CSC413 Neural Networks and Deep Learning

Lecture 1: Introduction and Review of Linear Models

January 9 / 11, 2024

Table of Contents

Section 1

Welcome to CSC413!

Introduction: Instructors

Amanjit Singh Kainth

- LEC 0101/2001: Tuesday (T) 1:00PM - 4:00PM
- Office Hours: TBA

Amir-massoud Farahmand

- LEC 0201/2101: Thursday (R) 1:00PM - 4:00PM
- Office Hours: TBA

Robert (Rupert) Wu

- LEC 5101/2501: Tuesday (T) 6:00PM - 9:00PM
- Office Hours: T4-5 BA 2272

Section 2

How can we make an intelligent agent?

How can we make an intelligent agent?

- What does it mean to have an intelligent agent?
- What do we need to create it?

An AI Agent

Figure 1: An agent ...

An AI Agent

Figure 2: ... observes the world ...

An AI Agent

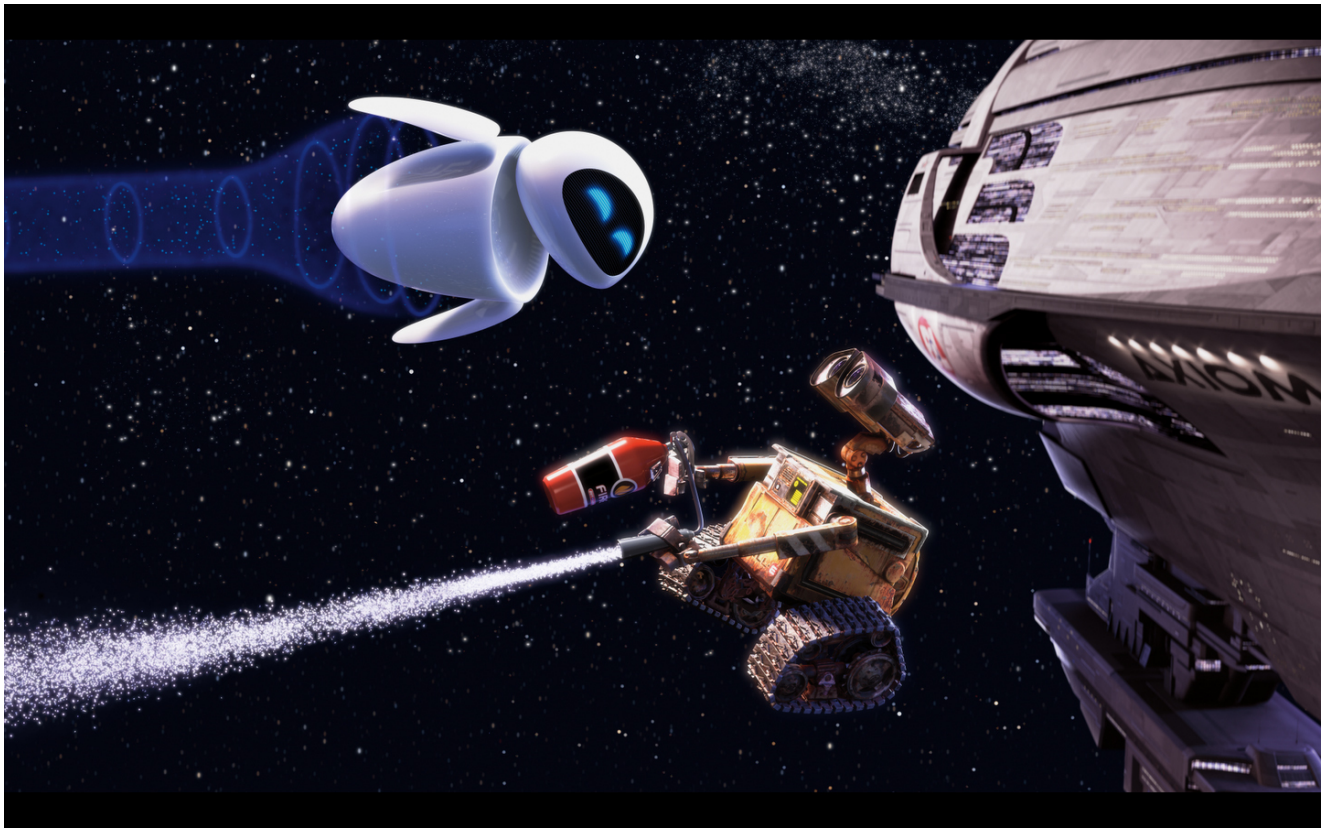


Figure 3: ... takes an action and its states changes ...

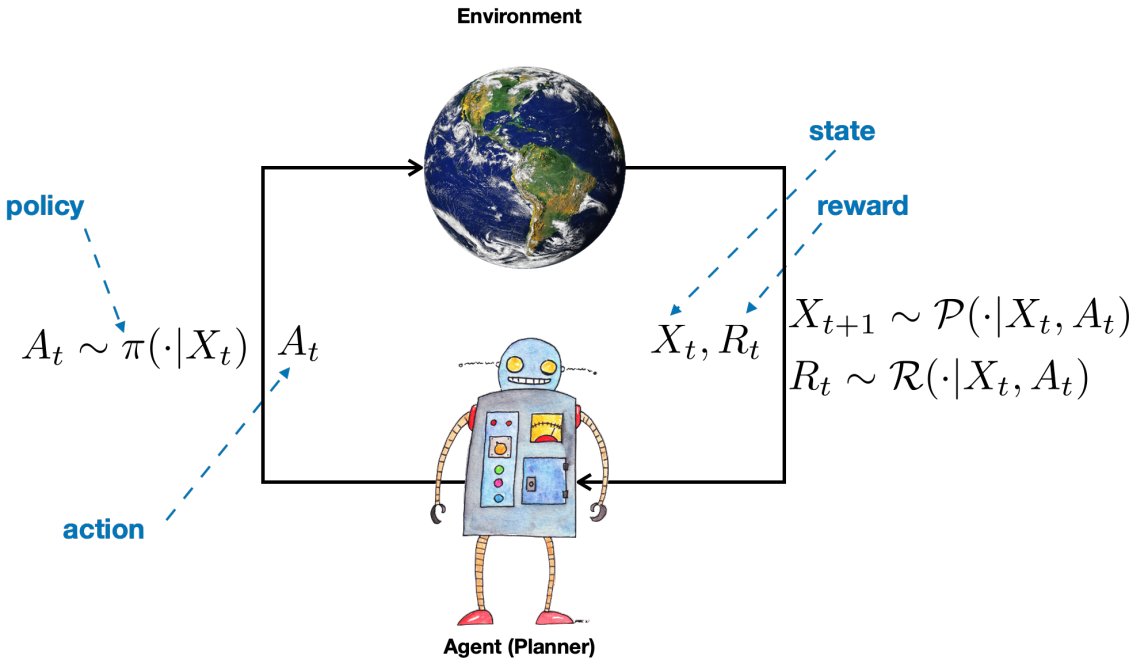
An AI Agent

Figure 4: ... with the goal of achieving long-term rewards.

An AI Agent: Some Requirements

This agent has to

- **predict** how the world works, e.g.,
  - classify different objects
  - estimate the probability of certain events happening in the future
- **plan** its actions in order to achieve its long-term goals



We use Machine Learning and Neural Networks to move towards this goal.

What is the difference between...

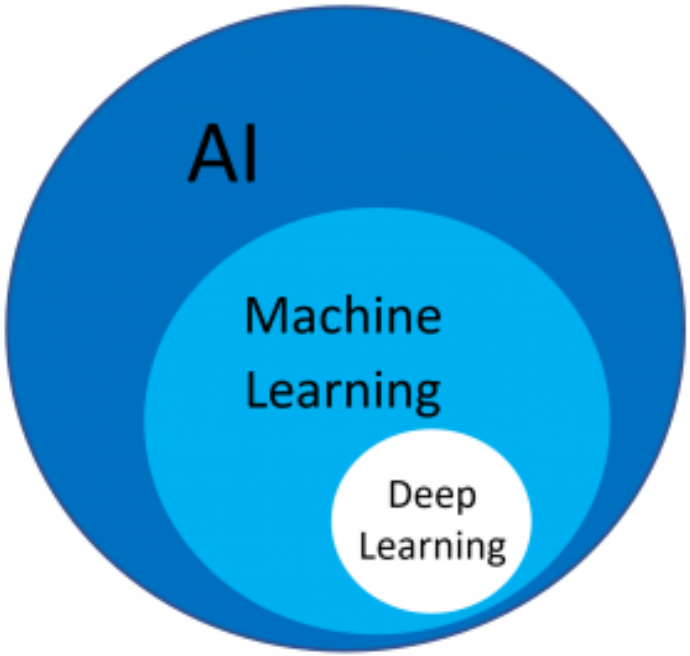
- Artificial Intelligence
- Machine Learning
- Deep Learning

AI vs ML vs DL

Artificial Intelligence: Create intelligent machines that perceive, reason, and act like humans. (CSC384)

Machine Learning: Design algorithms to automatically learn from data. (CSC311)

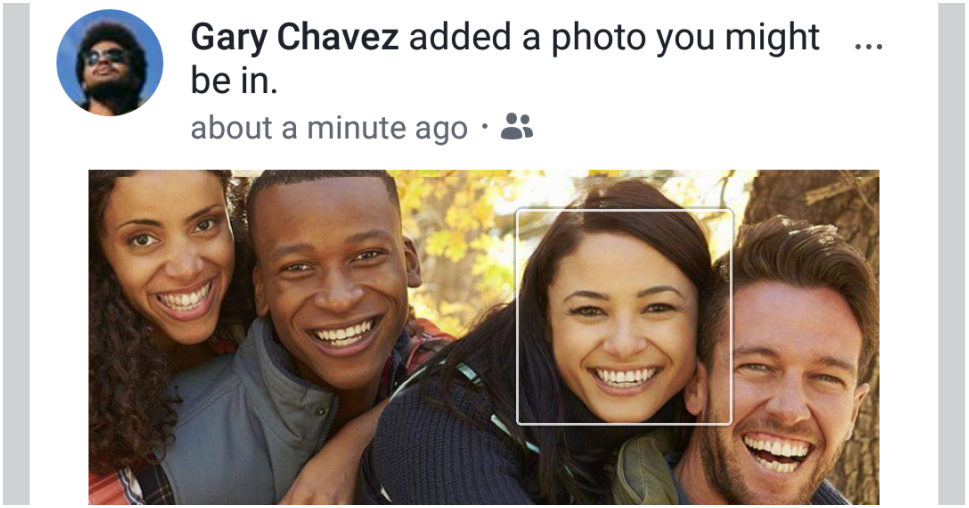
Deep Learning: Using deep neural networks to automatically learn from data. (CSC413)

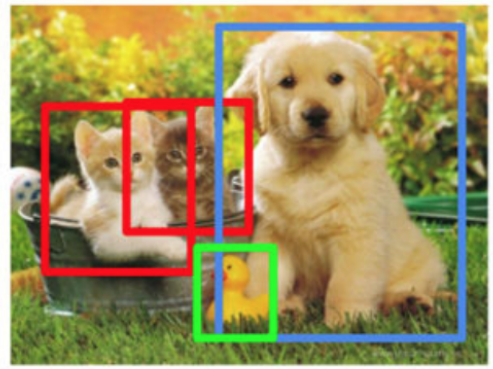


Why machine learning after all?

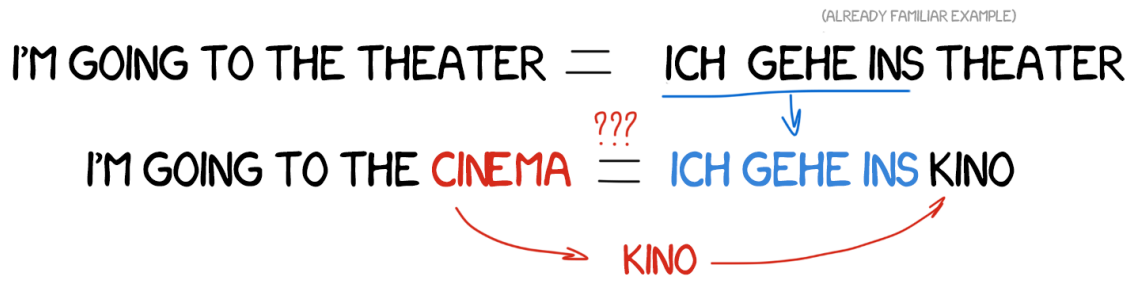
Our agent lives in a complicated world. It is difficult to program the correct behaviour (ex. recognizing chairs, apples, etc.) manually.

Machine learning approach: Write an algorithm to automatically learn from data.





