



# Linear and Logistic Regression

CS 780/880 Natural Language Processing Lecture 11

Samuel Carton, University of New Hampshire

# Last lecture

---

## Key concepts:

- Benchmark datasets
  - SuperGLUE
- Various NLP tasks
  - Inference detection
  - Translation
    - BLEU score
  - Common-sense reasoning
  - Reading comprehension

**Key insight:** Lots of tasks, not so many basic modeling approaches

- Language modeling objective gets you a long way if the model is good enough
- GPT-3 is a very good language model



# Linear regression

---



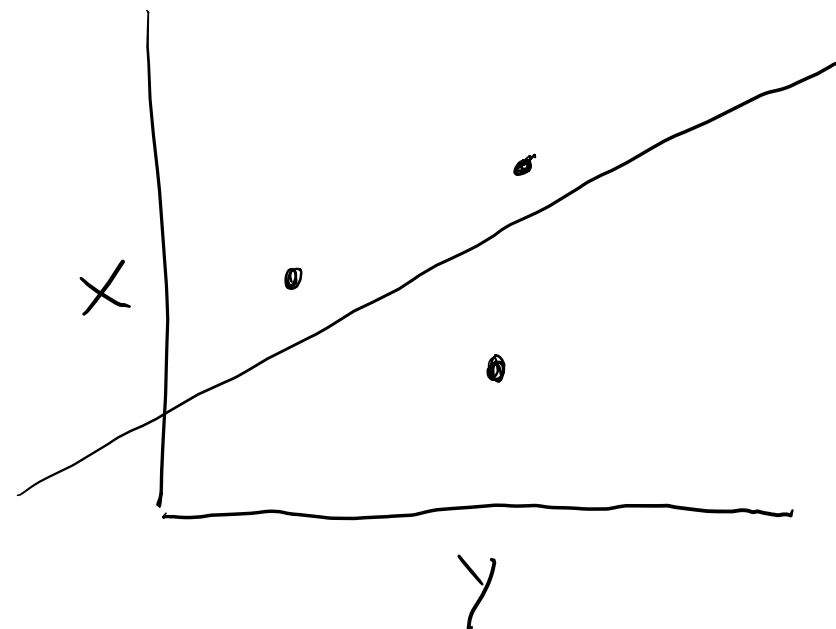
# Linear regression

---

**Basic idea:** given some points in N-dimensional space, find a “line of best fit” that is as close as possible to those points.

When the points are text:

- $N$  = vocabulary size
- Examples:
  - Grading essays 0-100
  - Scoring text complexity



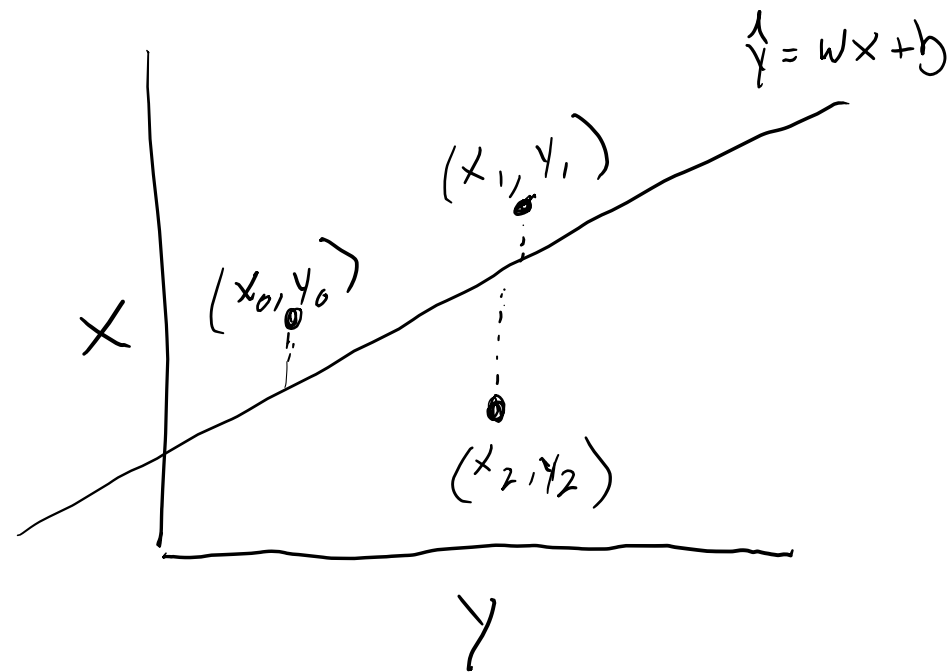
# Linear regression

Mathematically, what we're trying to do is figure out some function:

$$\hat{y} = Wx + b$$

...where  $W$  and  $b$  are values such that  $\hat{y}$  tends to be close to  $y$  for any given  $x$ .

Very common in ML to refer to predicted output as  $\hat{y}$  and true output as  $y$ .



# Loss function

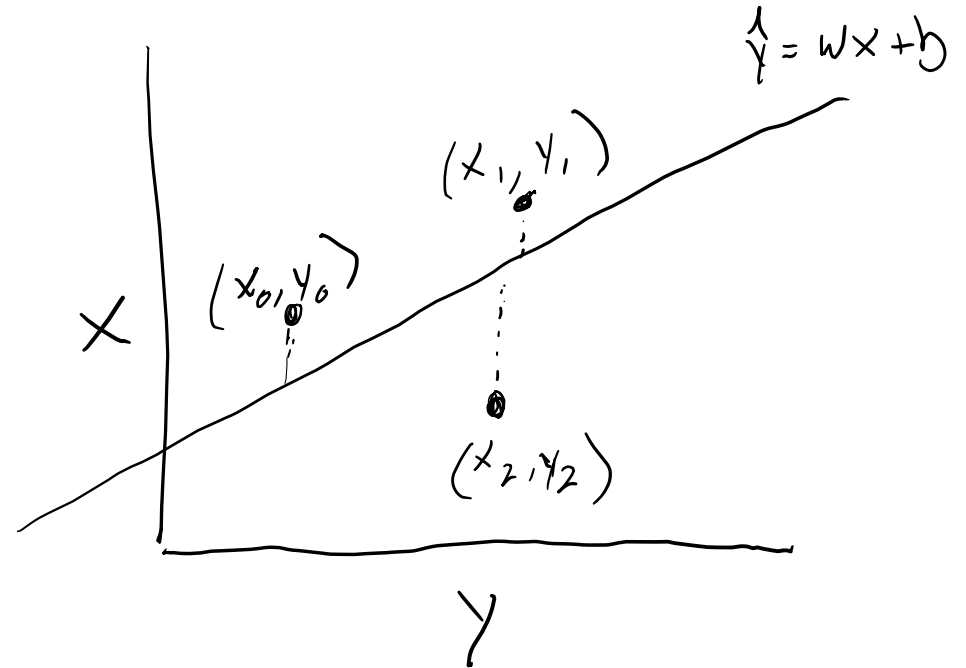
We generally articulate this goal with a **loss function** that describes the value we're trying to minimize with our choice of  $W$  and  $b$ .

AKA “Objective function”

It's very typical to minimize **squared loss** between expected and true output:  $\sum_i (\hat{y}_i - y_i)^2$

That would give us a loss function of:

$$\begin{aligned} L(W, b) &= \sum_i (\hat{y}_i - y_i)^2 \\ &= (\hat{y}_0 - y_0)^2 + (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \\ &= (Wx_0 + b - y_0)^2 + (Wx_1 + b - y_1)^2 + (Wx_2 + b - y_2)^2 \end{aligned}$$

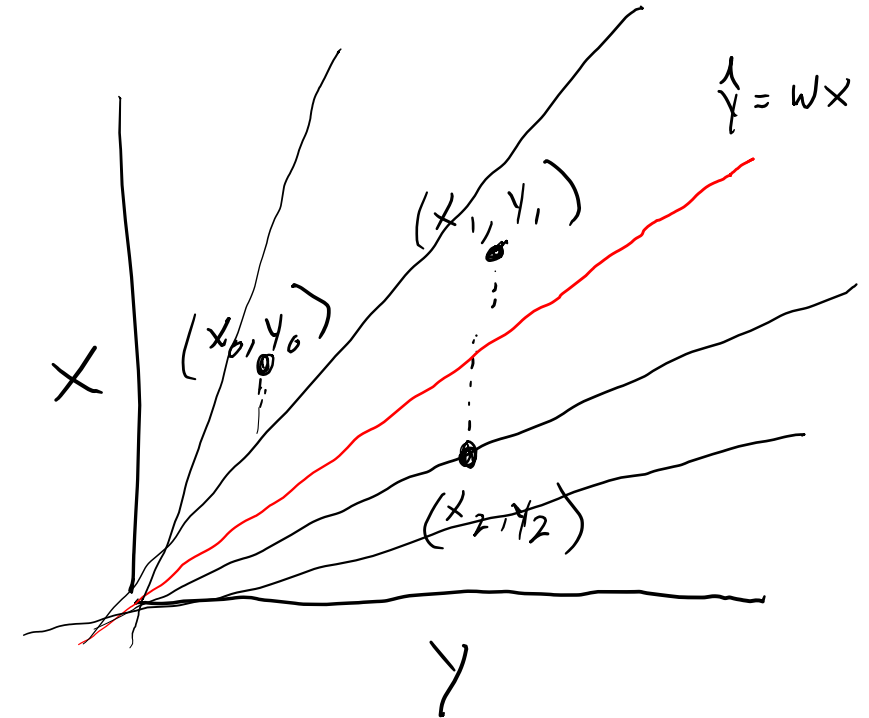


# Simple example

To show how we can solve this, I'll use a simple example with **no intercept** ( $b$ )

