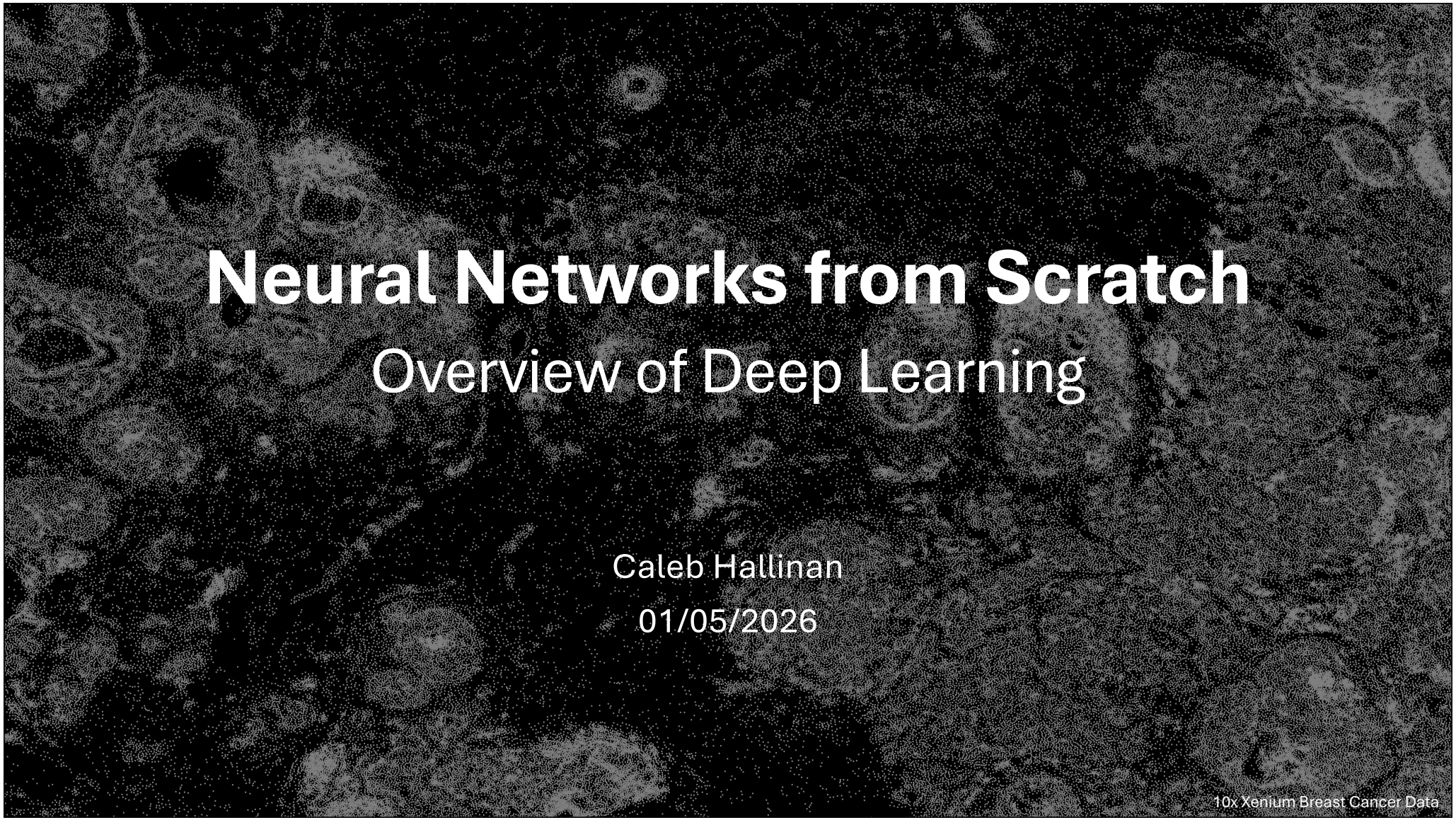


Neural Networks from Scratch

Overview of Deep Learning

Caleb Hallinan

01/05/2026

10x Xenium Breast Cancer Data

Quick Intro to Me

- 3rd year BME PhD Student in Dr. Jean Fan's lab
- Went to UVA for undergrad, lived in Boston for 2 years
- Research work is in deep learning for spatial transcriptomics :)

- Owner of a crazy dog named Eeko
- Celia, my wife, is a BCMB PhD Student in Dr. Mikala Egeblad's lab
- Love to hang with friends and play board games and sports!

Quick Intro of you!

- Let’s go around and share your:

1. Name
2. Year
3. Why you chose to take this class
4. A fun fact!

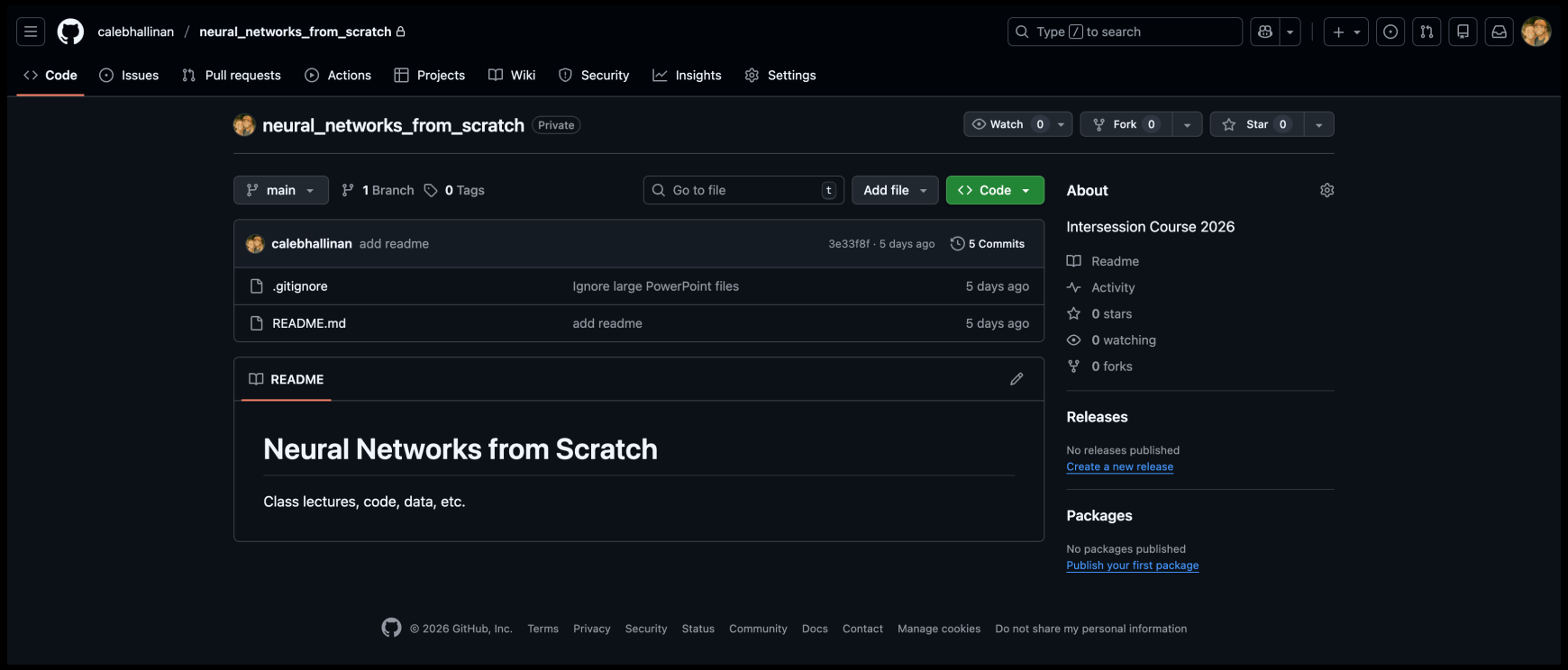
Goals of the Class

- Teach the basics of deep learning, including a few different types (CNNs and Transformers)
- Learn the basics of PyTorch, a python-based deep learning library, to build and utilize neural networks
- Complete a deep learning project using real-world data, and improve presentation skills by sharing your work

Syllabus

- No hws (except setting up github if not already done), final project
- Show up and participate ☺
- 10 classes total, each 2 hours 45min (break at 10:45am and may not last the entire time every class)
- Always happy to send you more material or chat, just email me
  - challin1@jh.edu

GitHub for class



https://github.com/calebhallinan/neural_networks_from_scratch

Please ask questions at any time!

- I may not always know the answer, but will do my best and get back to you if I don't ☺
- Also happy to meet outside of class and will have OH the last few days for project questions

Outline of class

1. Intro to git and overview of deep learning
2. Neural networks from scratch (live notes and code)
3. Neural networks from scratch (live notes and code)
4. Neural networks from scratch (live notes and code)
5. CNNs and other concepts to know (powerpoint and code)
6. Intro to PyTorch (code)
7. Intro to PyTorch (code)
8. Transformers and assign projects (powerpoint)
9. Project workday
10. Project presentations

Project Details

- You can choose from 1 of 3 different project ideas, all will require:
  - Math explanation of one component
  - Code for all your work (will be put on github for everyone to see)
  - Visualization of results
  - Key takeaways/conclusions

- 8-10 minute presentation and will be asked 1 question

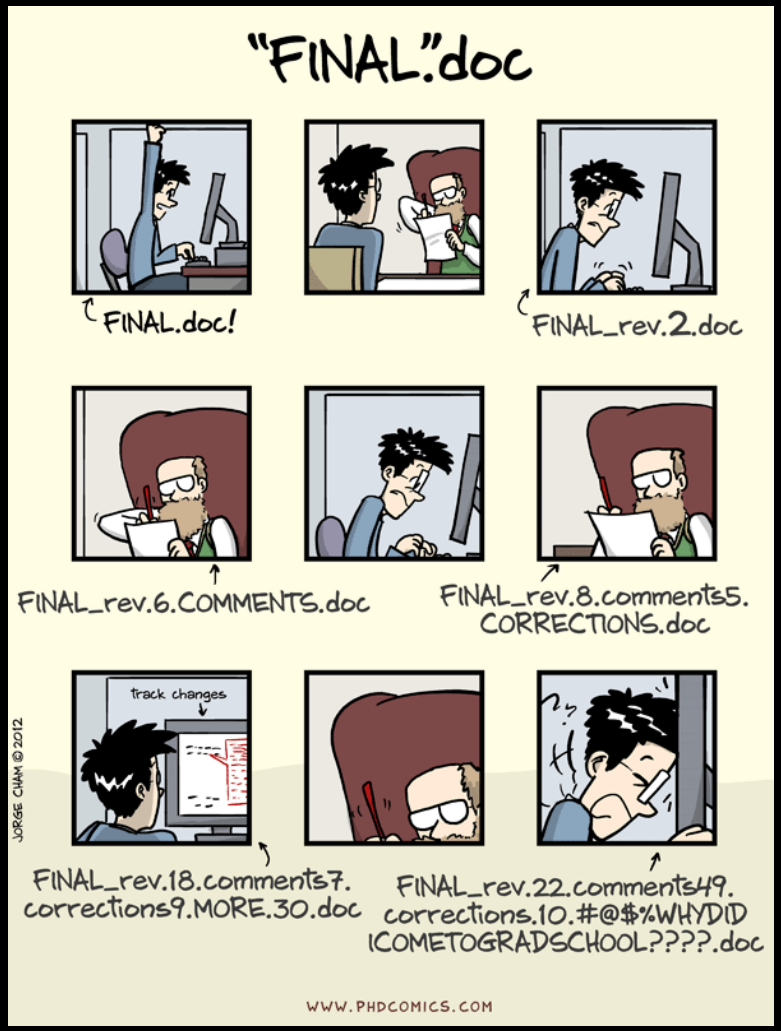
Outline for today

- Intro to the git/github and how to work with it
- Overview of Deep Learning

Github Intro

GitHub Overview

- All class material will be posted on GitHub
- Will go through a quick GitHub tutorial here!
- Next slides are heavily based on Dr. Stephanie Hicks' git slides!
  - https://docs.google.com/presentation/d/1MyhQ9JMXwpkv6xyOLs-FTE5ZPzweikj-NAaHIZpbZBQ/edit#slide=id.g31018913631_0_73
- Who has a GitHub account and has worked with github?

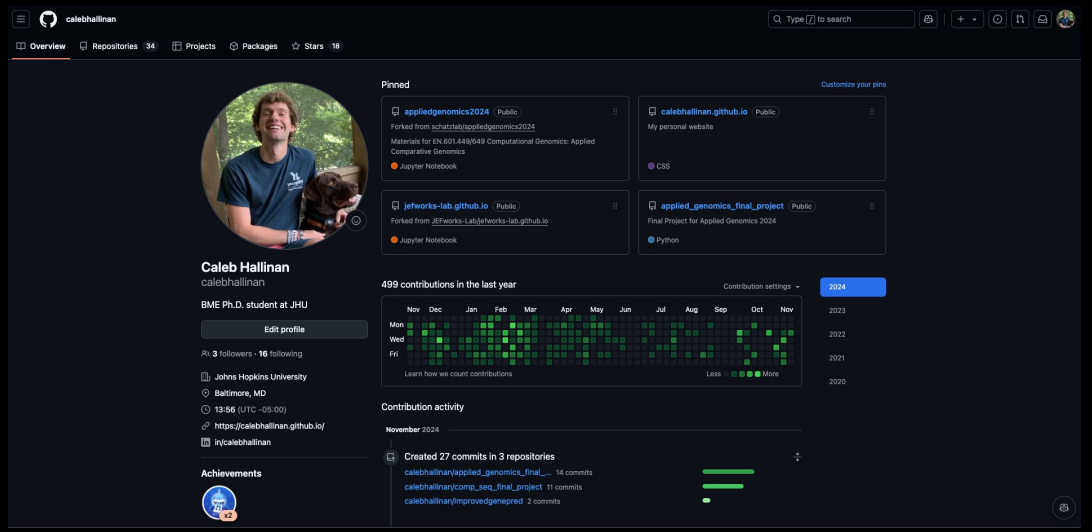

"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

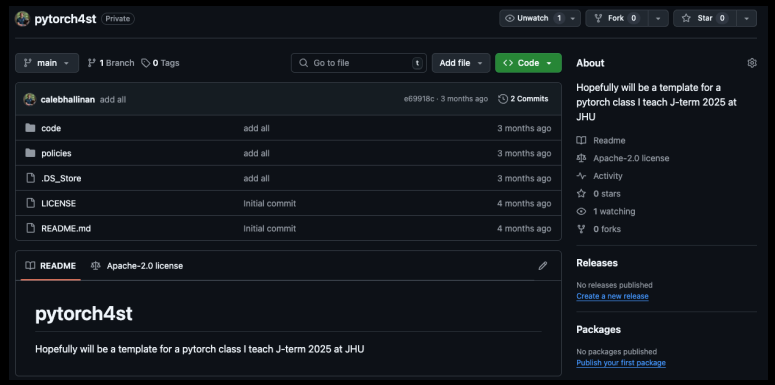
git

- **git** is a version control system used primarily to manage and track changes in files and source code during software development
  - Easy to work with collaborators
  - Secure backup and recovery
- Technically, git is a command-line tool or has a CLI (command line interface)
- Similar to the “track changes” features in Microsoft Word, but more rigorous, powerful, and scaled up to multiple files
- Version control systems start with a base version of the document and then record changes you make each step of the way

GitHub

- GitHub is a hosting service on internet for git-aware folders and projects
  - Similar to DropBox or Google, but more structured, powerful, and programmatic
  - Great for solo or collaborative work!
  - GitHub is distinct from Git. However, GitHub is in some sense the interface and Git the underlying engine (a bit like RStudio and R).
- Note: Other interfaces to Git exist, e.g., Gitlab, Bitbucket, etc





When to use GitHub

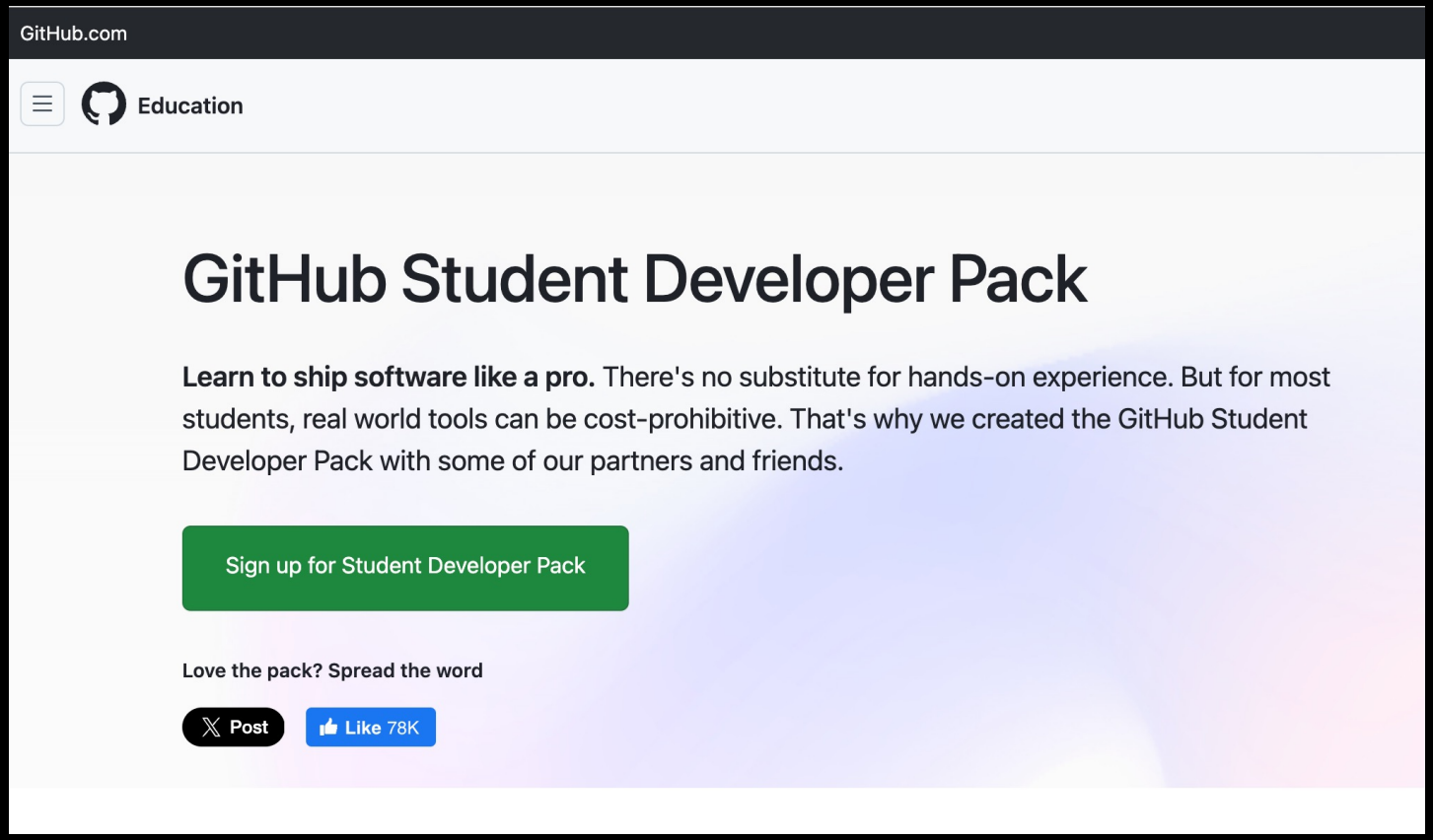
- **GitHub is ideal** for:
  - A project with a fair number of files, most of those files are text files (such as code, LaTeX, etc.) and different people work on different parts of the project.

- **GitHub is less useful** for:
  - Lots of non-text files (e.g. Word or Powerpoint) and when other team members want to edit the same document at the same time
    - Recommend to use Overleaf, Google Docs, Word+Dropbox, Word+Onedrive, etc.

Let’s learn how to use git/GitHub

- Git and GitHub is fundamentally based on commands you type into the command line
  - This is how I almost always interact with git/GitHub
  - I highly recommend learning this way!
- However, many people find this the most confusing way to use git/GitHub. Alternatively, there are GUIs
  - GitHub itself provides a graphical interface with basic functionality
  - RStudio also has GUI git/GitHub integration
  - VSCode has GUI git/Github integration – I use this a lot too

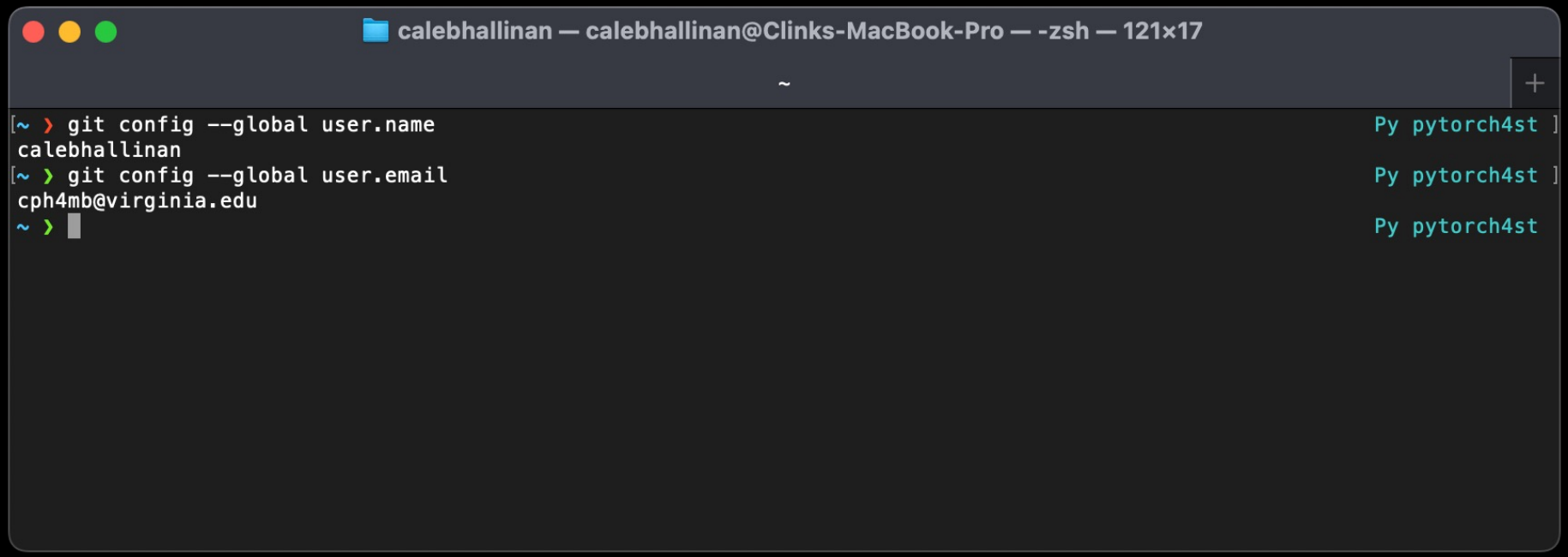
Free GitHub pro for students!



