

# Introduction to Deep Learning



# AGENDA

---

Introduction

---

Deep Learning in Practice

---

Evaluating Models

---

Optimizing Models

---

Summary

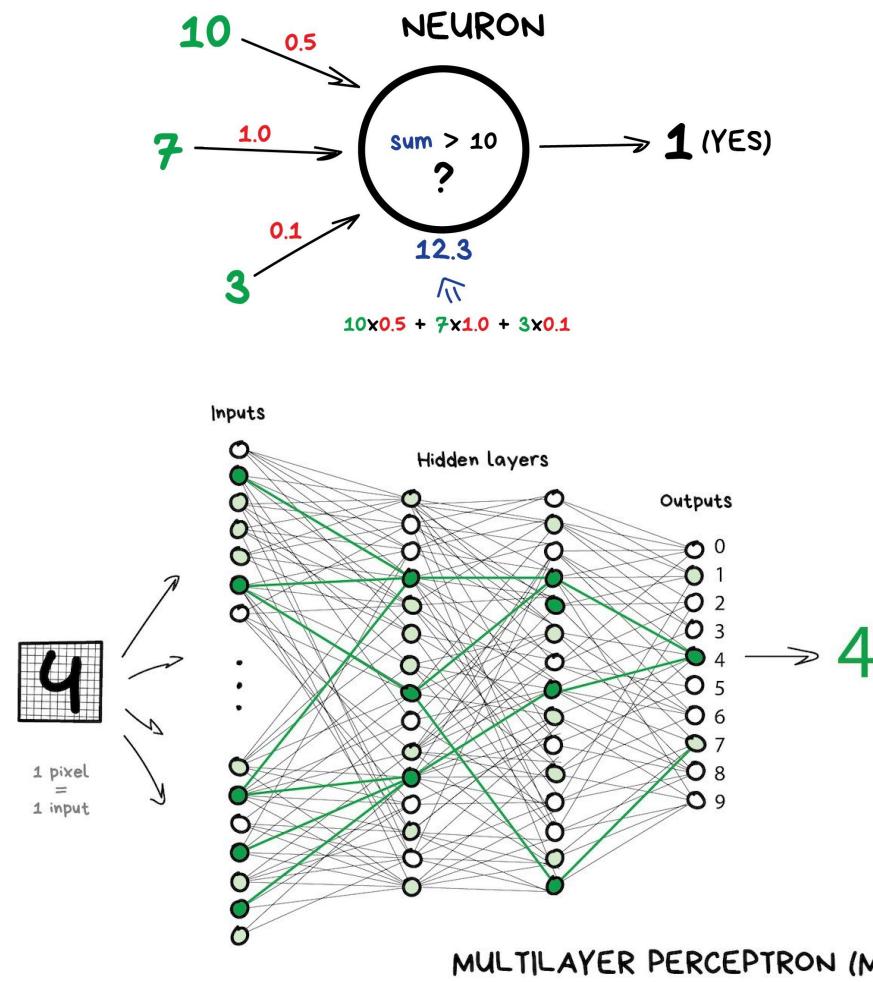
---

# Introduction

# WHAT IS DEEP LEARNING?

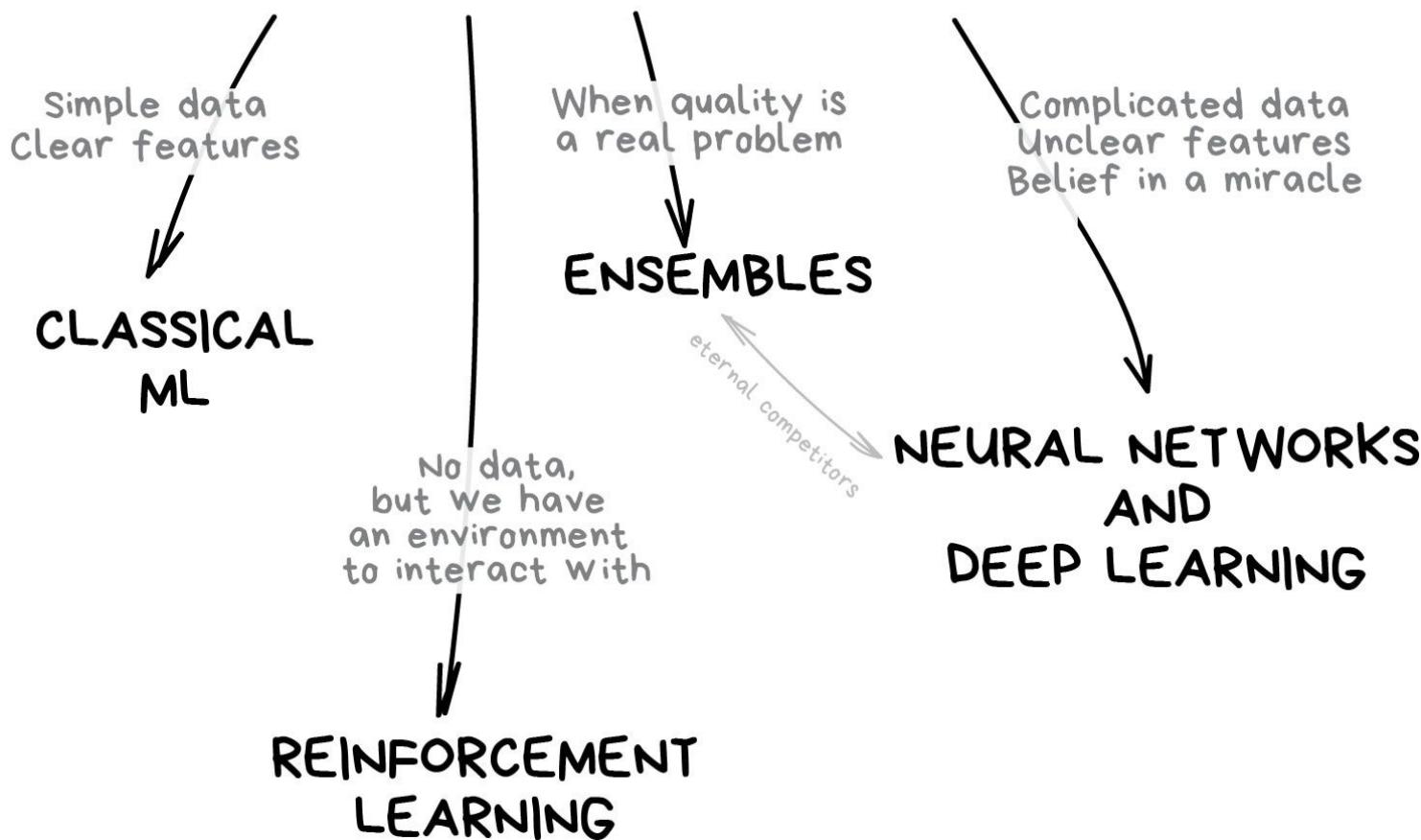
Deep Learning is a type of machine learning inspired by the structure of the brain

- Algorithm perform a task numerous times, via a system of weighted connections, making improvements each time
- Referred to as ‘deep’ learning because inputs are processed via deep layers that enable learning



# THE MACHINE LEARNING LANDSCAPE

## THE MAIN TYPES OF MACHINE LEARNING



# WHEN TO USE DEEP LEARNING

- Complex tasks
- Unstructured but large amount of data
- You do not need a highly interpretable model
- You have the time and/or resources to train a model

# DEEP LEARNING EXAMPLES

Object Identification and Facial Recognition

Speech Recognition - Virtual Assistants, apps for identifying songs

Deep Fakes

Self-Driving Vehicles

Personalized Entertainment

# DEEP LEARNING EXAMPLES - OBJECT RECOGNITION

Developed algorithm to speed up checkout at bakeries

A doctor saw a segment about the BakeryScan and realized that cancer cells, under a microscope, looked kind of like bread

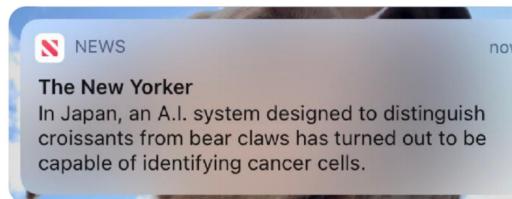


Aaron Fullerton

@AaronFullerton

:

Shoot for the moon. Even if you miss, you'll land among the stars.



ANNALS OF TECHNOLOGY

## THE PASTRY A.I. THAT LEARNED TO FIGHT CANCER

*In Japan, a system designed to distinguish croissants from bear claws has turned out to be capable of a whole lot more.*

By James Somers

March 18, 2021

<https://www.newyorker.com/tech/annals-of-technology/the-pastry-ai-that-learned-to-fight-cancer>

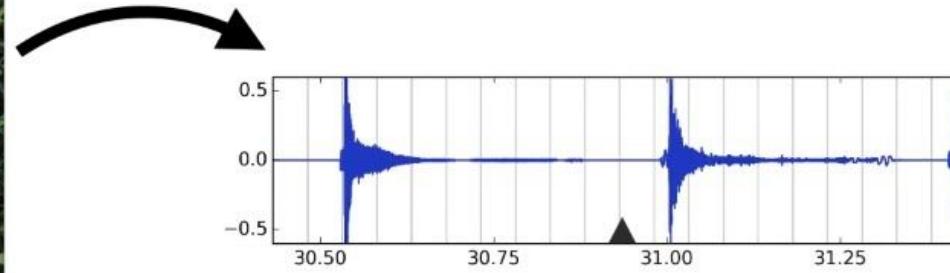
# DEEP LEARNING EXAMPLES - PRODUCING SOUND

MIT researchers developed an algorithm that can produce a sound for a silent video that is realistic enough to fool human viewers

Data consisted of 1,000 videos of an estimated 46,000 sounds



Silent video

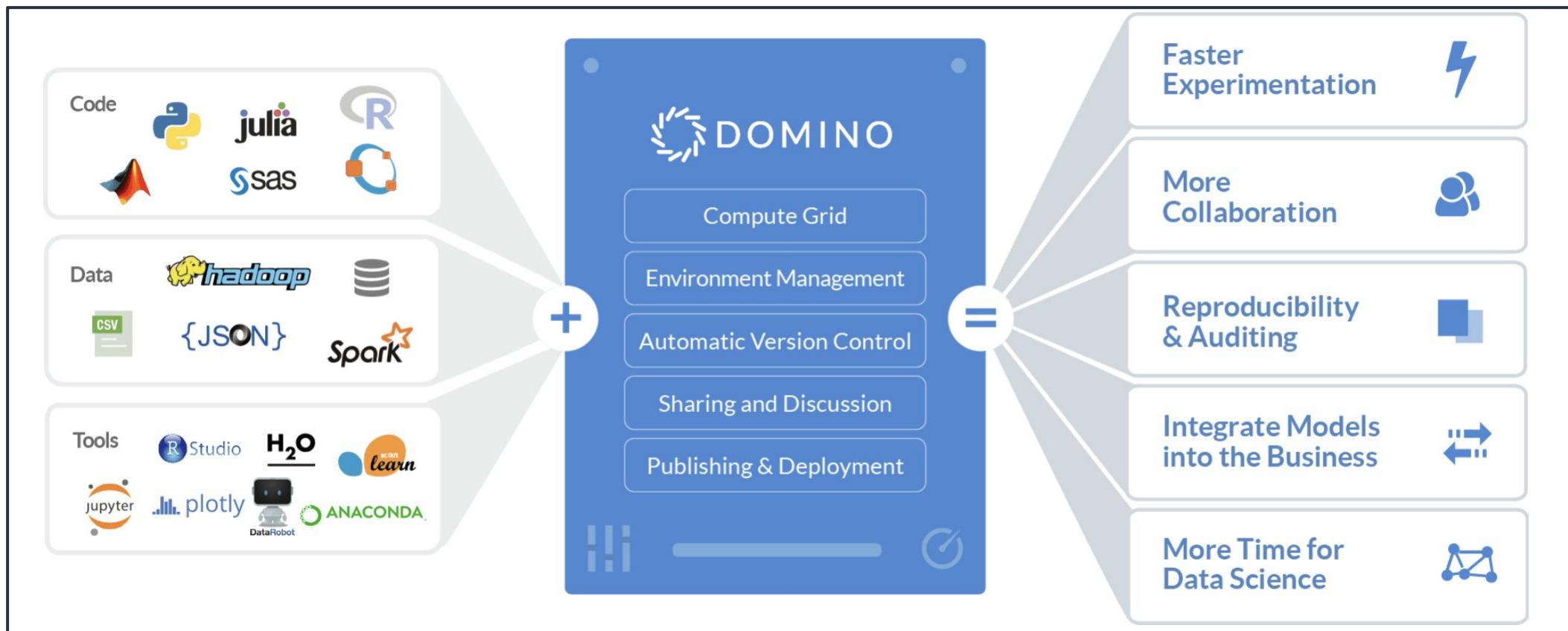


Predicted soundtrack

# Intro to Domino

# What is Domino?

Domino makes it easy to access scalable compute, update and share environments, keep track of your experiments and file changes, and easily deploy models



# Projects: The Main Unit of Organization

- Projects hold all of your files and revision history
- You will run code inside of a project
- Project settings are basic options on how you run your code:
  - Compute environments (software your project needs to run)
  - Hardware tiers (compute resources your project runs on)
  - Access control - who can see and collaborate on your projects

# Project Settings: Hardware Tiers

- Compute resources for that job or workspace
- Your Domino admin defines these
  - can be custom for organizations
- Choose a hardware tier that will meet the needs of your workflow

**Project settings**

Hardware & Environment   Access & Sharing   Results   Exports

**Hardware tier**

Tier	Resources	Cost
<b>Small</b>	1 core · 4 GB RAM · \$0.002/min	< 1 MIN ⓘ
<b>Small</b>	1 core · 4 GB RAM · \$0.002/min	< 1 MIN ⓘ
<b>Large.2</b>	6 cores · 28 GB RAM · \$0.012/min	< 1 MIN ⓘ
<b>Large.2-spark</b>	6 cores · 24 GB RAM · \$0.012/min	< 1 MIN ⓘ
<b>Medium</b>	2 cores · 8 GB RAM · \$0.03/min	< 1 MIN ⓘ
<b>Large</b>	64 cores · 256 GB RAM · \$0.15/min	< 7 MIN ⓘ
<b>GPU (1 V100) - Global</b>	6 cores · 28 GB RAM · \$0.408/min	< 7 MIN ⓘ

# Project Settings: Compute Environments

Specification for the container where your job or workspace will run

Automatically versioned and easily shared

Domino comes with [various base images](#) and it's easy to add packages and libraries, learn more in [Domino 201](#)

The screenshot shows the 'Compute environment' settings page. At the top, there is a search bar containing 'Domino Analytics Distribution Py3.6 R3.6'. To the right of the search bar is a 'Manage Environment' button and a 'What's this ?' link. Below the search bar, there are three sections: 'Global', 'test', and 'Private'. The 'Global' section contains a list of compute environments, with 'Domino Analytics Distribution Py3.6 R3.6' highlighted. Other items in the list include 'Domino Analytics Distribution py3.6 R3.5 - shap & lime', 'HyperOpt Domino Analytics Distribution Py3.6 R3.6', 'Image\_Drift\_Keras\_TF', 'OpenAI Gym with Python 3.6', and 'Tensorflow 2.0 Py3.6 R3.5'. The 'test' section contains 'testenv'. The 'Private' section contains a note: 'By default, values are passed verbatim, without any escaping.' On the right side of the interface, there is a vertical sidebar with a 'What's this ?' link.

# Project Settings: Access and Sharing

Set the visibility of your project

Project settings

Hardware & Environment Access & Sharing Results Exports Integrations Archive Project or Transfer Ownership

Visibility Learn more ?

- Public: Viewable by anyone
- Searchable: Discoverable by other Domino users
- Private: Only viewable or discoverable by your collaborators
- Allow run execution by anonymous users

Project Name & Description

Name

Intro-to-Domino

Rename Project Cancel

Description (optional)

This project contains the code, data, and artifacts for the Getting Started in Domino Tutorial, found here for Python and here for R.

A description makes it easier for colleagues to find your project.

Save Cancel

# Project Settings: Access and Sharing

Add colleagues or groups (via organizations) to your project and set permission levels

**Collaborators and permissions** Learn more ?

**Invite Collaborators**  
If your project has Git repositories connected, then collaborators will also need permissions with your Git host in order to start a workspace or make changes to Git files. [Learn more about working with Domino and Git](#).

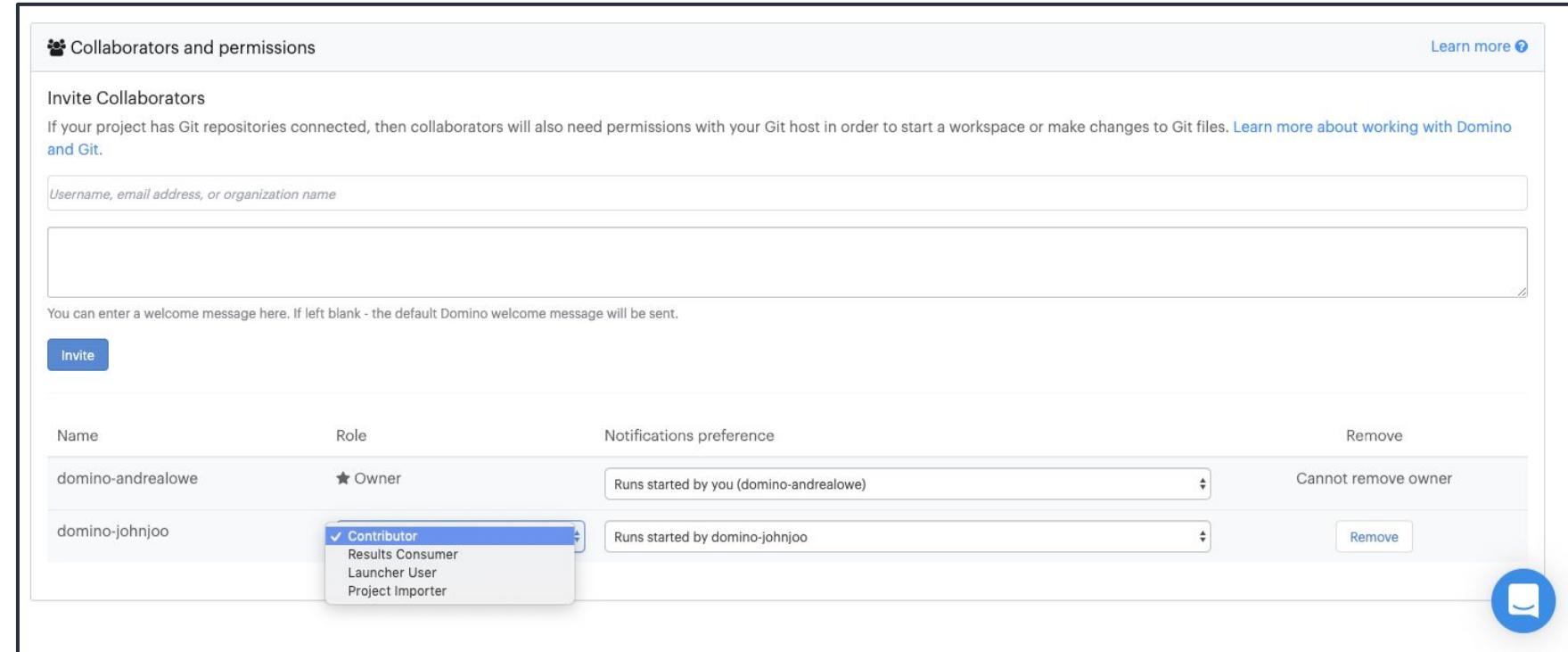
Username, email address, or organization name

You can enter a welcome message here. If left blank - the default Domino welcome message will be sent.

**Invite**

Name	Role	Notifications preference	Remove
domino-andrealowe	★ Owner	Runs started by you (domino-andrealowe)	Cannot remove owner
domino-johnjoo	Contributor	Runs started by domino-johnjoo	<b>Remove</b>

Contributor  
Results Consumer  
Launcher User  
Project Importer



# Project Settings: Permissions

Collaborators on your projects can have different [permissions levels](#)

- Contributor - read/write files, start runs
- Results Consumer - read files and access published apps
- Launcher User - run Launchers and access those results
- Project Importer - import the project
- Owner - archive, change collaborator types, alter settings

# Projects: Fork/Merge

domino-andrea-admin

- Intro-to-Dom...
- Ideation

Quick Action

Overview

Activity

Workspaces

Jobs

Scheduled Jobs

Files

Datasets

Reviews

Publish

Settings

Small (Kubernetes)

Getting-Started...

