Carnegie Mellon University
Heinz College

94-775 Unstructured Data Analytics for Policy

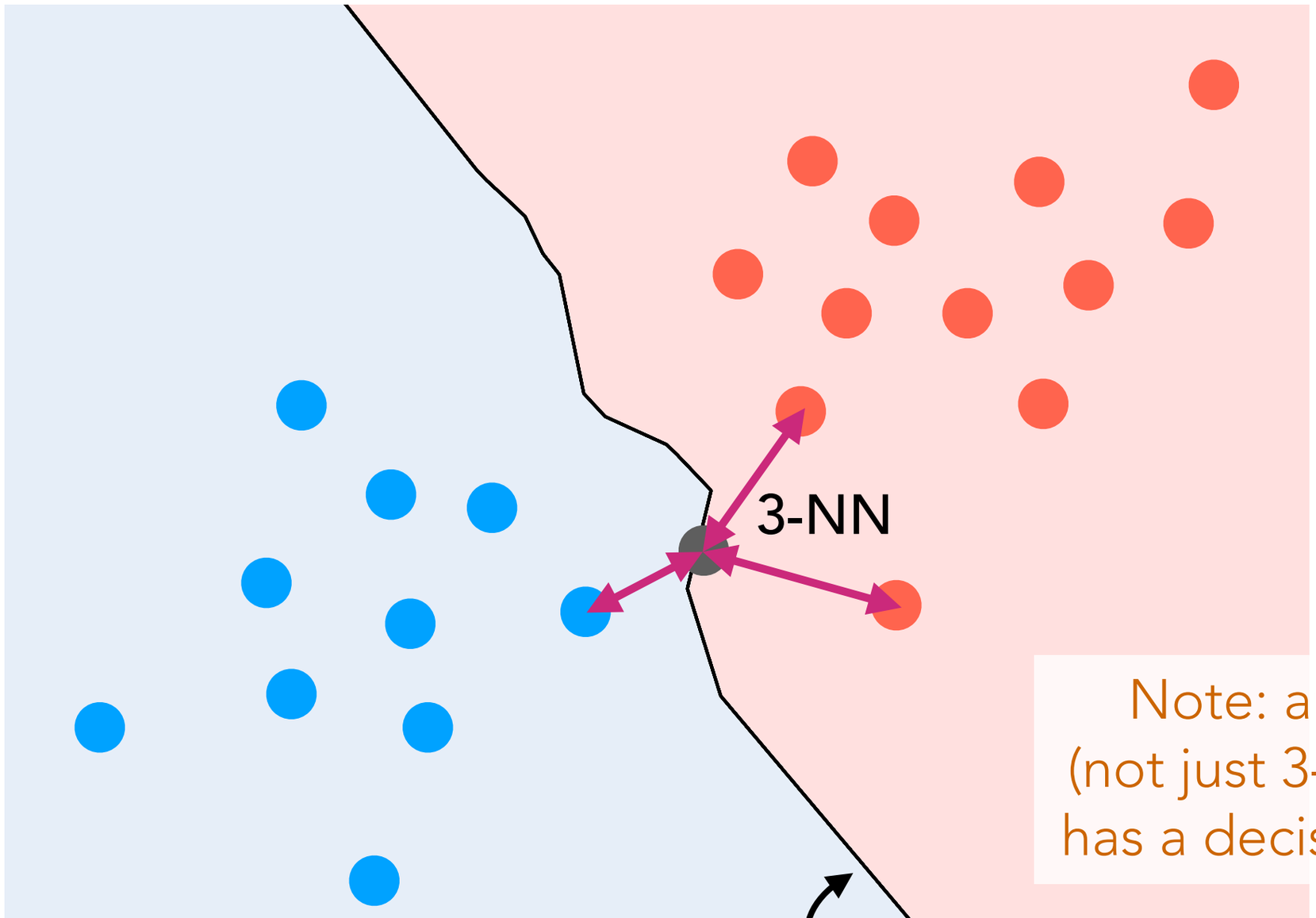
Lecture 12: Neural net basics

Slides by George H. Chen

Administrivia

- Reminder: Quiz 2 is tomorrow during your recitation slot (first 40 min)
    - Gaussian mixture models (GMMs) *used to be* considered in 94-775 but not anymore (the Spring 2025 94-775 Quiz 2 had a GMM question)
        - Your Quiz 2 will not have Gaussian mixture models (because we didn't cover them this semester)
- Required **informal** (low stress!) final project check-ins: for each final project group, please schedule to meet with either your TA Shurui or me during our office hours slots next week to give us an overview of where your group is at (not everyone in your group needs to show up)
    - If you would prefer to chat immediately after Quiz 2 tomorrow, that's also fine but let me know so that I know to allocate time appropriately for you

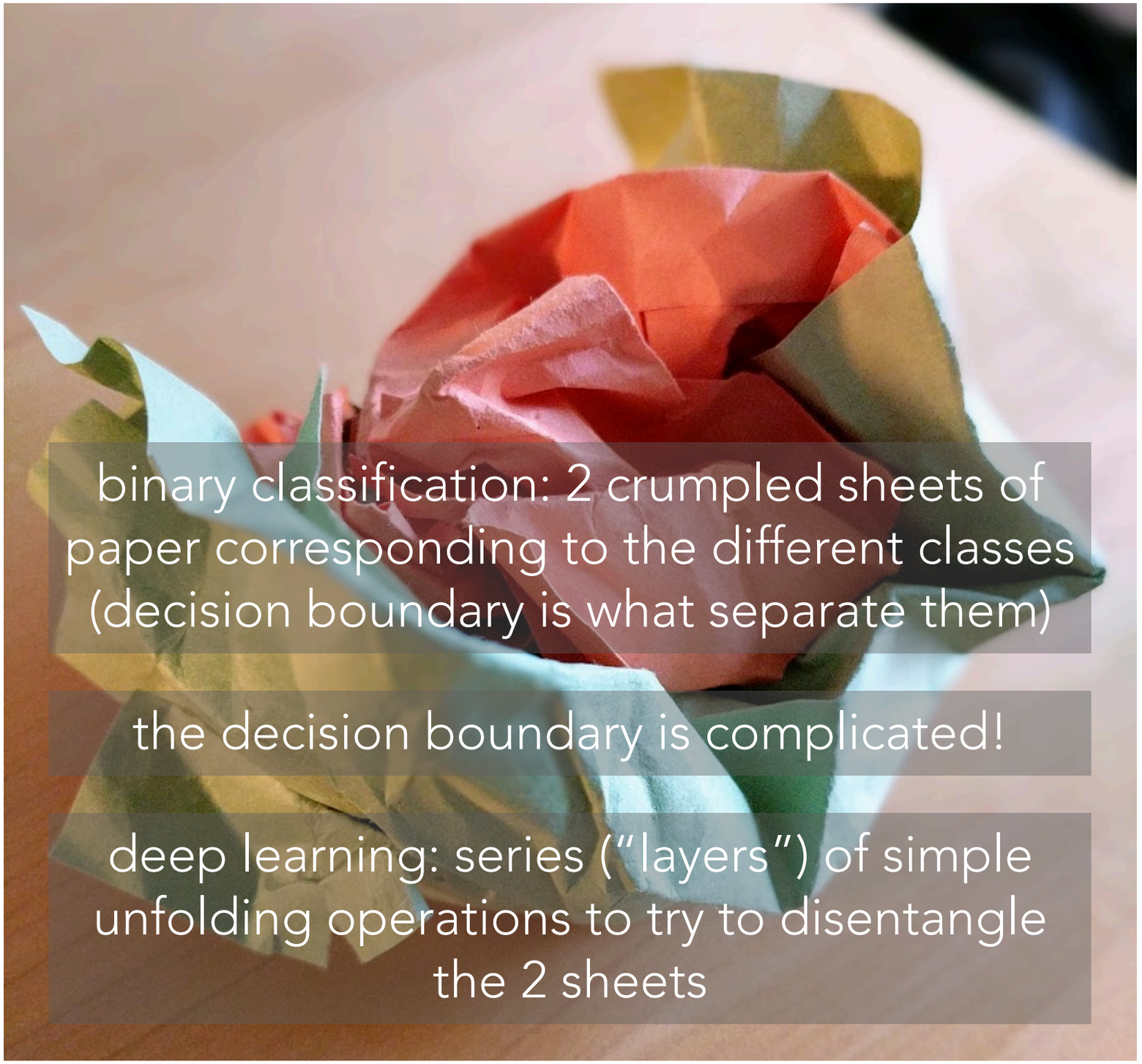
(Flashback) Example: *k*-NN Classification



