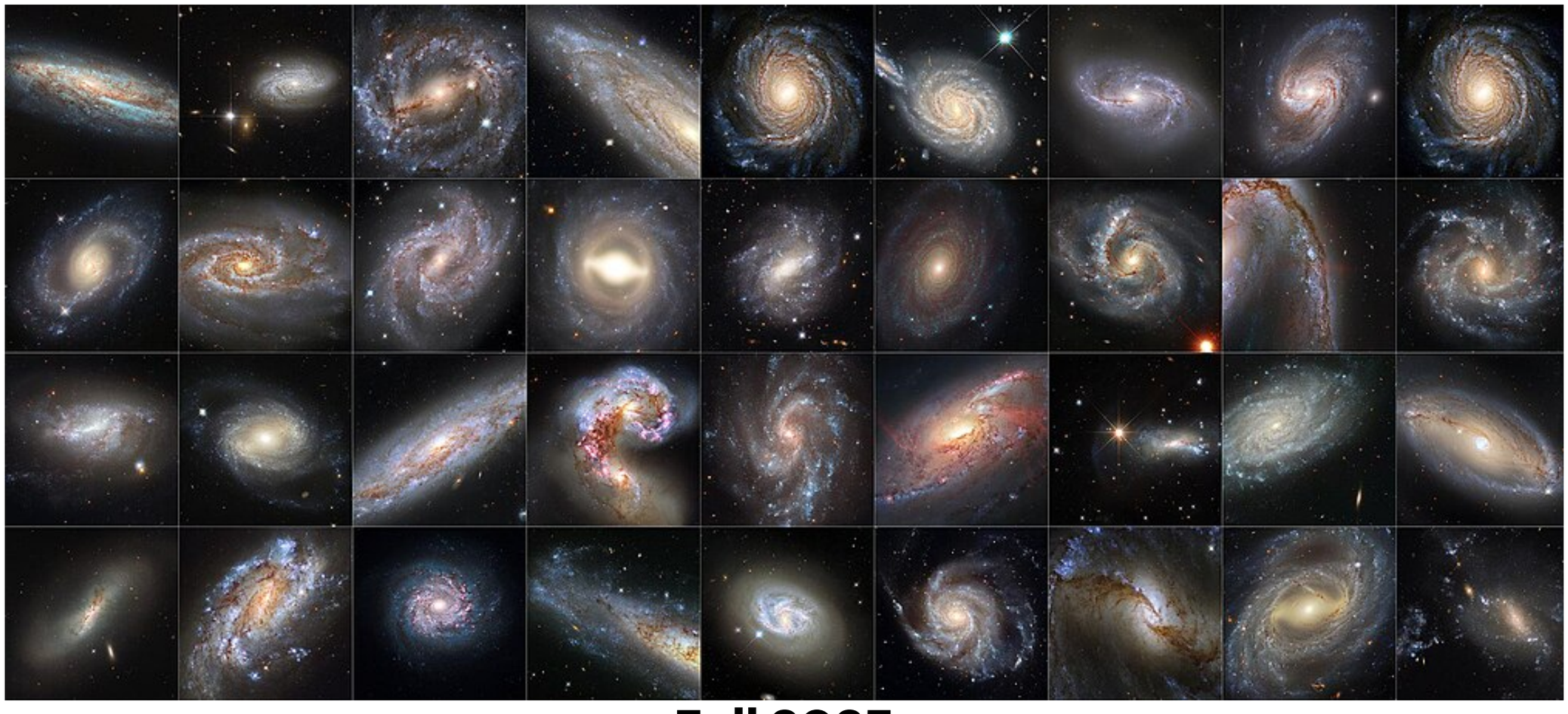
Classifying Galaxies
Lecture 7



Fall 2025

J Noronha-Hostler

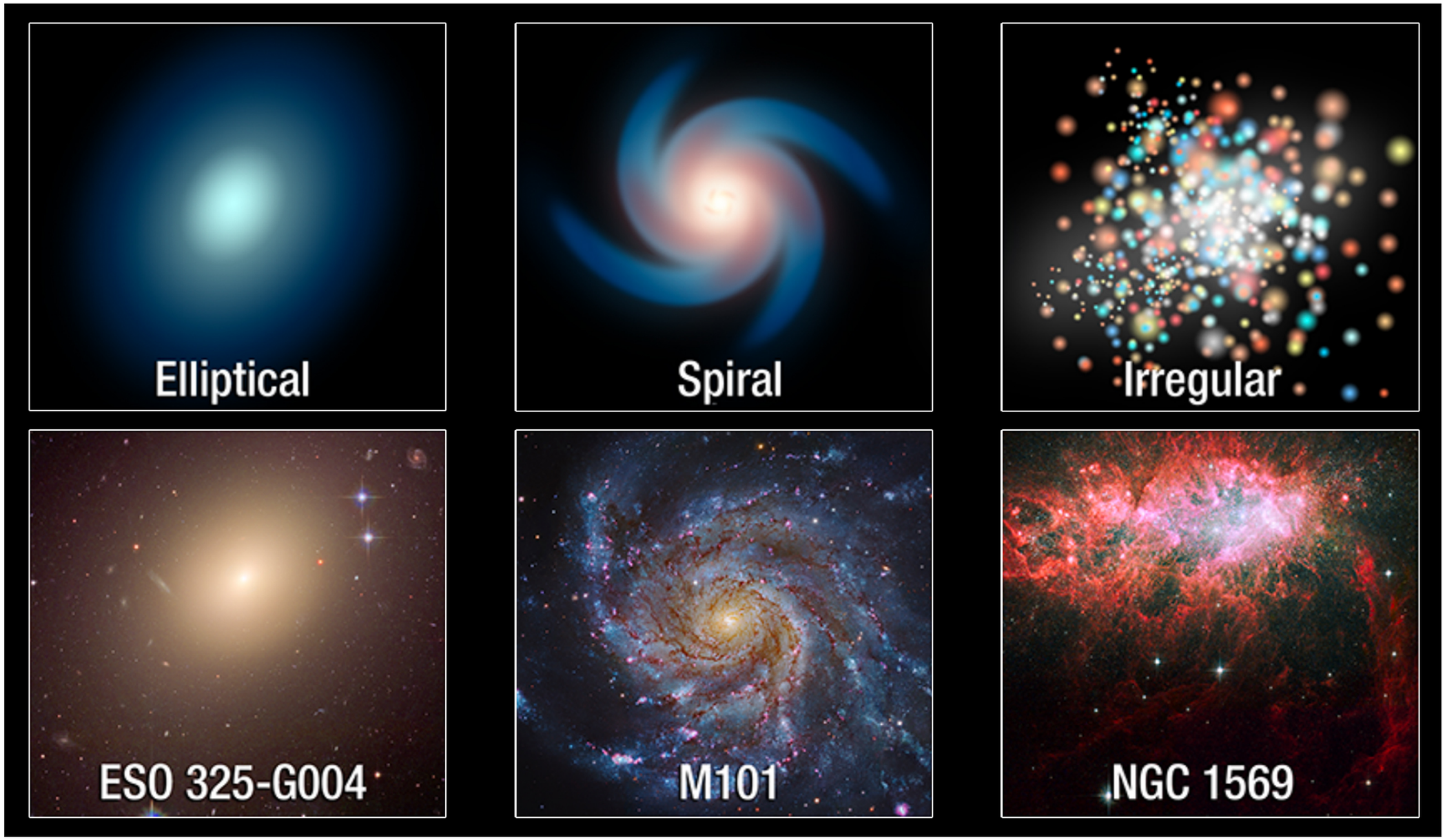
https://jnoronhahostler.github.io/IntroductionToComputationalPhysics/intro.html

Announcements/notes

- Plotting: If the data looks like a straight line, make sure to plot the data points so we can see there is data there
- We are taking off points for late assignments, if you've been given an extension and that wasn't graded correctly let us know:
    - Late grades = Original Grade(include extra credit)*0.8
- Error in the assignment length 10 should be length 5 (fixed on GitHub now)

Galaxies come in all shapes and sizes

How do we sort them?



