Linear Regression

Software Innovation Institute, ANU

Slides: Nathan Elazar & Alex Mathews

Regression: Overview

- We want to create a program that can make predictions/inferences about things.
  – given the text of an email, predict would a human consider this email as spam.
  – given the text of a movie review calculate a numeric score

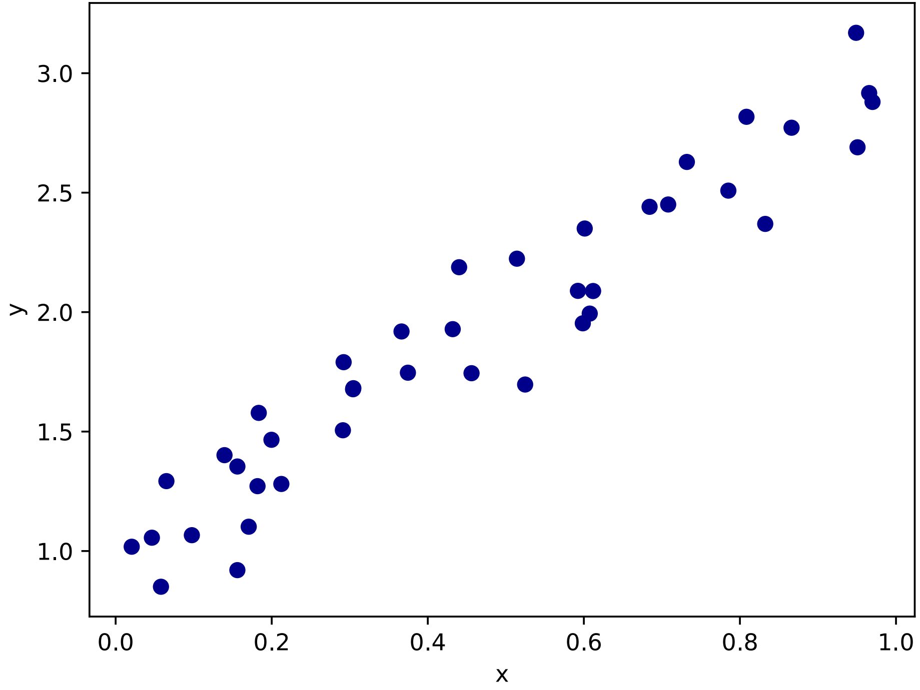
Regression: Overview

- One way we can do this is by having the program learn from examples (called a training dataset).
  - It needs to describe which kinds of example emails get labelled as spam.
  - The goal is to make predictions about new emails as well!

Regression: Overview

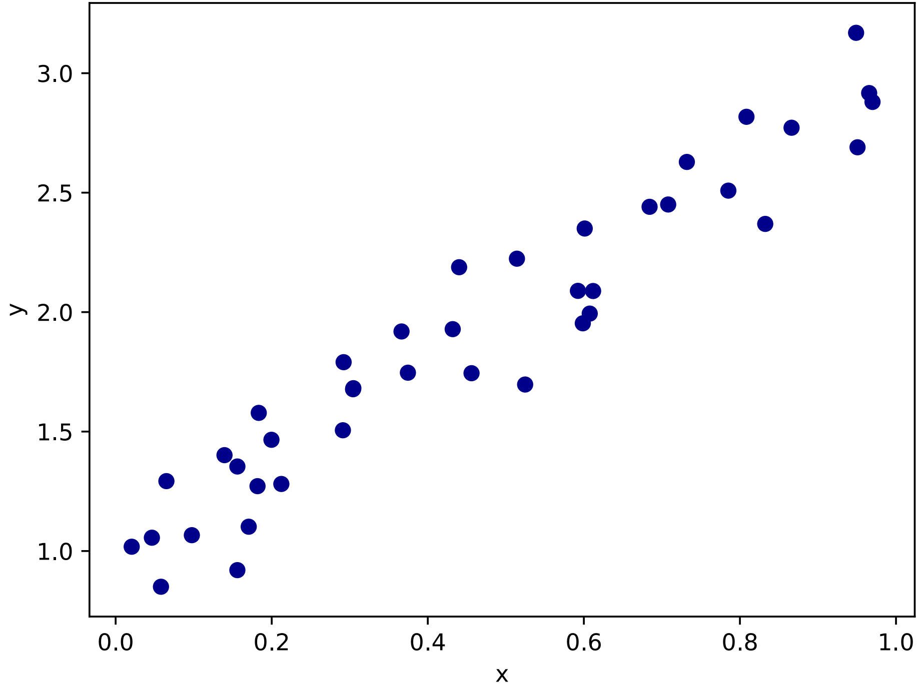
- In general, it is challenging to create programs that learn.
- Let’s start by considering a simple example.