If test point on right of decision boundary → 3-NN classifier predicts red

If test point on left of decision boundary → 3-NN classifier predicts blue

(Flashback) Crumpled Paper Analogy



Analogy: Francois Chollet, photo: George Chen

(Flashback) Representation Learning

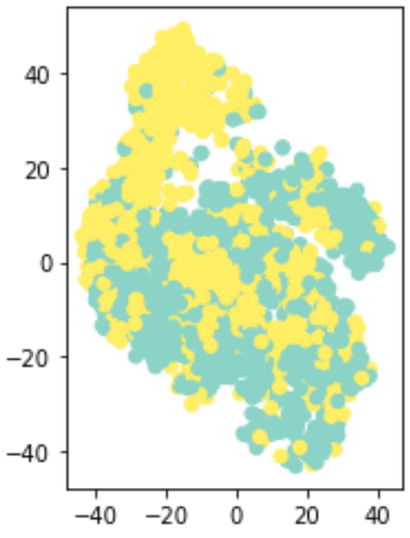
Each layer's output is *another way we could represent the input data*

Learned

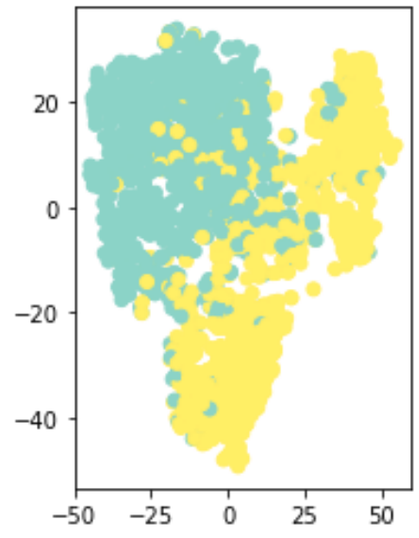
classifier

"clown fish"

Visualize (e.g., t-SNE)



Visualize



Why Does Deep Learning Work?

Actually the ideas behind deep learning are old (~1980's)

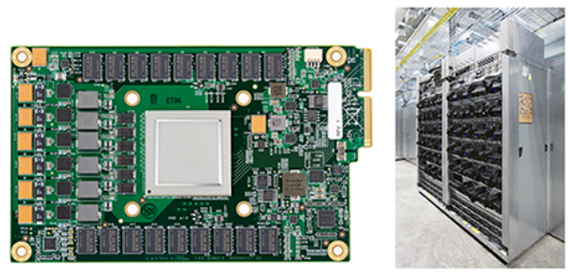
There's even a patent from 1961 that basically amounts to a convolutional neural net for OCR

- Big data

- Better hardware

CPU's
& Moore's law

GPU's



TPU's

Many companies now make dedicated hardware for deep nets (e.g., Google, Apple, Tesla)

- Better algorithms

Structure Present in Data Matters

Neural nets aren't doing black magic

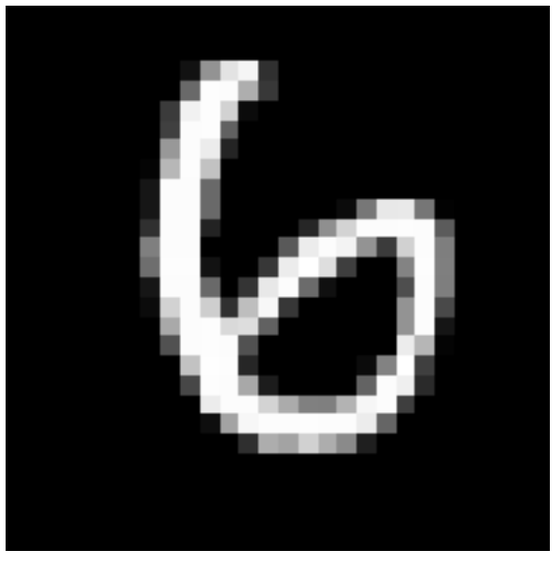
- **Image analysis:** convolutional neural networks (convnets) neatly incorporates basic image processing structure
- **Time series analysis:** transformers learn how to weight previous time steps' contributions to a prediction at the current time step

Text is a time series of tokens
& video is a time series of images!

Handwritten Digit Recognition Example

Walkthrough of 2 extremely simple neural nets

Handwritten Digit Recognition



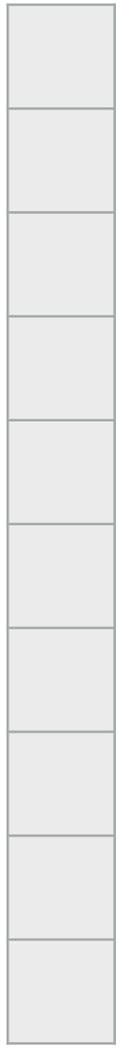
28x28 image

flatten

length 784 vector
(784 input nodes)

weighted sums

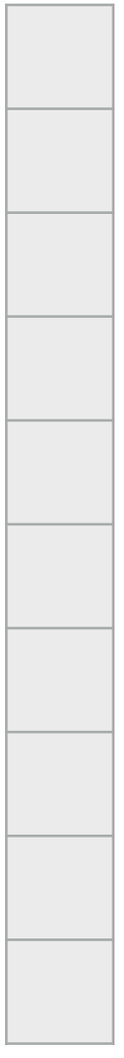
(parameterized by a weight matrix *W* and a bias *b*)