CAT, DOG, DUCK



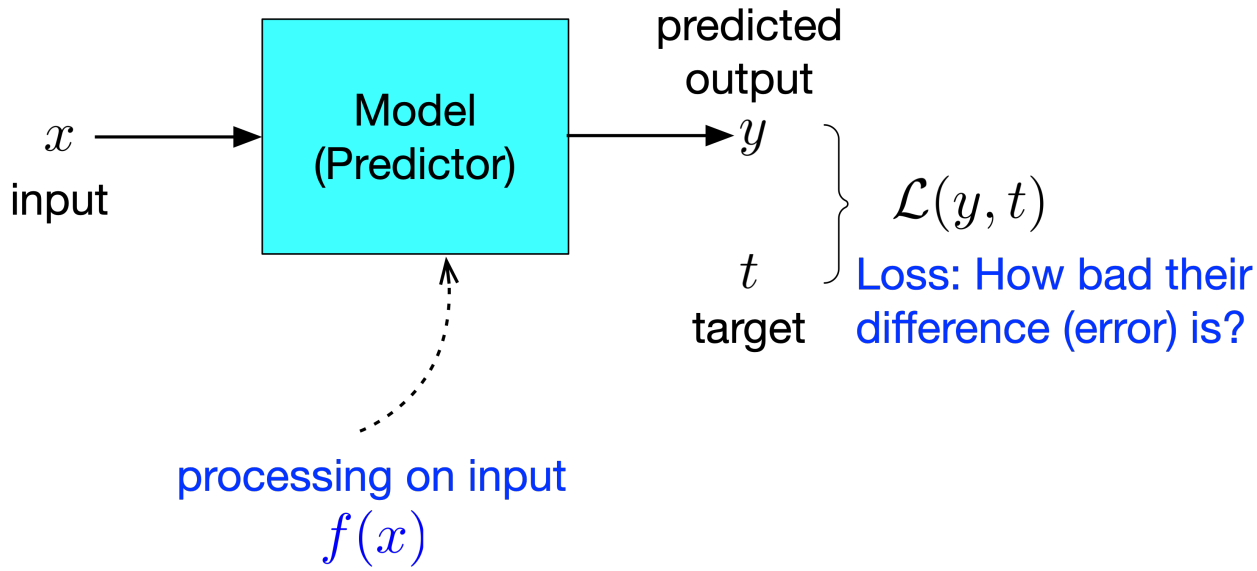
Types of problems in Machine Learning

- **Supervised Learning**: have labeled examples of the correct behaviour, i.e. ground truth input/output response
  - Regression (e.g. height prediction)
  - Classification (e.g. sentiment classification)
- **Unsupervised Learning**: no labeled examples; instead, looking for interesting patterns in the data (e.g. clustering)
- **Reinforcement Learning**: learning system receives a reward signal, tries to learn to maximize the reward signal (e.g. playing StarCraft)

This categorization is not strict. There are overlaps between these problems, and there are problems that belong to more than one category.

- Examples: semi-supervised learning, self-supervised learning, model learning for reinforcement learning, etc.

The Machine Learning Approach



Reframe **learning problems** into **optimization** problems by:

- Choosing a model (with parameters to be tuned)
- Choosing a loss/cost function to measure how well the model fits the data given a choice of parameters
- Choosing an optimizer to minimize the cost function

Different machine learning approaches differ in the model, loss, and optimizer choice.

Section 3

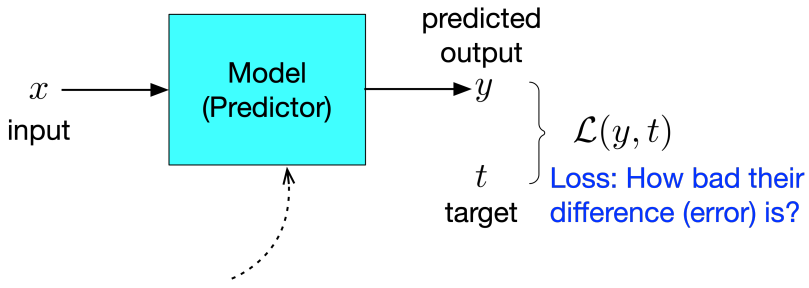
Overview of Supervised Learning with Linear Models

Overview of Supervised Learning with Linear Models

This is going to be quick! We assume that you are already familiar with these material because you have taken CSC311.

Supervised Learning with Linear Models

- Let us review Linear Models.
- They also appear as one of the building blocks in neural networks and deep learning.



Common supervised learning problems

Recall the types of supervised learning problems:

- **Regression**: predict a scalar-valued target (e.g. stock price)
- **Classification**: predict a label
  - **Binary classification**: predict a binary label (e.g. spam vs. non-spam email)
  - **Multi-class classification**: predict a discrete label (e.g. object category, from a list)

Problem Setup: Regression

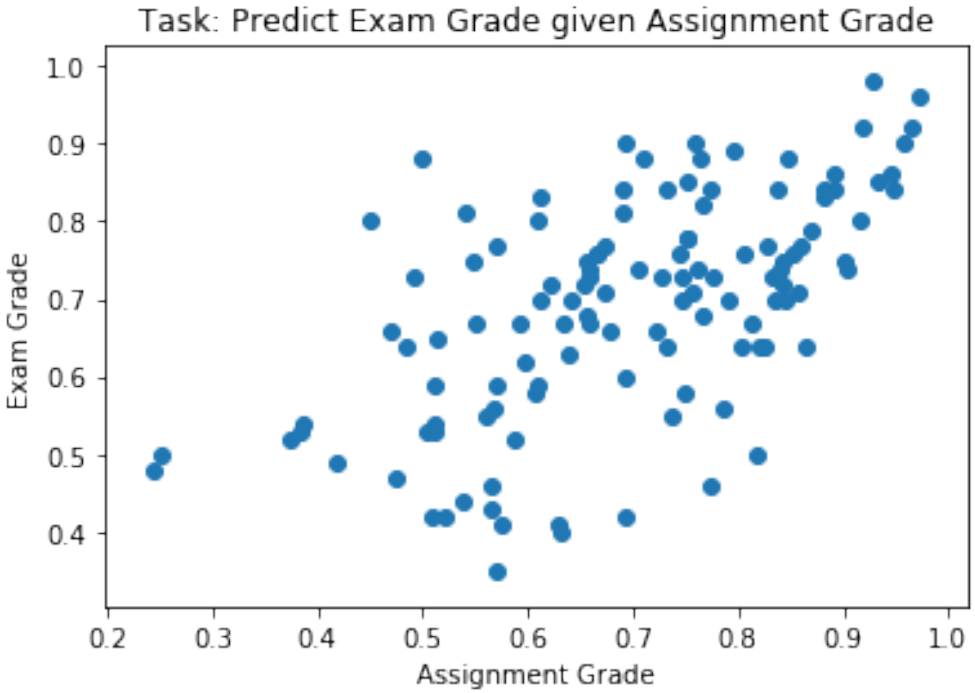
Input: Represented using the vector $\mathbf{x}$

- Example: $\mathbf{x}$ represents assignment grades (0-100)
- To start, let's assume that $\mathbf{x}$ is a scalar, and that we only have the cumulative assignment grade

Output: Represented using the scalar t

- Example: t represents the grade on an exam (0-100)
- We'll use the scalar y to denote a *prediction* of the value of t

Example: Exam Grade Prediction



- Data: $(\mathbf{x}^{(1)}, t^{(1)}), (\mathbf{x}^{(2)}, t^{(2)}), \ldots \ (\mathbf{x}^{(N)}, t^{(N)})$
- The $\mathbf{x}^{(i)}$ are called *inputs*
- The $t^{(i)}$ are called *targets*

Regression with a Linear Model

- A **model** implicitly or explicitly encodes our assumptions about the underlying nature of the data we wish to learn about.
- But recall the adage that *All models are wrong, but some are useful*.
  - In ML, we often choose models without really explicitly thinking about our assumptions about the data generation process.
- The **model**, or **architecture** defines the set of allowed family of **hypotheses**.

In linear regression, our **model** looks like

$$y = \sum_j w_j x_j + b,$$

where y is a prediction for t, and the w_j and b are **parameters** of the model, to be determined based on the data.

Linear Regression for Exam Grade Prediction

For the exam prediction problem, we only have a single feature, so we can simplify our model to:

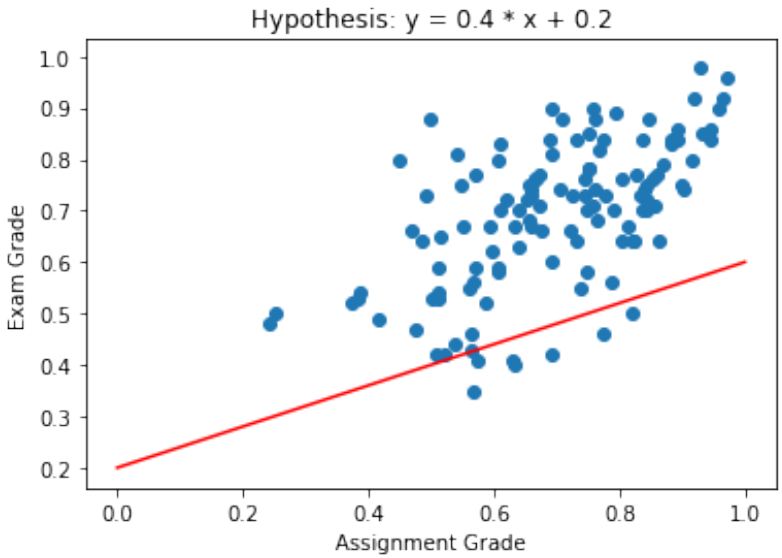
$$y = wx + b$$

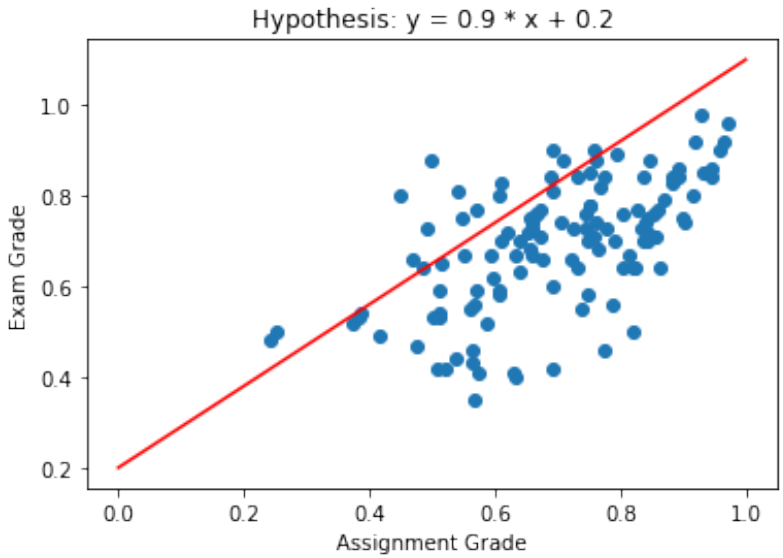
Our **hypothesis space** includes all functions of the form $y = wx + b$. Here are some examples:

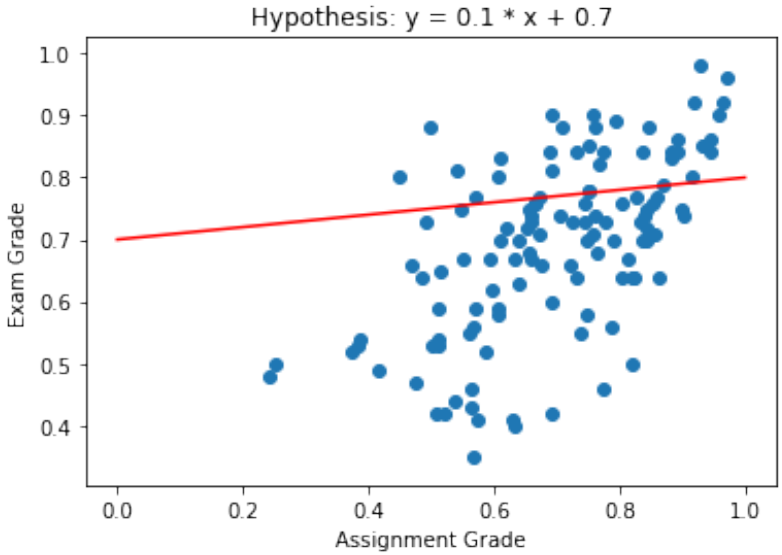
- $y = 0.4x + 0.2$
- $y = 0.9x + 0.2$
- $y = 0.1x + 0.7$
- $y = -x - 1$
- ...

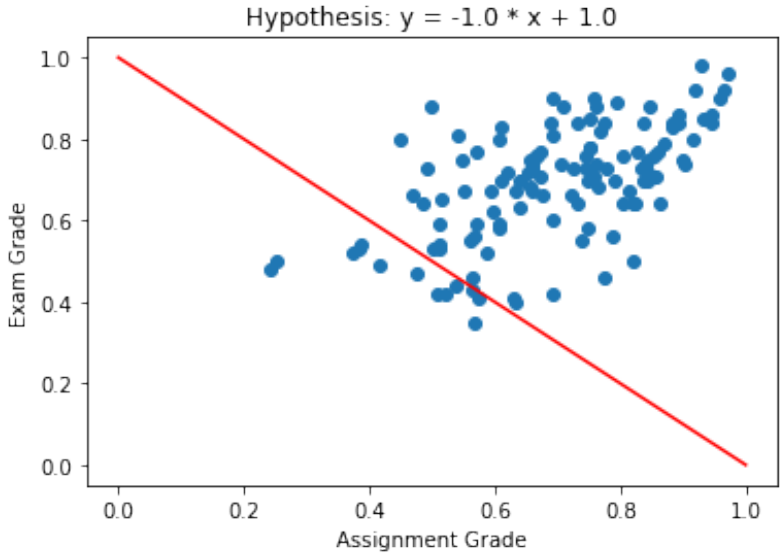
The variables w and b are called **weights** or **parameters** of our model. (Sometimes w and b are referred to as coefficients and intercept/bias, respectively.)

