Regression

7 February 2024
Assignment 2 clarification and extension
How can advances in machine learning advance our understanding of human development? In this talk, I’ll use deep neural networks to address two classic debates: (1) How much language is learnable from sensory input? Using head-mounted video recordings from a single child, we show how deep neural networks can acquire many word-referent mappings, generalize to novel visual referents, and achieve multi-modal alignment. (2) Can neural networks capture human-like systematic generalization? We address a 35-year-old argument that neural networks are not viable cognitive models because they lack systematic compositionality—the algebraic ability to understand and produce novel combinations from known components. Neural networks can achieve human-like systematic generalization when trained through meta-learning for compositionality, a new method for optimizing compositional skills through practice. These findings emphasize the power of neural networks and their increasing capability for addressing long standing issues in cognitive science.

Brenden M. Lake is an Assistant Professor of Psychology and Data Science at New York University. He received his M.S. and B.S. in Symbolic Systems from Stanford University in 2009, and his Ph.D. in Cognitive Science from MIT in 2014. He was a postdoctoral Data Science Fellow at NYU from 2014-2017. Brenden is a recipient of the Robert J. Glushko Prize for Outstanding Doctoral Dissertation in Cognitive Science, he is a MIT Technology Review Innovator Under 35, and his research was selected by Scientific American as one of the 10 most important advances of 2016.
Where are we?
A *supervised machine-learning* text classification problem takes this form:

**Input:**
- A document \( d \)
- A fixed set of classes \( C = \{c_1, c_2, \ldots, c_j\} \)
- A training set of \( m \) hand-labeled documents \((d_1, c_1), \ldots, (d_m, c_m)\)

**Output:**
- A learned classifier \( \gamma : d \rightarrow c \)
Supervised machine learning
Naïve Bayes is a simple classification method based on Bayes’ Rule.

For text classification, we can use it with a simple representation of a document as a bag of words.
I love this movie! It’s sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun… It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I’ve seen it several times, and I’m always happy to see it again whenever I have a friend who hasn’t seen it yet!
After finding the parameters for the classifier – i.e., training it – we test how well it does on a test set of examples that weren’t used for training.
### The 2x2 confusion matrix

<table>
<thead>
<tr>
<th>Predict</th>
<th>Gold +</th>
<th>Gold -</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>true positive</td>
<td>false positive</td>
</tr>
<tr>
<td>-</td>
<td>false negative</td>
<td>true negative</td>
</tr>
</tbody>
</table>
The most common $F$-score is $F_1$, which is the harmonic mean of precision and recall:

$$F_1 = \frac{2PR}{P + R}$$
Generative and discriminative classifiers
Naïve Bayes is a *generative* classifier.

Logistic regression is a *discriminative* classifier.
Suppose we’re distinguishing cat images from dog images:

*From me*

*From ImageNet*
Generative classifier

Build a model of what’s in a cat image:

- Knows about whiskers, ears, eyes
- Assigns a probability to any image: How cat-y is this image?

Also build a model for dog images:

- How dog-y is this image?

Now, given a new image, run both models and see which one fits better.
Discriminative classifier

Just try to distinguish dogs from cats.

It looks like dogs have collars and cats don’t. That’s all I need to know.
Naïve Bayes (generative)

\[ \hat{c} = \arg\max_{c \in C} P(d \mid c) P(c) \]

Logistic regression (discriminative)

\[ \hat{c} = \arg\max_{c \in C} P(c \mid d) \]
**Naïve Bayes**: Count up the number of co-occurrences of features and classes in one go

**Logistic regression**: Weights are learned using an iterative procedure
Logistic regression classifiers
Each input observation $x$ is represented by a feature vector $[x_1, x_2, \ldots, x_n]$.

The output of the classifier can be one of two predicted classes, 0 or 1.
To be able to correctly classify inputs, we learn how predictive a feature $x_i$ is of either class by finding a corresponding weight $w_i$, e.g.,

- $x_1 =$ “input contains awesome” $w_1 = +10$
- $x_2 =$ “review contains abysmal” $w_2 = -10$
- $x_3 =$ “review contains mediocre” $w_3 = -2$
To use the weights to classify an instance, we multiply each feature $x_i$ by its corresponding weight $w_i$ and add them up:

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

The last term, $b$, is the **bias** (or **intercept**).
To use the weights to classify an instance, we multiply each feature $x_i$ by its corresponding weight $w_i$ and add them up:

$$z = w \cdot x + b$$

The last term, $b$, is the bias (or intercept).
Otherwise, predict it belongs to the negative category (0).

