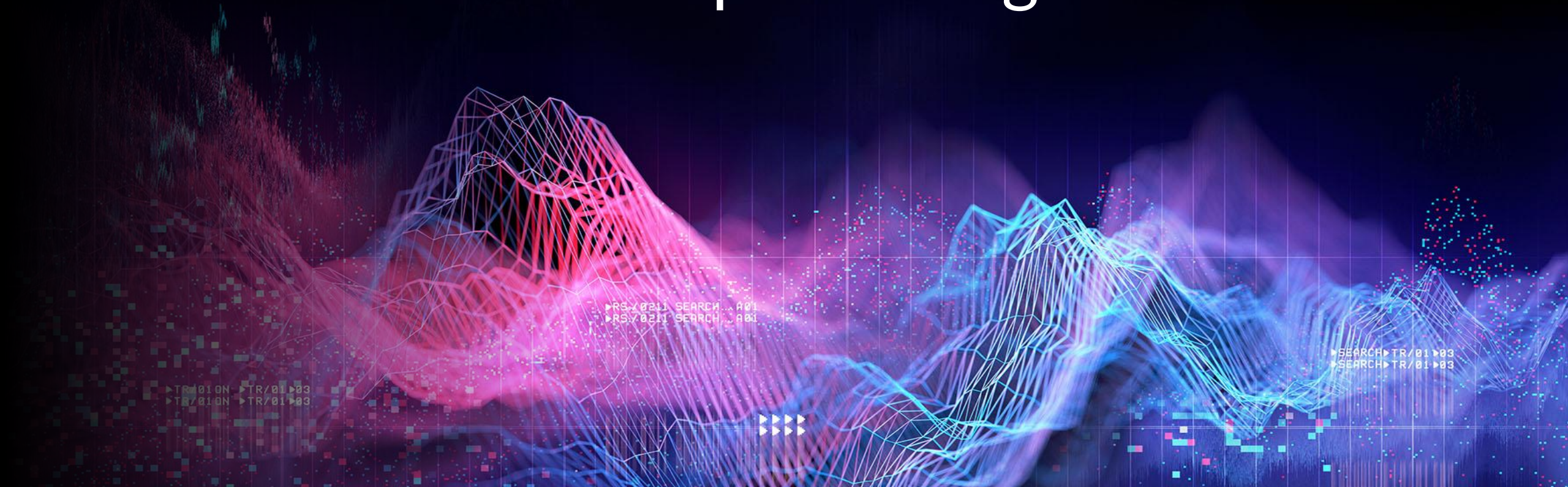
DOMINO

Introduction to Deep Learning



AGENDA

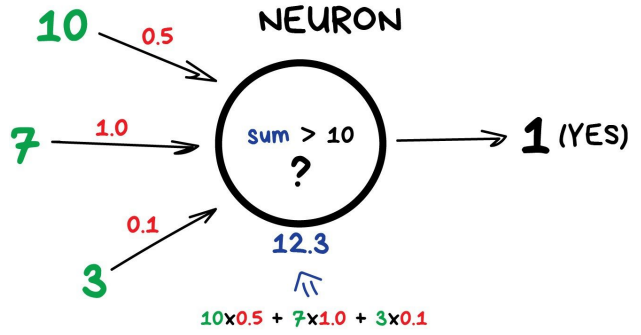
- Introduction
- Deep Learning in Practice
- Evaluating Models
- Optimizing Models
- Summary

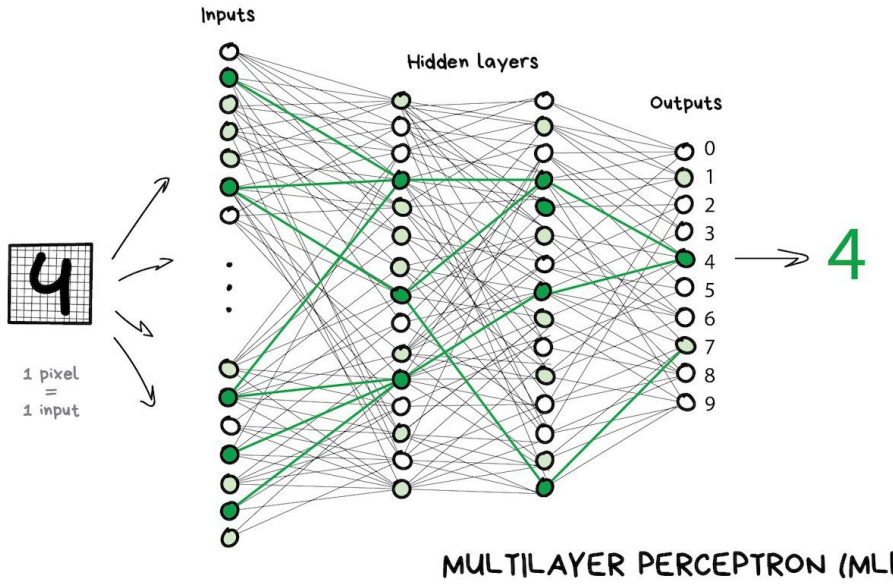
Introduction

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

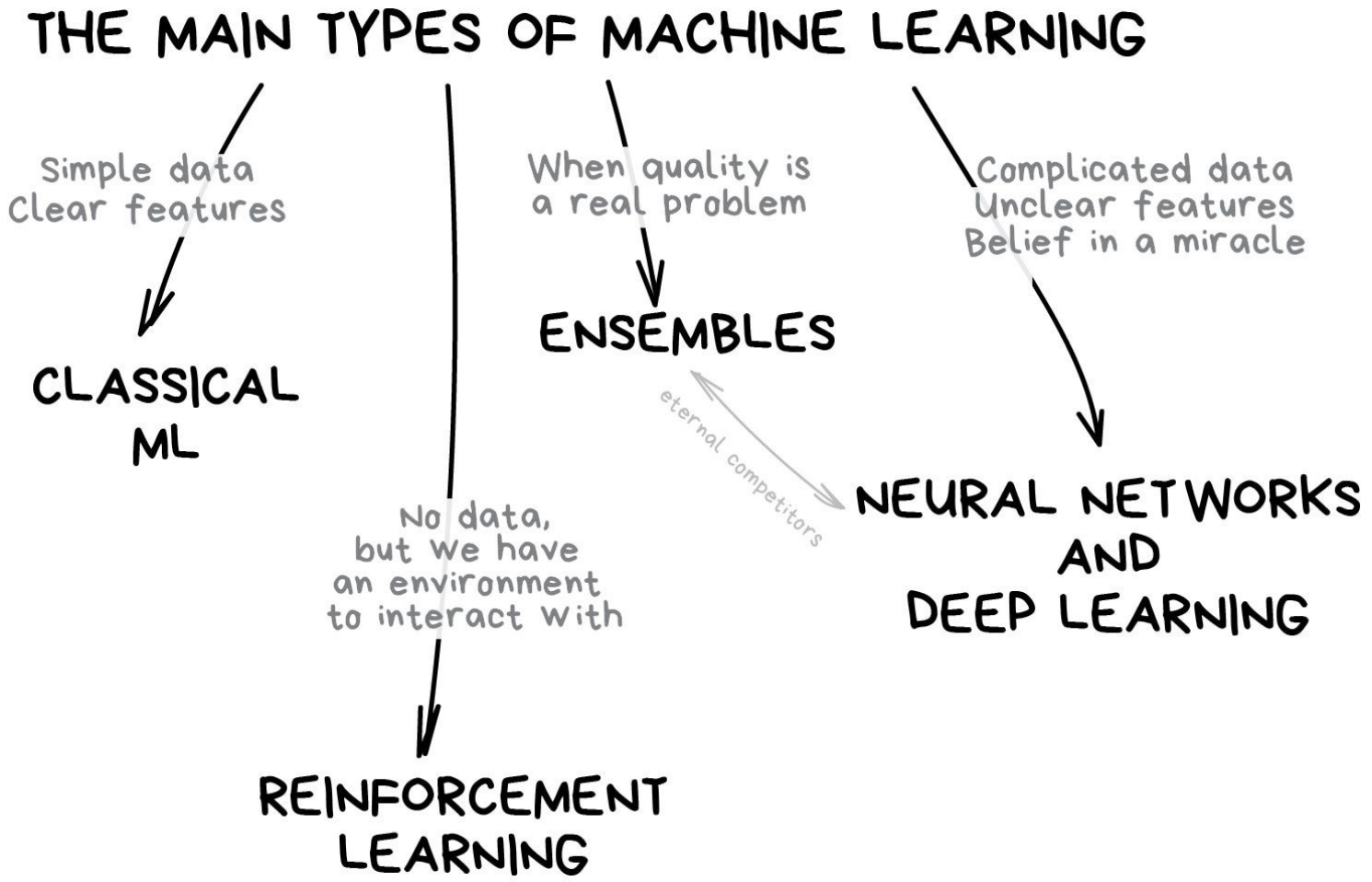




https://vas3k.com/blog/machine_learning/

DOMINO

THE MACHINE LEARNING LANDSCAPE



https://vas3k.com/blog/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

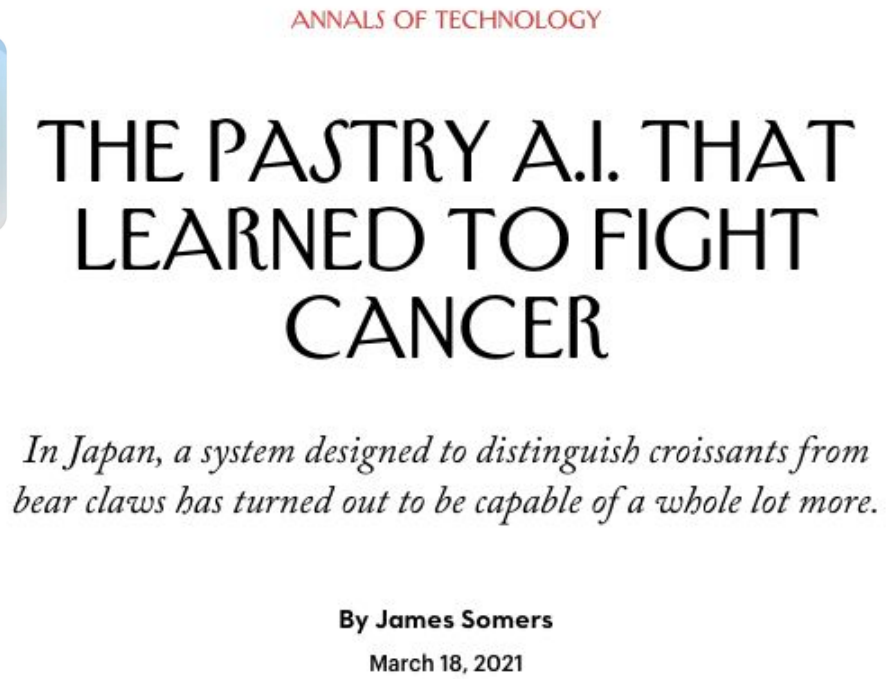
Self-Driving Vehicles

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



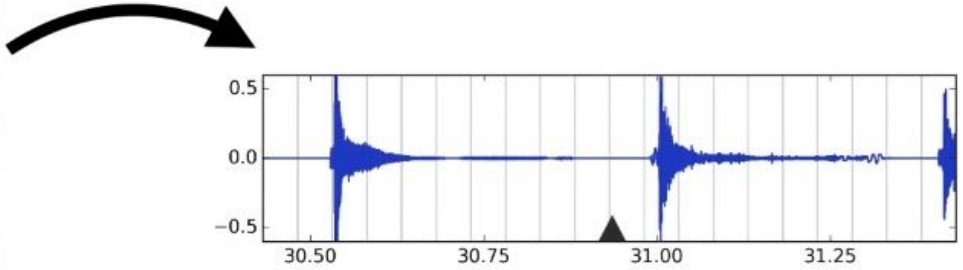
https://www.newyorker.com/tech/annals-of-technology/the-pastry-ai-that-learned-to-fight-cancer