Which hypothesis is better suited to the data?



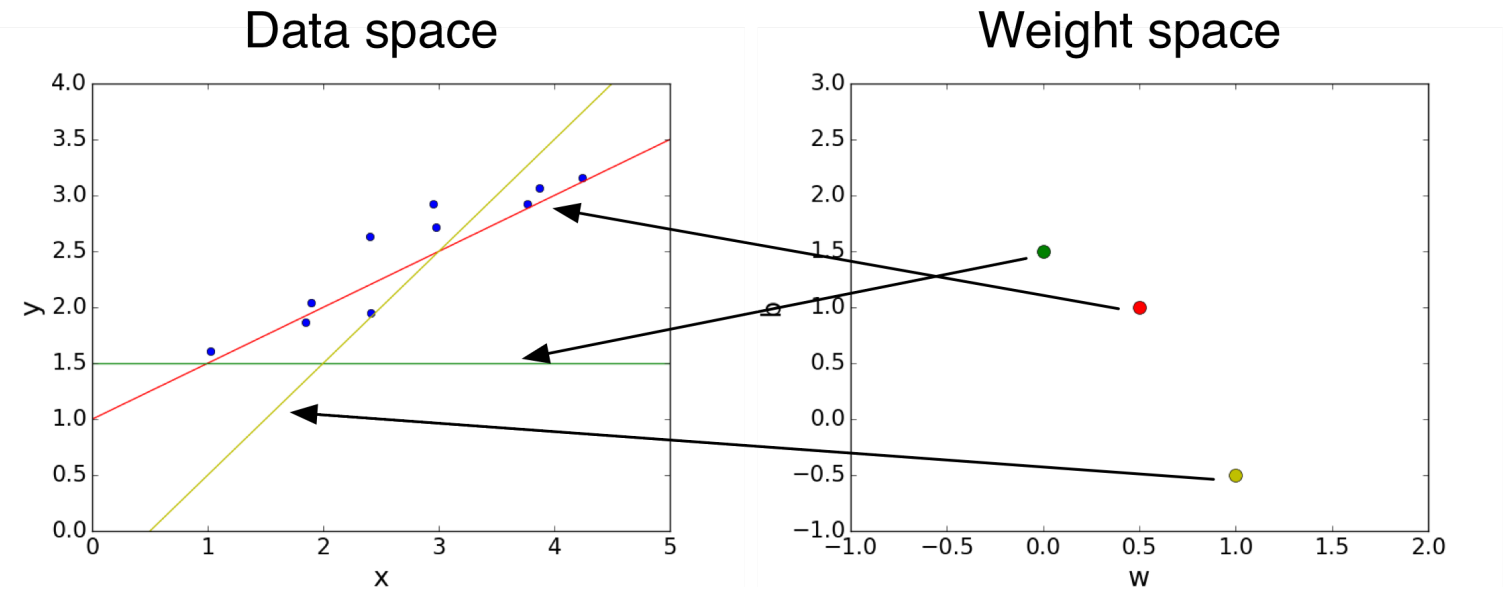






Hypothesis Space

We can visualize the hypothesis space or **weight space**:



Each *point* in the weight space represents a hypothesis.

Cost Function (Loss Function)

The "badness" of an entire hypothesis is the average badness across our labeled data.

$$\begin{aligned}\mathcal{E}(w, b) &= \frac{1}{N} \sum_i \mathcal{L}(y^{(i)}, t^{(i)}) \\ &= \frac{1}{2N} \sum_i (y^{(i)} - t^{(i)})^2 \\ &= \frac{1}{2N} \sum_i ((wx^{(i)} + b) - t^{(i)})^2\end{aligned}$$

This is called the **cost** of a particular hypothesis (in practice, "loss" and "cost" functions are used inter-changeably).

Since the loss depends on the choice of w and b, we call $\mathcal{E}(w, b)$ the **cost function**.

Minimize Cost: Direct Solution

Find a *critical point* by setting

$$\frac{\partial \mathcal{E}}{\partial w} = 0$$
$$\frac{\partial \mathcal{E}}{\partial b} = 0$$

Possible for our hypothesis space, and covered in the notes.

However, let's use a technique that can also be applied to more general models.

Minimize Cost: Gradient Descent

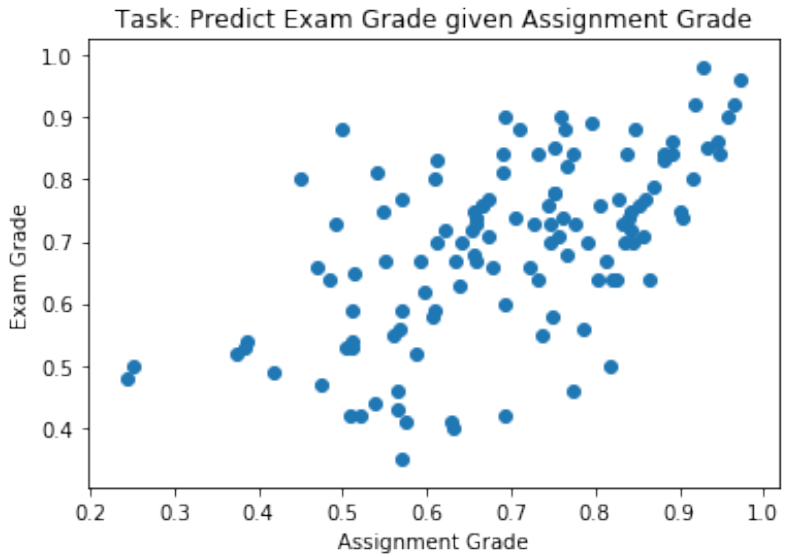
We can use gradient descent to minimize the cost function.

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{E}}{\partial \mathbf{w}}$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial \mathcal{E}}{\partial w_1} \\ \dots \\ \frac{\partial \mathcal{E}}{\partial w_D} \end{bmatrix}$$

The α is the **learning rate**, which we choose.

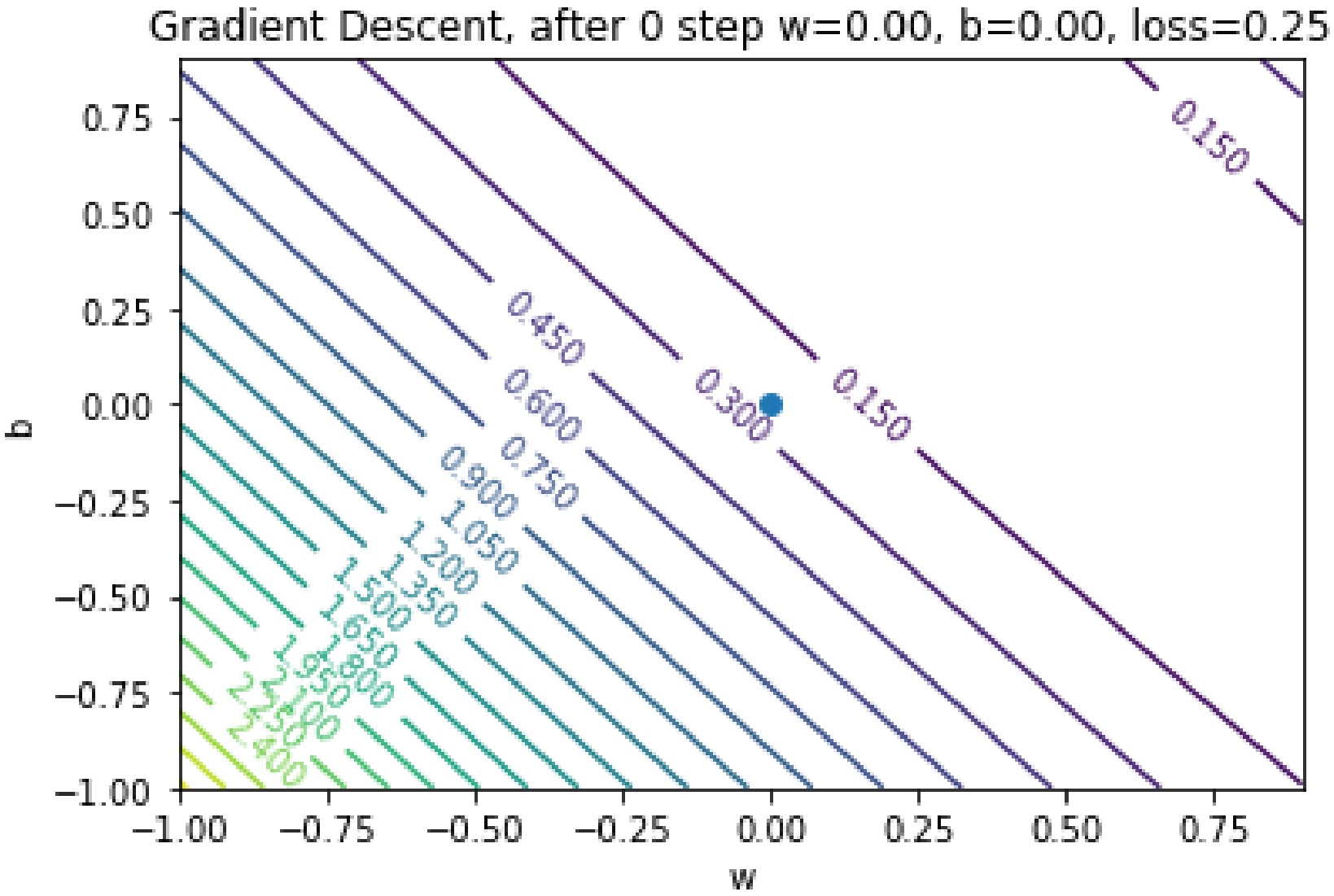
Gradient Descent for Grade Prediction



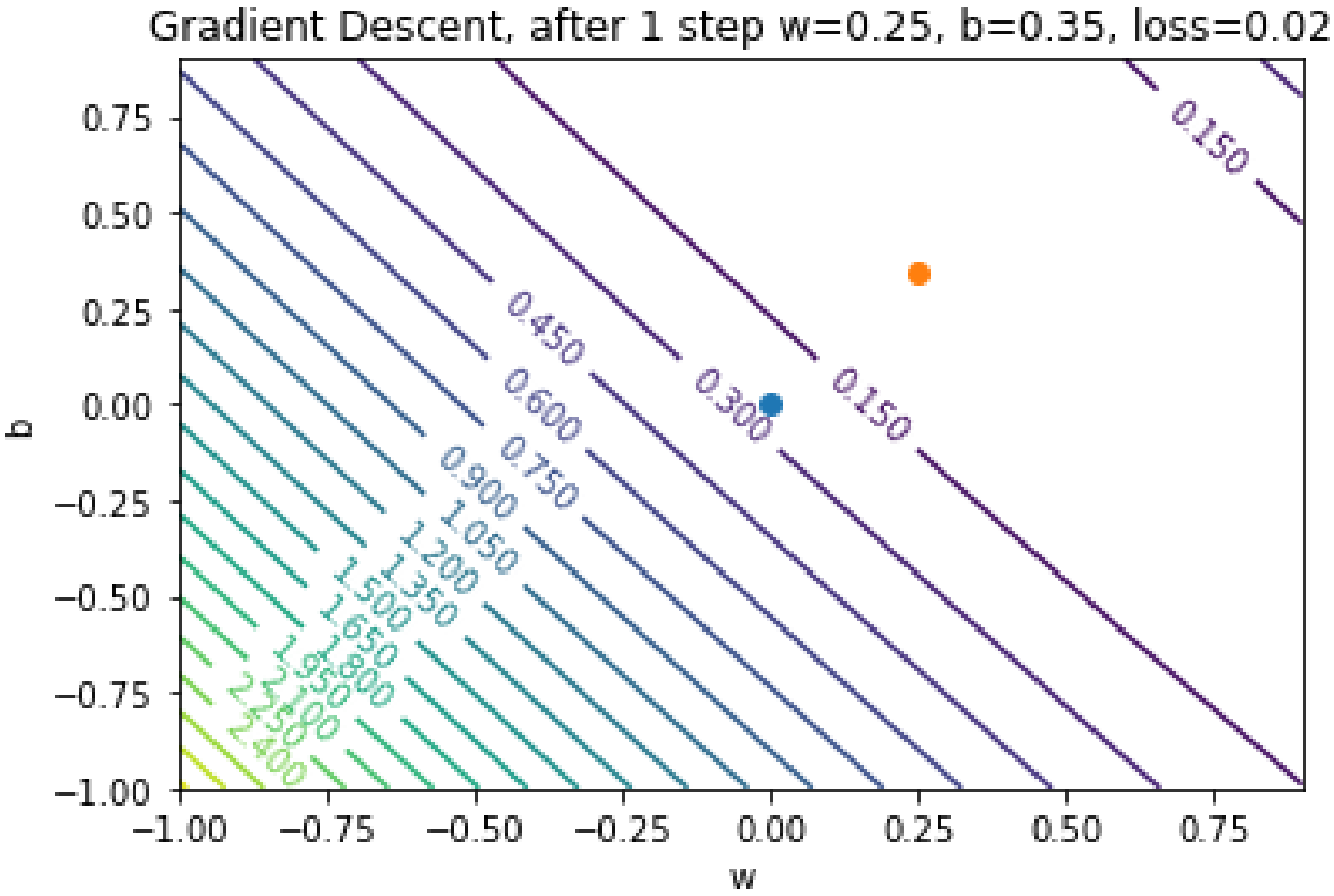
We'll initialize $w = 0$ and $b = 0$ (arbitrary choice)