Setting up git and GitHub

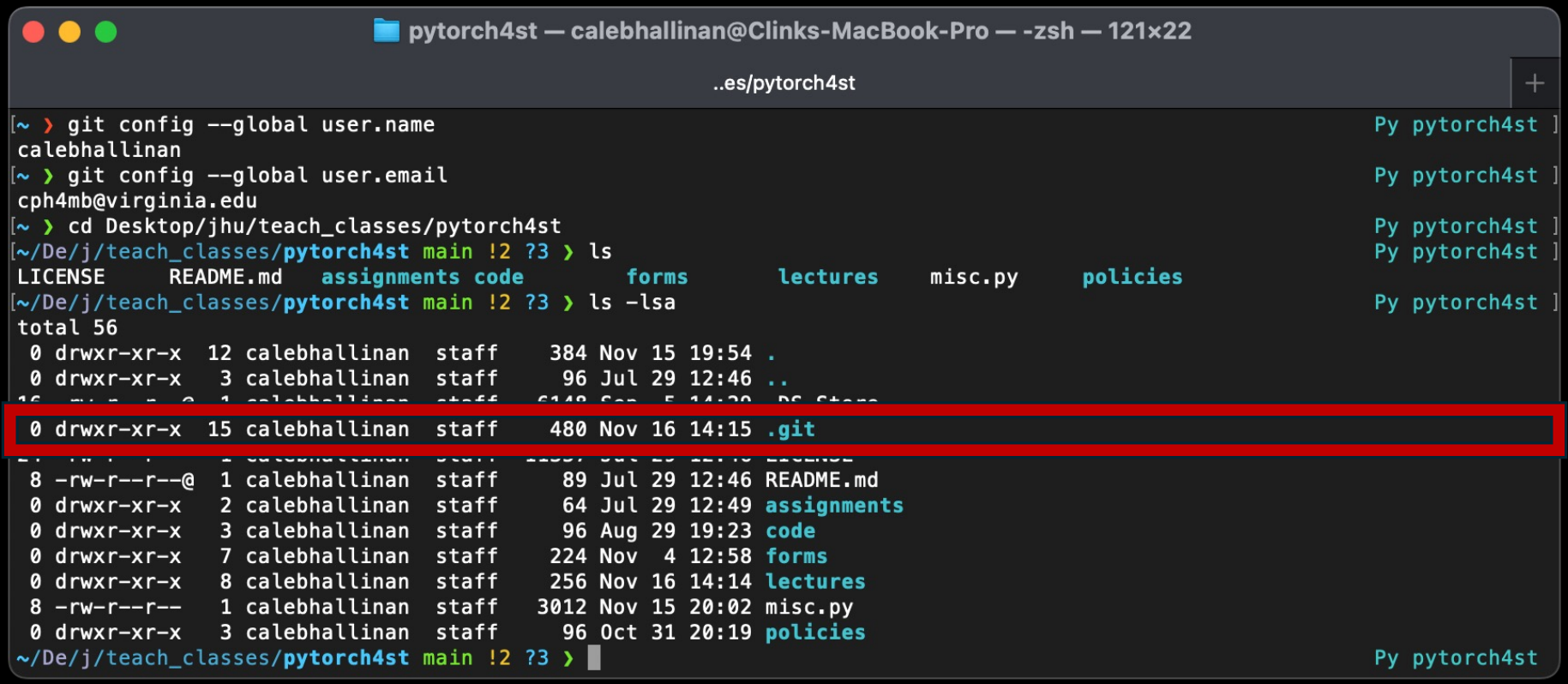
git config

- When you use Git on a new computer for the first time, we need to configure a few things:
  - name and email address
  - what your preferred text editor is
  - and that we want to use these settings globally (i.e. for every project)
- On a command line, Git commands are written as git verb options, where verb is what we actually want to do and options is additional optional information.
- Here is how to set up Git on a new laptop with git config:
  - $ git config --global user.name "My Name"
  - $ git config --global user.email "myemail@email.com"
- This username and email will be associated with your subsequent Git activity


calebhallinan — calebhallinan@Clinks-MacBook-Pro — -zsh — 121×17
~
[~ ❯ git config --global user.name Py pytorch4st]
calebhallinan
[~ ❯ git config --global user.email Py pytorch4st]
cph4mb@virginia.edu
~ ❯ Py pytorch4st

Creating a local repository

- **git init** = initializes a folder for being git-aware / version control
  - Creates a `.git` folder where git will store this history of the repository

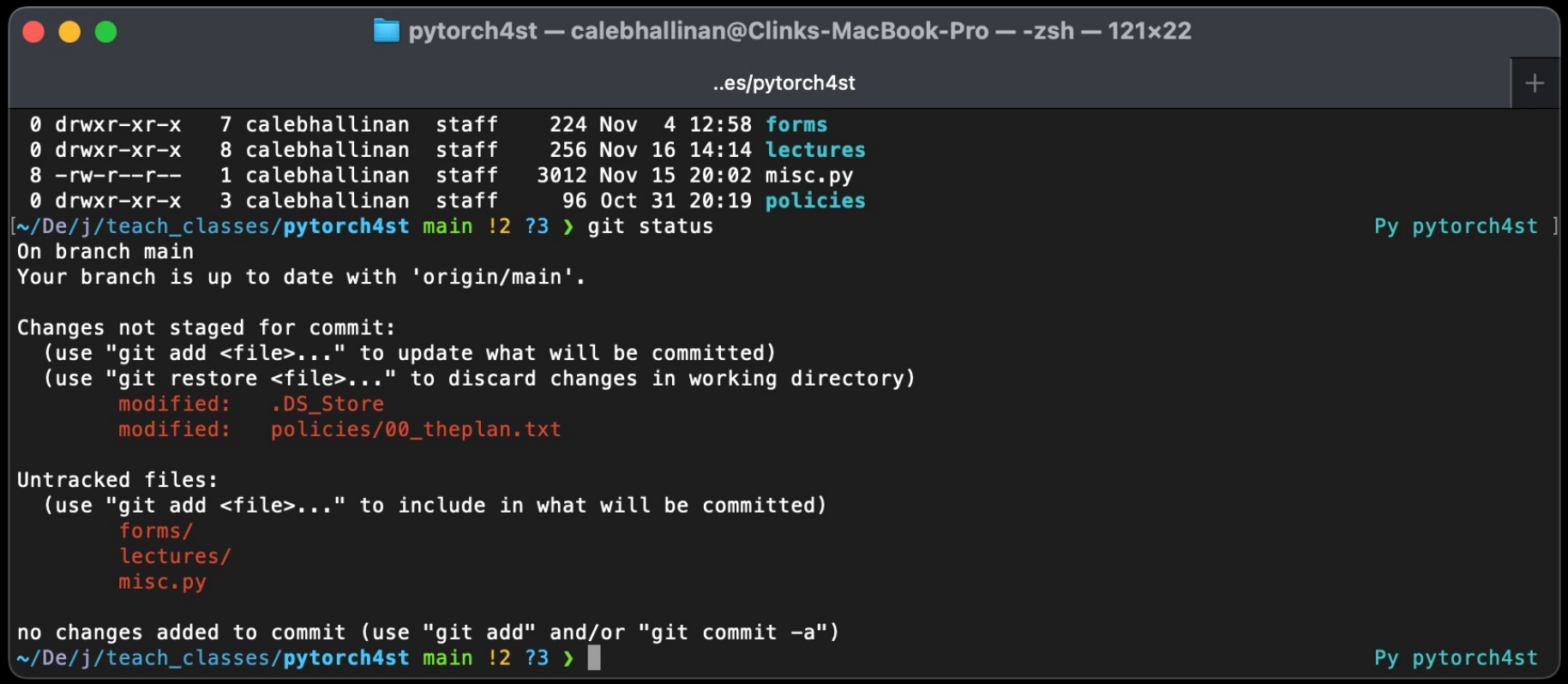

pytorch4st — calebhallinan@Clinks-MacBook-Pro — -zsh — 121×22
..es/pytorch4st
[~ ❯ git config --global user.name Py pytorch4st]
calebhallinan
[~ ❯ git config --global user.email Py pytorch4st]
cph4mb@virginia.edu
[~ ❯ cd Desktop/jhu/teach_classes/pytorch4st Py pytorch4st]
[~/De/j/teach_classes/pytorch4st main !2 ?3 ❯ ls Py pytorch4st]
LICENSE README.md assignments code forms lectures misc.py policies
[~/De/j/teach_classes/pytorch4st main !2 ?3 ❯ ls -lsa Py pytorch4st]
total 56
0 drwxr-xr-x 12 calebhallinan staff 384 Nov 15 19:54 .
0 drwxr-xr-x 3 calebhallinan staff 96 Jul 29 12:46 ..
0 drwxr-xr-x 15 calebhallinan staff 480 Nov 16 14:15 .git
8 -rw-r--r--@ 1 calebhallinan staff 89 Jul 29 12:46 README.md
0 drwxr-xr-x 2 calebhallinan staff 64 Jul 29 12:49 assignments
0 drwxr-xr-x 3 calebhallinan staff 96 Aug 29 19:23 code
0 drwxr-xr-x 7 calebhallinan staff 224 Nov 4 12:58 forms
0 drwxr-xr-x 8 calebhallinan staff 256 Nov 16 14:14 lectures
8 -rw-r--r-- 1 calebhallinan staff 3012 Nov 15 20:02 misc.py
0 drwxr-xr-x 3 calebhallinan staff 96 Oct 31 20:19 policies
~/De/j/teach_classes/pytorch4st main !2 ?3 ❯ Py pytorch4st

Creating a local repository

- **git init** = initializes a folder for being git-aware / version control
  - Creates a `.git` folder where git will store this history of the repository

Creating a local repository

- **git init** = initializes a folder for being git-aware / version control
  - Creates a `.git` folder where git will store this history of the repository
- **git status** = displays the state of the working directory and the staging area


pytorch4st — calebhallinan@Clinks-MacBook-Pro — -zsh — 121×22
..es/pytorch4st
0 drwxr-xr-x 7 calebhallinan staff 224 Nov 4 12:58 forms
0 drwxr-xr-x 8 calebhallinan staff 256 Nov 16 14:14 lectures
8 -rw-r--r-- 1 calebhallinan staff 3012 Nov 15 20:02 misc.py
0 drwxr-xr-x 3 calebhallinan staff 96 Oct 31 20:19 policies
[~/De/j/teach_classes/pytorch4st main !2 ?3 ❯ git status Py pytorch4st]
On branch main
Your branch is up to date with 'origin/main'.
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
modified: .DS_Store
modified: policies/00_theplan.txt
Untracked files:
(use "git add <file>..." to include in what will be committed)
forms/
lectures/
misc.py
no changes added to commit (use "git add" and/or "git commit -a")
~/De/j/teach_classes/pytorch4st main !2 ?3 ❯ Py pytorch4st

Creating a local repository

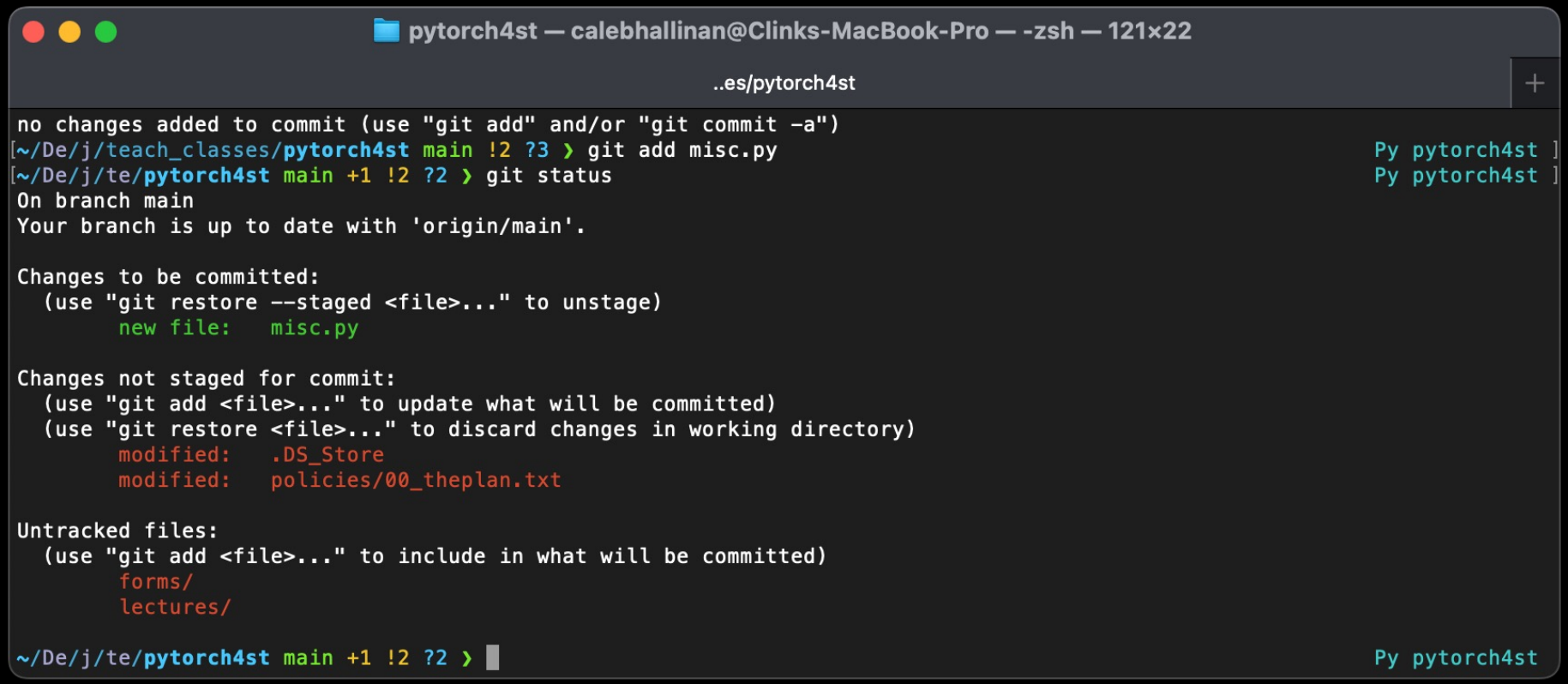
- **git init** = initializes a folder for being git-aware / version control
  - Creates a `.git` folder where git will store this history of the repository
- **git status** = displays the state of the working directory and the staging area

Creating a local repository

- **git init** = initializes a folder for being git-aware / version control
  - Creates a `.git` folder where git will store this history of the repository
- **git status** = displays the state of the working directory and the staging area

Tracking changes in a local repo

- **git add** = tell git which files you want to track changes for


pytorch4st — calebhallinan@Clinks-MacBook-Pro — -zsh — 121×22
..es/pytorch4st
no changes added to commit (use "git add" and/or "git commit -a")
[~/De/j/teach_classes/pytorch4st main !2 ?3 ❯ git add misc.py Py pytorch4st]
[~/De/j/te/pytorch4st main +1 !2 ?2 ❯ git status Py pytorch4st]
On branch main
Your branch is up to date with 'origin/main'.
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
new file: misc.py
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
modified: .DS_Store
modified: policies/00_theplan.txt
Untracked files:
(use "git add <file>..." to include in what will be committed)
forms/
lectures/
~/De/j/te/pytorch4st main +1 !2 ?2 ❯ Py pytorch4st

Creating a local repository

- **git init** = initializes a folder for being git-aware / version control
  - Creates a `.git` folder where git will store this history of the repository
- **git status** = displays the state of the working directory and the staging area

Tracking changes in a local repo

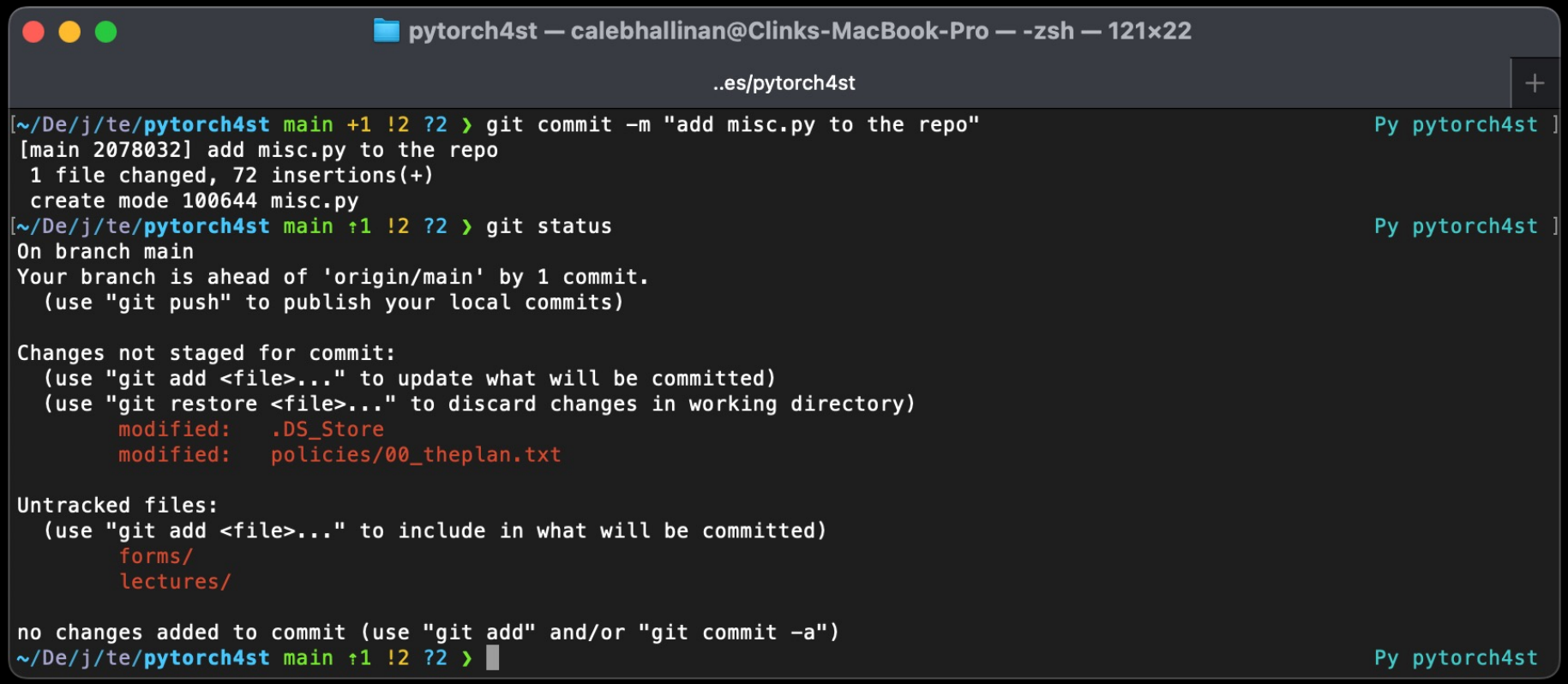
- **git add** = tell git which files you want to track changes for

Creating a local repository

- **git init** = initializes a folder for being git-aware / version control
  - Creates a `.git` folder where git will store this history of the repository
- **git status** = displays the state of the working directory and the staging area