So the loss function is:

$$\begin{aligned}L(W, b) &= \sum_i (\hat{y}_i - y_i)^2 \\&= (\hat{y}_0 - y_0)^2 + (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \\&= (Wx_0 - y_0)^2 + (Wx_1 - y_1)^2 + (Wx_2 - y_2)^2 \\&= (W^2x_0^2 - 2Wx_0y_0 + y_0^2) \\&\quad + (W^2x_1^2 - 2Wx_1y_1 + y_1^2) \\&\quad + (W^2x_2^2 - 2Wx_2y_2 + y_2^2) \\&= W^2(x_0^2 + x_1^2 + x_2^2) - 2W(x_0y_0 + x_1y_1 + x_2y_2) + (y_0^2 + y_1^2 + y_2^2)\end{aligned}$$



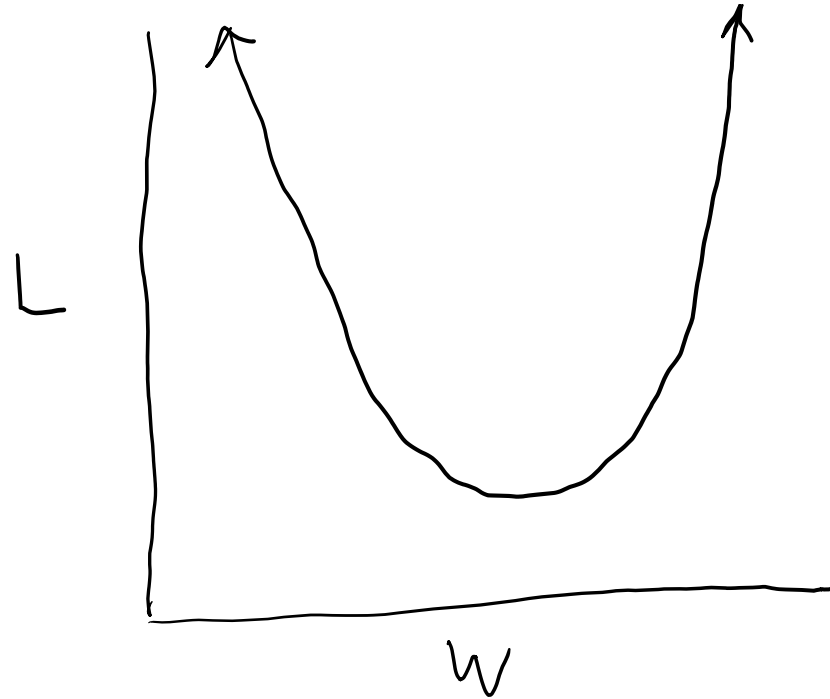
# Simple example

---

If the loss function for  $W$  is this:

$$L(W) = W^2(x_0^2 + x_1^2 + x_2^2) - 2W(x_0y_0 + x_1y_1 + x_2y_2) + (y_0^2 + y_1^2 + y_2^2)$$

Then the graph of  $L$  as a function of  $W$  looks like this:

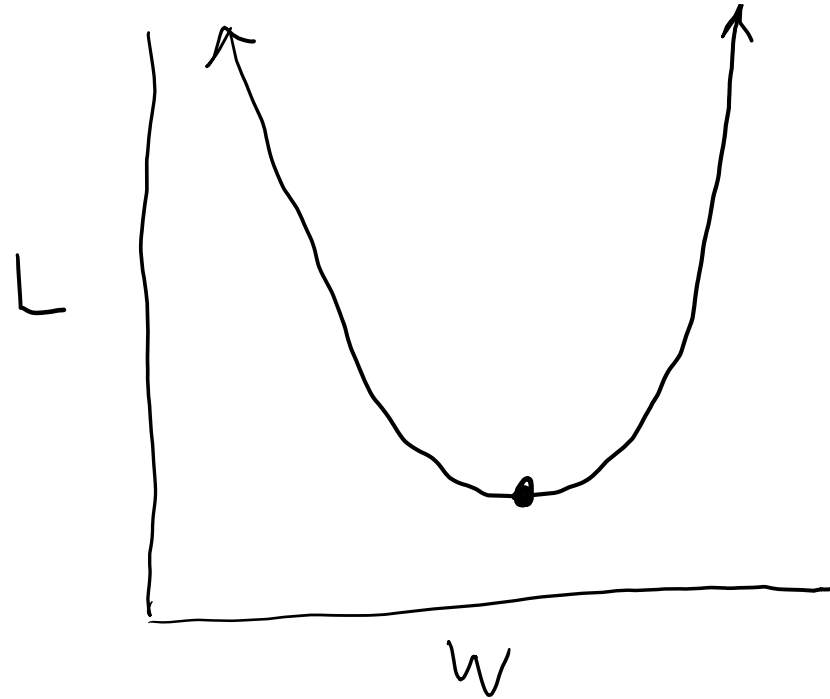




# Simple example

---

It's pretty obvious what value of  $W$  will minimize the loss here.



# Simple example

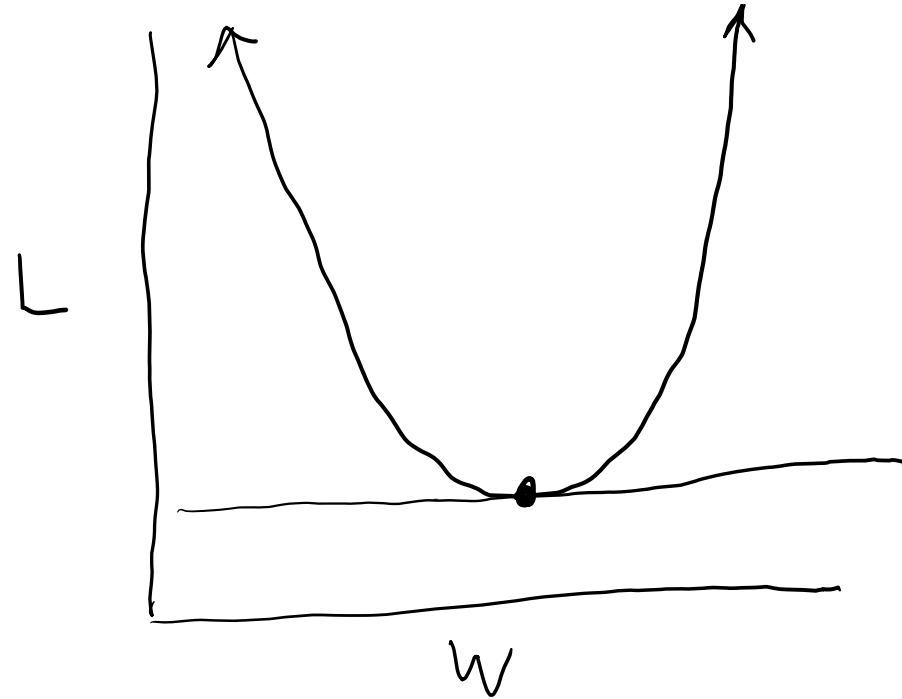
---

What may be less obvious is that this also happens to be the point where the derivative of  $L$  with respect to  $W$  is 0.

$$\frac{dL}{dW} = 0$$

In some sense it is the bottom of a pit.

**Gradient descent** is the process of gradually following the slope of the function down to these pit-bottoms



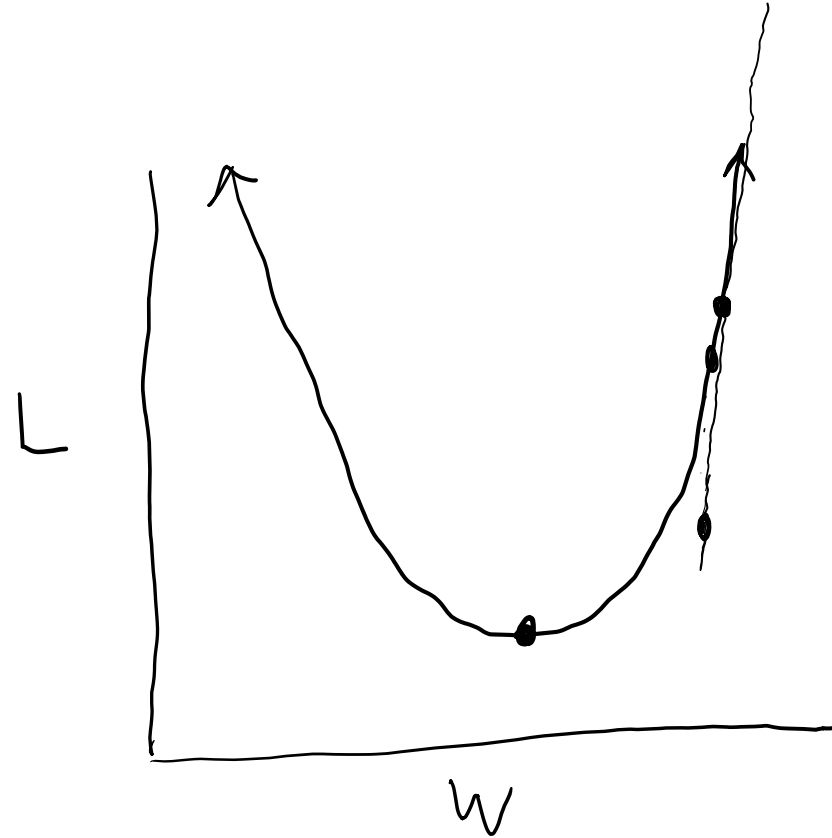
# Simple example

---

In the most simple case, we pick a random point on the function and find the slope (derivative)

Then we move some incremental distance in the direction that **reduces** the value of  $L$  (left in this case)