Regression: Example



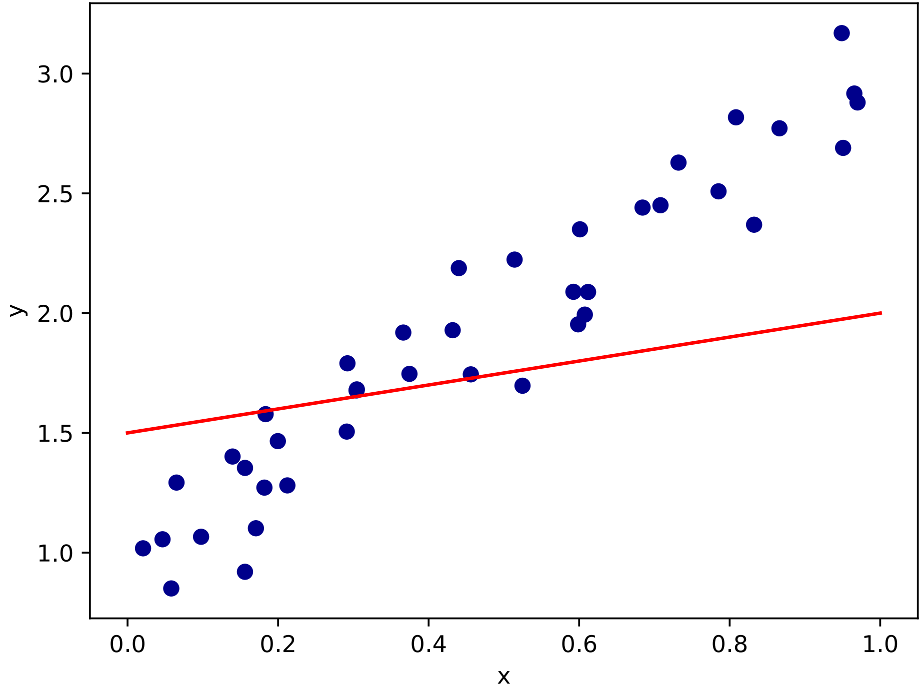
- Let this be our training dataset.
- We have sampled 40 different x values, each x value is labelled with its correct y value.
- We want to create a program that can tell us what the y value of a new point is.
- **What do you predict the y value of a point with $x = 1.5$ will be?**

Regression: Example



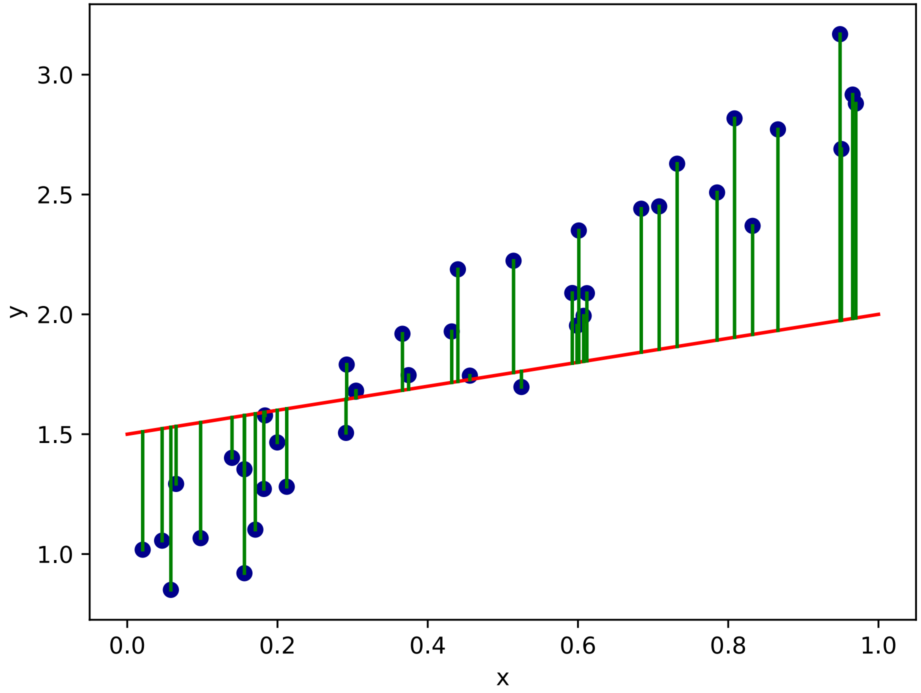
- Key idea: find a simple function which explains your training dataset.
- We need a function which maps inputs to outputs.
- For all of the x points in our training dataset, the output of our function should be as close as possible to the true y value.
- **Which function would you choose?**

Regression: Example



- Let's try fitting a straight line through the points.
- Is this a good model of the data?

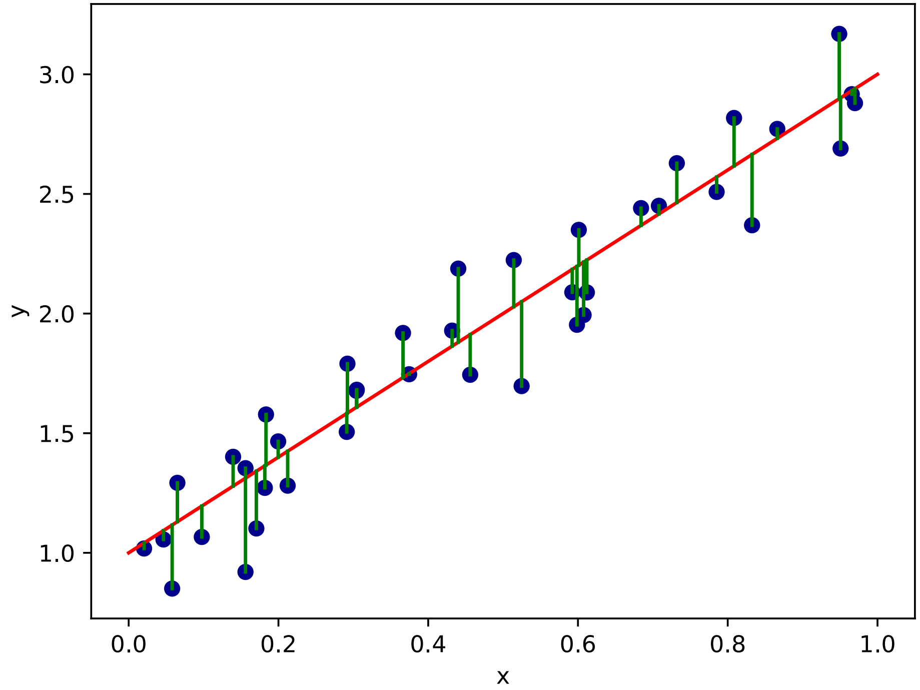
Regression: Example



- Let's try fitting a straight line through the points.
- Is this a good model of the data?