Tracking changes in a local repo

- **git add** = tell git which files you want to track changes for
- **git commit** = tell git to take everything we are tracking (using git add) and stores a copy permanently inside the special .git directory. This permanent copy is called a commit (or revision)


pytorch4st — calebhallinan@Clinks-MacBook-Pro — -zsh — 121×22
..es/pytorch4st
[~/De/j/te/pytorch4st main +1 !2 ?2 ❯ git commit -m "add misc.py to the repo" Py pytorch4st]
[main 2078032] add misc.py to the repo
1 file changed, 72 insertions(+)
create mode 100644 misc.py
[~/De/j/te/pytorch4st main ↑1 !2 ?2 ❯ git status Py pytorch4st]
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
modified: .DS_Store
modified: policies/00_theplan.txt
Untracked files:
(use "git add <file>..." to include in what will be committed)
forms/
lectures/
no changes added to commit (use "git add" and/or "git commit -a")
~/De/j/te/pytorch4st main ↑1 !2 ?2 ❯ Py pytorch4st

git commit messages

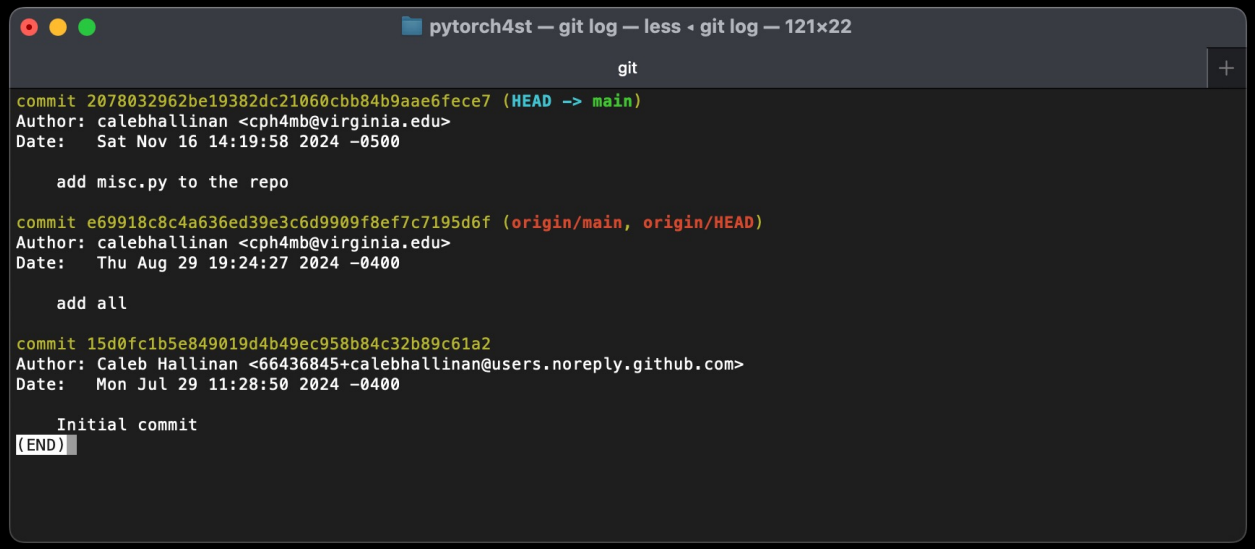
- We use the -m flag (for “message”) to record a short, descriptive, and specific comment that will help us remember later on what we did and why

- If we just run git commit without the -m option, git will launch nano (or whatever other editor is configured as core.editor) so that we can write a longer message

- Good commit messages start with a brief (<50 characters) statement about the changes made in the commit

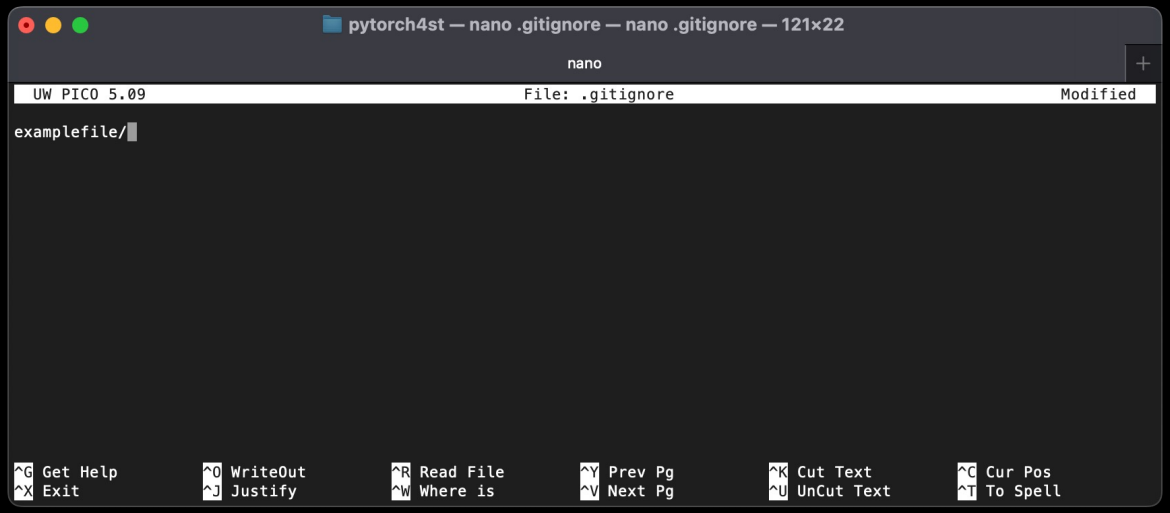
Show git commit history

- **git log** = shows the history of all the commits for the local repository



Ignoring files

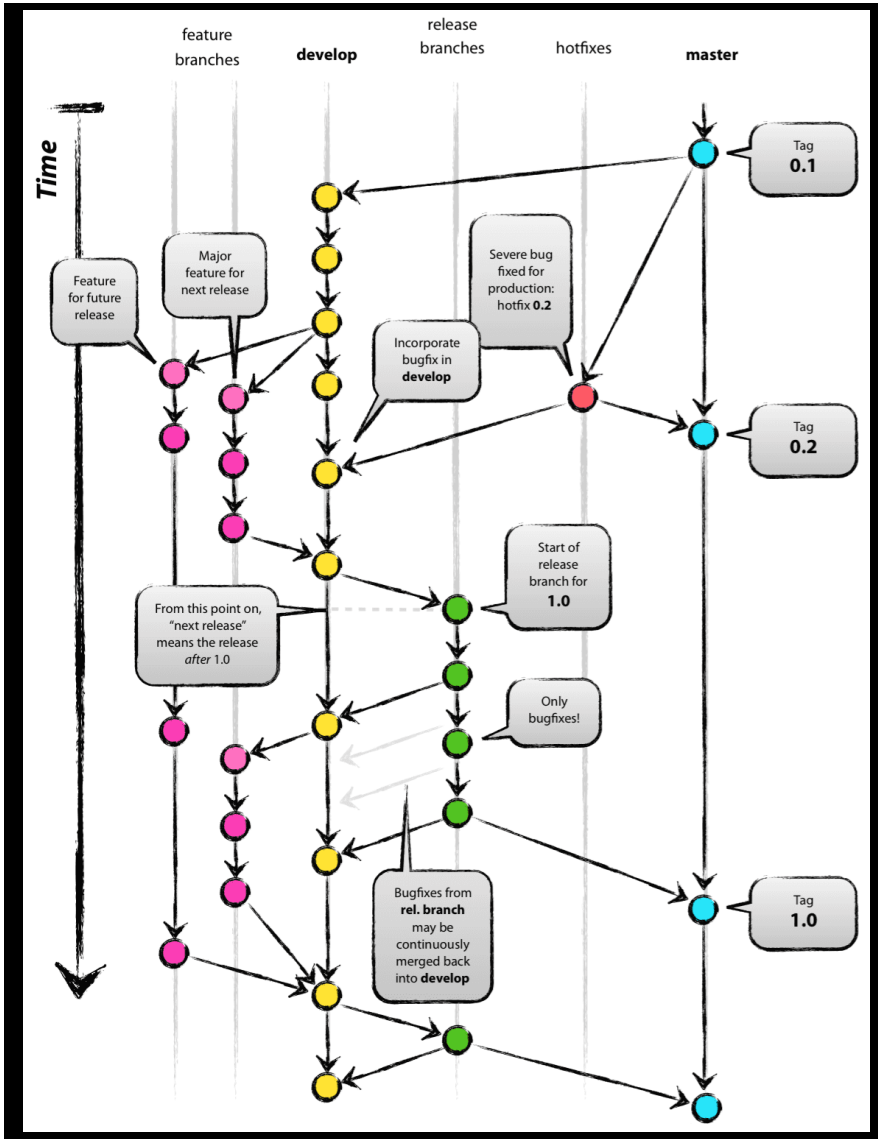
- Create a **.gitignore** file to specify specific types of files you do not want tracked



A few more to know...

- **git branch =** A command to create, list, or delete branches, which are independent lines of development in a Git repository. Branches let you work on different features or fixes without affecting the main codebase.

- **git checkout =** A command to switch between branches or restore files in your working directory to a specific commit. It updates the working directory to match the specified branch or commit.




```
$ git checkout -b myfeature develop
Switched to a new branch "myfeature"

$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff myfeature
Updating ea1b82a..05e9557
(Summary of changes)
$ git branch -d myfeature
Deleted branch myfeature (was 05e9557).
$ git push origin develop
```

https://nvie.com/posts/a-successful-git-branching-model/

Installing conda and necessary packages

- It’s good to have a package manager
  - I personally use conda, but jupyter env is fine as well if you prefer that
- Here is a tutorial on how to install conda on various machines
  - https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html
- If you need help setting this up reach out to me! You can also use google collab to do your project

VSCode

- I HIGHLY, HIGHLY recommend VSCode for coding
  - https://code.visualstudio.com/

- Awesome interface, handles multiple languages and formats, easily downloadable extensions for use

Any questions about git/GitHub?

Deep Learning Intro

A lot of these slides were taken from the MIT course

https://introtodeeplearning.com/

Questions

- Who here has used DL before?
  - How?
- Would you say you know how it works well?
- Can you give me a few applications of DL you know about?

Evolution of AI

March 2023

April 2025

What is Deep Learning?

Artificial Intelligence

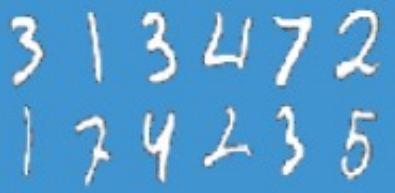
Any technique that enables computers to mimic human behavior

Machine Learning

Ability to learn without explicitly being programmed

Deep Learning

Extract patterns from data using neural networks



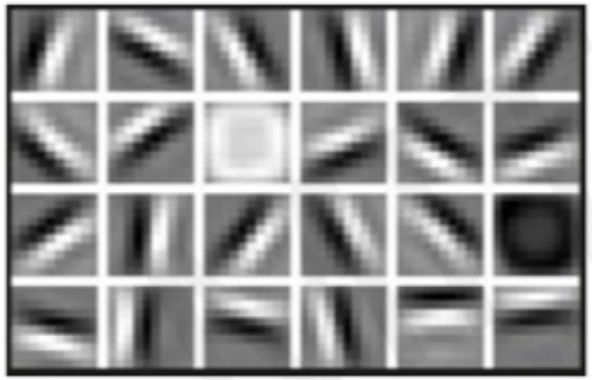
Teaching computers how to **learn a task** directly from **raw data**

Why Deep Learning?

Hand engineered features are time consuming, brittle, and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



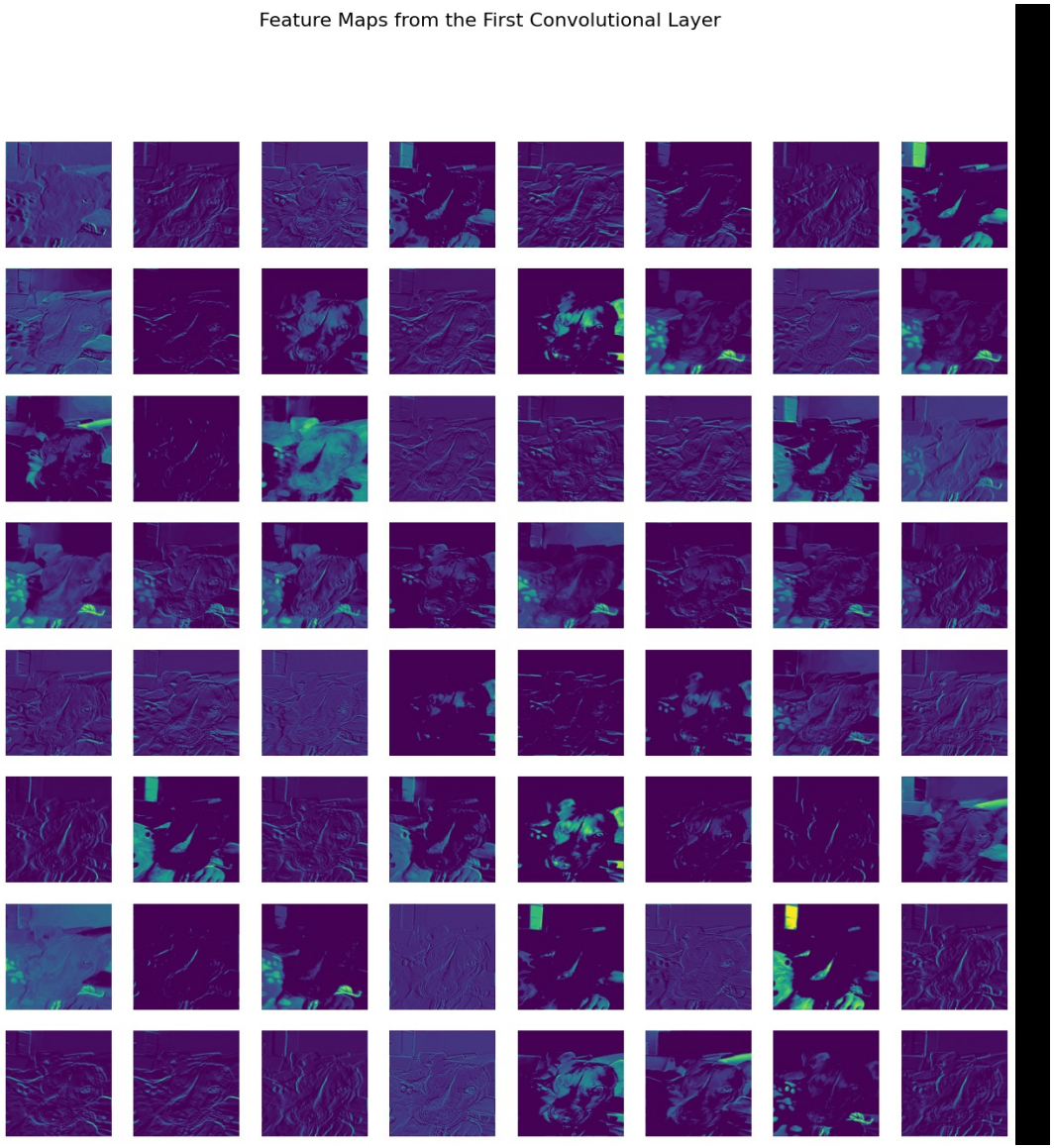
Lines & Edges

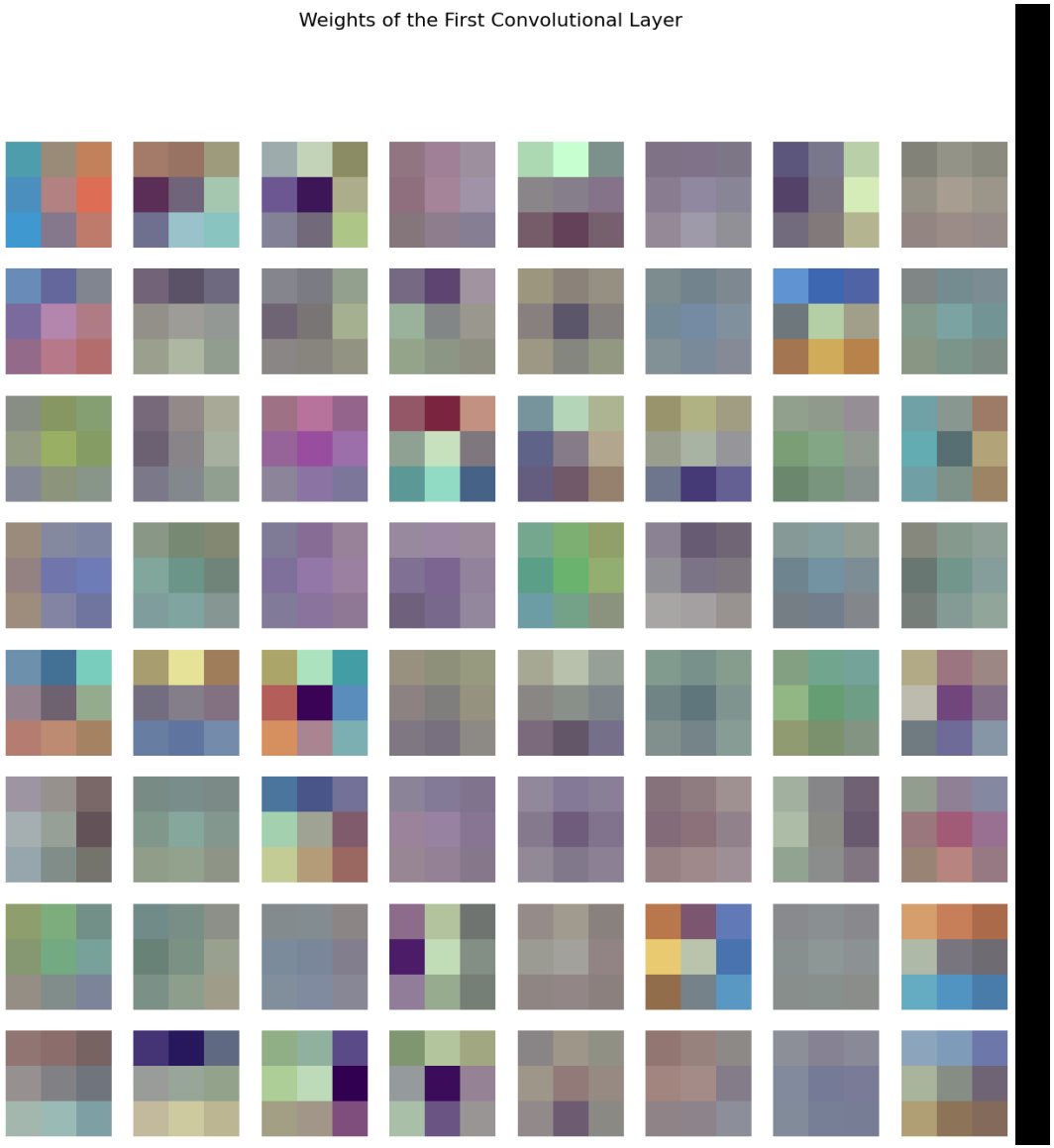
Mid Level Features

Eyes & Nose & Ears

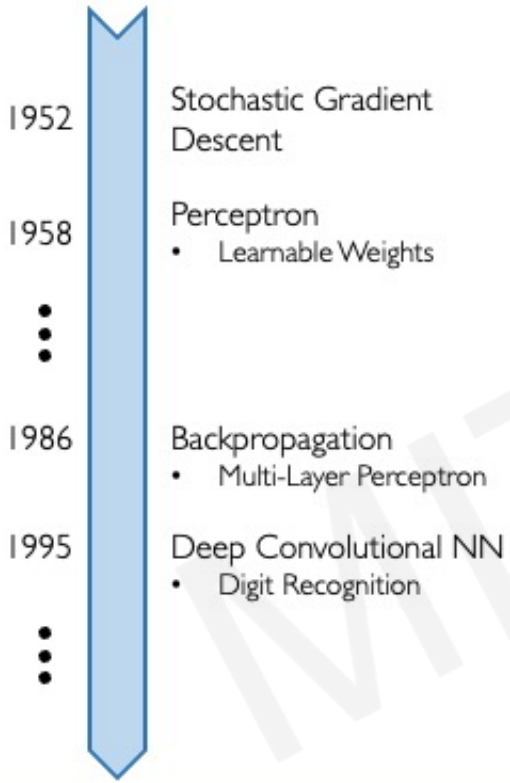
High Level Features

Facial Structure


Feature Maps from the First Convolutional Layer


Weights of the First Convolutional Layer

Why Now?



Neural Networks date back decades, so why the dominance?

1. Big Data

- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware

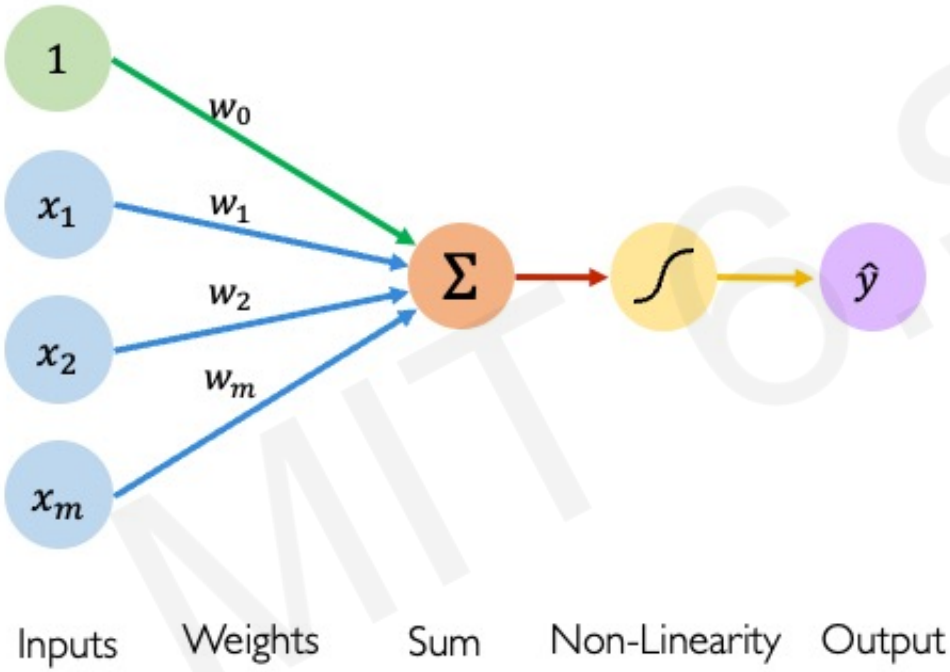
- Graphics Processing Units (GPUs)
- Massively Parallelizable

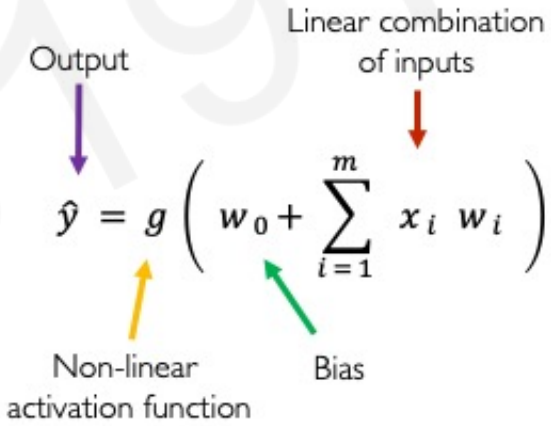
3. Software

- Improved Techniques
- New Models
- Toolboxes

Does anyone know the basic unit of DL?