If $z$ is high, predict $x$ belongs to the positive category (1).

$$z = \mathbf{w} \cdot \mathbf{x} + b$$
The problem is we don’t have a fixed range of values for the sum $z$, so it’s not clear what counts as being “high”.

**Solution**: Make it a probability, between 0 and 1.
We can turn $z$ into a probability by passing it through the \textit{sigmoid} function $\sigma$:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$
Making probabilities with sigmoids:

\[ P(y = 1) = \sigma(w \cdot x + b) \]
Making probabilities with sigmoids:

\[ P(y = 1) = \sigma(w \cdot x + b) \]

\[ = \frac{1}{1 + \exp(- (w \cdot x + b))} \]
Making probabilities with sigmoids:

\[
P(y = 1) = \sigma(w \cdot x + b) = \frac{1}{1 + \exp(- (w \cdot x + b))}
\]

\[
P(y = 0) = 1 - \sigma(w \cdot x + b)
\]
So, for a particular input \( x \), we can now compute \( P(y = 1 \mid x) \) and \( P(y = 0 \mid x) \).

To turn these probabilities into a classifier, we just use the *decision boundary* 0.5:

\[
\text{decision}(x) = \begin{cases} 
1 & \text{if } P(y = 1 \mid x) > 0.5 \\
0 & \text{otherwise}
\end{cases}
\]
Logistic regression example: 
Text classification
It's hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable?

For one thing, the cast is great.

Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.

Is this review positive (y = 1) or negative (y = 0)?
It's hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable?

For one thing, the cast is great. Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.

<table>
<thead>
<tr>
<th>Var</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>count(positive lexicon words $\in$ doc)</td>
<td>3</td>
</tr>
</tbody>
</table>
It's hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable?

For one thing, the cast is great.

Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.

<table>
<thead>
<tr>
<th>Var</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>count(positive lexicon words $\in$ doc)</td>
<td>3</td>
</tr>
<tr>
<td>$x_2$</td>
<td>count(negative lexicon words $\in$ doc)</td>
<td>2</td>
</tr>
</tbody>
</table>
It’s hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable?

For one thing, the cast is great.

Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it’ll do the same to you.

<table>
<thead>
<tr>
<th>Var</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>count(positive lexicon words $\in$ doc)</td>
<td>3</td>
</tr>
<tr>
<td>$x_2$</td>
<td>count(negative lexicon words $\in$ doc)</td>
<td>2</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$\begin{cases} 1 \text{ if &quot;no&quot;$\in$ doc} \ 0 \text{ otherwise} \end{cases}$</td>
<td>1</td>
</tr>
</tbody>
</table>
It's hokey. There are virtually no surprises, and the writing is second-rate.
So why was it so enjoyable?

For one thing, the cast is great.

Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.
It's hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable?

For one thing, the cast is great.

Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.

<table>
<thead>
<tr>
<th>Var</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>count(positive lexicon words $\in$ doc)</td>
</tr>
<tr>
<td>$x_2$</td>
<td>count(negative lexicon words $\in$ doc)</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$\begin{cases} 1 &amp; \text{if } \text{“no”} \in \text{doc} \ 0 &amp; \text{otherwise} \end{cases}$</td>
</tr>
<tr>
<td>$x_4$</td>
<td>count(1st and 2nd pronouns $\in$ doc)</td>
</tr>
<tr>
<td>$x_5$</td>
<td>$\begin{cases} 1 &amp; \text{if } \text{“!”} \in \text{doc} \ 0 &amp; \text{otherwise} \end{cases}$</td>
</tr>
</tbody>
</table>
It's hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable?

For one thing, the cast is great.

Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.