linear layer
with 10 nodes

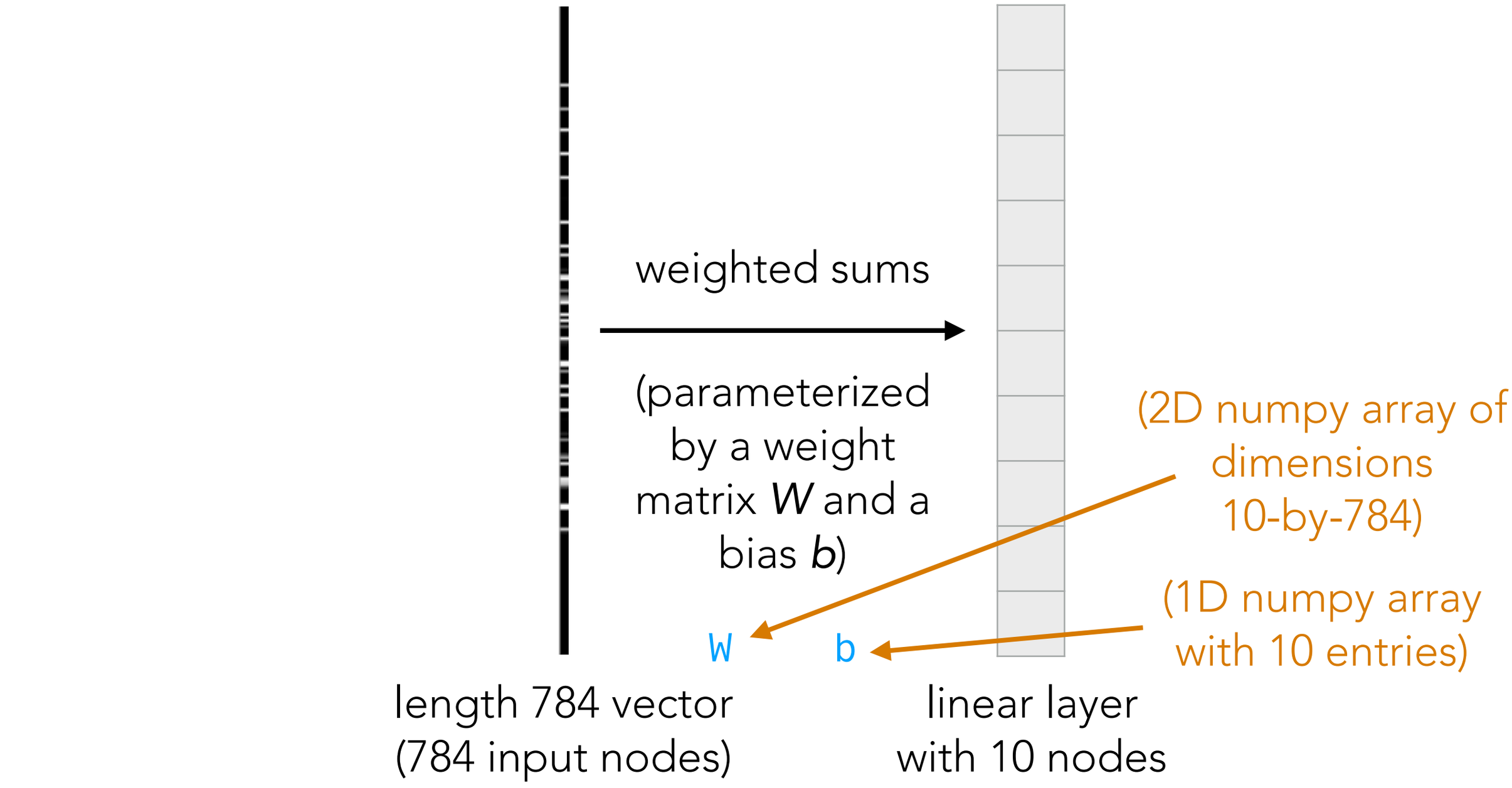
activation

(can be thought of as post-processing)

