Logistic Regression Recap
Regression Equation

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

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

- **\(X\): feature vector** \(X = [0, 1, 2, 0, 0]\)
- **\(W\): weight vector** \(W = [0, 1, -1, 3, -3]\)
- **\(b\): bias term** \(b = 1\)
- **\(y = WX + b\)** (intercept)

\[
X \cdot W = 0 \cdot 0 + 1 \cdot 1 + 2 \cdot -1 + 0 \cdot 3 + 0 \cdot -3 = -1
\]
Features in logistic regression

For feature $x_i$, weight $w_i$ tells us how important is $x_i$

- $x_i =$ "review contains ‘awesome’": $w_i = +10$
- $x_j =$ "review contains ‘abysmal’": $w_j = -10$
- $x_k =$ “review contains ‘mediocre’": $w_k = -2$
Logistic Regression for one observation $x$

Input observation: vector $x = [x_1, x_2, \ldots, x_n]$

Weights: one per feature: $W = [w_1, w_2, \ldots, w_n]$

- Sometimes we call the weights $\theta = [\theta_1, \theta_2, \ldots, \theta_n]$

Output: a predicted class $\hat{y} \in \{0, 1\}$
The two phases of logistic regression

**Training**: we learn weights $w$ and $b$ using stochastic gradient descent and cross-entropy loss.

**Testing**: Given a test example $x$, we compute $p(y|x)$ using learned weights $w$ and $b$, and return whichever label ($y = 1$ or $y = 0$) is higher probability.
Multi-class Regression
Multinomial Logistic Regression

Often we need more than 2 classes

- Positive/negative/neutral
- Parts of speech (noun, verb, adjective, adverb, preposition, etc.)
- Classify emergency SMSs into different actionable classes

If >2 classes we use **multinomial logistic regression**

= Softmax regression
= Multinomial logit
= Maximum entropy modeling or MaxEnt

So "logistic regression" means binary (2 classes)
Multinomial Logistic Regression

The probability of everything must still sum to 1

\[ P(\text{positive} \mid \text{doc}) + P(\text{negative} \mid \text{doc}) + P(\text{neutral} \mid \text{doc}) = 1 \]

Need a generalization of the sigmoid called \textit{softmax}

- Takes a vector \( z = [z_1, z_2, ..., z_k] \) of \( k \) arbitrary values
- Outputs a probability distribution
The softmax function

Turns a vector $z = [z_1, z_2, \ldots, z_k]$ of $k$ arbitrary values into probabilities

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \quad 1 \leq i \leq k$$

The denominator $\sum_{i=1}^{k} e^{z_i}$ is used to normalize all the values into probabilities.

$$\text{softmax}(z) = \left[ \frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$
The softmax function

Turns a vector $z = [z_1, z_2, ..., z_k]$ of $k$ arbitrary values into probabilities:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

$$\text{softmax}(z) = \left[ \frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, ..., \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

$$[0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$
Softmax in multinomial logistic regression

\[ p(y = c | x) = \frac{\exp (w_c \cdot x + b_c)}{\sum_{j=1}^{K} \exp (w_j \cdot x + b_j)} = \text{softmax} (wx + b) \]

Input is still the dot product between weight vector \( w \) and input vector \( x \), but now we need separate weight vectors for each of the \( K \) classes.
Features in binary versus multinomial logistic regression

Binary: positive weight

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

Multinominal: separate weights for each class:

<table>
<thead>
<tr>
<th>Feature ( f_5(x) )</th>
<th>Definition</th>
<th>( w_{5,+} )</th>
<th>( w_{5,-} )</th>
<th>( w_{5,0} )</th>
</tr>
</thead>
</table>
| \( f_5(x) \)        | \( \begin{cases} 
1 & \text{if } "!" \in \text{doc} \\
0 & \text{otherwise} 
\end{cases} \) | 3.5           | 3.1           | -5.3          |
Computing with Probabilities
So far we've been working with relatively small sample spaces. This means our probabilities have been decently large.

As we go on in this class, our sample spaces are going to get much larger. We want to be able to reason about the probabilities of things like:

- All words in English
- All pixels in a photo
- All possible game states for Pacman
Problem: when our probabilities get really really small, programming languages start making mistakes.

There is a **bound on precision** in numerical computing.

This is because of the limitations on space allocation for (floating point) numbers.
Intuition: we care about how big probabilities are relative to the other probabilities in our distribution, not the actual value.