We'll also choose $\alpha = 0.5$

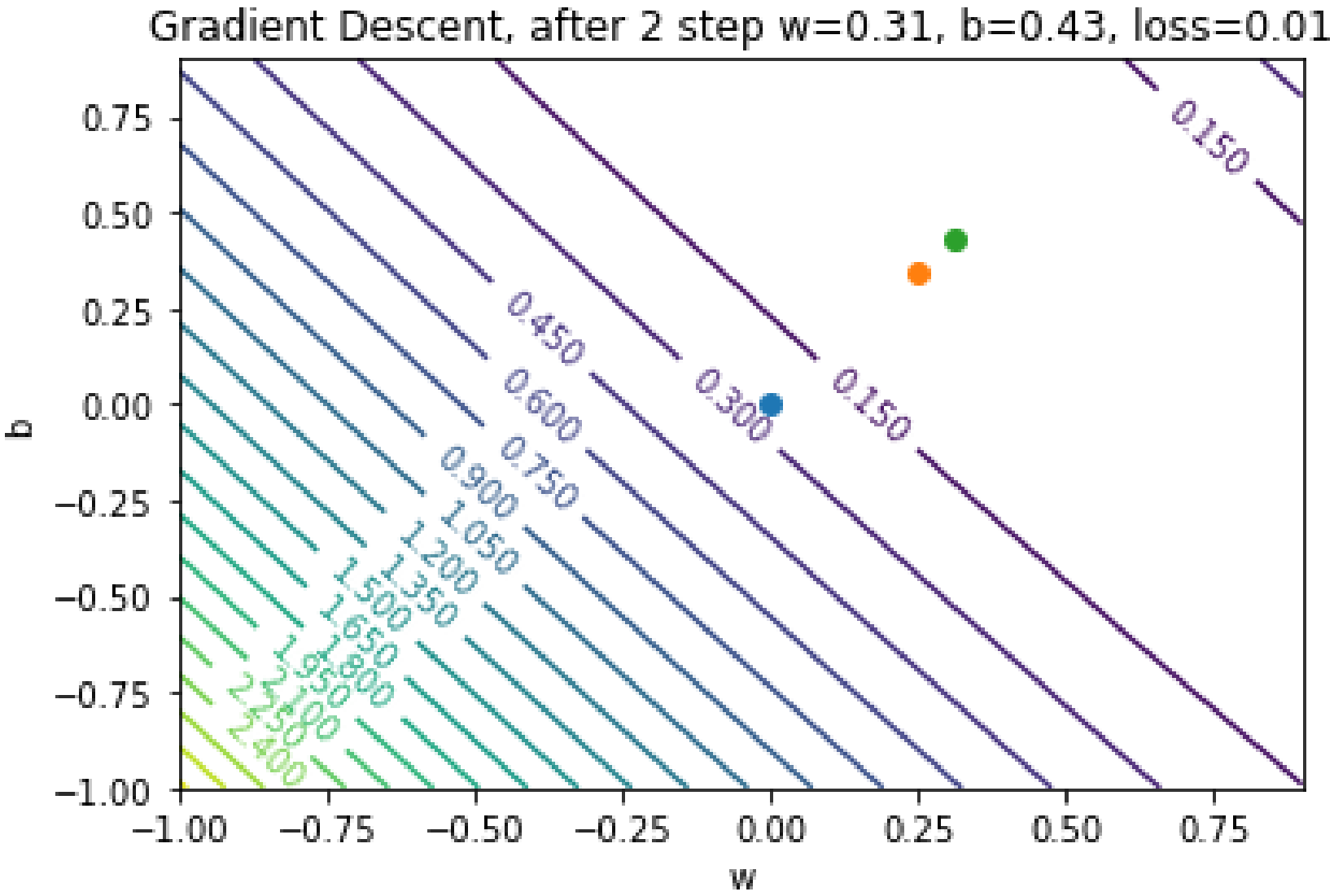
Gradient Descent: Step 0



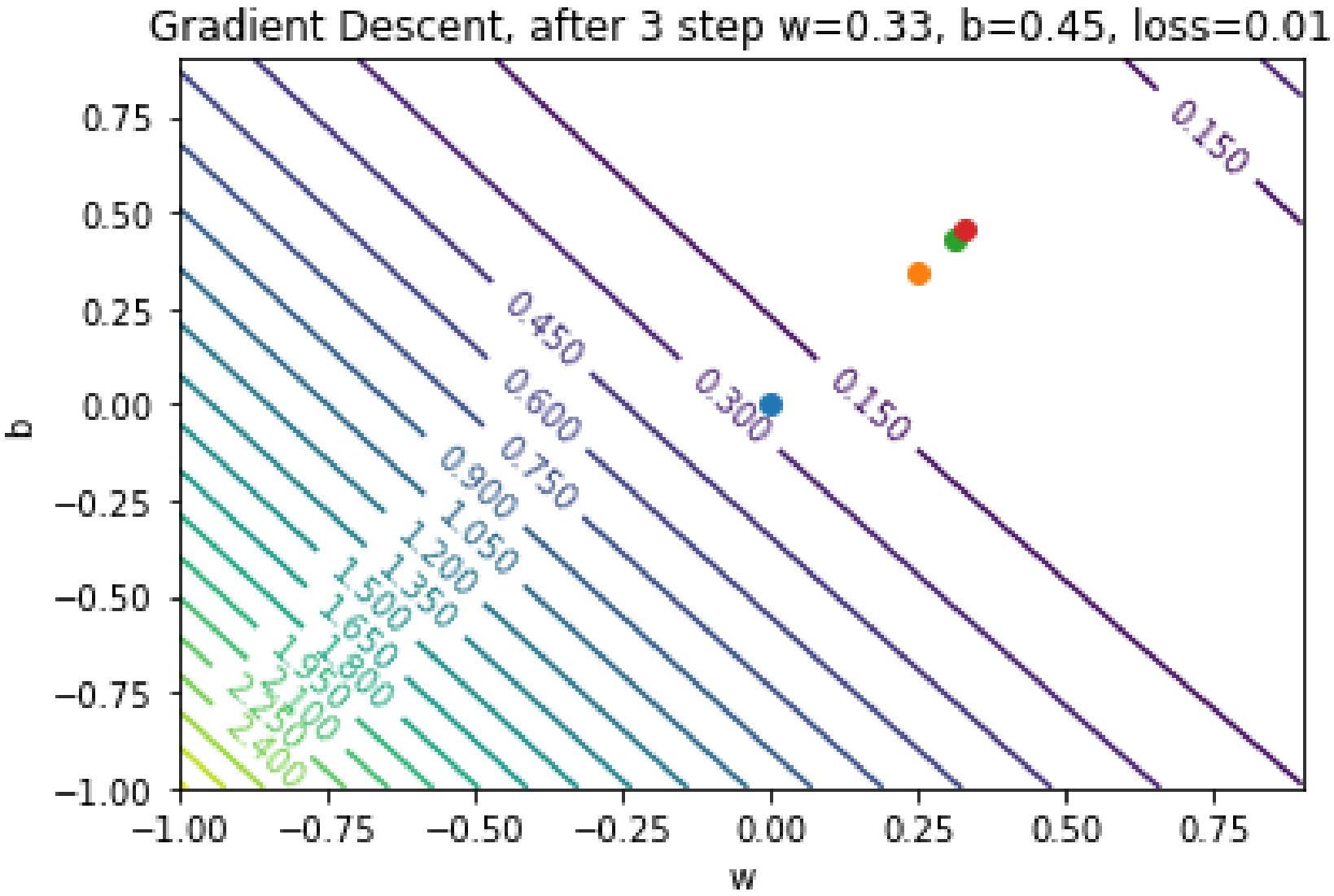
Gradient Descent: Step 1



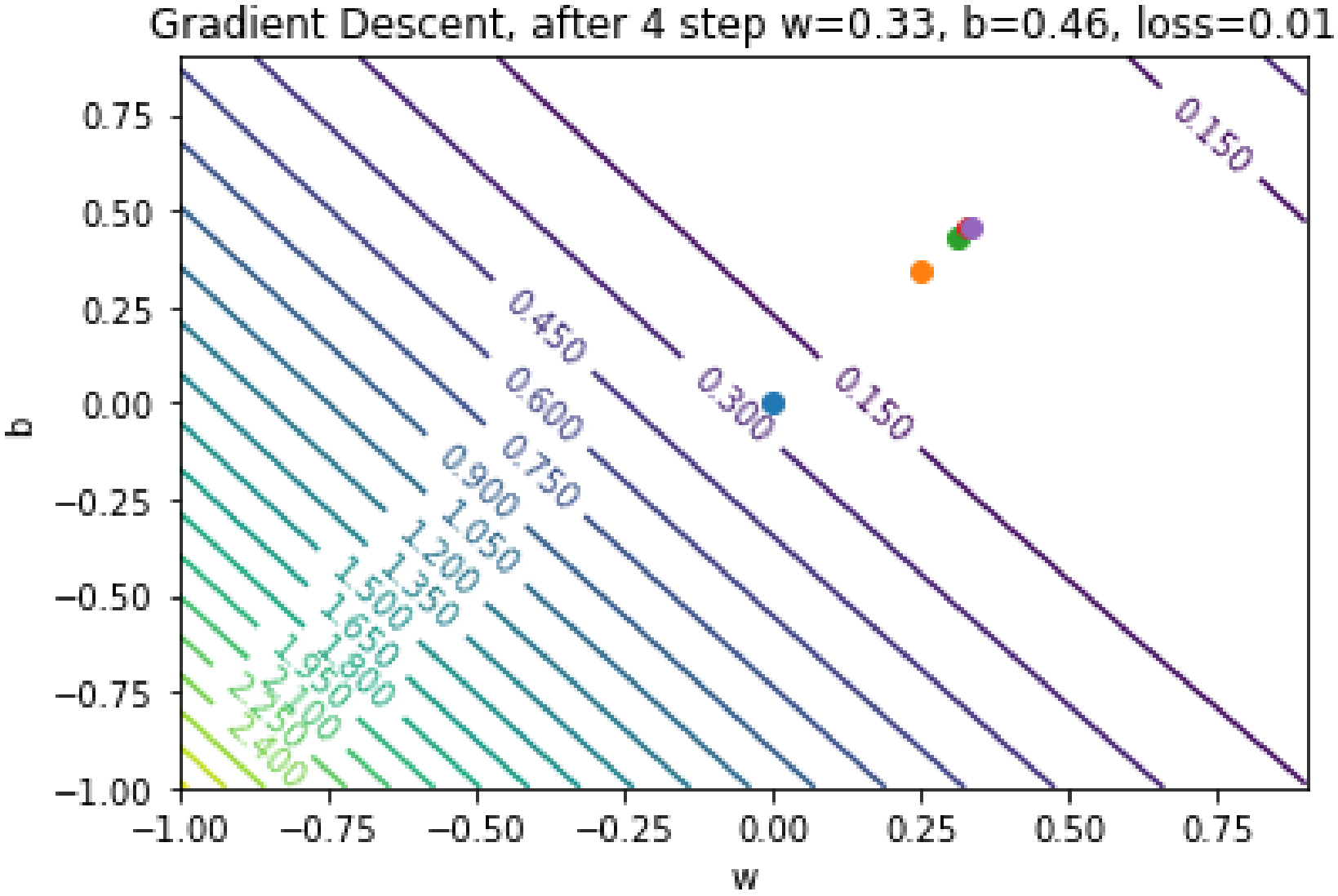
Gradient Descent: Step 2



Gradient Descent: Step 3



Gradient Descent: Step 4



Gradient Descent: When to Stop?

In theory:

- Stop when w and b stop changing (convergence)

In practice:

- Stop when $\mathcal{E}$ "almost" stops changing (approximate convergence)
- Stop until we run out of our computational budget or get tired of waiting more

Gradient Descent: How to Choose the Learning Rate?

- If α is too small, then training will be *slow*
  - Take a long time to (approximately) converge
- If α is too large, then we can have divergence!
  - It does not converge

Computing the Gradient

To compute the gradient $\frac{\partial \mathcal{E}}{\partial w}$

$$\frac{\partial \mathcal{E}}{\partial w} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{L}(y^{(i)}, t^{(i)})}{\partial w}$$

But this computation can be expensive if N is large!

Computing the Gradient

To compute the gradient $\frac{\partial \mathcal{E}}{\partial w}$

$$\frac{\partial \mathcal{E}}{\partial w} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{L}(y^{(i)}, t^{(i)})}{\partial w}$$

But this computation can be expensive if N is large!