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



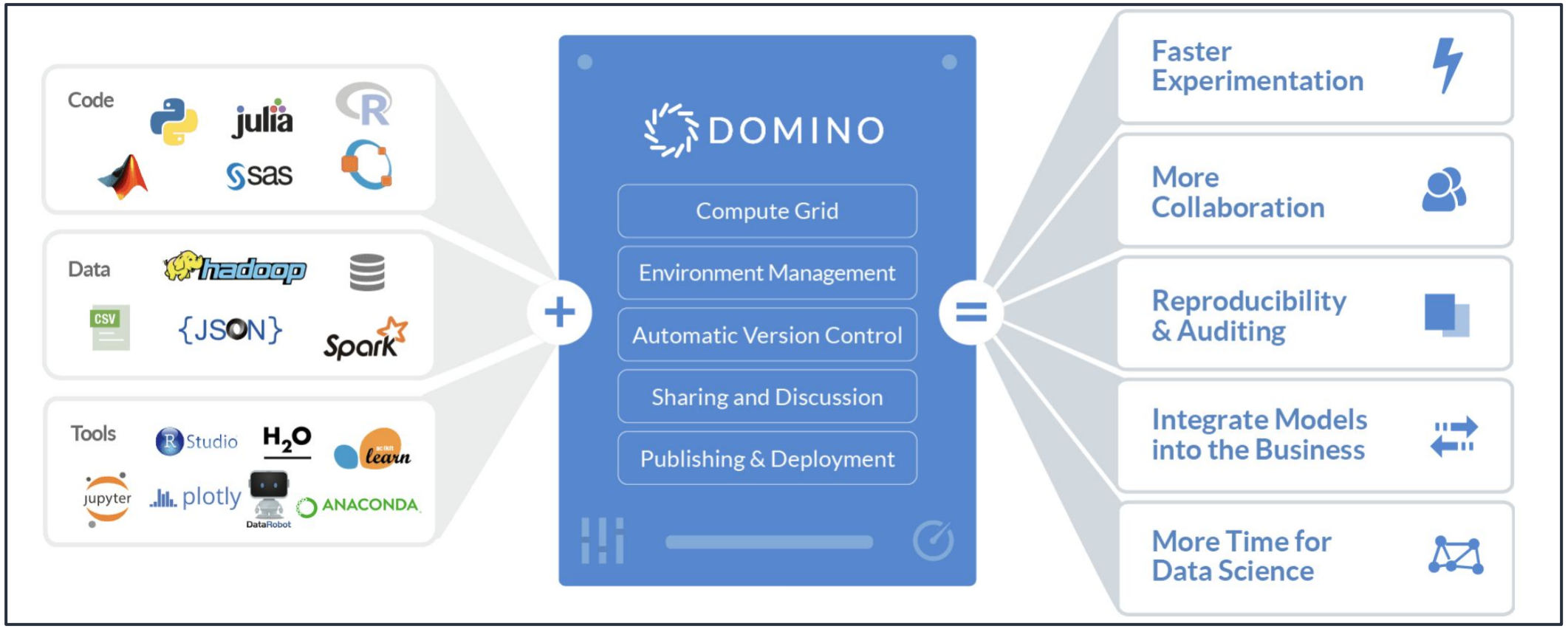
Predicted soundtrack

https://news.mit.edu/2016/artificial-intelligence-produces-realistic-sounds-0613

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

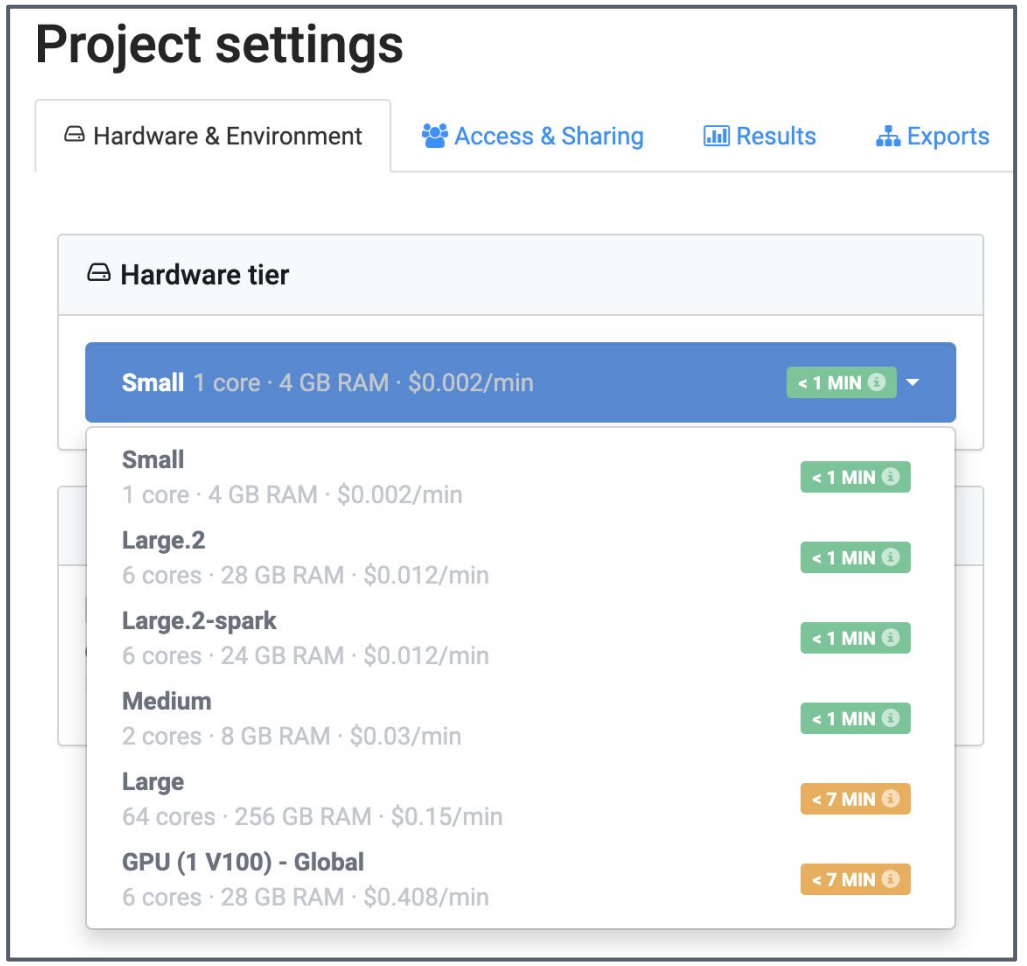


Projects: The Main Unit of Organization

- 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



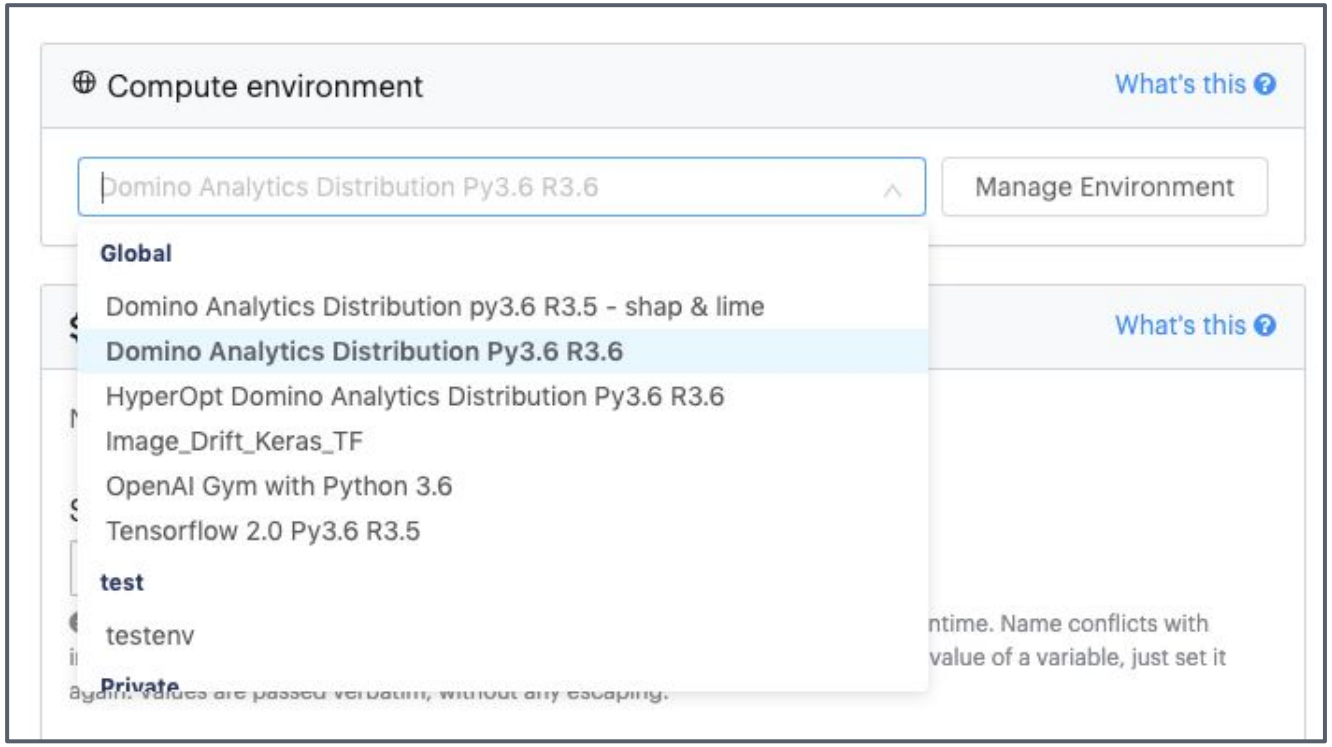
DOMINO

Project Settings: Compute Environments

Specification for the container where your job or workspace will run

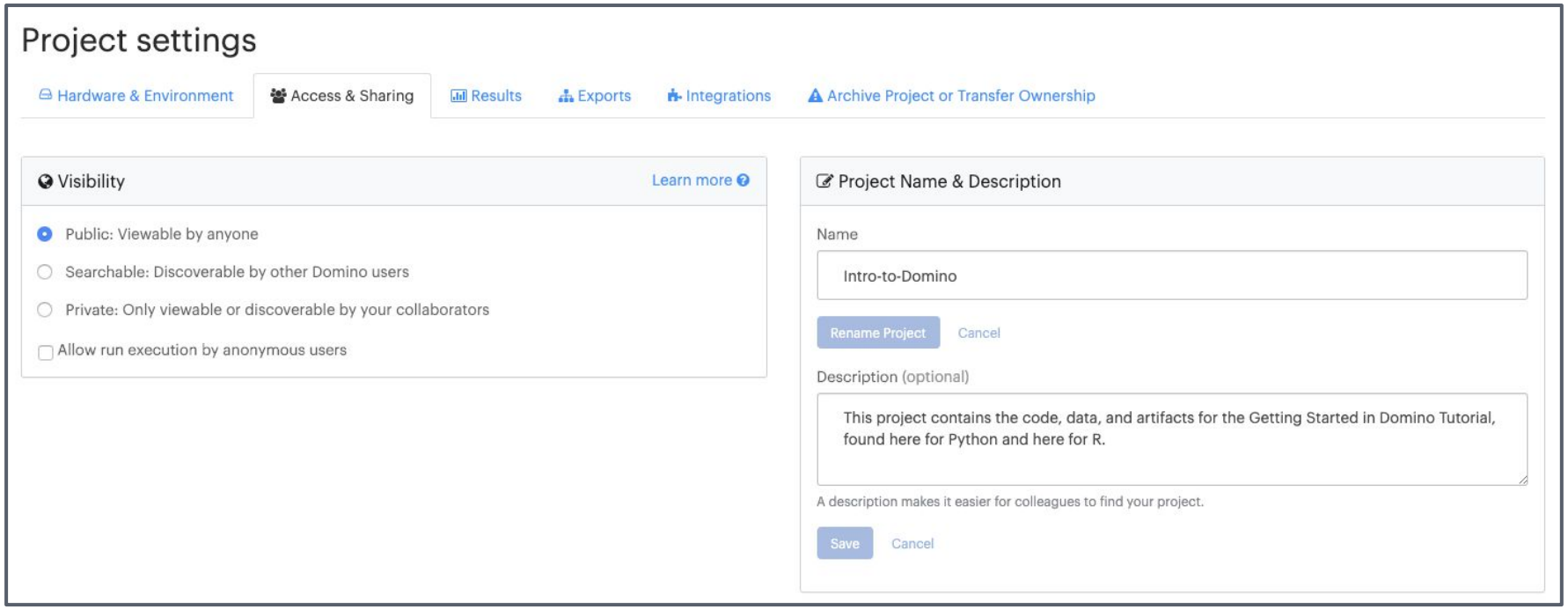
Automatically versioned and easily shared

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



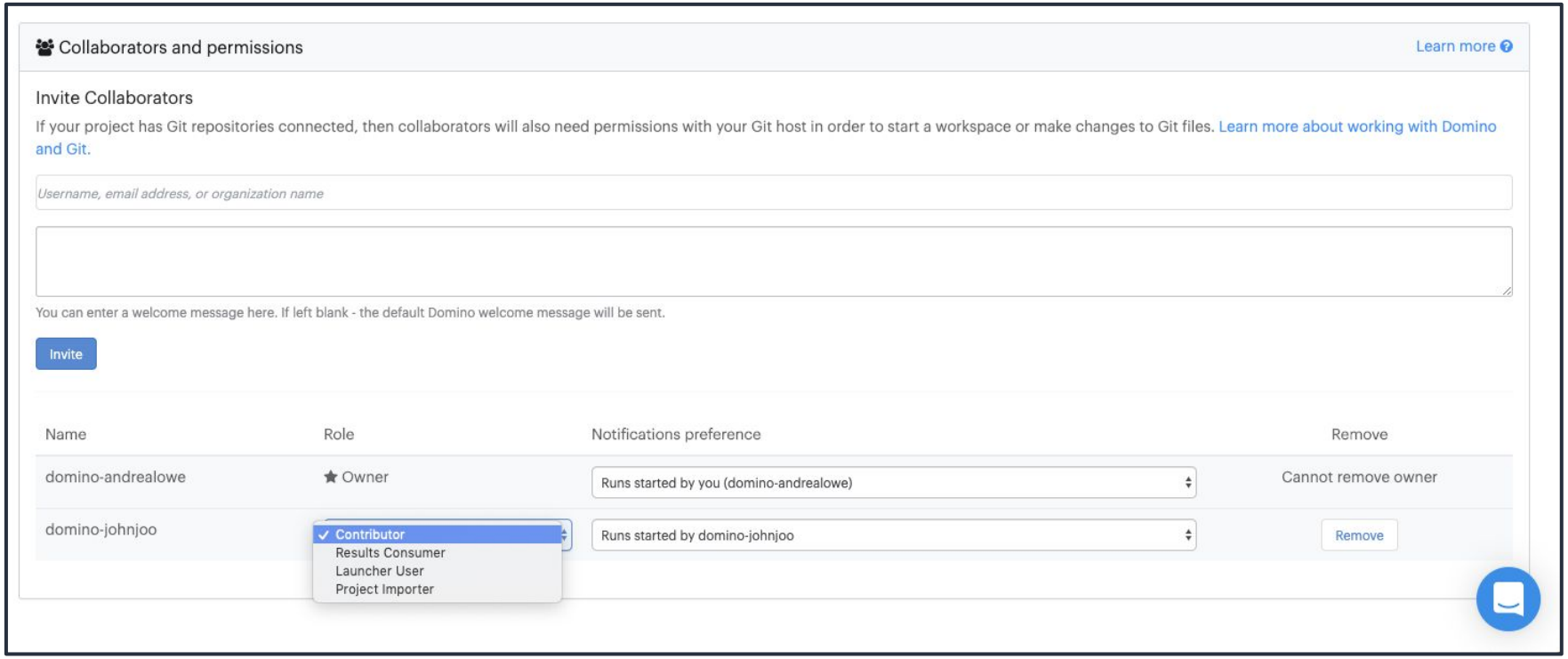
Project Settings: Access and Sharing

Set the visibility of your project



Project Settings: Access and Sharing

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

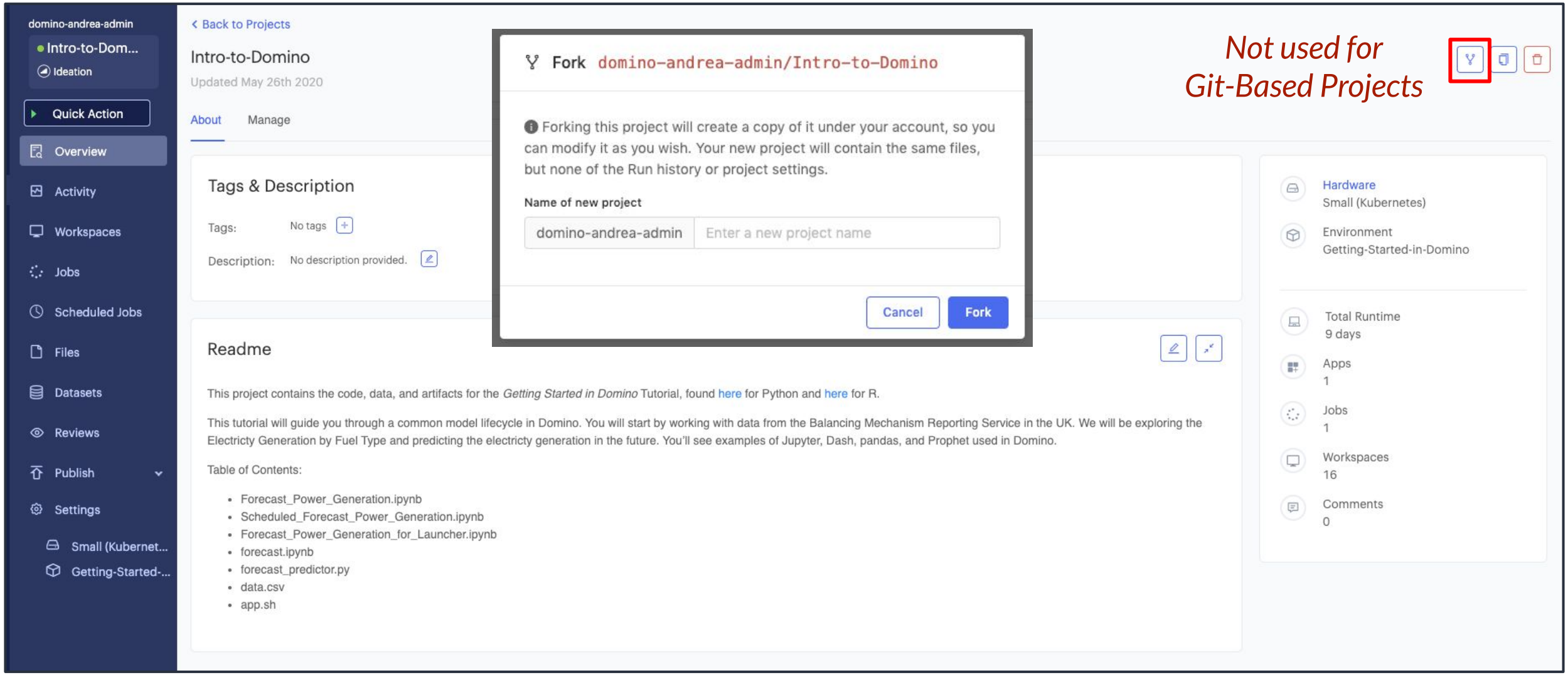


Project Settings: Permissions

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

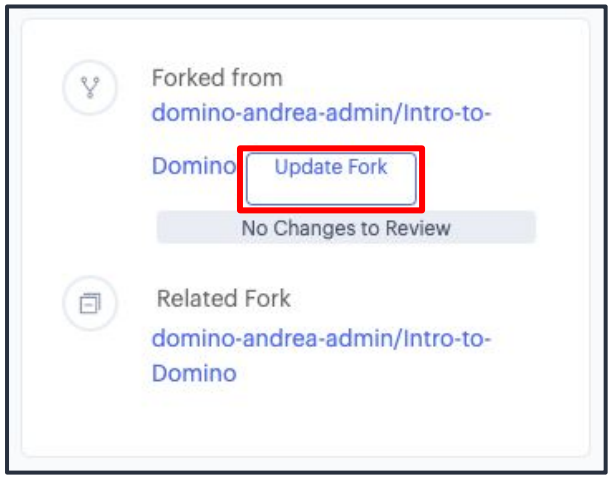
Projects: Fork/Merge



DOMINO

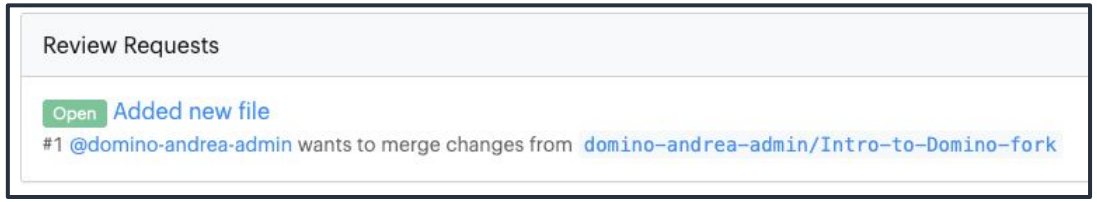
Projects: Fork/Merge

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:

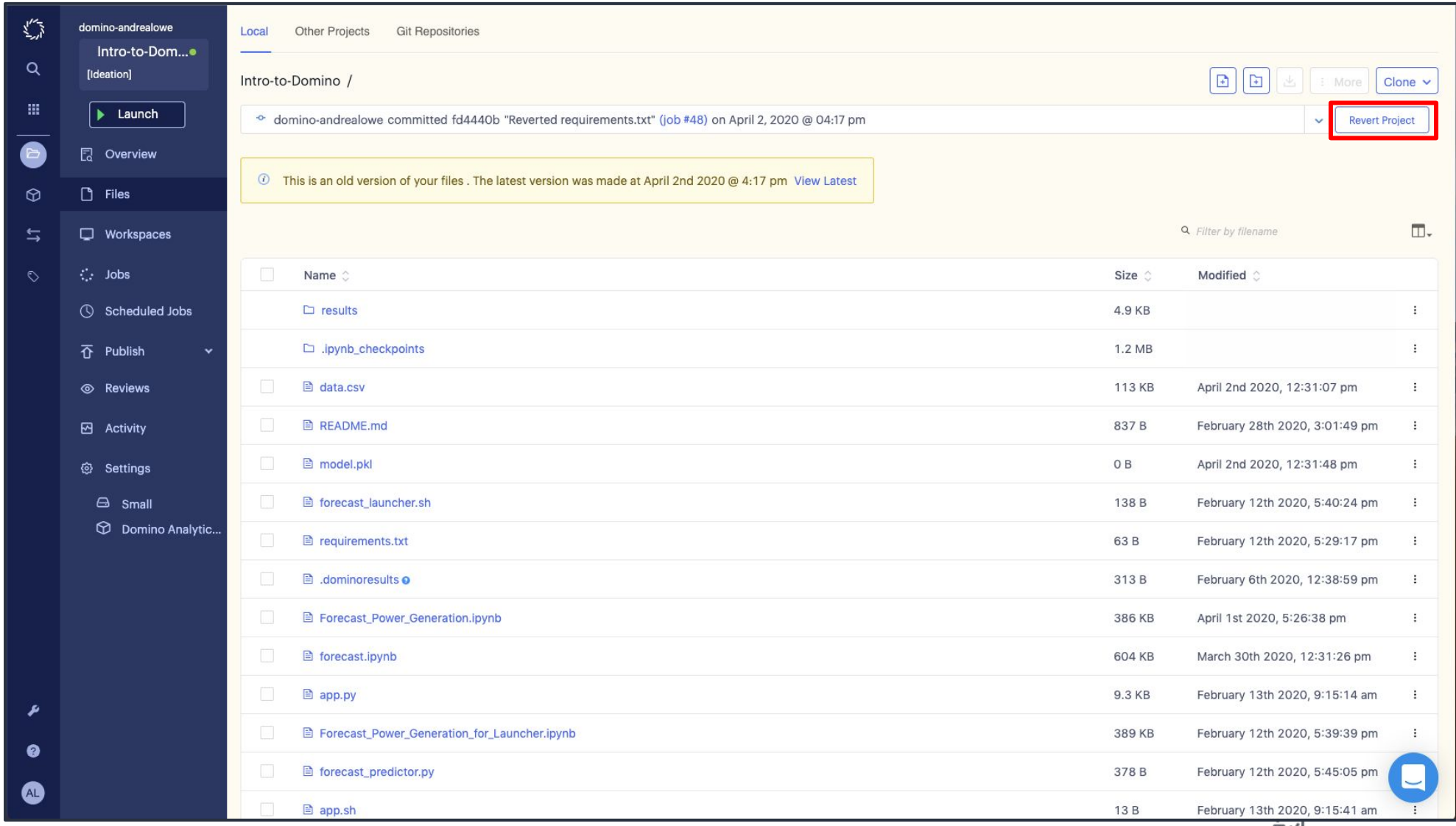


DOMINO

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*



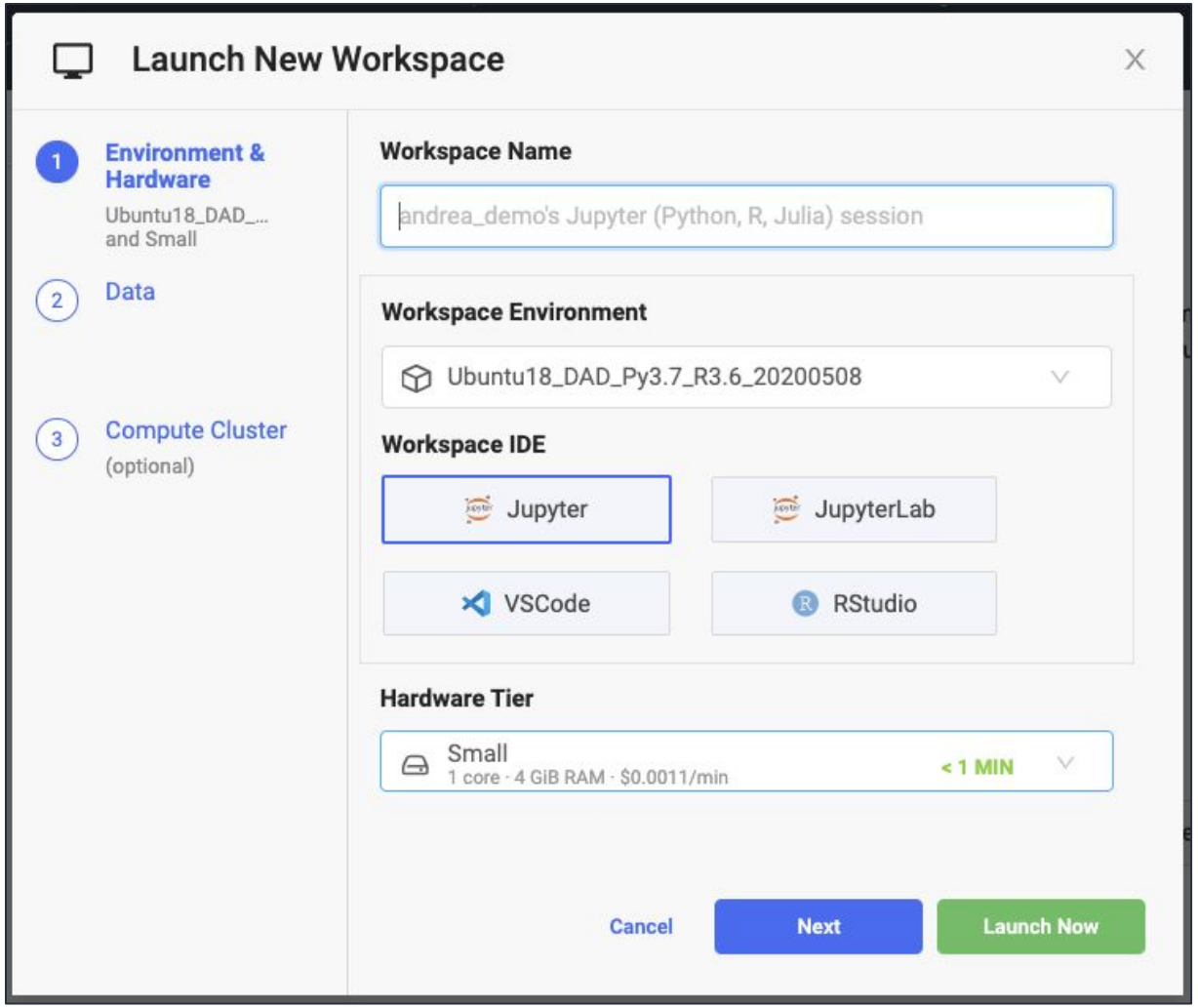
Run Code in Domino

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

Configure a Workspace

Configure your workspace at launch:

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

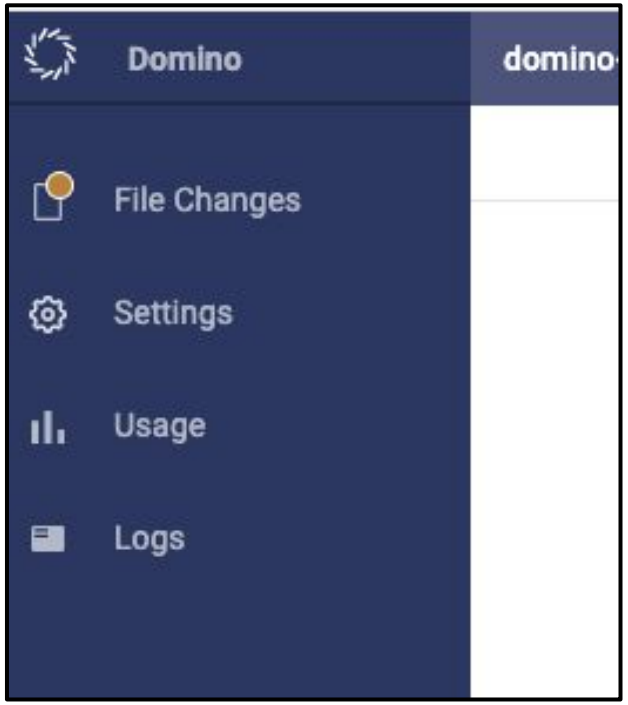


DOMINO

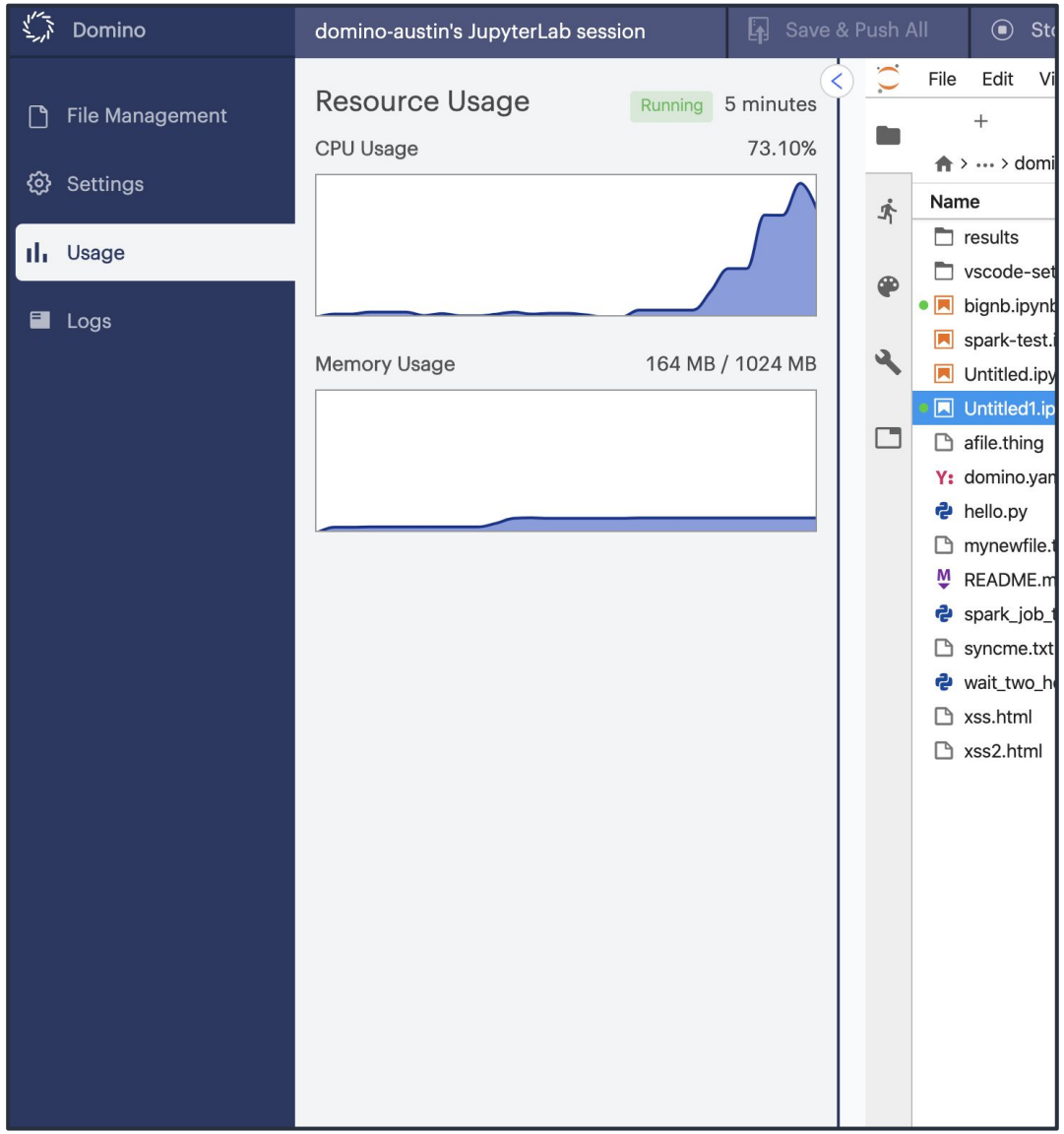
Workspace Navigation Bar

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



In the Usage pane, you'll find graphs of your workspace CPU Usage and Memory Usage over time.

DOMINO

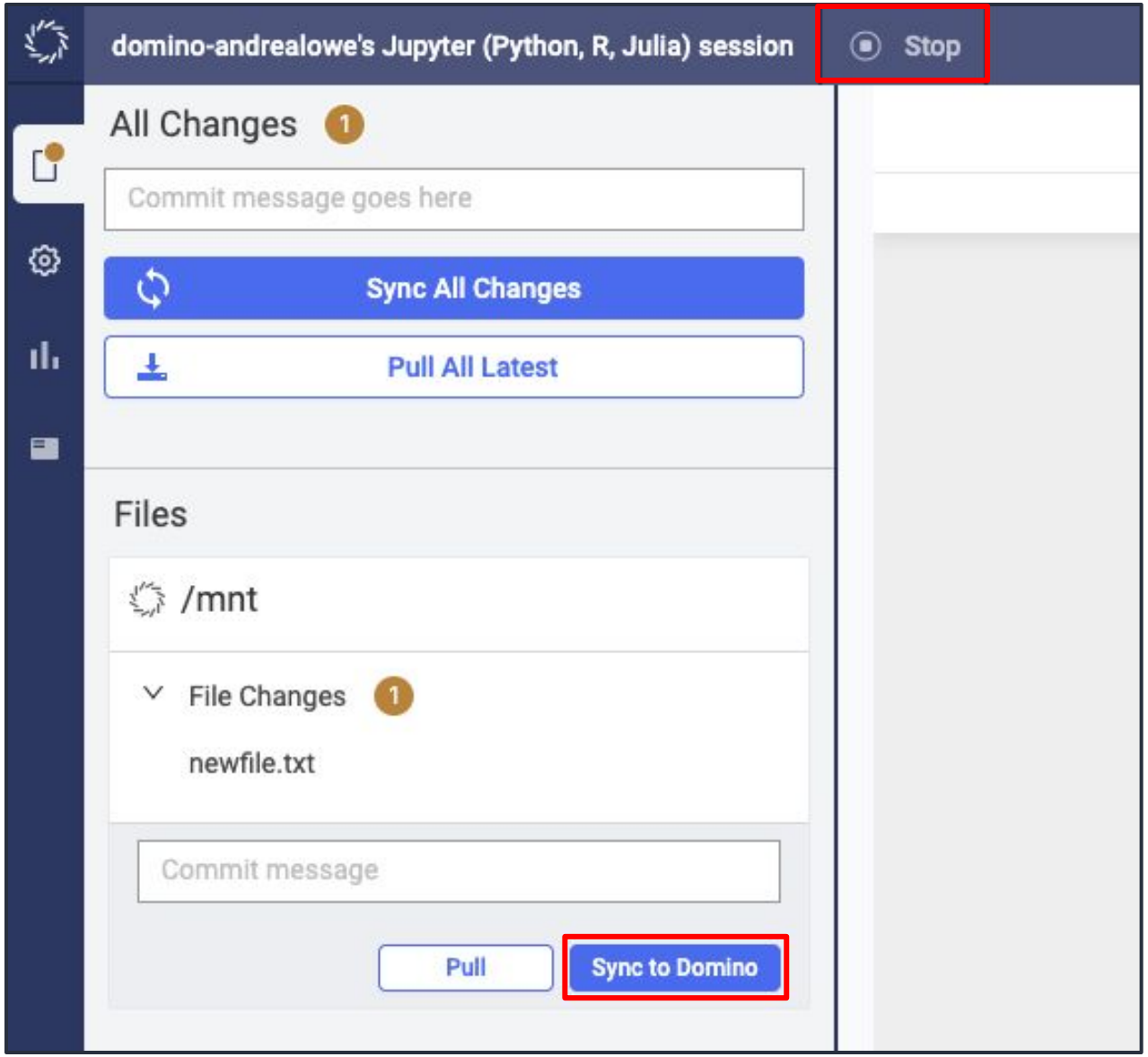
Workspace Sync and Stop

Sync work in left navigation bar

Important - *Stop* session when finished to save on costs

Closing the browser tab does *not* stop a workspace

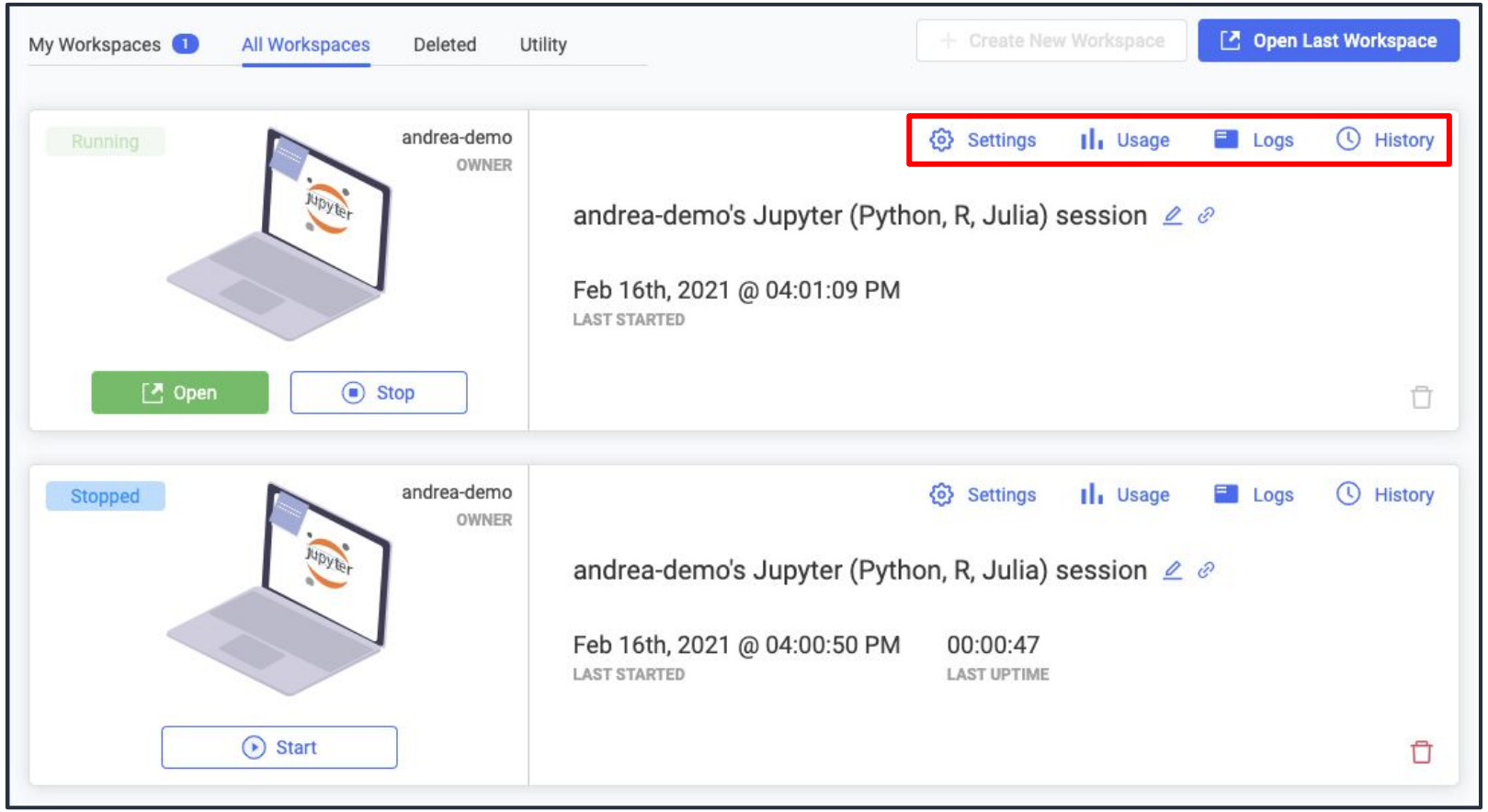
You can Stop workspaces without committing files