< Back to Projects

## Intro-to-Domino

Updated May 26th 2020

About Manage

### Tags & Description

Tags: No tags [+](#)

Description: No description provided. [Edit](#)

### Readme

This project contains the code, data, and artifacts for the *Getting Started in Domino* Tutorial, found [here](#) for Python and [here](#) for R.

This tutorial will guide you through a common model lifecycle in Domino. You will start by working with data from the Balancing Mechanism Reporting Service in the UK. We will be exploring the Electricity Generation by Fuel Type and predicting the electricity generation in the future. You'll see examples of Jupyter, Dash, pandas, and Prophet used in Domino.

Table of Contents:

- Forecast\_Power\_Generation.ipynb
- Scheduled\_Forecast\_Power\_Generation.ipynb
- Forecast\_Power\_Generation\_for\_Launcher.ipynb
- forecast.ipynb
- forecast\_predictor.py
- data.csv
- app.sh

Fork domino-andrea-admin/Intro-to-Domino

i Forking this project will create a copy of it under your account, so you can modify it as you wish. Your new project will contain the same files, but none of the Run history or project settings.

Name of new project

domino-andrea-admin

[Cancel](#) [Fork](#)

Not used for Git-Based Projects

Hardware Small (Kubernetes)

Environment Getting-Started-in-Domino

Total Runtime 9 days

Apps 1

Jobs 1

Workspaces 16

Comments 0

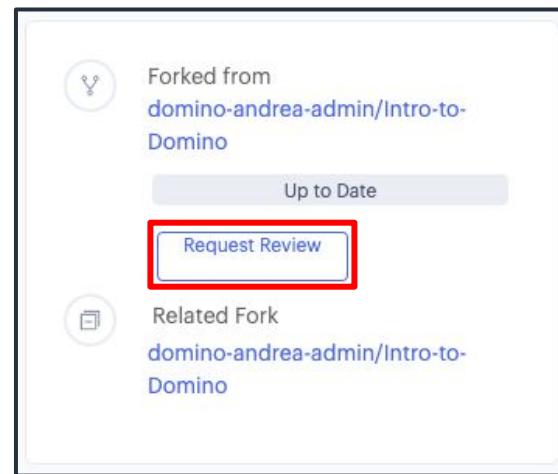
DOMINO

# Projects: Fork/Merge

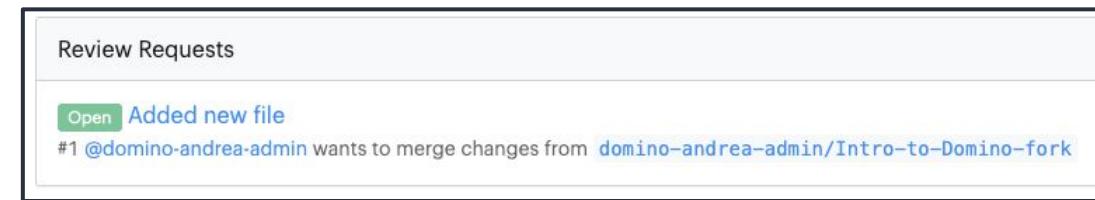
Bring changes from original project into your forked project:



Request to merge changes from forked project into original:



Review changes from forked versions of your project:



# Projects: Versioning and Reproducibility

You can revert your project or any individual file to any previous state

In Git-Based Projects this page is labeled Artifacts

The screenshot shows the Domino platform interface. On the left, a sidebar menu includes options like Launch, Overview, Files, Workspaces, Jobs, Scheduled Jobs, Publish, Reviews, Activity, Settings, Small, and Domino Analytics. The main area displays a list of files under the project 'Intro-to-Domino'. A message at the top indicates a recent commit: 'domino-andrealowe committed fd4440b "Reverted requirements.txt" (job #48) on April 2, 2020 @ 04:17 pm'. Below this, a note says 'This is an old version of your files . The latest version was made at April 2nd 2020 @ 4:17 pm' with a link to 'View Latest'. The file list includes: results, .ipynb\_checkpoints, data.csv, README.md, model.pkl, forecast\_launcher.sh, requirements.txt, .dominoresults, Forecast\_Power\_Generation.ipynb, forecast.ipynb, app.py, Forecast\_Power\_Generation\_for\_Launcher.ipynb, forecast\_predictor.py, and app.sh. The 'Revert Project' button in the top right corner is highlighted with a red box.

Name	Size	Modified
results	4.9 KB	
.ipynb_checkpoints	1.2 MB	
data.csv	113 KB	April 2nd 2020, 12:31:07 pm
README.md	837 B	February 28th 2020, 3:01:49 pm
model.pkl	0 B	April 2nd 2020, 12:31:48 pm
forecast_launcher.sh	138 B	February 12th 2020, 5:40:24 pm
requirements.txt	63 B	February 12th 2020, 5:29:17 pm
.dominoresults	313 B	February 6th 2020, 12:38:59 pm
Forecast_Power_Generation.ipynb	386 KB	April 1st 2020, 5:26:38 pm
forecast.ipynb	604 KB	March 30th 2020, 12:31:26 pm
app.py	9.3 KB	February 13th 2020, 9:15:14 am
Forecast_Power_Generation_for_Launcher.ipynb	389 KB	February 12th 2020, 5:39:39 pm
forecast_predictor.py	378 B	February 12th 2020, 5:45:05 pm
app.sh	13 B	February 13th 2020, 9:15:41 am

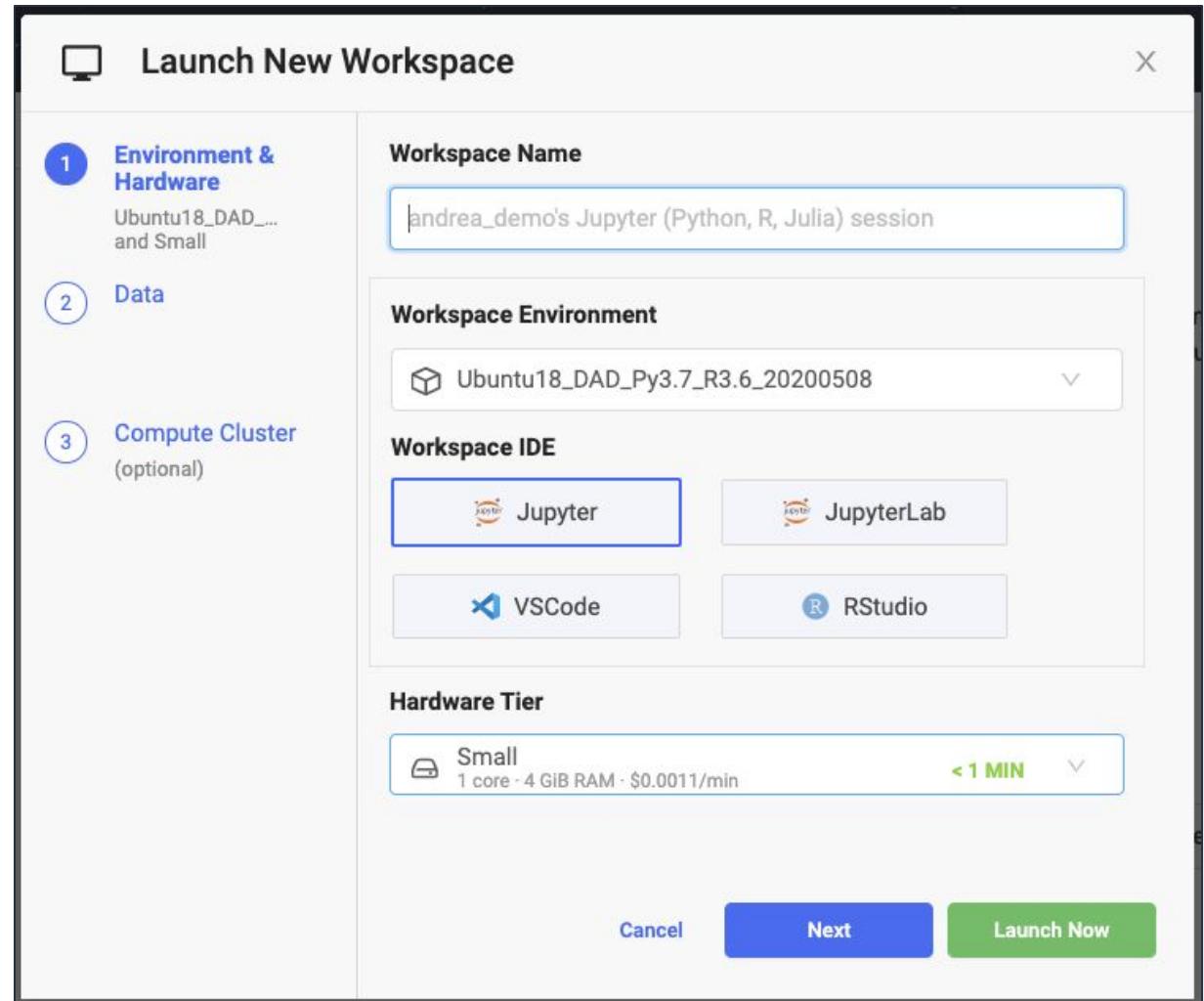
# Run Code in Domino

	<u>Workspaces</u>	<u>Jobs</u>
<b>For example</b>	RStudio Jupyter Notebook SAS Studio	my_script.py another_script.R wrapper_script.sh
<b>Saving to Domino</b>	Manual by default	Automatic
<b>Resource shutdown</b>	Manual	Automatic

# Configure a Workspace

Configure your workspace at launch:

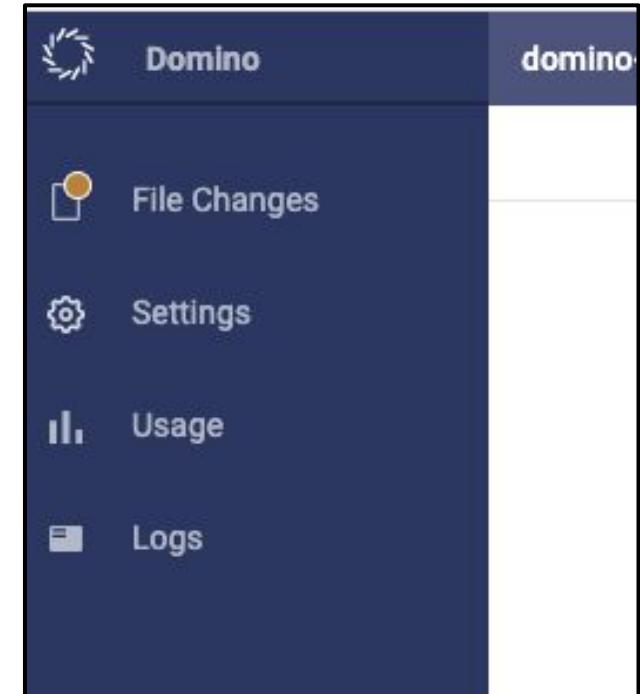
- Name (optional)
- Hardware Tier
- Environment
- Workspace IDE
- Data
- On-Demand Spark Cluster



# Workspace Navigation Bar

On the left of the screen are the panes for

- [File Changes](#), where changes to your files are tracked and synced
- [Settings](#), where you can see and edit your workspace configuration
- [Usage](#), where you can view how your workspace is consuming resources
- [Logs](#), where you can see the setup and user logs your workspace is generating



# Workspace Navigation Bar: Resource Usage

The screenshot shows the Domino workspace interface. On the left is a dark sidebar with icons for File Management, Settings, Usage (which is selected), and Logs. The main area is titled "domino-austin's JupyterLab session". It features a "Resource Usage" section with two graphs: "CPU Usage" (73.10% over 5 minutes) and "Memory Usage" (164 MB / 1024 MB). To the right is a file browser pane showing a directory structure with files like "bignb.ipynb", "Untitled.ipynb", "Untitled1.ipynb", and various Python and HTML files.

In the [Usage](#) pane, you'll find graphs of your workspace [CPU Usage](#) and [Memory Usage](#) over time.

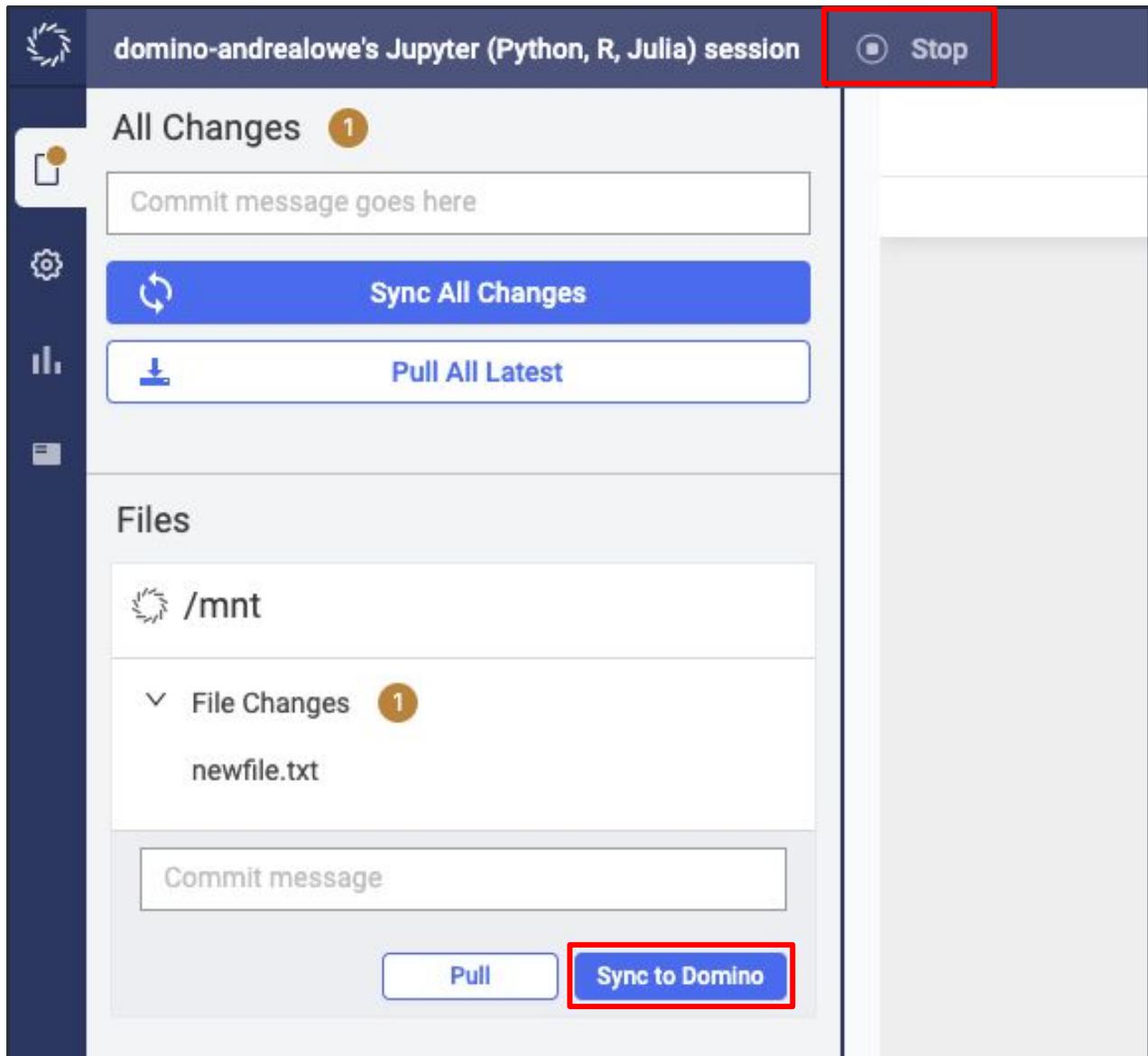
# Workspace Sync and Stop

Sync work in left navigation bar

Important - Stop session when finished to save on costs

Closing the browser tab does not stop a workspace

You can Stop workspaces without committing files



# Workspace Dashboard

The workspaces dashboard will show all stopped and running workspaces

See the state of the environment, files, and hardware for every workspace

The screenshot shows the Domino Workspace Dashboard interface. At the top, there are tabs: 'My Workspaces' (with 1 workspace), 'All Workspaces' (selected), 'Deleted', and 'Utility'. On the right, there are buttons for '+ Create New Workspace' and 'Open Last Workspace'. Below the tabs, there are two workspace cards.

**Running Workspace:** This card shows a laptop icon with a Jupyter logo on the screen. The workspace is owned by 'andrea-demo'. It has a green 'Running' status indicator. Below the icon are 'Open' and 'Stop' buttons. To the right, it displays the session details: 'andrea-demo's Jupyter (Python, R, Julia) session' started on 'Feb 16th, 2021 @ 04:01:09 PM'. A red box highlights the 'Settings', 'Usage', 'Logs', and 'History' buttons at the top right of this card.

**Stopped Workspace:** This card shows a laptop icon with a Jupyter logo on the screen. The workspace is owned by 'andrea-demo'. It has a blue 'Stopped' status indicator. Below the icon is a 'Start' button. To the right, it displays the session details: 'andrea-demo's Jupyter (Python, R, Julia) session' started on 'Feb 16th, 2021 @ 04:00:50 PM' with a '00:00:47' last uptime. A red box highlights the 'Settings', 'Usage', 'Logs', and 'History' buttons at the top right of this card.

# Domino Overview



# Access Materials

1. Log into [temporary training instance decommissioned]
2. Navigate to [temporary training instance decommissioned]
3. Fork the project

The screenshot shows a project overview page. On the left, there's a sidebar with icons for Home, Projects, Ideation, Overview (which is selected and highlighted in blue), and Activity. The main content area displays the project details:

- Project Name:** Deep-Learning-Tutorial
- Owner:** domino-andrea
- Status:** Ideation
- Last Updated:** January 26th 2022
- Tags:** No tags
- Description:** No description provided.

On the right side of the main content, there are two buttons: a red-bordered 'Y' button and a blue square button. At the bottom right of the page is the Domino logo.

# Launch an Interactive Workspace

Ideation

Overview

Activity

Reviews

RUN

Workspaces

Jobs

MATERIALS

Data

Files

PUBLISH

Scheduled Jobs

App

Model APIs

Launchers

Settings

Workspaces are now persistent development environments that you can start, stop, and restart. Each workspace has a persistent volume that houses your files, and your changes are kept from one workspace session to the next so that you can decide when to commit changes to version control.

Learn more at our docs

My Workspaces All Workspaces Deleted Utility + Create New Workspace

### Getting started with Workspaces

Workspace sessions are Domino hosted Interactive Development Environments (IDE) sessions where you can interact with notebooks like Jupyter and RStudio.

Learn more about Workspaces.