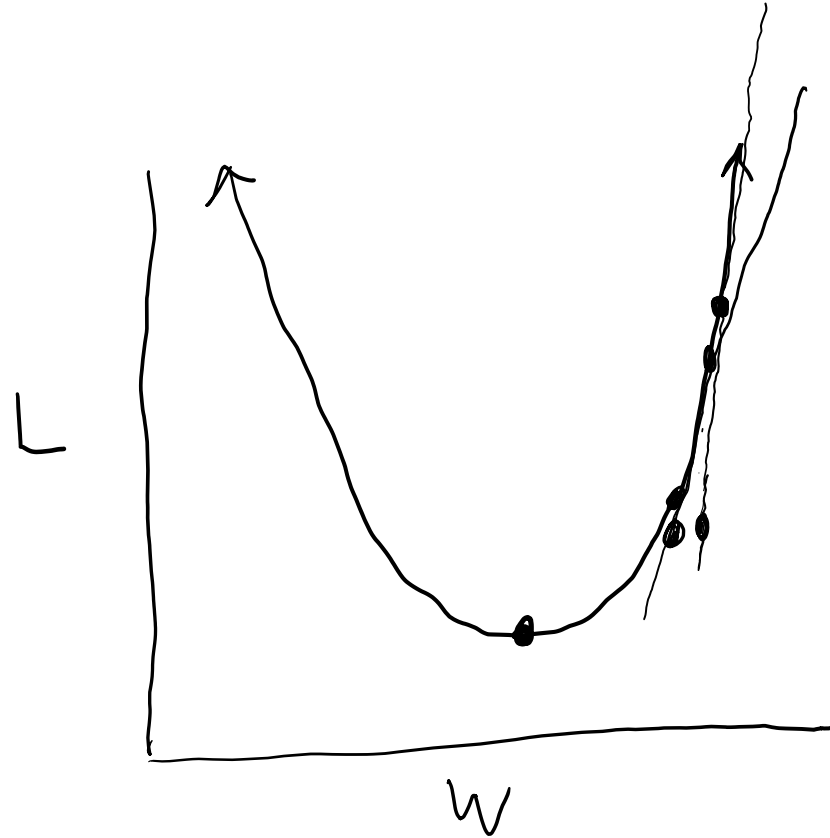
This increment that we move each step is called the **learning rate**



# Simple example

---

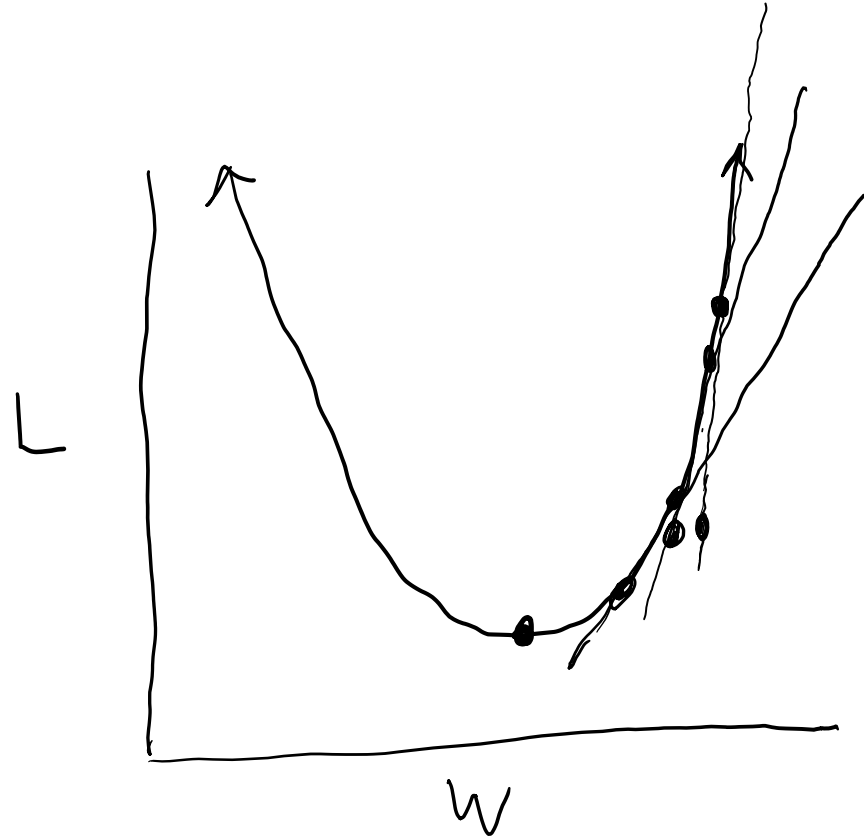
Then we calculate the slope again at this new point and move one increment in the reducing-L direction (still left).



# Simple example

---

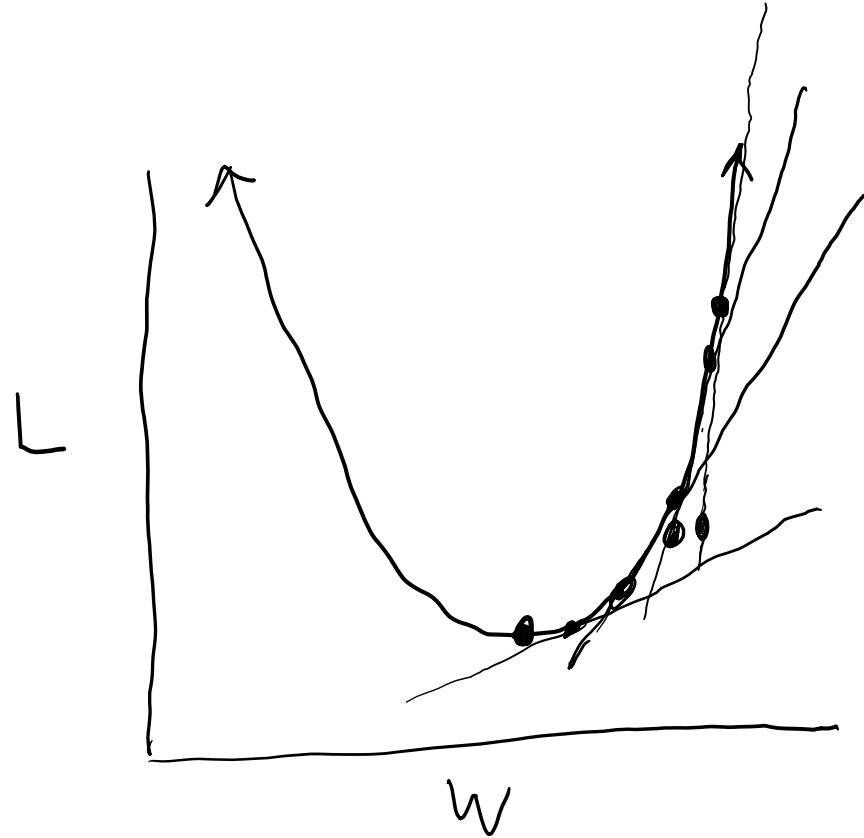
And we keep doing that...



# Simple example

---

And keep doing that...