Workspace Dashboard

The workspaces dashboard will show all stopped and running workspaces

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



DOMINO

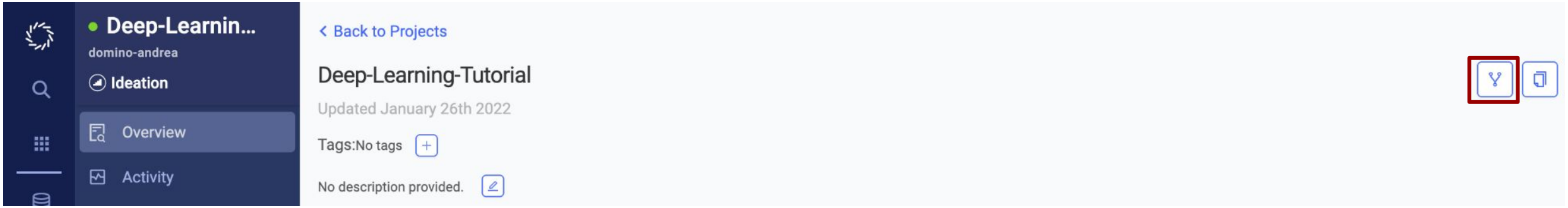
DEMO

Domino Overview

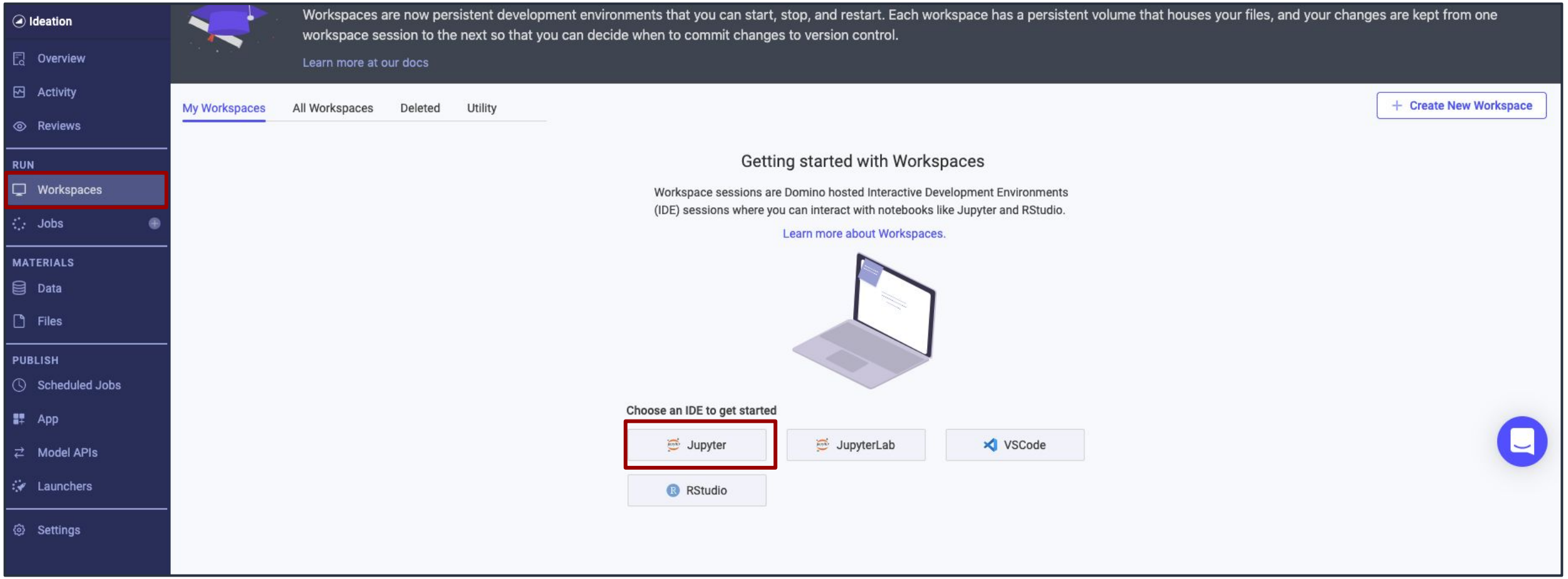
DOMINO

Access Materials

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

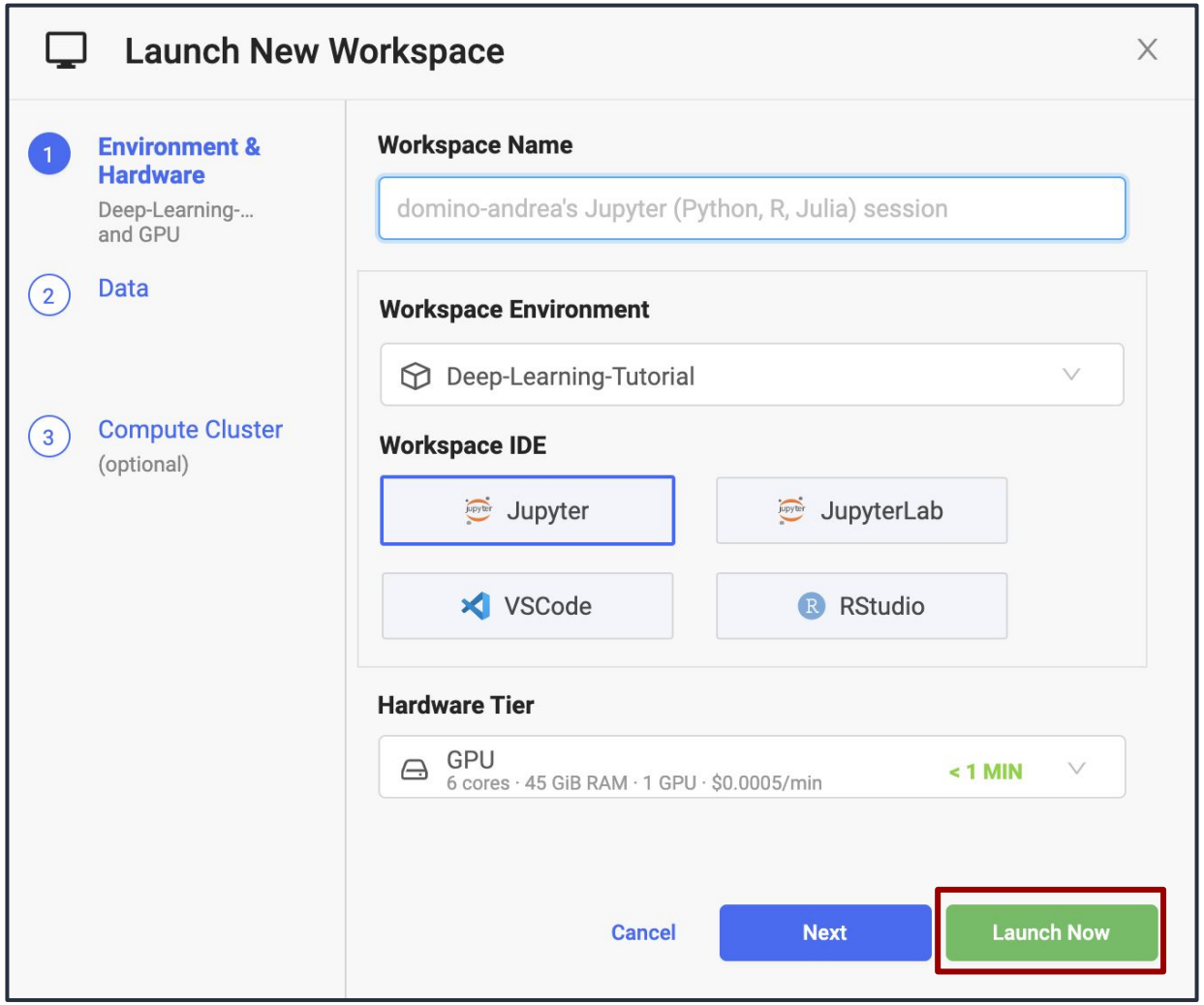


Launch an Interactive Workspace



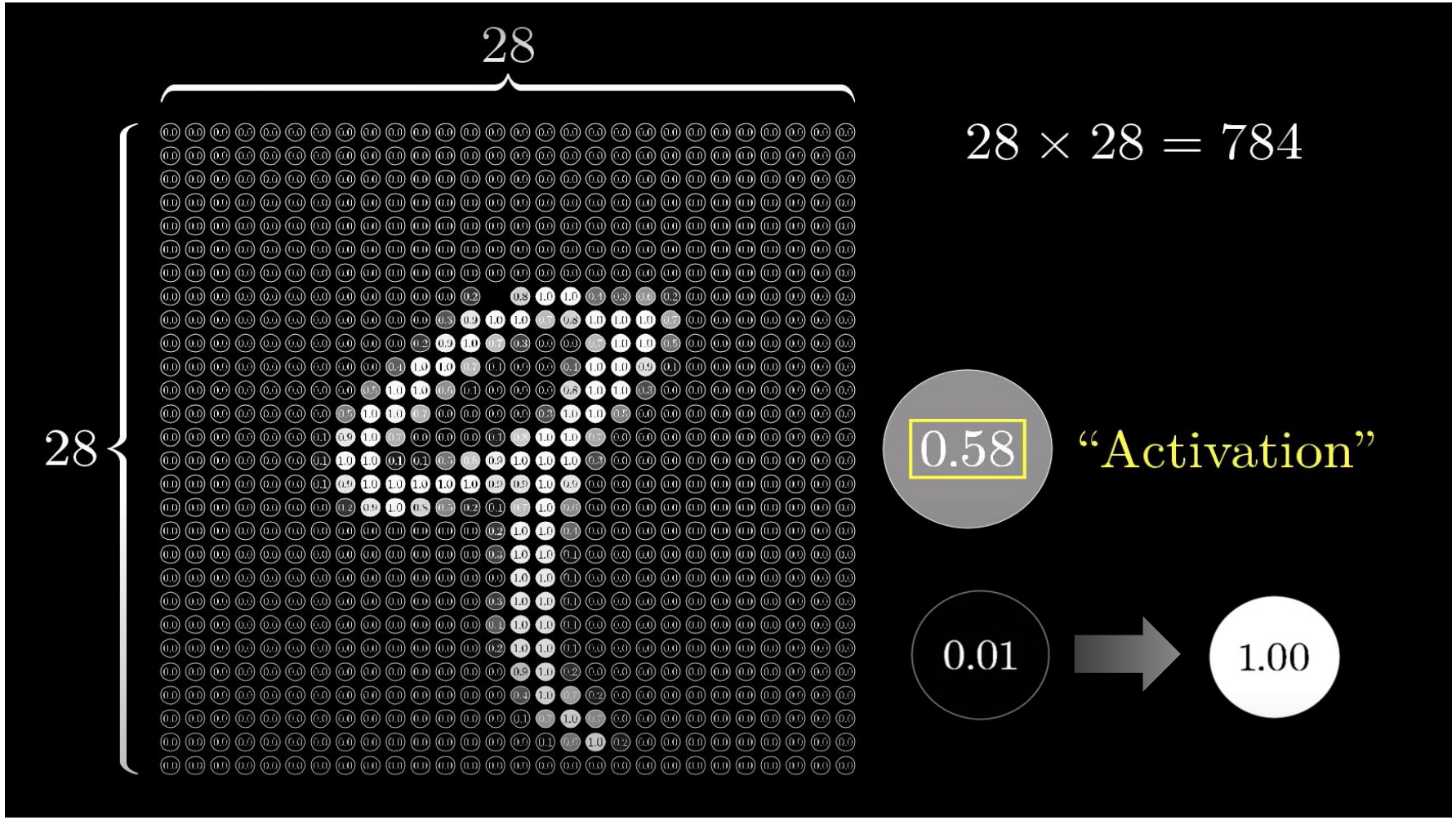
DOMINO

Launch an Interactive Workspace



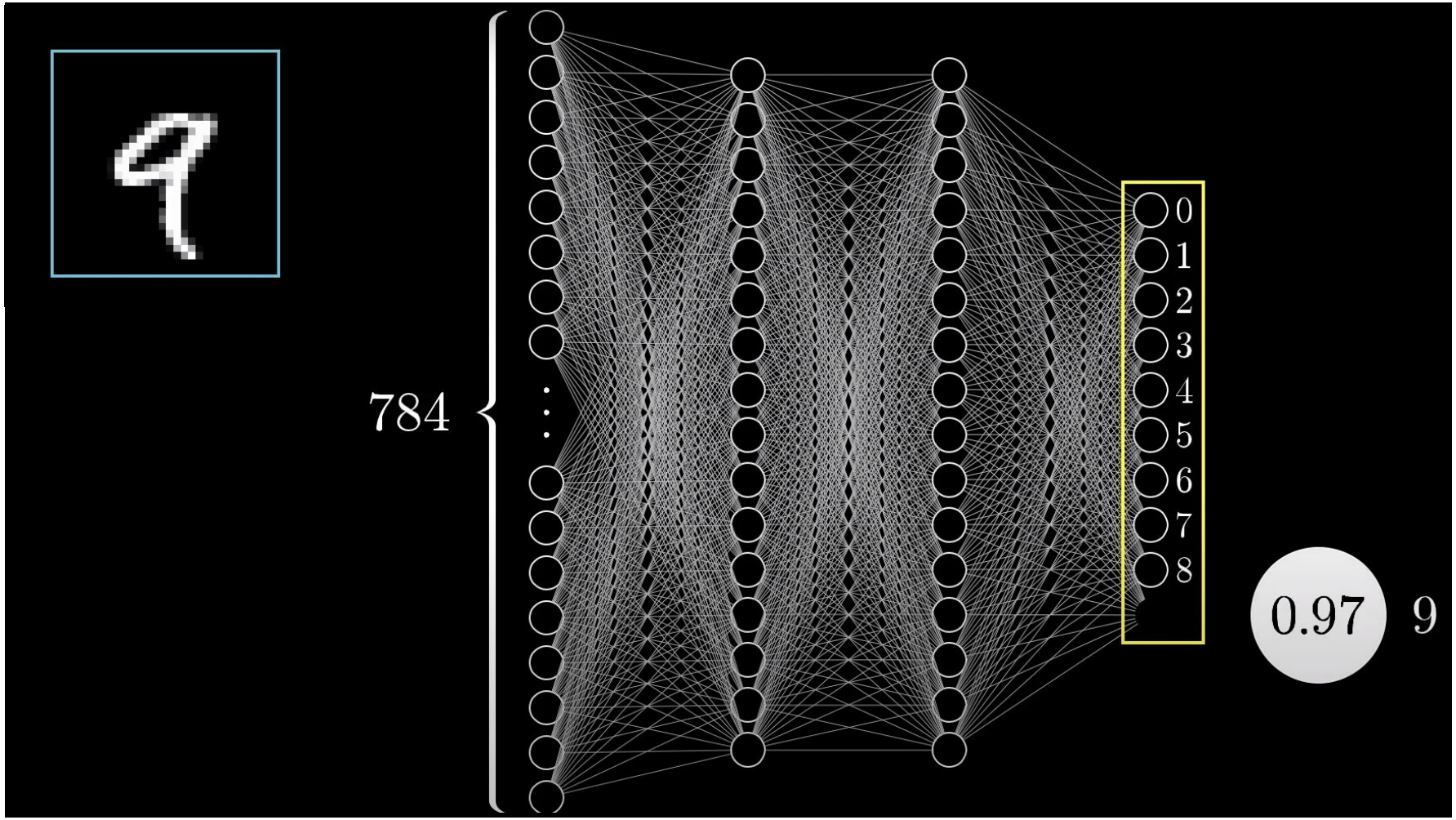
Intro to Neural Networks

WHAT IS A NEURAL NETWORK?



https://www.3blue1brown.com/

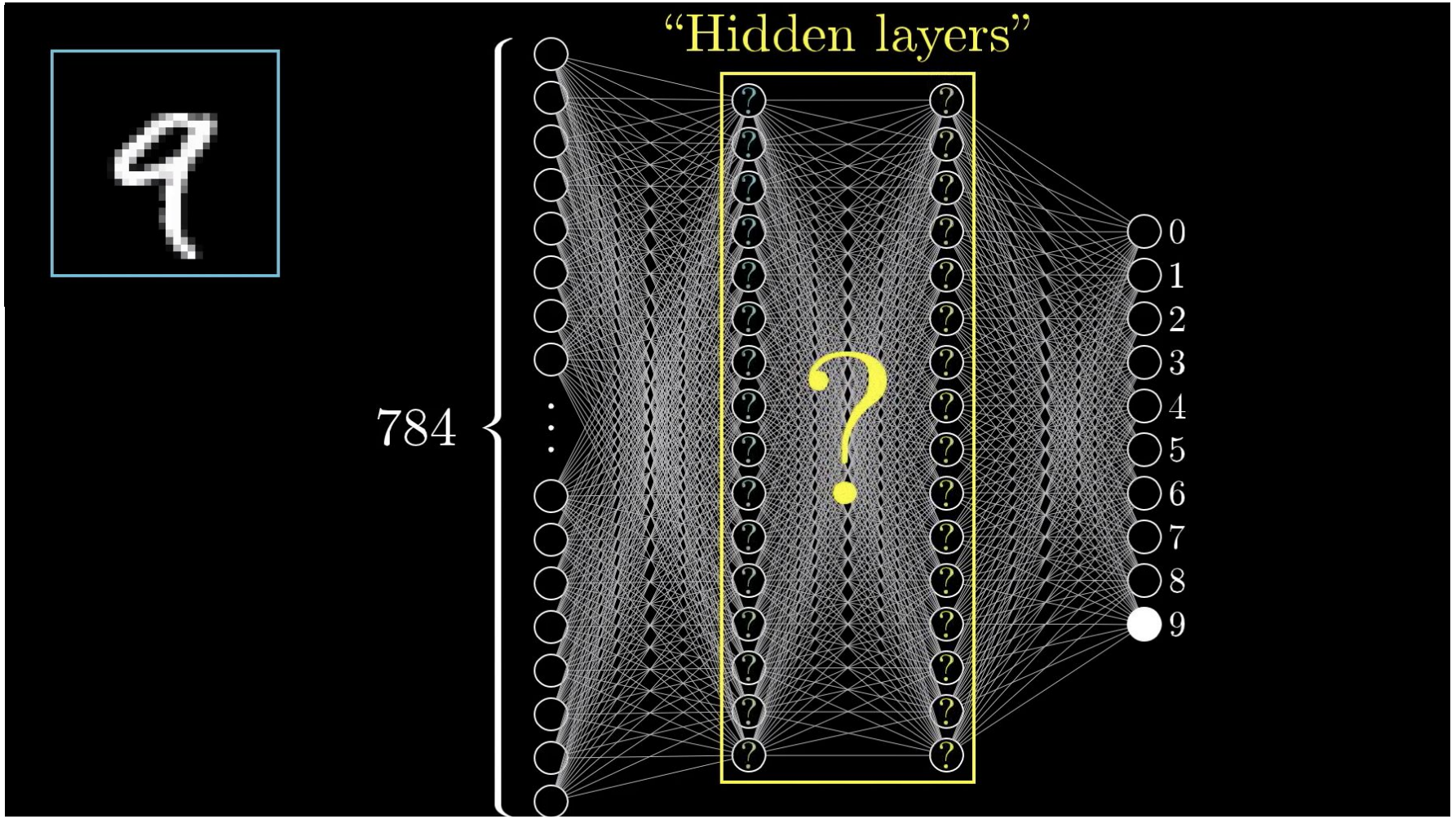
WHAT IS A NEURAL NETWORK?



https://www.3blue1brown.com/

DOMINO

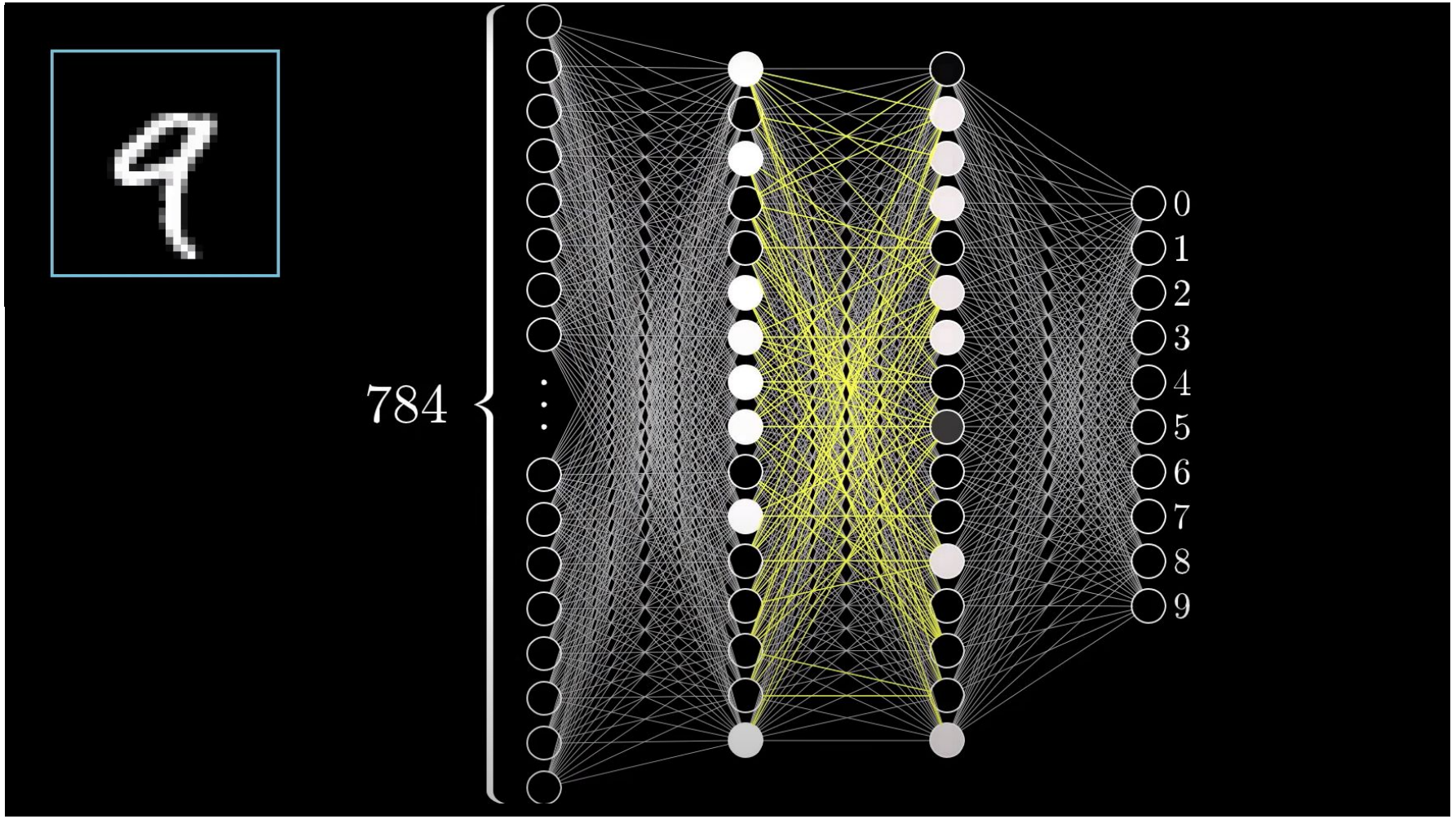
WHAT IS A NEURAL NETWORK?



https://www.3blue1brown.com/

DOMINO

WHAT IS A NEURAL NETWORK?



https://www.3blue1brown.com/

DOMINO

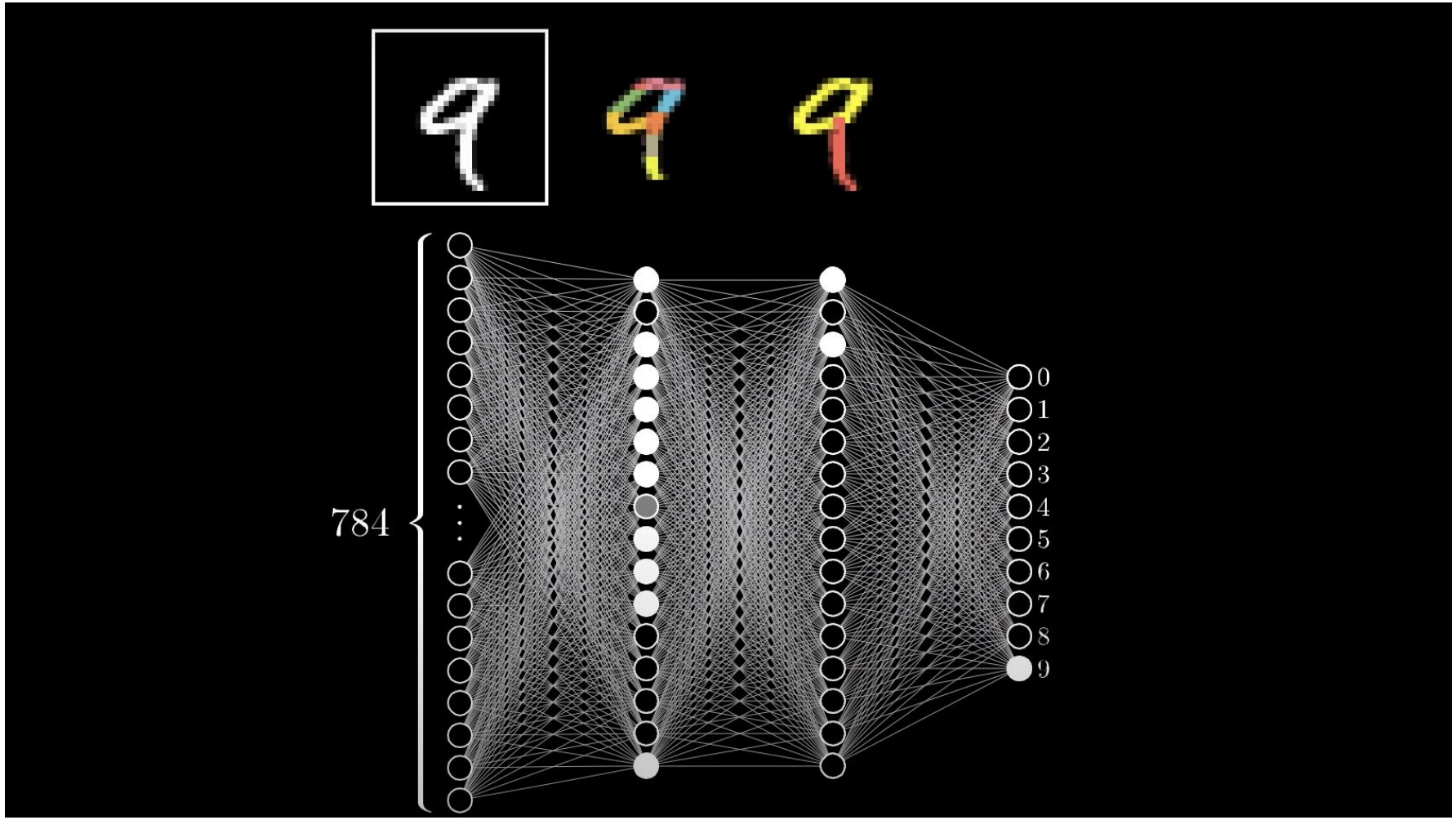
WHAT IS A NEURAL NETWORK?