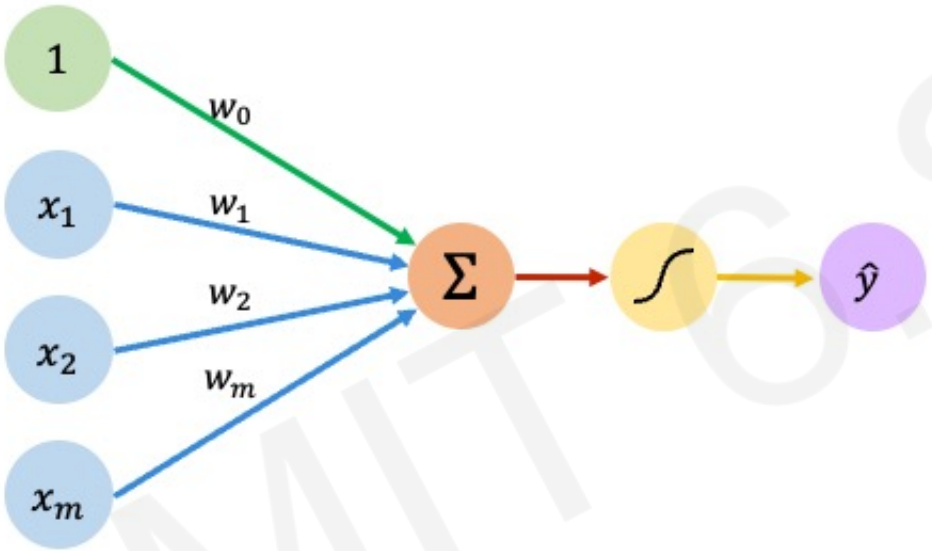
The Perceptron: Forward Propagation





Bias allows neurons to learn functions that aren't forced through the origin, making the network more flexible and powerful

The Perceptron: Forward Propagation

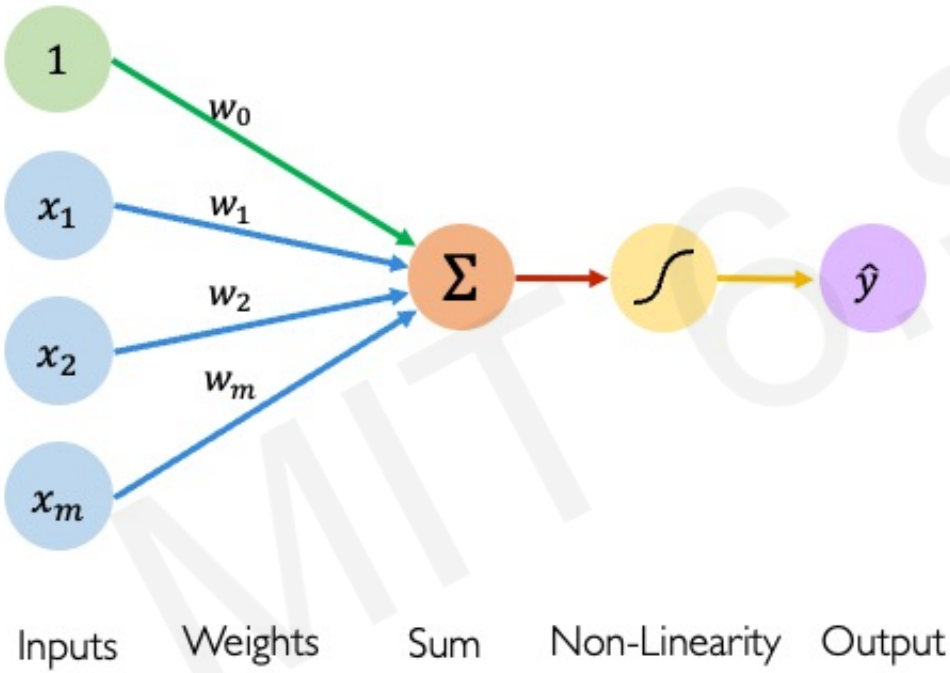


$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

The Perceptron: Forward Propagation

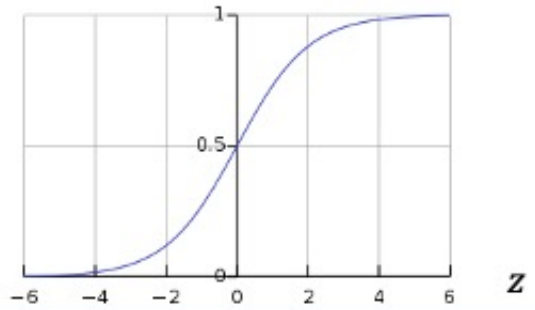


Activation Functions

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

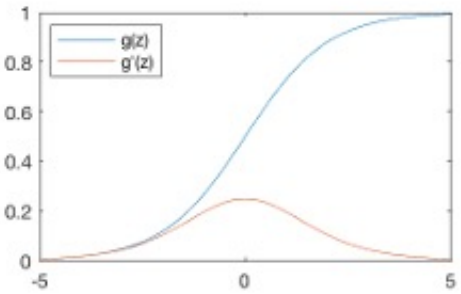
- Example: sigmoid function

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



Common Activation Functions

Sigmoid Function



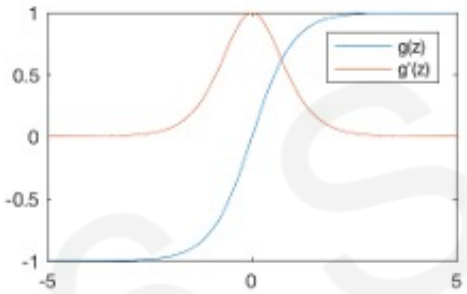
$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

`tf.math.sigmoid(z)`

`torch.sigmoid(z)`

Hyperbolic Tangent



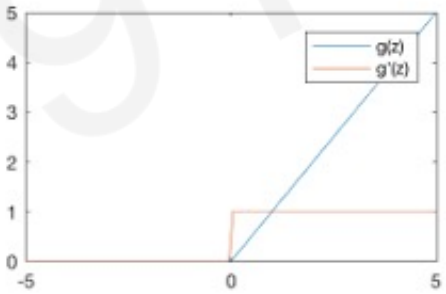
$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

`tf.math.tanh(z)`

`torch.tanh(z)`

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

`tf.nn.relu(z)`

`torch.nn.ReLU(z)`

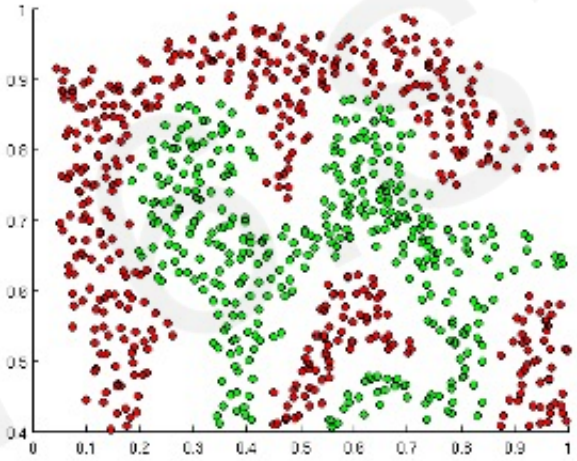
TensorFlow code blocks

NOTE: All activation functions are non-linear

PyTorch code blocks

Importance of Activation Functions

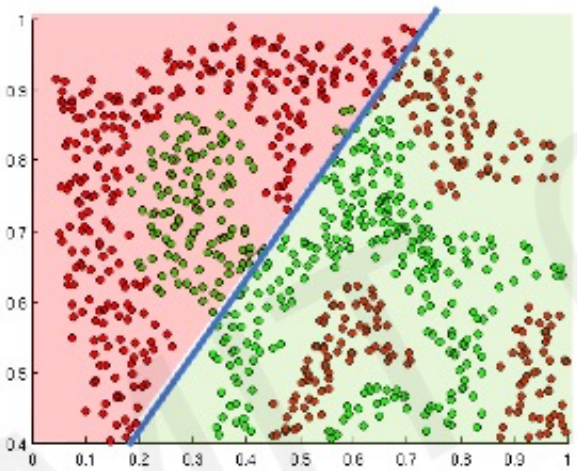
*The purpose of activation functions is to **introduce non-linearities** into the network*



What if we wanted to build a neural network to distinguish green vs red points?

Importance of Activation Functions

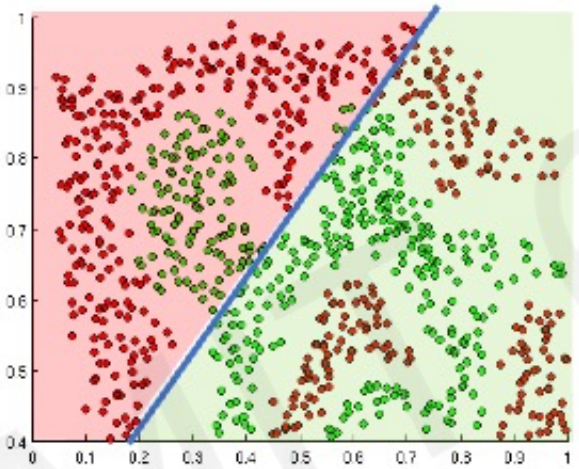
The purpose of activation functions is to ***introduce non-linearities*** *into the network*



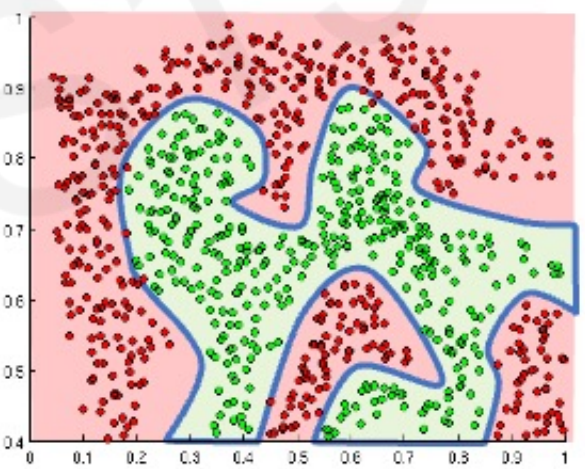
Linear activation functions produce linear decisions no matter the network size

Importance of Activation Functions

*The purpose of activation functions is to **introduce non-linearities** into the network*

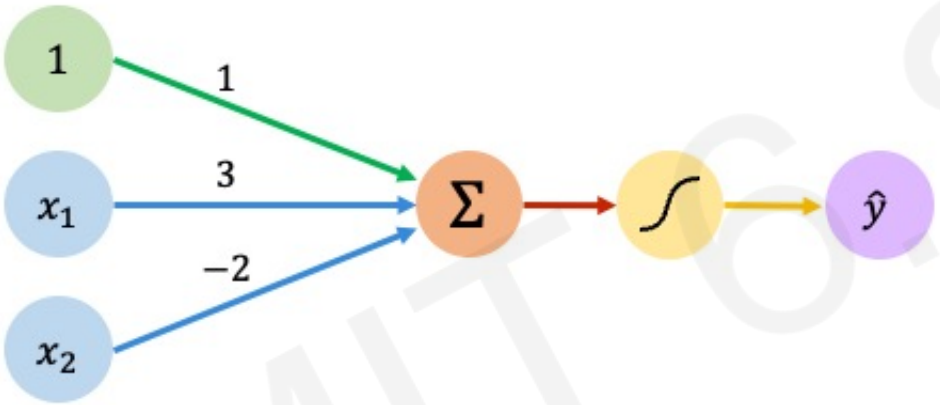


Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

The Perceptron: Example



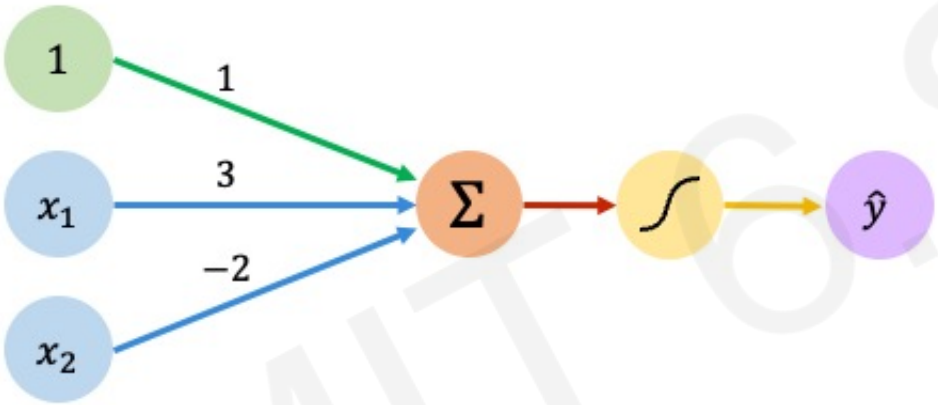
We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

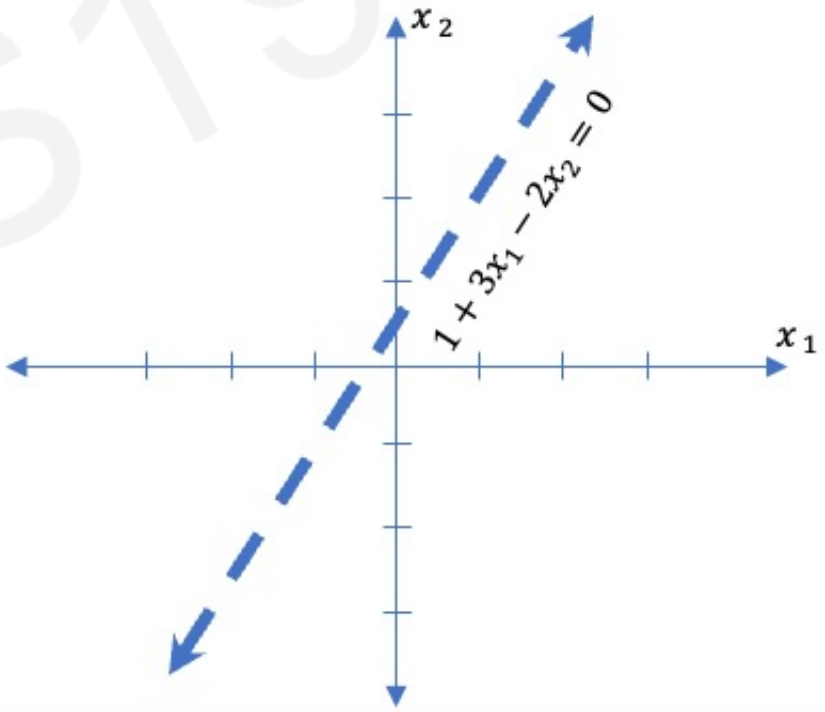
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g(\underbrace{1 + 3x_1 - 2x_2}_{\text{This is just a line in 2D!}})$$

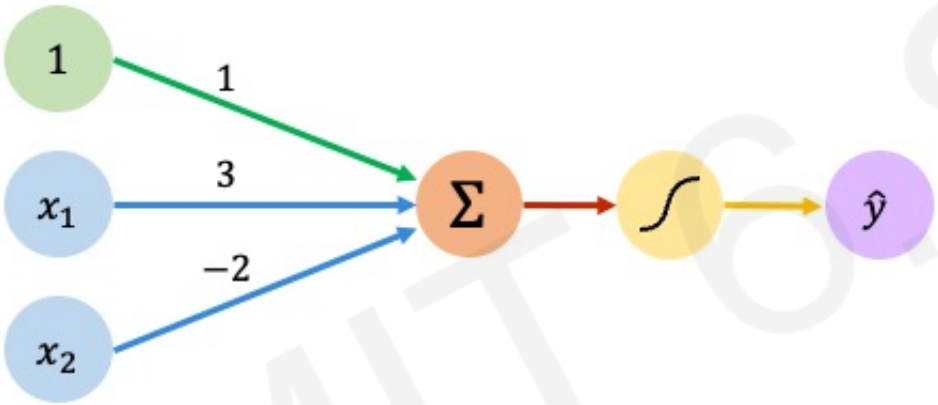
The Perceptron: Example

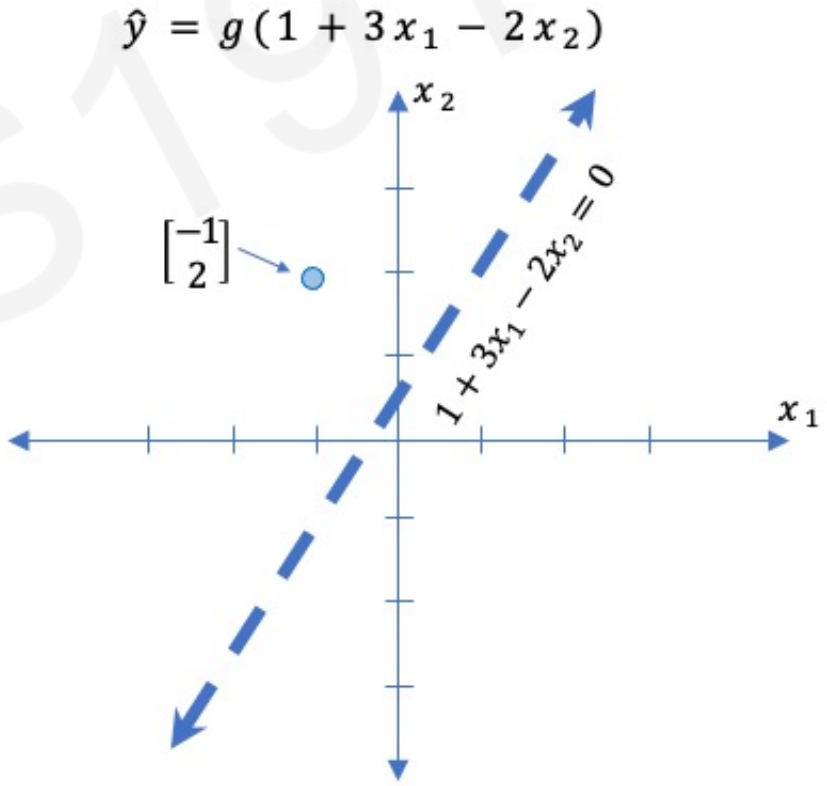


$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



The Perceptron: Example

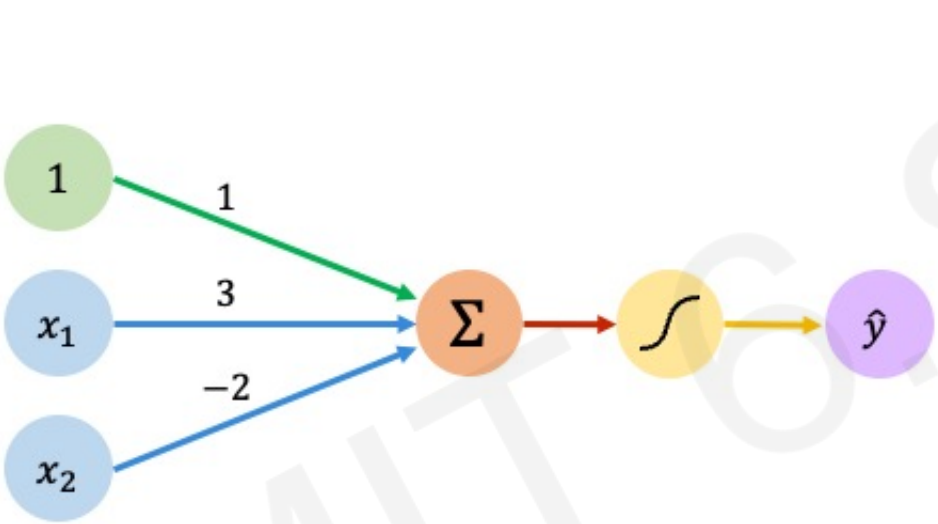




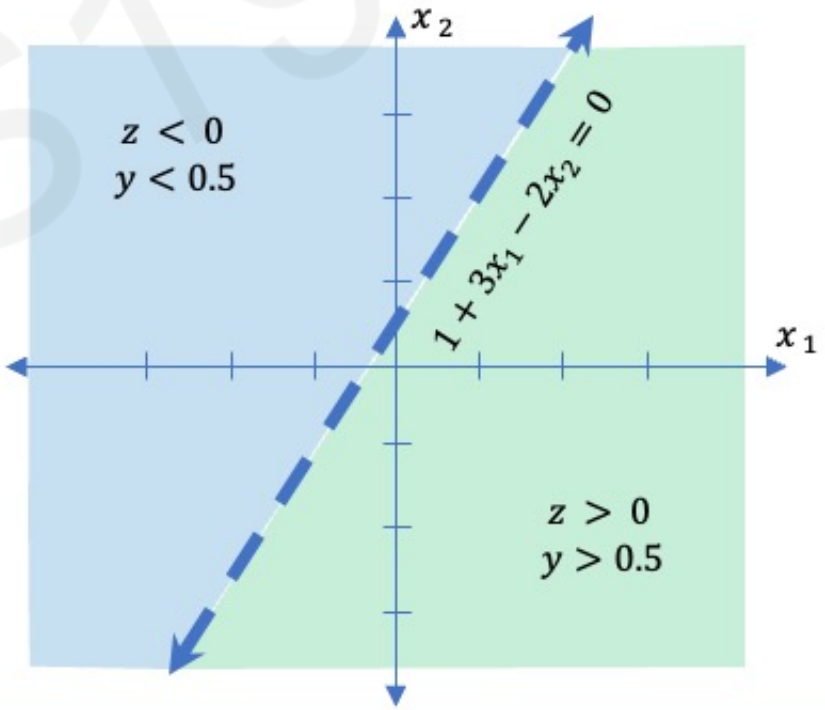
Assume we have input: $\boldsymbol{X} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

$$\hat{y} = g\big(1 + (3 * -1) - (2 * 2)\big)$$
$$= g(-6) \approx 0.002$$

The Perceptron: Example



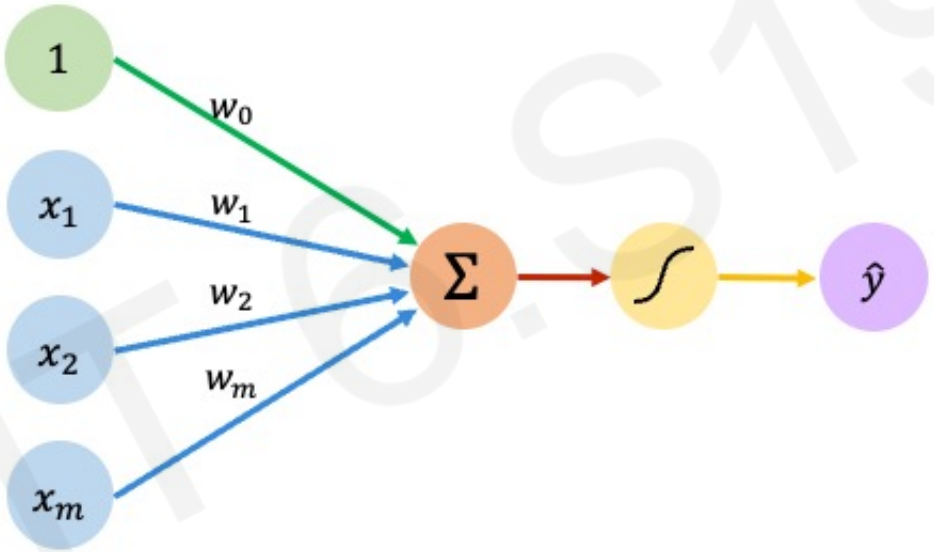
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



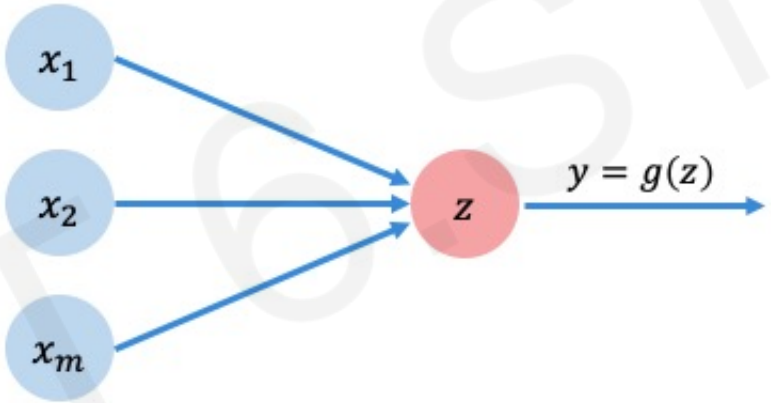
So how do we build neural networks with perceptrons?