Solution: estimate $\frac{\partial \mathcal{E}}{\partial w}$ using a *subset* of the data

Stochastic Gradient Descent

Full batch gradient descent:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{\partial \mathcal{L}(y^{(i)}, t^{(i)})}{\partial w}$$

Stochastic Gradient Descent:

Estimate the above quantity by computing the average of $\frac{\partial \mathcal{L}(y^{(i)}, t^{(i)})}{\partial w}$ across a small number of i's

The set of examples that we use to estimate the gradient is called a **mini-batch**.

The number of examples in each mini-batch is called the **mini-batch size** or just the **batch size**

Stochastic Gradient Descent Algorithm

In theory, any way of sampling a mini-batch is okay.

In practice, SGD is almost always implemented like this:

```
# repeat until convergence:
    # randomly split the data into mini-batches of size k
    # for each mini-batch:
        # estimate the gradient using the mini-batch
        # update the parameters based on the estimate
```

- Each pass of the inner loop is called an **iteration**.
  - One iteration = one update for each weight
- Each pass of the outer loop is called an **epoch**.
  - One epoch = one pass over the data set

Iterations, Epochs, and Batch Size

Suppose we have 1000 examples in our training set.

- Q: How many iterations are in one epoch if our batch size is 10?
- Q: How many iterations are in one epoch if our batch size is 50?

Batch size choice

- Q: What happens if the batch size is **too large**?
- Q: What happens if the batch size is **too small**?

Linear Regression Summary

Model	$y = \mathbf{w}^\top \mathbf{x} + b$
Loss Function	$\mathcal{L}(y, t) = (y - t)^2$
Optimization Method	$\min_{\mathbf{w}, b} \mathcal{E}(\mathbf{w}, b)$ via Gradient Descent
	$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{E}}{\partial \mathbf{w}}, b \leftarrow b - \alpha \frac{\partial \mathcal{E}}{\partial b}$

Linear Regression Vectorization

Use vectors rather than writing

$\mathcal{E}(\mathbf{w}, b) = \frac{1}{2N} \sum_i ((\mathbf{w}\mathbf{x}^{(i)} + b) - t^{(i)})^2$

So we have:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + b\mathbf{1}, \text{where}$$

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_D^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_D^{(2)} \\ \dots & & & \\ x_1^{(N)} & x_2^{(N)} & \dots & x_D^{(N)} \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_D \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(N)} \end{bmatrix}, \mathbf{t} = \begin{bmatrix} t^{(1)} \\ t^{(2)} \\ \dots \\ t^{(N)} \end{bmatrix}$$

(You can also fold the bias b into the weight $\mathbf{w}$, but we won't.)

Vectorized Loss Function

After vectorization, the loss function becomes:

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2N}(\mathbf{y} - \mathbf{t})^\top(\mathbf{y} - \mathbf{t})$$

or

$$\mathcal{E}(\mathbf{w}) = \frac{1}{2N}(\mathbf{X}\mathbf{w} + b\mathbf{1} - \mathbf{t})^\top(\mathbf{X}\mathbf{w} + b\mathbf{1} - \mathbf{t})$$

Vectorized Gradient Descent

$$b \leftarrow b - \alpha \frac{\partial \mathcal{E}}{\partial b}$$
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{E}}{\partial \mathbf{w}}$$

Where $\frac{\partial \mathcal{E}}{\partial \mathbf{w}}$ is the vector of partial derivatives:

$$\frac{\partial \mathcal{E}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial \mathcal{E}}{\partial w_1} \\ \dots \\ \frac{\partial \mathcal{E}}{\partial w_D} \end{bmatrix}$$

Why vectorize?

Vectorization is *not* just for mathematical elegance.

When using Python with numpy/PyTorch/Tensorflow/JAX, code that performs vector computations is faster than code that loops.

Same holds for many other high level languages and software.

Problem Setup: Classification

- Data: $(x^{(1)}, t^{(1)})$, $(x^{(2)}, t^{(2)})$, $\ldots$ $(x^{(N)}, t^{(N)})$
- The $x^{(i)}$ are called *inputs*
- The $t^{(i)}$ are called *targets*

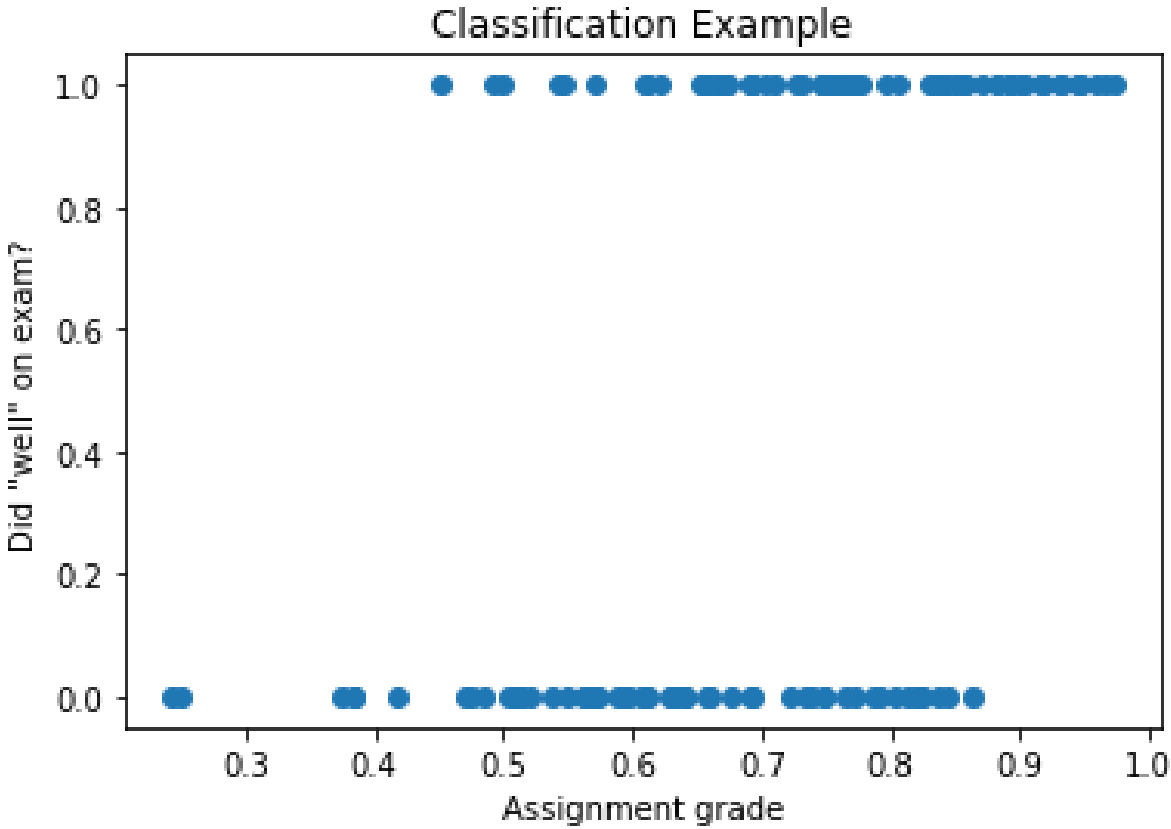
In classification, the $t^{(i)}$ are discrete.

In binary classification, we'll use the labels $t \in \{0, 1\}$ (or $\{-1, +1\}$). Training examples with

- $t = 1$ is called a **positive example**
- $t = 0$ is called a **negative example**

Classification Example

- $x^{(i)}$ represents a person's assignment grade
- $t^{(i)}$ represents whether that person had a "high" exam grade (arbitrary cutoff)



Q: Why not use regression?

Why can't we set up this problem as a regression problem?

Use the model:

$$y = wx + b$$

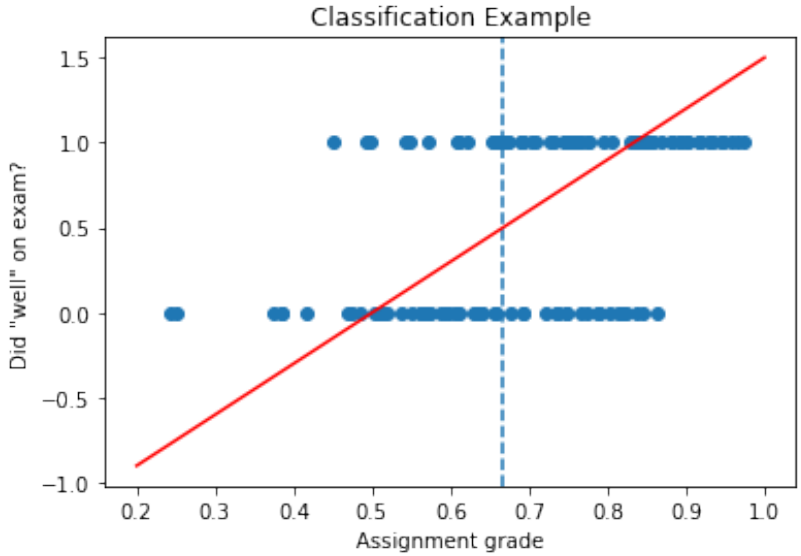
Our prediction for t would be 1 if $y >= 0.5$, and 0 otherwise.

With the loss function

$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

And minimize the cost function via gradient descent?

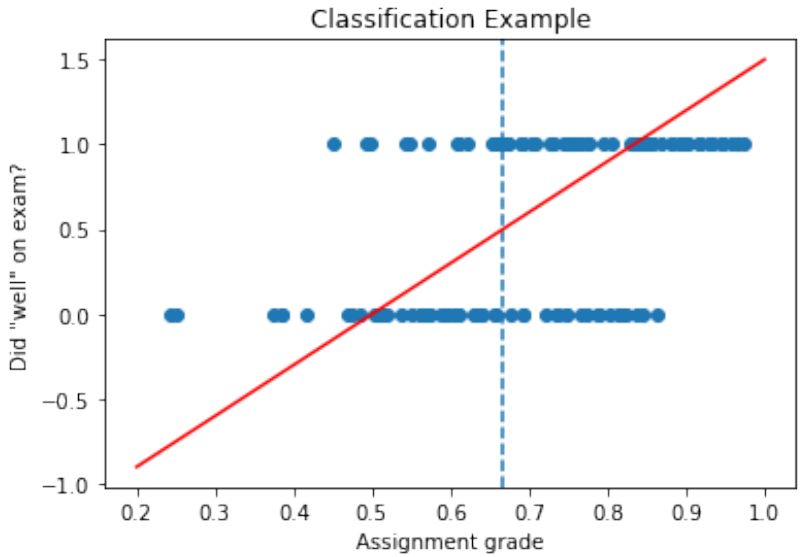
Classification as Regression: Problem



If we have $\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$, then **points that are correctly classified will still have high loss**!

(blue dotted line above = decision boundary)

The Problem (continued)



Example: a point on the top right

- Model makes the correct prediction for point on top right
- However, $(y - t)^2$ is large
- So we are penalizing our model, even though it is making the right prediction!

Q: Why not use classification error?

Why not still use the model:

$$y = \begin{cases} 1 & \text{if } \mathbf{w}^\top \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

But use this loss function instead:

$$\mathcal{L}(y, t) = \begin{cases} 0 & \text{if } y = t \\ 1 & \text{otherwise} \end{cases}$$

First attempt at a loss function: the 0-1 loss

$$\mathcal{L}(y, t) = \begin{cases} 0 & \text{if } y = t \\ 1 & \text{otherwise} \end{cases}$$

The gradient of this function is 0 almost everywhere!

So gradient descent will not change the weights! We need to define a surrogate loss function that is better behaved.

Logistic Model

Apply a **non-linearity** or **activation function** to the linear model z:

$$z = wx + b \qquad \text{also called the logit}$$
$$y = \sigma(z) \qquad \text{also called a log-linear model}$$

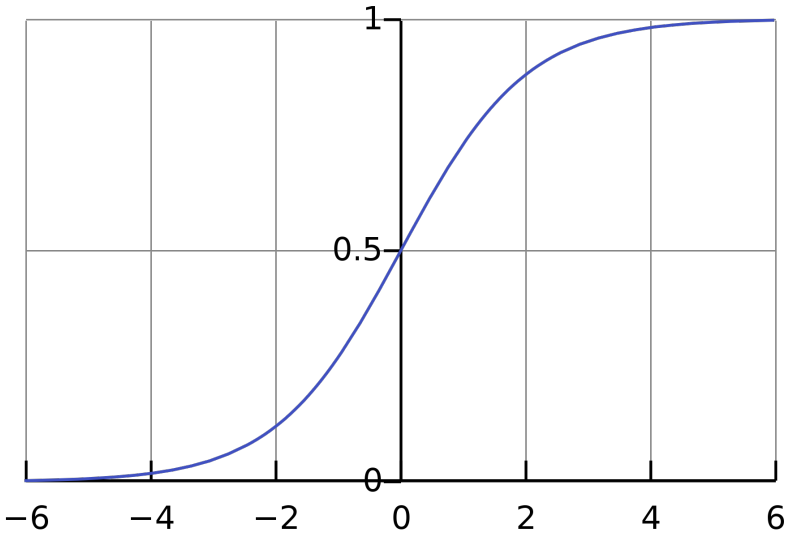
where

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

is called the logistic or sigmoid function.

The Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

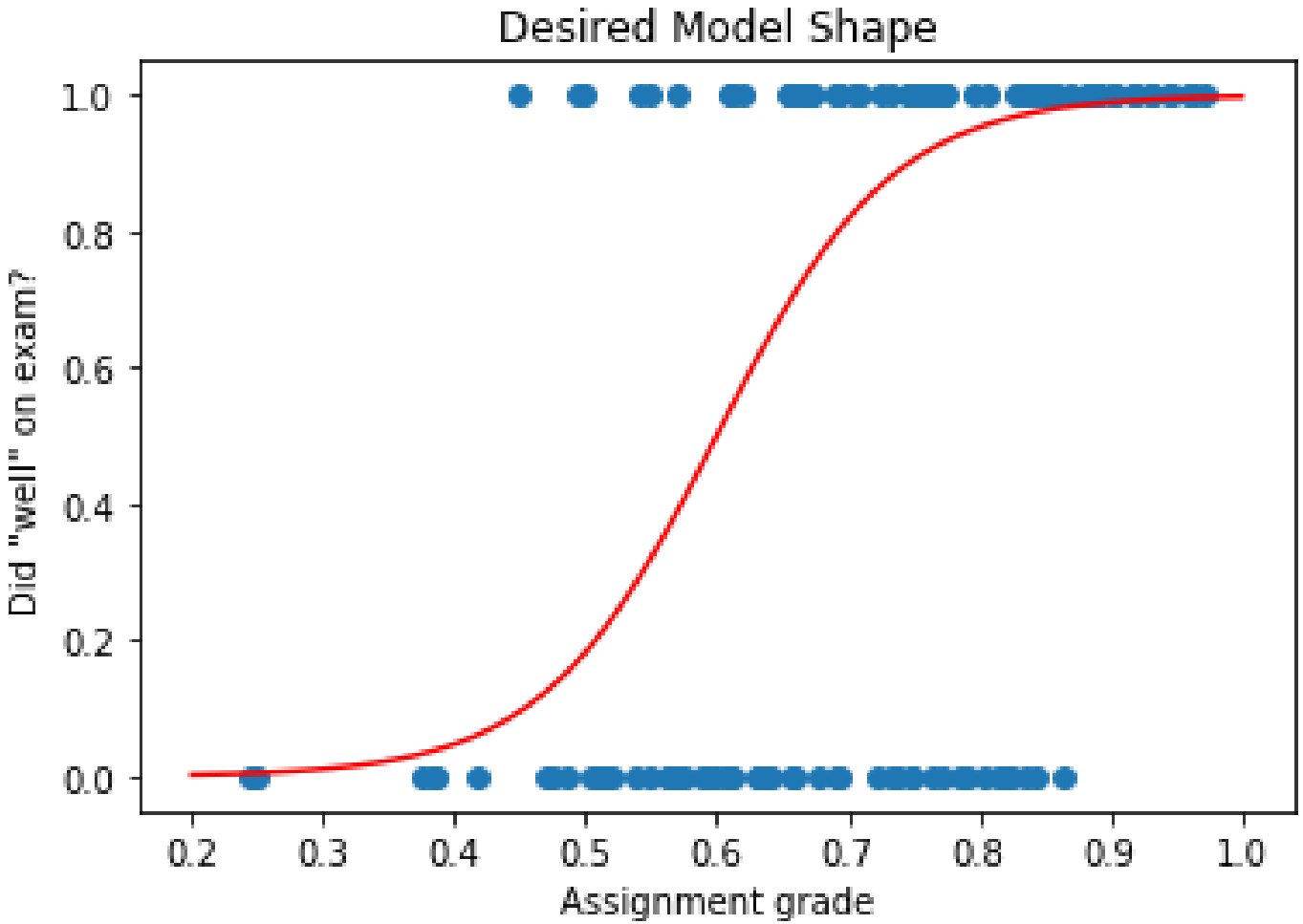


Properties:

- $\sigma(z)$ is between 0 and 1
- $\sigma(0)$ is 0.5

Logistic Regression Example

A logistic model has this shape:



But how do we train this model?

Logistic Model with Squared Error Loss?

Suppose we define the model like this:

$$
\begin{aligned}
z &= wx + b \\
y &= \sigma(z) \\
\mathcal{L}_{SE}(y, t) &= \frac{1}{2}(y - t)^2
\end{aligned}
$$

The gradient of $\mathcal{L}$ with respect to w is (homework):

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial \mathcal{L}}{\partial y} \frac{dy}{dz} \frac{\partial z}{\partial w} \\
&= (y - t)y(1 - y)x
\end{aligned}
$$

The Problem of Squared Error Loss

Suppose we have a positive example ($t = 1$) that our model classifies extremely wrongly ($z = -5$):

Then we have $y = \sigma(z) \approx 0.0067$

The Problem of Squared Error Loss

Suppose we have a positive example ($t = 1$) that our model classifies extremely wrongly ($z = -5$):

Then we have $y = \sigma(z) \approx 0.0067$

Ideally, the *gradient* should give us strong signals regarding how to update w to do better.

But... $\frac{\partial \mathcal{L}}{\partial w} = (y - t)y(1 - y)x$ is small!

The Problem of Squared Error Loss

Suppose we have a positive example ($t = 1$) that our model classifies extremely wrongly ($z = -5$):

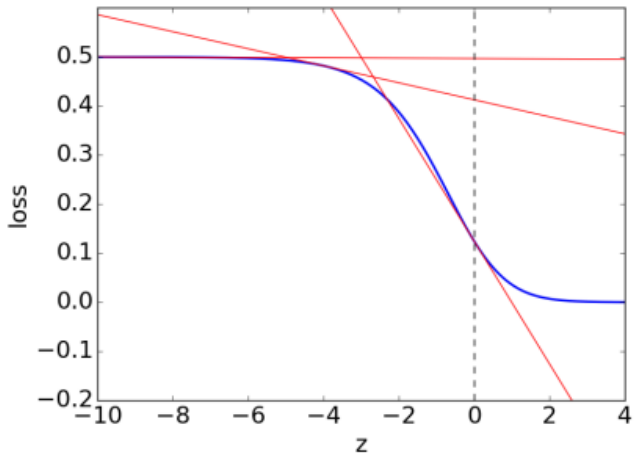
Then we have $y = \sigma(z) \approx 0.0067$

Ideally, the *gradient* should give us strong signals regarding how to update w to do better.

But... $\frac{\partial \mathcal{L}}{\partial w} = (y - t)y(1 - y)x$ is small!

Which means that the update $w \leftarrow w - \alpha \frac{\partial \mathcal{L}}{\partial w}$ won't change w much!

Gradient Signal



The problem with using sigmoid activation with square loss is that we get **poor gradient signal**.

- The loss for a *very wrong prediction* ($y = 0.0001$) vs a wrong prediction ($y = 0.01$) are similar
- This is a problem, because the gradients in the region would be close to 0

We need a loss function that distinguishes between a wrong prediction and a *very* wrong prediction.

The Cross Entropy Loss

The **cross entropy loss** provides the desired behaviour:

$$\mathcal{L}(y, t) = \begin{cases} -\log(y) & \text{if } t = 1 \\ -\log(1-y) & \text{if } t = 0 \end{cases}$$

We can write the loss as:

$$\mathcal{L}(y, t) = -t \log(y) - (1-t) \log(1-y)$$

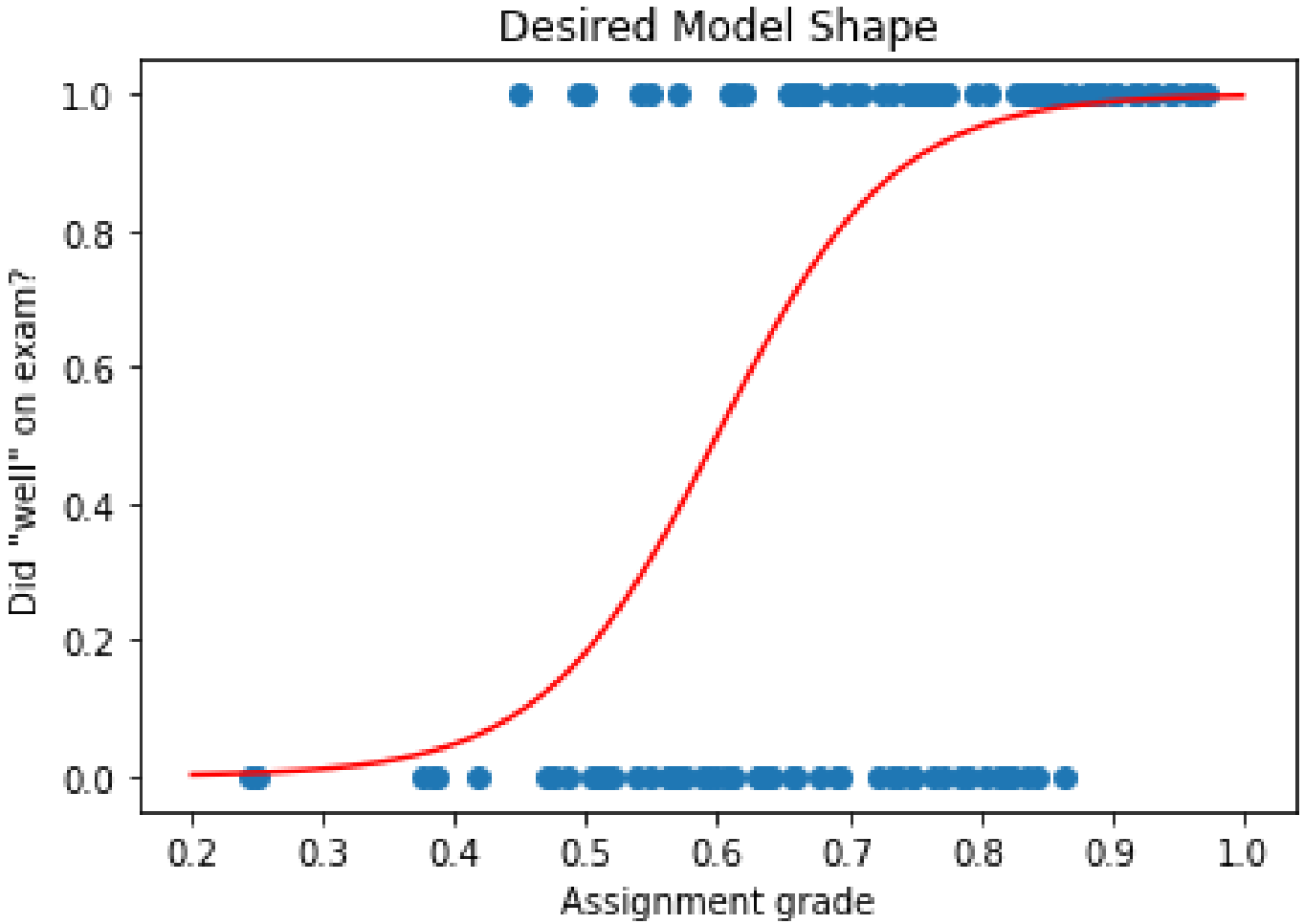
We we use the Logistic Model with cross entropy loss, the resulting model is called **Logistic Regression** model. Note that it is a classification method and not a regression one, as we use it in this course.

Logistic Regression Summary

Model	$y = \sigma(\mathbf{w}^\top \mathbf{x} + b)$
Loss Function	$\mathcal{L}(y, t) = -t \log(y) - (1 - t) \log(1 - y)$
Optimization Method	$\min_{\mathbf{w}, b} \mathcal{E}(\mathbf{w}, b)$ via Gradient Descent
	$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{E}}{\partial \mathbf{w}}, b \leftarrow b - \alpha \frac{\partial \mathcal{E}}{\partial b}$

Grade Classification Example

After running gradient descent, we'll get a model that looks something like:



Multi-Class Classification

Instead of there being two targets (pass/fail, cancer/not cancer, before/after 2000), we have $K > 2$ targets.

Example:

- Beatles ($K = 4$):
  - John Lennon, Paul McCartney, George Harrison, Ringo Starr
- Pets ($K =$ something large):
  - cat, dog, hamster, parrot, python, ...

Representing the Targets

We use a **one-hot vector** to represent the target:

$$\mathbf{t} = (0, 0, ..., 1, ..., 0)$$

This vector contains $K - 1$ zeros, and a single 1 somewhere.

Each *index* (column) in the vector represents one of the classes.

Representing the Predictor

The prediction $\mathbf{y}$ will also be a vector. Like in logistic regression there will be a linear part, and an activation function.

Linear part: $\mathbf{z} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$

So far, this is like having K separate logistic regression models, one for each element of the one-hot vector.

Q: What are the shapes of $\mathbf{z}$, $\mathbf{W}$, $\mathbf{x}$ and $\mathbf{b}$?

Activation Function

Instead of using a *sigmoid* function, we instead use a **softmax activation** function:

$$y_k = \text{softmax}(z_1, ..., z_K)_k = \frac{e^{z_k}}{\sum_{m=1}^{K} e^{z_m}}$$

The vector of predictions y_k is now a **probability distribution** over the classes!

Why Softmax?