Visual differences vs physical differences

Age, dust, gas, and star formation

Older, more stable

Little dust

Little star formation

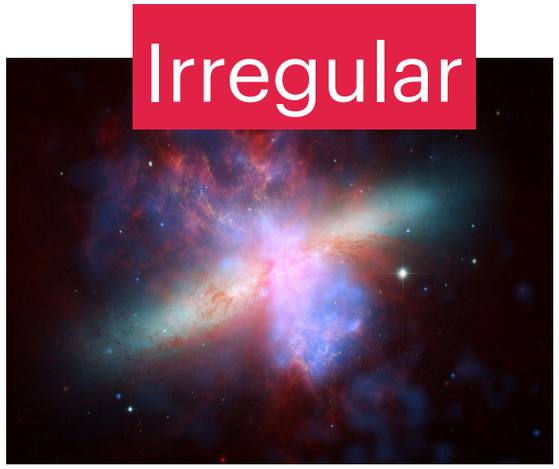
Founds in dense environments

“Newer”, less stable

Lots of gas and dust

Ongoing star formation

Low density regimes



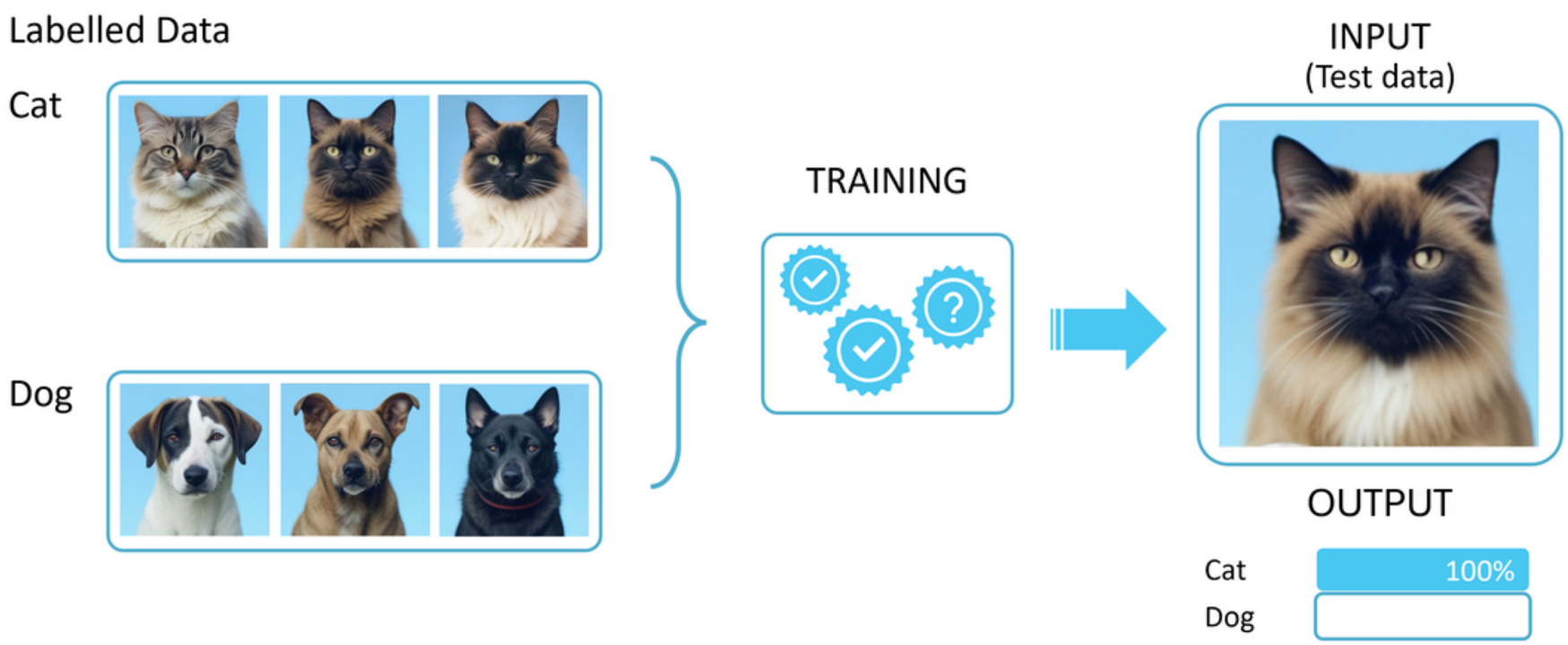
Often exist because of galaxy-galaxy mergers

Gas-rich and star forming

Machine Learning (ML)

Training data, classification

- How do machines learn?
- Generate (labeled) training data
- Train your ML model
- Check the accuracy of your ML model: Test data
- Add more training data (if needed)