Probabilities:
- $p(\text{heart}) = 0.1$
- $p(\text{rainbow}) = 0.2$
- $p(\text{letter}) = 0.7$

Interpretation: a letter is 7 times more likely than a heart!
Solution: make the numbers bigger

✧ Intuition: we care about how big probabilities are relative to the other probabilities in our distribution, not the actual value.

Probabilities:
\[
\begin{align*}
p(\text{heart}) &= 0.1 \times 100 \\
p(\text{rainbow}) &= 0.2 \times 200 \\
p(\text{letter}) &= 0.7 \times 700 
\end{align*}
\]

What if we just multiply all our probs by 100?
This preserves the ratio.
Solution: make the numbers bigger

✧ What if we just multiply all our probs by 100? This preserves the ratio.

<table>
<thead>
<tr>
<th>Probabilities:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p(heart) = 0.1 100</td>
<td></td>
</tr>
<tr>
<td>p(rainbow) = 0.2 200</td>
<td></td>
</tr>
<tr>
<td>p(letter) = 0.7 700</td>
<td></td>
</tr>
</tbody>
</table>

However, if we want to recover the probabilities later, we'll need to **renormalize** them. This means **remembering that we multiplied by 100**.
Solution: log-transform the numbers

- Instead, we use a log transformation. This changes the range from [0,1] to [-∞, 0].

Log base doesn't matter much but we usually use natural log (base e):

Probabilities:
- p(heart) = 0.1 -2.3
- p(rainbow) = 0.2 -1.6
- p(letter) = 0.7 -0.36

www.desmos.com/calculator/aczt76asao
Feature Representations
We can build features for logistic regression for any classification task: period disambiguation

This ends in a period.

The house at 465 Main St. is new.

\[
\begin{align*}
x_1 &= \begin{cases} 
1 & \text{if } \text{Case}(w_i) = \text{Lower} \\
0 & \text{otherwise}
\end{cases} \\
x_2 &= \begin{cases} 
1 & \text{if } w_i \text{ 2 AcronymDict} \\
0 & \text{otherwise}
\end{cases} \\
x_3 &= \begin{cases} 
1 & \text{if } w_i = \text{St.} \& \text{Case}(w_{i-1}) = \text{Cap} \\
0 & \text{otherwise}
\end{cases}
\]
We represent text using word vectors.

Idea: a word meaning is based on its **distance** from other word meanings.

Each word = a vector (not just "good" or "w_{45}")

Similar words are "**nearby in semantic space**"

We build this space automatically by seeing which words are **nearby in text**
What does recent English borrowing *ongchoi* mean?

Suppose you see these sentences:

- Ong choi is delicious *sautéed with garlic*.
- Ong choi is superb *over rice*
- Ong choi *leaves* with salty sauces

And you've also seen these:

- ...spinach *sautéed with garlic over rice*
- Chard stems and *leaves* are *delicious*
- Collard greens and other *salty* leafy greens

Conclusion:

- Ongchoi is a leafy green like spinach, chard, or collard greens
- We could conclude this based on words like "leaves" and "delicious" and "sautéed"
What kinds of contexts does the word occur in?

<table>
<thead>
<tr>
<th></th>
<th>garlic</th>
<th>rice</th>
<th>green</th>
<th>leaves</th>
</tr>
</thead>
<tbody>
<tr>
<td>spinach</td>
<td>75</td>
<td>10</td>
<td>339</td>
<td>290</td>
</tr>
<tr>
<td>salmon</td>
<td>81</td>
<td>102</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>sunflower</td>
<td>4</td>
<td>2</td>
<td>279</td>
<td>301</td>
</tr>
<tr>
<td>dinosaur</td>
<td>2</td>
<td>1</td>
<td>418</td>
<td>113</td>
</tr>
<tr>
<td>because</td>
<td>13</td>
<td>17</td>
<td>21</td>
<td>44</td>
</tr>
<tr>
<td>ongchoi</td>
<td>91</td>
<td>113</td>
<td>381</td>
<td>287</td>
</tr>
</tbody>
</table>
Ongchoi: *Ipomoea aquatica* "Water Spinach"

空心菜
kangkong
rau muống
...

Yamaguchi, Wikimedia Commons, public domain
How to represent word meaning numerically?

Idea: represent each word using a vector. These vectors are called "embeddings" because they are embedded into a space.