<table>
<thead>
<tr>
<th>Var</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>count(positive lexicon words $\in$ doc)</td>
<td>3</td>
</tr>
<tr>
<td>$x_2$</td>
<td>count(negative lexicon words $\in$ doc)</td>
<td>2</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$\begin{cases} 1 &amp; \text{if &quot;no&quot; $\in$ doc} \ 0 &amp; \text{otherwise} \end{cases}$</td>
<td>1</td>
</tr>
<tr>
<td>$x_4$</td>
<td>count(1st and 2nd pronouns $\in$ doc)</td>
<td>3</td>
</tr>
<tr>
<td>$x_5$</td>
<td>$\begin{cases} 1 &amp; \text{if &quot;!&quot; $\in$ doc} \ 0 &amp; \text{otherwise} \end{cases}$</td>
<td>0</td>
</tr>
<tr>
<td>$x_6$</td>
<td>ln(word count of doc)</td>
<td>$\ln(66) = 4.19$</td>
</tr>
</tbody>
</table>
Suppose we learned these weights and bias

\[ \mathbf{w} = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \quad b = 0.1 \]

<table>
<thead>
<tr>
<th>Var</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>count(positive lexicon words (\in) doc)</td>
<td>3</td>
</tr>
<tr>
<td>(x_2)</td>
<td>count(negative lexicon words (\in) doc)</td>
<td>2</td>
</tr>
<tr>
<td>(x_3)</td>
<td>(\begin{cases} 1 &amp; \text{if “no” (\in) doc} \ \ 0 &amp; \text{otherwise} \end{cases})</td>
<td>1</td>
</tr>
<tr>
<td>(x_4)</td>
<td>count(1st and 2nd pronouns (\in) doc)</td>
<td>3</td>
</tr>
<tr>
<td>(x_5)</td>
<td>(\begin{cases} 1 &amp; \text{if “!” (\in) doc} \ \ 0 &amp; \text{otherwise} \end{cases})</td>
<td>0</td>
</tr>
<tr>
<td>(x_6)</td>
<td>(\ln(\text{word count of doc}))</td>
<td>(\ln(66) = 4.19)</td>
</tr>
</tbody>
</table>
\[
\mathbf{w} = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \quad b = 0.1
\]

\[
\mathbf{x} = [3, 2, 1, 3, 0, 4.19]
\]

<table>
<thead>
<tr>
<th>Var</th>
<th>Definition</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>count(positive lexicon words (\in) doc)</td>
<td>3</td>
</tr>
<tr>
<td>(x_2)</td>
<td>count(negative lexicon words (\in) doc)</td>
<td>2</td>
</tr>
</tbody>
</table>
| \(x_3\) | \[
\begin{cases}
1 & \text{if "no" \(\in\) doc} \\
0 & \text{otherwise}
\end{cases}
\] | 1     |
| \(x_4\) | count(1st and 2nd pronouns \(\in\) doc)                       | 3     |
| \(x_5\) | \[
\begin{cases}
1 & \text{if "!" \(\in\) doc} \\
0 & \text{otherwise}
\end{cases}
\] | 0     |
| \(x_6\) | \(\ln(\text{word count of doc})\)                           | \(\ln(66) = 4.19\) |
\[ \mathbf{w} = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \quad b = 0.1 \]

\[ \mathbf{x} = [3, 2, 1, 3, 0, 4.19] \]

\[ P(y = 1) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) \]
\[ w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \quad b = 0.1 \]
\[ x = [3, 2, 1, 3, 0, 4.19] \]

\[ P(y = 1) = \sigma(w \cdot x + b) \]
\[ = \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \]
\[ \mathbf{w} = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \quad b = 0.1 \]

\[ \mathbf{x} = [3, 2, 1, 3, 0, 4.19] \]

\[
P(y = 1) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)
= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1)
= \sigma(0.833)
\]
\[ w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \quad b = 0.1 \]

\[ x = [3, 2, 1, 3, 0, 4.19] \]

\[ P(y = 1) = \sigma(w \cdot x + b) \]
\[ = \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \]
\[ = \sigma(0.833) \]
\[ = 0.70 \]
\[ w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \quad b = 0.1 \]

\[ x = [3, 2, 1, 3, 0, 4.19] \]

\[
P(y = 1) = \sigma(w \cdot x + b) \\
= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\
= \sigma(0.833) \\
= 0.70
\]

\[
P(y = 0) = 1 - \sigma(w \cdot x + b) = 0.30
\]
How does learning work?
In supervised classification, for each example in the training data, we know the correct label $y$ (0 or 1).