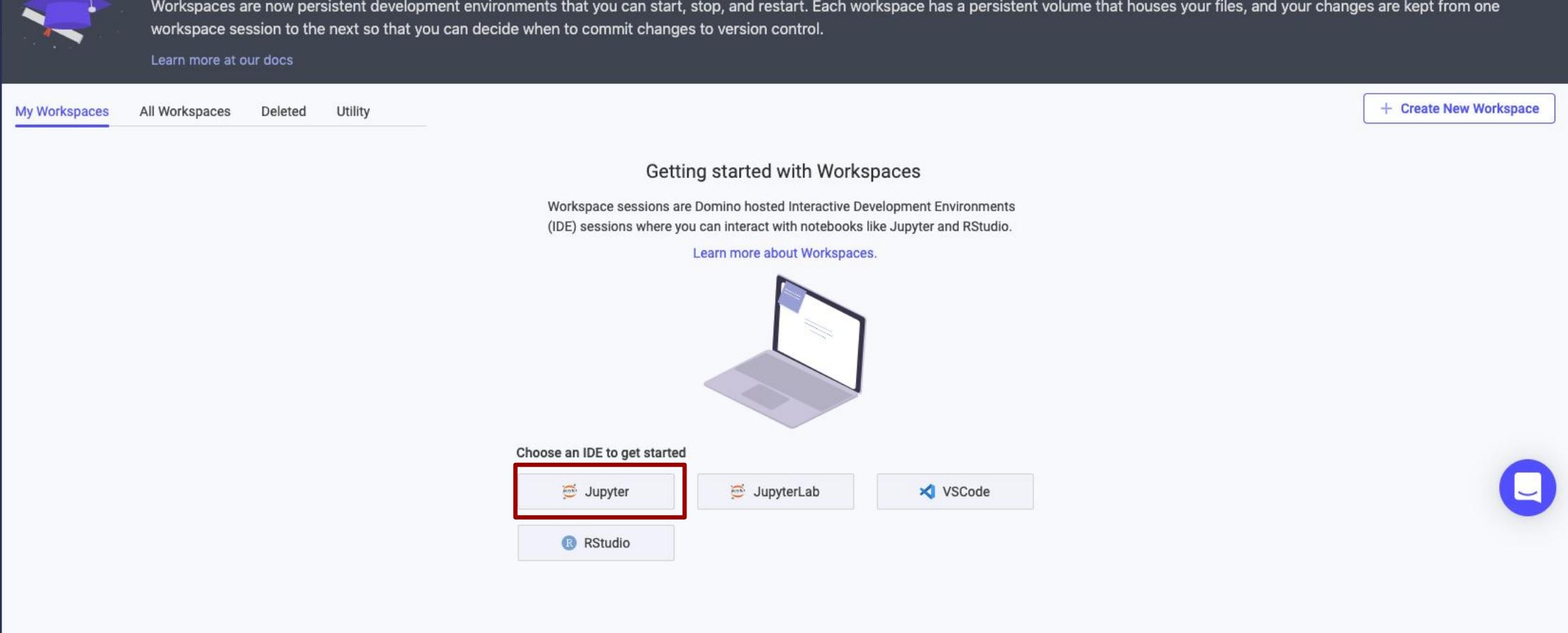
Choose an IDE to get started

Jupyter

JupyterLab

VSCode

RStudio



# Launch an Interactive Workspace

 **Launch New Workspace** X

1 **Environment & Hardware**  
Deep-Learning-... and GPU

2 **Data**

3 **Compute Cluster (optional)**

**Workspace Name**  
domino-andrea's Jupyter (Python, R, Julia) session

**Workspace Environment**  
Deep-Learning-Tutorial

**Workspace IDE**

Jupyter (selected)

JupyterLab

VSCode

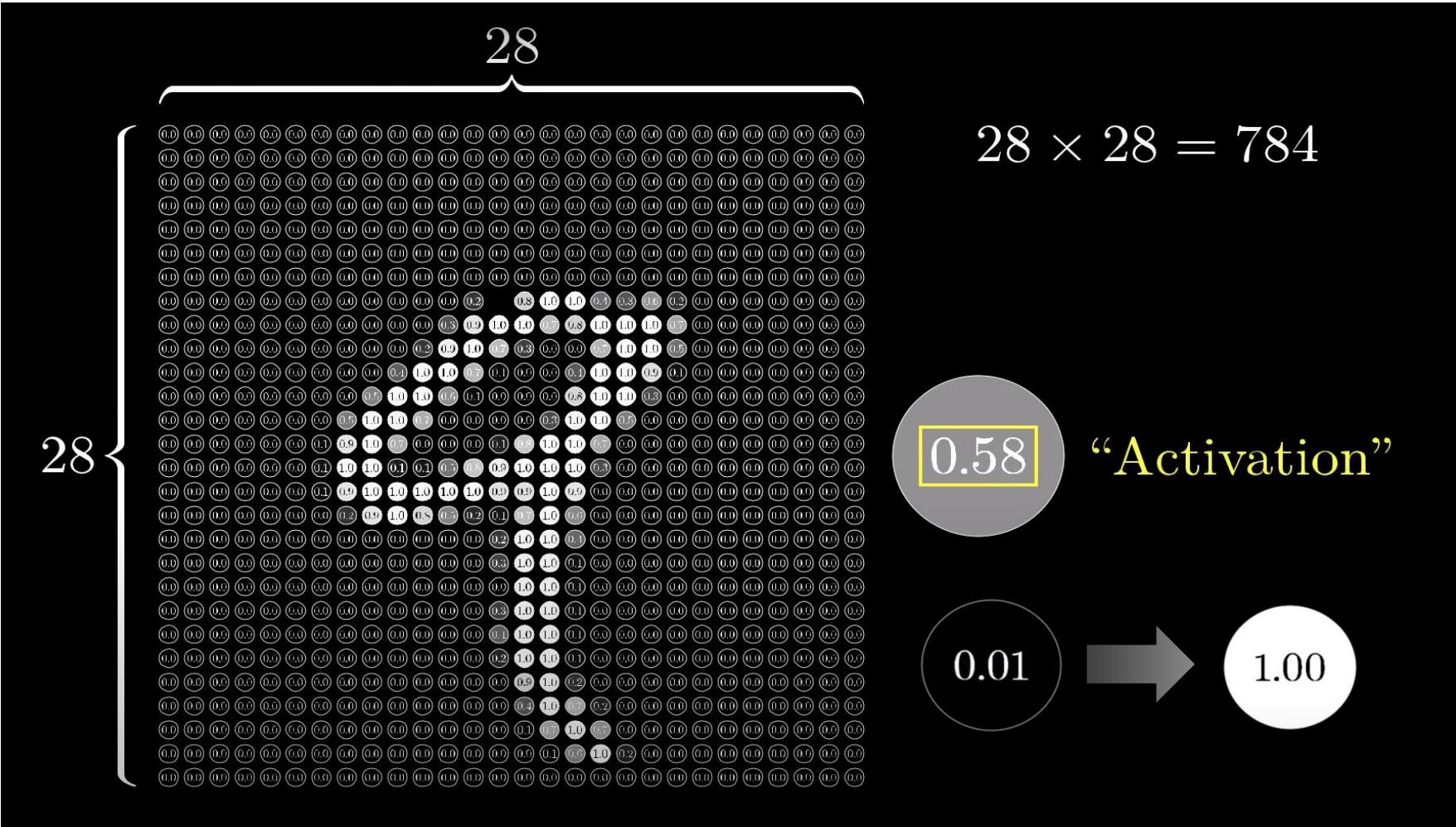
RStudio

**Hardware Tier**  
GPU  
6 cores · 45 GiB RAM · 1 GPU · \$0.0005/min < 1 MIN

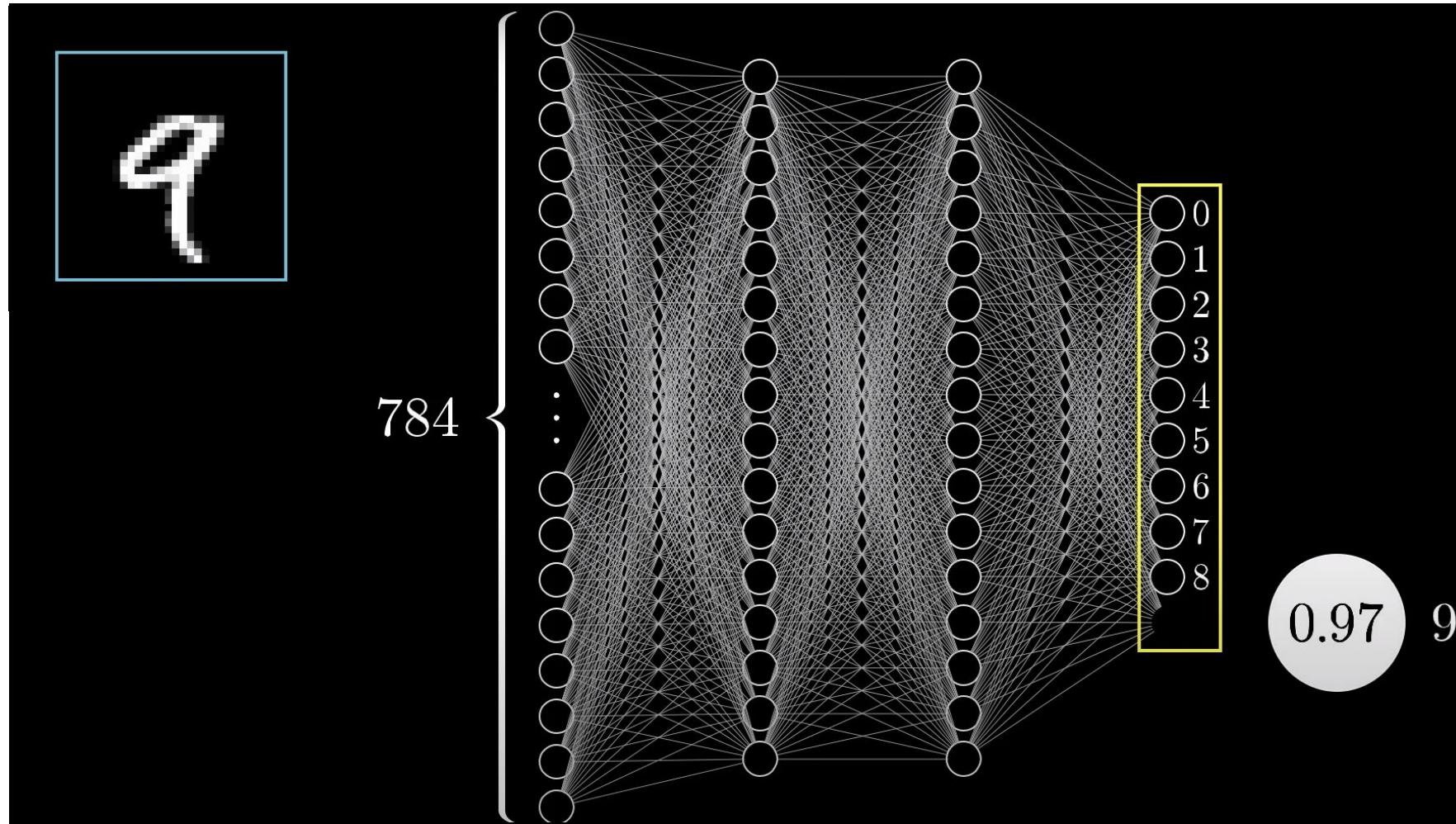
Cancel Next Launch Now

# Intro to Neural Networks

# WHAT IS A NEURAL NETWORK?

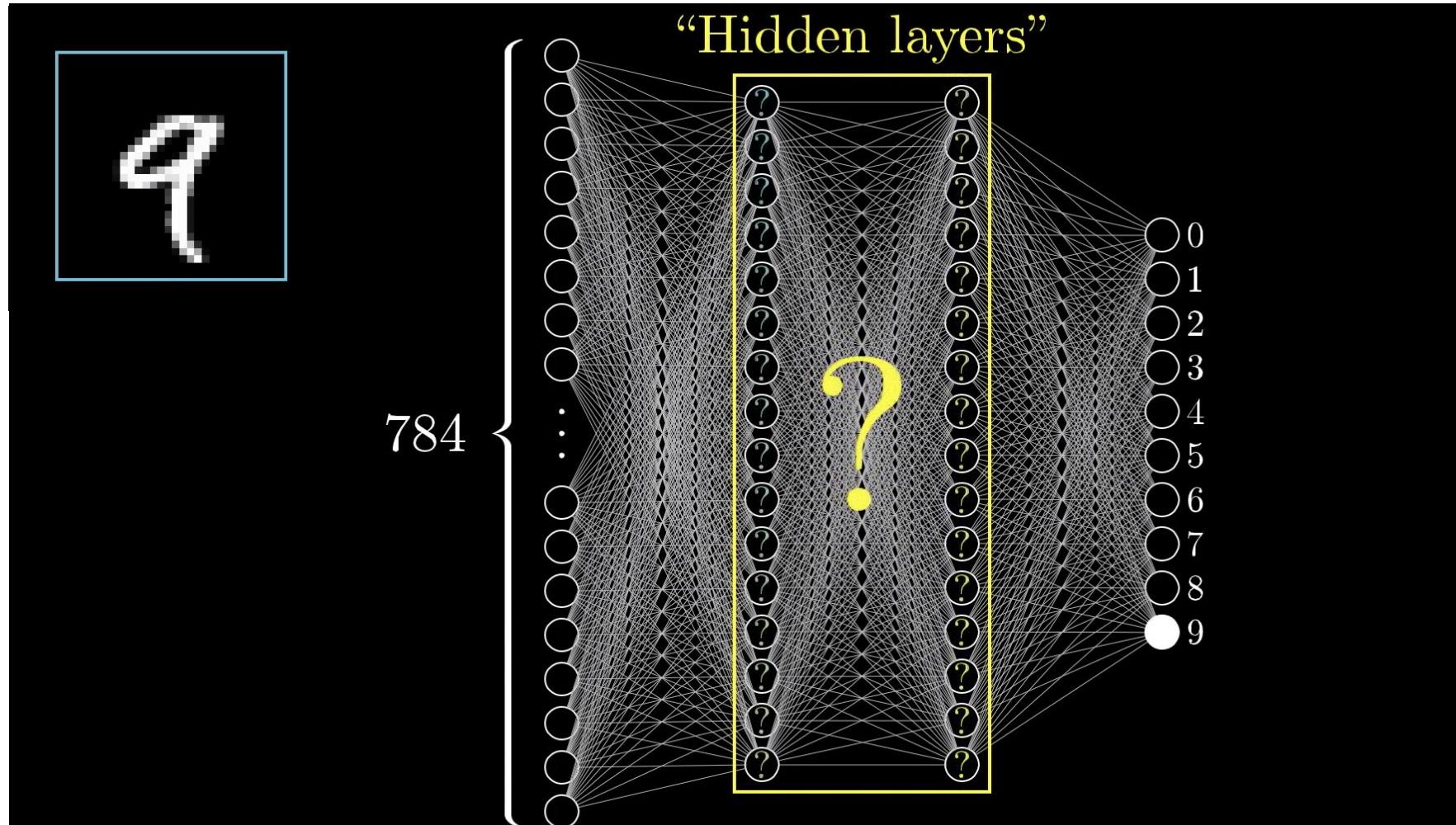


# WHAT IS A NEURAL NETWORK?

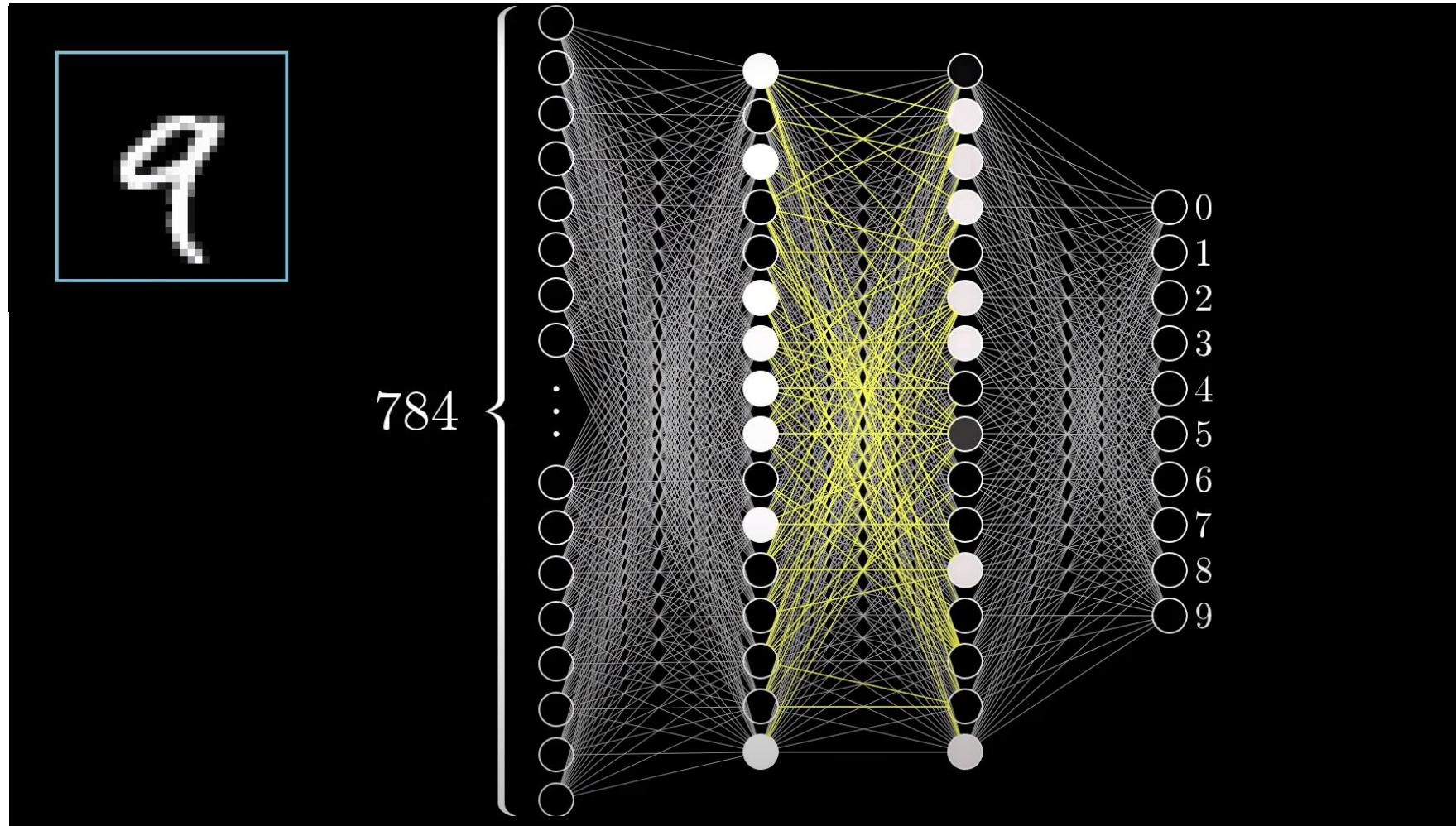


<https://www.3blue1brown.com/>

# WHAT IS A NEURAL NETWORK?



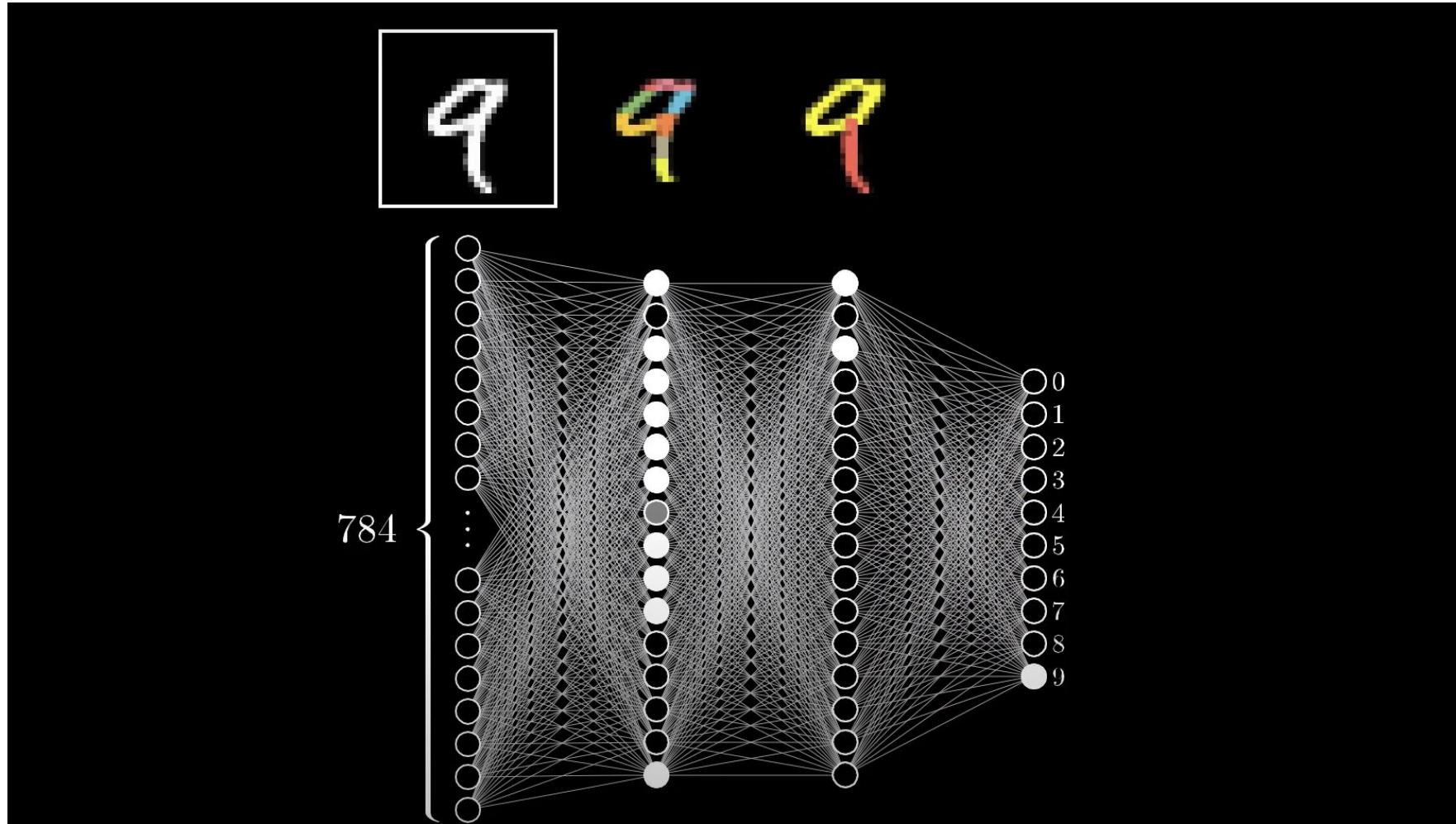
# WHAT IS A NEURAL NETWORK?