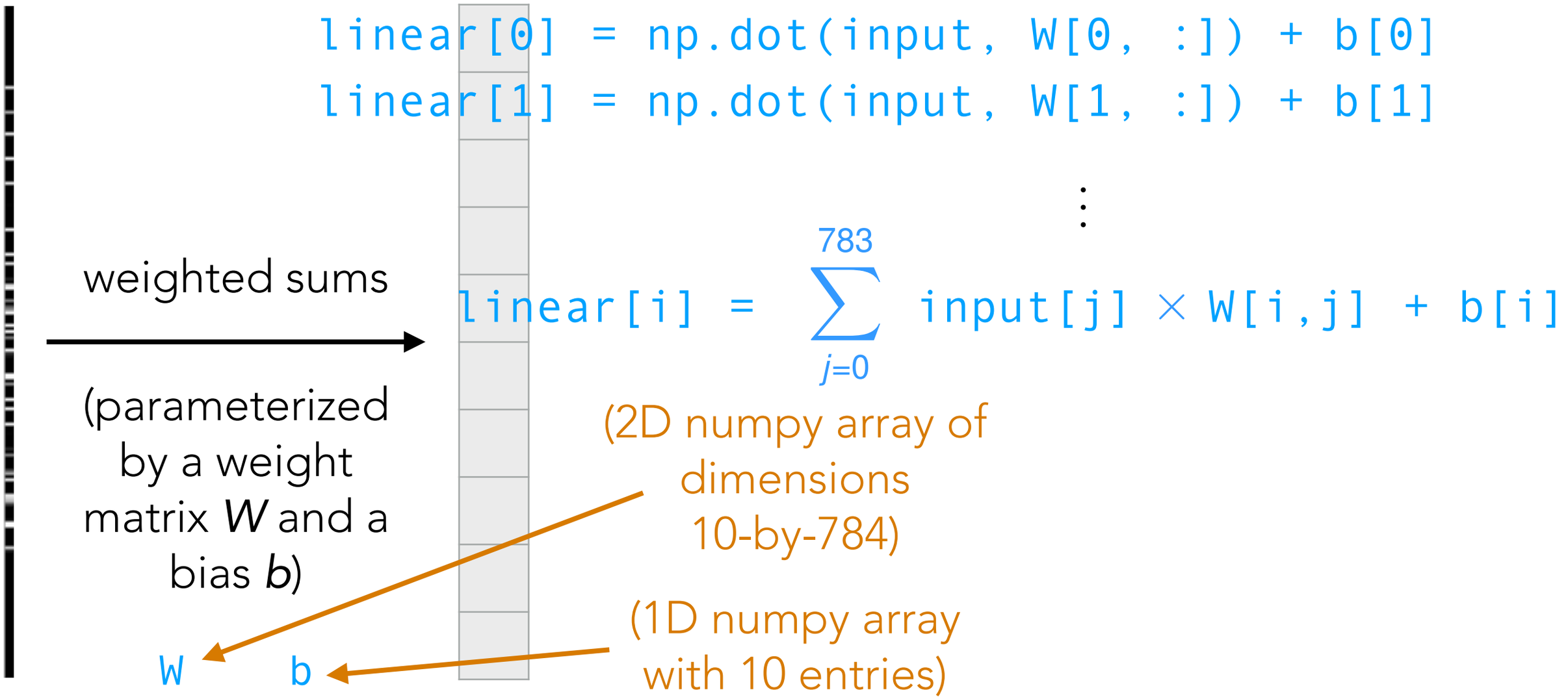


final
output

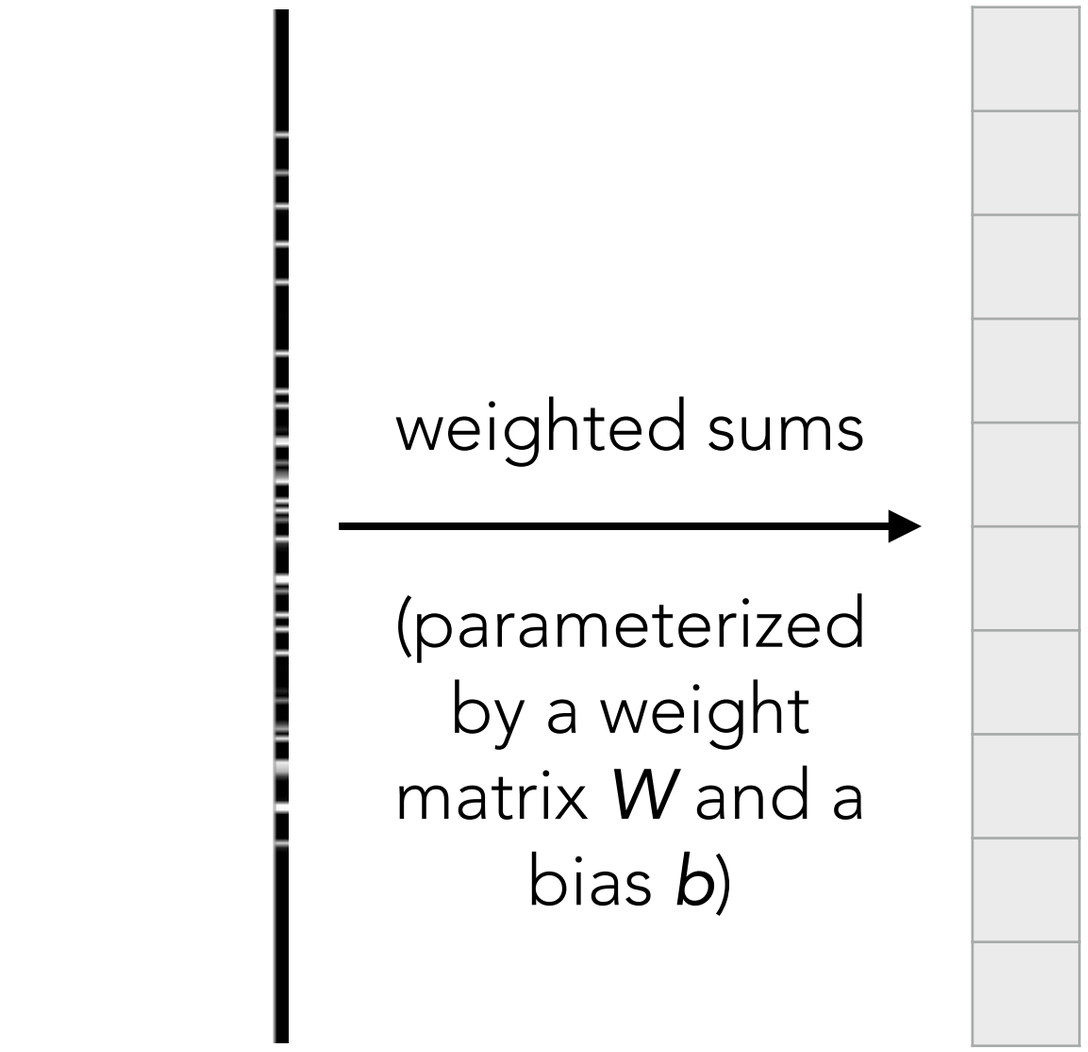
Handwritten Digit Recognition



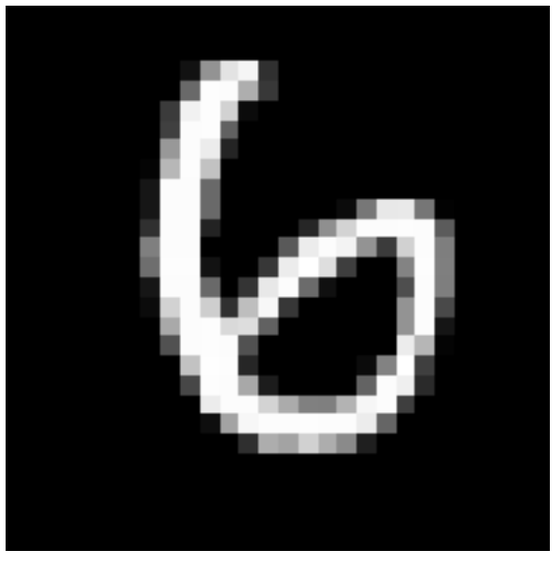
Handwritten Digit Recognition



Handwritten Digit Recognition



Handwritten Digit Recognition



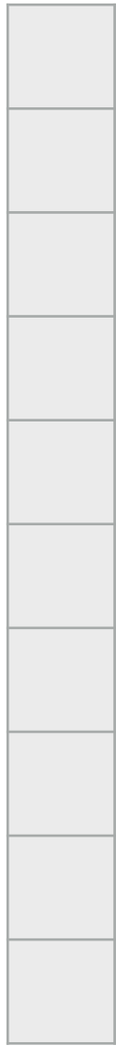
28x28 image

flatten

length 784 vector
(784 input nodes)

weighted sums

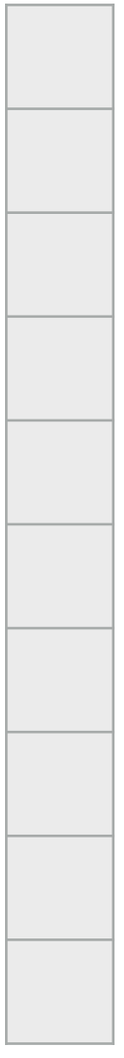
(parameterized by a weight matrix *W* and a bias *b*)



linear layer
with 10 nodes

activation

(can be thought of as post-processing)



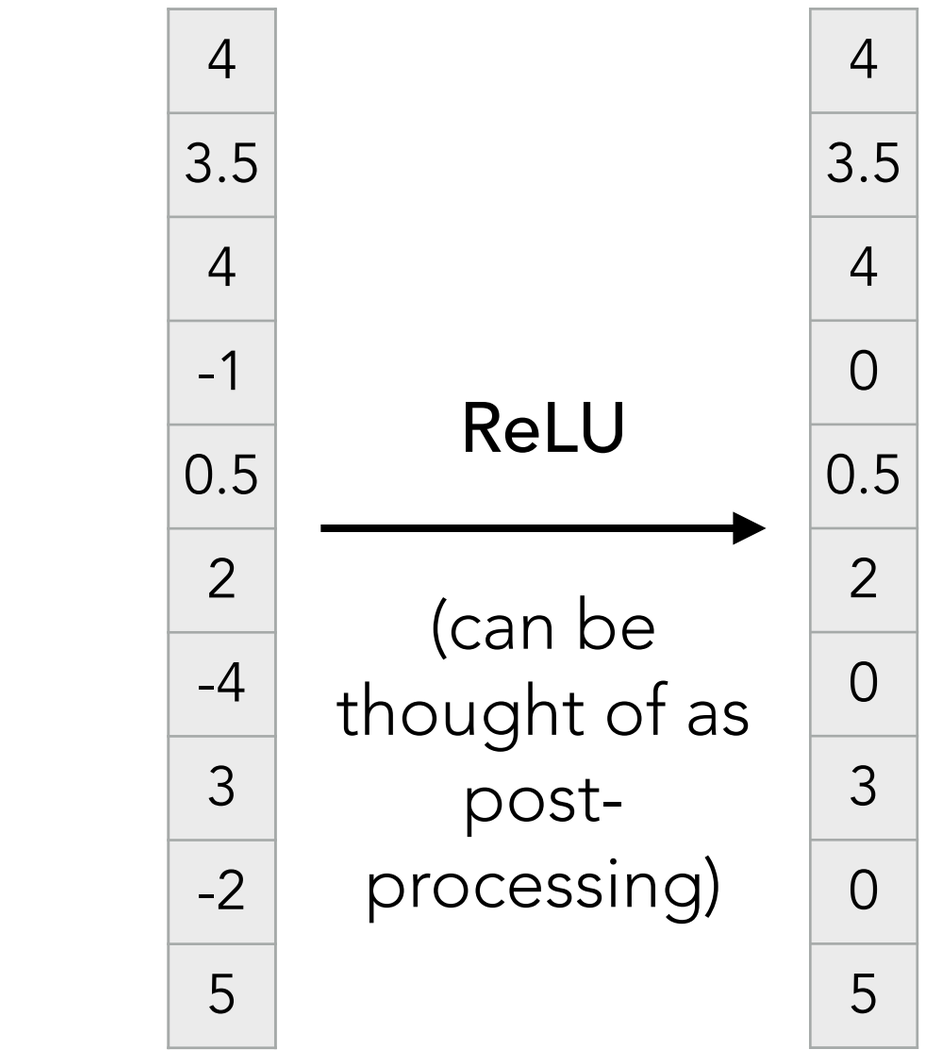
final
output

Handwritten Digit Recognition

Many different activation functions possible

Example: **Rectified linear unit (ReLU)**
zeros out entries that are negative

```
final = np.maximum(0, linear)
```