- 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

https://www.3blue1brown.com/

WHAT IS A NEURAL NETWORK?



https://www.3blue1brown.com/

DOMINO

IMAGE RECOGNITION

https://www.3blue1brown.com/

DOMINO

SPEECH RECOGNITION

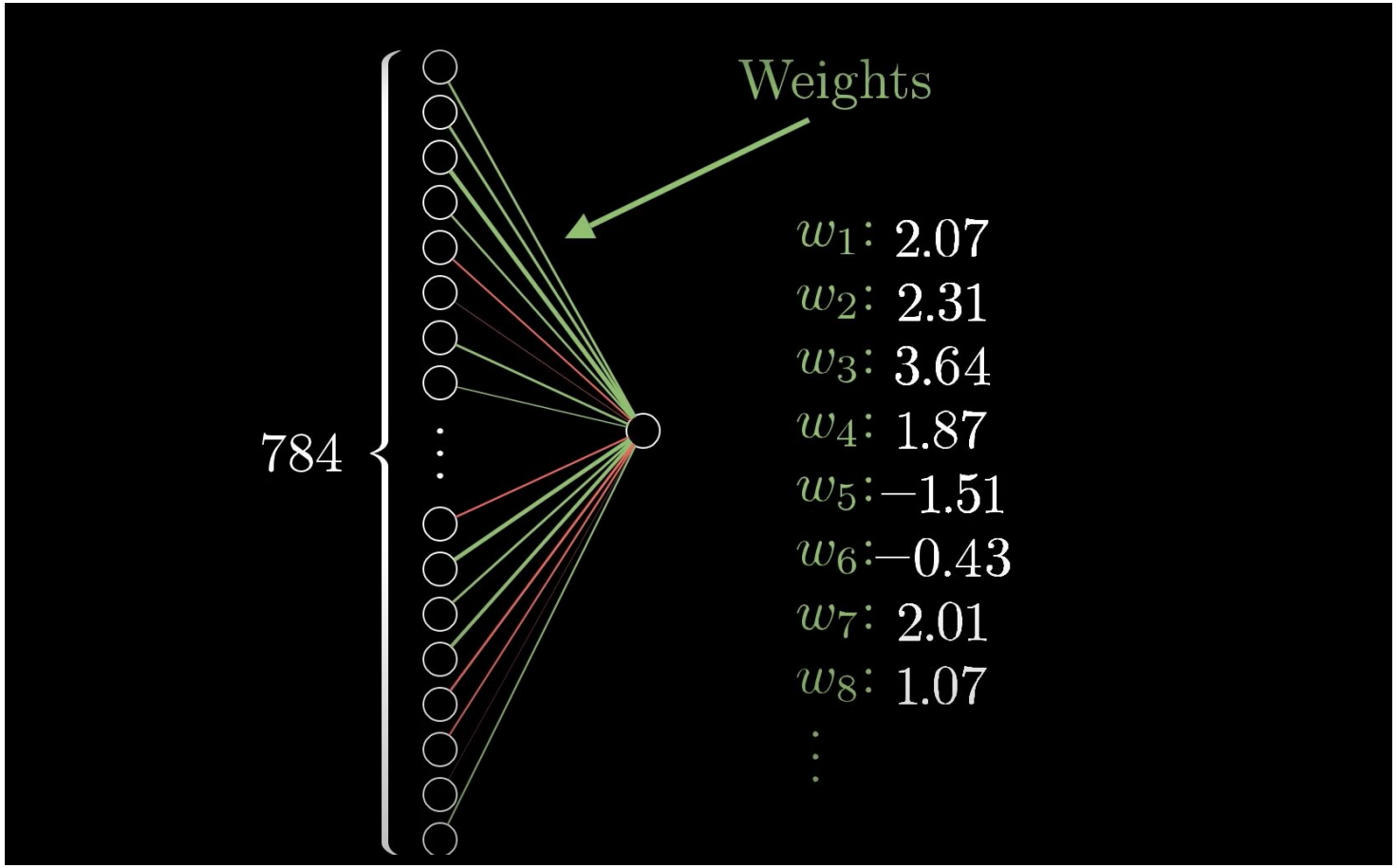
Parse raw audio → Break into distinct sounds →

Form syllables → Combine into words →

Phrases → etc

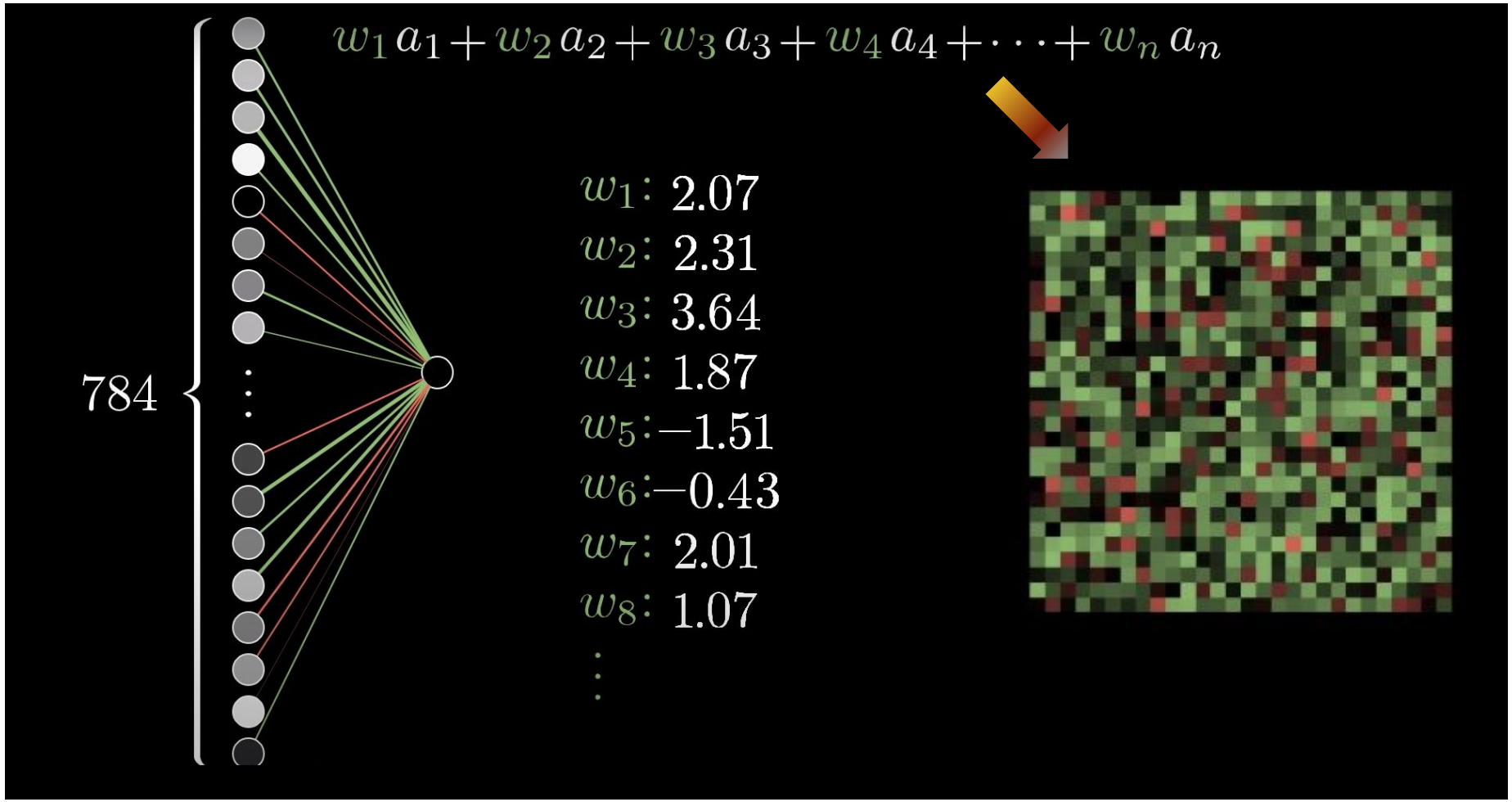
NEURAL NETWORK WEIGHTS

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



https://www.3blue1brown.com/

NEURAL NETWORK WEIGHTS



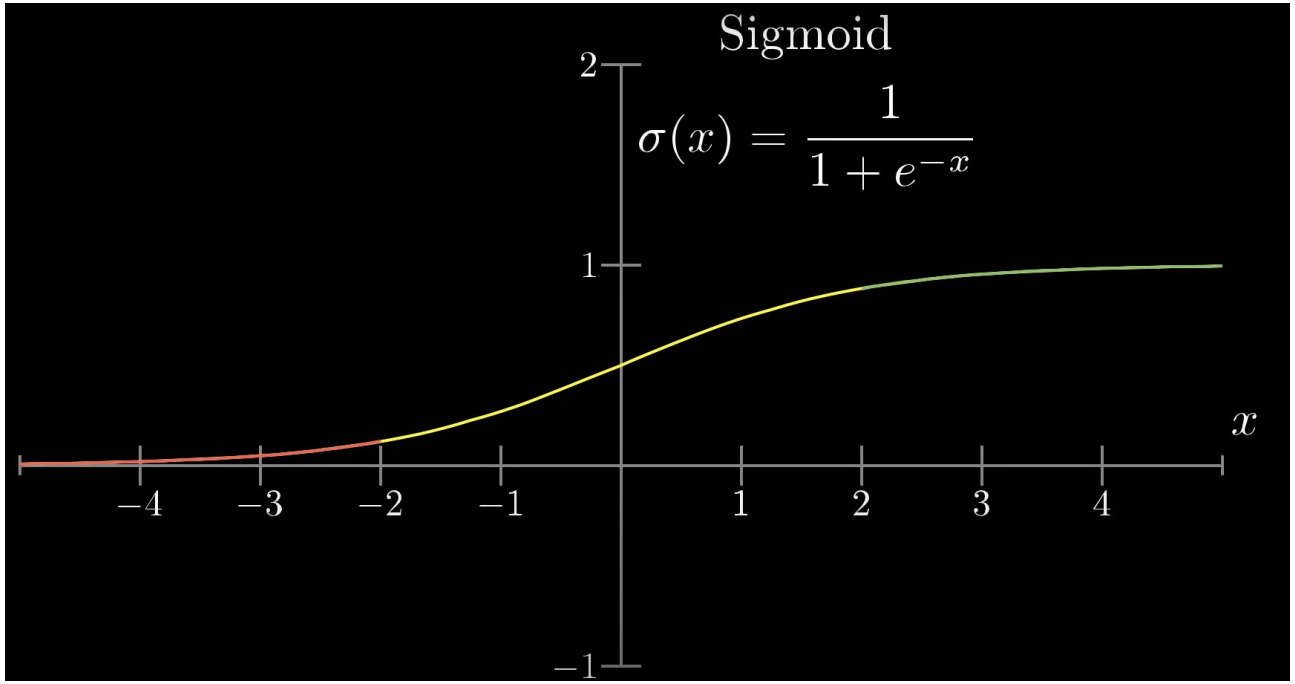
https://www.3blue1brown.com/

NEURAL NETWORK WEIGHTS

Each of those sums could be any number

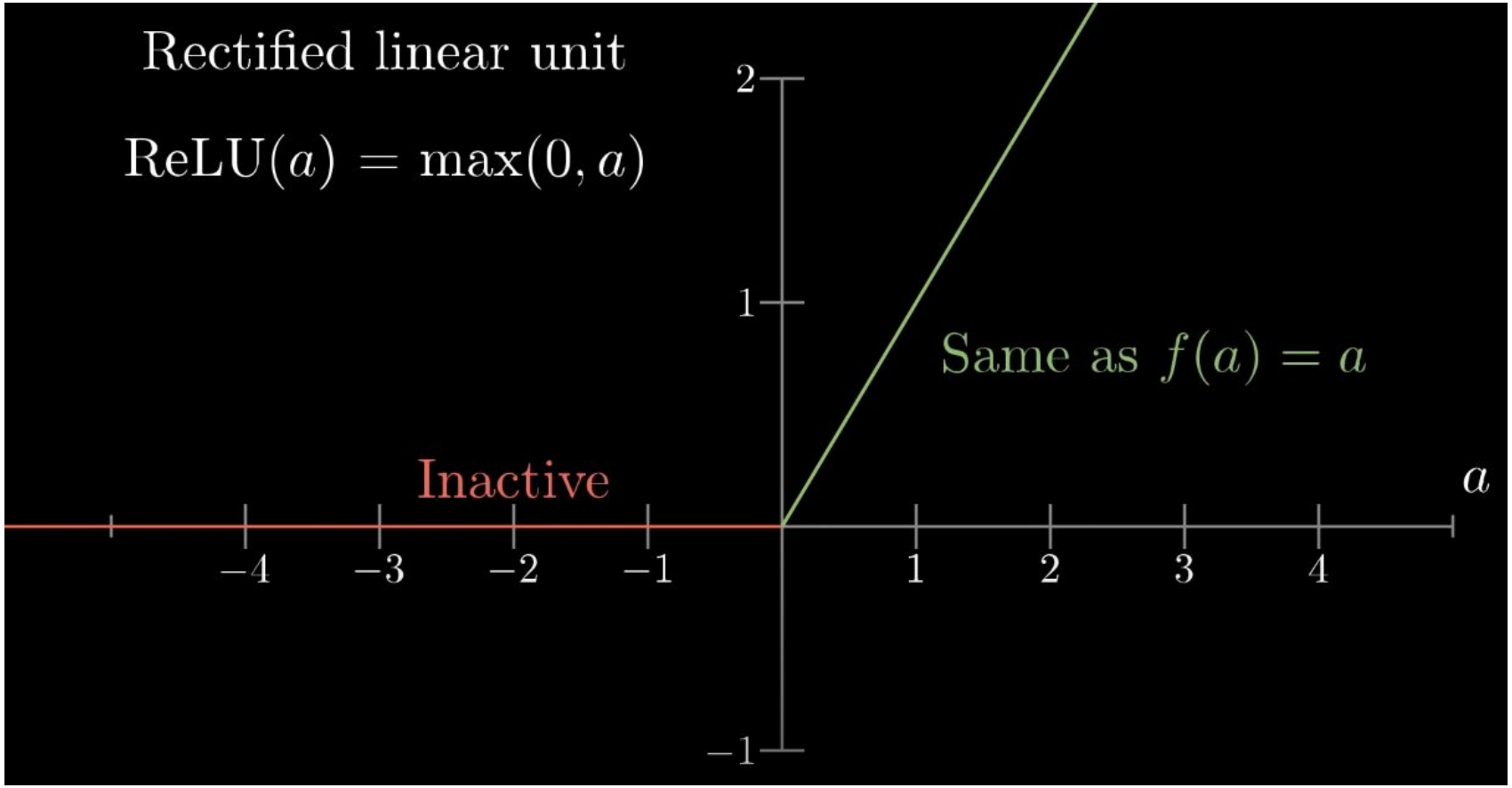
To ensure we get an activation between 0 and 1 for the next layer we put those sums through a function