The classifier produces an estimated label, $\hat{y}$.

We want to set the weights $\mathbf{w}$ and bias $b$ to minimize the distance between our estimate $\hat{y}$ and the true $y$ for each example.
In supervised classification, for each example in the training data, we know the correct label $y$ (0 or 1).

The classifier produces an estimated label, $\hat{y}$.

We want to set the weights $w$ and bias $b$ to minimize the distance between our estimate $\hat{y}$ and the true $y$ for each example.

We estimate this distance using a “loss function” or “cost function”.
In supervised classification, for each example in the training data, we know the correct label $y$ (0 or 1).

The classifier produces an estimated label, $\hat{y}$.

We want to **set the weights $w$ and bias $b$ to minimize the distance between our estimate $\hat{y}$ and the true $y$ for each example.**

We need an algorithm to iteratively update these to minimize the loss.
We want to choose the parameters $\mathbf{w}$ and $b$ that maximize the probability of the correct label $P(y \mid x)$. This is called \textit{conditional maximum likelihood estimation}. 
Since there are only two discrete outcomes (0 or 1), we can express the probability $P(y \mid x)$ from our classifier generically as

$$P(y \mid x) = \hat{y}^{y} (1 - \hat{y})^{1-y}$$

noting that

- if $y = 1$, this simplifies to $\hat{y}$
- if $y = 0$, this simplifies to $1 - \hat{y}$
\[ P(y \mid x) = \hat{y}^y (1 - \hat{y})^{1-y} \]

\[ \log P(y \mid x) = \log [ \hat{y}^y (1 - \hat{y})^{1-y} ] \]

\[ = y \log \hat{y} + (1 - y) \log(1 - \hat{y}) \]
This is a probability to maximize

\[ P(y \mid x) = \hat{y}^y (1 - \hat{y})^{1-y} \]

\[ \log P(y \mid x) = \log [ \hat{y}^y (1 - \hat{y})^{1-y} ] \]

\[ = y \log \hat{y} + (1 - y) \log(1 - \hat{y}) \]
This is a probability to maximize

\[ P(y \mid x) = \hat{y}^y (1 - \hat{y})^{1-y} \]

\[
\log P(y \mid x) = \log [ \hat{y}^y (1 - \hat{y})^{1-y} ] \\
= y \log \hat{y} + (1 - y) \log(1 - \hat{y})
\]

This is a measure of loss, to minimize

\[-\log P(y \mid x) = - [ y \log \hat{y} + (1 - y) \log(1 - \hat{y}) ]\]
$L(\hat{y}, y)$ is the loss function, expressing how far the classifier output $\hat{y}$ is from the correct output $y$.

We just derived the **cross-entropy loss**:

$$L_{\text{CE}}(\hat{y}, y) = -\log P(y \mid x)$$

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

$$= - [y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))]$$
We want the loss to be

smaller if the model estimate is close to correct

bigger if the model is confused
Suppose the true label is $y = 1$ (it’s a positive review).

How well is our classifier doing?

It's hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable?

For one thing, the cast is great.

Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.
\[ \mathbf{w} = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \quad b = 0.1 \]
\[ \mathbf{x} = [3, 2, 1, 3, 0, 4.19] \]

\[ P(y = 1) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = 0.70 \]
\( \mathbf{w} = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \quad b = 0.1 \)

\( \mathbf{x} = [3, 2, 1, 3, 0, 4.19] \)

\[ P(y = 1) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = 0.70 \]

Pretty well! What’s the loss?
\[ \mathbf{w} = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \quad b = 0.1 \]
\[ \mathbf{x} = [3, 2, 1, 3, 0, 4.19] \]

\[ P(y = 1) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = 0.70 \]

Pretty well! What’s the loss?

\[
\begin{align*}
L_{CE}(\hat{y}, y) &= -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1 - y) \log (1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b))] \\
&= -[\log \sigma(\mathbf{w} \cdot \mathbf{x} + b)] \\
&= -\log(.70) \\
&= .36
\end{align*}
\]
What if the true label were $y = 0$ (it’s a negative review)?