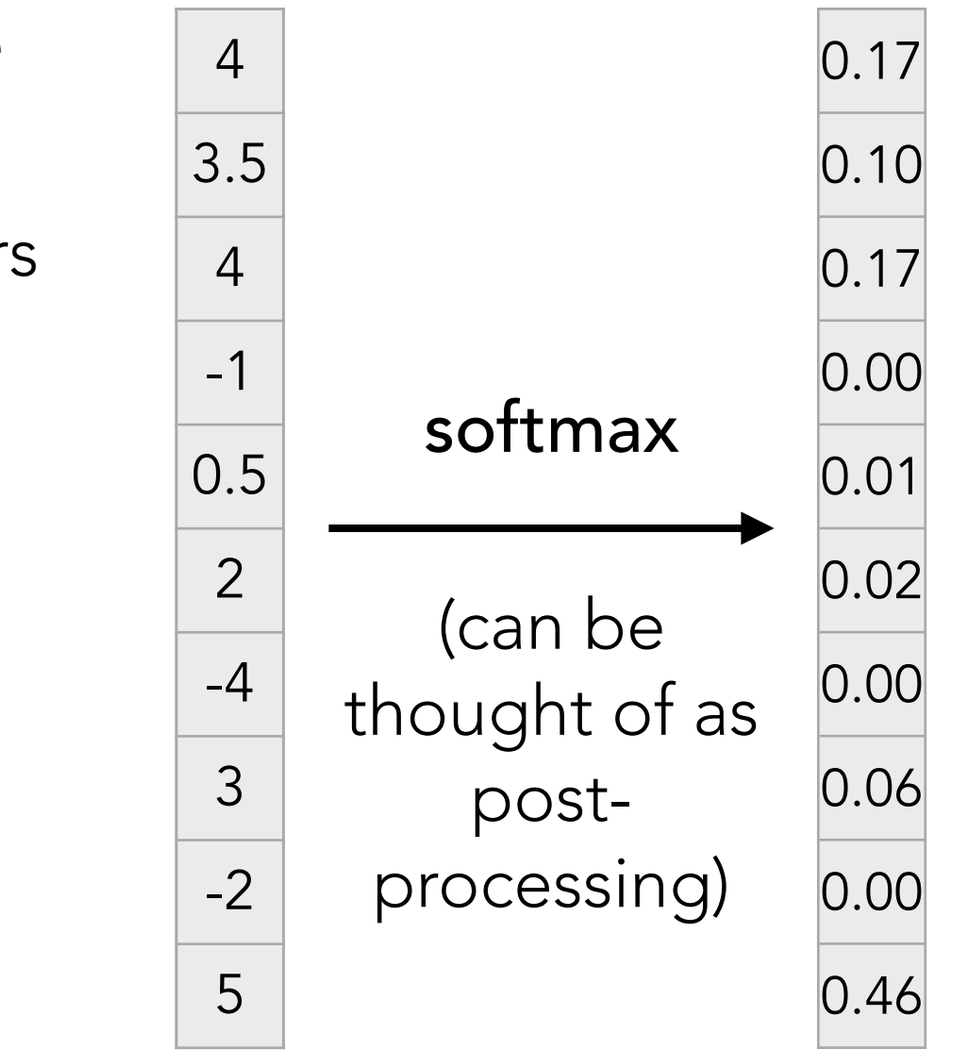


Handwritten Digit Recognition

Many different activation functions possible

Example: **softmax** converts a table of numbers into a probability distribution

```
exp = np.exp(linear)
final = exp / exp.sum()
```



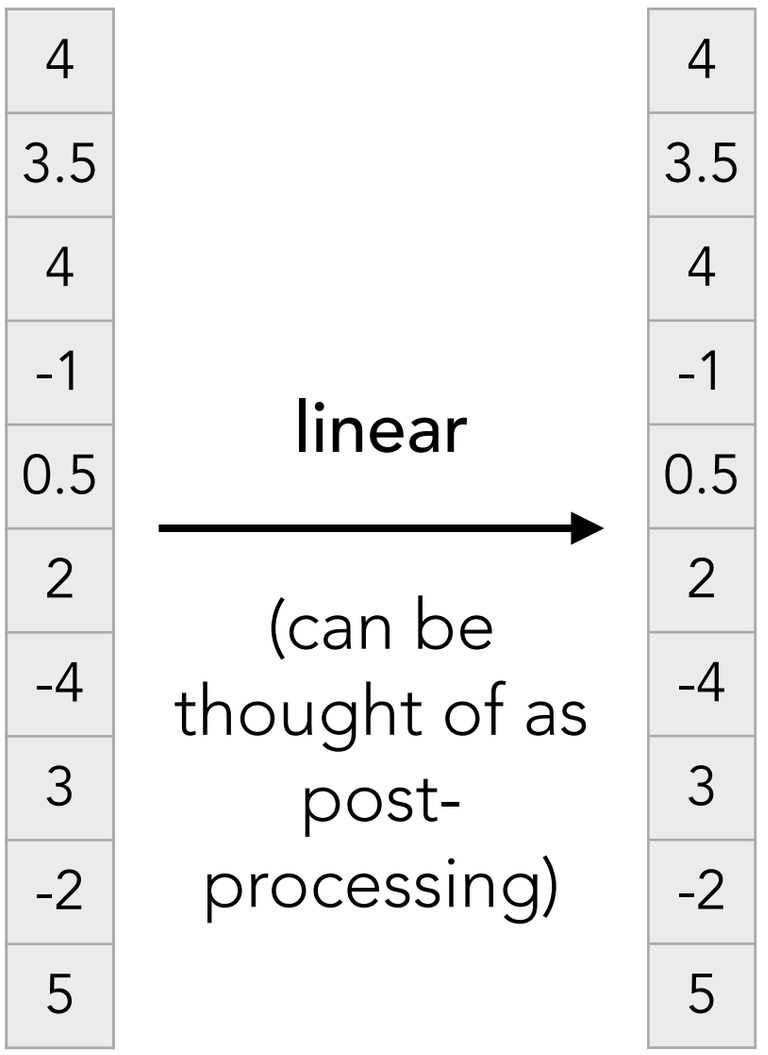
Handwritten Digit Recognition

Many different activation functions possible

Example: **linear** activation does nothing

This is equivalent to there being no activation function

```
final = linear
```



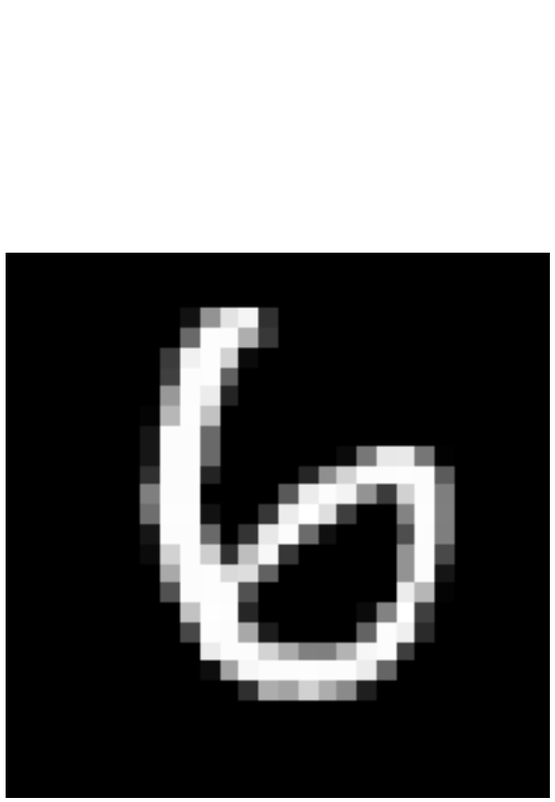
linear layer with 10 nodes

```
linear
```