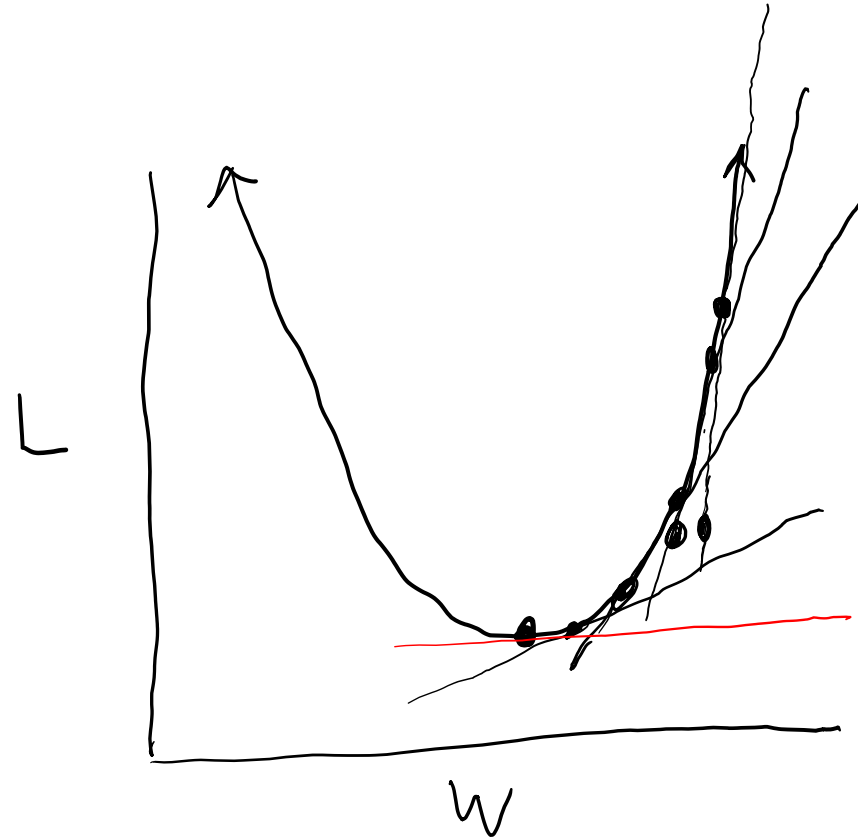


# Simple example

Until we hit a point on  $W$  where the slope seems to have levelled out

$$\text{That is, } \frac{dL}{dW} = 0$$

And we conclude that we've found the value of  $W$  that minimizes  $L$

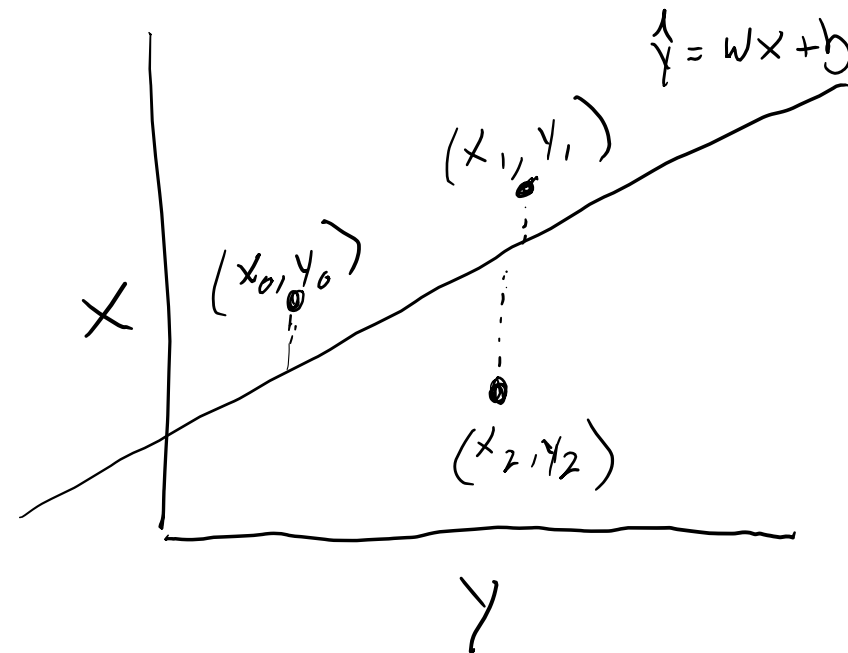


# Adding back the intercept

So what if our function **does** have an intercept?

$$\begin{aligned}L(W, b) &= \sum_i (\hat{y}_i - y_i)^2 \\&= (\hat{y}_0 - y_0)^2 + (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \\&= (Wx_0 + b - y_0)^2 + (Wx_1 + b - y_1)^2 + (Wx_2 + b - y_2)^2 \\&= (W^2x_0^2 - 2Wx_0y_0 + 2Wbx_0 - b2y_0 + y_0^2 + b^2) \\&\quad + (W^2x_1^2 - 2Wx_1y_1 + 2Wbx_1 - b2y_1 + y_1^2 + b^2) \\&\quad + (W^2x_2^2 - 2Wx_2y_2 + 2Wbx_2 - b2y_2 + y_2^2 + b^2)\end{aligned}$$

$$\begin{aligned}&= W^2(x_0^2 + x_1^2 + x_2^2) \\&\quad - 2W(x_0y_0 + x_1y_1 + x_2y_2) \\&\quad + 2Wb(x_0 + x_1 + x_2) \\&\quad - 2b(y_0 + y_1 + y_2) \\&\quad + (y_0^2 + y_1^2 + y_2^2) \\&\quad + 3b^2\end{aligned}$$



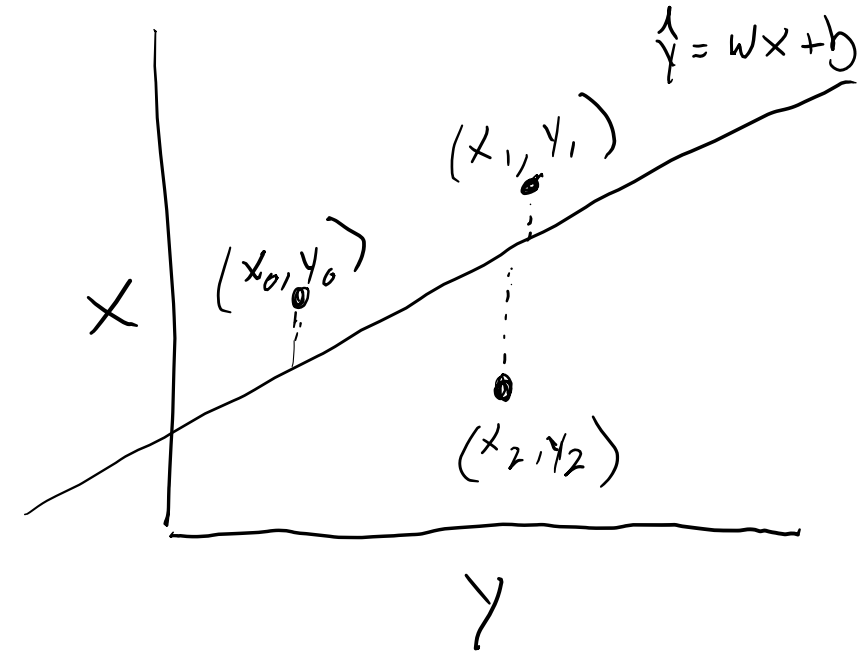


# Adding back the intercept

So what if our function **does** have an intercept?

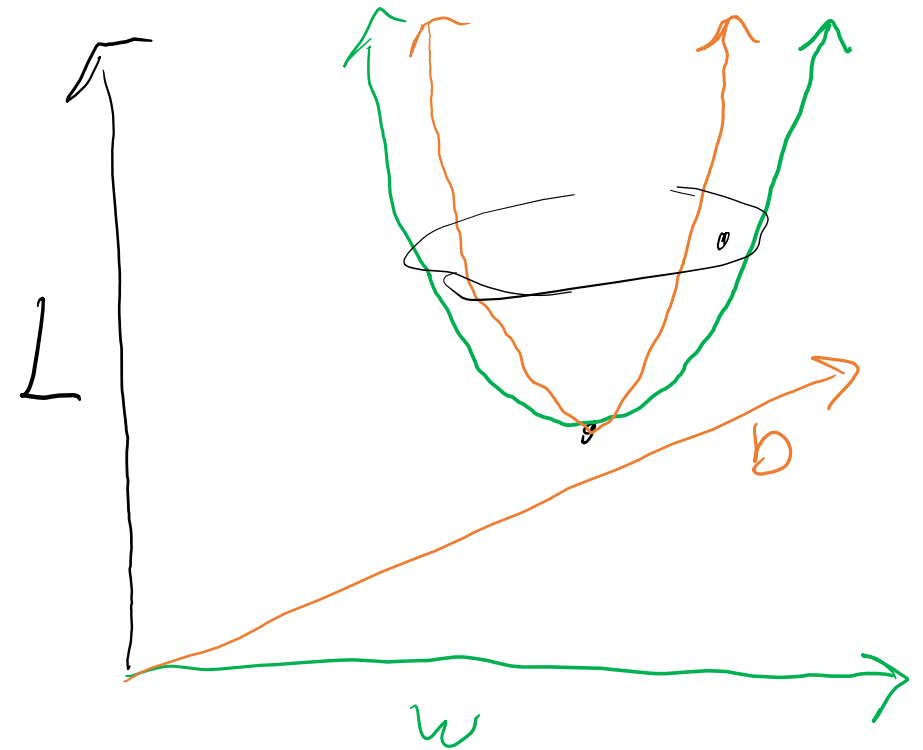
$$\begin{aligned}L(W, b) = & W^2(x_0^2 + x_1^2 + x_2^2) \\ & -2W(x_0y_0 + x_1y_1 + x_2y_2) \\ & +2Wb(x_0 + x_1 + x_2) \\ & -2b(y_0 + y_1 + y_2) \\ & +(y_0^2 + y_1^2 + y_2^2) \\ & +3b^2\end{aligned}$$

More complicated, but the key thing is that  $L$  is still just a quadratic function of  $W$  and  $b$



# Adding back the intercept

So the loss becomes a 2-dimensional function, and we're trying to find a value for  $W$  and a value for  $b$ , which, taken together, minimize  $L$



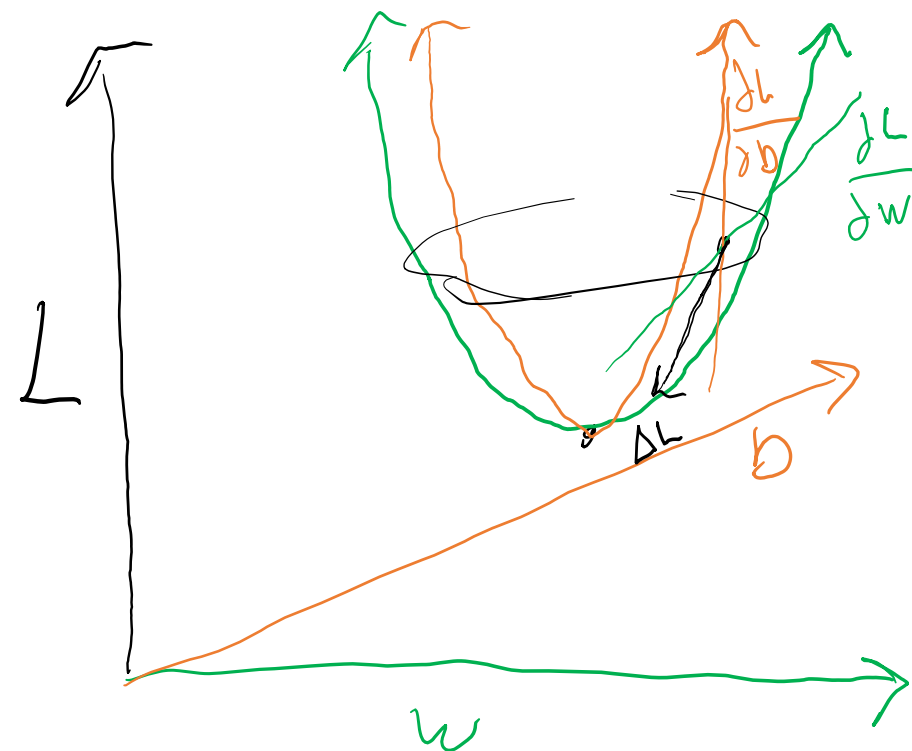
# Adding back the intercept

We can still use gradient descent though!