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



https://www.3blue1brown.com/

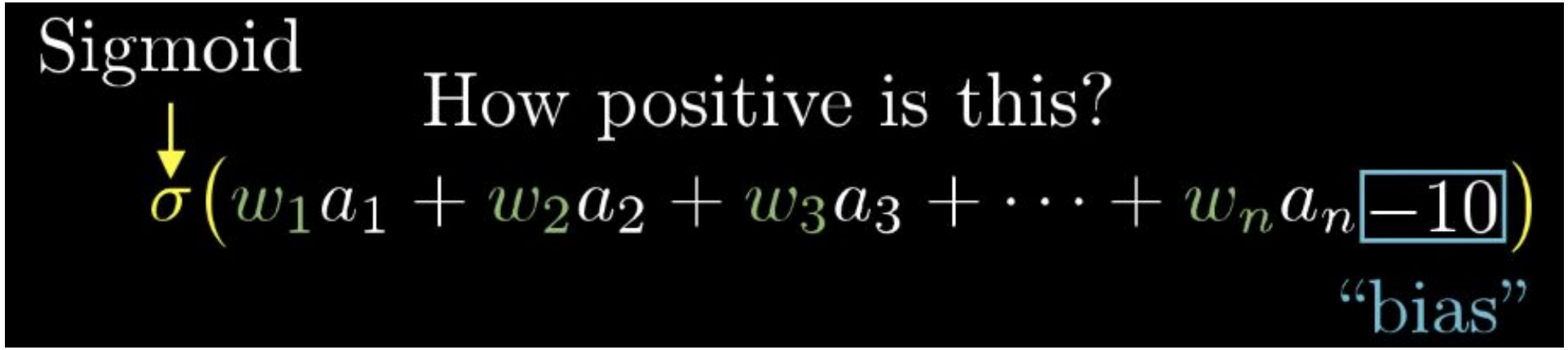
NEURAL NETWORK WEIGHTS



https://www.3blue1brown.com/

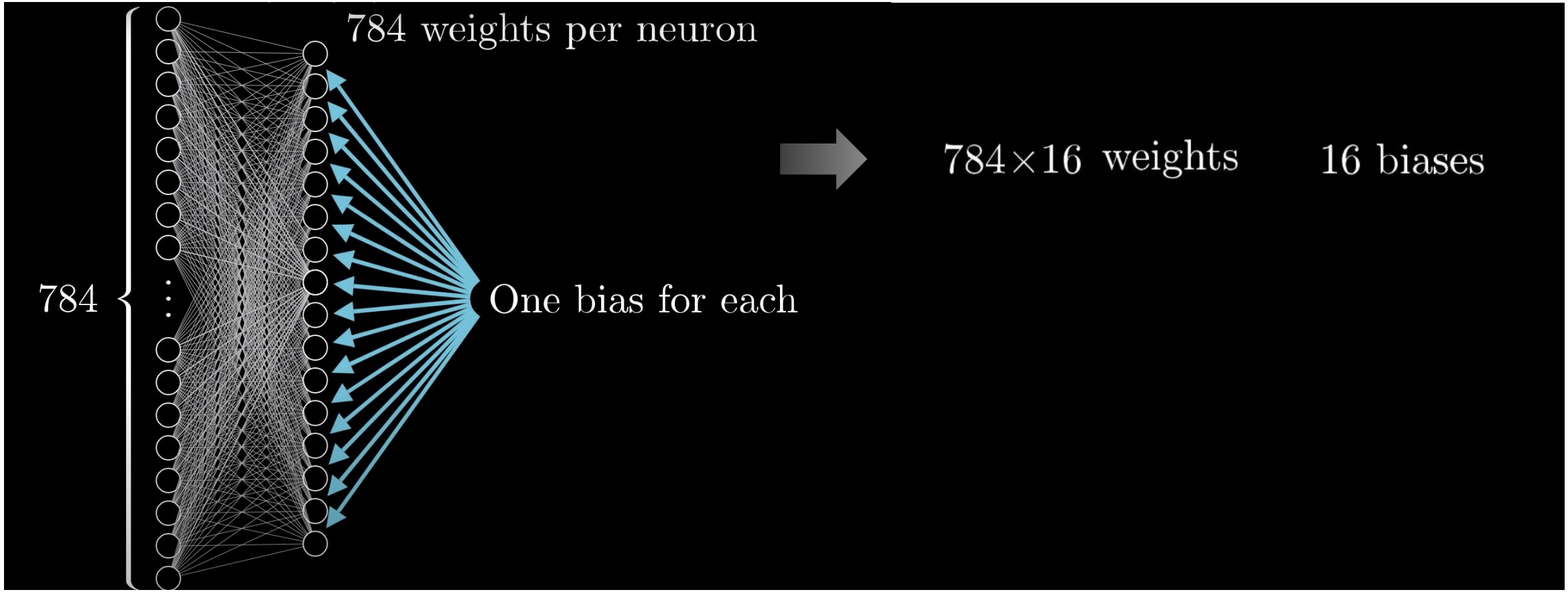
NEURAL NETWORK BIAS

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



https://www.3blue1brown.com/

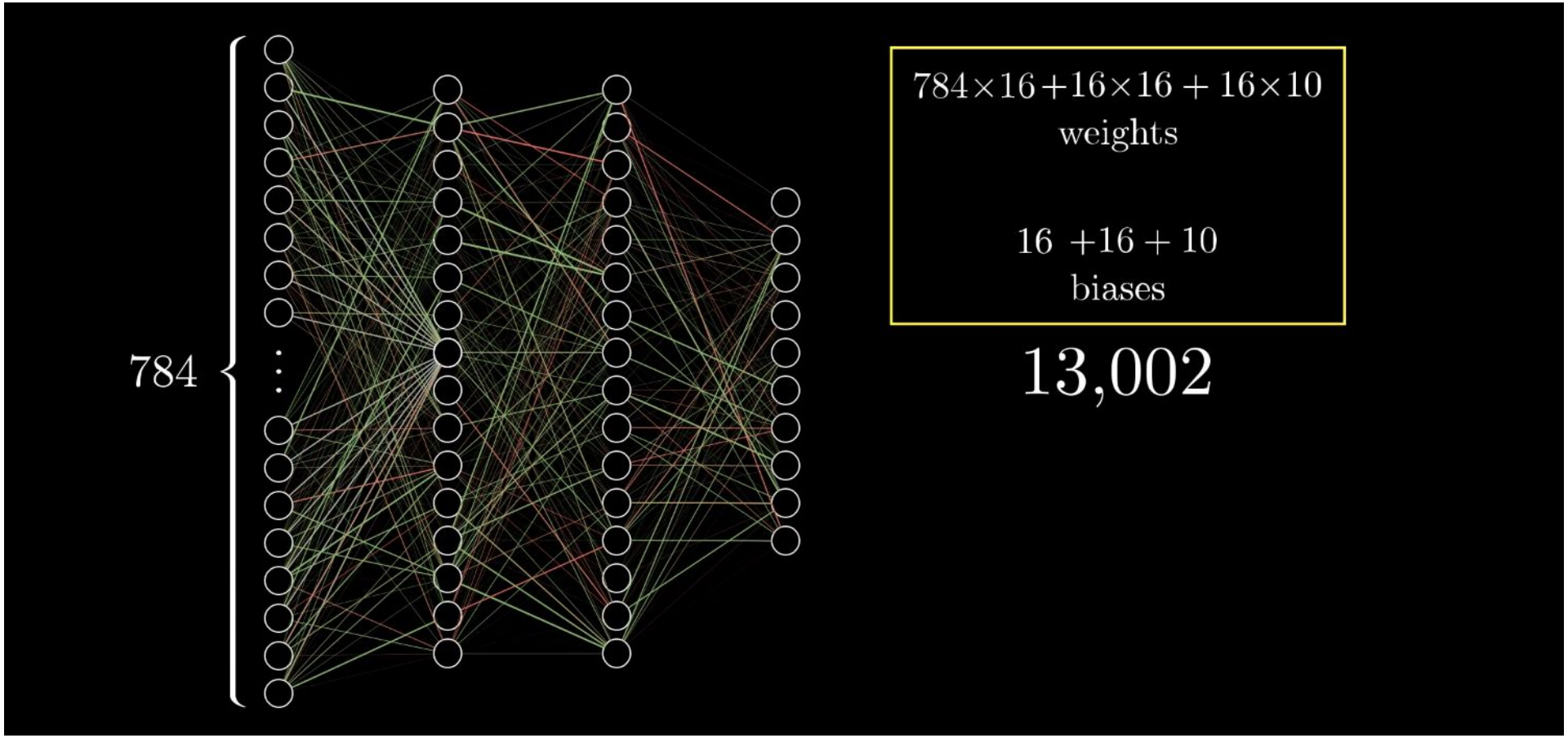
NEURAL NETWORK LEARNING



https://www.3blue1brown.com/

DOMINO

NEURAL NETWORK LEARNING



https://www.3blue1brown.com/

DOMINO

NEURAL NETWORK LEARNING

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

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

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

https://www.3blue1brown.com/

DOMINO

NEURAL NETWORK LEARNING

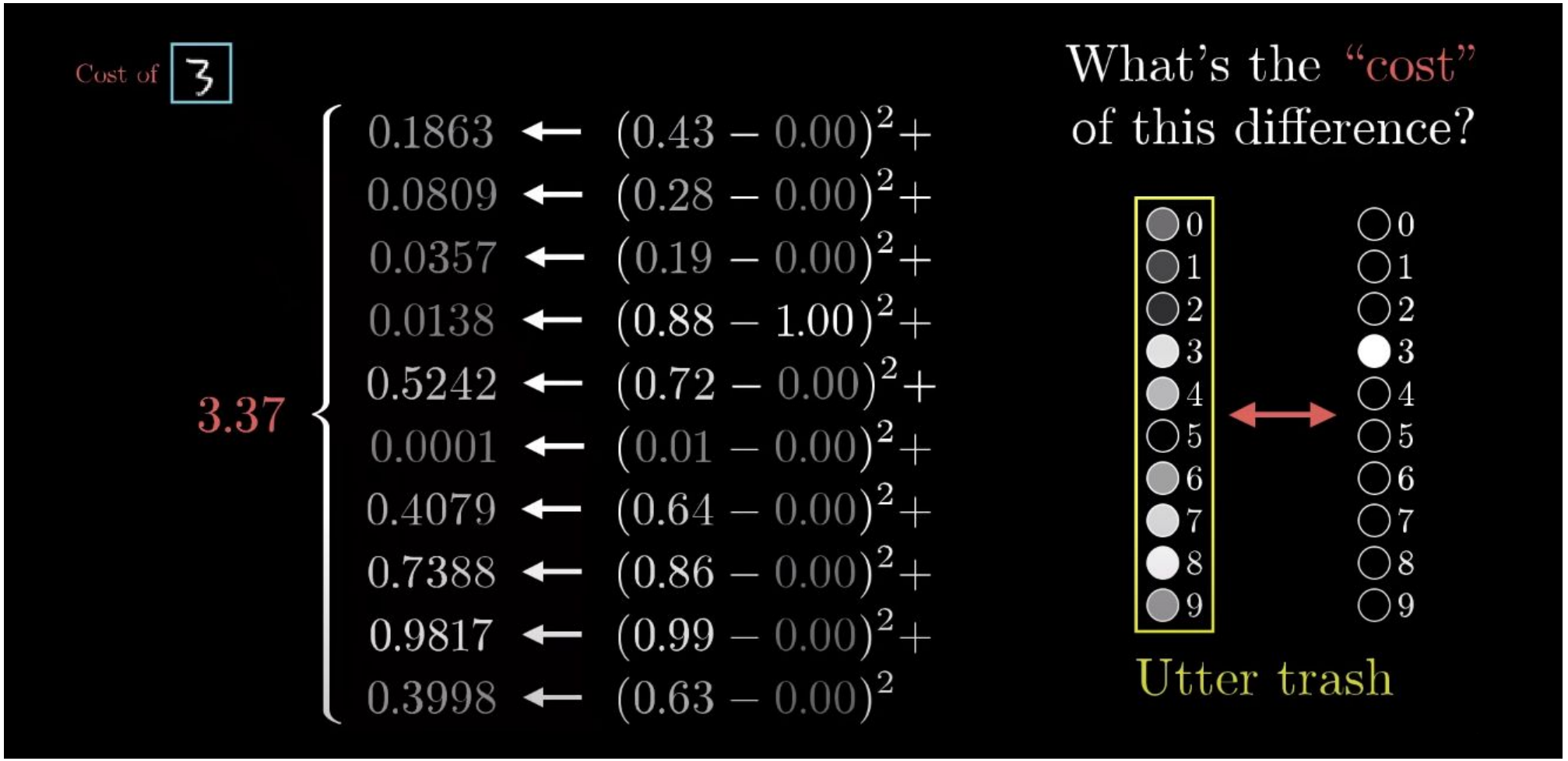
- In this example, each neuron is connected to all the neurons in the previous layer:
  - The *weight* describes the strength of each of those connections
  - The *bias* describes whether the neuron tends to be active or not
- These form a function for each neuron which is transformed into a value between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

https://www.3blue1brown.com/

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?



https://www.3blue1brown.com/

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

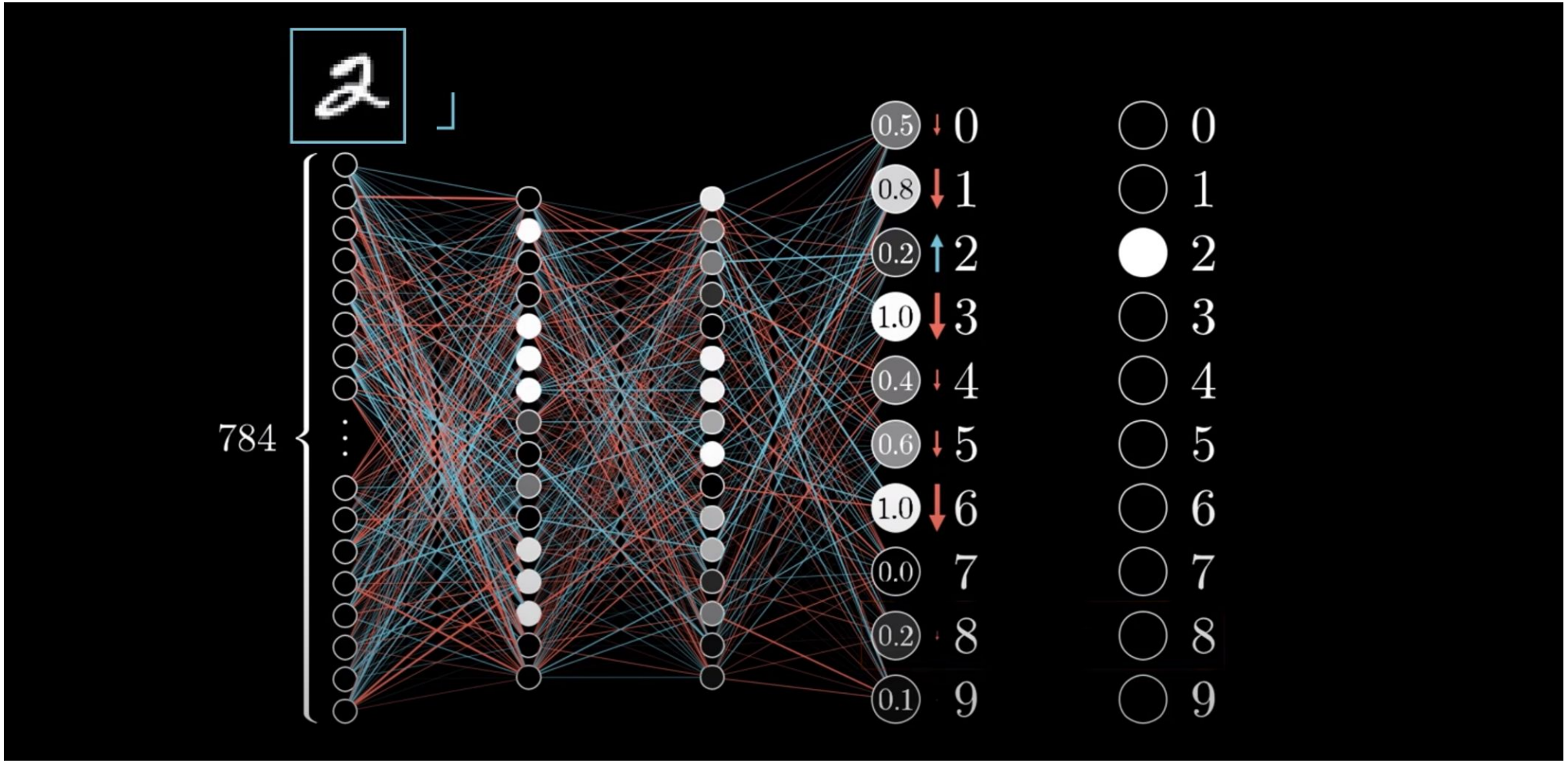
$w_{13,000}$ should increase a lot
$w_{13,001}$ should decrease somewhat
$w_{13,002}$ should increase a little