final output

```
final
```

Handwritten Digit Recognition



28x28 image

flatten

length 784 vector
(784 input nodes)

weighted sums

(parameterized by a weight matrix *W* and a bias *b*)

linear layer
with 10 nodes

softmax

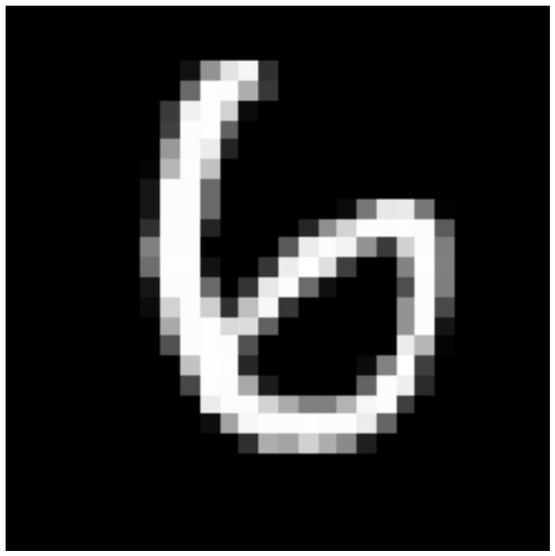
Pr(digit 0)
Pr(digit 1)
Pr(digit 2)
Pr(digit 3)
Pr(digit 4)
Pr(digit 5)
Pr(digit 6)
Pr(digit 7)
Pr(digit 8)
Pr(digit 9)

final output

Desired result

Handwritten Digit Recognition

Training label: 6



Input

Flatten

Linear
(10 nodes)

Softmax

Loss

error

Error is averaged across training examples

Popular loss function for classification:
categorical cross entropy

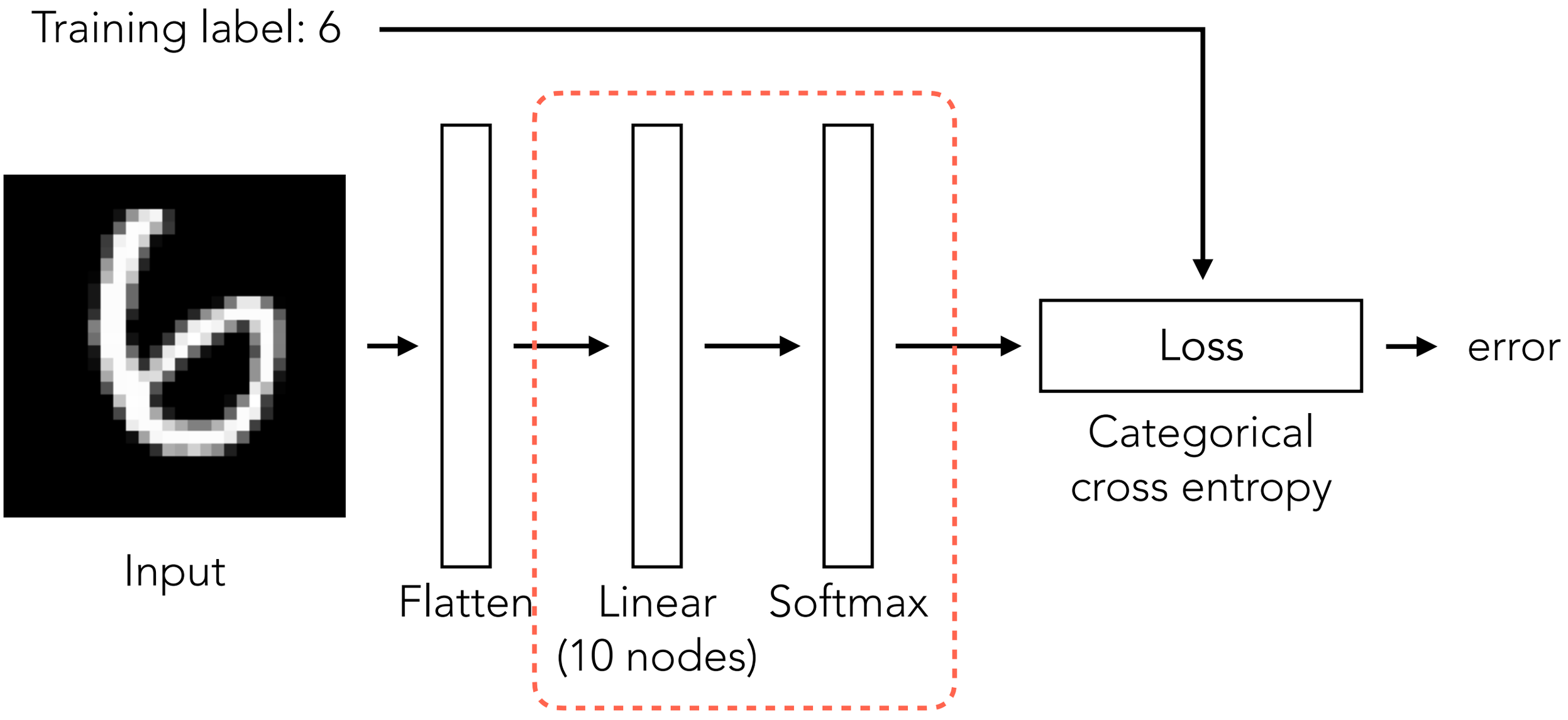
$$\log \frac{1}{\text{estimated Pr(digit 6)}}$$

Learning this neural net means finding W and b that minimize categorical cross entropy loss

Also called *fully-connected* or *dense* layer

⚠️ In PyTorch, softmax is included as part of the cross entropy loss

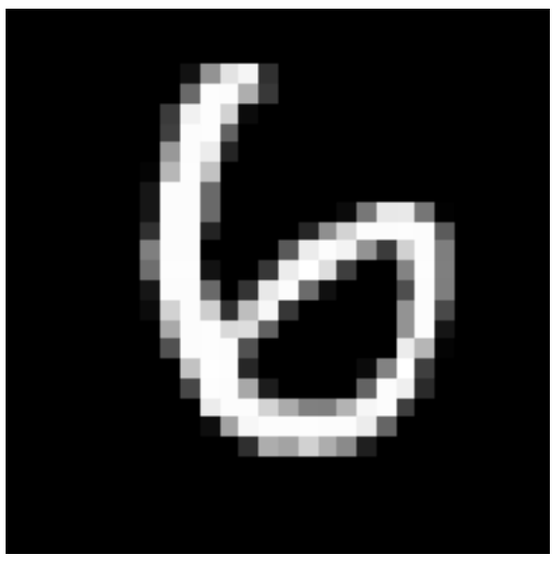
Handwritten Digit Recognition



This neural net has a name: **multinomial logistic regression**
(when there are only 2 classes, it's called **logistic regression**)

Handwritten Digit Recognition

Training label: 6



Input

Flatten

Linear (512 nodes)

ReLU

Linear (10 nodes)