The Perceptron: Simplified

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$



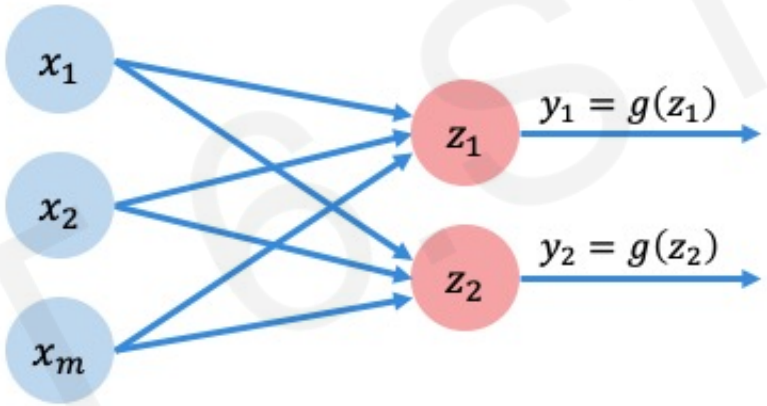
The Perceptron: Simplified



$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$

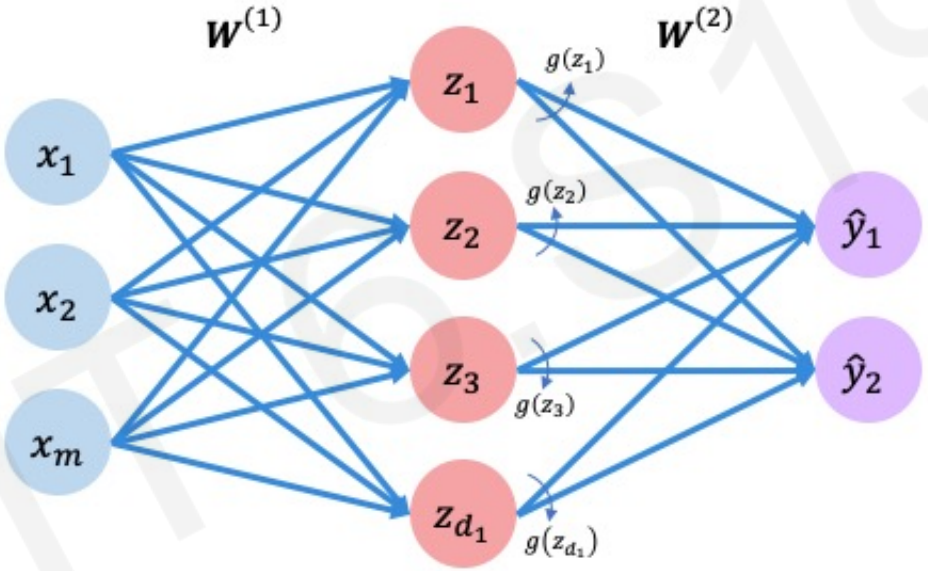
Multi Output Perceptron

Because all inputs are densely connected to all outputs, these layers are called **Dense** layers



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

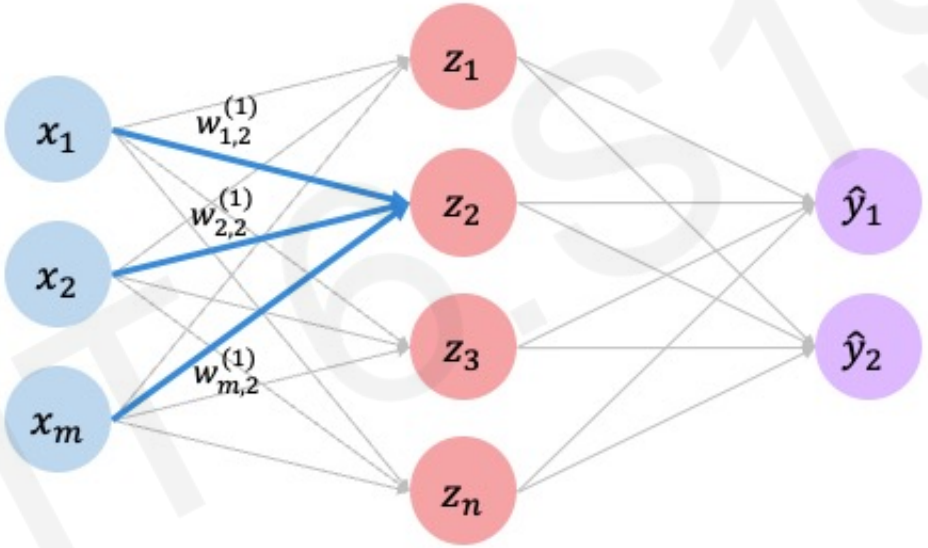
Single Layer Neural Network



Inputs Hidden Final Output

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) \, w_{j,i}^{(2)}\right)$$

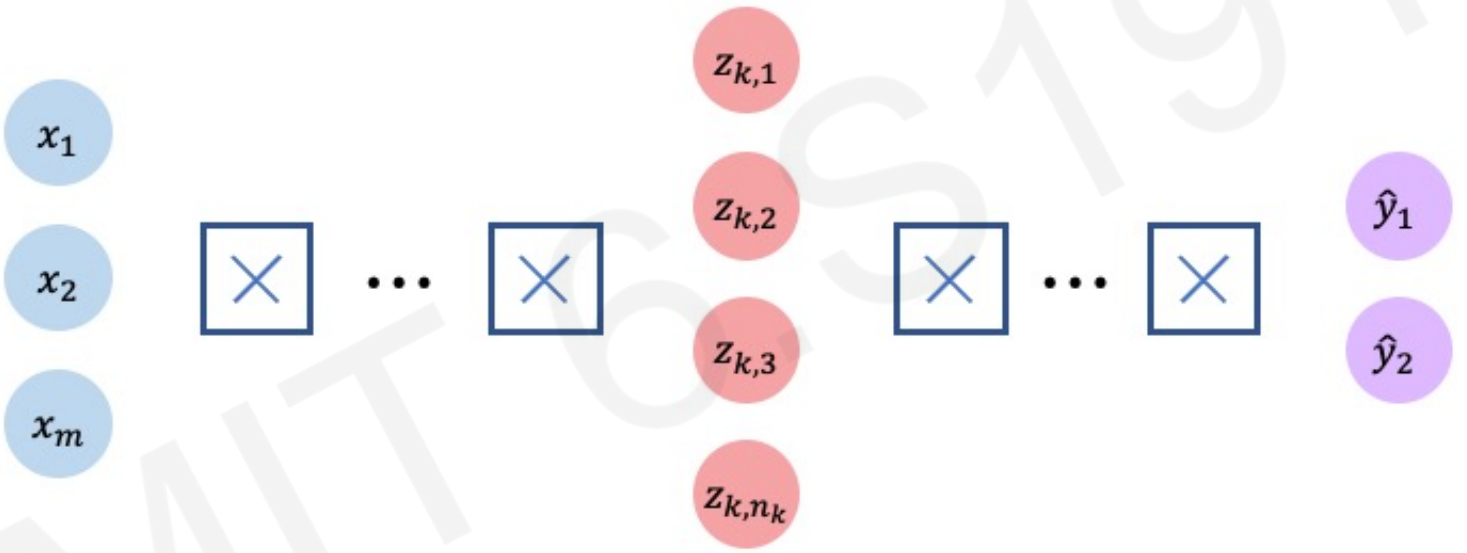
Single Layer Neural Network



$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}$$

Deep Neural Network



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Example Problem

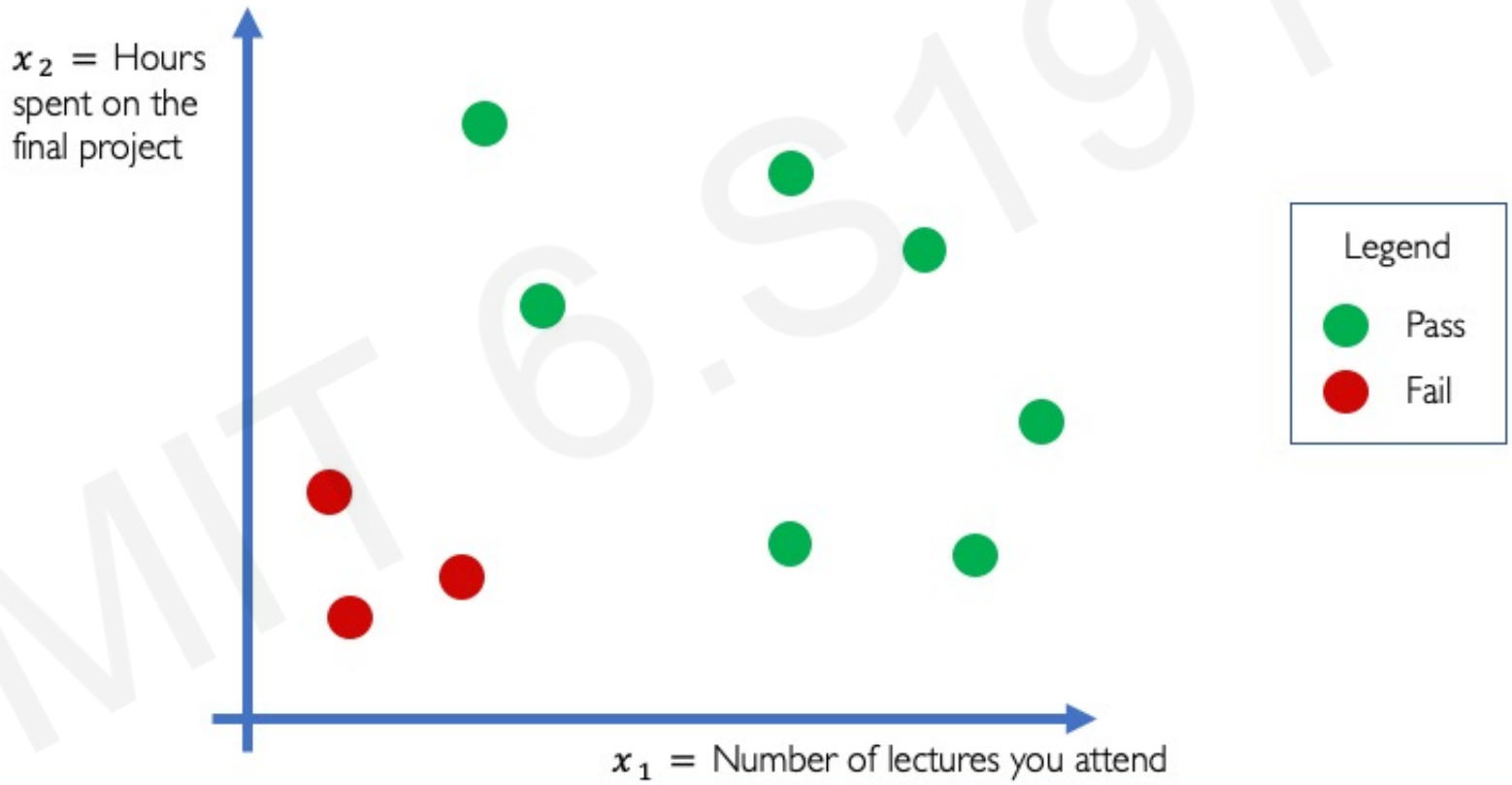
Will I pass this class?

Let's start with a simple two feature model

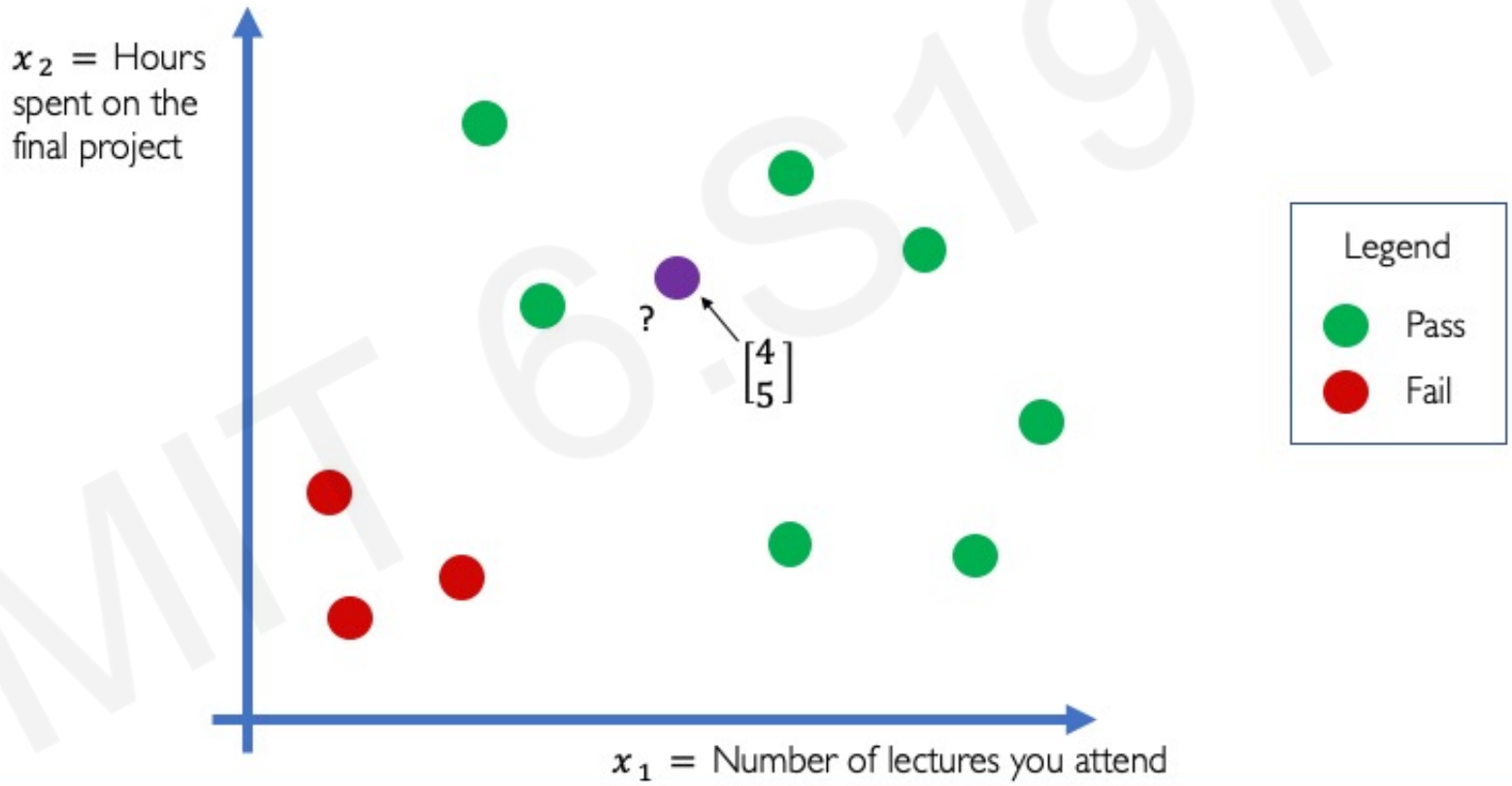
x_1 = Number of lectures you attend

x_2 = Hours spent on the final project

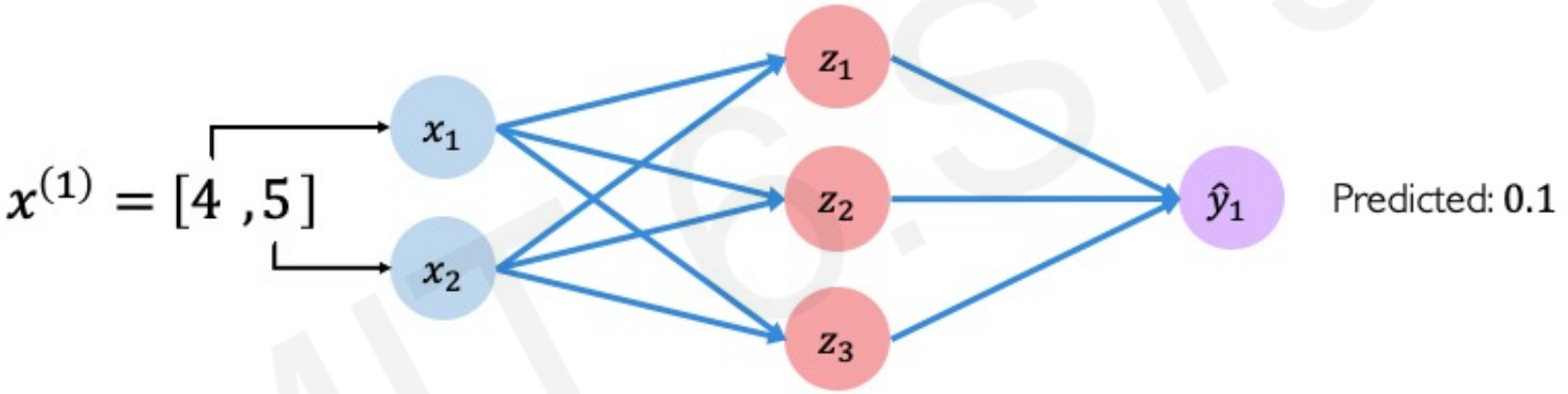
Example Problem: Will I pass this class?



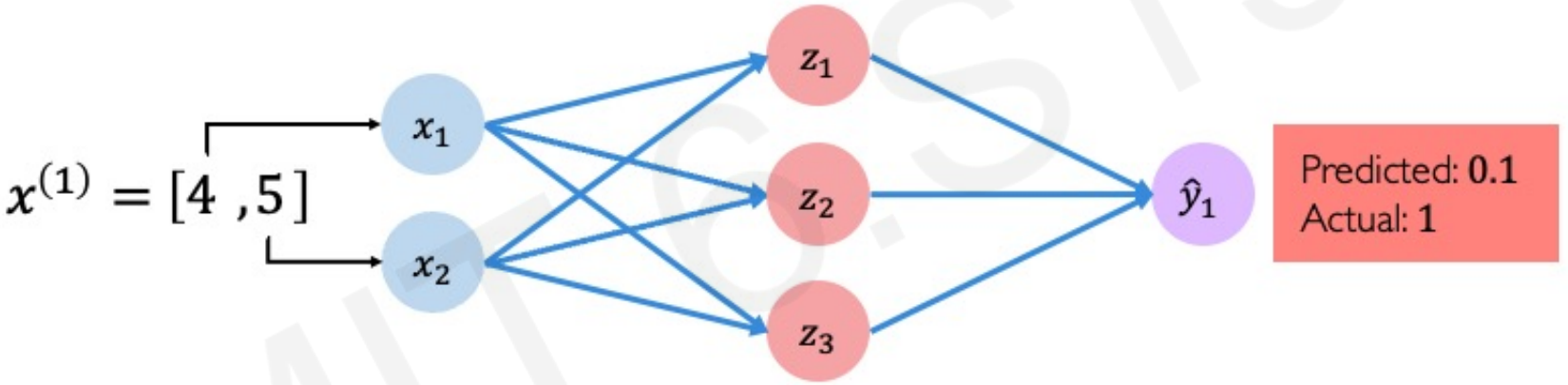
Example Problem: Will I pass this class?