# WHAT IS A NEURAL NETWORK?

- The middle layers *ideally* represent subcomponents of the input
- In this example:
  - 2nd hidden layer = lines or loops that make up a digit
  - 1st hidden layer = edges that make up subcomponents of digits
- The final layer can then link to the combination of subcomponents that equal each digit

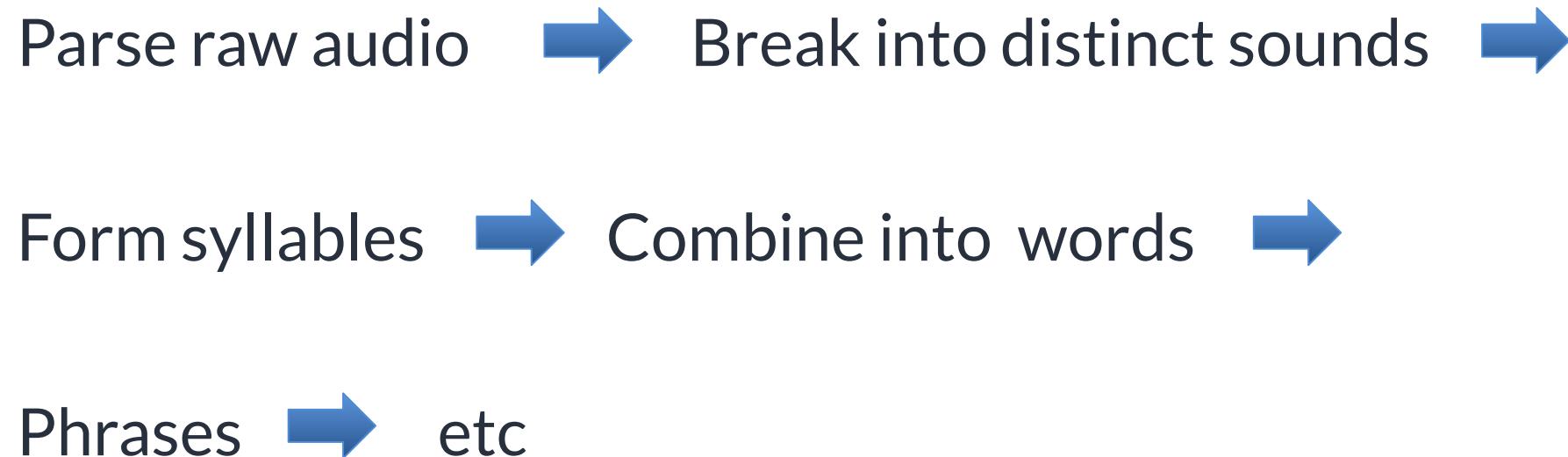
# WHAT IS A NEURAL NETWORK?



# IMAGE RECOGNITION

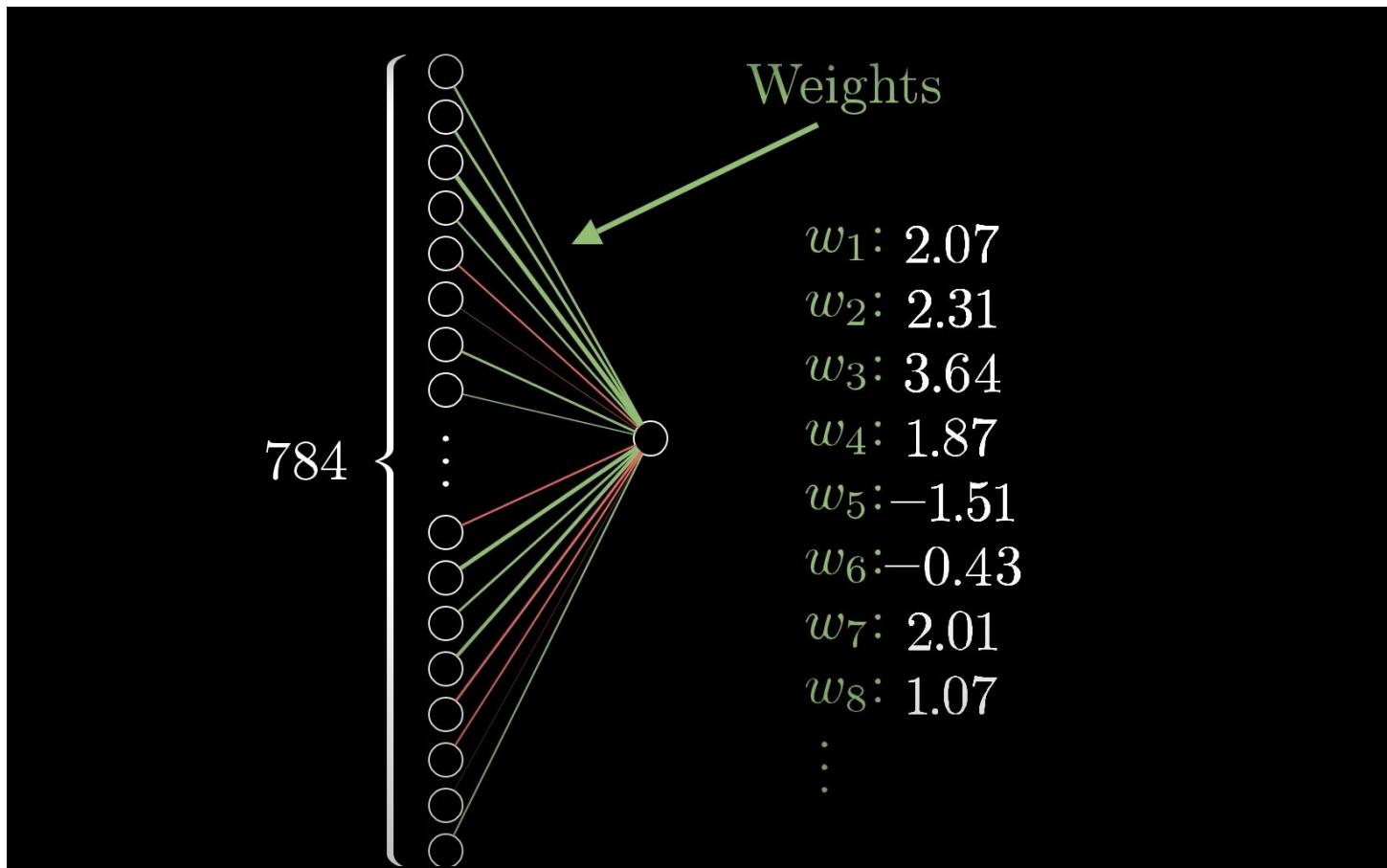


# SPEECH RECOGNITION

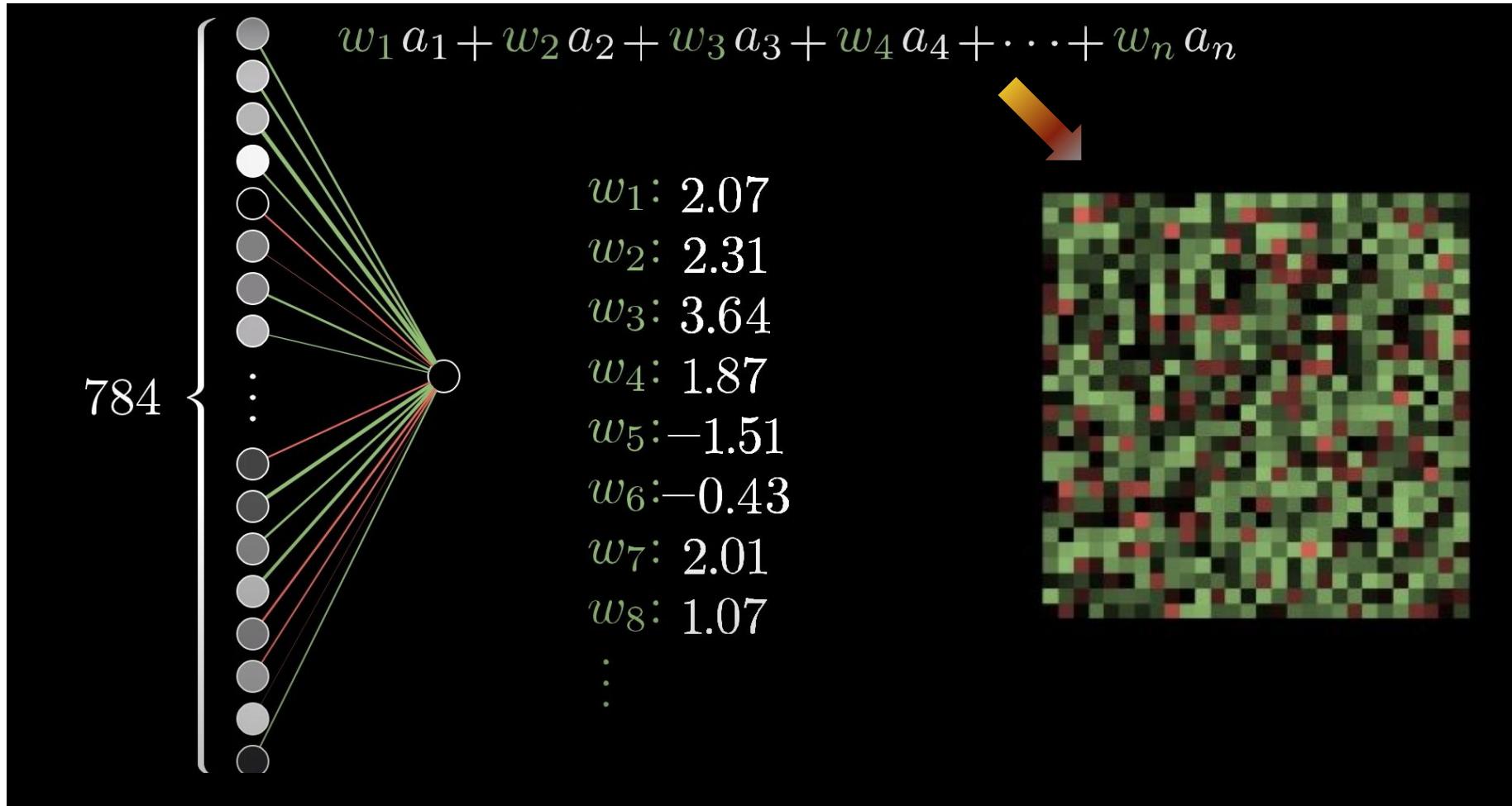


# NEURAL NETWORK WEIGHTS

How can we ensure the hidden layers are capturing the desired patterns?



# NEURAL NETWORK WEIGHTS

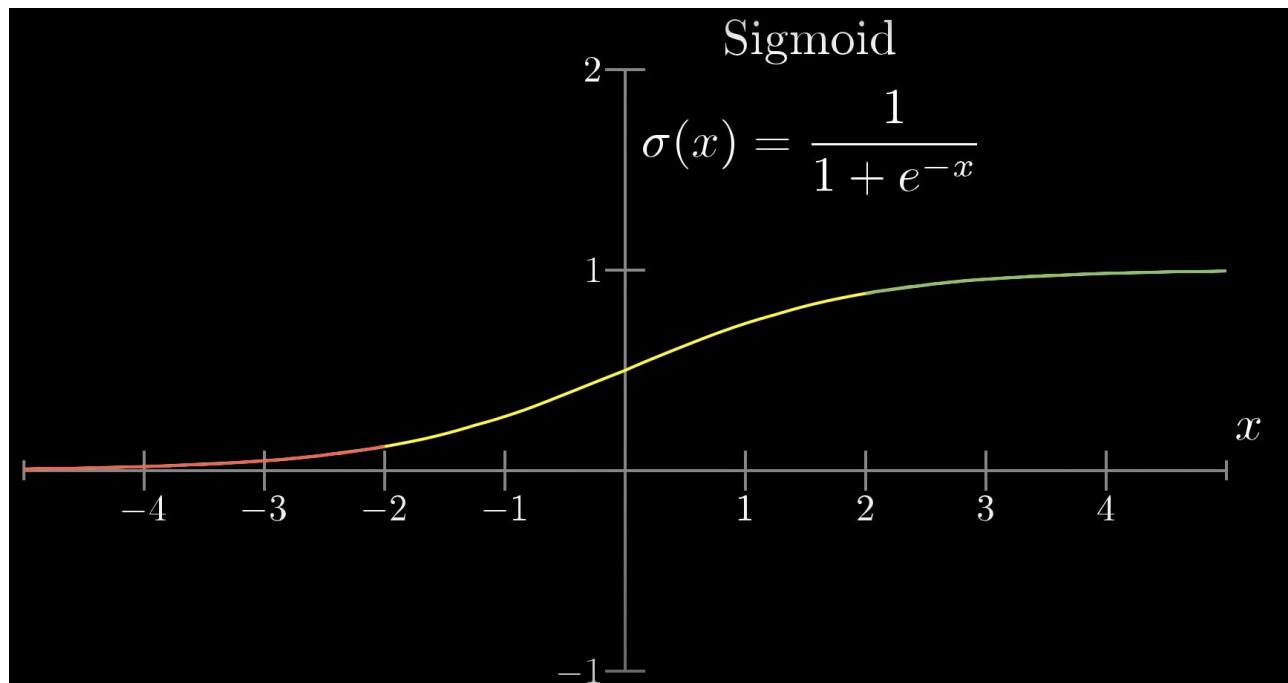


# NEURAL NETWORK WEIGHTS

Each of those sums could be any number

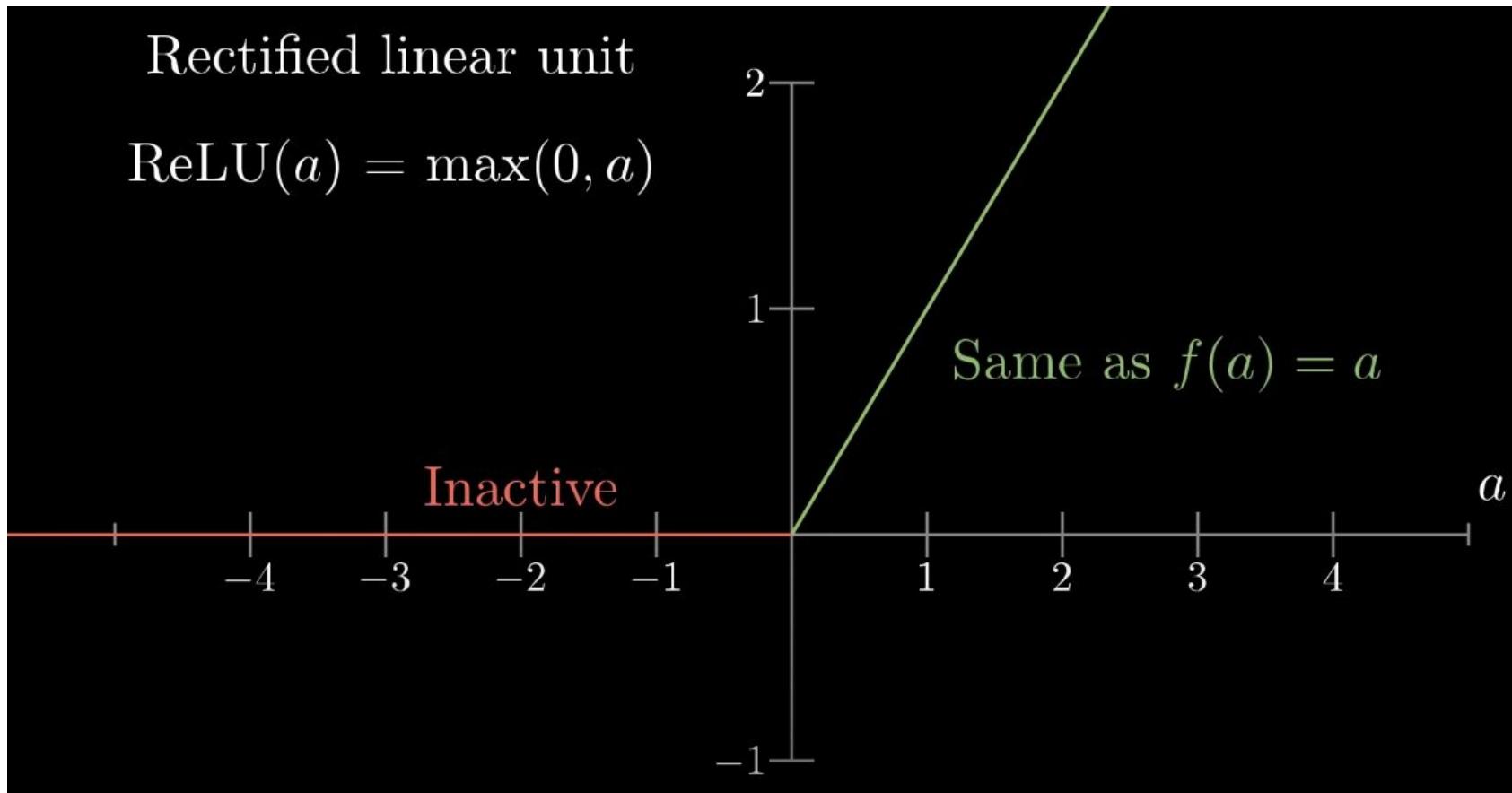
To ensure we get an activation between 0 and 1 for the next layer we put those sums through a function

This is a sigmoid function, where very negative numbers are close to zero and very positive close to 1



<https://www.3blue1brown.com/>

# NEURAL NETWORK WEIGHTS



# NEURAL NETWORK BIAS

We don't necessarily want a neuron to light up when the activation sum is greater than 0, instead we may prefer a different threshold

Sigmoid

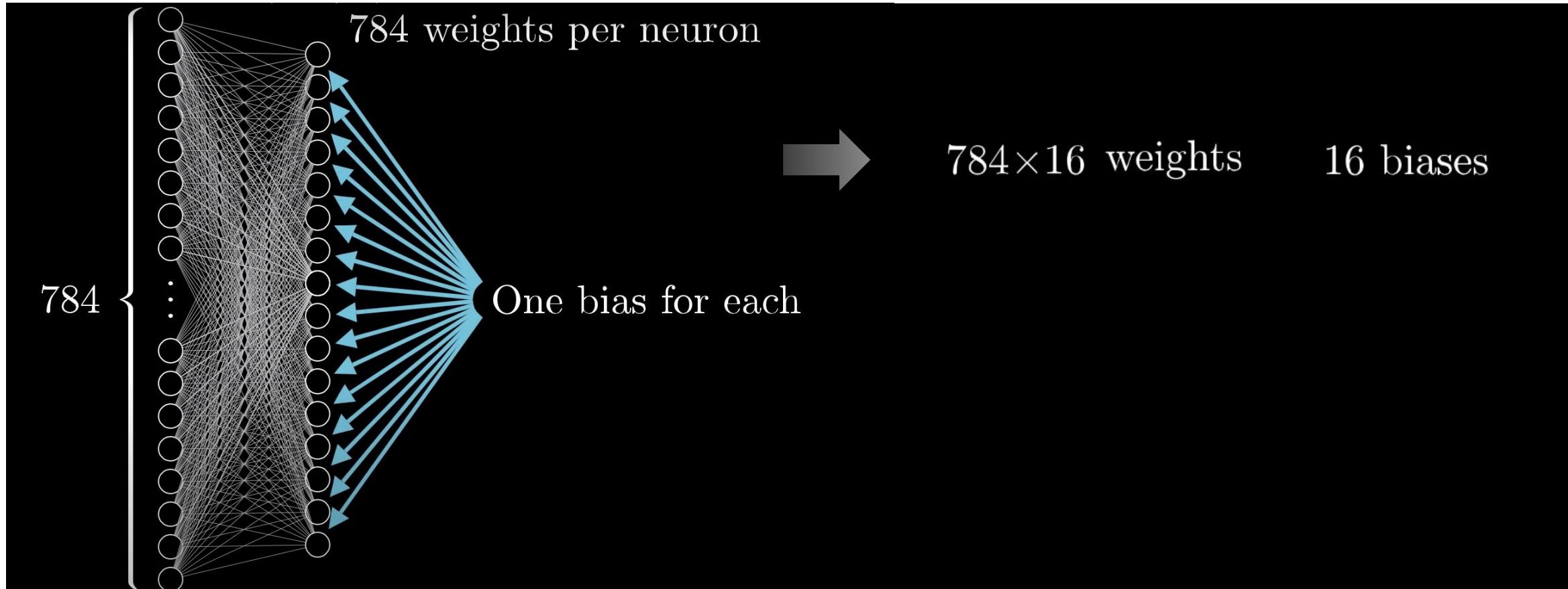


How positive is this?