Softmax

Loss

error

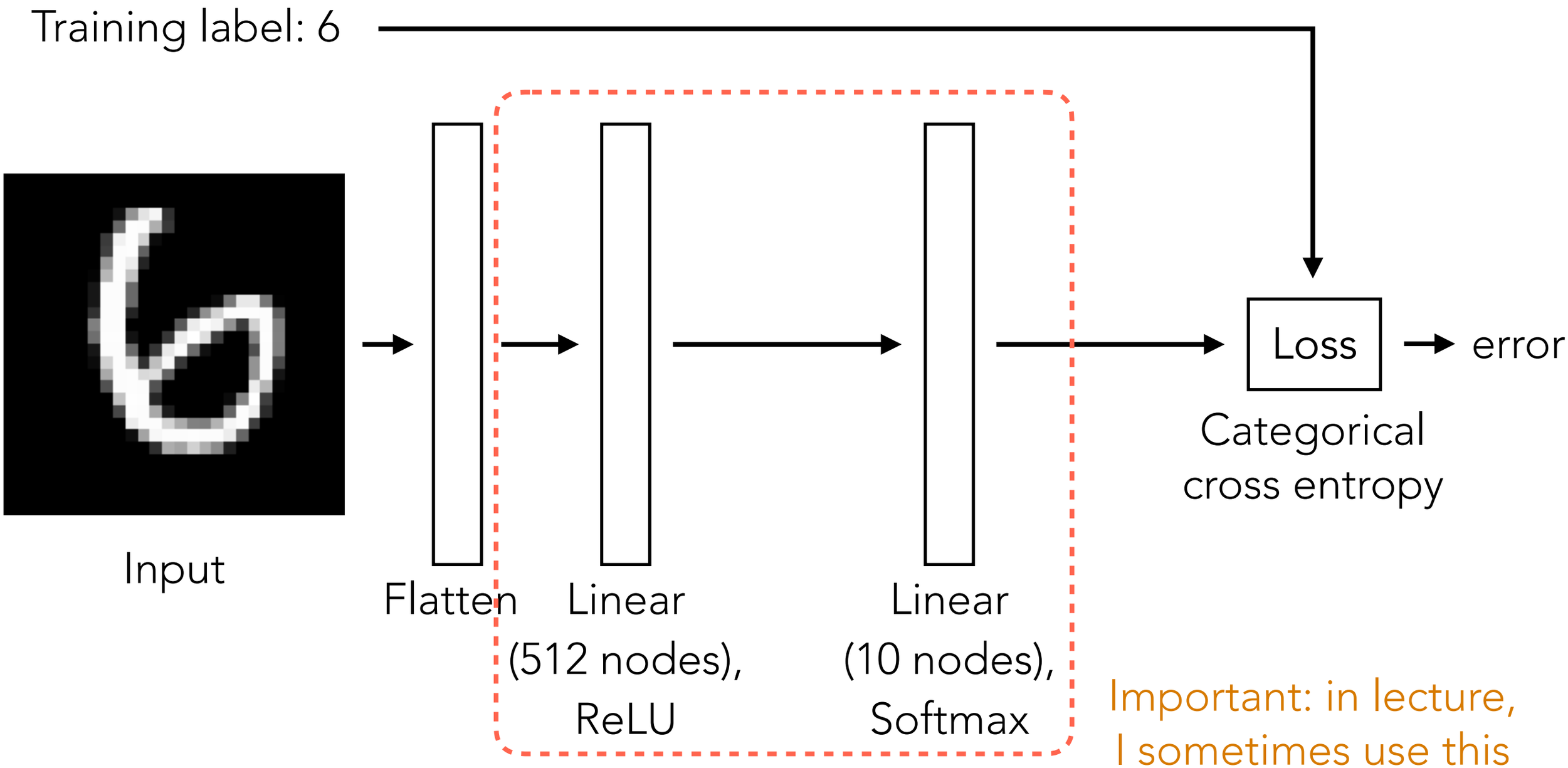
Categorical cross entropy

Different linear layers; each has its own weight matrix and bias vector

Basic building block of neural nets: *linear layer with nonlinear activation*

Learning this neural net ⇒ learn parameters of both linear layers

Handwritten Digit Recognition



This neural net is called a multilayer perceptron
(# nodes need not be 512 & 10;
activations need not be ReLU and softmax)

Important: in lecture,
I sometimes use this
shorthand notation
(specifying activation to
go with each linear layer)

PyTorch

- Designed to be like NumPy
  - A lot of (but not all) function names are the same as numpy (e.g., instead of calling `np.sum`, you would call `torch.sum`, etc)
  - ⚠️ PyTorch does not use NumPy arrays and instead uses tensors (so instead of `np.array`, you use `torch.tensor`)
- What's the big difference then? Why not just use NumPy?
  - **PyTorch tensors keep track of what device they reside on**
    - ⚠️ For example, trying to add a tensor stored on the CPU and a tensor stored on a GPU will result in an error!
  - **PyTorch tensors can automatically store "gradient" information** (important for learning model parameters; details in later lecture)

PyTorch code is often harder to debug than NumPy code

There's a PyTorch tutorial posted in supplemental materials

Handwritten Digit Recognition

Demo

Architecting Neural Nets

- Basic building block that is often repeated:
  linear layer followed by *nonlinear* activation
    - Without nonlinear activation, two consecutive linear layers is mathematically equivalent to having a single linear layer!
- How to select # of nodes in a layer, or # of layers?
    - These are hyperparameters! *Infinite* possibilities!
    - Choose between different hyperparameter settings by using the strategy from last lecture (choose based on validation accuracy)
        - Very expensive in practice!
          (Active area of research: neural architecture search)
    - Much more common in practice: modify existing architectures that are known to work well

PyTorch Has Lots of Examples

pytorch.org/examples/

PyTorch | Get Started | Ecosystem | PyTorch Edge | Blog | Tutorials | Docs | Resou

Search Docs

PyTorch Examples

Docs > PyTorch Examples

PYTORCH EXAMPLES

This pages lists various PyTorch examples that you can use to learn and experiment with PyTorch.

Image Classification using Vision Transformer

This example shows how to train a Vision Transformer from scratch on the CIFAR10 database.

GO TO EXAMPLE

Image Classification Using ConvNets

This example demonstrates how to run image classification with Convolutional Neural Networks ConvNets on the MNIST database.