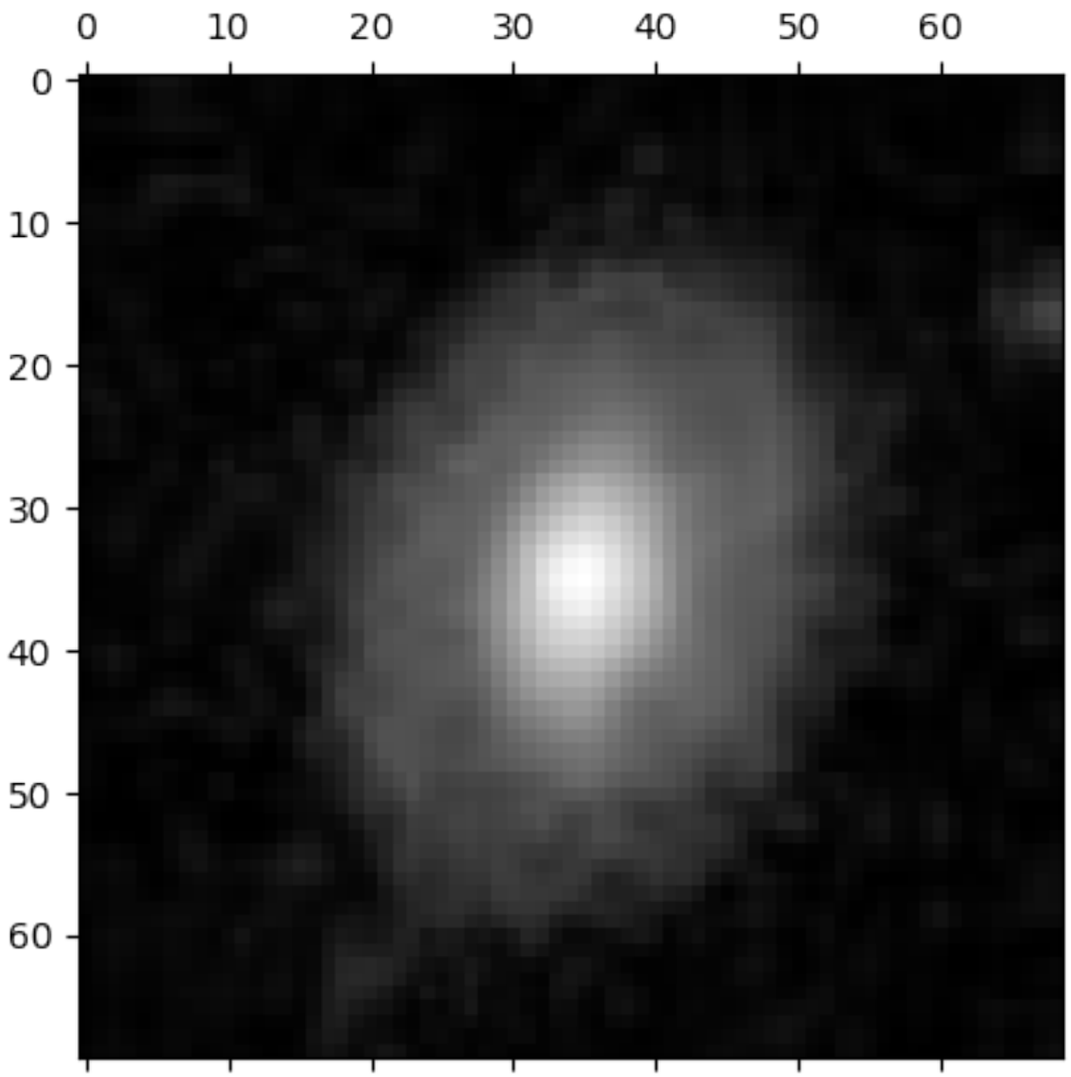
What we'll do today

Neural Network

- Take hand-labeled galaxies from the Sloan Digital Sky Survey
  - Read in external data:
    - gzip (compresses data)
    - pickle (python only) - stores the "state" of things like classes, lists, functions etc
- Write and train a deep neural network to classify them
  - Training data vs testing data
- Goal: 66% accuracy

From imagine to vector

$$f : \mathbb{R}^{4761} \rightarrow \mathbb{R}^{5}$$



69x69 values with value [0,1]

Labels:

[0.9,

0.05,

0.05,

0.02,

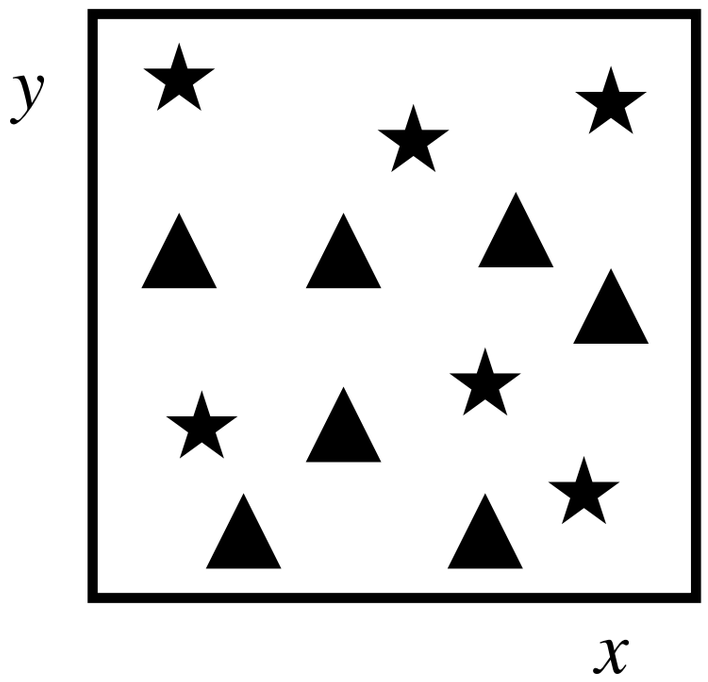
0.1]

5 numbers that represent confidence in each category.