Only now, instead of following the **derivative**  $\frac{dL}{dW}$  of  $L$  with respect to  $W$  to the bottom...

We now follow a **vector** composed of the partial derivatives of  $L$  with respect to  $W$  and  $b$ :  $\left(\frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}\right)$

We call this vector the **gradient** of  $L$  with respect to  $W$  and  $b$ , and usually denote it with the  $\Delta$  symbol, e.g.  $\Delta_L(W,b)$



# Gradient descent

---

Gradient descent is the method by which all neural nets are trained.

It works\* in any situation where it is possible to calculate the gradient of the loss with respect to the model parameters

- $\hat{y} = Wx + b$  has two parameters:  $W$  and  $b$
- ChatGPT has 175 billion parameters

\*: it works more or less well depending on the shape of the loss function.

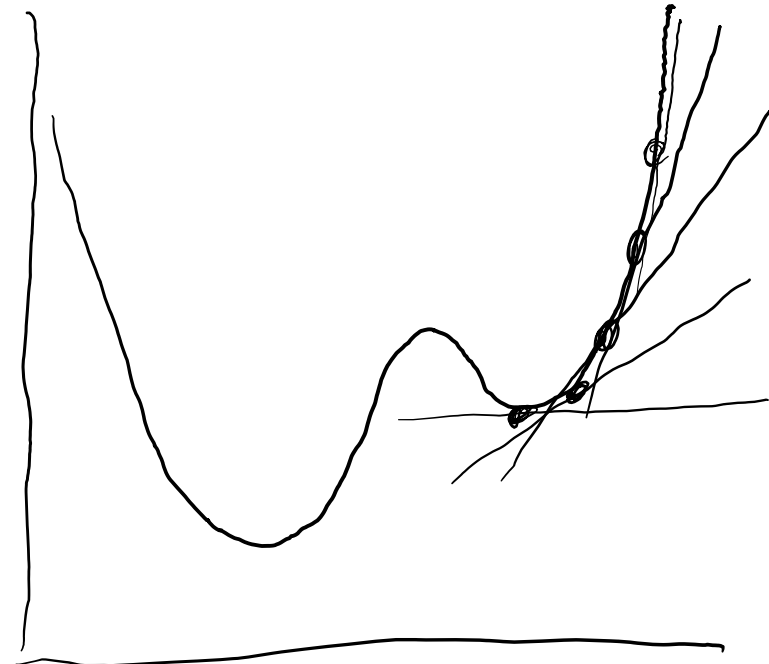
- If the function is a nice **convex** “bucket”, then it will **always** find the global minimum.
- But this is not usually true



# Local minima

---

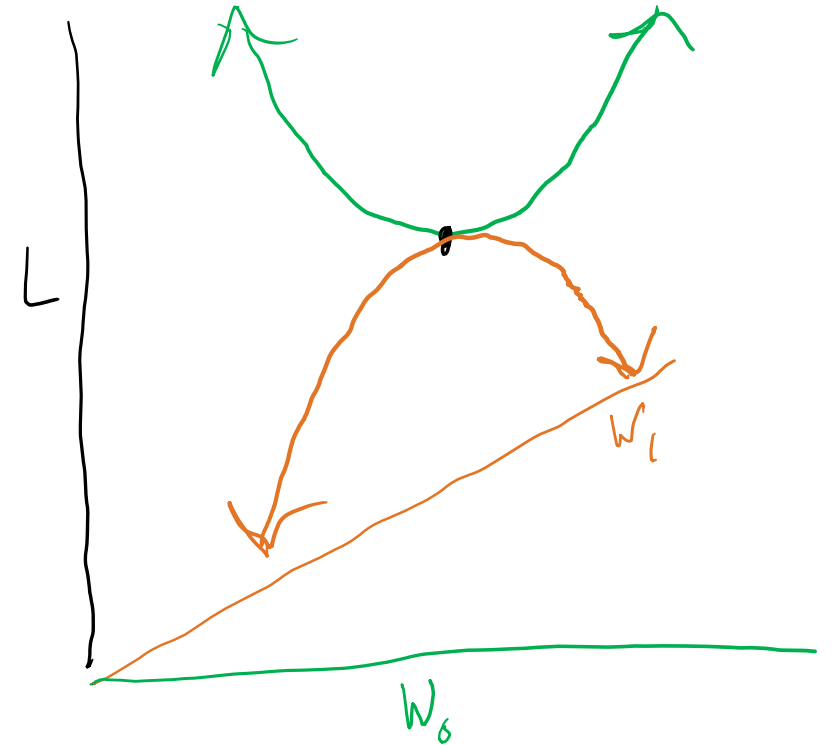
One issue in gradient descent is “local minima” which are false “dips” the gradient descent can get stuck in.



# Saddle points

Another issue is “saddle points” which represent a minimum for one parameter but a maximum for a different one.

The gradient for both can be zero here... but it's not necessarily a very good solution.



# Advanced gradient descent

---

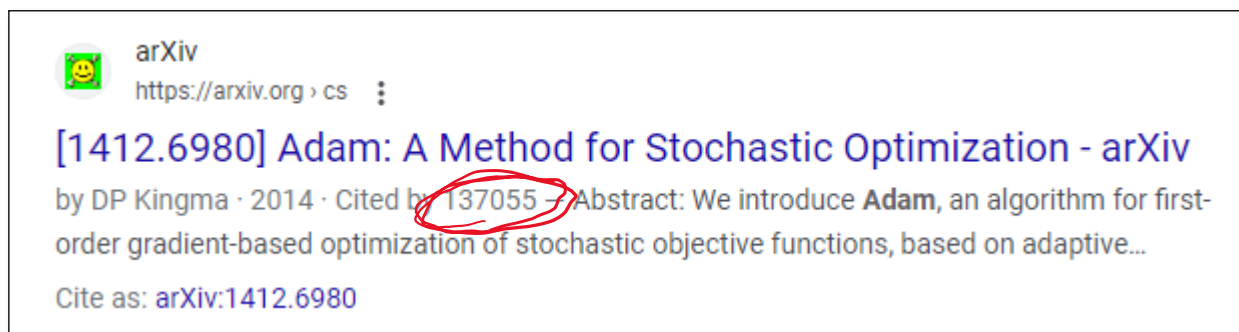
Advanced gradient descent algorithms have various tricks to help them avoid local minima and other issues

Most popular: **Adam**

- Uses “momentum” to learn adaptive learning rate for each parameter
- Generally the default choice for optimizing any arbitrary neural net

Long story short: just use Adam for everything

- Unless you have a good reason not to



The screenshot shows an arXiv entry for the paper 'Adam: A Method for Stochastic Optimization' by DP Kingma, 2014. The entry includes the arXiv ID [1412.6980], the title, the author, the year, the number of citations (137055, circled in red), the abstract, and the citation key arXiv:1412.6980.

arXiv  
https://arxiv.org › cs

[1412.6980] Adam: A Method for Stochastic Optimization - arXiv  
by DP Kingma · 2014 · Cited by 137055 - Abstract: We introduce **Adam**, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive...  
Cite as: arXiv:1412.6980



# Logistic regression

---





# Logistic regression

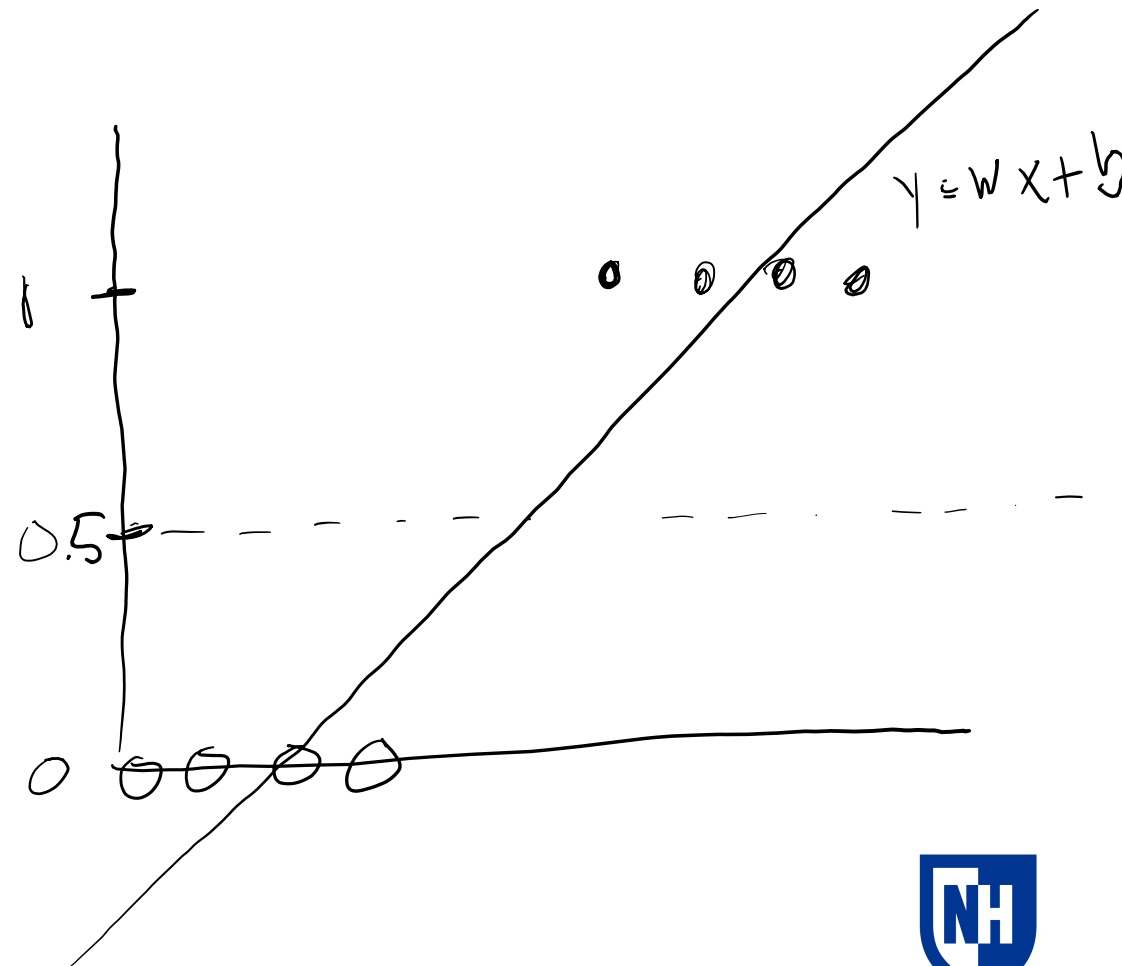
Generally in NLP we're more interested in **classification** than regression