It's hokey. There are virtually no surprises, and the writing is second-rate.
So why was it so enjoyable?

For one thing, the cast is great.

Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.
\[ P(y = 1) = \sigma(w \cdot x + b) = 0.70 \]

\[ P(y = 0) = 1 - P(y = 1) = 0.30 \]
\[ P(y = 1) = \sigma(w \cdot x + b) = 0.70 \]
\[ P(y = 0) = 1 - P(y = 1) = 0.30 \]

\[
L_{CE}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))] \\
= -[\log (1 - \sigma(w \cdot x + b))] \\
= -\log (.30) \\
= 1.2
\]
The loss when the model was right (true $y = 1$) is lower than the loss when the model was wrong (true $y = 0$), which is exactly what we want for a measure we’re going to minimize!
Stochastic gradient descent
Our goal: Minimize the loss

Let’s make explicit that the loss function is parameterized by weights $\theta = (w, b)$.

We’ll represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on $\theta$ more obvious.

We want the weights that minimize the loss, averaged over all examples:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(f(x^{(i)}; \theta), y^{(i)})$$
Intuition of gradient descent

How do I get to the bottom of this river canyon?

Look around me 360°.

Find the direction of steepest slope down.

Go that way.
For logistic regression, the loss function is convex.

A convex function has just one minimum, so gradient descent starting from any point is guaranteed to find the minimum.
Gradient descent is designed for vectors – like the weights we’re learning – but it’s easier to think about the simpler case of a scalar.
Given the current (scalar) $w$, should we make it bigger or smaller?
Given the current (scalar) $w$, should we make it bigger or smaller?

Move $w$ in the reverse direction from the slope of the function.
The slope of loss at $w^1$ is negative, so we should move positive.
Loss

One step of gradient descent

\[ w^1 \rightarrow w^{\text{min}} \]

(goal)
The diagram represents a loss function with respect to the variable $w$. The function has a global minimum at $w_{\text{min}}$, and two local minima at $w^1$ and $w^2$. The point $w_{\text{min}}$ is marked as the goal. The loss function decreases as $w$ moves from $w^1$ towards $w_{\text{min}}$.
Loss

\( w^1 \quad w^2 \quad w^{\text{min}} \)

○ (goal)
Loss \( w \) 

\( w^1 \) \( w^2 \) \( w^{\text{min}} \) 

○ \((\text{goal})\)
$\text{loss}(w) = \min w^1_0, w^3_0, w^\text{min}$
Loss

$w^1$ $w^3$ $w^{\text{min}}$

○ (goal)
Loss \( w \)

\( w_0 \)

\( w_3 \)

\( w_{\text{min}} \)

\( (\text{goal}) \)
Loss

\( w^1 \) \( \overset{\circ}{w} \) \( w^{\min} \)

\( \text{goal} \)
Loss

\begin{align*}
  w_1 &< w_4 < w_{\text{min}}
\end{align*}

\text{\textbf{goal}}
Loss

$w$}

$w_{10}$

$w_{4}$

$w_{\text{min}}$

$O (\text{goal})$

$w$

$W^1 W^4 W^{\text{min}}$
Loss

\[ w_{\text{min}} \]

\[ w^1 \]

\[ w^5 \]

\( (\text{goal}) \)
Loss

\(w \rightarrow w_{min}\) (goal)

\(w^n\) \(w^5\) \(w_{min}\)

\(w\)
Loss

\[ w_{\min}(\text{goal}) \]

\[ w_1 \]

\[ (\text{goal}) \]
Loss

$w^1$  $w^{min}$  (goal)  $w$
To go further would be foolish.
The gradient of a function of many variables is a vector pointing in the direction of the greatest increase in a function.

The gradient descent algorithm works by finding the gradient of the loss function at the current point and moving in the opposite direction.
Gradient descent takes the slope
\[ \frac{d}{dw} L(f(x; w), y) \]
and multiplies it by a learning rate \( \eta \).