Example Problem: Will I pass this class?

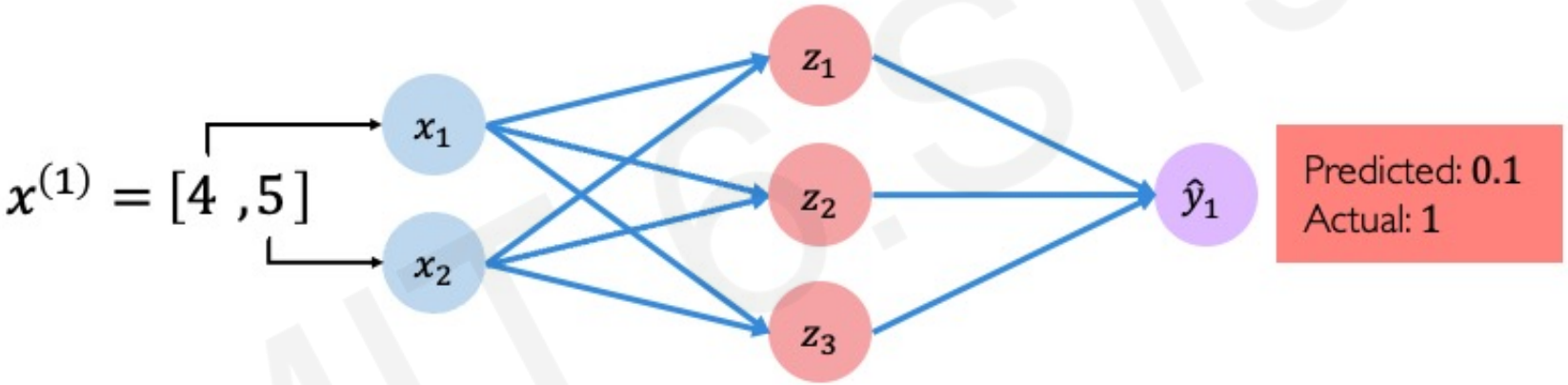


Example Problem: Will I pass this class?



Quantifying Loss

*The **loss** of our network measures the cost incurred from incorrect predictions*

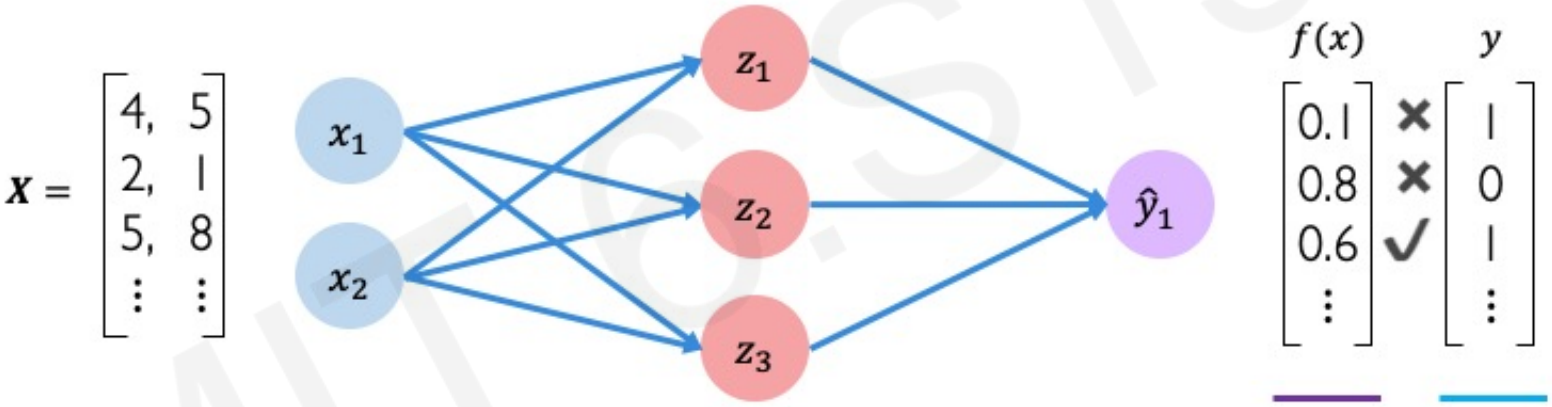


$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Empirical Loss

*The **empirical loss** measures the total loss over our entire dataset*



$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

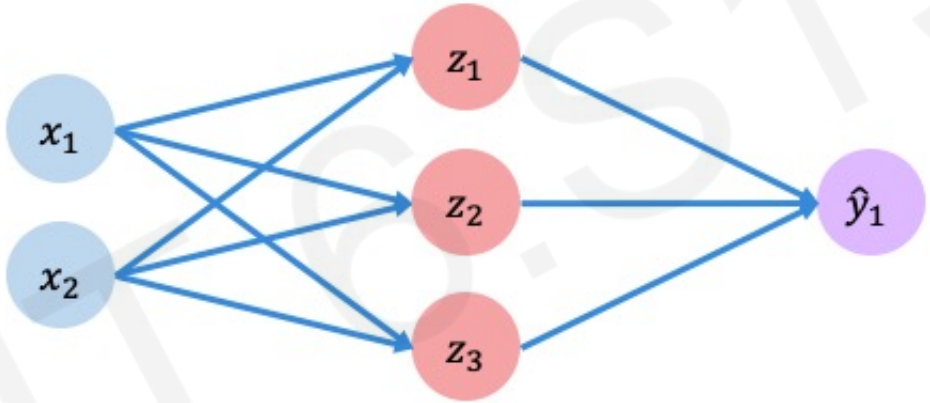
Predicted Actual

Also known as:
- Objective function
- Cost function
- Empirical Risk

Binary Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*

$$X = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$f(x)$ y

$$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

$$J(W) = -\frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log \underbrace{\left(f\left(x^{(i)}; W\right)\right)}_{\text{Predicted}} + \underbrace{(1 - y^{(i)})}_{\text{Actual}} \log \underbrace{\left(1 - f\left(x^{(i)}; W\right)\right)}_{\text{Predicted}}$$

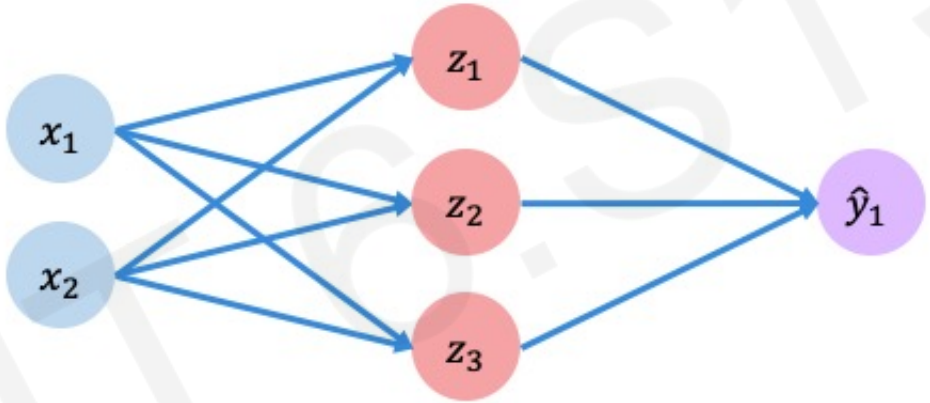
```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

```
loss = torch.nn.functional.cross_entropy( predicted, y )
```

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; \boldsymbol{W}\right)\right)^2$$

Actual Predicted

$f(x)$		y
30	✗	90
80	✗	20
85	✓	95
⋮		⋮

Final Grades (percentage)

```
loss = tf.reduce_mean( tf.square(tf.subtract(y, predicted)) )
loss = tf.keras.losses.MSE( y, predicted )
```