- **No** - there are large differences between our line (predicted y) and the target points (actual y).
- We can do better.

Regression: Example



- This looks much better!

Linear Regression

- We find the best linear function to model our data points.
- Then to make a prediction for a new point we just evaluate our function at that point.

So what do we need?

- A way of expressing linear functions with programs
  - Easy, a straight line corresponds to $wx + b$.
- A way to measure how good a line is (Loss function)
  - In the example before it was $L(w, b) = \sum_{i=1}^{n} |wx + b - y|$, there are other measures.
- A way of automatically finding the best line, given a dataset and a loss function.
  - This is the most complicated part.

Optimization

- We have a loss function L which measures how good a particular line (w, b) is for our training dataset.
- We want to solve: $\underset{w,b}{\arg\min}\, L(w, b)$
- There are many different algorithms for solving optimization problems, gradient descent is a very popular one.

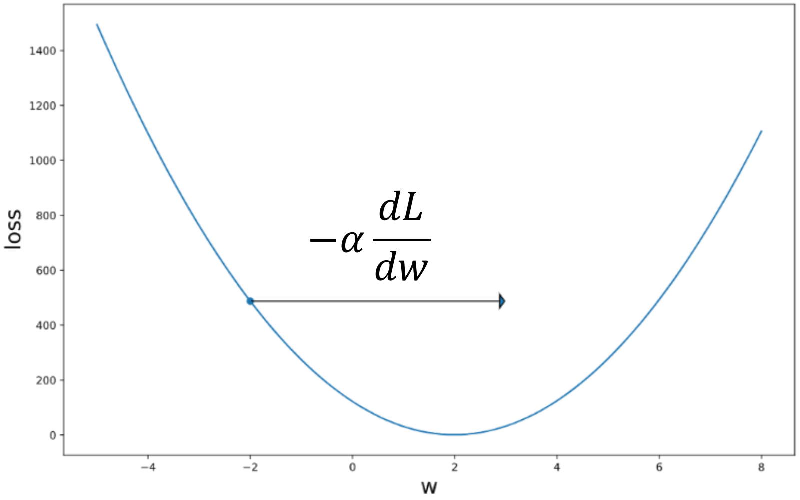
Gradient Descent

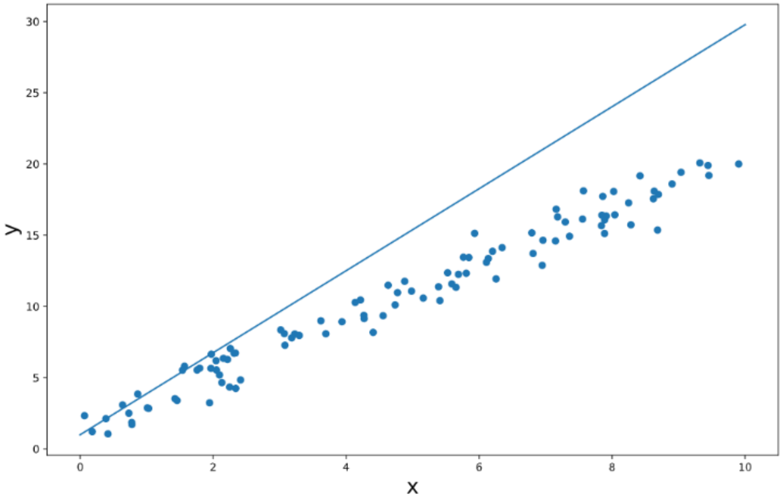
- Start by picking completely random values for w and b, e.g. by sampling from $N(0,1)$.
- Then compute the gradients $\frac{dL}{dw}$ and $\frac{dL}{db}$.
- Then update:

$$w := w - \alpha \frac{dL}{dw} \quad b := b - \alpha \frac{dL}{db}$$

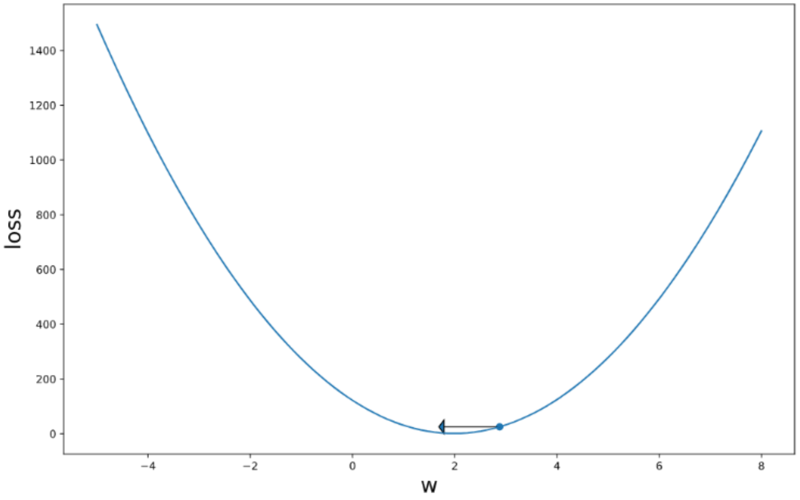
α is the learning rate,
It controls how much the values change each step.