- Mapping input  $x$ 's to a discrete category rather than a continuous value

Linear regression not ideal for this

We can do it hackily with a threshold on the predicted value

- But this has problems



# Logistic function

To solve this problem, we are going to wrap our original function, which we will call  $f$ , in a **logistic function**

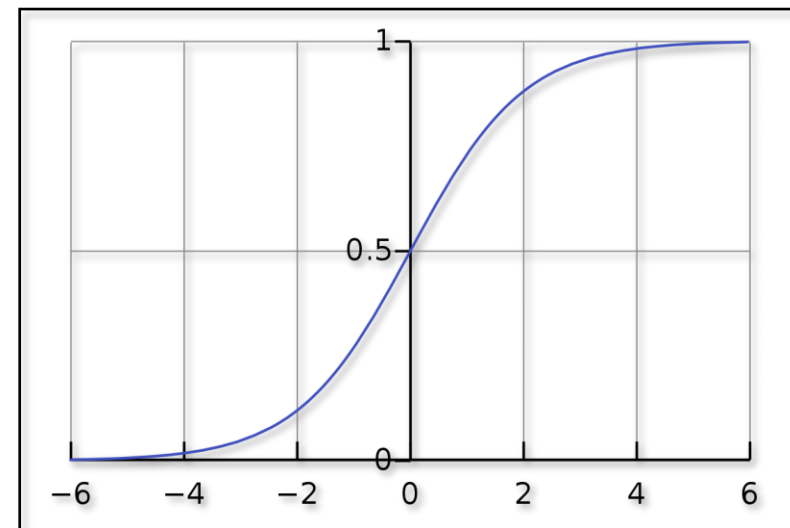
$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \sigma(f(x)) = \frac{1}{1 + e^{-(Wx+b)}}$$

One nice thing about it is that it is easy to differentiate because of the property that:

$$\frac{d}{dx} \sigma(x) = \sigma(x)(1 - \sigma(x))$$

So if we call our original function  $f$ :

$$\frac{d}{dx} \sigma(f(x)) = \sigma(f(x))(1 - \sigma(f(x)))f'(x) = W\sigma(Wx + b)(1 - \sigma(Wx + b))$$



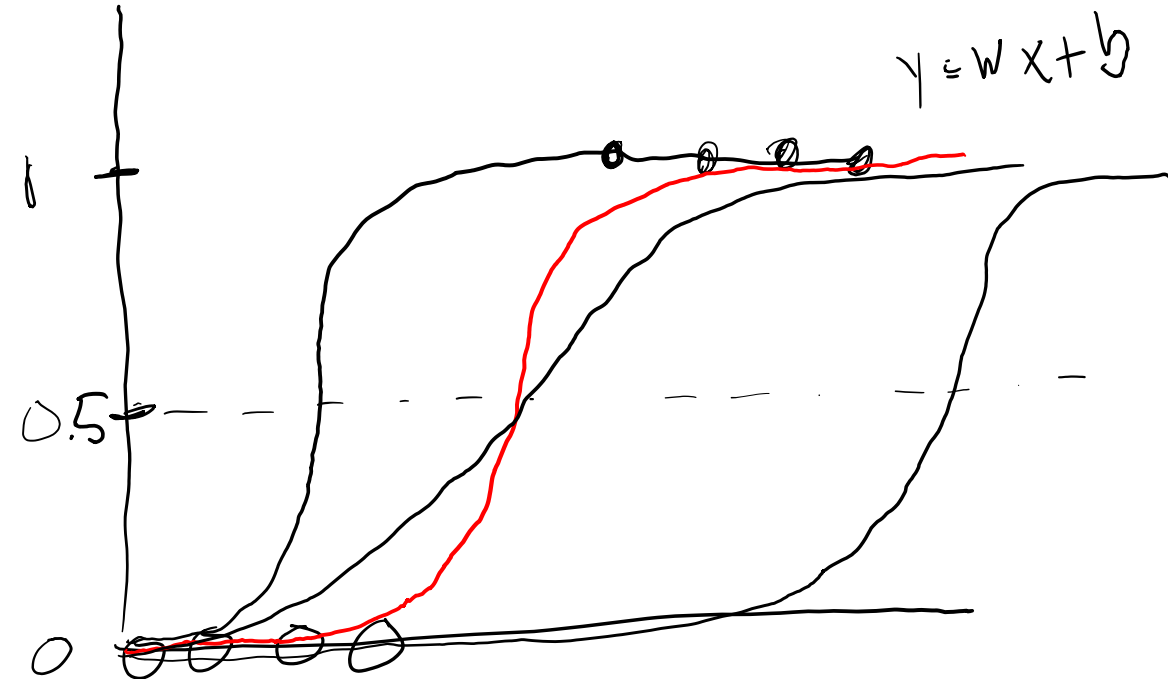
[https://en.wikipedia.org/wiki/Logistic\\_function](https://en.wikipedia.org/wiki/Logistic_function)



# Logistic regression

So now instead of trying to fit a straight line to the data, we're trying to choose  $W$  and  $b$  to fit this S-shaped logistic curve to the data

Different choices for  $W$  and  $b$  change how steep the curve is and where it is centered.



# Gradient descent for logistic regression

---

I won't do the full derivation, but:

- The function we're trying to fit is differentiable
- Which means we can create a differentiable loss function
- Which means we can do gradient descent!

**However:** mean squared error is not always convex for logistic regression

So we typically use **cross-entropy** loss as our objective:

$$L(y, \hat{y}) = \sum_c y_c \log(\hat{y}_c)$$

- Sum across possible classes of true value for that class multiplied by predicted log-probability of that class

More detailed discussion available in Speech and Language Processing chapter 5: <https://web.stanford.edu/~jurafsky/slp3/5.pdf>

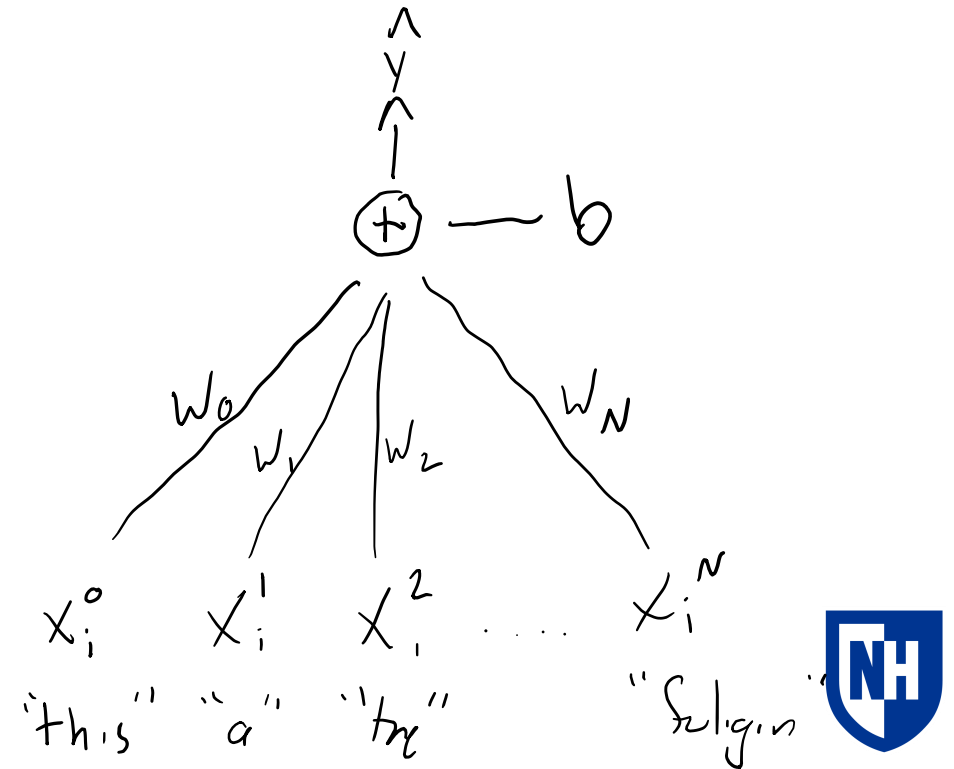


# Visualizing linear regression

You can think of linear regression as a vector operation between matrices of x's, W's and y's

$$\begin{bmatrix} x_0^0 & x_0^1 & x_0^2 & \dots & x_0^N \\ x_1^0 & x_1^1 & x_1^2 & \dots & x_1^N \\ \dots & \dots & \dots & \dots & \dots \\ x_m^0 & x_m^1 & x_m^2 & \dots & x_m^N \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_N \end{bmatrix} + b = \begin{bmatrix} \hat{y}_0 \\ \hat{y}_1 \\ \vdots \\ \hat{y}_m \end{bmatrix}$$