The standard way to represent meaning in NLP

Every modern NLP algorithm uses embeddings as the representation of word meaning

Fine-grained model of meaning for similarity
What kinds of contexts does the word occur in?

<table>
<thead>
<tr>
<th></th>
<th>garlic</th>
<th>rice</th>
<th>green</th>
<th>leaves</th>
</tr>
</thead>
<tbody>
<tr>
<td>spinach</td>
<td>75</td>
<td>10</td>
<td>339</td>
<td>290</td>
</tr>
<tr>
<td>salmon</td>
<td>81</td>
<td>102</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>sunflower</td>
<td>4</td>
<td>2</td>
<td>279</td>
<td>301</td>
</tr>
<tr>
<td>dinosaur</td>
<td>2</td>
<td>1</td>
<td>418</td>
<td>113</td>
</tr>
<tr>
<td>because</td>
<td>13</td>
<td>17</td>
<td>21</td>
<td>44</td>
</tr>
<tr>
<td>ongchoi</td>
<td>91</td>
<td>113</td>
<td>381</td>
<td>287</td>
</tr>
</tbody>
</table>
Intuition: why embeddings?

Consider sentiment analysis:

- With **words**, a feature is a word identity
  - Feature 5: 'The previous word was "terrible"'
  - requires **exact same word** to be in training and test

- With **embeddings**:
  - Feature is a word vector
  - 'The previous word was vector [35,22,17…]
  - Now in the test set we might see a similar vector [34,21,14]
  - We can generalize to **similar but unseen** words!!!
For computer vision applications, we need a way of describing images. We represent images as matrices of pixel values.

Grayscale images can be represented with a single matrix.

Color images need to be represented with a 3D tensor (3rd dimension encodes color channel).

Why matrices for images and vectors for text? Language is sequential, which makes it more useful to concatenate vectors lengthwise rather than stack them.
grayscale images are matrices

what range of values can each pixel take?

Slides adapted from Mohit Iyyer
color images are tensors

Channels are usually RGB: Red, Green, and Blue
Other color spaces: HSV, HSL, LUV, XYZ, Lab, CMYK, etc
Logistic Regression Example: Pet Picture Classification
Goal: Classify Pet Pictures

- Dataset: cat + dog pictures
- Goal: classify a picture as either a cat or a dog
- Input: grayscale images
Building a Model

We’ll build our model using a machine learning library called **Tensorflow**.

Tensorflow is a Python library, but most functions are implemented in C (so they are fast!).

Tensorflow provides useful abstractions for models:

- **tensor**: n-dimensional container for data
- **layer**: apply functions to an input tensor of n dimensions to produce an output tensor of m dimensions.
- **model**: consist of layers connected together
Example Data
Splitting Our Data

Generate a Dataset

In [4]:

```python
image_size = (180, 180)
batch_size = 32

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "PetImages",
    color_mode='grayscale',
    validation_split=0.2,
    subset="training",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "PetImages",
    color_mode='grayscale',
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
```

Found 23410 files belonging to 2 classes.
Using 18728 files for training.
Found 23410 files belonging to 2 classes.
Using 4682 files for validation.
Creating Our Model Architecture

```python
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    x = layers.Flatten()(inputs)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)

model = make_model(input_shape=image_size, num_classes=2)
k keras.utils.plot_model(model, show_shapes=True)
```
Creating Our Model Architecture
Train the model

ePOCHS = 50

callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
]
model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
model.fit(
    train_ds, epochs=epochs, callbacks=callbacks, validation_data=val_ds,
)
Avoiding Harms in Classification

Slides borrowed from Jurafsky & Martin Edition 3
Harms in sentiment classifiers

Kiritchenko and Mohammad (2018) found that most sentiment classifiers assign lower sentiment and more negative emotion to sentences with African American names in them.

This perpetuates negative stereotypes that associate African Americans with negative emotions.
Harms in toxicity classification

Toxicity detection is the task of detecting hate speech, abuse, harassment, or other kinds of toxic language.

But some toxicity classifiers incorrectly flag as being toxic sentences that are non-toxic but simply mention identities like blind people, women, or gay people.

This makes it harder for members of these communities to connect, organize, and report mistreatment.
What causes these harms?

Can be caused by:

◦ Biases in training data (machine learning systems can amplify biases in their training data)
◦ Imbalance/lack of representation in training data
◦ Annotator bias
◦ Problems in the resources used (like lexicons)
◦ Problems in model architecture (like how the model's goal is defined)

Mitigation of these harms is an open research area