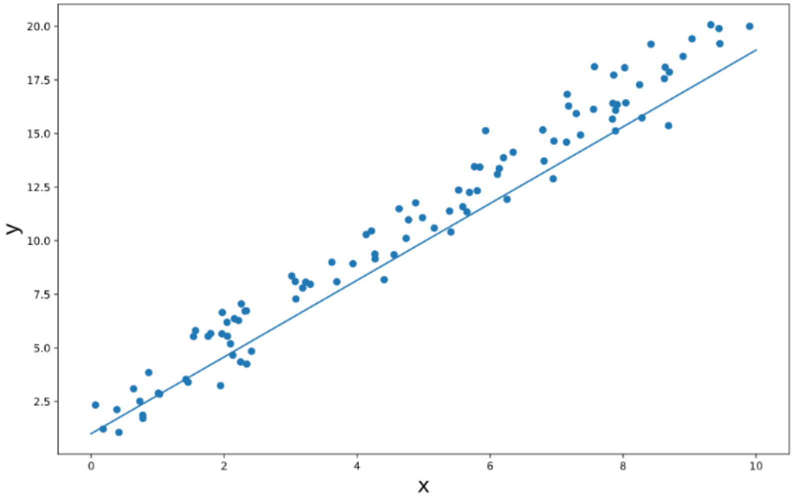
Gradient Descent:



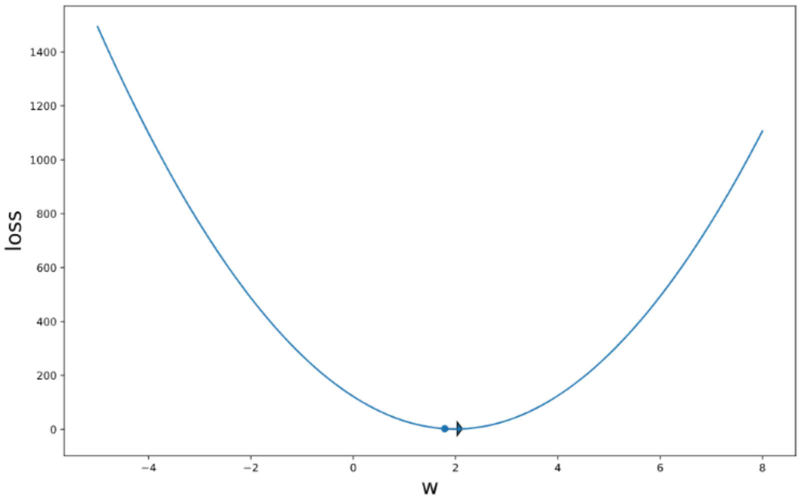


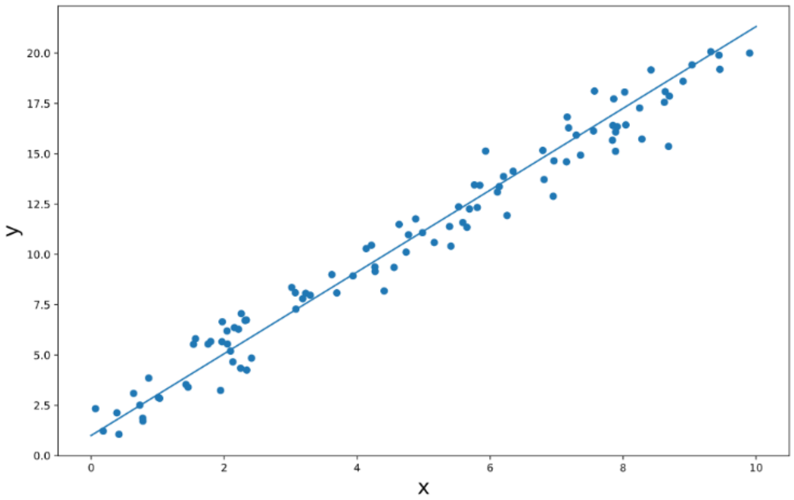
Gradient Descent:



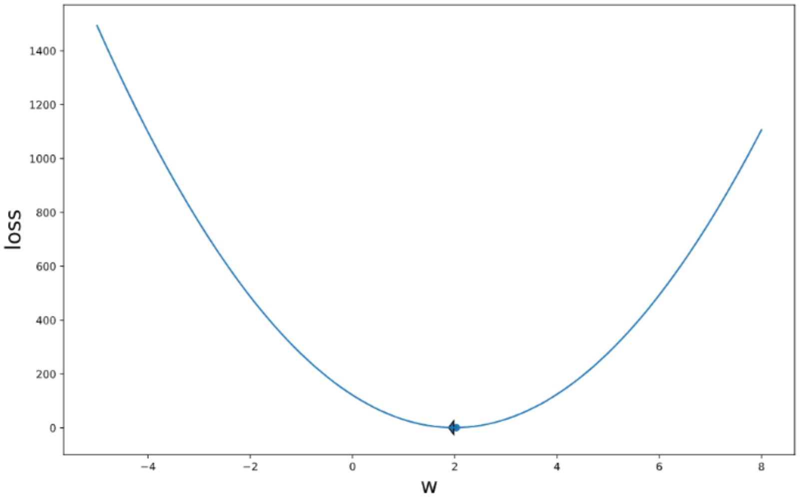


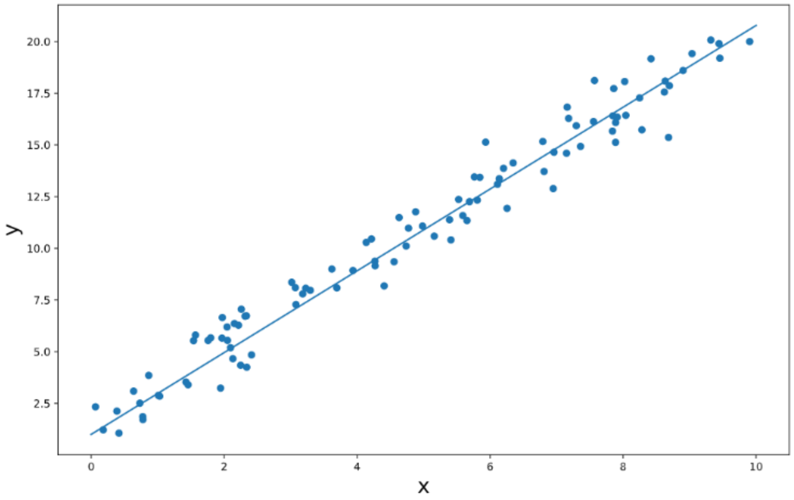
Gradient Descent:



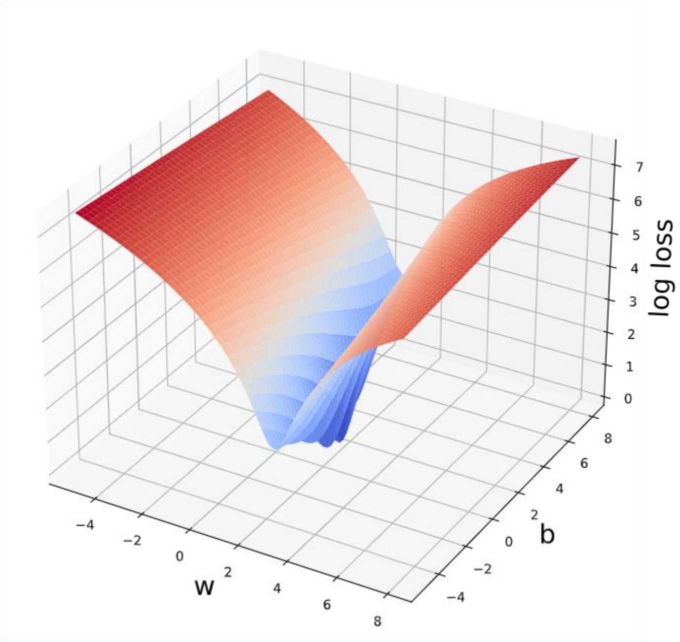


Gradient Descent:



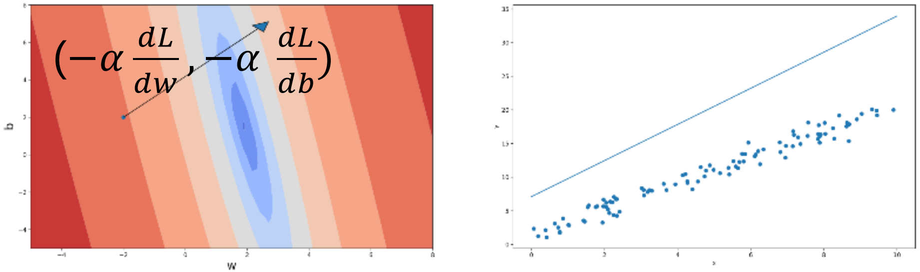


Gradient Descent w and b

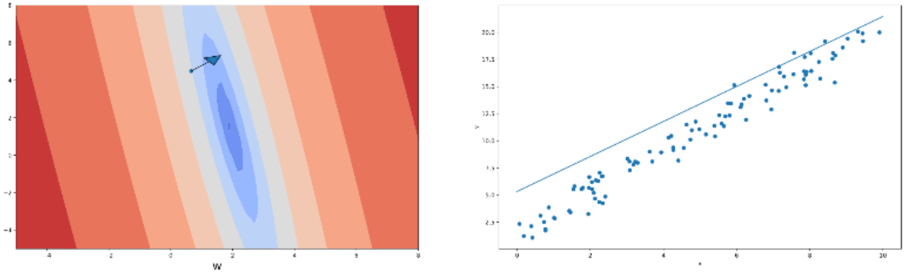


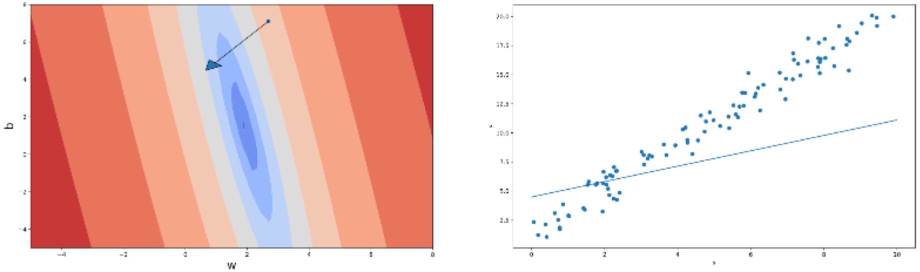
Gradient Descent w and b

Iteration 1

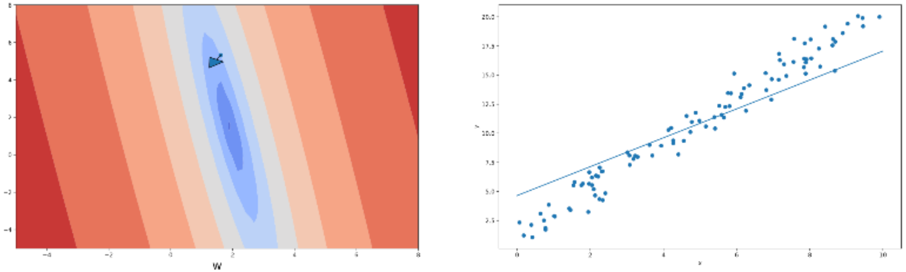


Iteration 3





Iteration 2



Iteration 4

Gradient Descent

$$w := w - \alpha \frac{dL}{dw} \qquad\qquad b := b - \alpha \frac{dL}{db}$$

- Keep doing these updates over and over again and until w and b stop changing.
- You have now found a (locally) optimal solution.

The role of learning rate

- The learning rate α is a hyper-parameter that you set, it controls how much the weights are changed in each step.
- Too small $\alpha \Rightarrow$ need to take many steps.
- Too large $\alpha \Rightarrow$ may not converge.
- Up to you to choose a good value of α.

What about higher dimensions?

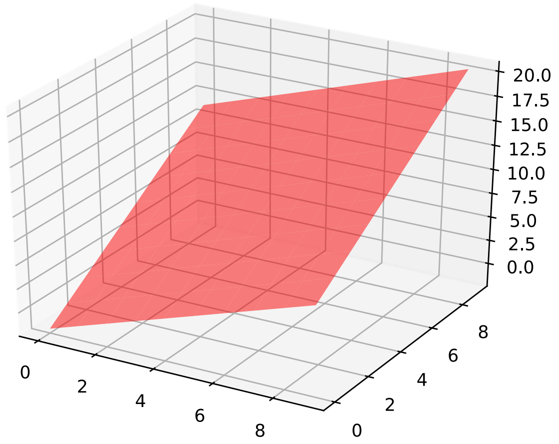
- Up until now our training dataset has contained single-valued inputs and outputs.
- Often we will need more than just one number/feature (in NLP this is often 100s to 10 of thousands)

What about higher dimensions?

- A linear function from I inputs to O outputs can be represented as

$$y = Wx + b$$

  - W is a matrix with shape $[O, I]$
  - b is a vector with shape $[O]$
  - x is a vector with shape $[I]$
  - y is a vector with shape $[O]$



Plot of $y = [1 \quad 1.5] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$+[-2] = $x_1 + 1.5x_2 - 2$