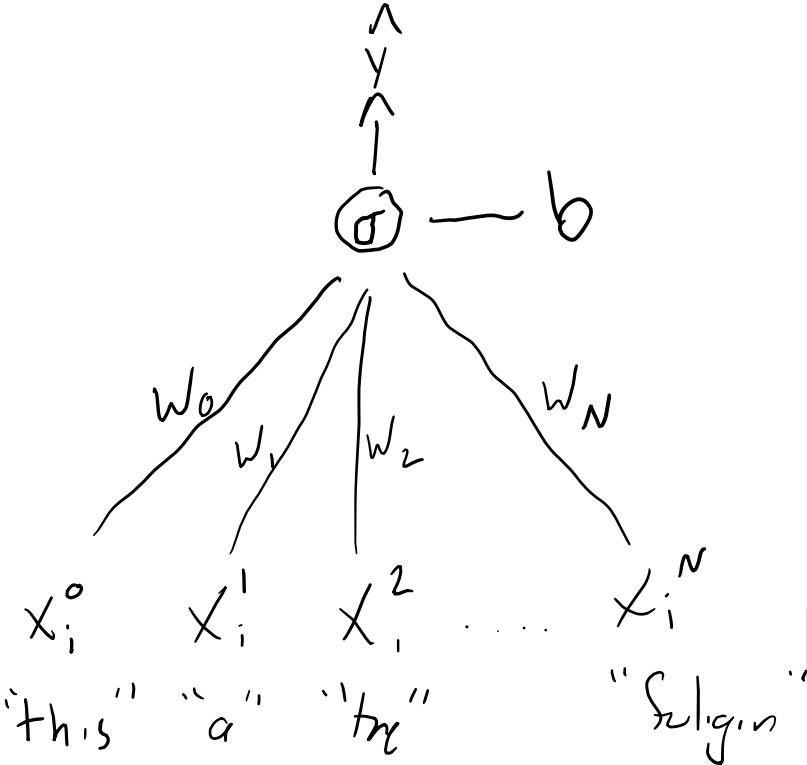
Or in a graphical form which shows how the individual x's come together to form  $\hat{y}$



# Visualizing logistic regression

You can do the same thing for logistic regression by adding the  $\sigma$  function

$$\begin{pmatrix} x_0^0 & x_0^1 & x_0^2 & \dots & x_0^N \\ x_1^0 & x_1^1 & x_1^2 & \dots & x_1^N \\ \dots & \dots & \dots & \dots & \dots \\ x_M^0 & x_M^1 & x_M^2 & \dots & x_M^N \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_N \end{pmatrix} + b = \begin{pmatrix} \hat{y}_0 \\ \hat{y}_1 \\ \dots \\ \hat{y}_M \end{pmatrix}$$



# Visualizing logistic regression

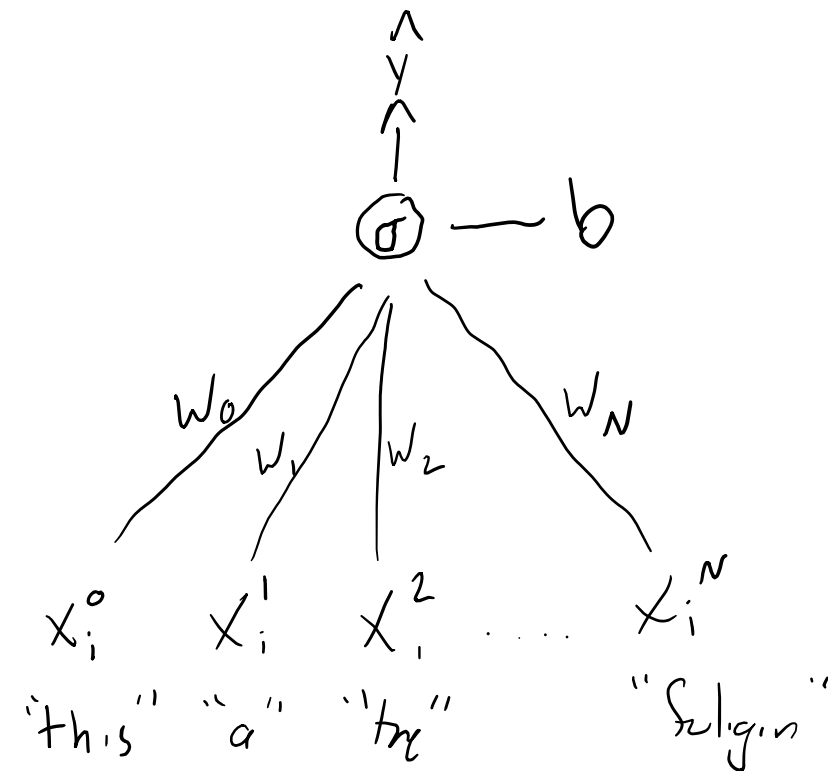
When we think of the logistic function as a final step being placed on top of the weighted sum  $Wx + b$  in order to squeeze it down to  $[0,1]$ , then we call it an **activation function**

There are a bunch of activation functions commonly used in neural nets:

- Rectified linear (relu), tanh, etc...

<https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity>

But logistic is the classic one



# Linear vs logistic regression in practice

---





# Read the SST-2 dataset

---

```
1 display(dev_df)
```

	sentence	label
0	it 's a charming and often affecting journey .	1
1	unflinchingly bleak and desperate	0
2	allows us to hope that nolan is poised to emba...	1
3	the acting , costumes , music , cinematography...	1
4	it 's slow -- very , very slow .	0
...	...	...
867	has all the depth of a wading pool .	0
868	a movie with a real anarchic flair .	1
869	a subject like this should inspire reaction in...	0
870	... is an arthritic attempt at directing by ca...	0
871	looking aristocratic , luminous yet careworn i...	1

872 rows × 2 columns



# Preprocessing and vectorizing the data

```
1 # SST-2 is actually already lowercased and tokenized, but it's good to get in
2 # the habit of always doing these during preprocessing
3 stemmer = PorterStemmer()
4 def preprocess(s):
5 | return ' '.join([stemmer.stem(token) for token in word_tokenize(s.lower())])
```

```
1 train_df['preprocessed'] = train_df['sentence'].apply(preprocess)
2 dev_df['preprocessed'] = dev_df['sentence'].apply(preprocess)
```

```
1 display(dev_df)
```

	sentence	label	preprocessed
0	it 's a charming and often affecting journey .	1	it 's a charm and often affect journey .
1	unflinchingly bleak and desperate	0	unflinchingly bleak and desper
2	allows us to hope that nolan is poised to emba...	1	allow us to hope that nolan is pois to embark ...
3	the acting , costumes , music , cinematography...	1	the act , costum , music , cinematographi and ...
4	it 's slow -- very , very slow .	0	it 's slow -- veri , veri slow .
...	...	...	...
867	has all the depth of a wading pool .	0	ha all the depth of a wade pool .
868	a movie with a real anarchic flair .	1	a movi with a real anarch flair .
869	a subject like this should inspire reaction in...	0	a subject like thi should inspir reaction in i...
870	... is an arthritic attempt at directing by ca...	0	... is an arthrit attempt at direct by calli k...
871	looking aristocratic , luminous yet careworn i...	1	look aristocrat , lumin yet careworn in jane h...

872 rows x 3 columns



# Preprocessing and vectorizing the data

---

```
1 from sklearn.feature_extraction.text import CountVectorizer

1 vectorizer = CountVectorizer()
2 train_X = vectorizer.fit_transform(train_df['preprocessed'])
3 display(train_X)

<67349x10106 sparse matrix of type '<class 'numpy.int64''>'
  with 535539 stored elements in Compressed Sparse Row format>

1 dev_X = vectorizer.transform(dev_df['preprocessed'])
2 display(dev_X)

<872x10106 sparse matrix of type '<class 'numpy.int64''>'
  with 12939 stored elements in Compressed Sparse Row format>
```



# Linear regression - Training

---

```
1 from sklearn.linear_model import LinearRegression

1 # As with most cases, it's very easy to instantiate and train a linear regression model
2 # in scikit-learn
3
4 # Note: SST-2 is NOT a regression task. It is a classification task.
5
6 # However, because the label is either 0 or 1, we can still sort of treat it as
7 # a regression task, by treating those as target values for the model.
8
9 # This only works for ordinal classification tasks, where there's an easy
10 # way of converting from labels to numbers
11
12 lin_reg_model = LinearRegression()
13 lin_reg_model.fit(train_X, train_df['label'])

LinearRegression()
```



# Linear regression - Training

```
1 # Notice that because it is a regression model, the predictions are continuous
2 # values that aren't necessarily bounded between 0 and 1
3 train_df['lin_reg_prediction'] = lin_reg_model.predict(train_X)
4 dev_df['lin_reg_prediction'] = lin_reg_model.predict(dev_X)
5
6 display(dev_df)
```

	sentence	label	preprocessed	lin_reg_prediction
0	it's a charming and often affecting journey .	1	it 's a charm and often affect journey .	1.156662
1	unflinchingly bleak and desperate	0	unflinchingly bleak and desper	-0.069508
2	allows us to hope that nolan is poised to emba...	1	allow us to hope that nolan is pois to embark ...	0.919836
3	the acting , costumes , music , cinematography...	1	the act , costum , music , cinematographi and ...	0.934645
4	it 's slow -- very , very slow .	0	it 's slow -- veri , veri slow .	0.031911
...	...	...	...	...
867	has all the depth of a wading pool .	0	ha all the depth of a wade pool .	0.514821
868	a movie with a real anarchic flair .	1	a movi with a real anarch flair .	1.922209
869	a subject like this should inspire reaction in...	0	a subject like thi should inspir reaction in i...	0.914590
870	... is an arthritic attempt at directing by ca...	0	... is an arthrit attempt at direct by calli k...	0.326833
871	looking aristocratic , luminous yet careworn i...	1	look aristocrat , lumin yet careworn in jane h...	0.098285