GO TO EXAMPLE

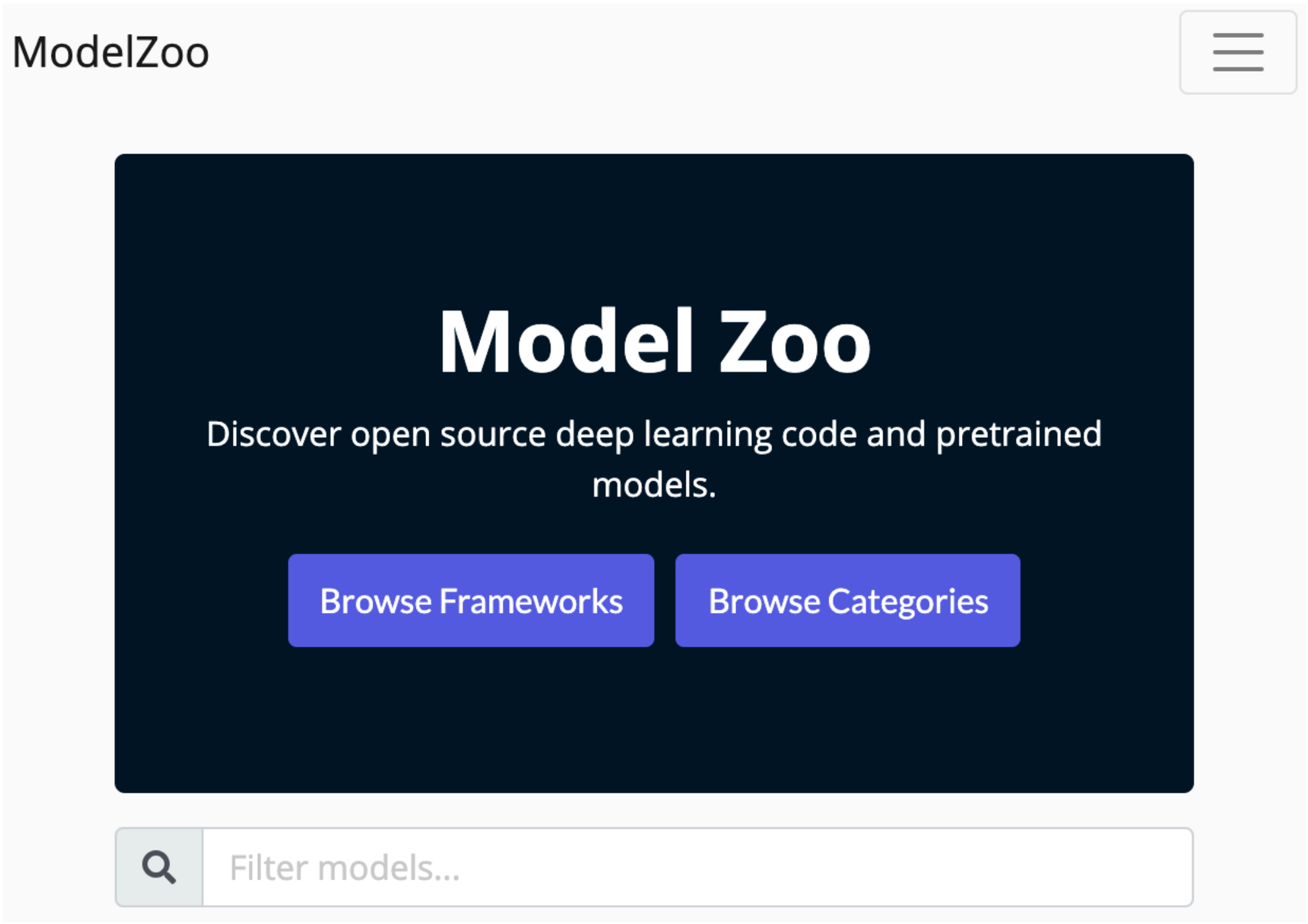
Measuring Similarity using Siamese Network

This example demonstrates how to measure similarity between two images using Siamese

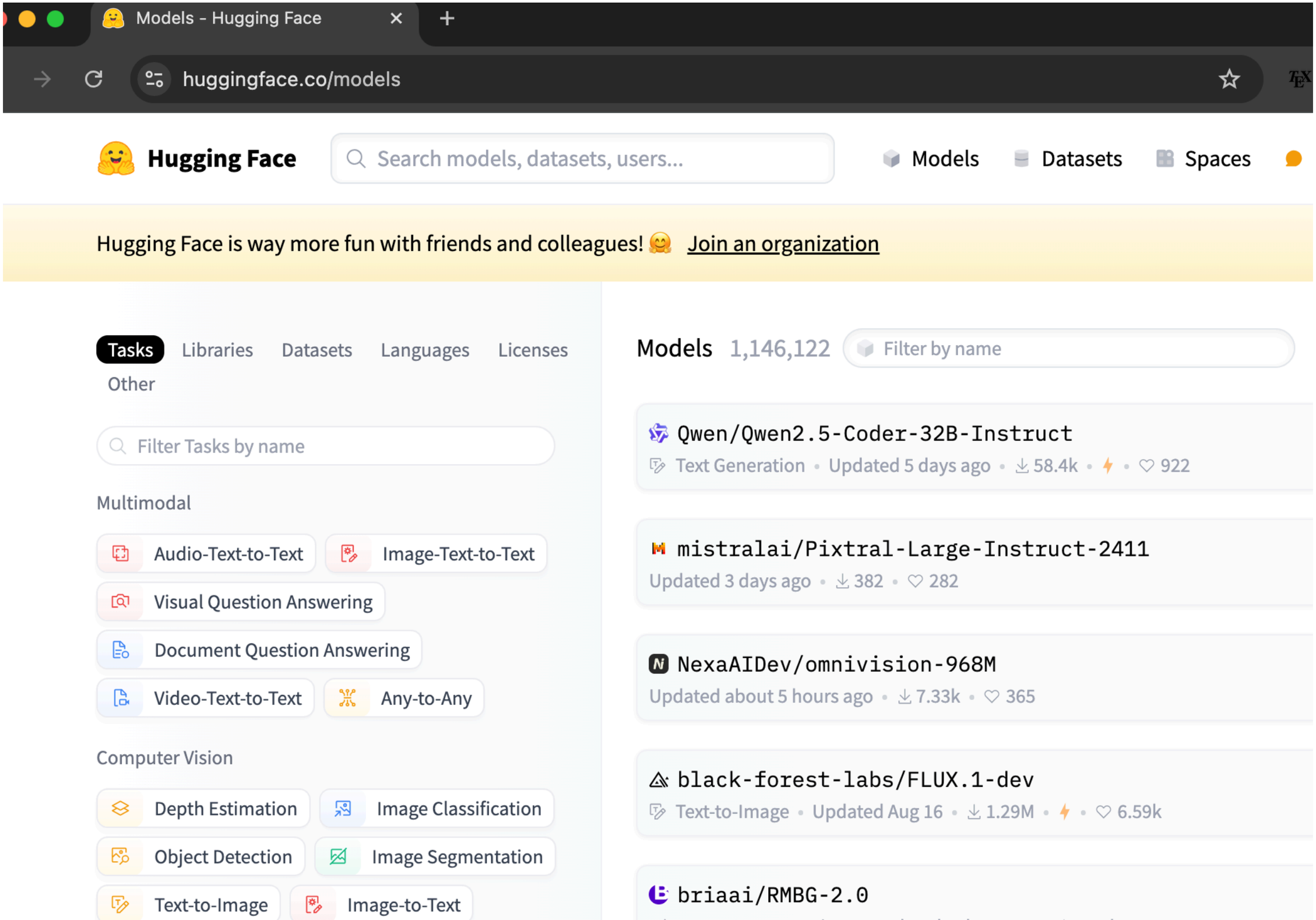
Word-level Language Modeling using RNN and Transformer

This example demonstrates how to train a multi-

Find a Massive Collection of Models at the Model Zoo



More Recently: Lots of Models are on Hugging Face 🤗



Learning a Neural Net Can Be Thought of as Learning a Computer Program

Given `input`, learn a computer program that computes `output`

this is a function

Multinomial logistic regression:

```
def f(input):
    linear = np.dot(input, W.T) + b
    output = softmax(linear)
    return output
```

the only things that we are learning
(we fix their dimensions in advance)

We are fixing what the function `f` looks like in code and are only adjusting `W` and `b`!!!

Learning a Neural Net Can Be Thought of as Learning a Computer Program

Given `input`, learn a computer program that computes `output`

Multinomial logistic regression:

```
linear = np.dot(input, W.T) + b
output = softmax(linear)
```

Multilayer perceptron:

```
linear1 = np.dot(input, W1.T) + b1
relu = np.maximum(0, linear1)  # ReLU
linear2 = np.dot(relu, W2.T) + b2
output = softmax(linear2)
```

Learning a neural net: learning a simple computer program that maps inputs (raw feature vectors) to outputs (predictions)

More layers ⇒ computer program has more lines of code