$$\sigma(w_1a_1 + w_2a_2 + w_3a_3 + \dots + w_na_n - 10)$$

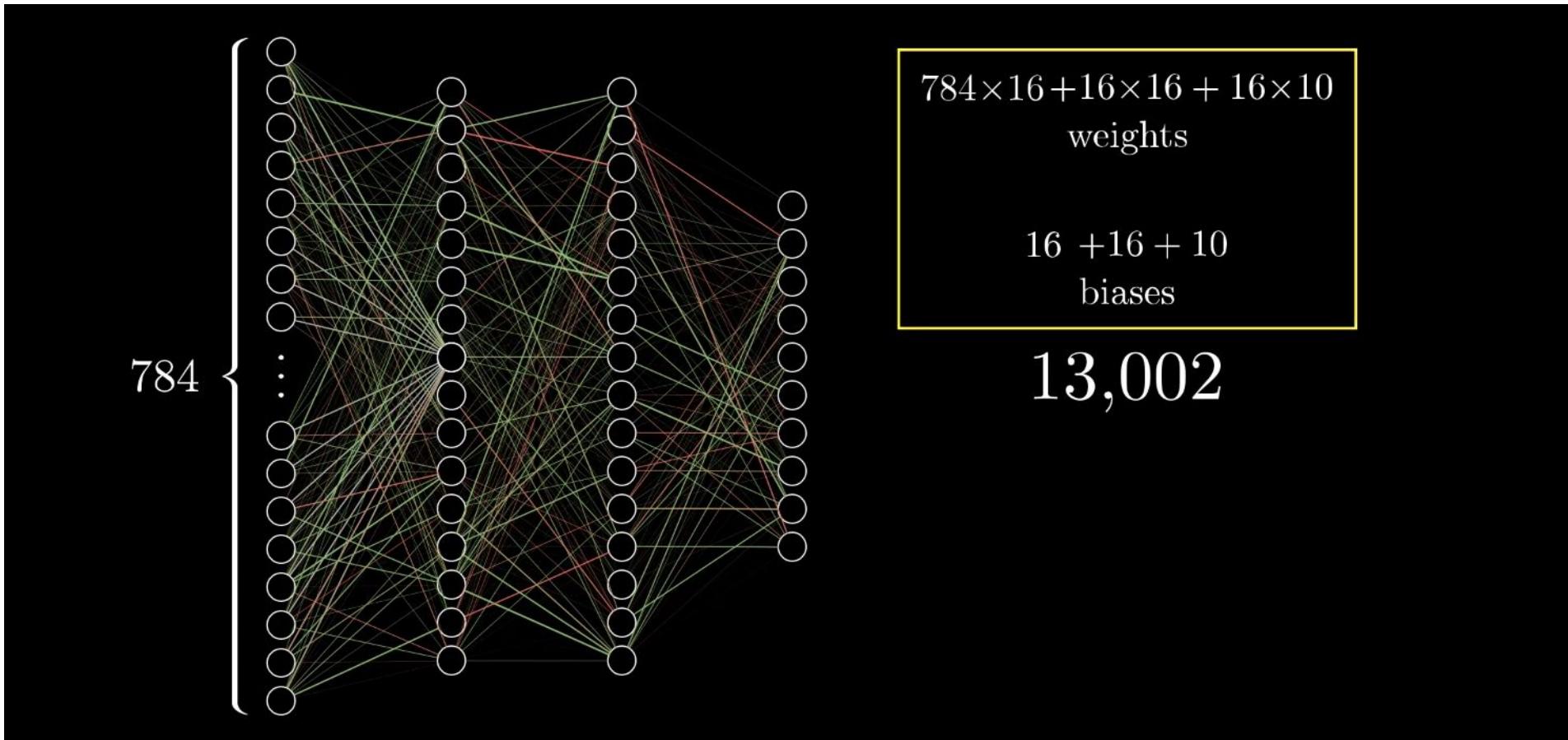
“bias”

# NEURAL NETWORK LEARNING



<https://www.3blue1brown.com/>

# NEURAL NETWORK LEARNING



<https://www.3blue1brown.com/>

# NEURAL NETWORK LEARNING

$$a_0^{(1)} = \sigma \left( w_{0,0} a_0^{(0)} + w_{0,1} a_1^{(0)} + \cdots + w_{0,n} a_n^{(0)} + b_0 \right)$$

$$\sigma \left( \begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \right)$$

$$\mathbf{a}^{(1)} = \sigma(\mathbf{W}\mathbf{a}^{(0)} + \mathbf{b})$$

# NEURAL NETWORK LEARNING

- In this example, each neuron is connected to all the neurons in the previous layer:
  - The *weight* describes the strength of each of those connections
  - The *bias* describes whether the neuron tends to be active or not
- These form a function for each neuron which is transformed into a value between 0 and 1
  - This indicates the activation of the next neuron

# HOW DOES THIS NETWORK LEARN THE WEIGHTS AND BIASES?

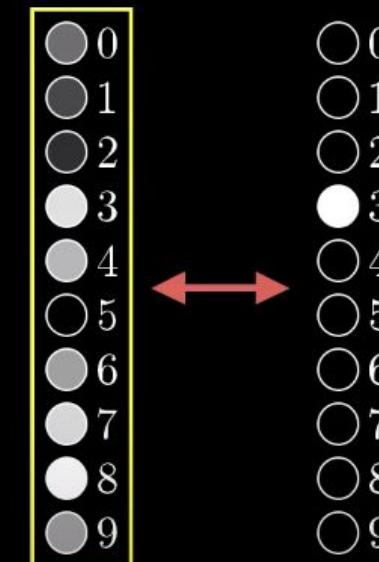
- The *weights* and *biases* are initialized randomly
- The (initially wrong) final output layer is compared to the correct answer to generate a **cost function**
  - The cost function shows the “cost” of the difference - this is the squares of the differences between the output activations and the actual value
- Average cost over all the training examples is used to evaluate how the network performed

# HOW DOES THIS NETWORK LEARN THE WEIGHTS AND BIASES?

Cost of 

$$3.37 \left\{ \begin{array}{l} 0.1863 \leftarrow (0.43 - 0.00)^2 + \\ 0.0809 \leftarrow (0.28 - 0.00)^2 + \\ 0.0357 \leftarrow (0.19 - 0.00)^2 + \\ 0.0138 \leftarrow (0.88 - 1.00)^2 + \\ 0.5242 \leftarrow (0.72 - 0.00)^2 + \\ 0.0001 \leftarrow (0.01 - 0.00)^2 + \\ 0.4079 \leftarrow (0.64 - 0.00)^2 + \\ 0.7388 \leftarrow (0.86 - 0.00)^2 + \\ 0.9817 \leftarrow (0.99 - 0.00)^2 + \\ 0.3998 \leftarrow (0.63 - 0.00)^2 \end{array} \right.$$

What's the “cost”  
of this difference?



# COST FUNCTION

- Takes in all weights and biases and returns a single number measuring the network performance
- The cost function needs to be minimized - this is done through gradient descent
- The negative gradient of the cost function tells us what changes to the weights and biases will induce the fastest change to the value of the cost function
- Therefore, the network ‘learning’ is just minimizing a cost function

# HOW DOES THIS NETWORK LEARN THE WEIGHTS AND BIASES?

$$\vec{\mathbf{W}} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{13,000} \\ w_{13,001} \\ w_{13,002} \end{bmatrix}$$

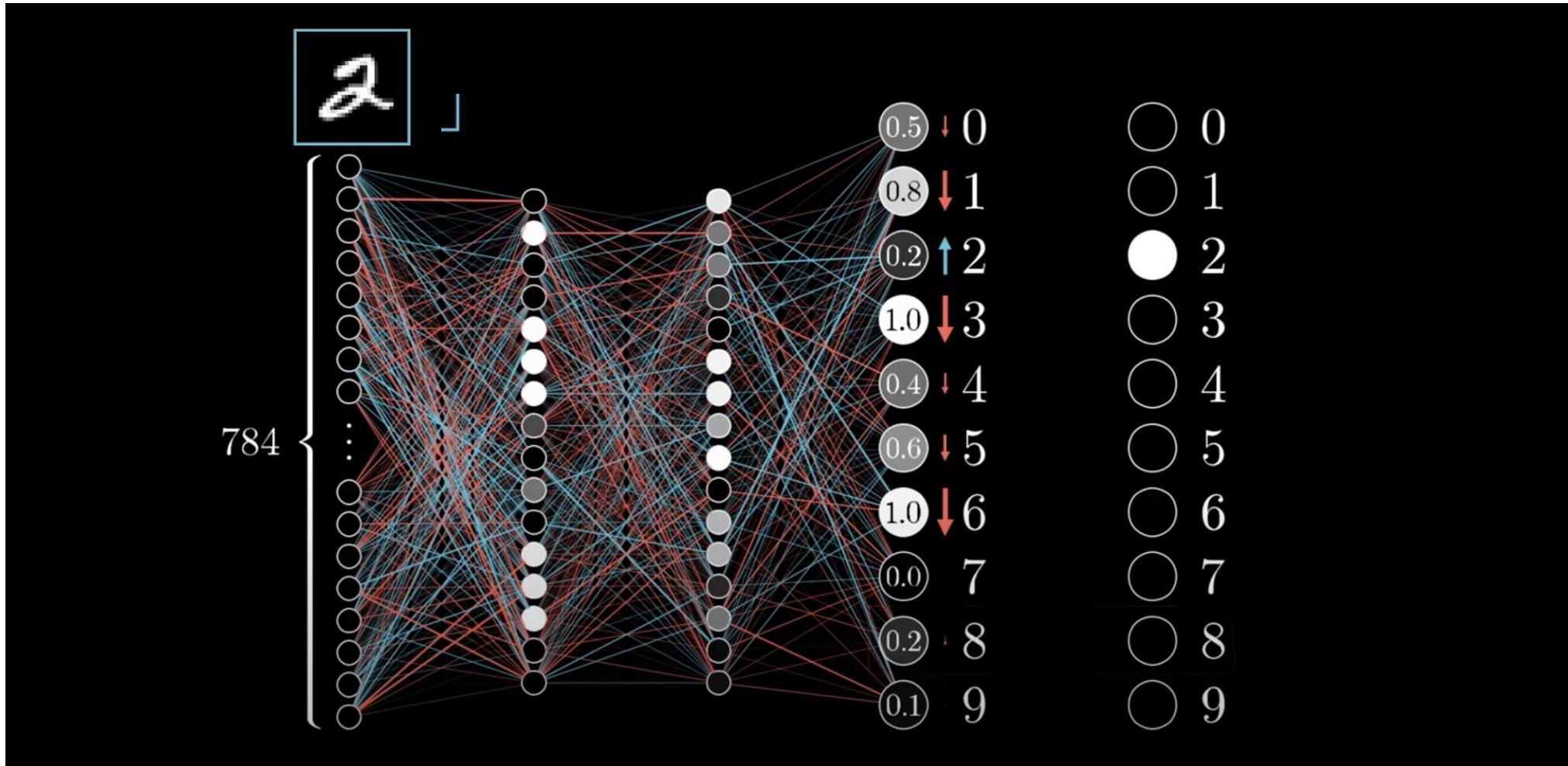
$$-\nabla C(\vec{\mathbf{W}}) = \begin{bmatrix} 0.31 \\ 0.03 \\ -1.25 \\ \vdots \\ 0.78 \\ -0.37 \\ 0.16 \end{bmatrix}$$

$w_0$  should increase somewhat  
 $w_1$  should increase a little  
 $w_2$  should decrease a lot  
 $w_{13,000}$  should increase a lot  
 $w_{13,001}$  should decrease somewhat  
 $w_{13,002}$  should increase a little

# BACKPROPAGATION

- “Backward propagation of errors”
- Algorithm for calculating the gradient of the error function, done so from the last layer to the first layer
- Essentially determines how a single training example would like to alter the weights and biases, based on the cost function, including what relative proportions of those changes cause the most rapid decrease to the cost

# BACKPROPAGATION



# BACKPROPAGATION

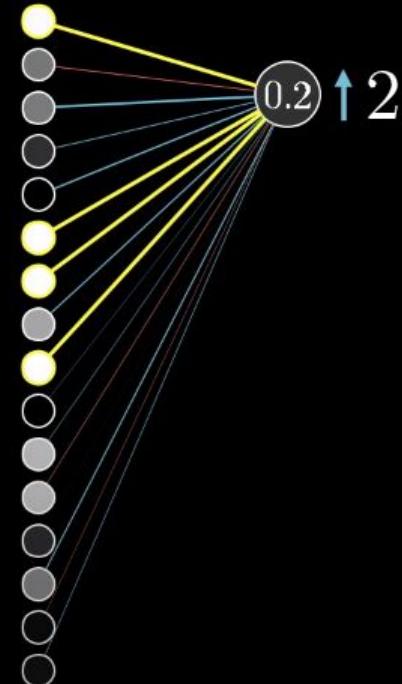


$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \dots + w_{n-1} a_{n-1} + b)$$