https://www.3blue1brown.com/

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

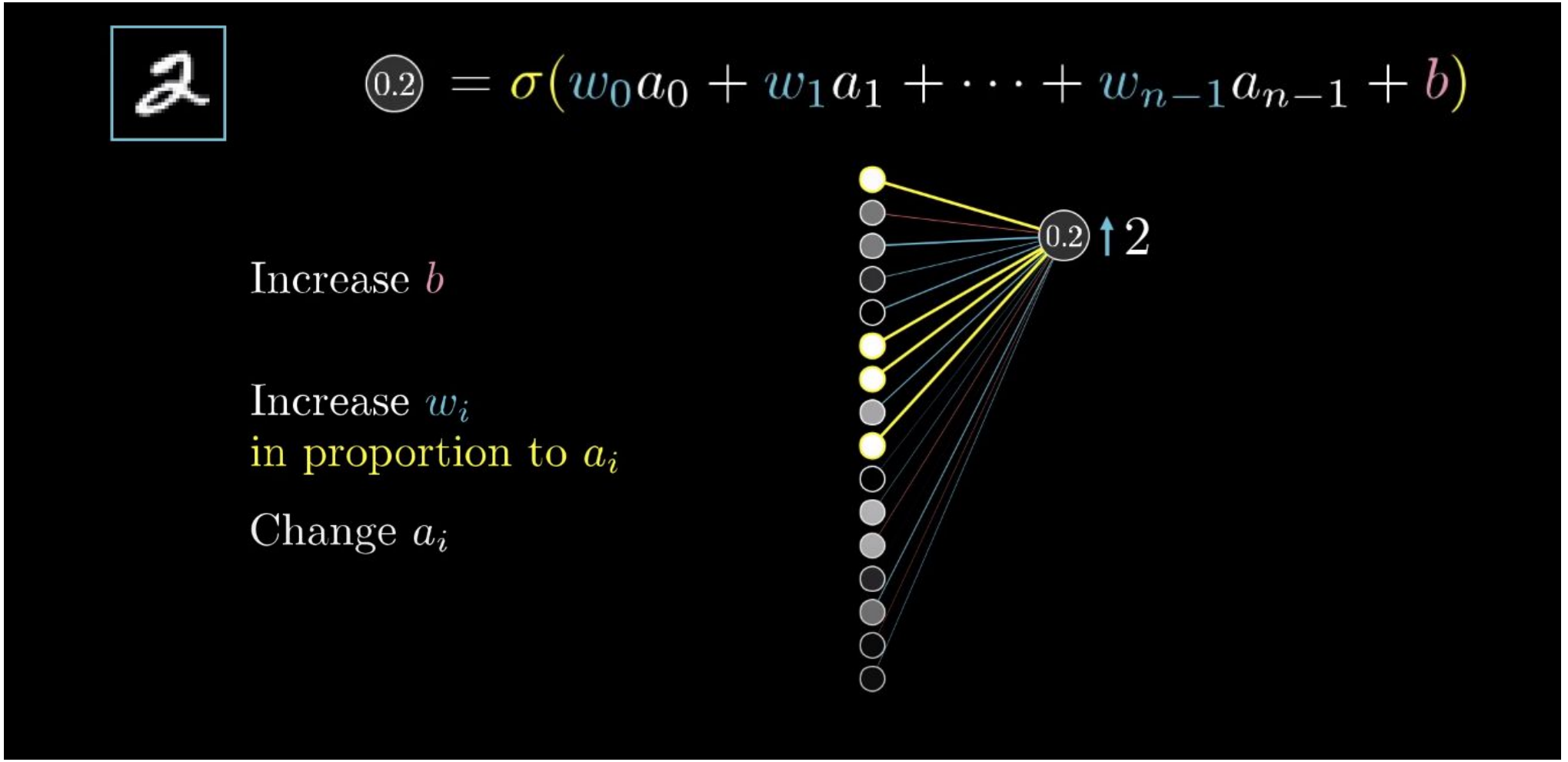
BACKPROPAGATION



https://www.3blue1brown.com/

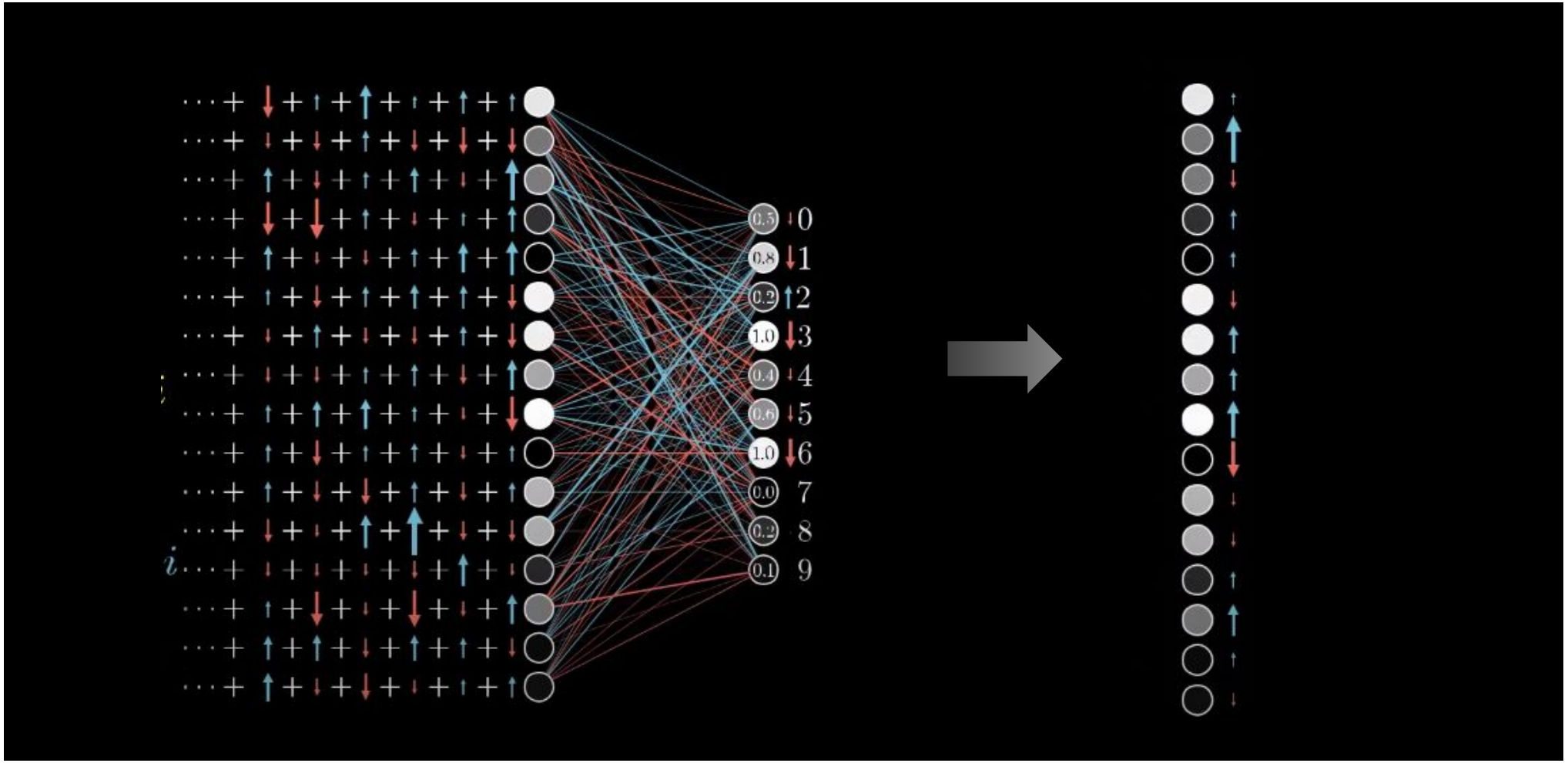
DOMINO

BACKPROPAGATION



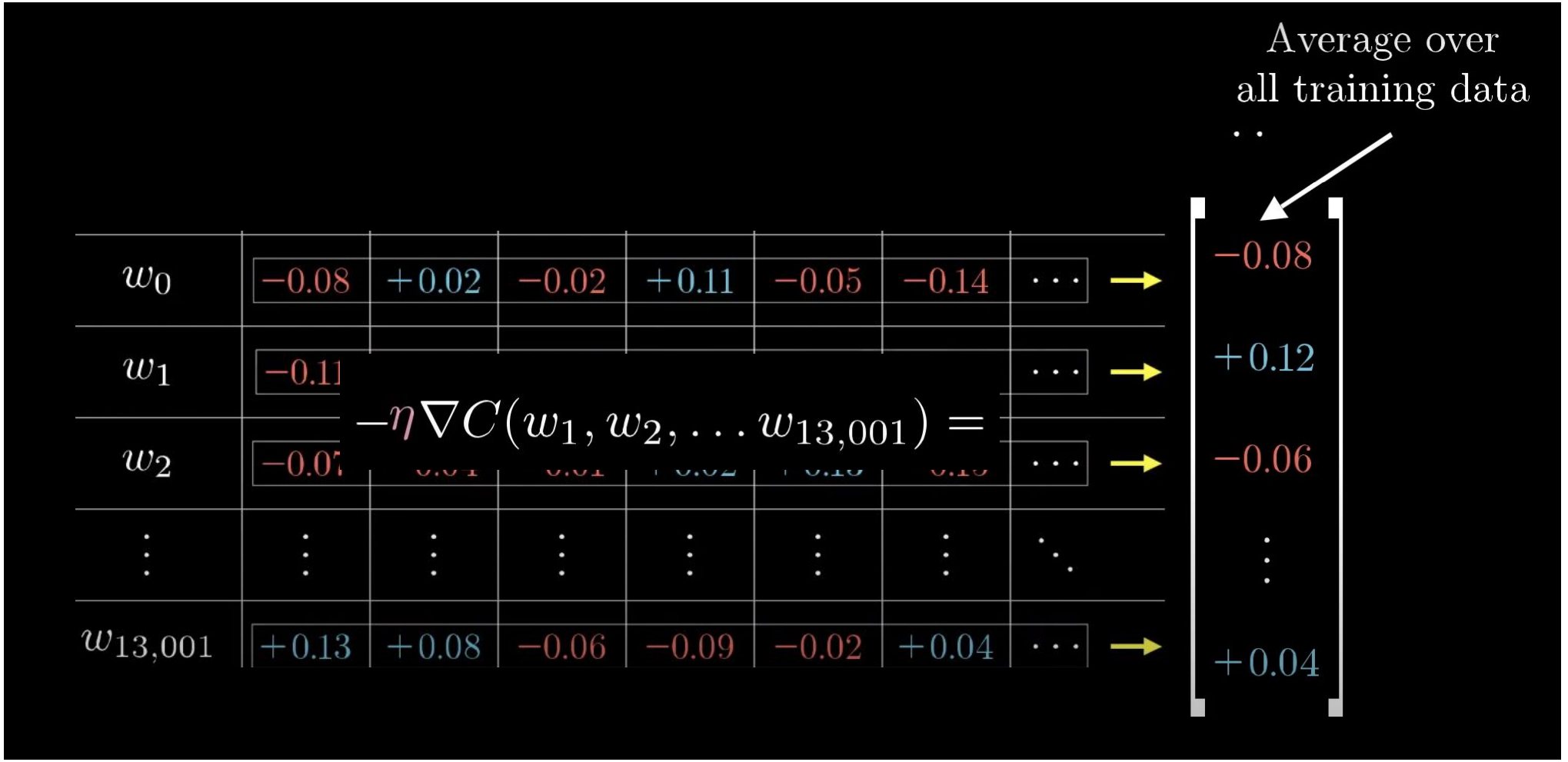
https://www.3blue1brown.com/

BACKPROPAGATION



https://www.3blue1brown.com/

GRADIENT DESCENT



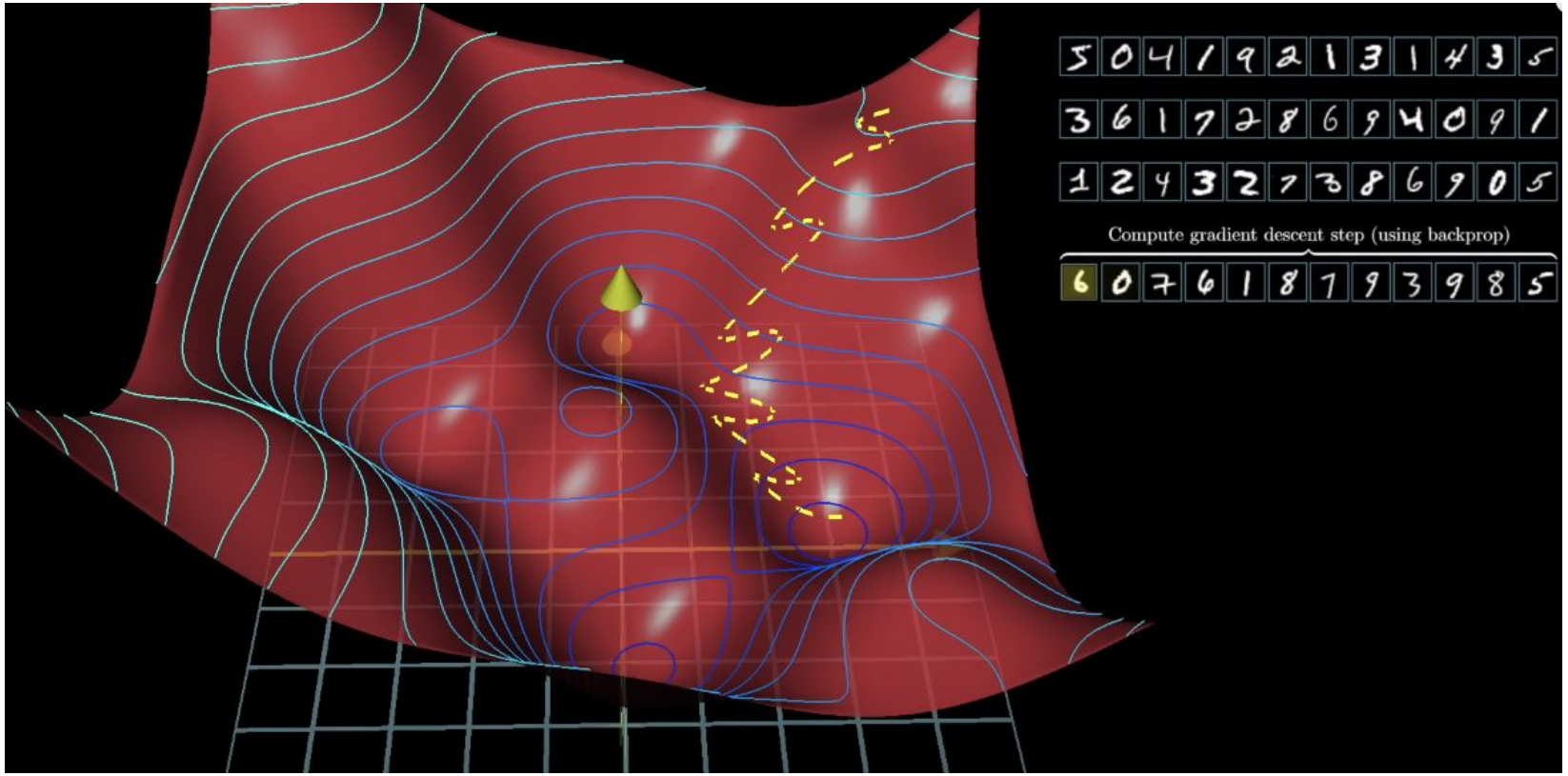
https://www.3blue1brown.com/

DOMINO

STOCHASTIC GRADIENT DESCENT

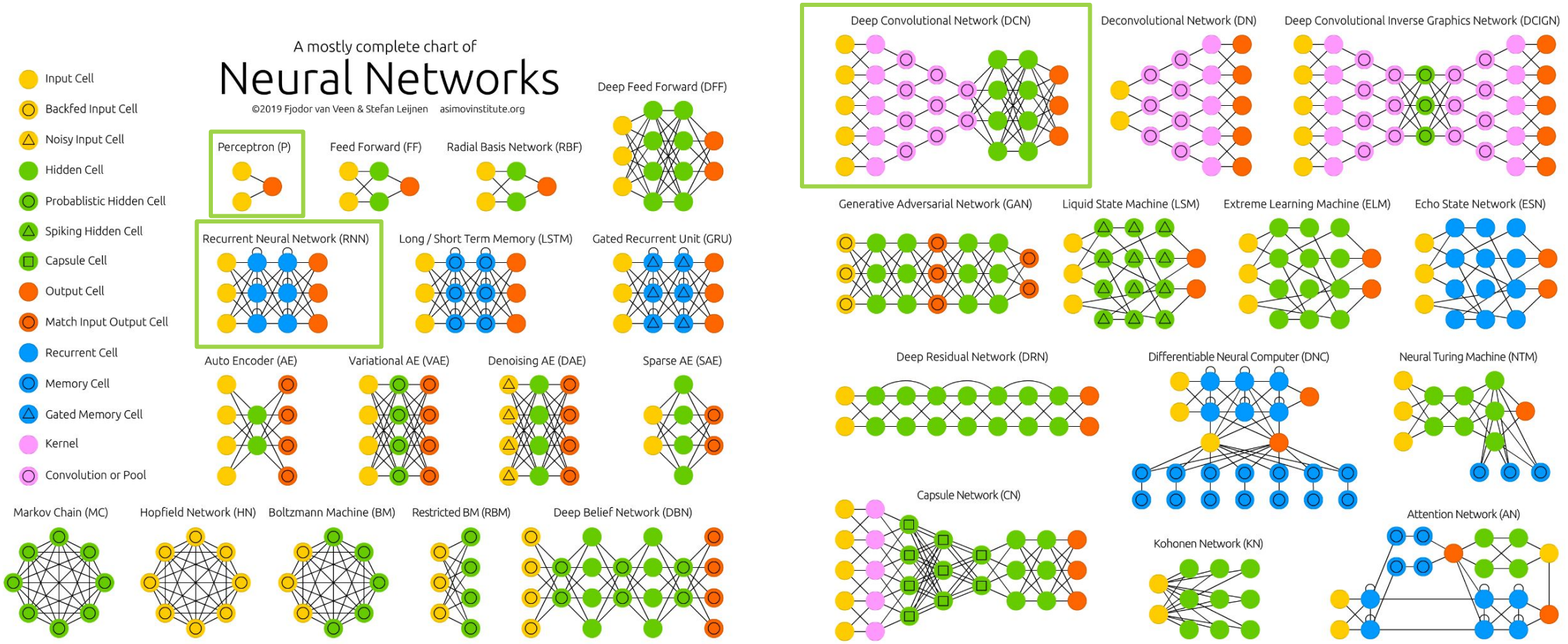
Much faster to calculate

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/

DOMINO

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

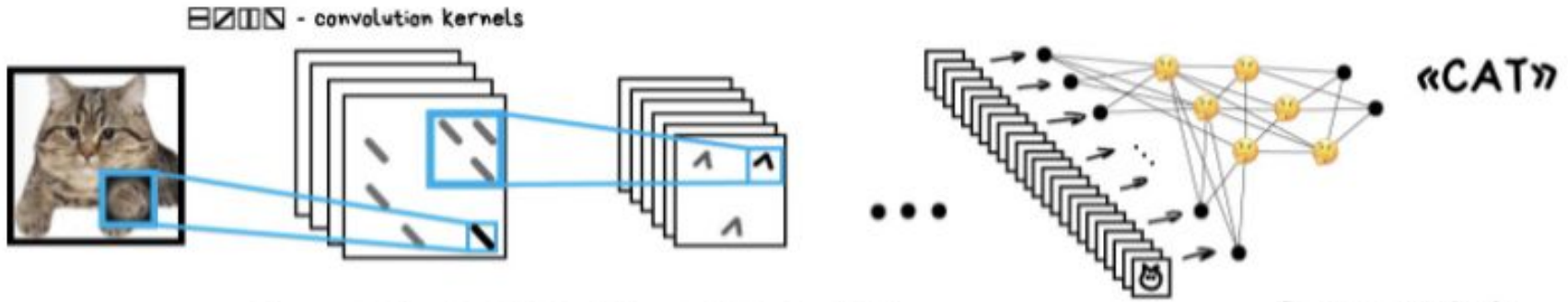
https://www.3blue1brown.com/

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



https://vas3k.com/blog/machine_learning/

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

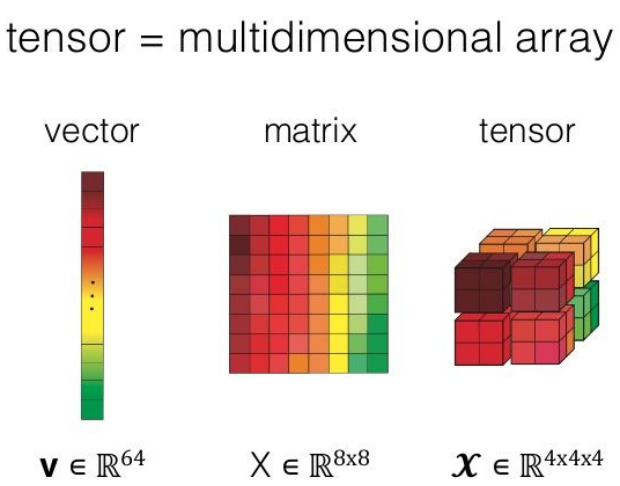
- Theano
- Keras
- TensorFlow
- PyTorch
- MXNet, Lasagne

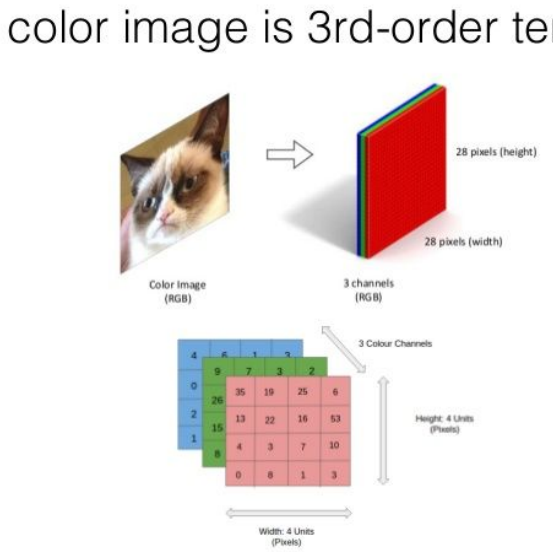
PYTORCH OVERVIEW

- 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





https://www.slideshare.net/BertonEarnshaw/a-brief-survey-of-tensors

DOMINO

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

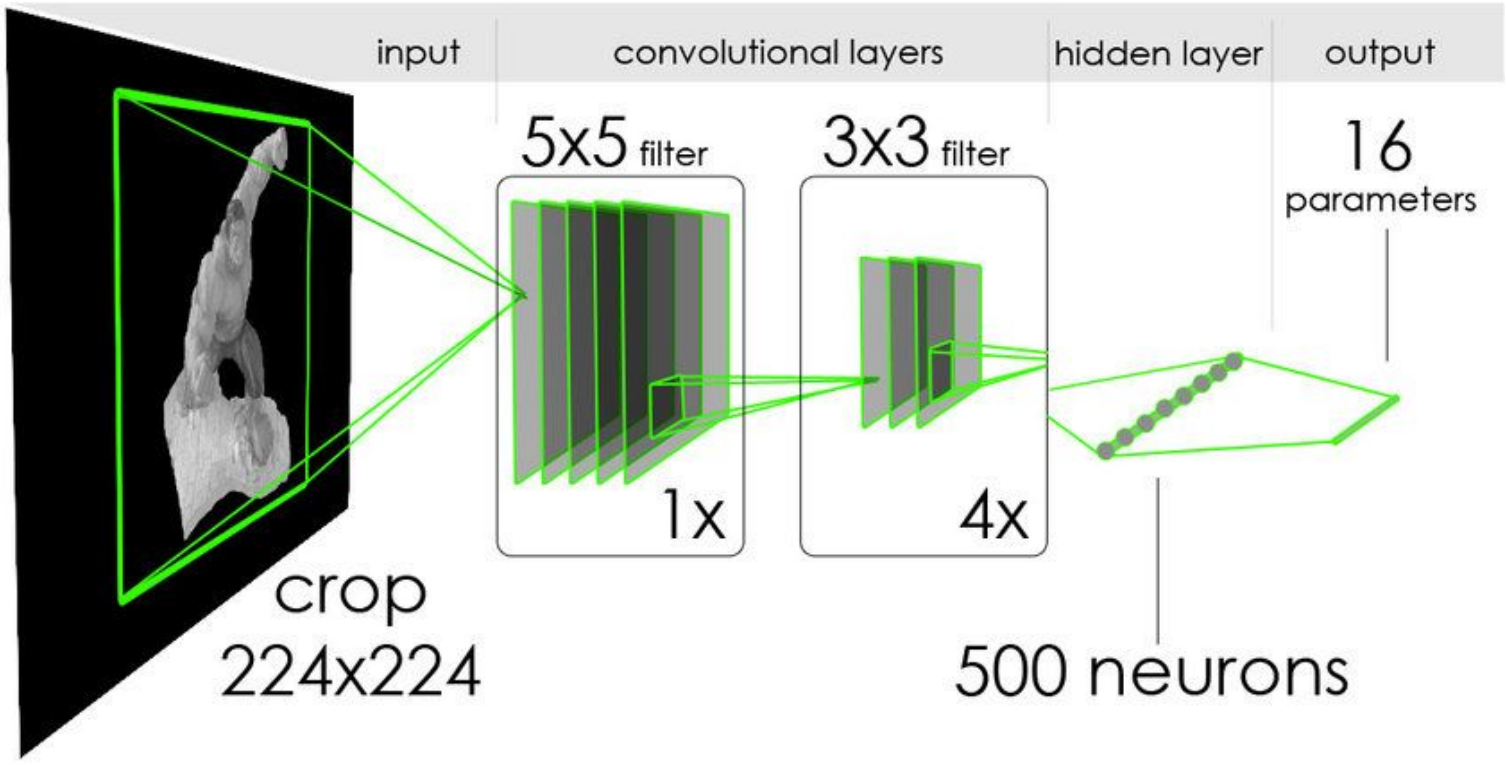
1. Open notebook ‘1-Load-Data-PyTorch.ipynb’
2. Try different transforms to see how they change the images

Building the Model

INPUT AND OUTPUT LAYERS

Input layer - features of the image

Output layer - dataset labels



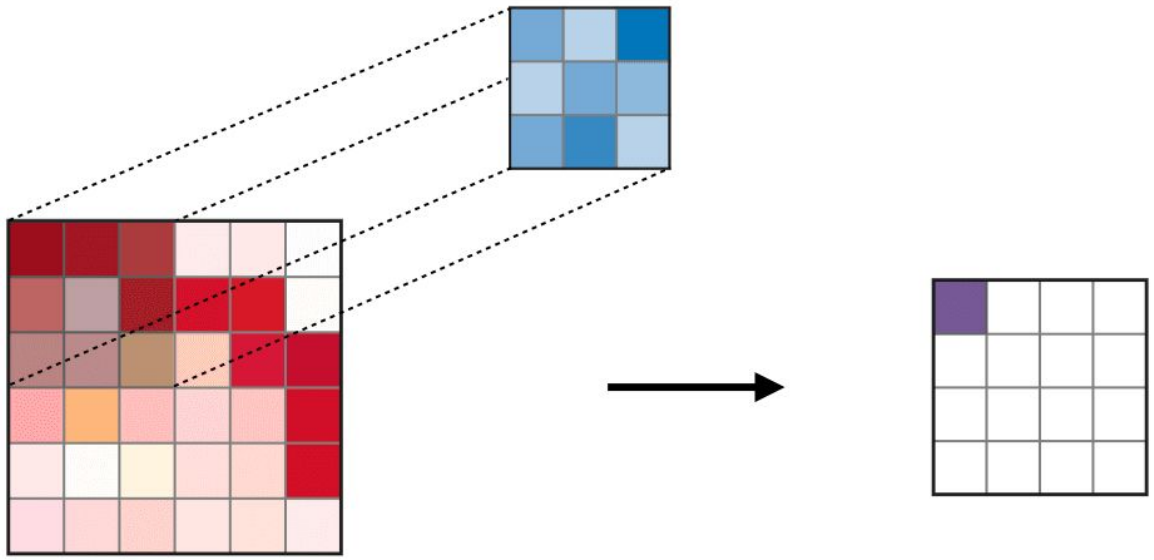
https://www.researchgate.net/publication/321260305_Learning_Lightprobes_for_Mixed_Reality_Illumination

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



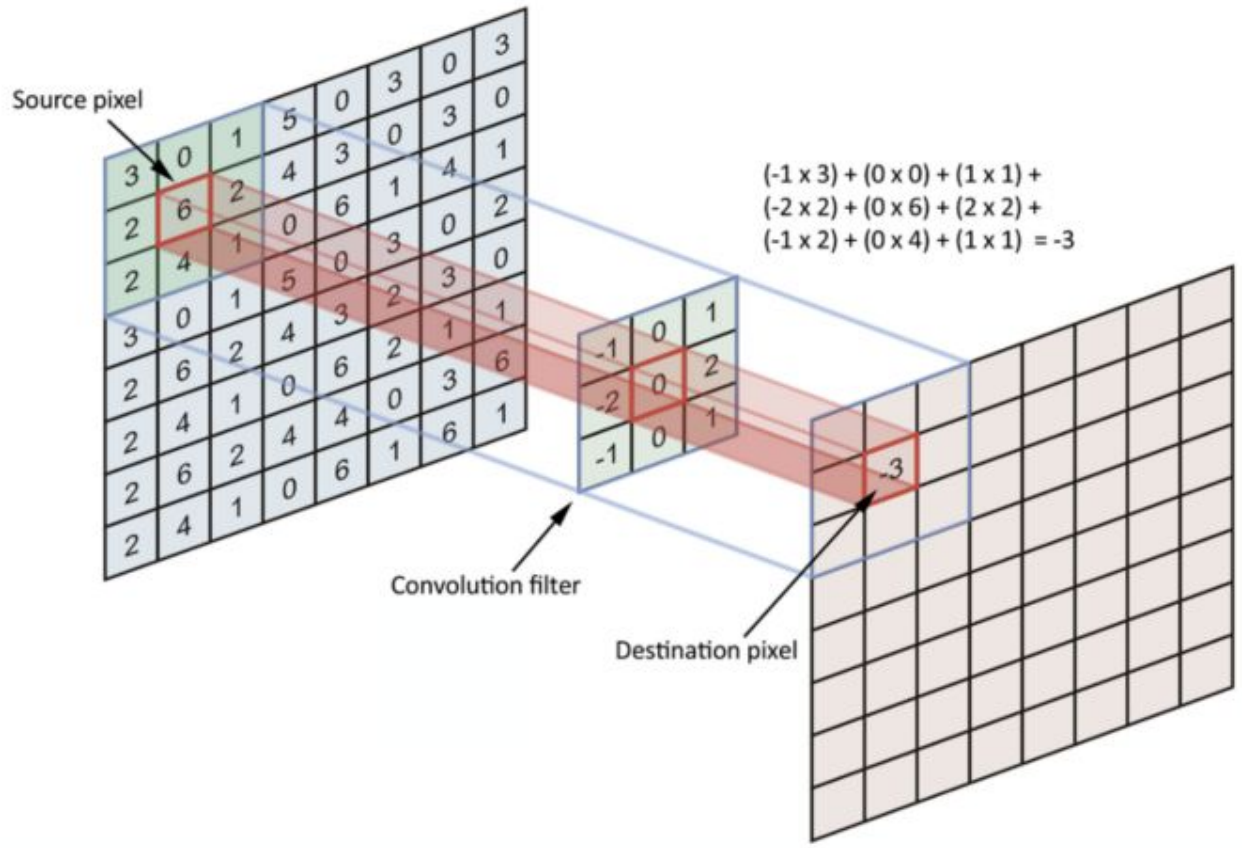
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

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

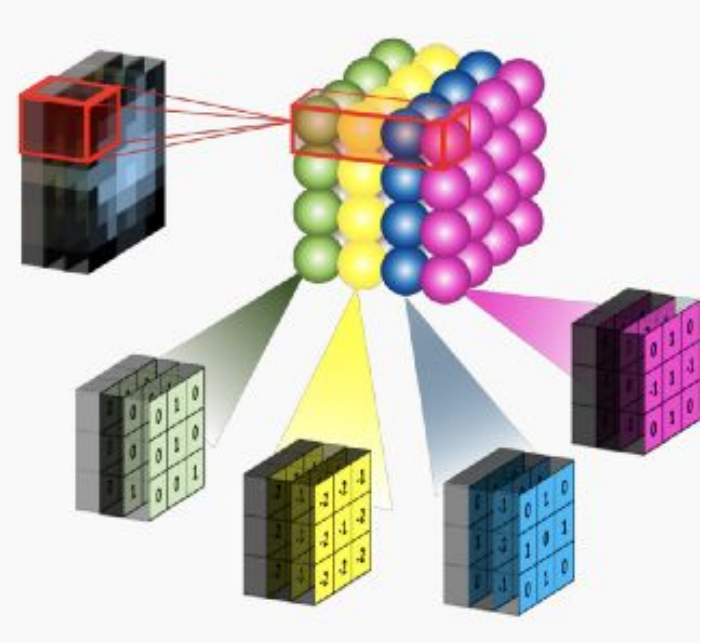
Summary:

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



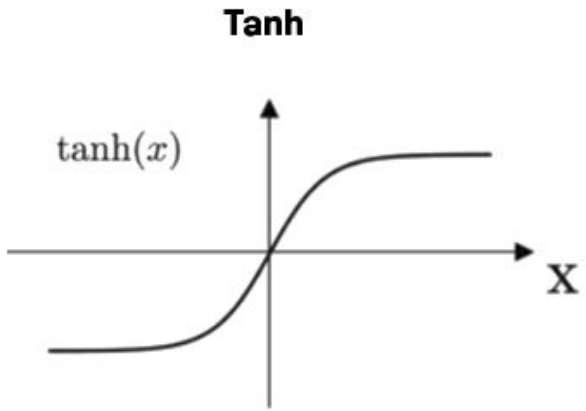
https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

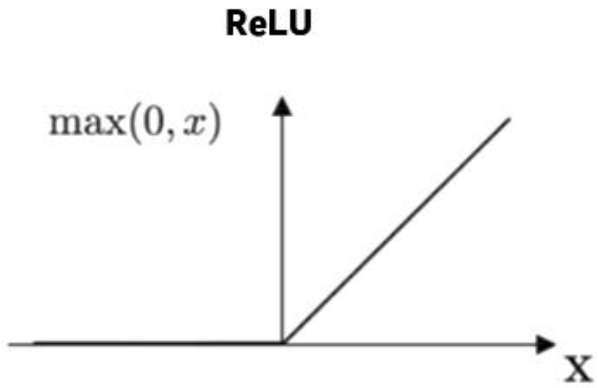
ACTIVATION FUNCTIONS

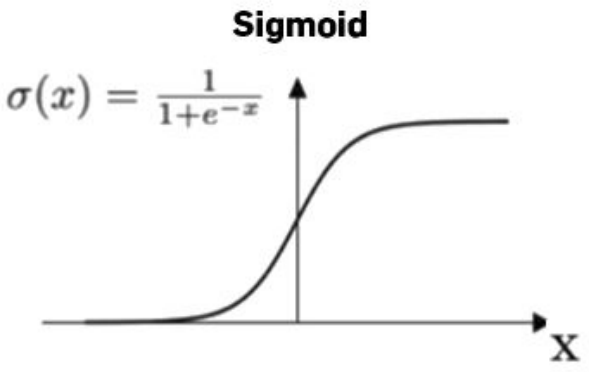
Adds non-linearity

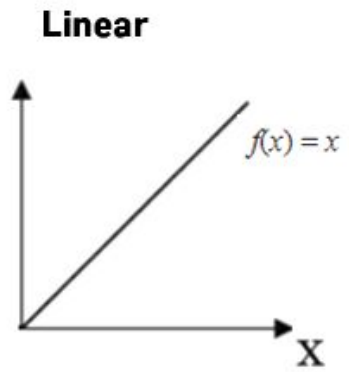
ReLU is typically a good start

- Cheap
- Stable









WHY DO WE NEED NON-LINEARITY?