Our goal is for our function to confidently

Training vs testing data

ML methods

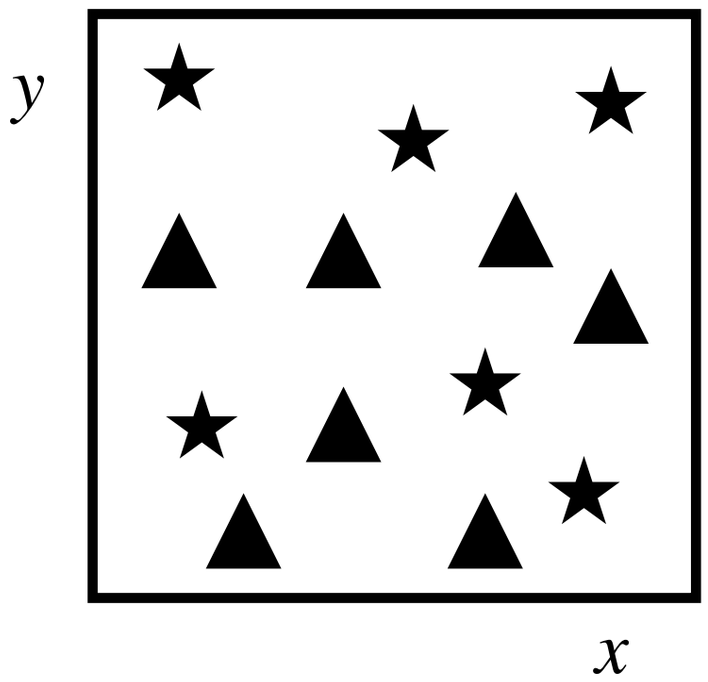


Imagine your parameter space in 2D (x,y)

How do we know a machine learning model is working?

Training vs testing data

ML methods

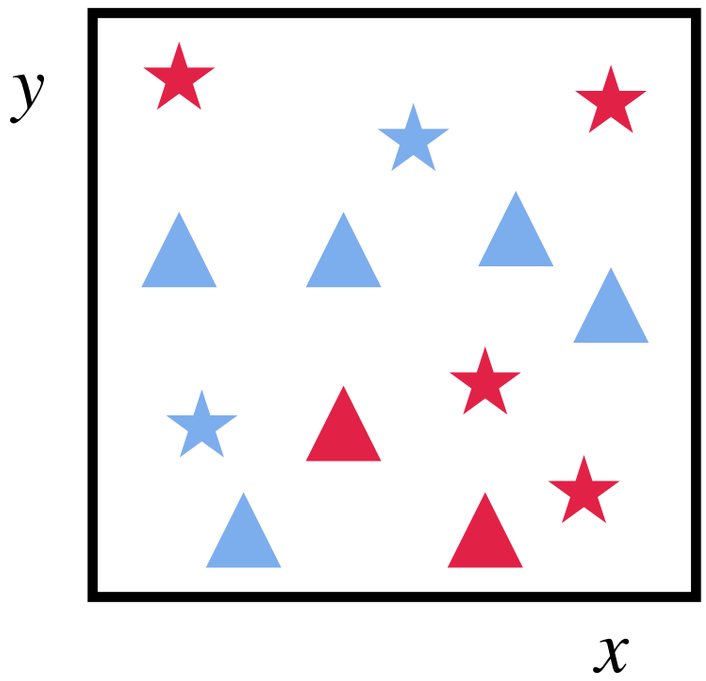


Imagine your parameter space in 2D (x,y)

How do we know a machine learning model is working?

Training vs testing data

ML methods



Pick out training points (red)
and testing points (blue)

Number of training points $d = 6$

Number of galaxy types $typ = 5$

Neural Network:

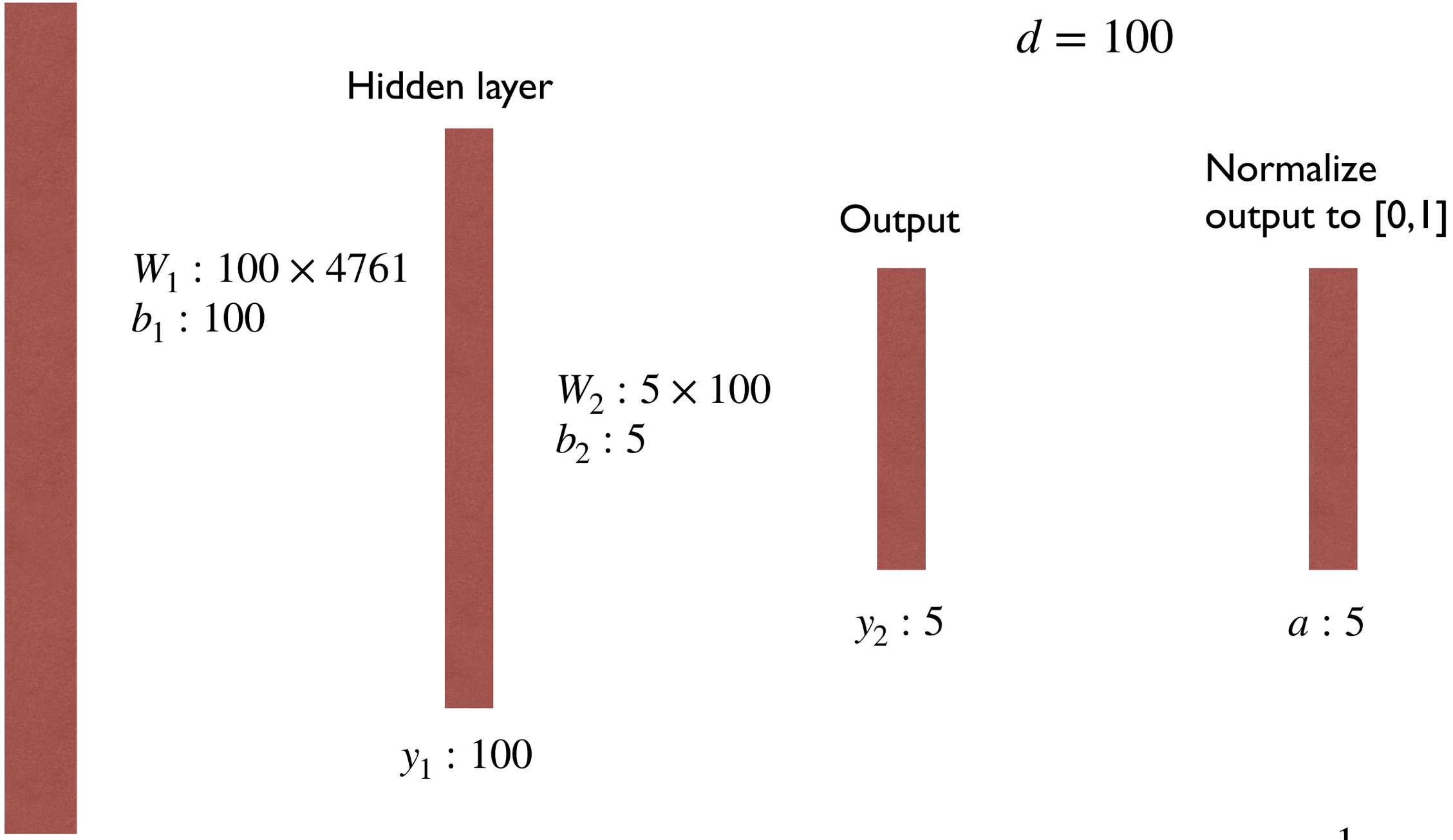
$$v_{out} = 1.0/(1 + \exp(-(W_2 \tanh(W_1 v_{in} + b_1) + b_2))$$

Input data $W_1 \rightarrow d \times 4761$ (vector from each image)

$W_2 \rightarrow typ \times d$ $\qquad$ $b_1 \rightarrow d$

Output vector $b_2 \rightarrow typ$ (5 classifications of galaxies)