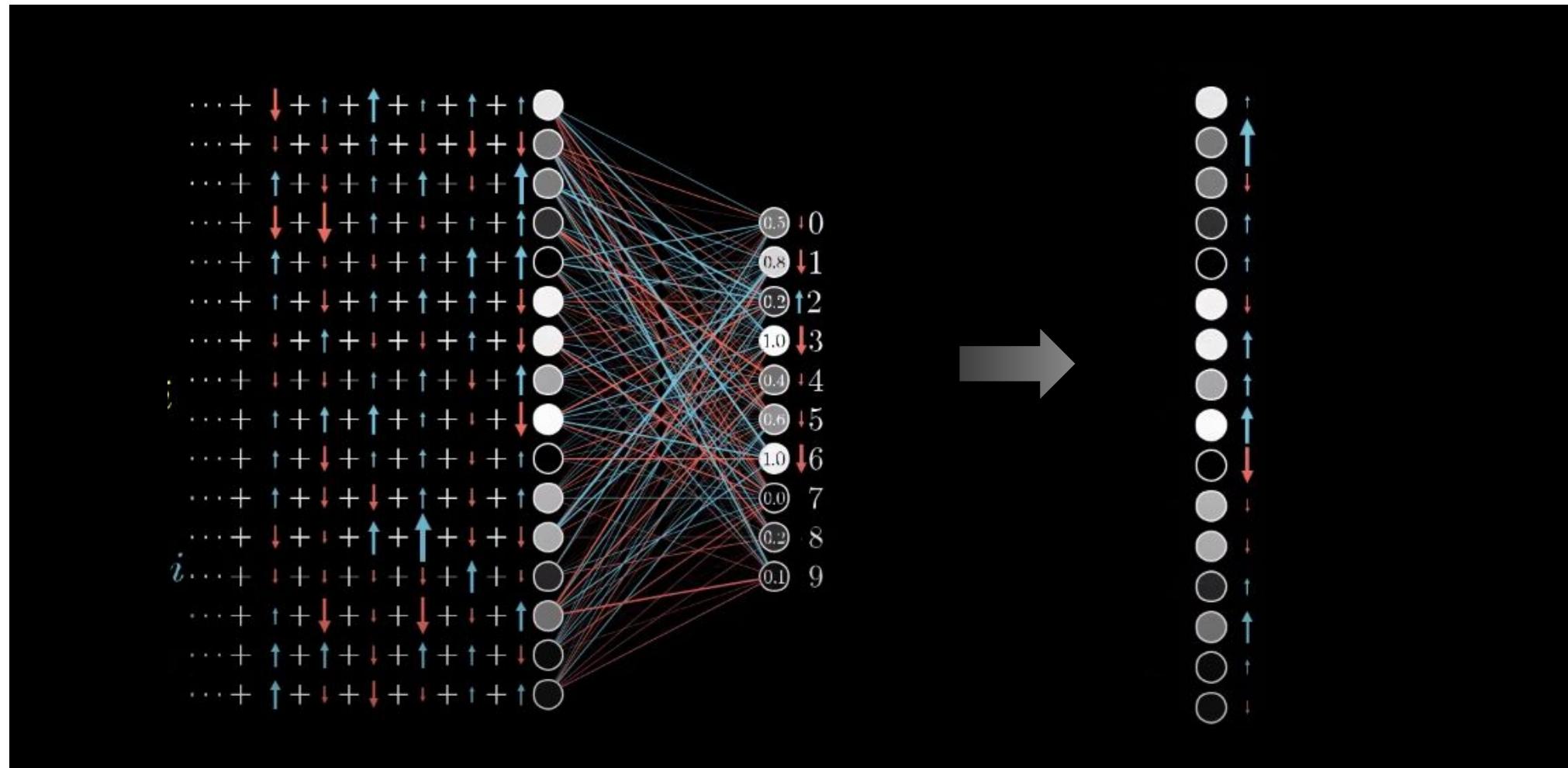
Increase  $b$

Increase  $w_i$   
in proportion to  $a_i$

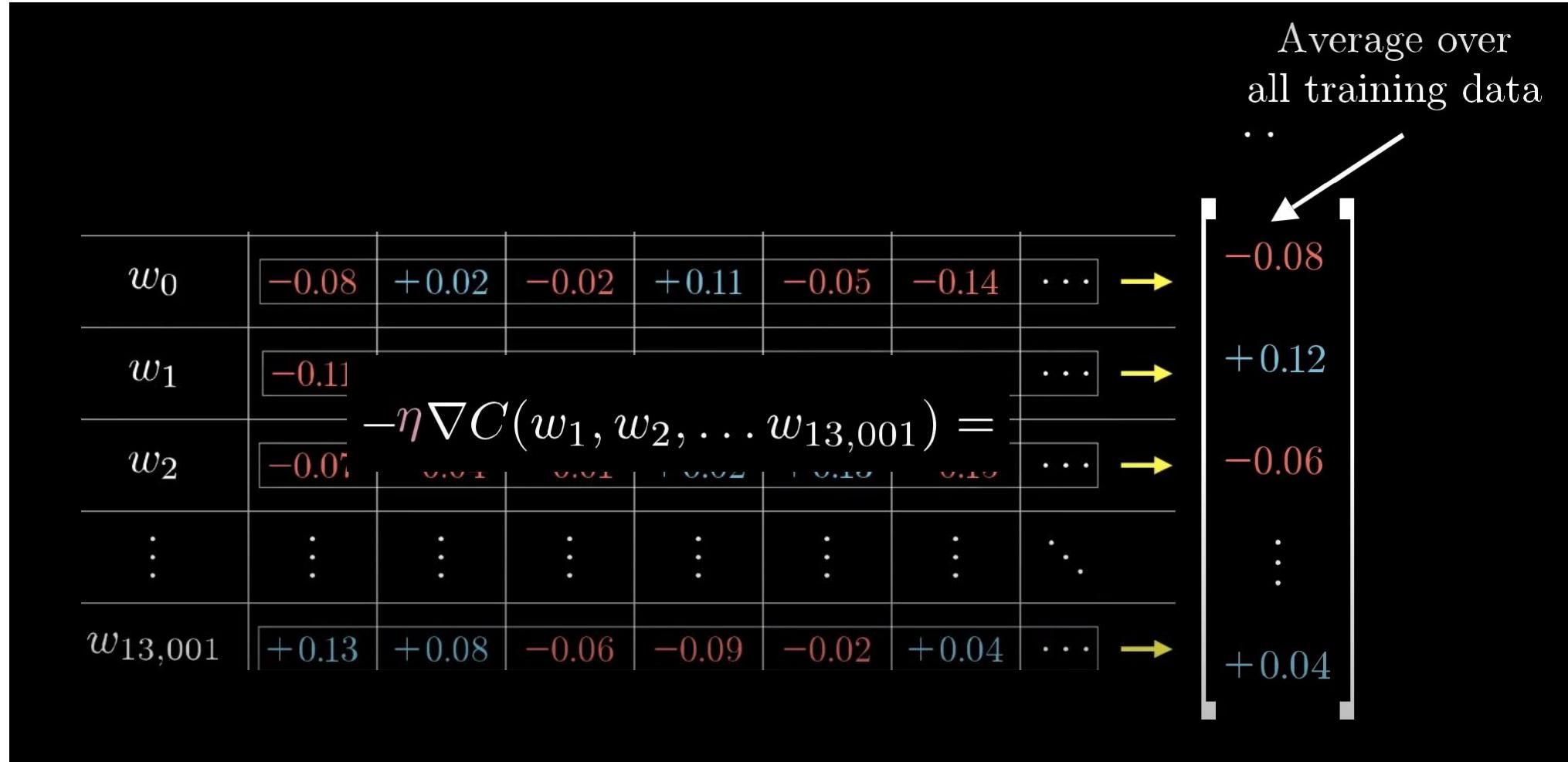
Change  $a_i$



# BACKPROPAGATION



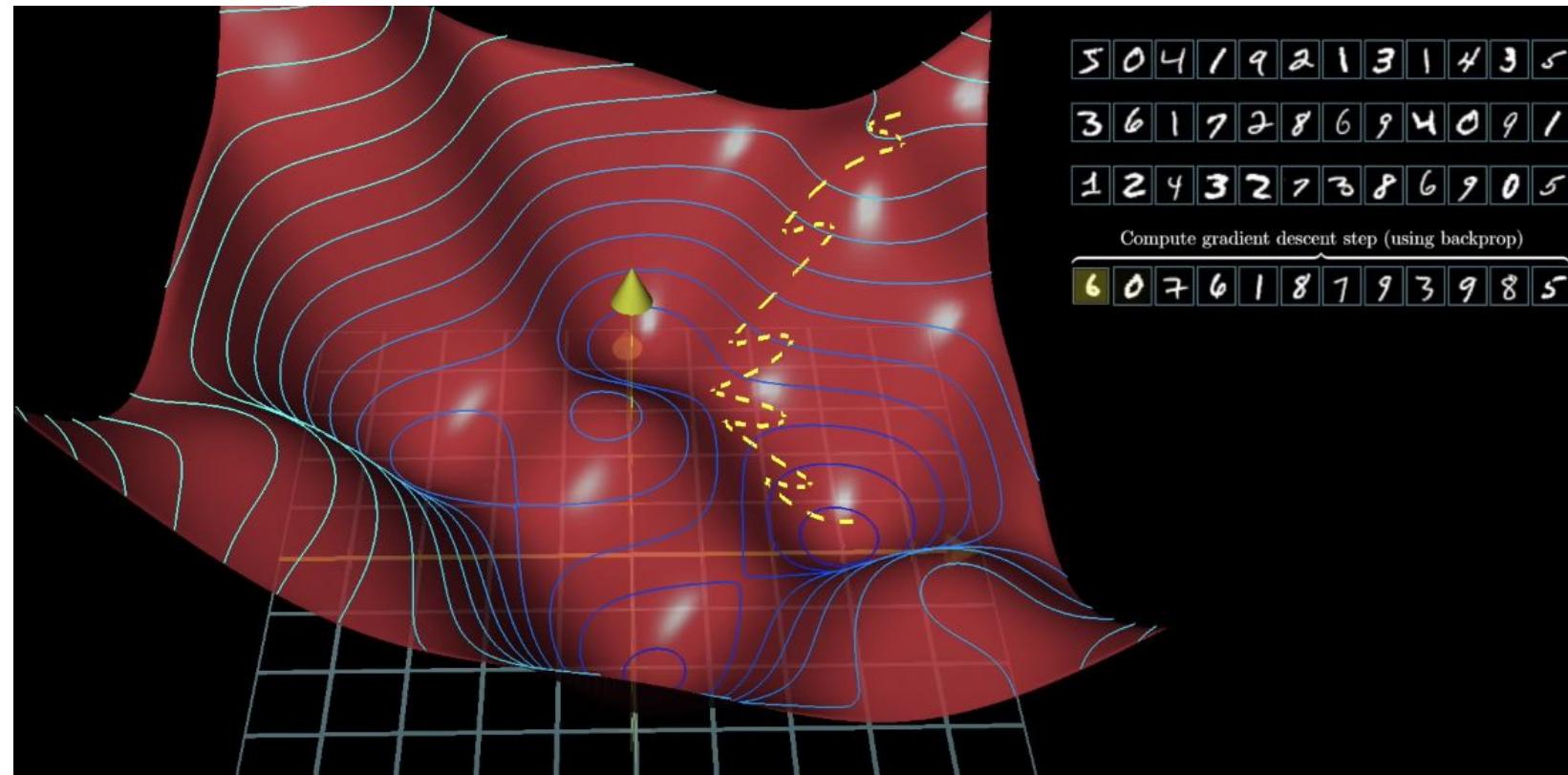
# GRADIENT DESCENT



# STOCHASTIC GRADIENT DESCENT

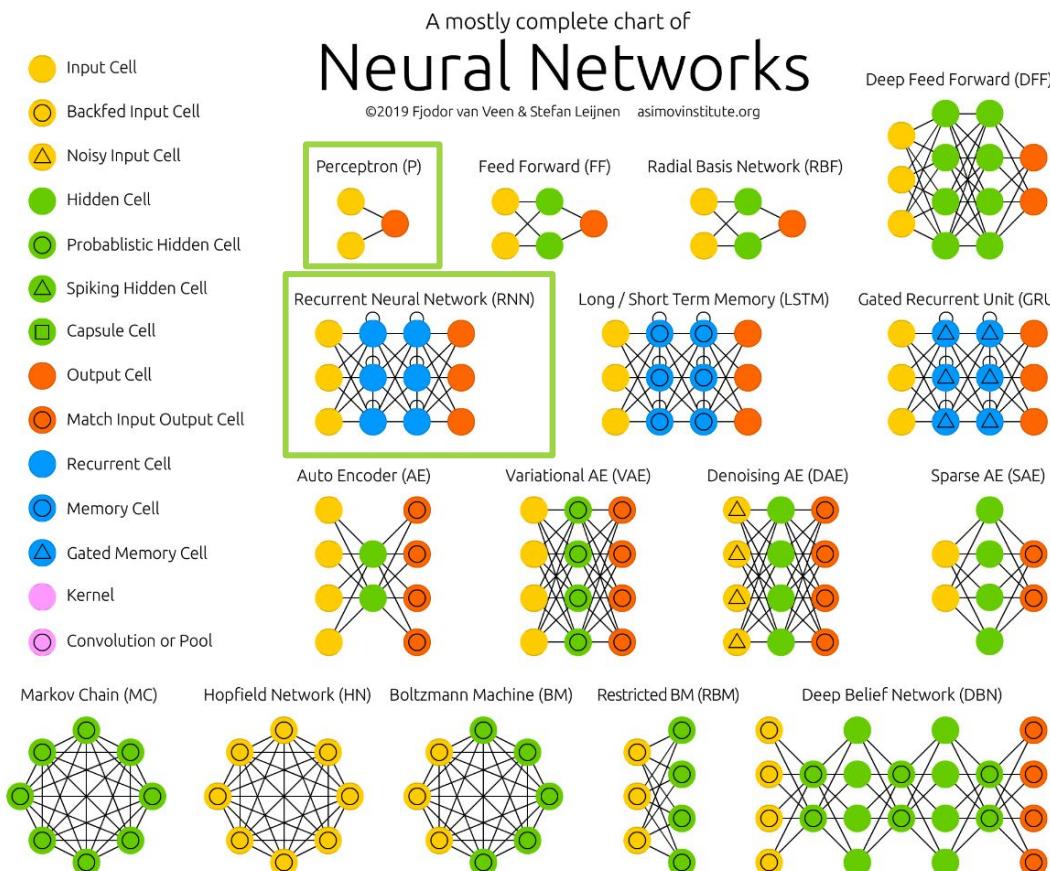
Much faster to calculate

1. Divides training data into mini-batches
2. Compute gradient descent step
3. Uses the approximation instead of the actual gradient

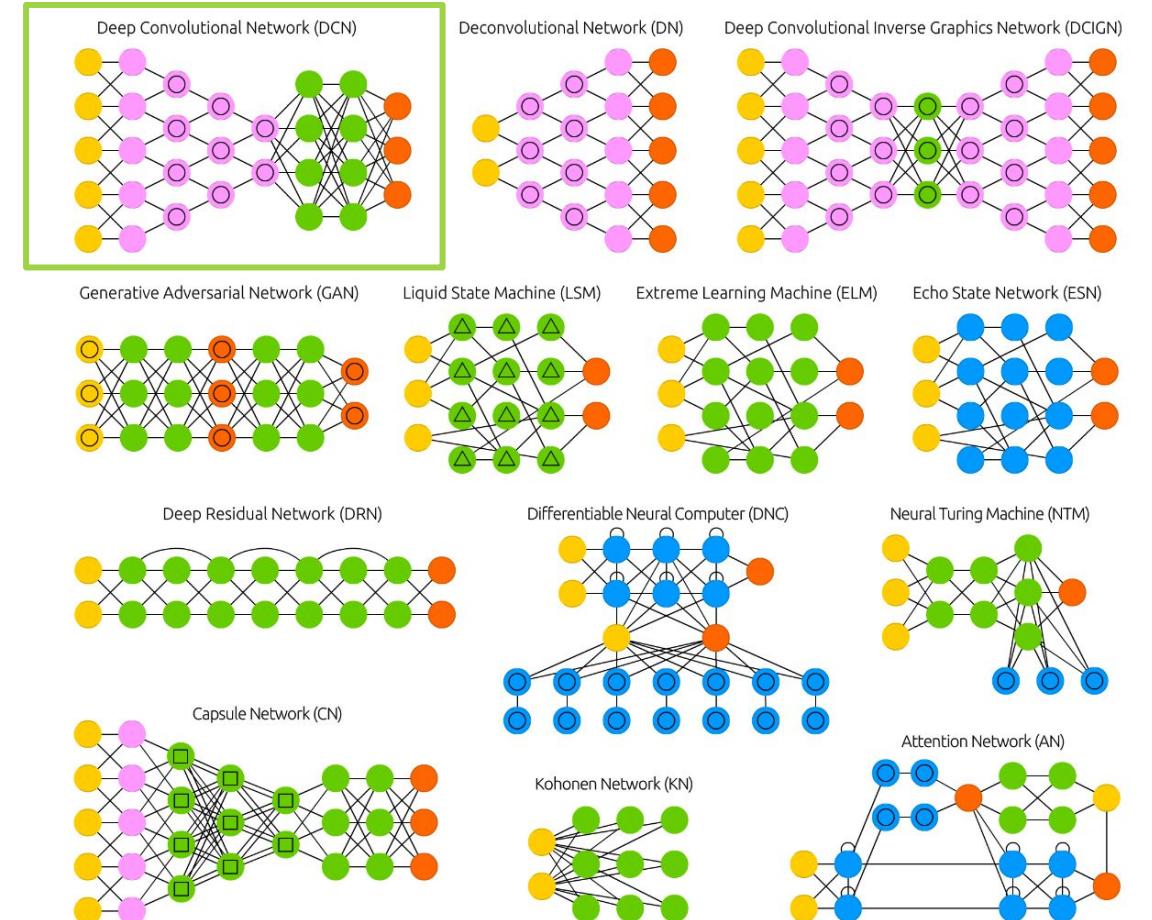


<https://www.3blue1brown.com/>

# TYPES OF NETWORKS



<https://www.asimovinstitute.org/neural-network-zoo/>



# MULTILAYER PERCEPTRON

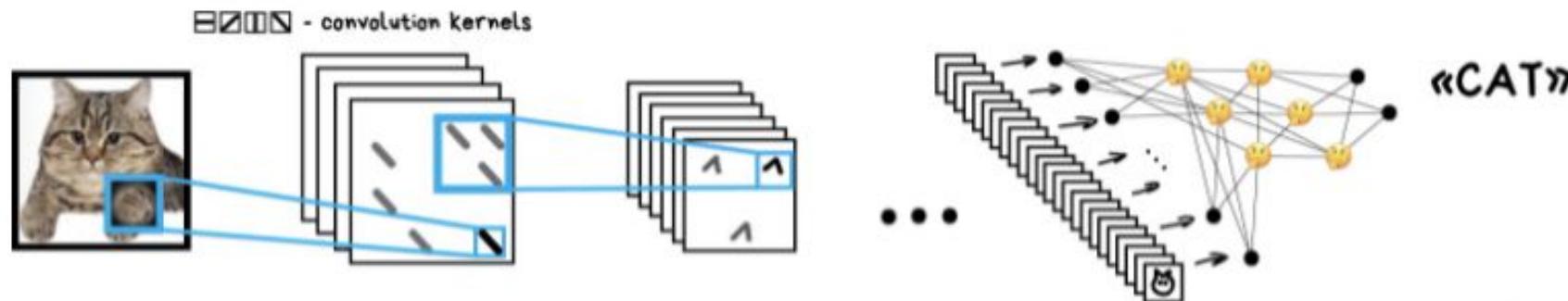
- One perceptron for each input
  - number of parameters rapidly becomes unmanageable
- Reacts differently to shifted versions of an input
  - i.e. the image needs to be in the same place each time
- Spatial information is lost - does not take into account pixel position and correlation with neighbors

# CONVOLUTIONAL NEURAL NETWORKS

Most commonly used for pictures and videos in tasks such as identifications, style transfer, enhancing images, slo-mo effects, etc

Solved the problem of hand-crafting features

Assembles patterns of increasing complexity using smaller and simpler patterns



# RECURRENT NEURAL NETWORKS

Best for sequential data like music or text, used for speech recognition, voice synthesis

Idea is to add memory to each neuron - so they get not only info from the previous layer but also from themselves in the previous pass

Neuron can reset when memory is not longer needed, leaving the connection

# PyTorch Overview

# DEEP LEARNING IN PYTHON

- Theano
- Keras
- TensorFlow
- PyTorch
- MXNet, Lasagne

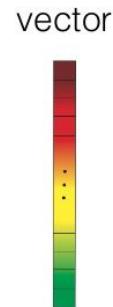
# PYTORCH OVERVIEW

- Easy debugging - can be used with many standard Python debugging tools
- Data parallelism - distribute work among CPU/GPU cores
- Many built in loss functions, optimizers, transformations, and easy to build custom data loaders/transformers
- Large community, used in many research papers and online courses

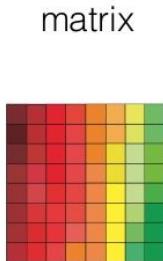
# TENSORS

- Data structure similar to arrays and matrices
- Used to encode the input (i.e. pixel info) and output (classes)
- Share underlying memory with numpy arrays but can run on GPUs

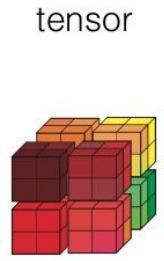
tensor = multidimensional array



$$\mathbf{v} \in \mathbb{R}^{64}$$

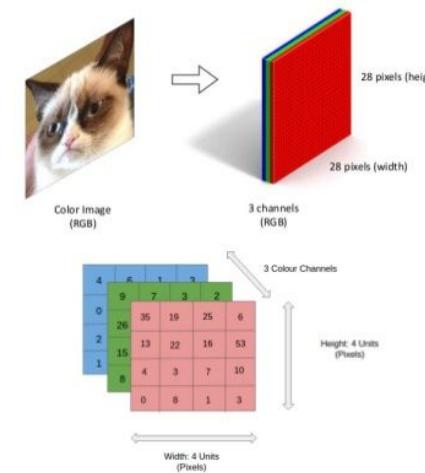


$$X \in \mathbb{R}^{8 \times 8}$$



$$\mathcal{X} \in \mathbb{R}^{4 \times 4 \times 4}$$

color image is 3rd-order tensor



# DATALOADERS AND TRANSFORMS

- Dataloaders let us pass samples in “minibatches”, reshuffle the data at every epoch to help with generalization, speed up retrieval, and set the number of workers to take full advantage of all of the GPUs
- Transforms - modify data or labels
  - Avoid overfitting and increase the number of training data
  - Crop, resize, normalize color channels, change to grayscale, etc
  - `torchvision.transforms` - offers several commonly used transforms

# ACTIVITY: EXPLORE TRANSFORMS

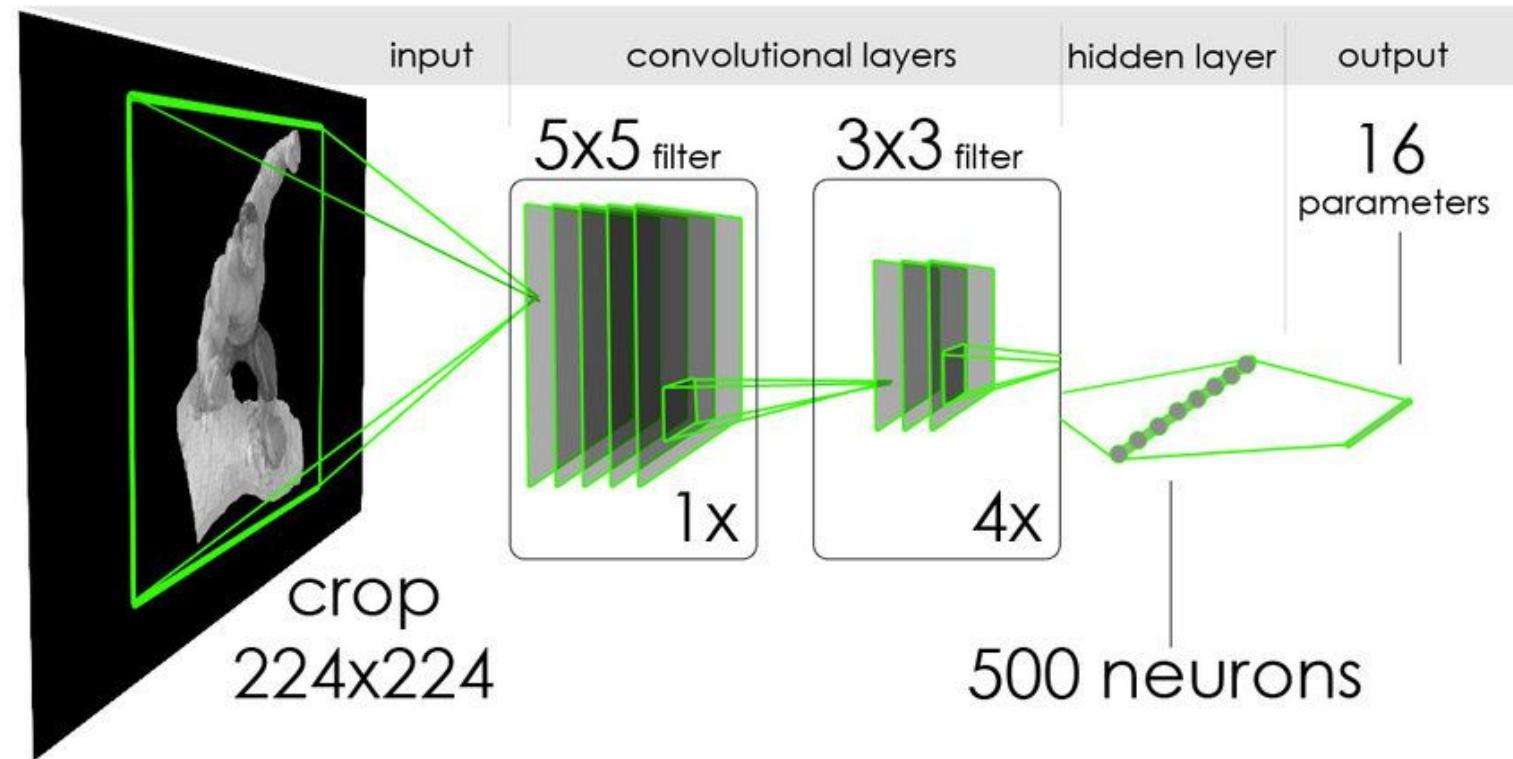
1. Open notebook ‘1-Load-Data-PyTorch.ipynb’
2. Try different transforms to see how they change the images

# Building the Model

# INPUT AND OUTPUT LAYERS

Input layer - features of the image

Output layer - dataset labels



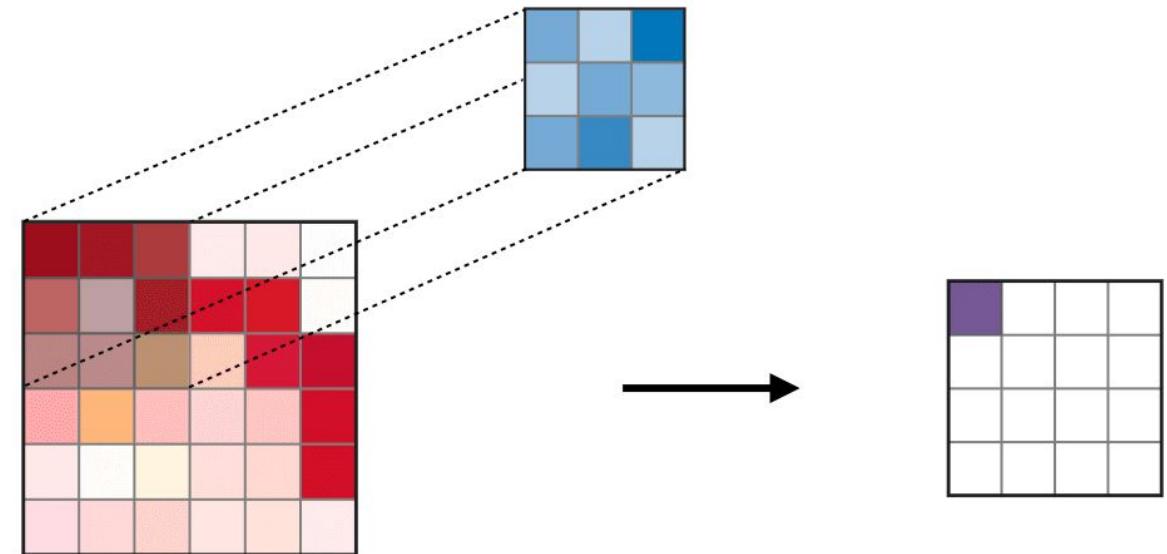
[https://www.researchgate.net/publication/321260305\\_Learning\\_Lightprobes\\_for\\_Mixed\\_Reality\\_Illumination](https://www.researchgate.net/publication/321260305_Learning_Lightprobes_for_Mixed_Reality_Illumination)