```
loss = torch.nn.functional.mse_loss(
        predicted, y )
```

Training Neural Networks

Loss Optimization

We want to find the network weights that ***achieve the lowest loss***

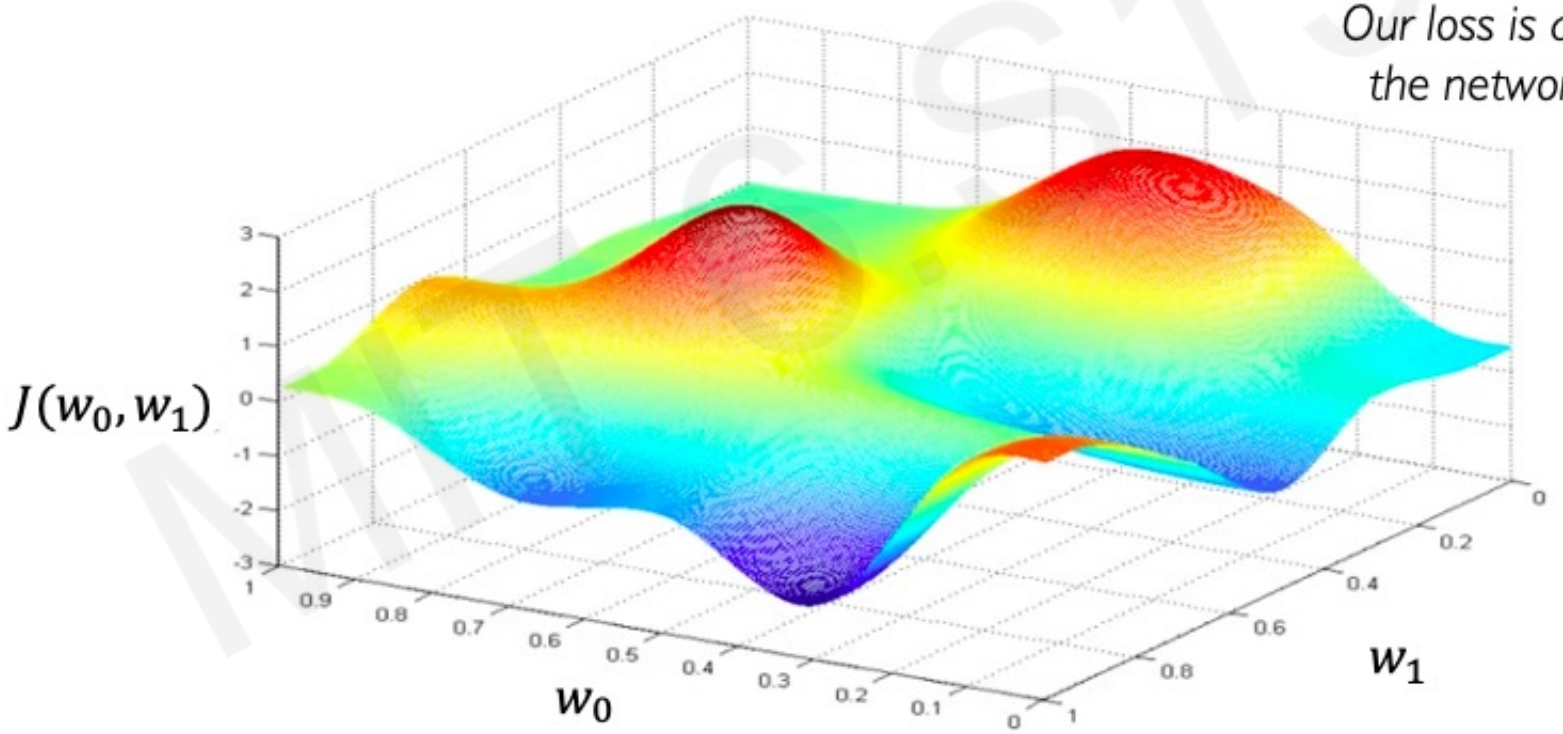
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\text{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\text{argmin}}\, J(\boldsymbol{W})$$

Loss Optimization

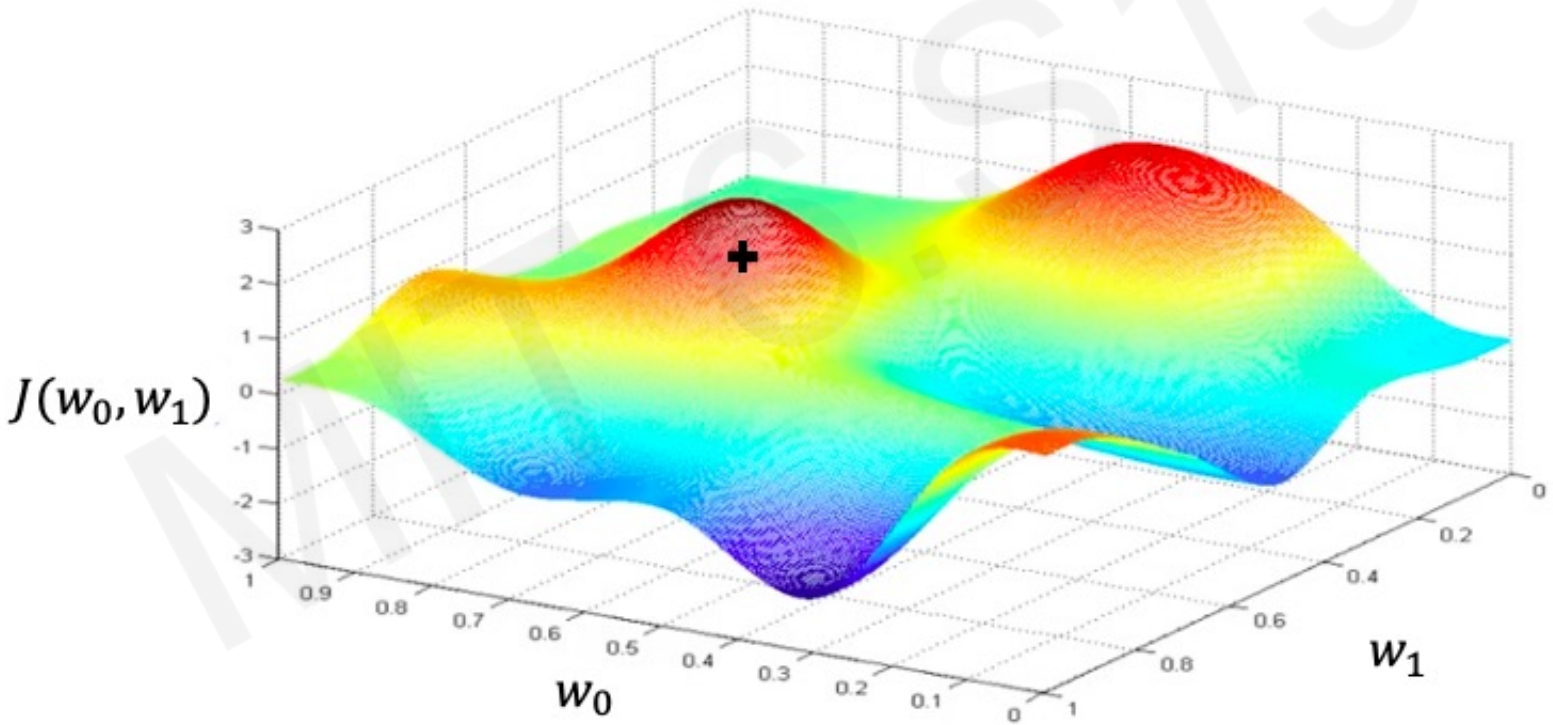
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$

Remember:
Our loss is a function of the network weights!



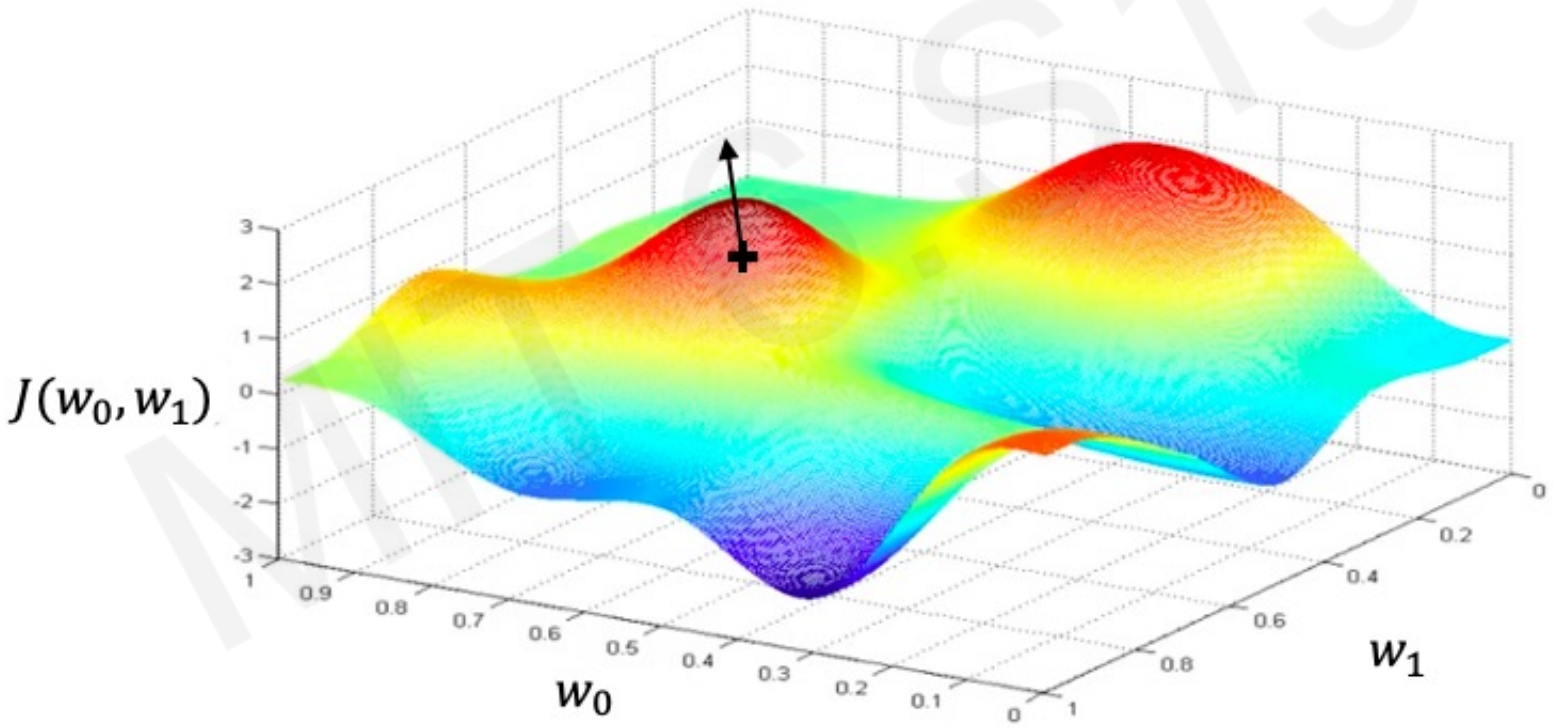
Loss Optimization

Randomly pick an initial (w_0, w_1)



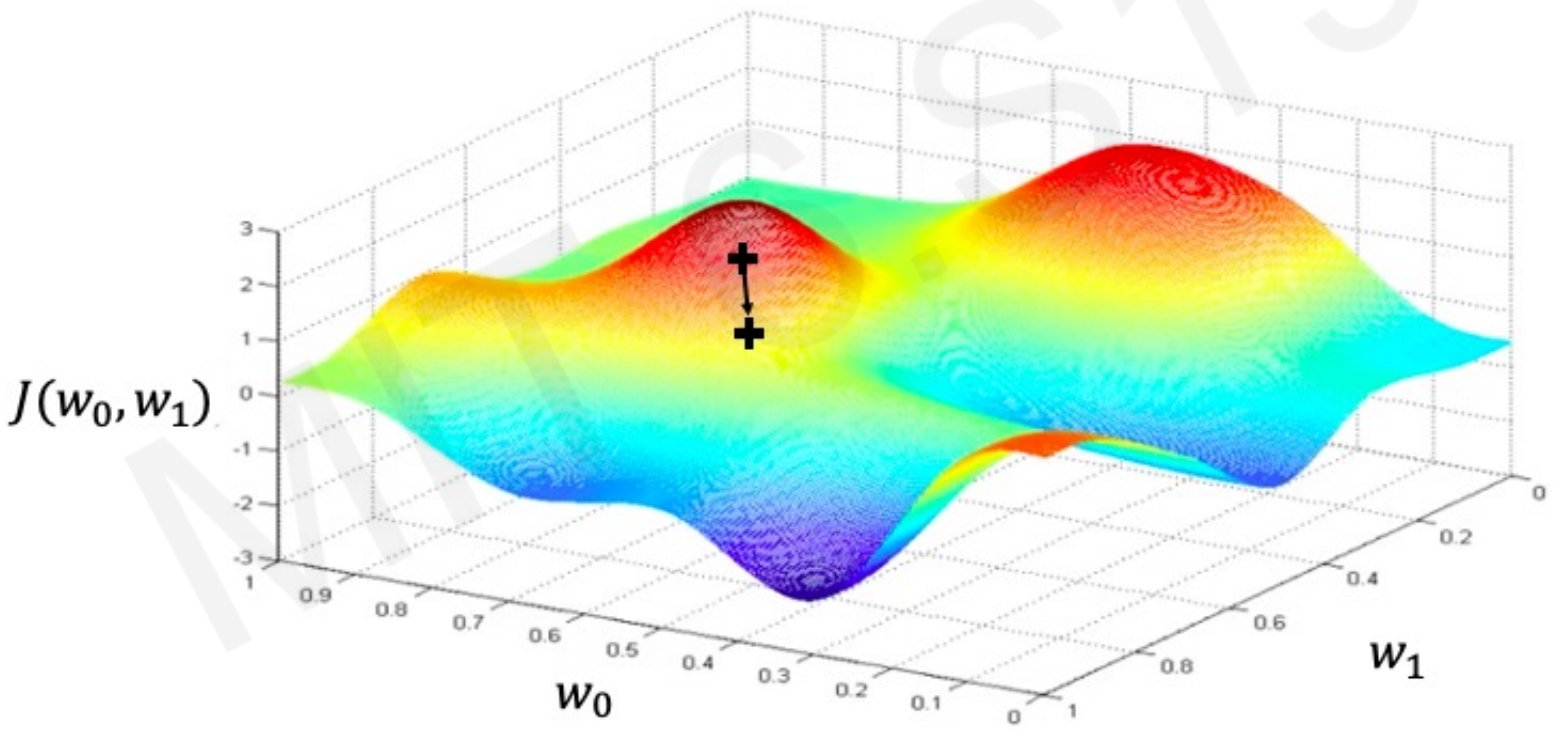
Loss Optimization

Compute gradient, $\frac{\partial J(W)}{\partial W}$



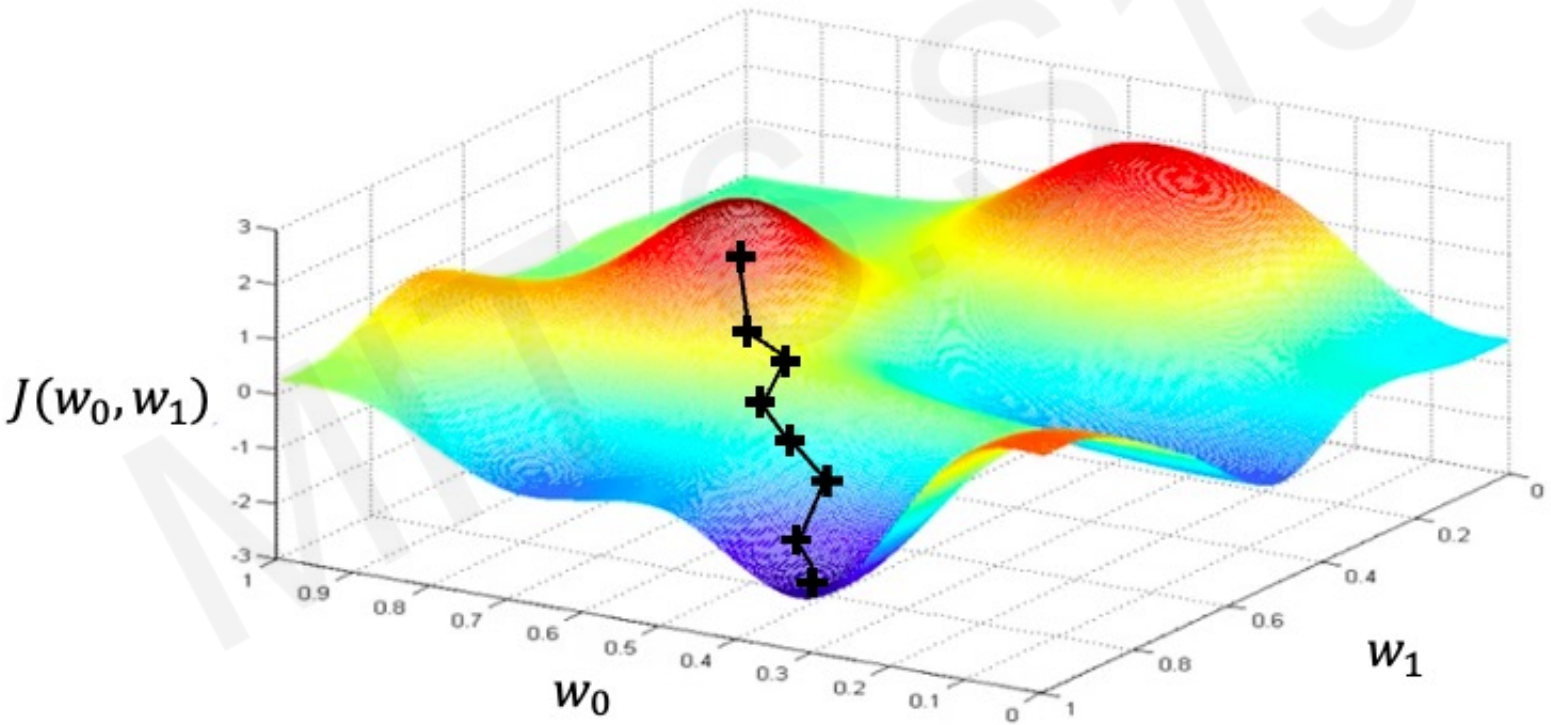
Loss Optimization

Take small step in opposite direction of gradient



Gradient Descent

Repeat until convergence



Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Computing Gradients: Backpropagation

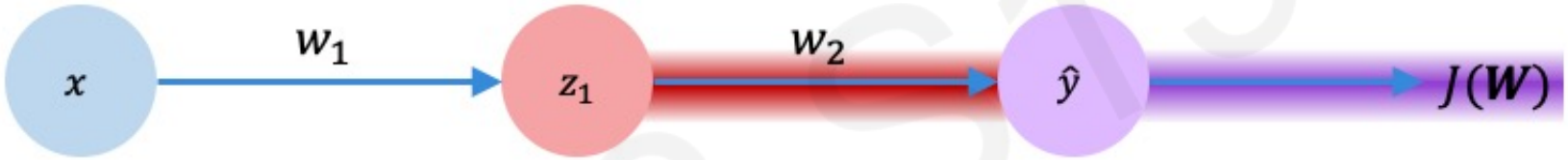
How does a small change in one weight (ex. $\boldsymbol{w_2}$) affect the final loss $\boldsymbol{J(W)}$?

Computing Gradients: Backpropagation

$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} =$$

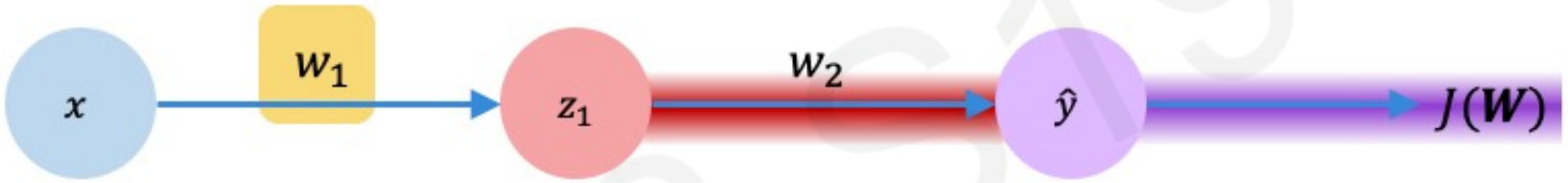
Let's use the chain rule!

Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

Computing Gradients: Backpropagation

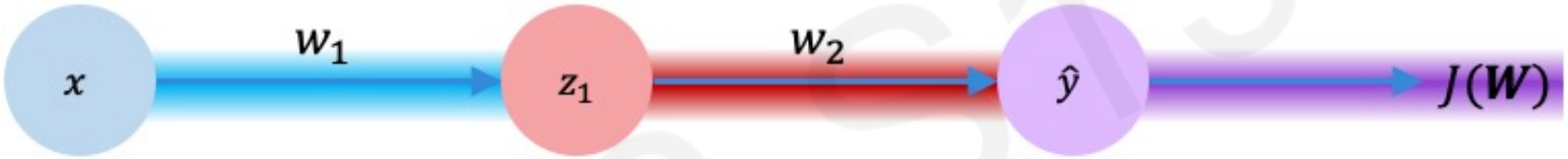


$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

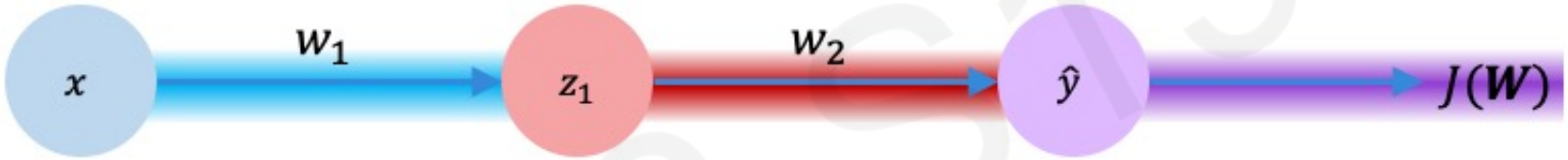
Apply chain rule!

Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Computing Gradients: Backpropagation

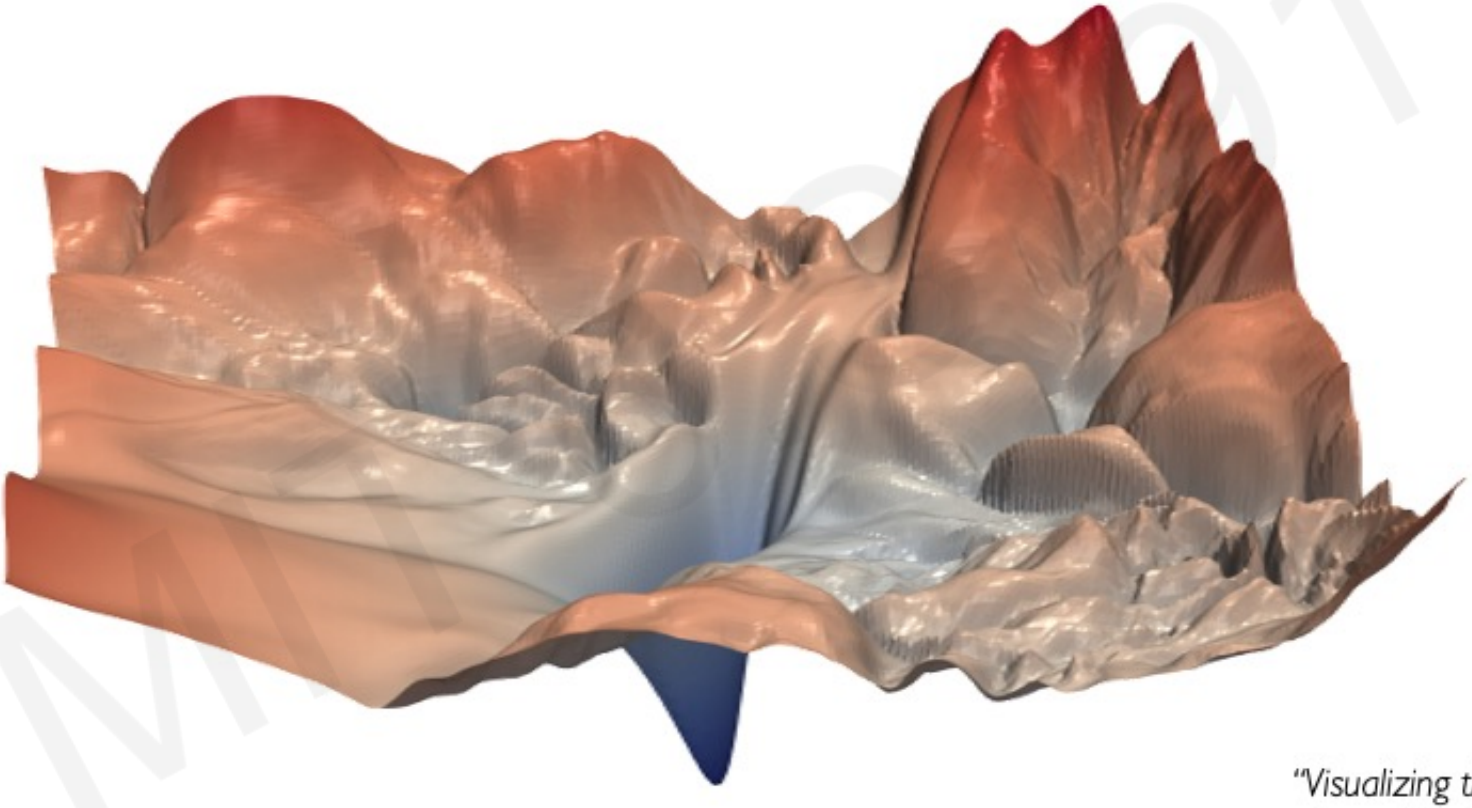


$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for ***every weight in the network*** *using gradients from later layers*

Optimization

Training Neural Networks is Difficult



"Visualizing the loss landscape of neural nets". Dec 2017.

Loss Functions Can Be Difficult to Optimize

Remember:

Optimization through gradient descent

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

Loss Functions Can Be Difficult to Optimize

Remember:

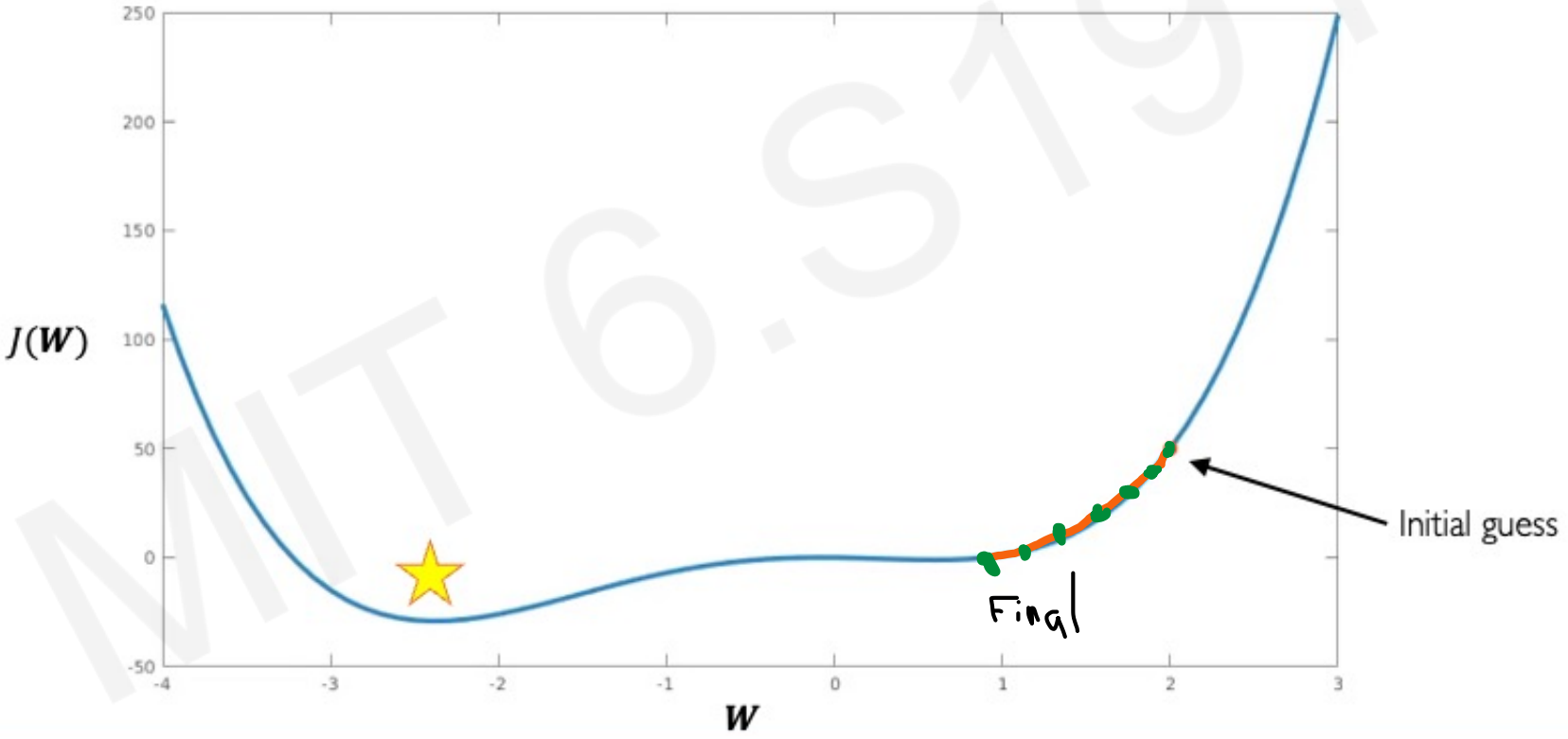
Optimization through gradient descent

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

How can we set the learning rate?

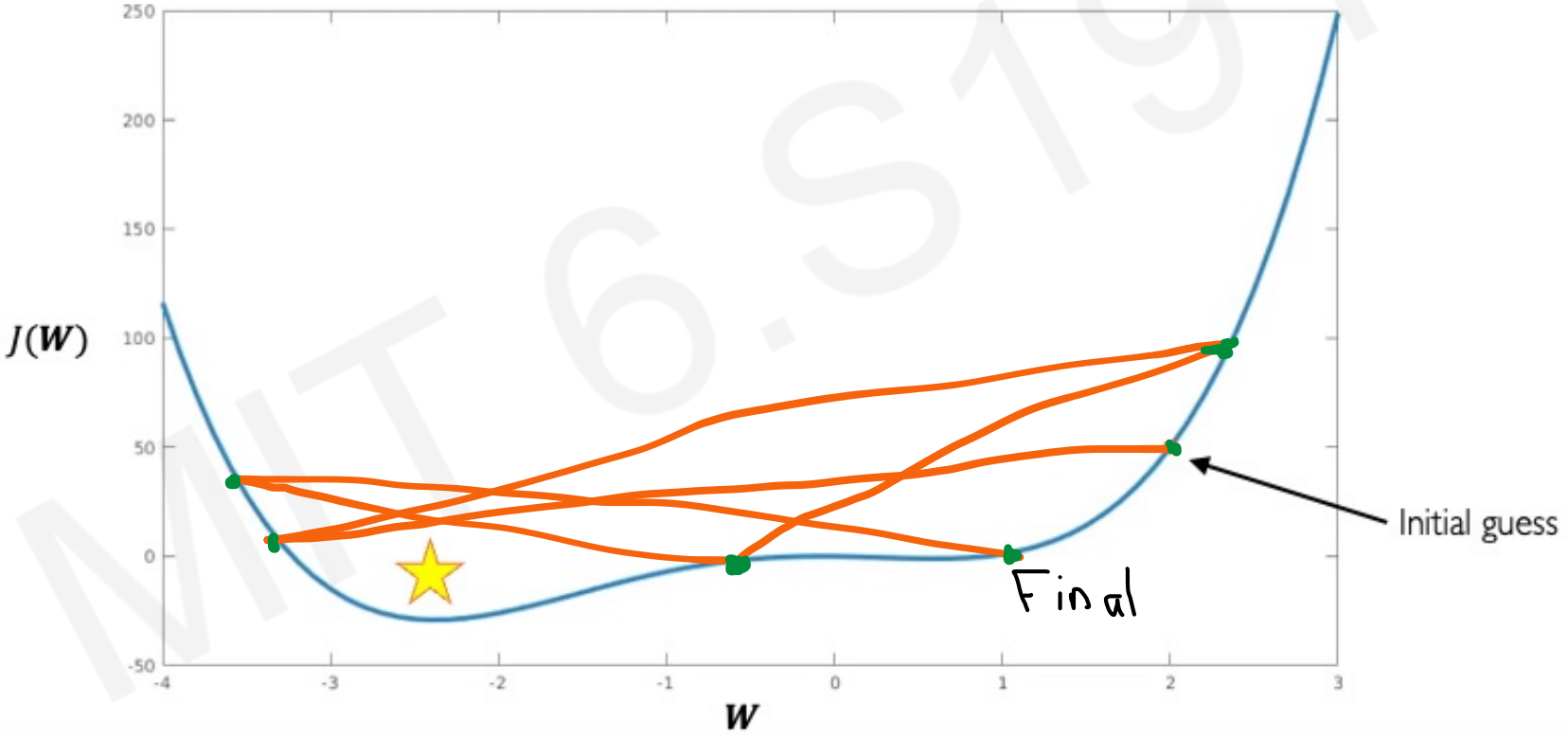
Setting the Learning Rate

Small learning rate *converges slowly and gets stuck in false local minima*



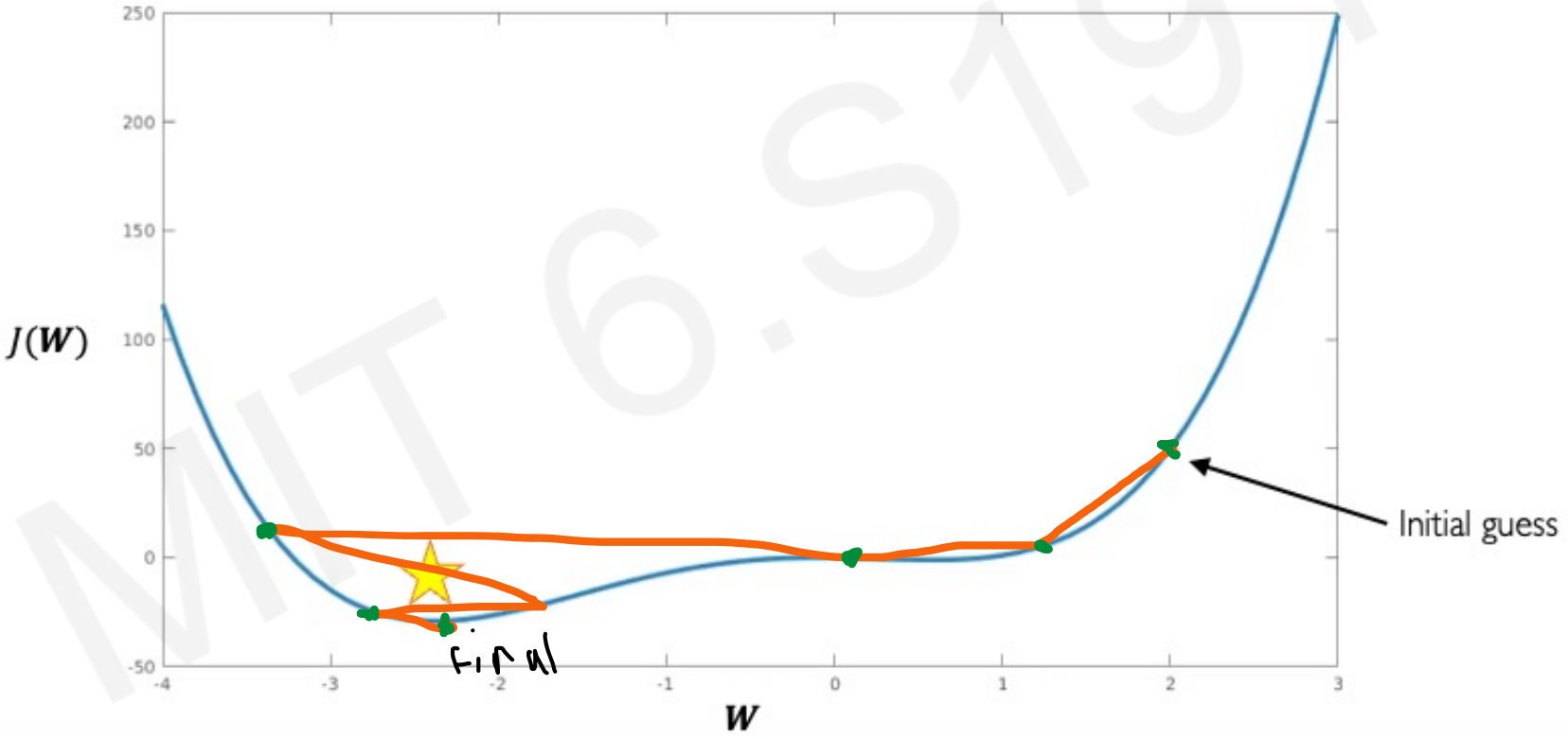
Setting the Learning Rate

Large learning rates *overshoot, become unstable and diverge*



Setting the Learning Rate

Stable learning rates *converge smoothly and avoid local minima*



How to deal with this?

Idea 1:

Try lots of different learning rates and see what works "just right"

How to deal with this?

Idea 1:

Try lots of different learning rates and see what works "just right"

Idea 2:

Do something smarter!

Design an adaptive learning rate that "adapts" to the landscape

Adaptive Learning Rates

- Learning rates are no longer fixed
- Can be made larger or smaller depending on:
  - how large gradient is
  - how fast learning is happening
  - size of particular weights