A simple neural network $f : \mathbb{R}^{4761} \to \mathbb{R}^5$



$v_{in} : 4761$

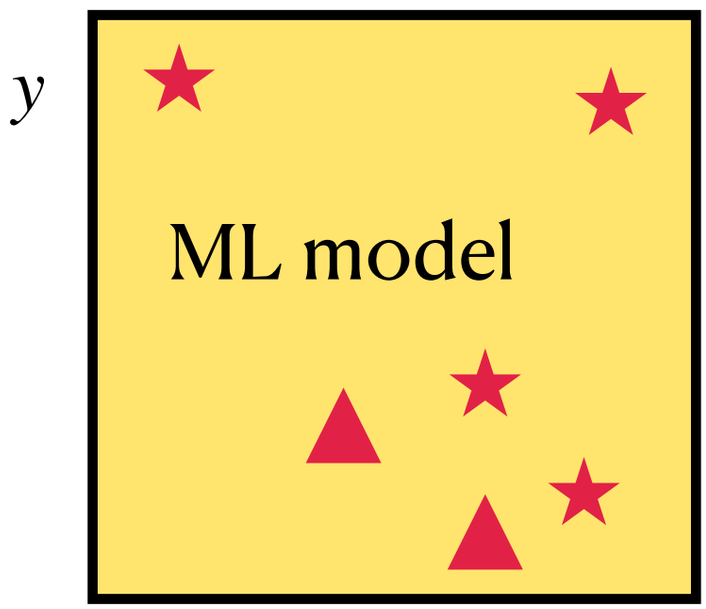
$y_1 = \tanh(W_1 v_{in} + b_1)$

$y_2 = W_2 y_1 + b_2$

$$a = \frac{1}{1 - \exp(-y_2)}$$

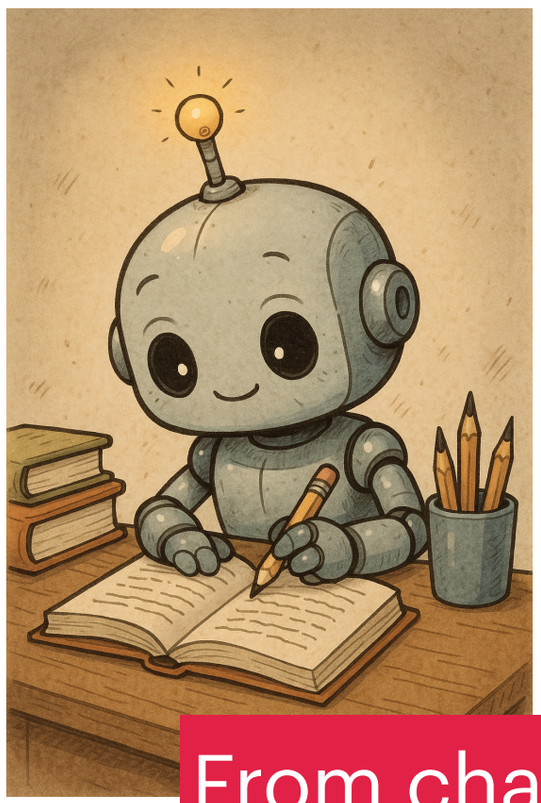
ML against the Testing data

ML methods



Train your ML model on just the training points

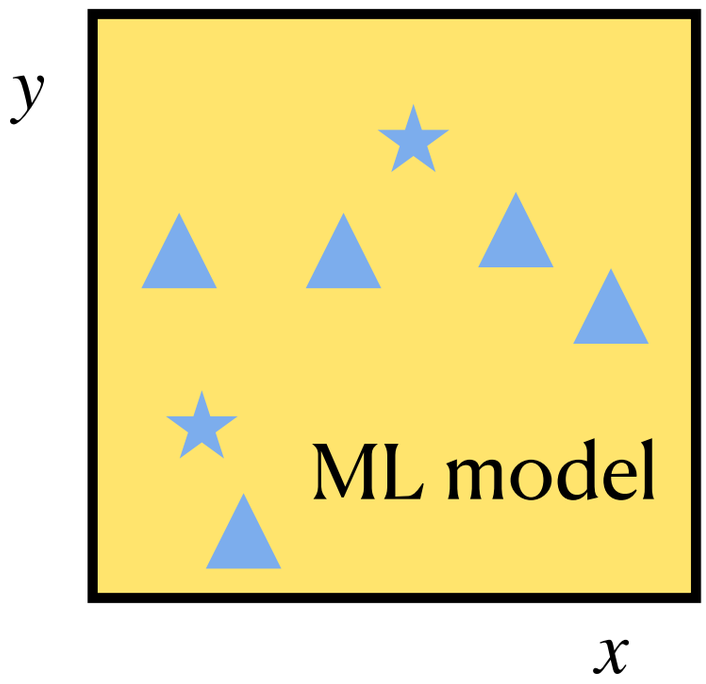
ML model "learning" about your data



From chatGPT

Cross-entropy

ML methods



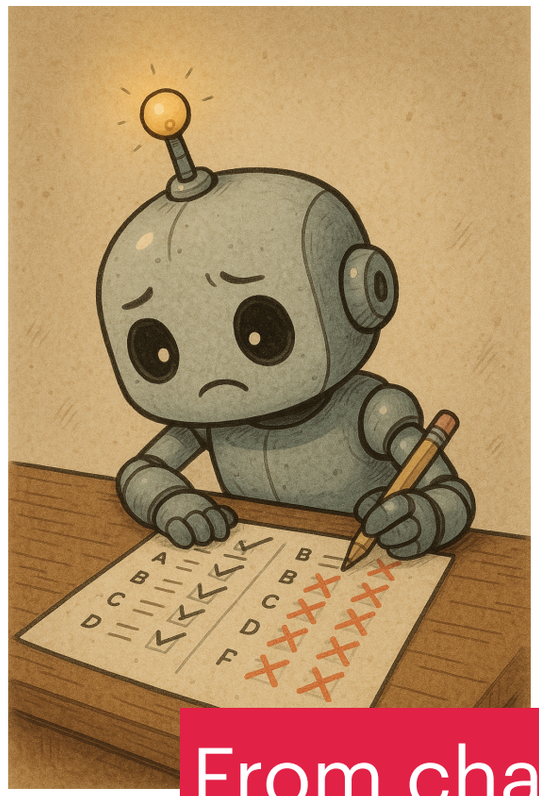
Test your ML model on the testing points, calculate the “cross-entropy”.