# CONVOLUTIONAL LAYERS

Filters scan across the input and calculate a value using a convolution operation

Each filter can be related to anything - for example one could locate tails

There are multiple filters which results in a 3D output



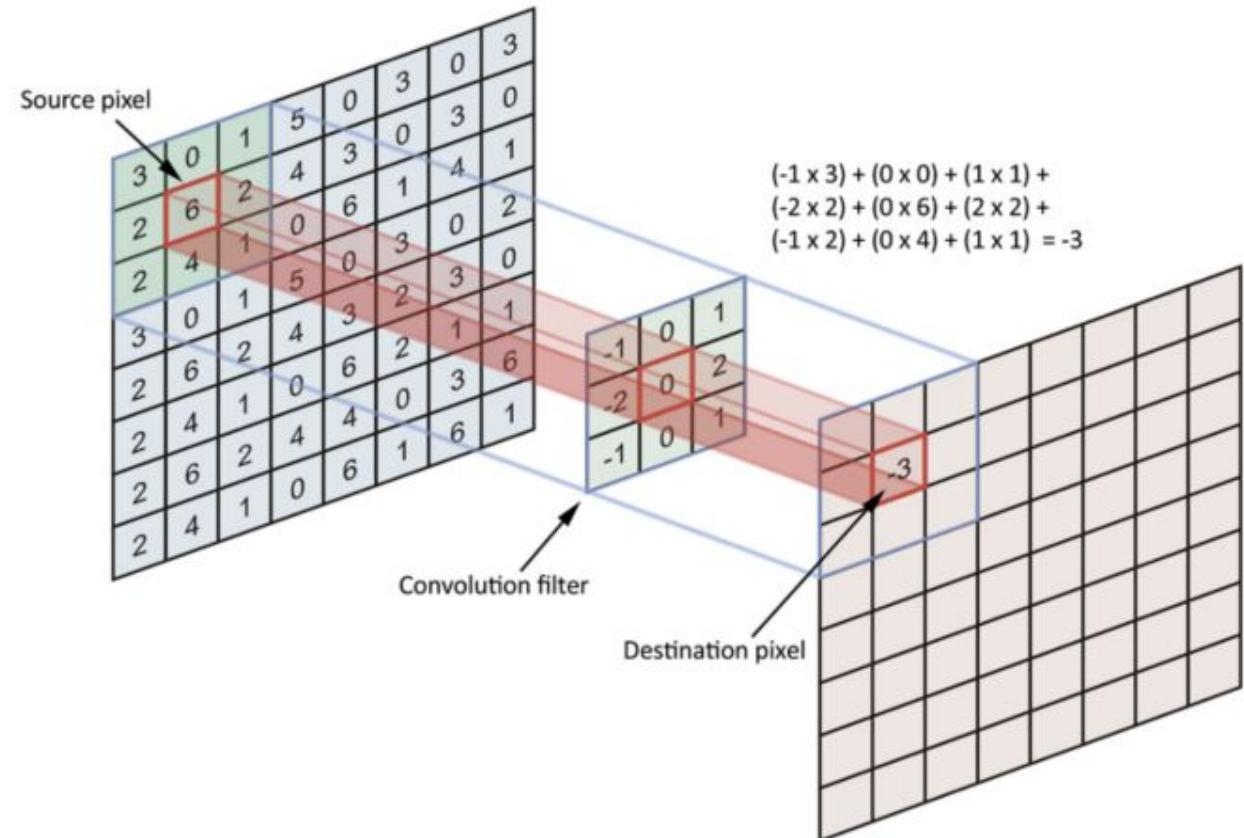
<https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks>

# CONVOLUTIONAL LAYERS

Each 2D slice of the filters is called a kernel

A feature map is generated for each kernel

These are passed through an activation function - determines if a given features is present at that location



<https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac>

# CONVOLUTIONAL LAYERS

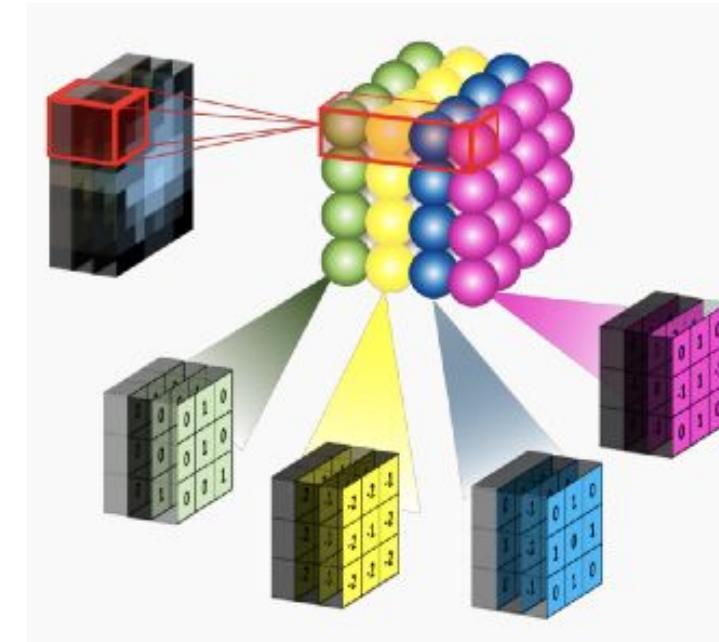
Summary:

Filters are composed of kernels (learned) and extract features

One bias per filter

Applied to original image or feature map from another layer

An activation function is used on every value of the feature map



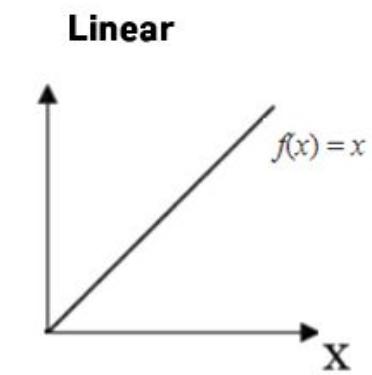
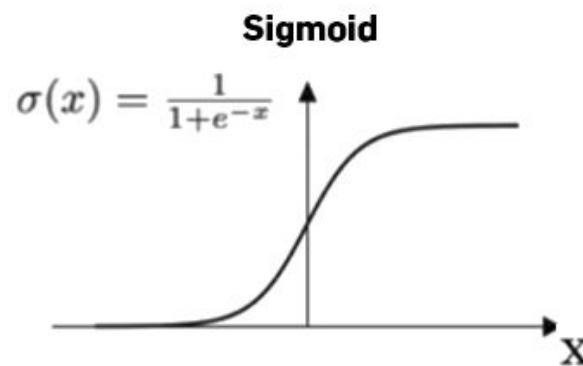
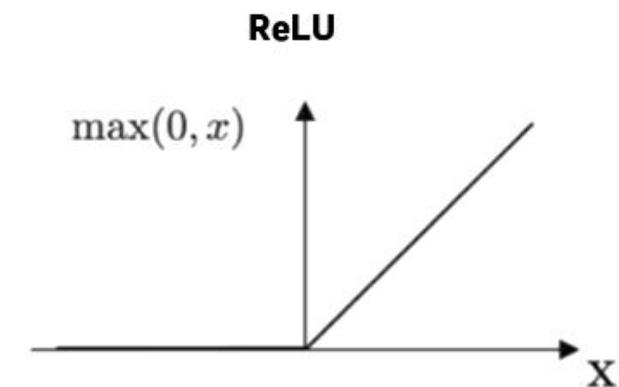
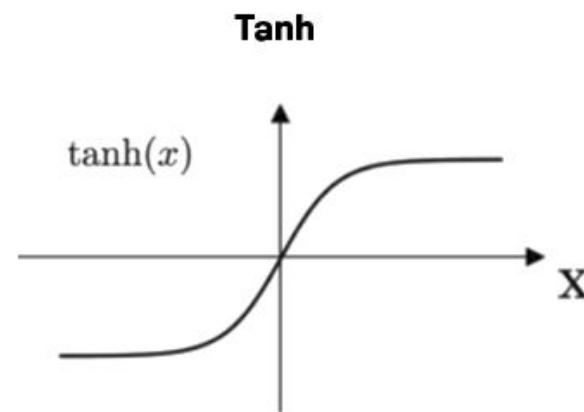
<https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac>

# ACTIVATION FUNCTIONS

Adds non-linearity

ReLU is typically a good start

Cheap  
Stable



# WHY DO WE NEED NON-LINEARITY?

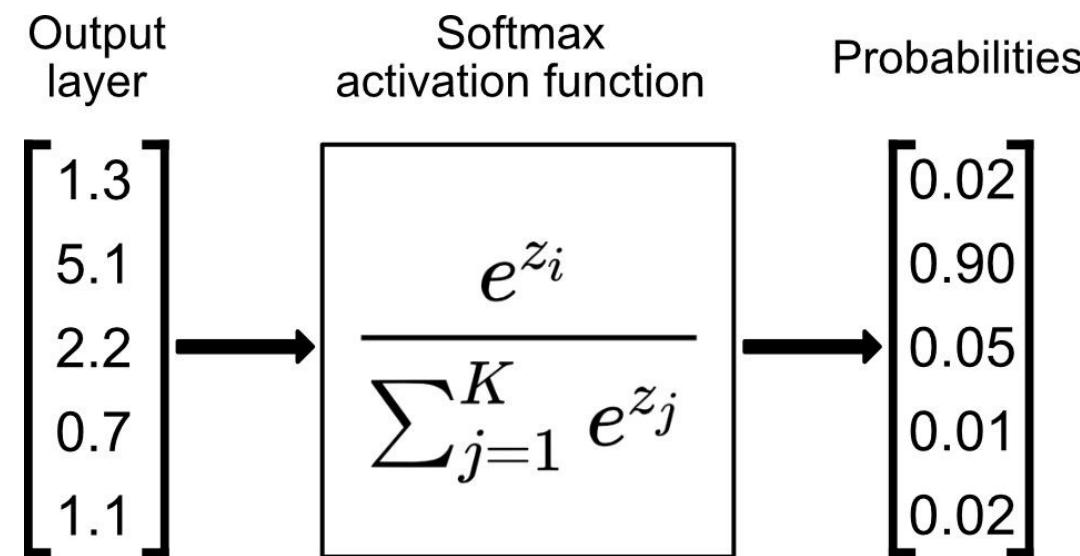
A neural network without an activation function is essentially just a linear regression model

The non-linear transformation to the input making it capable to learn and perform more complex tasks

# ACTIVATION FUNCTIONS - SOFTMAX

Used for building a multi-class classifier

Different from a Sigmoid function because it ensures the sum of the outputs along channels is 1



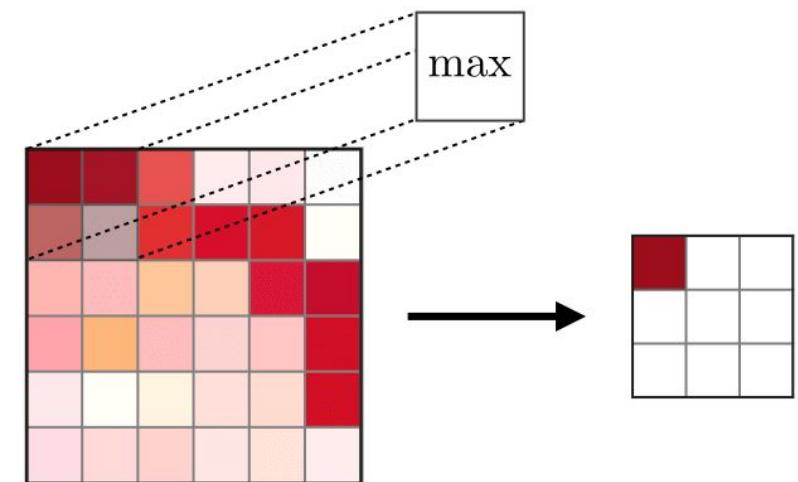
# POOLING LAYER

Used to reduce dimensionality, typically applied after a convolution layer

Max pooling (most common) - selects the largest values on the feature maps and passes these to next layer

Outliers are when the network sees the feature

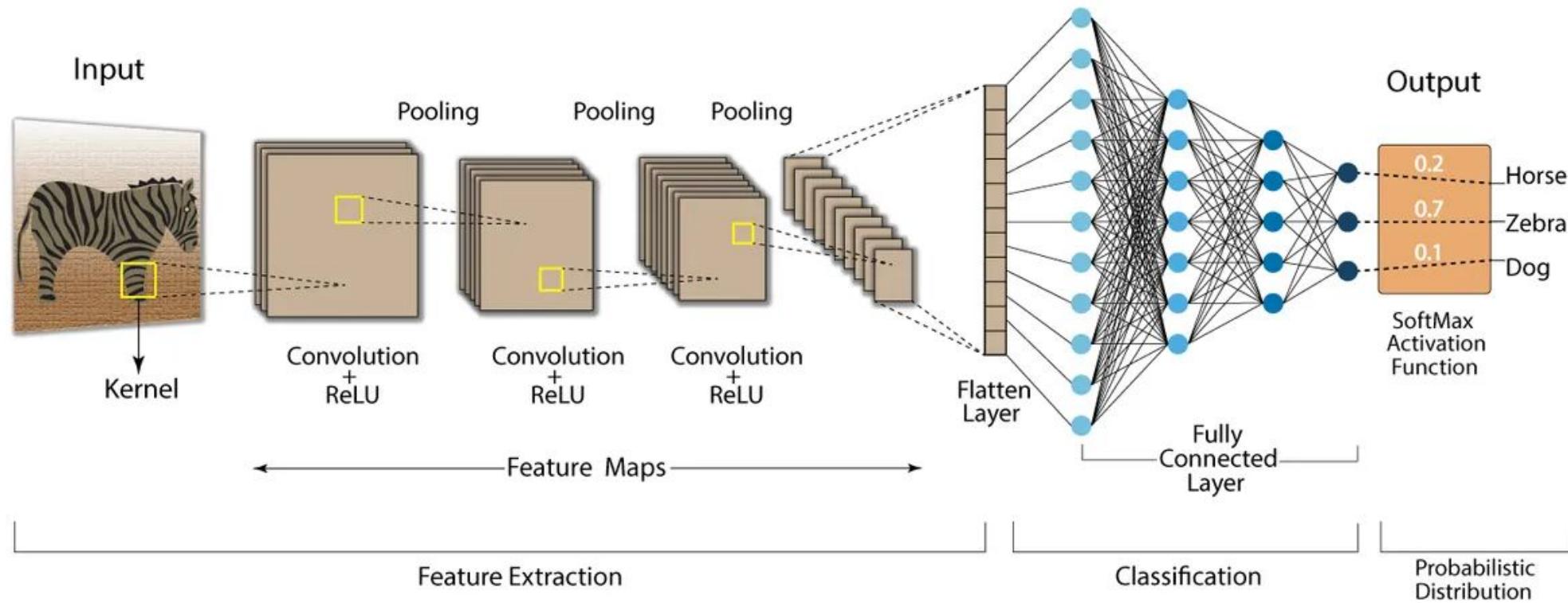
Average pooling - selects the average value in a filter region



<https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks>

# FULLY CONNECTED LAYERS

Aggregates input from the final feature maps



# MODEL LAYERS IN PYTORCH

`torch.nn` - PyTorch's basic building blocks for graphs

- various functions for convolutional, pooling, and many other types of layers
- non-linear activations
- loss functions

`torch.nn.Module` - basic class for all neural network modules

`torch.nn.Sequential` - sequential container

# AUTOGRAD

Backpropagation is used to calculate the gradient of a weight

For each iteration, several gradients are calculated and something called a computation graph is built for storing these gradient functions

PyTorch does it by building a Dynamic Computational Graph (DCG)

You may disable gradient tracking:

- To mark some parameters in your neural network as frozen parameters (common when using a pre-trained network)
- To speed up computations when you are only doing forward pass

# TRANSFER LEARNING (PRE-TRAINED CNNS)

Large networks trained on [ImageNet](#) (image database) contain information in the final convolutional layers or early fully-connected layers about:

- How an image is composed and
- Combinations of edges/shapes

Fixed feature extractor - freeze all layers except the last full-connected layer to keep the features but not the classifier

Fine-tuning the model - train with a small learning rate to ensure it doesn't 'unlearn' previous knowledge

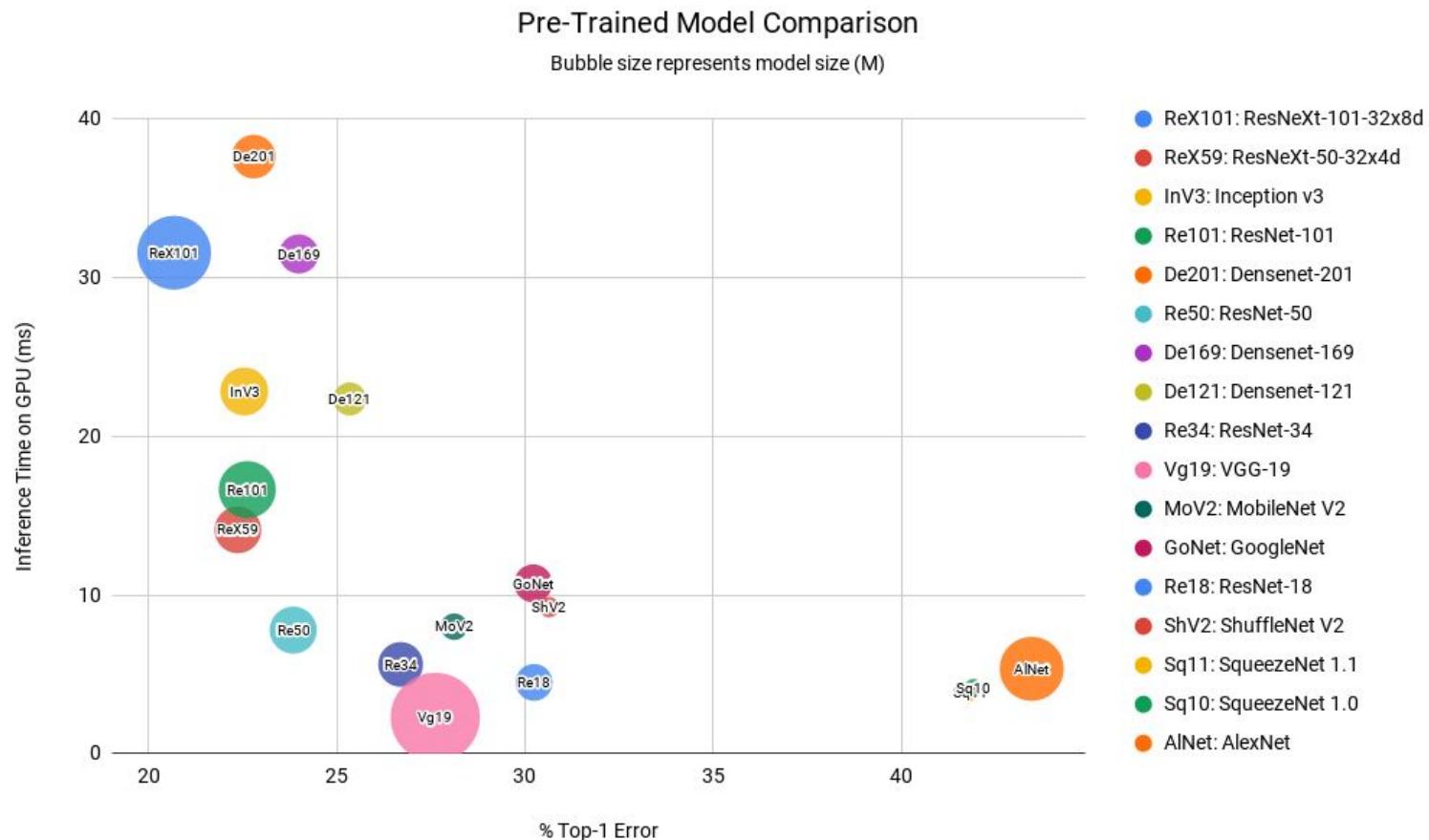
# TRANSFER LEARNING IN PYTORCH

PyTorch `torchvision.models` - contains image classification models of various architectures with the option for `pretrained=True` to download the weights, shown [here](#)

All pre-trained models expect input images normalized in mini-batches of 3-channel RGB images in shape  $(3 \times H \times W)$  where  $H$  and  $W$  are at least 224 and then  $\text{mean}=[0.485, 0.456, 0.406]$  and  $\text{std}=[0.229, 0.224, 0.225]$

Check out [this article](#) for more information on reproducibility and considerations before using pre-trained models

# TRANSFER LEARNING IN PYTORCH



<https://learnopencv.com/pytorch-for-beginners-image-classification-using-pre-trained-models/>

# MACHINE LEARNING REVIEW

Clean and explore your data

Imbalanced classes - use sampling to rebalance

Train/Test sets - use stratification to ensure distribution of classes in both sets, don't apply transforms to test set

Encode categorical variables (one-hot, etc)

# LOSS FUNCTIONS

## Regression

- Mean-squared error loss
- Mean absolute error loss

## Binary Classification

- Binary cross entropy
- Hinge Loss

## Multi-Classification

- Multi-class cross entropy
- Kullback Liebler Divergence

# CROSS ENTROPY

Default loss function

Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for all classes in the problem

The score is minimized and a perfect cross-entropy value is 0

Note: Binary cross-entropy is a special case for two classes.