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

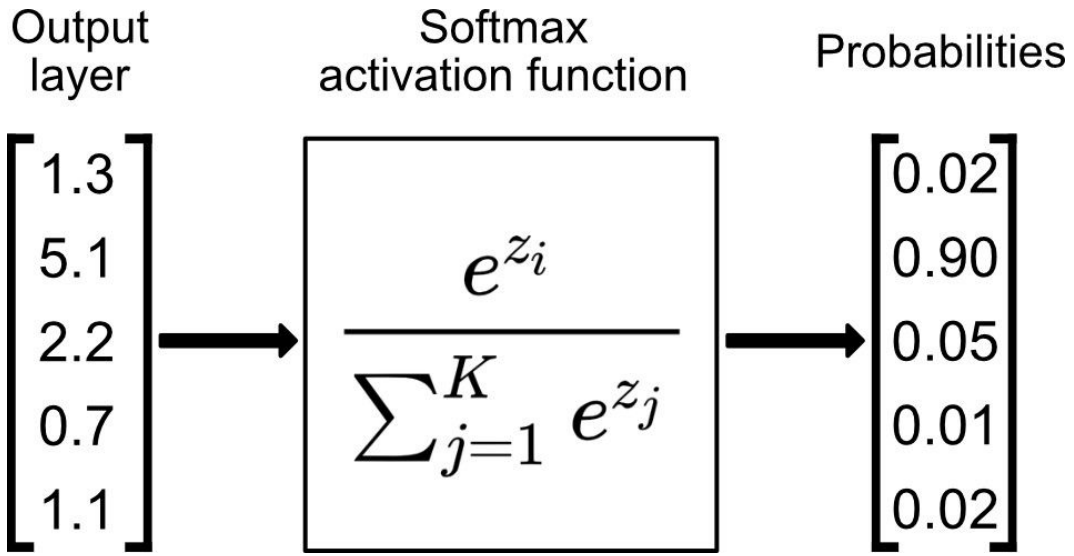
The non-linear transformation to the input making it capable to learn and perform more complex tasks

https://www.3blue1brown.com/

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



https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60

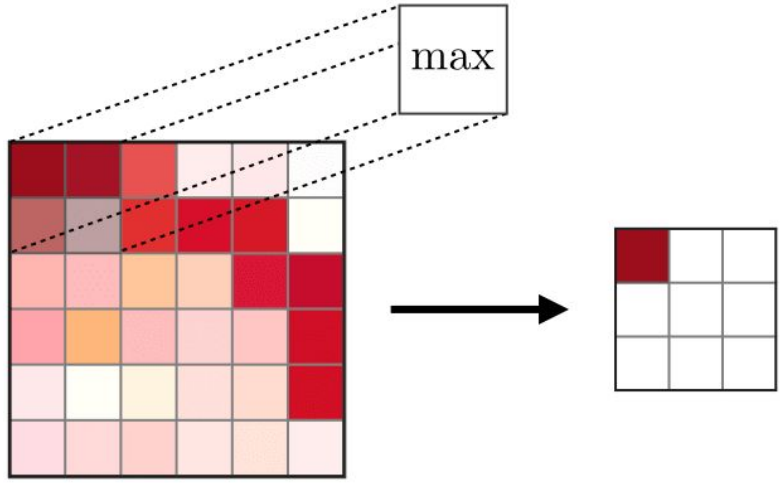
POOLING LAYER

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

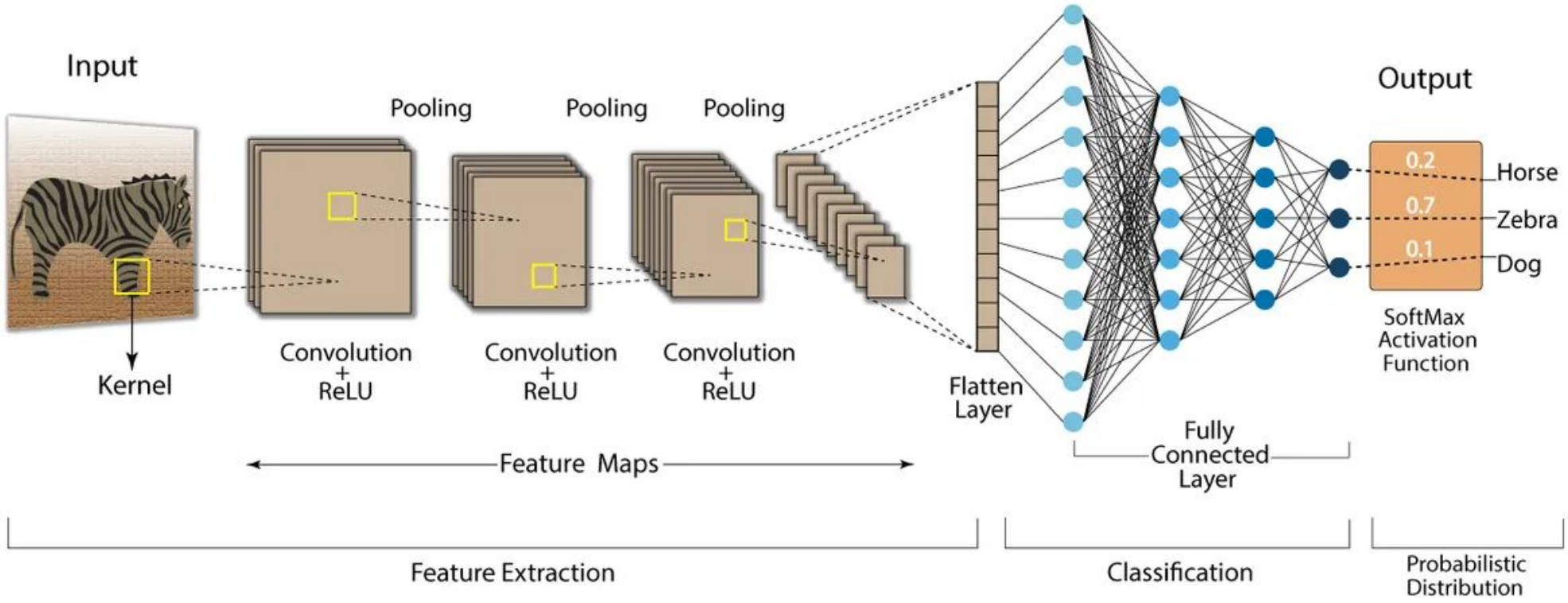
Average pooling - selects the average value in a filter region



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

FULLY CONNECTED LAYERS

Aggregates input from the final feature maps



https://developersbreach.com/convolution-neural-network-deep-learning/

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 at 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

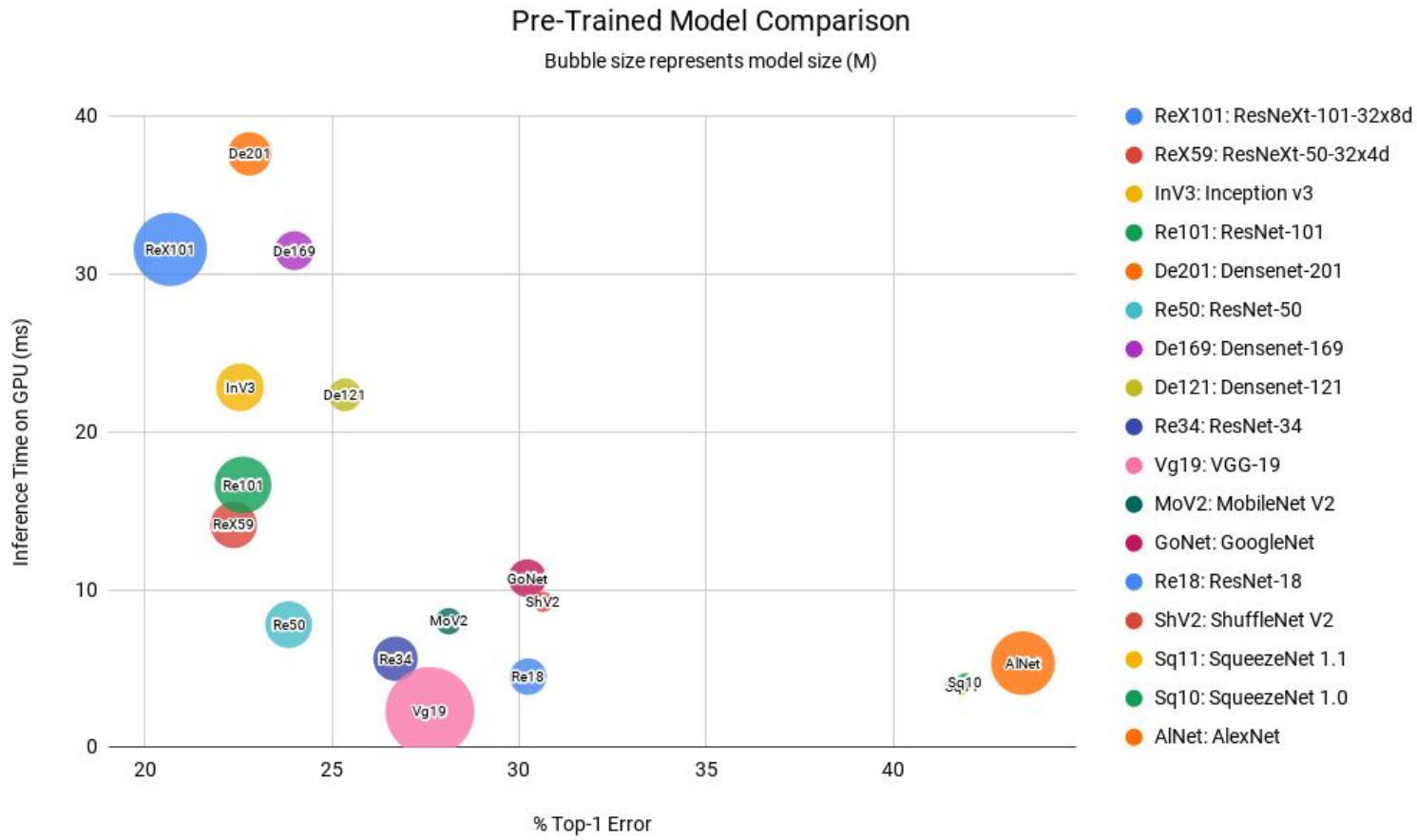
TRANSFER LEARNING IN PYTORCH

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 x H x W) where H and W are at least 224 and then mean=[0.485, 0.456, 0.406] and 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/

DOMINO

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 a 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

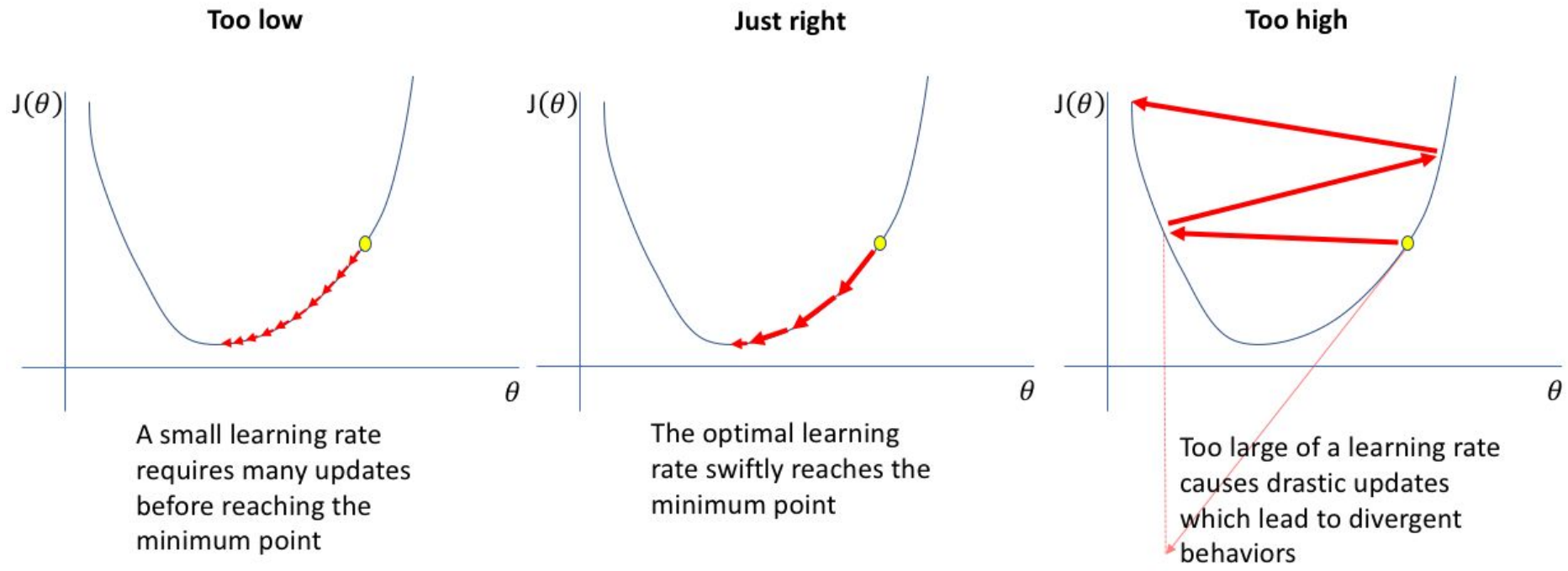
AdaGrad - adapts the learning rate to individual features, so some of the weights will have different learning weights

- 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 to the estimated error each time the weights are updated



https://www.jeremyjordan.me/nn-learning-rate/

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`

DOMINO

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

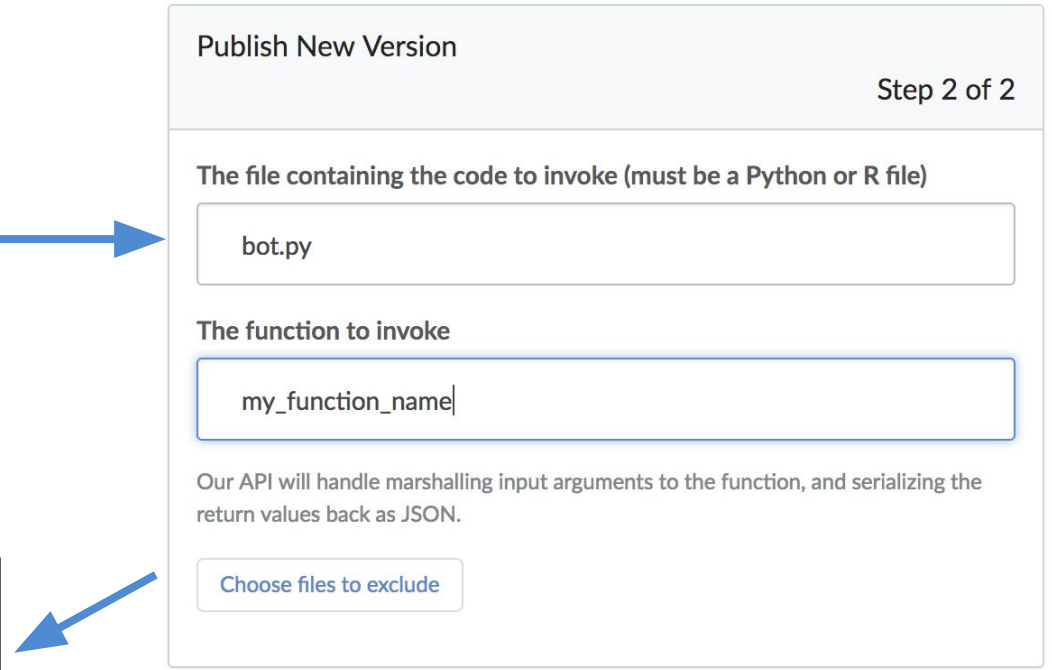
- Developer can create an application in the language of their choice AND still use your R or Python model
- Domino model APIs serve your code as a low-latency web service
- You don't need any API design experience to turn your Python or R code into a Model API
- Details in our Publishing course

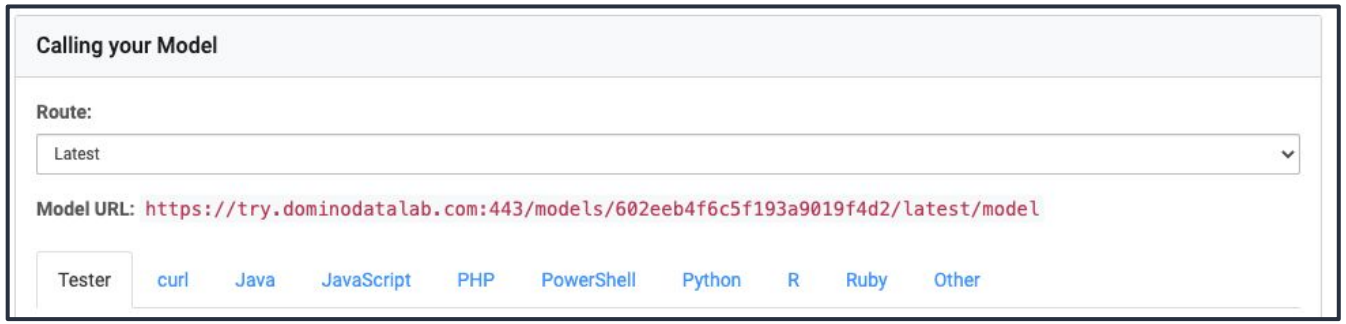
BUILD MODEL API IN DOMINO

Script:

```
load model

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





DOMINO

Advanced Features in Domino

DOMINO

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)

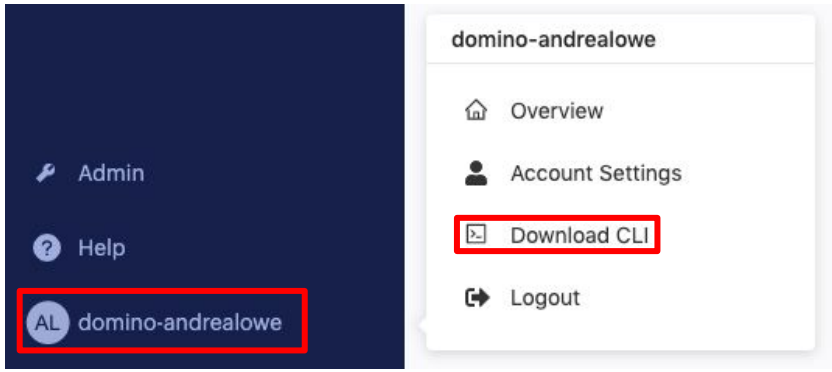
See list of possible commands here and in Domino 201

```
andrealowe — -bash — 120×29
(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$
```



DOMINO

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/

DOMINO

Thank You

www.dominodatalab.com