That’s a good model with 99% accuracy
Feedforward Networks
Neural Network Unit

This is not in your brain

Slide borrowed from Jurafsky & Martin Edition 3
Multinomial Logistic Regression as a 1-layer Network

Fully connected single layer network

\[ y = \text{softmax}(Wx + b) \]
Two-Layer Network with scalar output

Input layer

Output layer

Hidden units

$y = \sigma(z)$

$z = Uh$

$h = \sigma(Wx + b)$

Could be ReLU or tanh

Number

$z = Ux + b$
Two-Layer Network with softmax output

\[ y = \text{softmax}(z) \]
\[ z = Uh \]
\[ h = \sigma(Wx + b) \]

- **Input layer**
  - \( x_1 \) to \( x_n \) and +1

- **Hidden units (\( \sigma \) node)**

- **Output layer (\( \sigma \) node)**
Using feedforward networks
Use cases for feedforward networks

Let's reconsider text classification

(State-of-the-art systems use more powerful architectures)
Classification: Sentiment Analysis

We could do exactly what we did with logistic regression.

Input layer are binary features as before.

Output layer is 0 or 1.
## Sentiment Features

<table>
<thead>
<tr>
<th>Var</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>count(positive lexicon) $\in$ doc</td>
</tr>
<tr>
<td>$x_2$</td>
<td>count(negative lexicon) $\in$ doc</td>
</tr>
</tbody>
</table>
| $x_3$ | \[
\begin{cases}
1 & \text{if "no" } \in \text{ doc} \\
0 & \text{otherwise}
\end{cases}
\]
| $x_4$ | count(1st and 2nd pronouns $\in$ doc) |
| $x_5$ | \[
\begin{cases}
1 & \text{if "!" } \in \text{ doc} \\
0 & \text{otherwise}
\end{cases}
\]
| $x_6$ | log(word count of doc) |
Feedforward nets for simple classification

REGRESSION

NEURAL NETWORK
Even better: representation learning

The real power of deep learning comes from the ability to learn features from the data, instead of using hand-built human-engineered features for classification.
Neural Net Classification with embeddings as input features!

Word embeddings: amazing

 comes from a pretrained model

Assume: we map words to N-dimension word embeddings (vectors).
Neural Net Classification with embeddings as input features!

Assume: word embeddings

A great movie
Issue: texts come in different sizes

This assumes a fixed size length (3)!

![Diagram showing embeddings for different words with varying lengths.]

1. Make the input length the length of the longest review.
   - Shorter reviews will get padded with embeddings of all zeros.

2. Truncate reviews to a fixed length.

3. Create a sentence embedding.
Neural Net Classification with embeddings as input features!

\[
\text{Avg} \left( \begin{array}{c}
great \\
\text{movie} \\
\end{array} \right) = \text{sentence embedding}
\]

1 unit per dimension in embedding
YLLATALY Discussion
Drawing on the last two chapters of YLLATAILY, come up with some rules of thumb for identifying misleading AI headlines
1. Potential overfitting
2. Sensational language
3. AI as a term seems to imply consciousness
4. How can AI do it if humans can’t?
5. Technological solutions vs. structural change
6. Overly optimistic or dystopian
7. Not acknowledging error cases
8. Differences between how average readers vs. expert readers interpret reporting
Training Neural Networks
Intuition: training a 2-layer Network

Actual answer $y$

System output $\hat{y}$

Training instance $X_1, X_n$

Loss function $L(\hat{y}, y)$

Data flows forward

Loss flows backward

$w_{n_2}$

$w_{n_1}$
Intuition: Training a 2-layer network

For every training tuple \((x, y)\)

- Run \textit{forward} computation to find our estimate \(\hat{y}\)
- Run \textit{backward} computation to update weights:
  - For every output node
    - Compute loss \(L\) between \(\hat{y}\) and \(\hat{y}\)
    - For every weight from hidden layer to output layer
      - Update the weight
  - For every hidden node
    - Assess how much blame it deserves
    - For every weight \(w\) from input layer to the hidden layer
      - Update the weight proportionally to its blame.
Loss Function: a measure of how far off the current answer is from the right answer.

For binary logistic regression, we use cross entropy loss:

$$L_{CE}(\hat{y}, y) = - [y \log \sigma(wx+b) + (1-y) \log(1-\sigma(wx+b))]$$
Loss Function: a measure of how far off the current answer is from the right answer.

For multinomial classification, we use cross entropy loss:

\[ L_{CE}(\hat{y}, y) = - \log \frac{\exp(z_i)}{\sum_k \exp(z_k)} \]

Takeaway: cross-entropy loss = the log of the output probability of the correct class.
Gradient descent for weight updates

The derivative of the loss function with respect to weights tells us how to adjust the weights to make better predictions.

Derivative of the loss function:

\[ \frac{\partial L(f(x;w), y)}{\partial w} \]

We want to move the weights in the opposite direction of the gradient:

\[ w^{t+1} = w^t - \text{LR} \left( \frac{\partial L(f(x;w), y)}{\partial w} \right) \]

For logistic regression:

\[ \frac{\partial L_{CE}(w, b)}{\partial w_j} = (\hat{y} - y)x_j \]

\[ = \sigma((wx + b) - y)x_j \]
Where did that derivative come from?

Each node takes an upstream gradient, multiplies it by the local gradient (the gradient of its output with respect to its input), and uses the chain rule to compute a downstream gradient to be passed on to a prior node.

A node may have multiple local gradients if it has multiple inputs.