A higher learning rate means that we make bigger adjustments to the weights each time.

So, at time \( t \) we calculate the weights for time \( t + 1 \):

\[ w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y) \]
That was for a scalar, but we actually have $N$ parameters making up $\theta$, so we need to know where to move in an $N$-dimensional space!

The gradient is just such a vector; it expresses the directional components of the sharpest slope along each of the $N$ dimensions.
Imagine we just add one more parameter – now we have a scalar $w$ and a scalar $b$. 

Cost($w$, $b$)
We’ll have more dimensions, making it harder to visualize – but the idea will remain the same: Nudge all the parameters in the direction that minimizes the loss.
function STOCHASTIC GRADIENT DESCENT($L(\cdot)$, $f(\cdot)$, $x$, $y$) returns $\theta$

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

$\theta \leftarrow 0$

repeat til done

For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

1. Optional (for reporting): # How are we doing on this tuple?
   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 $y^{(i)}$?

2. $g \leftarrow \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move $\theta$ to maximize loss?

3. $\theta \leftarrow \theta - \eta \ g$ # Go the other way instead

return $\theta$
The learning rate $\eta$ is a hyperparameter.

Too high: The learner will take big steps and overshoot

Too low: The learner will take too long

Hyperparameters are chosen by the algorithm designer instead of being learned from the data like the regular parameters are.
Regularization
If a model perfectly matches the training data, that’s actually not good.

It will **overfit** the data, modeling noise:

A random word (maybe a typo) that perfectly predicts $y$ because it only occurs in one class will get a very high weight.

The resulting model will fail to generalize to a test set without this word
This movie drew me in, and it’ll do the same to you.

I can’t tell you how much I hated this movie. It sucked.

Useful (or, at least, harmless) features:

\[ x_1 = \text{this} \]
\[ x_2 = \text{movie} \]
\[ x_3 = \text{hated} \]
\[ x_4 = \text{drew me in} \]

Overfitting:

\[ x_5 = \text{the same to you} \]
\[ x_6 = \text{tell you how much} \]
To avoid overfitting, we use a regularization term, which penalizes large weights that might come from these spurious associations.

Details in the reading!
Multi-class regression
Often we need more than two classes:

- Positive / negative / neutral
- Noun / verb / adjective / etc.

In this case, we use **multinomial logistic regression**.
The probability of everything still needs to sum to 1, e.g.,

\[ P(\text{positive} \mid \text{doc}) + P(\text{negative} \mid \text{doc}) + P(\text{neutral} \mid \text{doc}) = 1 \]
For two classes we used the sigmoid function to squash the classifier’s values into probabilities.

For three or more classes, we use a generalization of the sigmoid called *softmax*, which takes a vector $\mathbf{z} = [z_1, z_2, \ldots, z_K]$ of $K$ arbitrary values and outputs a probability distribution.
softmax(\(z\)) = \left[ \frac{\exp(z_1)}{\sum_{i=1}^{K} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{K} \exp(z_i)}, \ldots, \frac{\exp(z_K)}{\sum_{i=1}^{K} \exp(z_i)} \right]

*The denominator is used to normalize all the values into probabilities.*
The input to softmax is still $\mathbf{w} \cdot \mathbf{x} + b$, but now we need separate weight vectors for each class.

Instead of

$$x_5 = \begin{cases} 1 & \text{if "!" \in \text{doc}} \\ 0 & \text{otherwise} \end{cases} \quad w_5 = 3.0$$

we'd have something like

<table>
<thead>
<tr>
<th>Feature</th>
<th>Definition</th>
<th>$w_{5,+}$</th>
<th>$w_{5,-}$</th>
<th>$w_{5,0}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_5(x)$</td>
<td>$\begin{cases} 1 &amp; \text{if &quot;!&quot; \in \text{doc}} \ 0 &amp; \text{otherwise} \end{cases}$</td>
<td>3.5</td>
<td>3.1</td>
<td>-5.3</td>
</tr>
</tbody>
</table>
Acknowledgments

This class incorporates material from:

Nancy Ide, Vassar College

Jurafsky & Martin, *Speech and Language Processing*, 3rd ed. draft