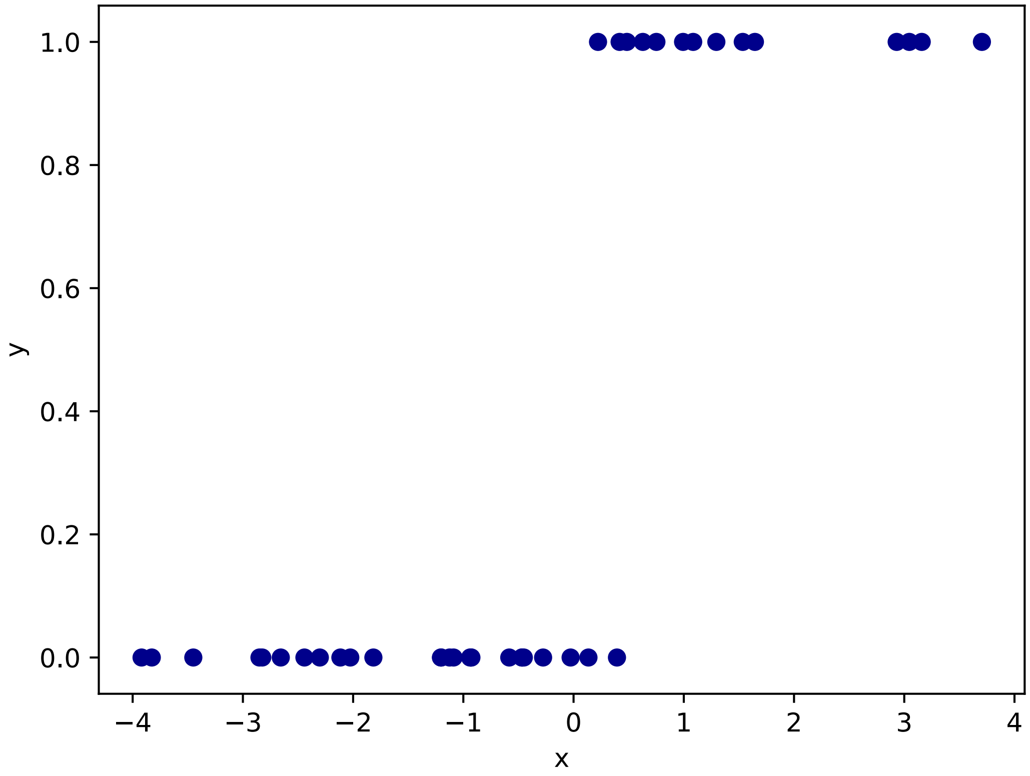
$W \qquad x \qquad b$

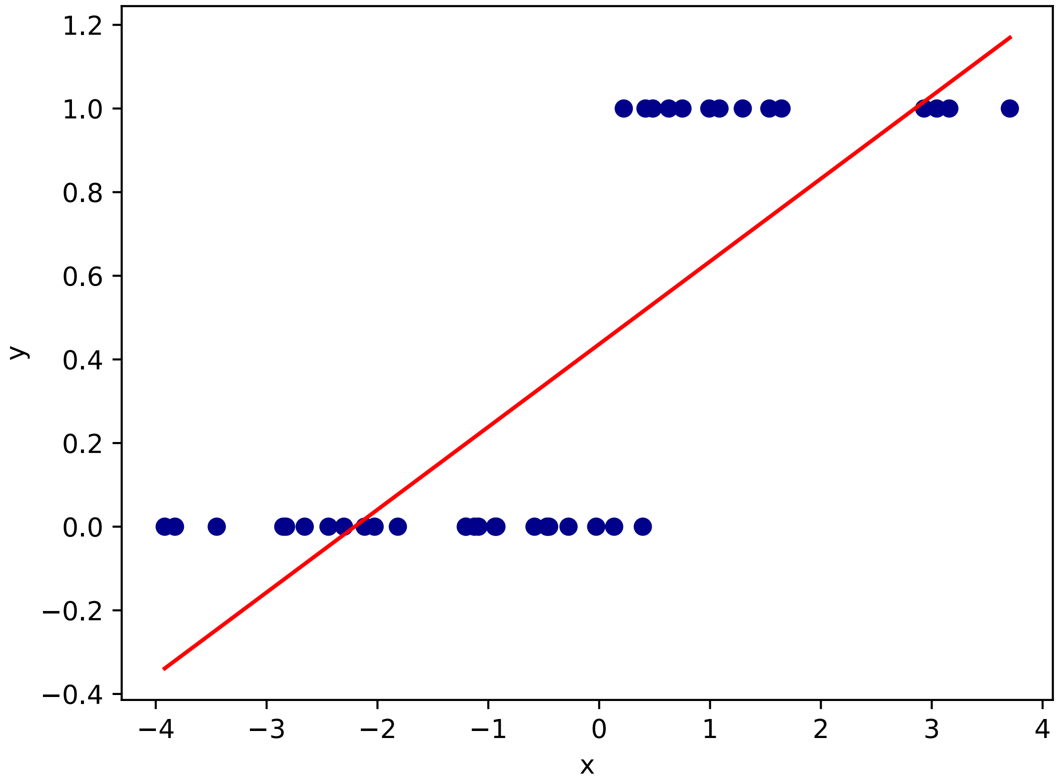
Linear Regression: Formal Description

- Given a dataset $D = \{(x_i \in \mathbb{R}^I, y_i \in \mathbb{R}^O)\}_{i=1}^n$ and a loss function $L_D: (\mathbb{R}^{O\times I} \times \mathbb{R}^O) \to \mathbb{R}$.
- Solve $\underset{(W\in\mathbb{R}^{O\times I}, b\in\mathbb{R}^O)}{argmin} \; L_D(W, b)$

Classification

- What if the target we wish to predict is discrete?
  – Example: classify emails as spam vs not spam.
- Will our linear regression model work?


1.0
0.8
0.6
0.4
0.2
0.0
y
−4
−3
−2
−1
0
1
2
3
4
x


1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
y
−4
−3
−2
−1
0
1
2
3
4
x

Classification

- Is this a problem for our linear model?
  - Yes in its current form
  - We can make a change so it outputs the *probability* that an example belongs to a class

Multinomial Logistic Regression

- Suppose that each point can be labelled with one of O different classes.
- Then our model outputs a vector of size O.
- The output values could be any real numbers, so we need to turn them into probabilities.
  - Each probability is non-negative.
  - The probabilities sum to 1.

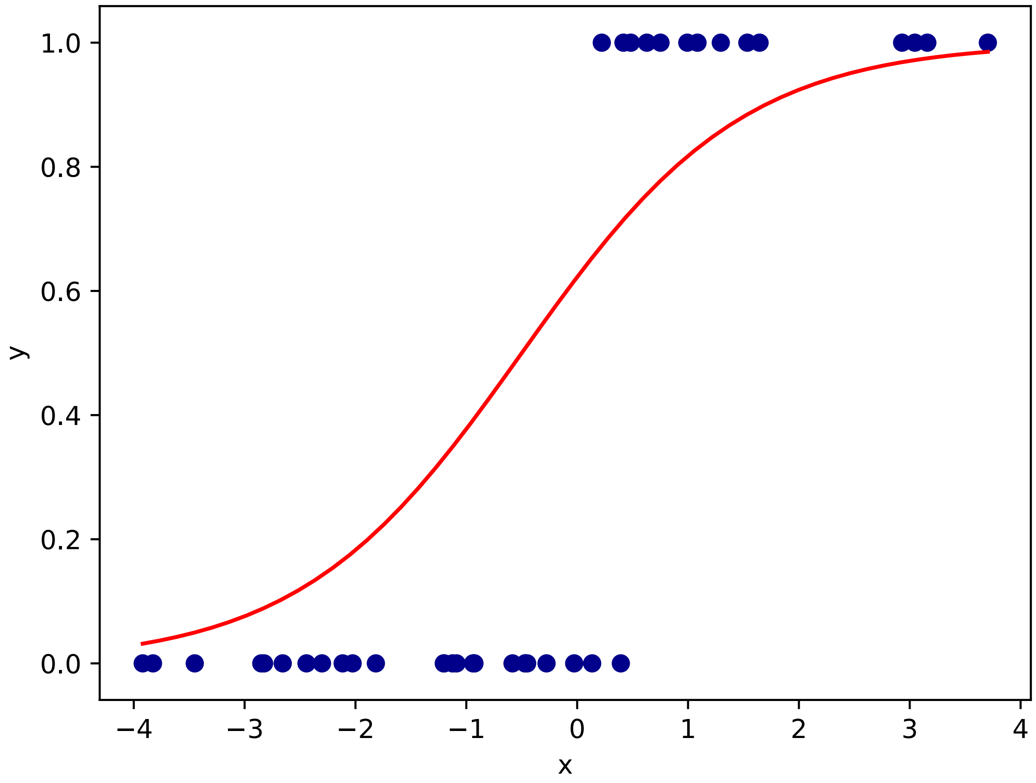
$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{j=0}^{O} \exp(x_j)}$$

- Step 1: Apply exp to all values. This makes them positive.
- Step 2: Divide each value by the sum of all values. This makes them sum to 1.
- Now we can interpret the output as probabilities.

