{D} w_j x_j
\]
Outline

• Review: perceptron
• Linear regression: a neuron without the nonlinearity
• Mean-squared error
• Learning the solution: stochastic gradient descent
• Multiple linear regression
What does it mean that $f(x) \approx y$?

- Generally, we want to choose the weights and bias, $\mathbf{w}$, in order to minimize the errors.
- The errors are the vertical green bars in the figure at right, $\epsilon = f(\bar{x}) - y$
- Some of them are positive, some are negative. What does it mean to “minimize” them?
First: count the training tokens

Let’s introduce one more index variable. Let $i=$ the index of the training token.

$$\vec{x}_i = \begin{bmatrix} x_{i,1} \\ \vdots \\ x_{i,D} \\ 1 \end{bmatrix}$$

$$f(\vec{x}_i) = \vec{x}_i^T \vec{w} = b + \sum_{j=1}^{D} x_{i,j} w_j$$
Training token errors

Using that notation, we can define a signed error term for every training token:

\[ \epsilon_i = f(\hat{x}_i) - y_i \]

The error term is positive for some tokens, negative for other tokens. What does it mean to minimize it?
Mean-squared error

One useful criterion (not the only useful criterion, but perhaps the most common) of “minimizing the error” is to minimize the mean squared error:

\[
MSE = \frac{1}{n} \sum_{i=1}^{n} \epsilon_i^2
\]

\[
= \frac{1}{n} \sum_{i=1}^{n} (\tilde{x}_i^T \tilde{w} - y_i)^2
\]

Literally,

• ... the mean ...
• ... of the square ...
• ... of the error terms.
Outline

- Review: perceptron
- Linear regression: a neuron without the nonlinearity
- Mean-squared error
- Learning the solution: stochastic gradient descent
- Multiple linear regression
Minimizing the MSE

Our goal is to find the coefficients $\vec{w} = [w_1, ..., w_D, b]^T$ that minimize the MSE:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\vec{x}_i^T \vec{w} - y_i)^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left( b + \sum_{j=1}^{D} w_j x_{i,j} - y_i \right)^2$$
MSE = Parabola

Notice that, although it looks kind of complicated, the MSE is just a parabola in terms of $b$ and $w_j$:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \left( b + \sum_{j=1}^{D} w_j x_{i,j} - y_i \right)^2$$

Since it’s a parabola, it has a unique minimum that you can compute in closed form! But we won’t do that today. Instead…
The iterative solution to linear regression

Instead of minimizing MSE in closed form, we’re going to use an iterative algorithm called gradient descent. It works like this:

• Start from a random initial value of \( \vec{w} \) (at \( t = 0 \)).
• Adjust \( \vec{w} \) in order to reduce MSE (\( t = 1 \)).
• Repeat until you reach the optimum (\( t = \infty \)).
The gradient descent algorithm

• Start from a random initial value of $\vec{w}$.
• Calculate the derivative of MSE with respect to $\vec{w}$:

$$\nabla_{\vec{w}} MSE = \begin{bmatrix}
\frac{\partial MSE}{\partial w_1} \\
\vdots \\
\frac{\partial MSE}{\partial w_D} \\
\frac{\partial MSE}{\partial b}
\end{bmatrix}$$

• Take a step “downhill” (in the direction of the negative gradient)

$$\vec{w} \leftarrow \vec{w} - \frac{\eta}{2} \nabla_{\vec{w}} MSE$$

...where $\eta$ is a constant called the “learning rate,” that determines how big of a step you take. Usually, you need to adjust $\eta$ in order to get optimum performance on a dev set, but often $\eta \approx 0.001$. 

$$MSE = aw^2 + bw + c$$
Stochastic gradient descent

• If $n$ is large, computing or differentiating MSE can be expensive.
• The stochastic gradient descent algorithm picks one training token $(\tilde{x}_i, y_i)$ at random (“stochastically”), and adjusts $\vec{w}$ in order to reduce the error a little bit for that one token:

$$\vec{w} \leftarrow \vec{w} - \frac{\eta}{2} \nabla_\vec{w} \epsilon_i^2$$

...where

$$\epsilon_i^2 = (\tilde{x}_i^T \vec{w} - y_i)^2$$
Stochastic gradient descent

\[ \epsilon_i^2 = (\hat{x}_i^T \bar{w} - y_i)^2 \]