  - etc...

Gradient Descent Algorithms

Algorithm	TF Implementation	Torch Implementation	Reference
• SGD	`tf.keras.optimizers.SGD`	`torch.optim.SGD`	Kiefer & Wolfowitz, 1952.
• Adam	`tf.keras.optimizers.Adam`	`torch.optim.Adam`	Kingma et al., 2014.
• Adadelta	`tf.keras.optimizers.Adadelta`	`torch.optim.Adadelta`	Zeiler et al., 2012.
• Adagrad	`tf.keras.optimizers.Adagrad`	`torch.optim.Adagrad`	Duchi et al., 2011.
• RMSProp	`tf.keras.optimizers.RMSProp`	`torch.optim.RMSProp`	

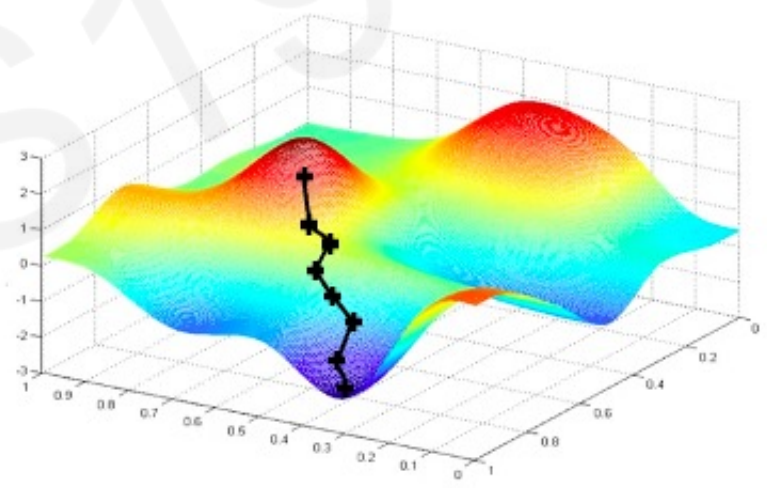
Additional details: http://ruder.io/optimizing-gradient-descent/

Batches

Gradient Descent

Algorithm

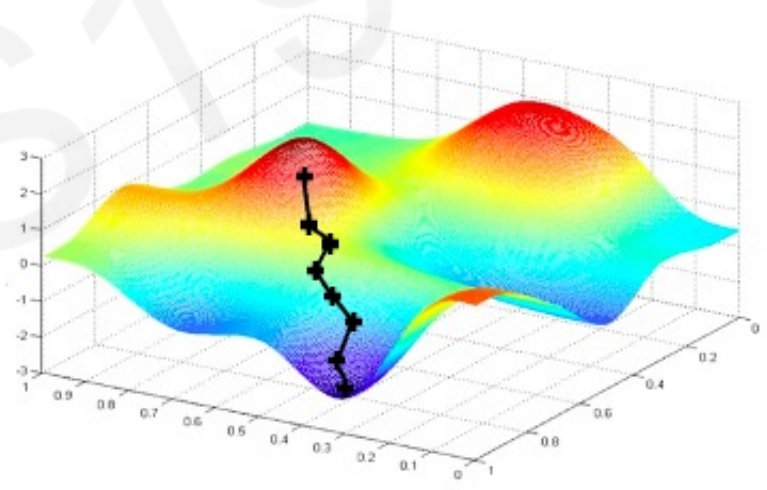
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights



Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

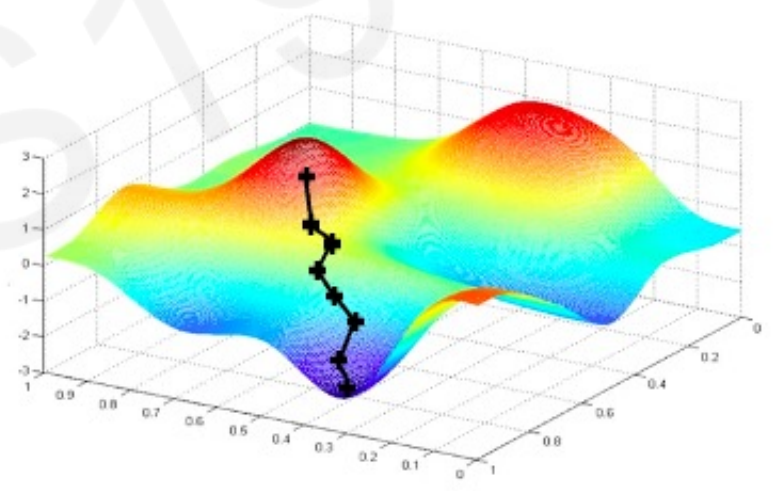


Can be very **computationally intensive** to compute!

Stochastic Gradient Descent

Algorithm

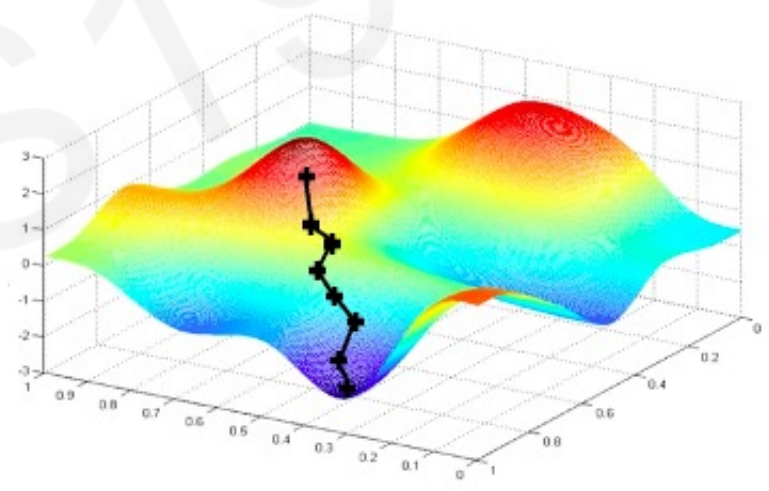
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point $\boldsymbol{i}$
4. Compute gradient, $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(W)}{\partial W}$
5. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
6. Return weights

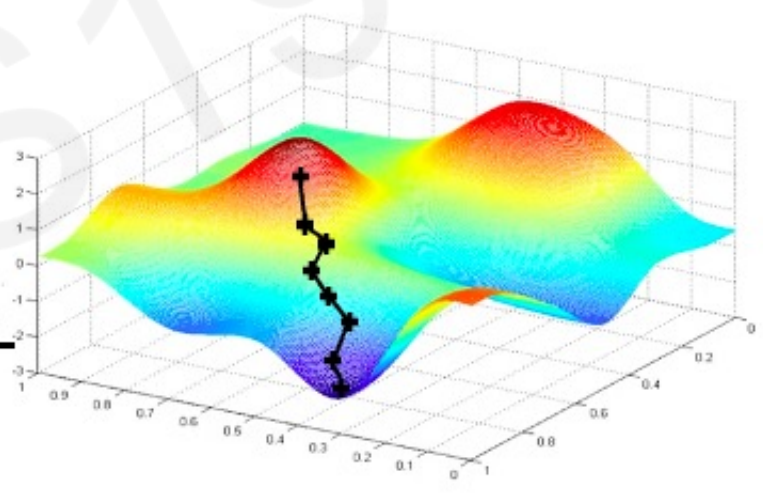


Easy to compute but **very noisy** (stochastic)!

Stochastic Gradient Descent

Algorithm

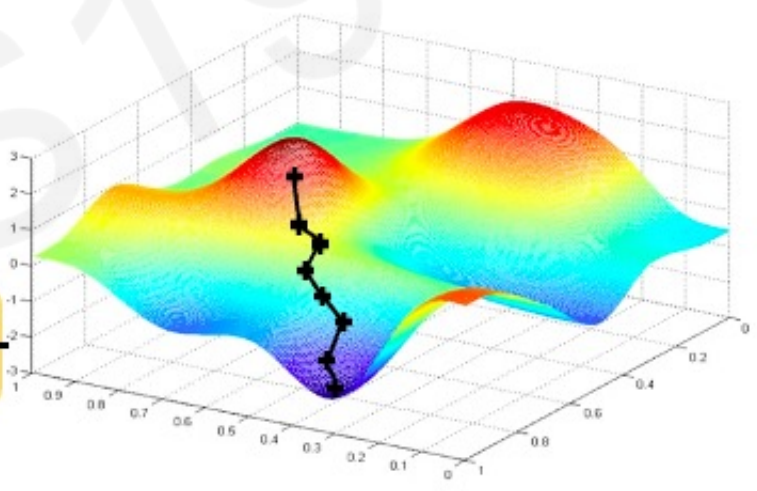
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



Fast to compute and a much better estimate of the true gradient!

Mini-batches while training

More accurate estimation of gradient

Smoother convergence

Allows for larger learning rates

Mini-batches while training

More accurate estimation of gradient

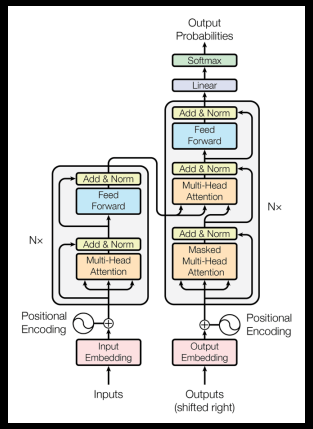
Smoother convergence

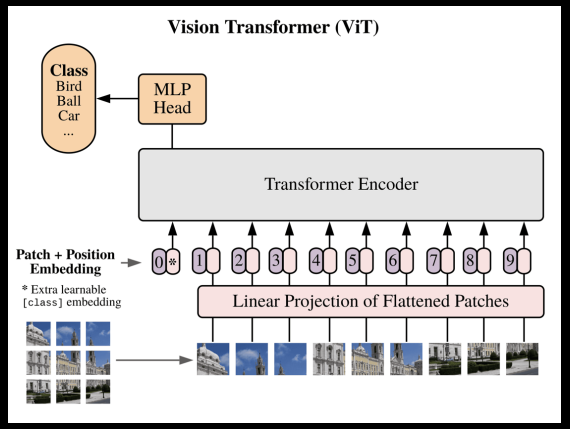
Allows for larger learning rates

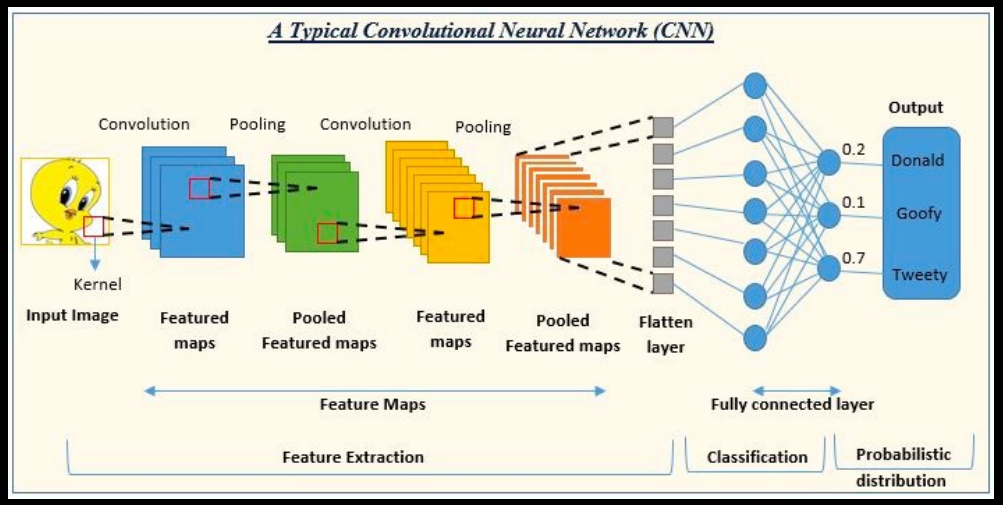
Mini-batches lead to fast training!

Can parallelize computation + achieve significant speed increases on GPU's

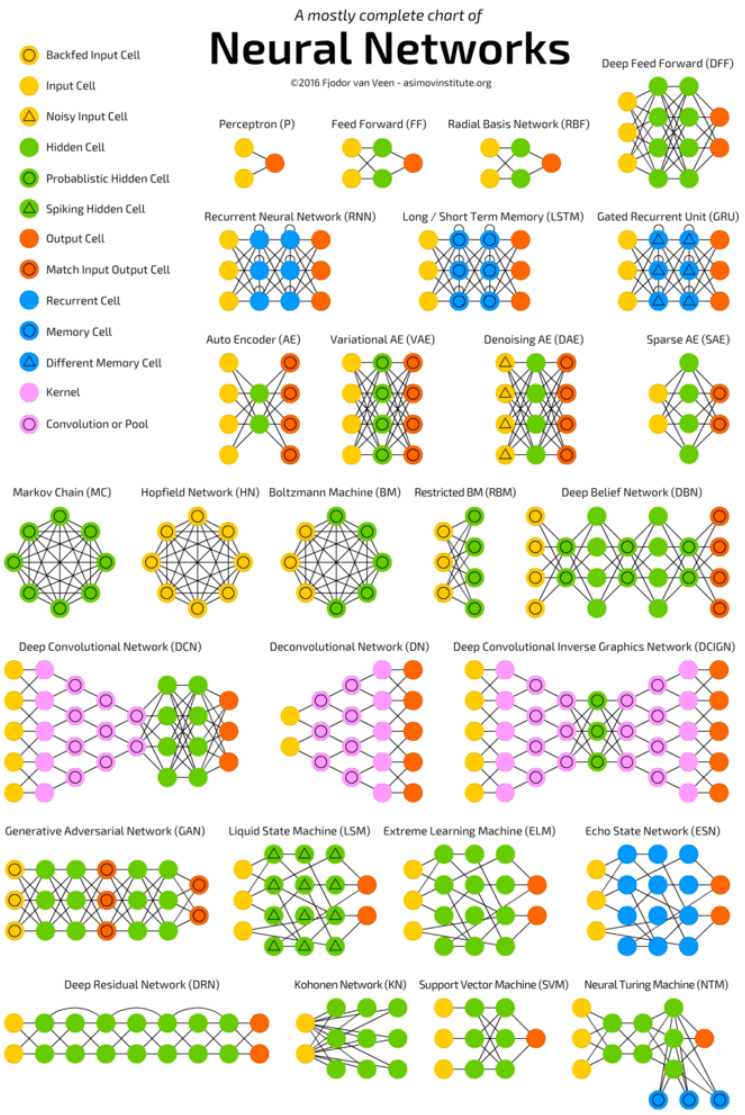
Types of Neural Networks







https://deepgram.com/learn/deep-learning-101-building-blocks-of-machine-intelligence



Use cases of Deep Learning

- Natural Language Processing (NLP)
  - Machine Translation: Google Translate
  - Chatbots and virtual assistants: Siri, ChatGPT

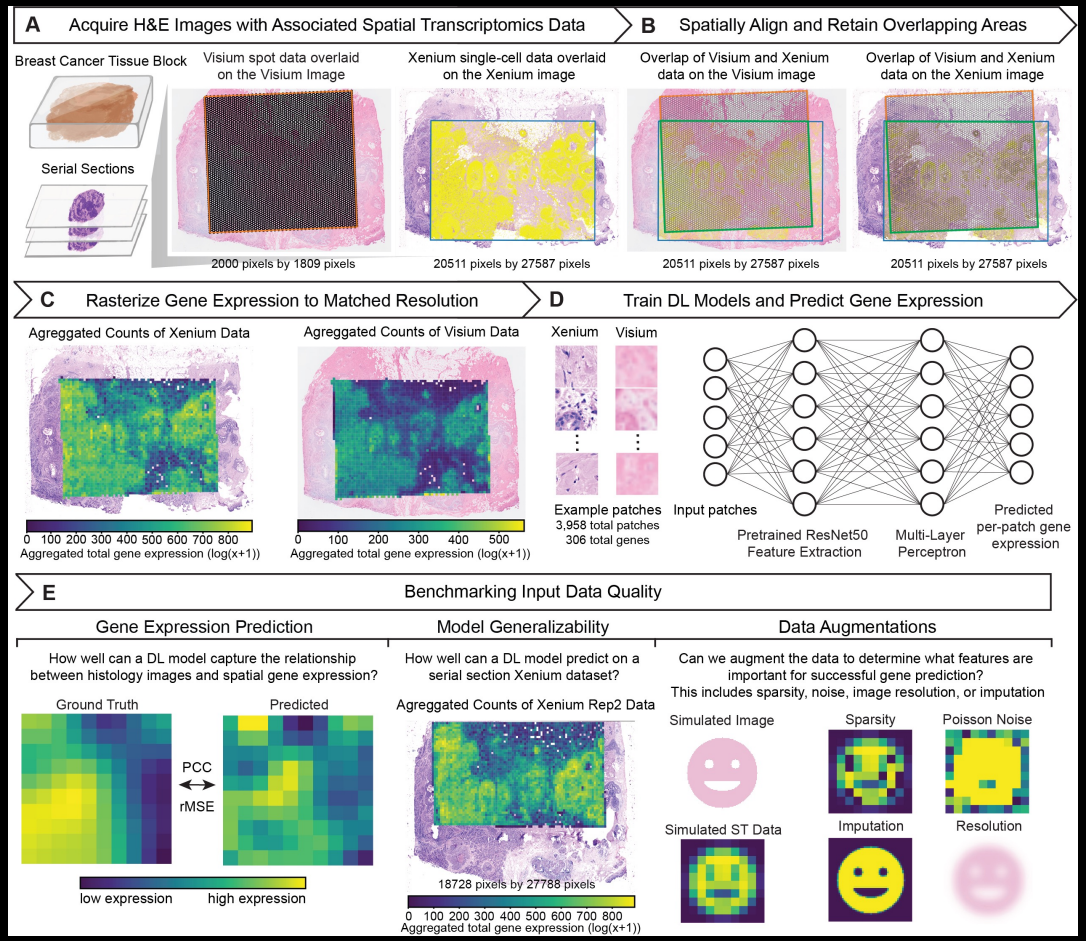
https://thispersondoesnotexist.com/

- Generative AI
  - Image generation
  - Music and audio generation

- Computer Vision
  - Image classification
  - Autonomous vehicles

Many, MANY more

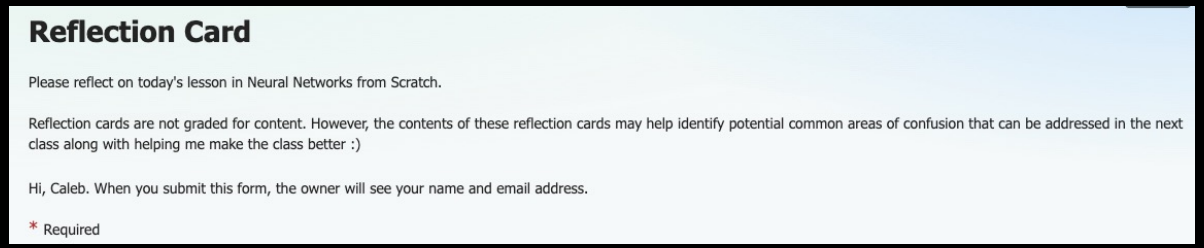
My Research – gene prediction from histology images



Next class

- We will begin our in-depth adventure of neural networks!
  - Learn how the forward pass through a NN works and why

Reflection Cards



- Essentially a means to help me make this class better!
  - NOTE: for this first class please put a a number from 1-10 on how comfortable you are in Python for question 3

1. What is something that you learned in today's lecture?
2. What is something that you are still confused about from today's lecture?
3. Do you have any other comments/feedback/thoughts/suggestions/concerns?

https://forms.office.com/r/Kv8LtW4iJH