Classification: Loss

- Compute loss between our predicted probabilities and one-hot encoding of class label y, i.e. the probability of the correct class should be 1, all others should be 0.

$$\mathrm{L}(\mathrm{softmax}(Wx + b), y)$$


1.0
0.8
0.6
0.4
0.2
0.0
y
−4
−3
−2
−1
0
1
2
3
4
x

Classification: Cross-Entropy

- We could use the sum of absolute differences as our loss function, just like for regression.
- In practice, using cross-entropy as the loss function for classification gives better results.

$$\mathrm{CrossEntropy}(x, y) = -\sum_{i=1}^{O} y_i \log x_i$$

$\boldsymbol{x_1}$	$\boldsymbol{x_2}$	$\boldsymbol{y}$
$\mathbf{1.24}$	$\mathbf{-0.32}$	$\mathbf{2}$
$\mathbf{-0.11}$	$\mathbf{2.64}$	$\mathbf{0}$

⋮

Suppose that each of our points is labelled with one of three possible values {0, 1, 2}, then we would train a classifier like this.

$$\mathrm{L}(\mathrm{softmax}\left(\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} 1.24 \\ -0.32 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right), \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix})$$

$$+$$

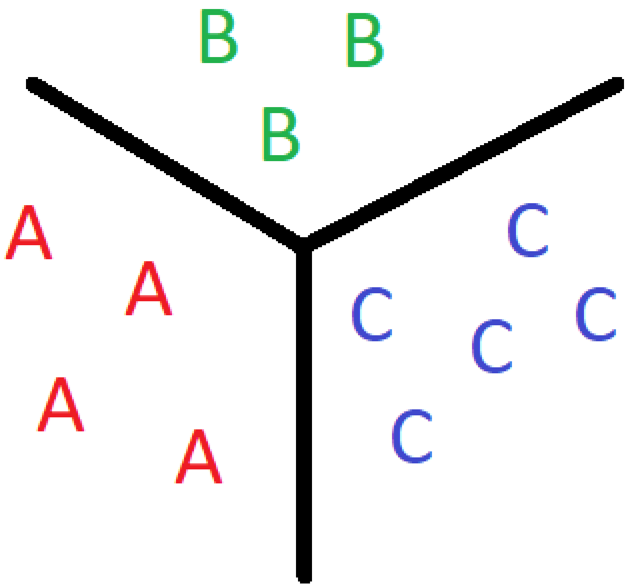
$$\mathrm{L}(\mathrm{softmax}\left(\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} -0.11 \\ 2.64 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right), \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix})$$

$$+$$

⋮

Classification: Prediction

- To make a prediction for a new data point, apply your model to compute probabilities for each class. Predict the class with the largest probability.



When you predict a class in this way the decision boundaries are straight lines.

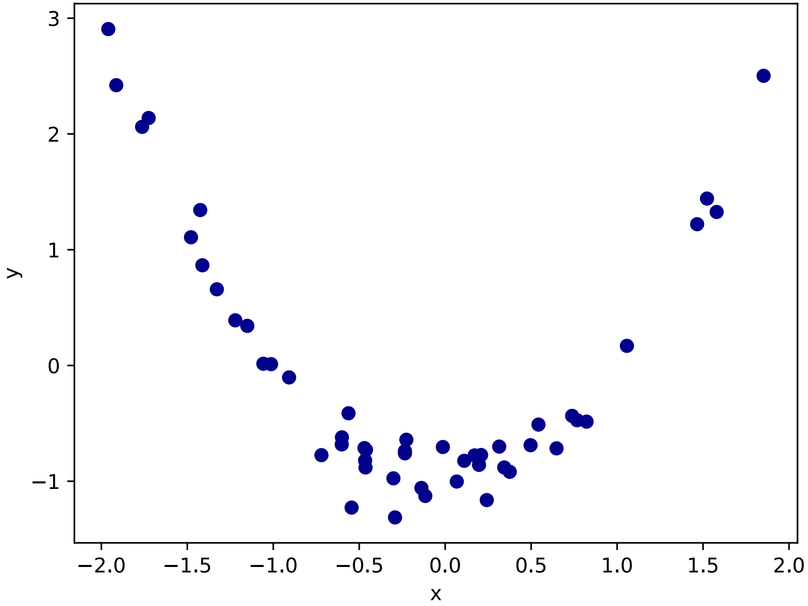
Why? (hint: think about how the probability is calculated.)

Applying Linear Regression to Text

- Our model needs numeric (continuous) input values.
- How do we describe text with numbers?
  - Think back to Information retrieval section. Can we do the same thing here?
  - Next lecture: vector representations of text.

Shortcomings of Linear Regression

- What do we do if our data looks like this?



Summary

- We can create programs which make predictions/inferences about their input.
- Select the (linear) function which achieves the smallest training loss, use it to make predictions.