`nn.BCEWithLogitsLoss` can take raw unnormalized logits, used in our tutorial because of one hot encoding

# KULLBACK-LEIBLER DIVERGENCE

Also called relative entropy, this is a common form of statistical distancing

Measures the difference between two probability distributions -

- Entropy tells you how much information is in a probability distribution given some data
- Therefore you can find out how much is *lost* when you change the distribution

# ACTIVITY: BUILD CNN ARCHITECTURE AND TRAIN MODEL

1. Open notebook ‘2-Build-Model.ipynb’ and run each cell to see the CNN architecture and training code

# Improving Training Performance

# GPUS

Perform multiple, simultaneous computations enabling the distribution of training processes

Large number of simple cores vs a few complicated cores (CPU)

Note that GPUs can have a bandwidth bottleneck issue - transferring large amounts of data to the GPU can be slow

- Optimizing this process is covered in the tutorial notebooks

# GPUs

It's best practice to use a command like the one below instead of hard-coding GPUs, so your code can still run (very slowly!) on a cpu-only machine:

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'  
print('Using {} device'.format(device))
```

# SYSTEM UTILIZATION

- While a model is training, you may want to confirm that it's making full use of your hardware.
- If it isn't, that's a sign that you can do things like train larger batches, a larger model, or do more things in parallel.
- `nvidia-smi` shows information about the GPUs in your system and their memory/processor usage.
- `top` is a commonly-used linux command showing the most active processes.

# SYSTEM UTILIZATION

- You may occasionally need to clear cached data from the GPU between model trainings
- `torch.cuda.empty_cache()`

# OPTIMIZATION

In software development, you don't want to put effort into making something faster unless you know that the part you are working on is the bottleneck.

`line_profiler` is a great Jupyter extension to profile functions line-by-line:

```
%load_ext line_profiler
```

# ACTIVITY: BUILD CNN ARCHITECTURE AND TRAIN MODEL

1. Open notebook ‘3-Improve-Performance.ipynb’ and run each cell to see the increases in speed after optimization techniques have been applied

# Optimizing Model Performance

# EPOCHS

The number of times the learning algorithm will evaluate the entire training dataset

More epochs means more time to converge, but also more training time

Too many epochs may lead to overfitting

# OPTIMIZERS

Stochastic Gradient Descent - calculates gradients on batches of examples at a time

Adam - stands for adaptive moment estimation, utilizes momentum by adding fractions of previous gradients to the current one

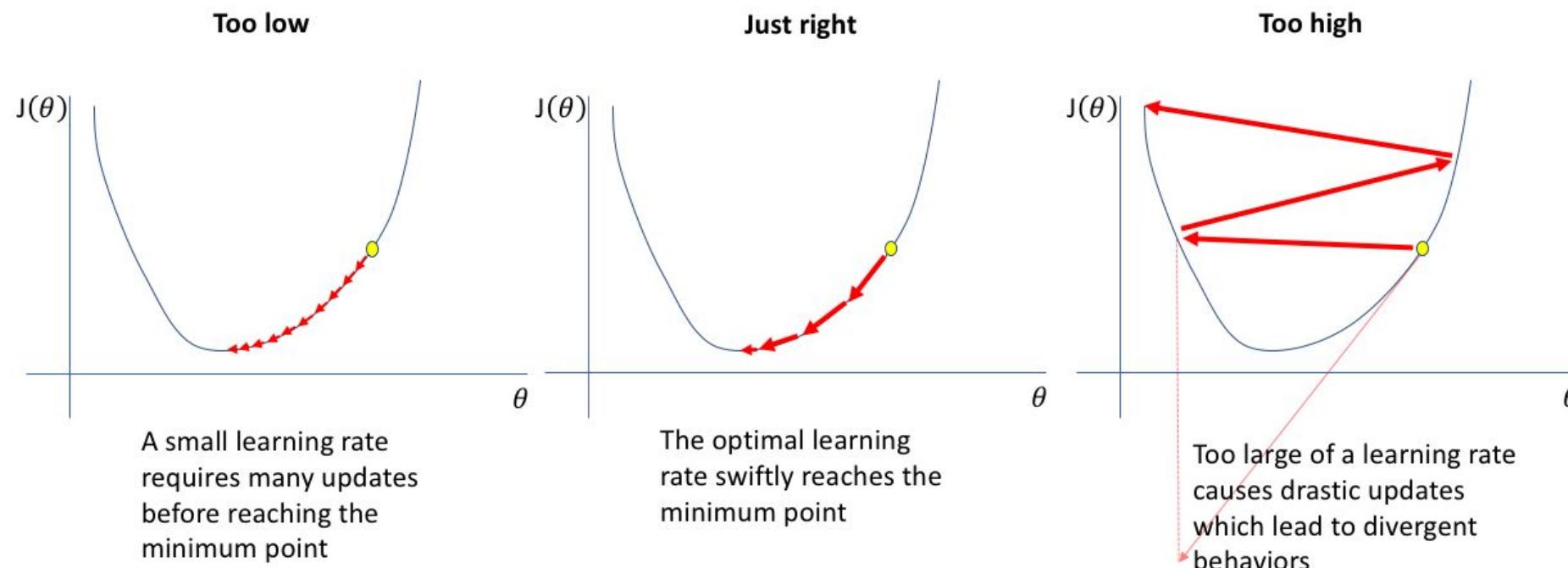
AdaGrad - adapts the learning rate to individual features, so some of the weights will have different learning weights

- good for sparse data
- learning rate tends to get really small

RMSProp - special version of AdaGrad but only accumulates gradients in fixed window

# LEARNING RATE

Hyperparameter between 0 and 1 that controls how much to change the model in response to the estimated error each time the weights are updated



# LEARNING RATE

Decaying Learning Rate - decreasing the learning rate with increases in epochs

Scheduled Learning Rate - dropped at a predetermined frequency to quickly move to a range of 'good' values and then fine tune

Adaptive Learning Rate - monitoring the performance of the model on the training dataset by the learning algorithm and adjusting the learning rate in response

- Methods include Adam (most popular), AdaGrad, and RMSProp

# LAYER PARAMETERS

Convolutional layers - number and size of kernels (most important), activation function, stride, padding, regularization type and value

Pooling layers - stride, size of window

Fully connected layers - number of nodes, activation function (depends on role of layer, typically ReLu or softmax)

# ACTIVITY: MODEL TUNING COMPETITION

1. Open notebook '4-Hyperparameter-Tuning.ipynb'
2. Try different changes to hyperparameters/optimizers/etc.
3. When you're happy with your result, run the last cell to submit your results to a leaderboard
  - a. You can rerun the cell to overwrite your old result

# Summary

# SUMMARY

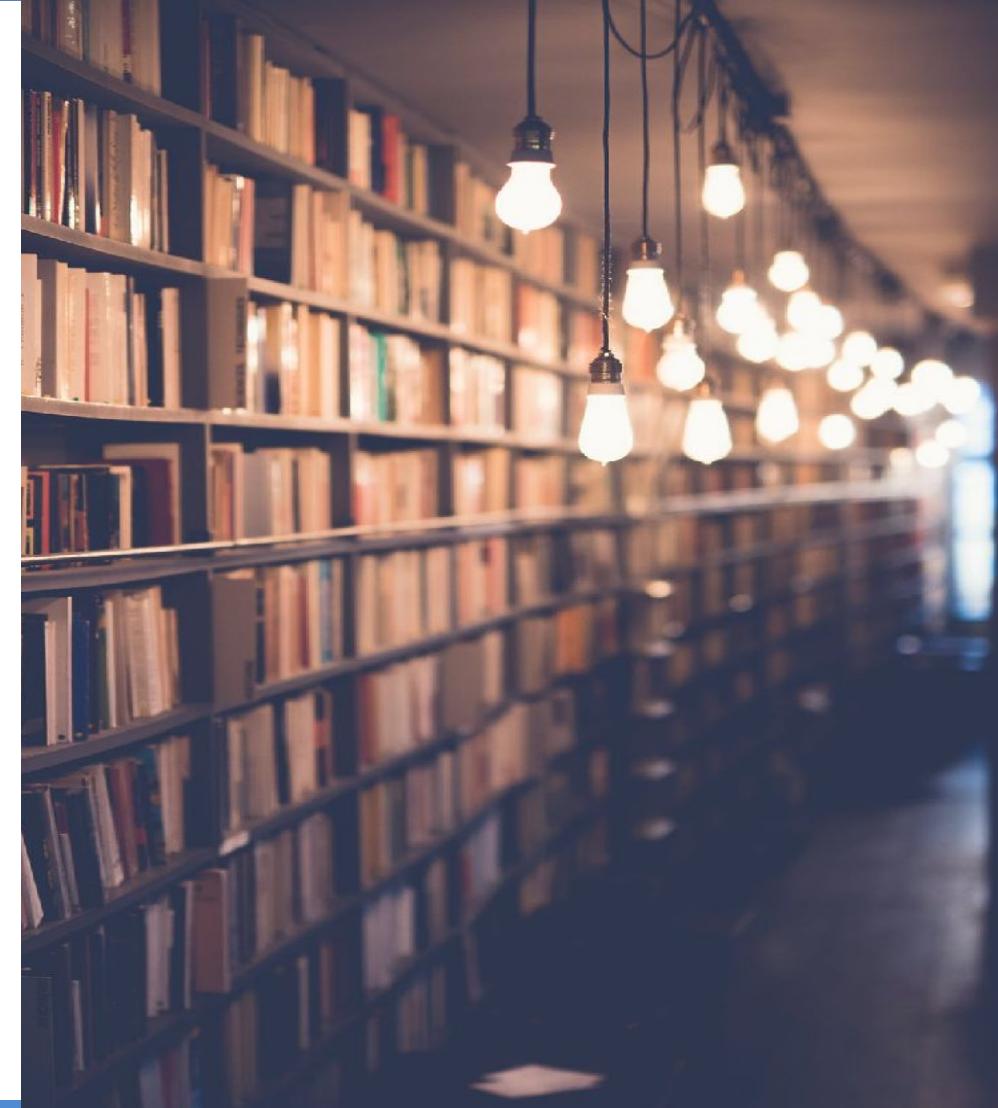
While it is important to understand the parameters of different layer types, you will often use a pre-built architecture

Test out different parameters for number of images, transforms, learning rate, epochs, and optimizers

Optimization for speed can occur when reading and processing images, setting Dataloader parameters, and freezing layers

# WRAPPING UP A PROJECT

- Best Practices
  - Store positive and negative results
  - Preserve synthesis, intermediate results, code, data, and environment
- Common Pitfalls
  - Repeated quiet failures
  - Old analysis doesn't run



# Sharing Models

# CREATE INTERACTIVE DATA PRODUCTS

- Interactive dashboards
- Model driven applications
- Visualizations to explain model results
- Tailor design to your audience:  
customer, business user, or executive

# WEB APPLICATIONS

- Each project can host one web app
- Detailed instructions for [Dash](#), [Flask](#), [Shiny](#), [Streamlit](#), and [Django](#)
- See our [Publishing course](#) for more



# WEB APPS IN DOMINO

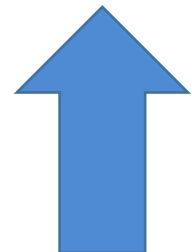
App Files



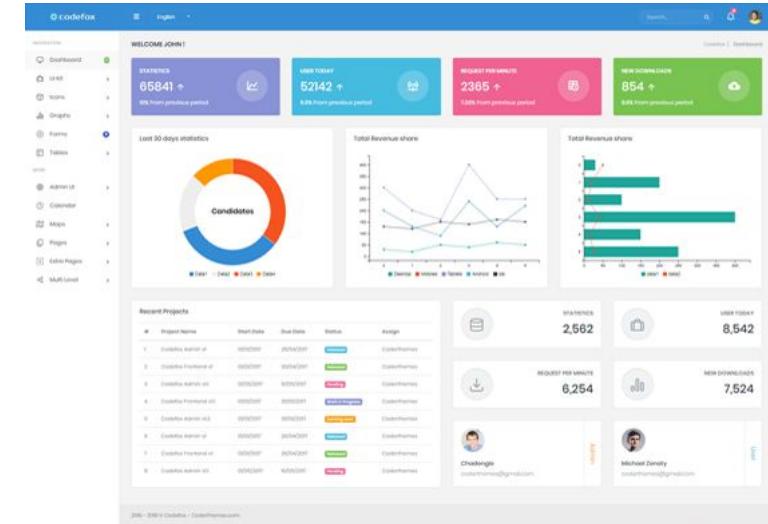
/mnt/...



Start an app server



app.sh



# VISUALIZATION PACKAGES

- Matplotlib - low level, can customize easily
- Pandas visualization - easy to use, built on matplotlib
- Seaborn - high level, better default styles
- Plotly - easy to use interactive graphs
- Bokeh - interactive, targets web browsers for presentation

# MODEL API: UNIVERSAL TRANSLATOR

- Developer can create an application in the language of their choice AND still use your R or Python model
- Domino model APIs serve your code as a low-latency web service
- You don't need any API design experience to turn your Python or R code into a Model API
- Details in our [Publishing course](#)



# BUILD MODEL API IN DOMINO

Script:

```
load model  
  
function (input):  
    result = call model (input)  
    return result
```



Publish New Version Step 2 of 2

The file containing the code to invoke (must be a Python or R file)

The function to invoke

Our API will handle marshalling input arguments to the function, and serializing the return values back as JSON.

Calling your Model

Route:

Model URL: <https://try.dominodatalab.com:443/models/602eeb4f6c5f193a9019f4d2/latest/model>



# Advanced Features in Domino

# DEMO

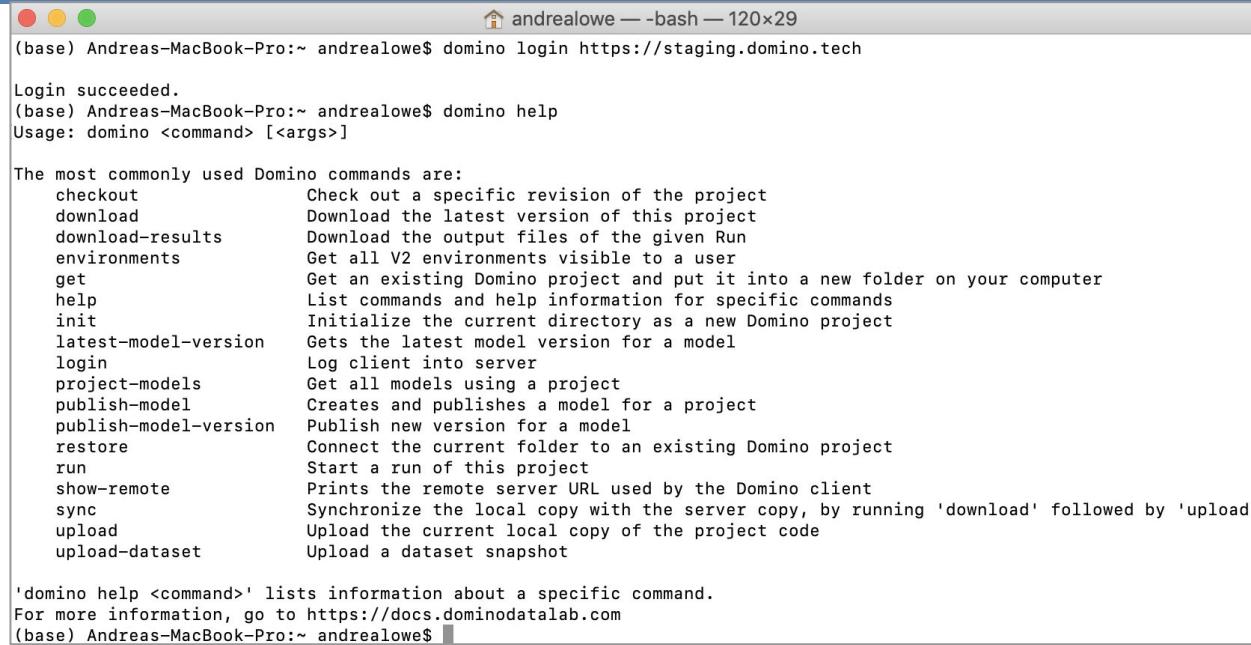
# GET MATERIALS - DOMINO COMMAND LINE INTERFACE (CLI)

- Work locally but run your code in Domino
- Sync a local folder with a Domino project
- Easily upload many/large files to Domino

To start, [download the CLI](#) by clicking on your user name in the bottom left

After the download completes, login using the terminal (you may be given a link for authentication)

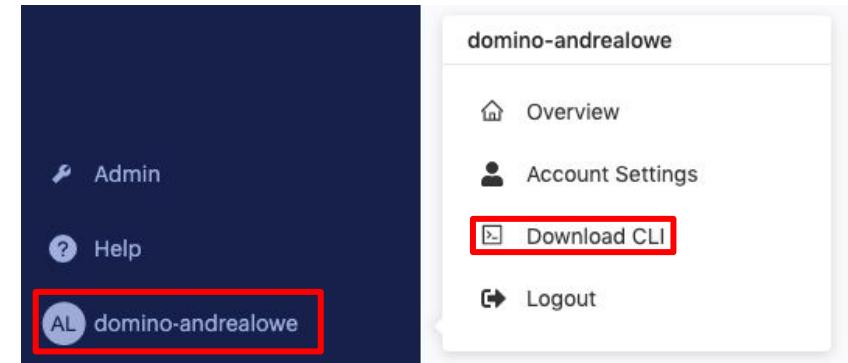
See list of possible commands [here](#) and in [Domino 201](#)



```
andrealowe -- bash -- 120x29
(base) Andreas-MacBook-Pro:~ andrealowe$ domino login https://staging.domino.tech
Login succeeded.
(base) Andreas-MacBook-Pro:~ andrealowe$ domino help
Usage: domino <command> [<args>]

The most commonly used Domino commands are:
  checkout          Check out a specific revision of the project
  download          Download the latest version of this project
  download-results  Download the output files of the given Run
  environments      Get all V2 environments visible to a user
  get               Get an existing Domino project and put it into a new folder on your computer
  help              List commands and help information for specific commands
  init              Initialize the current directory as a new Domino project
  latest-model-version Gets the latest model version for a model
  login             Log client into server
  project-models    Get all models using a project
  publish-model     Creates and publishes a model for a project
  publish-model-version Publish new version for a model
  restore           Connect the current folder to an existing Domino project
  run               Start a run of this project
  show-remote       Prints the remote server URL used by the Domino client
  sync              Synchronize the local copy with the server copy, by running 'download' followed by 'upload'
  upload            Upload the current local copy of the project code
  upload-dataset   Upload a dataset snapshot

'domino help <command>' lists information about a specific command.
For more information, go to https://docs.dominodatalab.com
(base) Andreas-MacBook-Pro:~ andrealowe$
```



# RESOURCES

Tutorial materials: <https://github.com/dominodatalab/DeepLearningTutorial>

PyTorch docs - <https://pytorch.org/tutorials/>

Great visualization videos on math and machine learning concepts -  
<https://www.3blue1brown.com/>



Thank You  
[www.dominodatalab.com](http://www.dominodatalab.com)