- Softmax is like the multi-class equivalent of sigmoid
- Softmax is a continuous analog of the "argmax" function
- If one of the z_k is much larger than the other, then the softmax will be approximately the argmax, in the one-hot encoding

Cross-Entropy Loss

The cross-entropy loss naturally generalizes to the multi-class case:

$$\begin{aligned}\mathcal{L}(\mathbf{y}, \mathbf{t}) &= -\sum_{k=1}^{K} t_k \log(y_k) \\ &= -\mathbf{t}^\top \log(\mathbf{y})\end{aligned}$$

Recall that only one of the t_k is going to be 1, and the rest are 0.

Multi-class Classification Summary

Model	$\mathbf{y} = \text{softmax}(\mathbf{W}^\top \mathbf{x} + \mathbf{b})$
Loss Function	$\mathcal{L}(\mathbf{y}, \mathbf{t}) = -\mathbf{t}^\top \log(\mathbf{y})$
Optimization Method	Minimize $\mathcal{E}(\mathbf{w}, b)$ via Gradient Descent
	$\mathbf{W} \leftarrow \mathbf{W} - \alpha \frac{\partial \mathcal{E}}{\partial \mathbf{W}}, \mathbf{b} \leftarrow \mathbf{b} - \alpha \frac{\partial \mathcal{E}}{\partial \mathbf{b}}$

Example: Beatle Recognition

Given a 100x100 pixel colour image of a face of a Beatle, identify the Beatle

Four possible labels:

- John Lennon
- Paul McCartney
- George Harrison
- Ringo Starr

Side Note: Representing an Image

This is what John Lennon looks like to a computer:

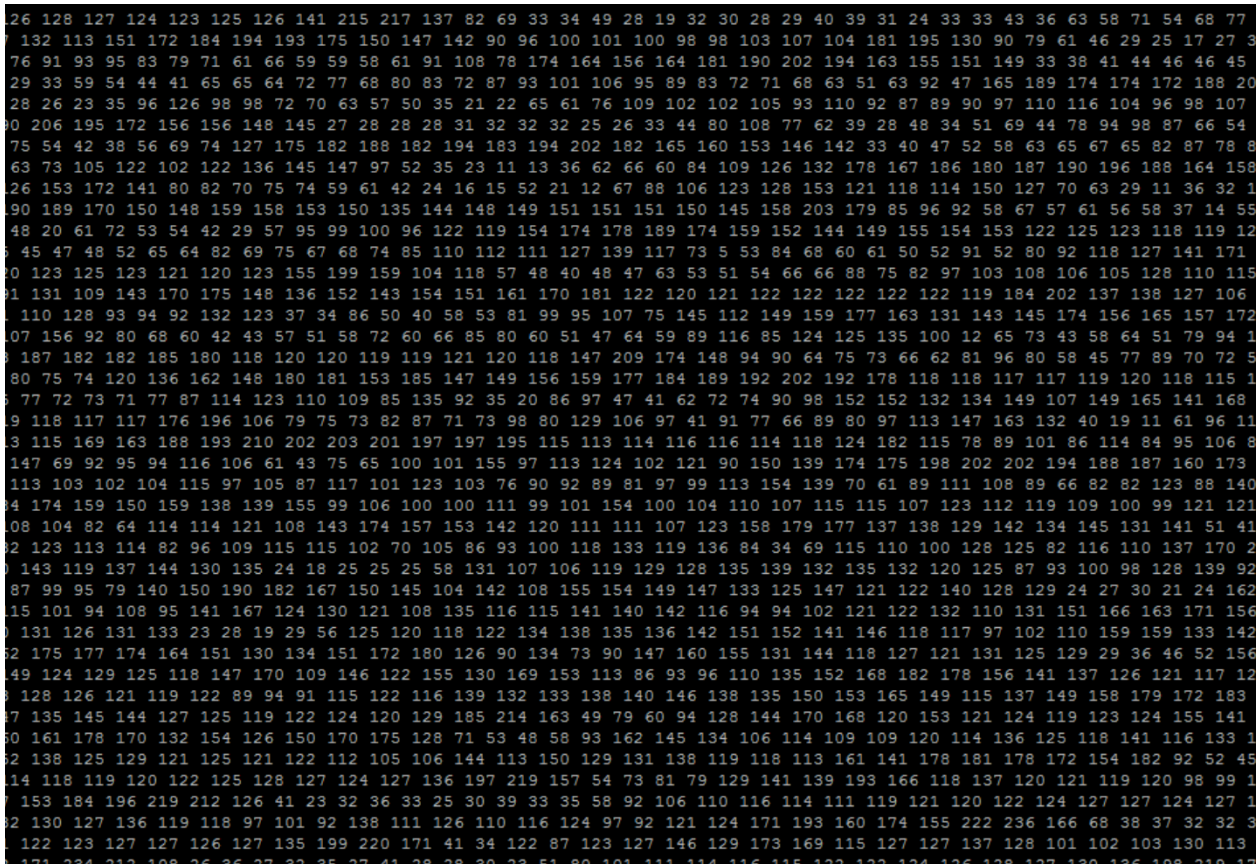
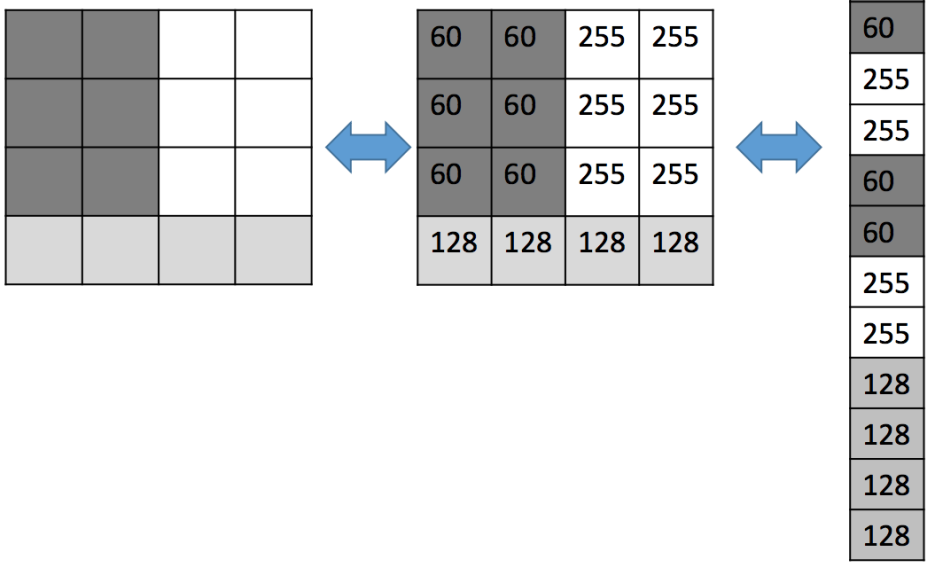


Image as a Vector of Features

Images ⟺ Vectors



Features and Targets

Each of our input images are 100×100 pixels

$\mathbf{y} = \text{softmax}(\mathbf{W}^\top \mathbf{x} + \mathbf{b})$

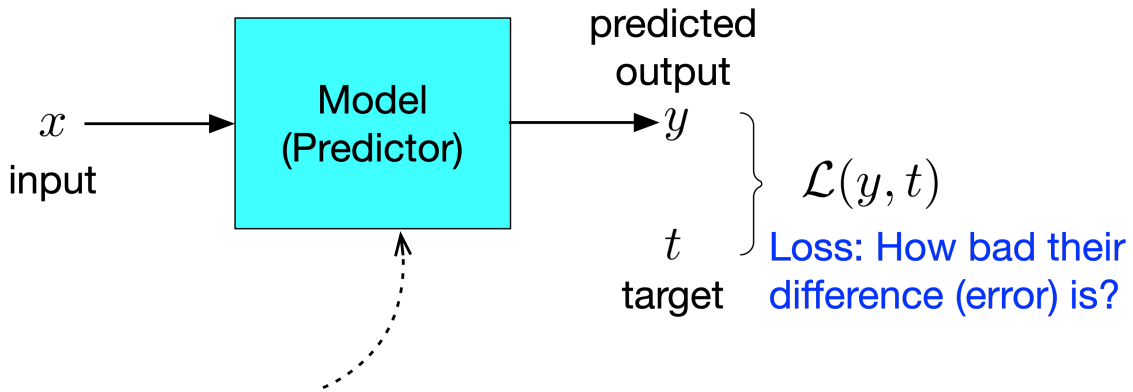
Questions:

- What will be the length of our input (feature) vectors $\mathbf{x}$?
- What will be the length of our one-hot targets $\mathbf{t}$?
- What are the shapes of $\mathbf{W}$ and $\mathbf{b}$?
- How many (scalar) parameters are in our model, in total?

Section 4

Deep Learning

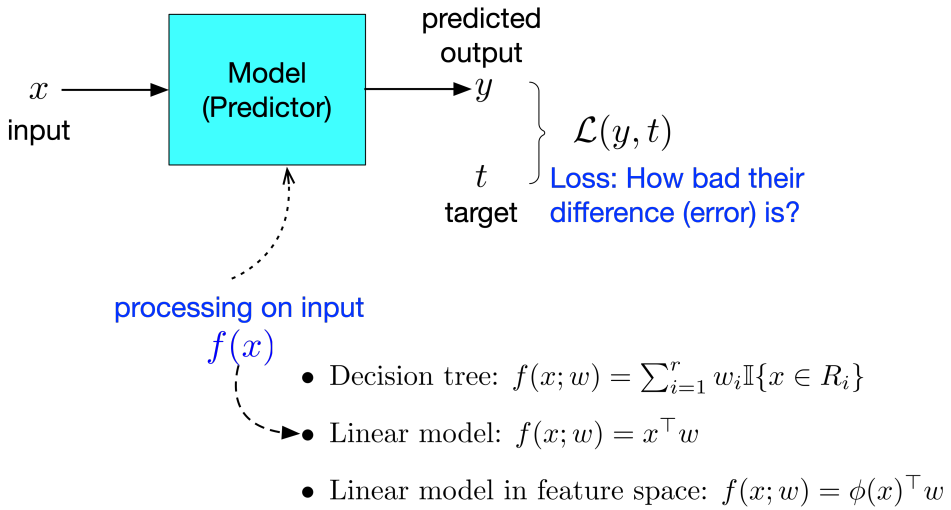
From Linear Models to Neural Networks



- Decision tree: $f(x;w) = \sum_{i=1}^{r} w_i \mathbb{I}\{x \in R_i\}$
- Linear model: $f(x;w) = x^\top w$
- Linear model in feature space: $f(x;w) = \phi(x)^\top w$

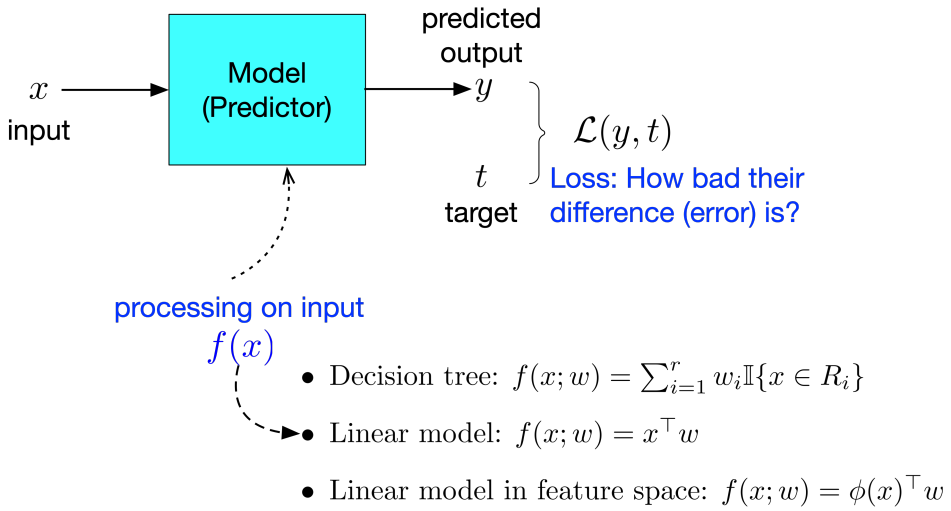
- Design of an ML system follows a modular approach. We have to make choice on
  - Model
  - Loss function
  - Regularizer, etc.
- In your Introduction to ML course (ex. CSC311), you have seen many examples.

Beyond Linear Models



- Feature mapping can make linear models much more powerful.
- Coming up with feature mapping can be challenging.
- Kernel-based approach is a way to partially address it.
- (Artificial) Neural Networks (NN) is a general approach to represent complex models.

What is a Model After All?



- The predictor can be seen as a **computer program** that processes the input in order to generate the output. Some programs are simpler, some are more complex.
- Neural networks are general and flexible ways to specify a computer program.
- Different NN architectures correspond to different ways of specifying the overall architecture of the program.
- We are going to study several different ways in this course.

What are Neural Networks? A Sneak Peak!

Neural networks are a class of models originally inspired by the brain.

- Most of the biological details aren't essential, so we use vastly simplified models of neurons.
- Nowadays we mostly think about math, statistics, etc
- Neural networks are collections of thousands (or millions) of these simple processing units that together perform useful computations

$$y = \phi(\mathbf{w}^\mathbf{T}\mathbf{x} + \mathbf{b})$$

Why Neural Networks?

- Very effective across a range of applications (vision, text, speech, medicine, robotics, etc.)
- Widely used in both academia and the tech industry
- Powerful software frameworks (PyTorch, TensorFlow, JAX, etc.) let us quickly implement sophisticated algorithms

What is Deep Learning?