If we differentiate that, we discover that:

\[ \nabla_{\bar{w}} \epsilon_i^2 = 2 \epsilon_i \hat{x}_i \]

So the stochastic gradient descent algorithm is:

\[ \bar{w} \leftarrow \bar{w} - \eta \epsilon_i \hat{x}_i \]
Comparison of perceptron and linear regression

Perceptron:
• If $f(\vec{x}) = y$ then do nothing
• If $f(\vec{x}) = 1$ but $y = -1$:
  \[ \vec{w} \leftarrow \vec{w} - \eta \vec{x} \]
• If $f(\vec{x}) = -1$ but $y = 1$:
  \[ \vec{w} \leftarrow \vec{w} + \eta \vec{x} \]

Linear regression:
• If $\epsilon_i = 0$ then do nothing
• If $\epsilon_i > 0$:
  \[ \vec{w} \leftarrow \vec{w} - \eta \epsilon_i \vec{x}_i \]
• If $\epsilon_i < 0$:
  \[ \vec{w} \leftarrow \vec{w} - \eta \epsilon_i \vec{x}_i \]
Outline

• Review: perceptron
• Linear regression: a neuron without the nonlinearity
• Mean-squared error
• Learning the solution: stochastic gradient descent
• Multiple linear regression
What if $\hat{y}$ is a vector?

Sometimes we want to model a system with many inputs, and many outputs. In that case, $y$ and $f(x)$ both become vectors:

$$\hat{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_V \end{bmatrix}, \quad f(\hat{x}) = \begin{bmatrix} f_1(\hat{x}) \\ \vdots \\ f_V(\hat{x}) \end{bmatrix}$$

...and our goal is to find a function $f(\hat{x})$ so that $f(\hat{x}) \approx \hat{y}$.
What if $\hat{y}$ is a vector?

We can model this using multiple $\mathbf{w}_c$ vectors, a little like we did for multi-class perceptron:

$$f_c(\hat{x}) = \mathbf{w}_c^T \hat{x} = b_c + \sum_{j=1}^{D} w_{c,j} x_j$$
... it means $\hat{\epsilon}_i$ is a vector!

The gradient of the error with respect to each of the weights is:

$$\epsilon_{c,i} = f_c(\hat{x}_i) - y_{c,i}$$
This means that we have a different error term for each of the weight vectors:

$$\nabla \tilde{w}_c \epsilon^2_{c,i} = 2 \epsilon_{c,i} \tilde{x}_i$$

... so the stochastic gradient descent update step is

$$\tilde{w}_c \leftarrow \tilde{w}_c - \eta \epsilon_{c,i} \tilde{x}_i$$
Comparison of Multi-Class Perceptron to Multiple Regression

**Multi-Class Perceptron**

$$\text{Input: } x_1, x_2, \ldots, x_D$$

$$\text{Weights: } w_0, b_0, w_1, b_1, \ldots, w_{V-1}, b_{V-1}, \ldots, w_D$$

$$\text{Output: } \arg\max_{c=1}^V (w_c^T \tilde{x})$$

**Multiple Regression**

$$\text{Input: } x_1, x_2, \ldots, x_D$$

$$\text{Weights: } w_0, b_0, w_1, b_1, \ldots, w_{V-1}, b_{V-1}, \ldots, w_D$$

$$f_1(\tilde{x}) = \tilde{w}_1^T \tilde{x}$$

$$f_2(\tilde{x}) = \tilde{w}_2^T \tilde{x}$$

$$f_V(\tilde{x}) = \tilde{w}_V^T \tilde{x}$$
Summary

- Review: perceptron
- Linear regression: a neuron without the nonlinearity
  \[ f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \]
- Mean-squared error
  \[ MSE = \frac{1}{n} \sum_{i=1}^{n} \epsilon_i^2 = \frac{1}{n} \sum_{i=1}^{n} (\hat{x}_i^T \mathbf{w} - y_i)^2 \]
- Learning the solution: stochastic gradient descent
  \[ \mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{2} \nabla_{\mathbf{w}} \epsilon_i^2 = \mathbf{w} - \eta \epsilon_i \hat{x}_i \]
- Multiple linear regression
  \[ \mathbf{w}_c \leftarrow \mathbf{w}_c - \frac{\eta}{2} \nabla_{\mathbf{w}_c} \epsilon_{c,i}^2 = \mathbf{w}_c - \eta \epsilon_{c,i} \hat{x}_i \]