872 rows x 4 columns



# Linear regression - Evaluation

---

```
1 # We can't use classification metrics when we do regression, because those only work
2 # for discrete categorical values
3
4 # Instead, we measure the average discrepancy between target value and true value
5 # The two most common metrics are mean squared error (MSE) and mean absolute error (MAE)
6
7 from sklearn.metrics import mean_squared_error, mean_absolute_error
8
9 def evaluate_regression(y, py):
10 |     print(f'Mean squared error: {mean_squared_error(y,py):.3f}')
11 |     print(f'Mean absolute error: {mean_absolute_error(y,py):.3f}')
```

```
1 # Smaller values are better. Notice how overfitted we are to the training data
2
3 print('Train linear regression results:')
4 evaluate_regression(train_df['label'], train_df['lin_reg_prediction'])
5 print('\nDev linear regression results:')
6 evaluate_regression(dev_df['label'], dev_df['lin_reg_prediction'])
```

```
Train linear regression results:
Mean squared error: 0.078
Mean absolute error: 0.219
```

```
Dev linear regression results:
Mean squared error: 0.389
Mean absolute error: 0.472
```



# Linear regression - Evaluation

```
1 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
2 import numpy as np
3
4 # 0.5 is a reasonable value for the threshold in this circumstance
5 def evaluate_thresholded_regression(y, py, threshold=0.5):
6
7     # Numpy vectors/pandas Series can be compared to scalars, producing a bool
8     # vector, which the classification metric functions can handle
9     t_y = (y >= threshold)
10    t_py = (py >= threshold)
11
12    print(f'Accuracy: {accuracy_score(t_y, t_py):.3f}')
13    print(f'Precision: {precision_score(t_y, t_py):.3f}')
14    print(f'Recall: {recall_score(t_y, t_py):.3f}')
15    print(f'F1: {f1_score(t_y, t_py):.3f}')
```

```
1 # We were doing better with Naive Bayes a couple weeks ago.
2 print('Thresholded train linear regression results:')
3 evaluate_thresholded_regression(train_df['label'], train_df['lin_reg_prediction'])
4 print('\nThresholded dev linear regression results:')
5 evaluate_thresholded_regression(dev_df['label'], dev_df['lin_reg_prediction'])
```

Thresholded train linear regression results:

Accuracy: 0.927  
Precision: 0.919  
Recall: 0.953  
F1: 0.936

Thresholded dev linear regression results:

Accuracy: 0.744  
Precision: 0.735  
Recall: 0.779  
F1: 0.756



# Linear regression - Global explanations

```
5 def explain_binary_linear_model(model, vocabulary, k=10):
6     # This is necessary because the coefs for logistic regression can be a 2D matrix
7     # (though they won't be in this notebook)
8     model_coefs = np.squeeze(model.coef_)
9     sorted_coef_indices = np.argsort(np.abs(model_coefs))
10    # Grab the indices of the top k indices and flip their order to descending
11    top_k_indices = sorted_coef_indices[:k:-1]
12    # Numpy vectors can be indexed by multiple indices at once
13    top_k_coefs = model_coefs[top_k_indices]
14    # Python lists aren't so flexible, so we have to use a comprehension
15    top_k_words = [vocabulary[index] for index in top_k_indices]
16    for word, coef in zip(top_k_words, top_k_coefs):
17        print(f'\tWord: {word} - Coef: {coef:.3f}')
```

```
1 # These words make... a lot less sense than the ones we got back for Naive Bayes
2 # Why might that be?
3 print(f'Top 10 coefficients in our linear regression model:')
4 explain_binary_linear_model(lin_reg_model, vocabulary)
```

Top 10 coefficients in our linear regression model:

```
Word: edmund - Coef: -2.093
Word: embed - Coef: 1.660
Word: schtick - Coef: 1.619
Word: interchang - Coef: 1.581
Word: drung - Coef: 1.561
Word: purgatori - Coef: -1.548
Word: size - Coef: -1.380
Word: wire - Coef: 1.357
Word: kangaroo - Coef: -1.353
```







# “Kangaroo” and “Edmund”


---

“Kangaroo”



**KANGAROO JACK**  
PG 2003, Comedy, 1h 29m

 **8%**  
TOMATOMETER  
114 Reviews

 **29%**  
AUDIENCE SCORE  
50,000+ Ratings

“Edmund”

???



# Linear regression – Local explanations

---

```
1 # We can also generate explanations for individual predictions using similar logic
2
3 def explain_binary_linear_model_prediction(input,model, vectorizer):
4
5     # This is necessary because the coefs for logistic regression can be a 2D matrix
6     # (though they won't be in this notebook)
7     model_coefs = np.squeeze(model.coef_)
8
9     # Assume the input hasn't been preprocessed, so do that and then vectorize it
10    preprocessed = preprocess(input)
11    tokens = preprocessed.split(' ')
12    input_X = vectorizer.transform([preprocessed])
13
14    py = model.predict(input_X)[0]
15    print(f'Prediction: {py}')
16
17    print(f'Word coefficients:')
18    for token in tokens:
19        if token in vectorizer.vocabulary_: # Skip any tokens that are not in the vectorizer vocab
20            token_index = vectorizer.vocabulary_[token]
21            token_coef = model_coefs[token_index]
22            print(f'\tWord: {token} - Coef: {token_coef:.3f}')
23
24    print(f'Model intercept: {model.intercept_}')
```





# Binary logistic regression – Training

---

```
1 from sklearn.linear_model import LogisticRegression
```

```
1 log_reg_model = LogisticRegression()  
2 log_reg_model.fit(train_X, train_df['label'])
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (`max_iter`) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(  
LogisticRegression())
```



# Linear regression – Training

```
1 train_df['log_reg_prediction'] = log_reg_model.predict(train_X)
2 dev_df['log_reg_prediction'] = log_reg_model.predict(dev_X)
3
4 display(dev_df)
```

	sentence	label	preprocessed	lin_reg_prediction	log_reg_prediction
0	it 's a charming and often affecting journey .	1	it 's a charm and often affect journey .	1.156662	1
1	unflinchingly bleak and desperate	0	unflinchingly bleak and desper	-0.069508	0
2	allows us to hope that nolan is poised to emba...	1	allow us to hope that nolan is pois to embark ...	0.919836	1
3	the acting , costumes , music , cinematography...	1	the act , costum , music , cinematographi and ...	0.934645	1
4	it 's slow -- very , very slow .	0	it 's slow -- veri , veri slow .	0.031911	0
...	...	...	...	...	...
867	has all the depth of a wading pool .	0	ha all the depth of a wade pool .	0.514821	0
868	a movie with a real anarchic flair .	1	a movi with a real anarch flair .	1.922209	1
869	a subject like this should inspire reaction in...	0	a subject like thi should inspir reaction in i...	0.914590	0
870	... is an arthritic attempt at directing by ca...	0	... is an arthrit attempt at direct by calli k...	0.326833	0
871	looking aristocratic , luminous yet careworn i...	1	look aristocrat , lumin yet careworn in jane h...	0.098285	1

872 rows x 5 columns

# Logistic regression – Evaluation

```
1 # Now we don't have to do any thresholding of our own, so we can
2 # just use the classification metrics as-is
3 def evaluate_classification(y, py):
4     print(f'Accuracy: {accuracy_score(y, py):.3f}')
5     print(f'Precision: {precision_score(y, py):.3f}')
6     print(f'Recall: {recall_score(y, py):.3f}')
7     print(f'F1: {f1_score(y, py):.3f}')
```

```
1 # Looking better! This is pretty similar to what we were getting with Naive Bayes
2 # Still some overfitting happening though.
3 print('Train logistic regression results:')
4 evaluate_classification(train_df['label'], train_df['log_reg_prediction'])
5 print('\nDev logistic regression results:')
6 evaluate_classification(dev_df['label'], dev_df['log_reg_prediction'])
```

```
Train logistic regression results:
Accuracy: 0.926
Precision: 0.927
Recall: 0.940
F1: 0.934
```

```
Dev logistic regression results:
Accuracy: 0.808
Precision: 0.794
Recall: 0.842
F1: 0.817
```

```
Thresholded train linear regression results:
Accuracy: 0.927
Precision: 0.919
Recall: 0.953
F1: 0.936
```

```
Thresholded dev linear regression results:
Accuracy: 0.744
Precision: 0.735
Recall: 0.779
F1: 0.756
```



# Logistic regression – global explanations

---

```
1 # These look better
2 print(f'Top 10 coefficients in our logistic regression model:')
3 explain_binary_linear_model(log_reg_model, vocabulary)
```

Top 10 coefficients in our logistic regression model:

Word: lack - Coef: -4.389  
Word: worst - Coef: -4.069  
Word: anatom - Coef: 4.015  
Word: devoid - Coef: -3.871  
Word: failur - Coef: -3.703  
Word: refresh - Coef: 3.654  
Word: stupid - Coef: -3.521  
Word: assum - Coef: -3.437  
Word: mess - Coef: -3.407







# Concluding thoughts

---

## Linear regression

- Learn  $Wx + b$  from data
- Predict continuous values
- Optimize mean squared error

## Logistic regression

- Learn  $\sigma(Wx + b)$  from data
- Predict (close to) 0 or 1
- Optimize cross-entropy

## Key concepts:

- Loss function
  - I.e. objective function
- Gradient of loss with respect to parameters
- Gradient descent
- Activation function