- A “deep” neural network contains many “layers”.
- Later layers use the output of earlier layers as input.
- The term **deep learning** emphasizes that the neural network algorithms often involve hierarchies with many stages of processing.

Deep Learning Caveats: Interpretability

Before getting deep in studying NN and Deep Learning, it is good to know some of issues common with them.

Figure 5: from https://xkcd.com/1838/

Deep Learning Caveats: Adversarial Examples

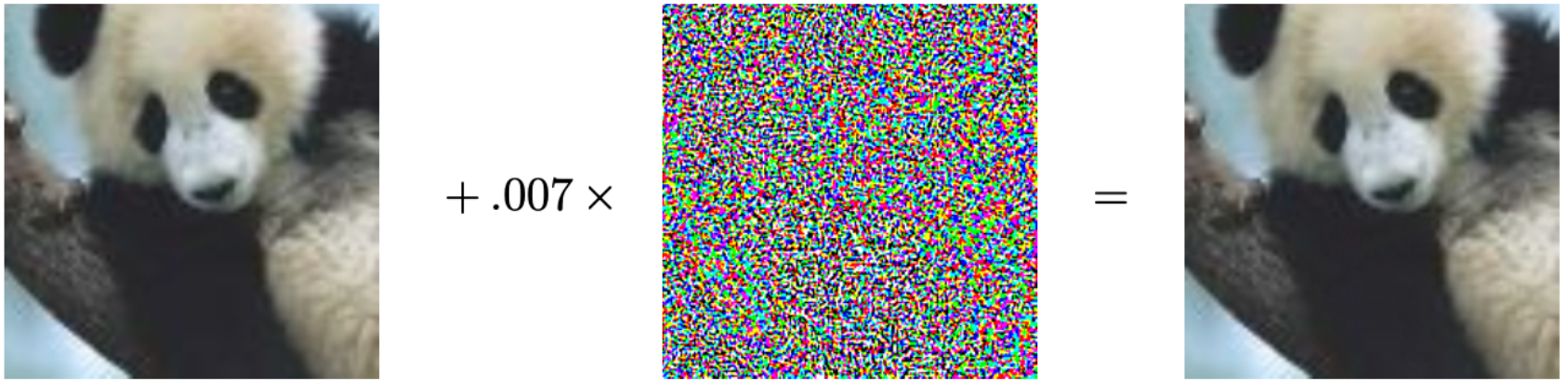


Image Credit: Goodfellow, Shlens, Szegedy, "Explaining and Harnessing Adversarial Examples," ICLR, 2015.

Deep Learning Caveats: Fairness

The U.S. military built an AI tool to find suitable combat personnel but had to shut it down because it was discriminating against women

News

Fairness in Machine Learning

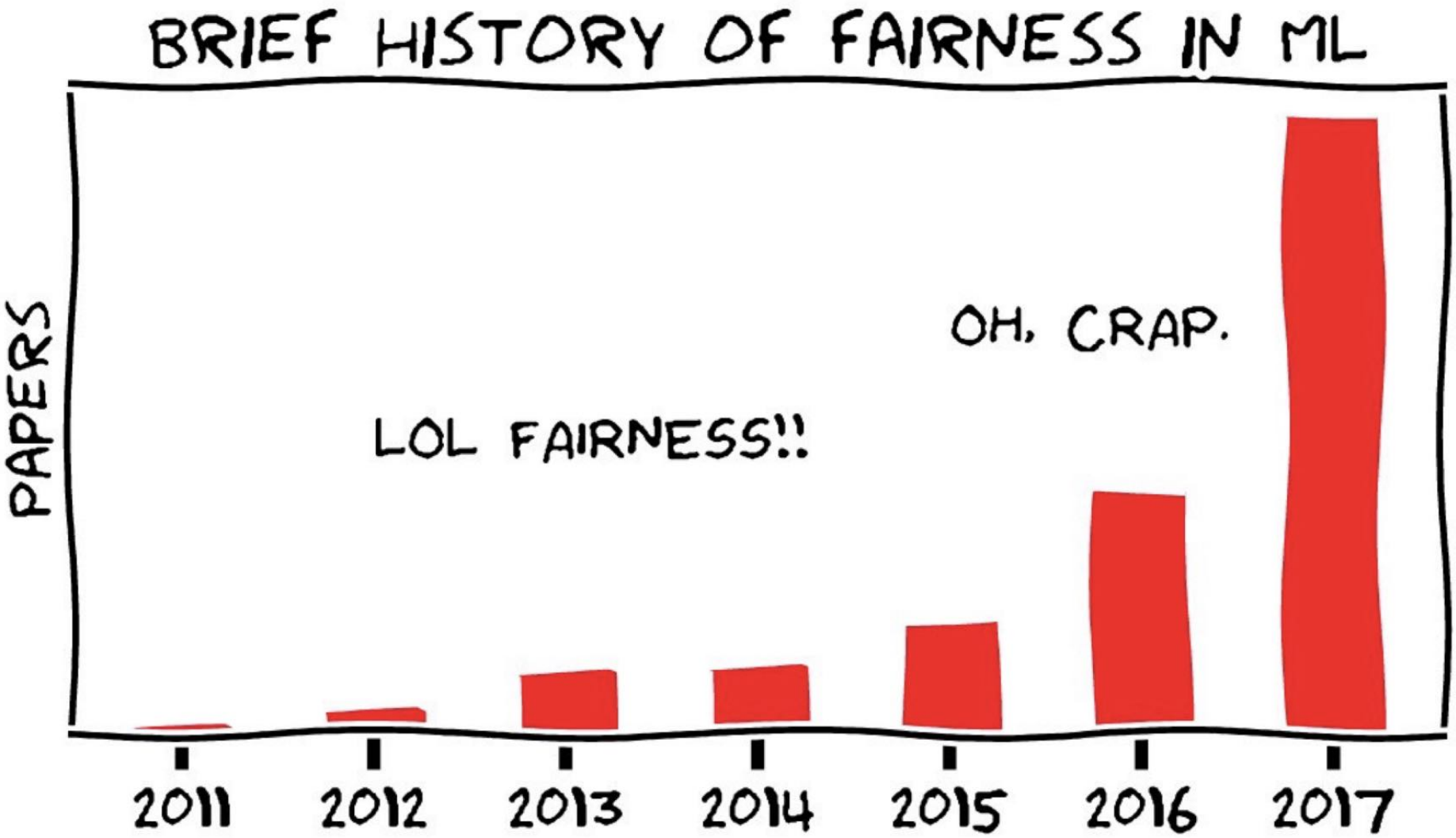


Image Credit: Solon Barocas and Moritz Hardt, "Fairness in Machine Learning", NeurIPS 2017

Course Coverage

This is the tentative schedule and may slightly change.

- Mostly supervised learning
  - Multi-layer Perceptron (fully connected feedforward)
  - Convolutional Neural Networks for images
  - Recurrent Neural Networks for sequences
- Some unsupervised learning
  - Autoencoders
  - Generative Adversarial Networks
- Special topic TBD/TBA

Section 5

Logistics

This Course

- This is the second course in machine learning, with a focus on neural networks and deep learning.
- First 75%: Mostly supervised learning
- Last 25%: Mostly unsupervised learning
- Three sections: equivalent content, different instructors, same deliverables.
- Only attend the section you are officially enrolled.
- Coursework is aimed at advanced undergrads. We will use multivariate calculus, probability, and linear algebra.

What happens every week?

- We have three sessions per week.
- Each session (usually) consists of:
  - 2h of lecture
  - 1h of tutorial/practice (bring your laptop).
  - You start working on a homework/practice assignment during the tutorial/practice session. You have until next week to submit it.
  - We sometimes may use the tutorial time for lecture.

How we communicate with you

- Course Website: **https://amfarahmand.github.io/NN-Winter2024/**
- Main source of information is the course webpage. Check regularly!
- We will also use Quercus for **announcements**.
- We will use Piazza for **discussions**.
- We will use MarkUs for **assignment submission**.

Do I have the right background?

- **Machine Learning**: linear models for regression and classification, maximum likelihood estimation, PCA, EM, etc.
- **Linear algebra**: vector/matrix manipulations, basic properties of matrices.
- **Calculus**: partial derivatives/gradient.
- **Probability**: common distributions; Bayes' Rule.
- **Statistics**: expectation, variance, covariance, median; maximum likelihood.

In the Classroom

- Feel free to ask questions from the instructors. Don't be shy! Asking questions helps you learn better. We try to answer as many questions as we can.
- Cell phones and other electronics are allowed in lecture (it might help your learning to annotate slides as we go through them).
- Talking with others is discouraged in the middle of lecture, as it would be distracting to us and others. We will have breaks so you can discuss among yourselves.
- Recording or taking pictures in class is **strictly prohibited** without the consent of your instructor. Please ask before doing!
- Even though pandemic is not in news anymore, COVID-19 and other respiratory diseases are still around. Some of your friends and peers might have weak immune system. Long COVID can be serious.
    - We encourage you to wear mask in the classroom and during in-person office hours

Accomodations

- Please refer to http://www.illnessverification.utoronto.ca in case of illness (you need to fill out an absence declaration form on ACORN and contact me).
- If you require additional academic accommodations, please contact UofT Accessibility Services: https://studentlife.utoronto.ca/department/accessibility-services/

Course Evaluation

This is tentative and may change in the next few days:

- Ten (10) practice assignments (40%)
  - Most (8) are small programming exercises.
  - Two (2) are derivation-based exercises.
  - You start working on them during the Tutorial part of each session. Our TAs will help you understand the assignment and guide you in solving them.
- Research Project (30%)
  - Research proposal (10%), written report and submitted codebase (20%).
- Take-Home Test: 20% (close to the end of the semester)
- Readings (10%)
  - Read some (5) research papers from a paper bank of 10-15 papers.
  - Write a short 1-paragraph summary and two questions on how the method(s) can be used or extended.

Collaboration and Assignments

- Collaboration:
  - Collaboration on the Homework Assignments **is allowed**, under certain conditions:
    - You can discuss the assignment with another student (group of two).
    - In your submission, you need to be very clear about the contribution of each individual. For example, you should say we did a pair-programming or person A solved this part while person B solved another part.
    - You *can* use copilot, ChatGPT, etc. to solve the problem, but that would consider as your group member. That is, you can have either a 2-human group or 1 human + 1 machine group (no 2 machine group). If you do it, you should report it as well.
  - You need to form a team of 3–4 members to work on your projects (the exact number will be determined after finalizing the number of students enrolled).
    - Similar to the homework assignments, you need to report the contribution of each collaborator.
    - If you get the help of a machine, you need to clearly indicate that. The machine will cost you one of the team members.
  - Collaboration on the Take-home Test or Paper Readings is **not allowed**. These should be done as individual.

Collaboration and Assignments

- Late Submissions (assignments, proposals, reports, etc)
  - Submissions should be handed in by deadline; a late penalty of 10% per day will be assessed thereafter (up to 3 days, then submission is blocked).
  - Extensions will be granted only in special situations, and you will need a Student Medical Certificate or a written request approved by the course coordinator at least one week before the due date.

Academic Integrity

- By this point in your studies, you should know how to follow the academic integrity rules. You need to know what cheating and plagiarism are.
- If you need a review, read the U of T's Code of Behaviour on Academic Matters.
- Don't cheat or plagiarize!

Useful Resources

Recommended readings will be given for each lecture. But the following will be useful throughout the course:

- **Deep Learning**, a textbook by Yoshua Bengio, Ian Goodfellow, and Aaron Courville.
- **Dive into Deep Learning**
- Video lectures for the U of T Professor Geoffrey Hinton's **course**.
- Andrej Karpathy's **lecture notes** on convolutional networks.
- Richard Socher's **lecture notes**, focusing on RNNs.
- Video lectures for Hugo Larochelle's **neural networks course**.
- If you need to brush up your basic knowledge of ML, you can take a look at one of the previous offerings of it at the U of T. For example, **CSC2515 - Fall 2022** (the content is almost the same as CSC311).

Other Useful Resources:

- Trevor Hastie, Robert Tibshirani, and Jerome Friedman, The Elements of Statistical Learning, Second Edition, 2009.
- Christopher M. Bishop, Pattern Recognition and Machine Learning, 2006
- Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, Second Edition, 2018.
- Amir-massoud Farahmand, Lecture Notes on Reinforcement Learning, 2021.
- Kevin Murphy, Machine Learning: A Probabilistic Perspective, 2012.
- Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, An Introduction to Statistical Learning, 2017.
- Shai Shalev-Shwartz and Shai Ben-David, Understanding Machine Learning: From Theory to Algorithms, 2014.
- David MacKay, Information Theory, Inference, and Learning Algorithms, 2003.

Compute

- Colaboratory: Programming assignments are to be completed in Google Colab, which is a web-based iPython Notebook service that has access to a free Nvidia K80 GPU per Google account. Highly recommended for homeworks and some course projects.
- Department Teaching Labs: Linux compute servers with desktop or datacentre-class GPUs. Recommended for course project.
- Google Compute Engine: GCE delivers virtual machines running in Google's data center. Recommended for course project.

Section 6

What to do this week?

What to do this week?

- No tutorials this week.
- Review your linear algebra, probability, and ML background