Want to minimize the cross-entropy!

$$\sum_i -y_i \log(a_i) - (1 - y_i)\log(1 - a_i)$$

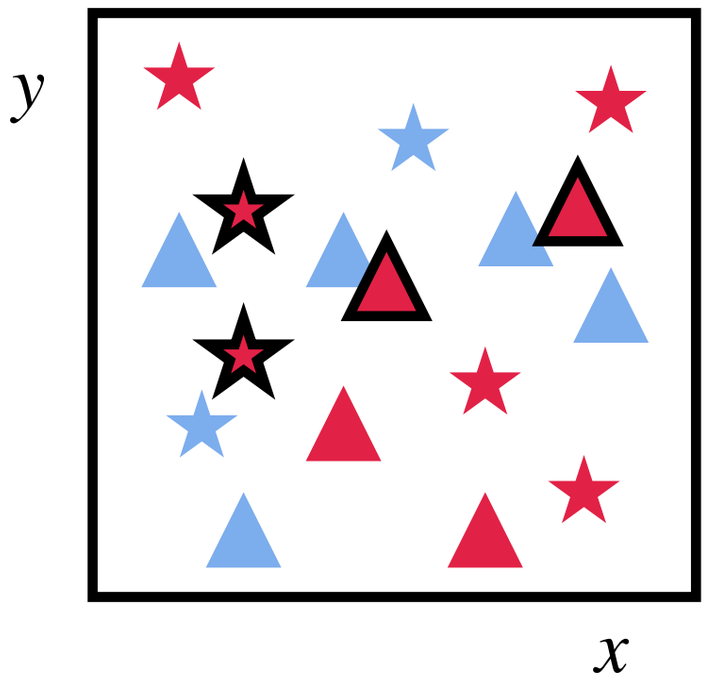
Exact Label y_i

ML label Label a_i



Calculate the gradient

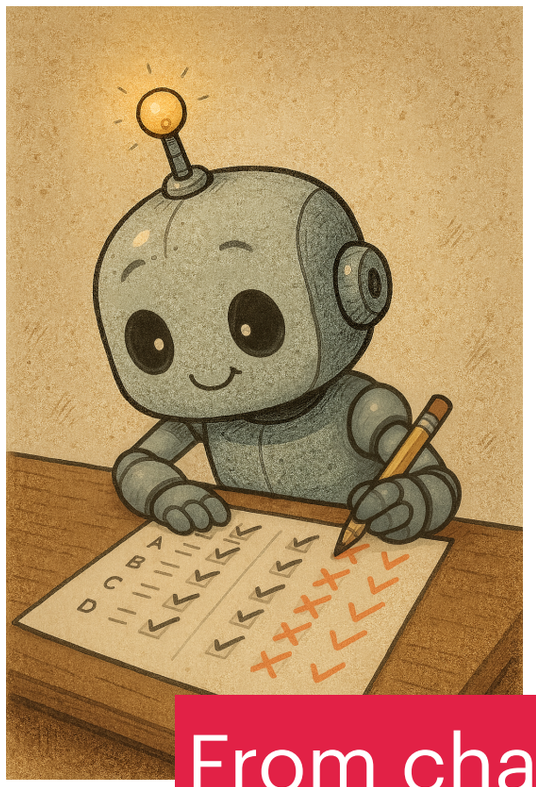
Improving accuracy



Add in new random data, recalculate the gradient

Keep doing it until you achieve the desired accuracy

CAUTION overtraining: DON'T include your testing data!



Example for accuracy

ML prediction vs truth

Prediction: a = [0.70281476, 0.62053204, 0.7327105, 0.3870779, 0.46099216]

Truth: y = [1. 0. 0. 0. 0.]

Highest probability is i=2, which is wrong! Return false.

We want to train our model to have the highest accuracy.

Loss function

Prediction: $a = [0.70281476, 0.62053204, 0.7327105, 0.3870779, 0.46099216]$

Truth: $y = [1. 0. 0. 0. 0.]$

Want to have a small loss function when the first number is large:

$-\log(a_0)$

And a large loss function when the other numbers are large:

$$\sum_{i>0} \log(1 - a_i)$$

General formula:

$$\sum_{i} - y_i \log(a_i) - (1 - y_i)\log(1 - a_i) \text{ : "cross-entropy"}$$

Optimizing the loss function

476705 total parameters

$W_1 : 100 \times 4761$

$b_1 : 100$

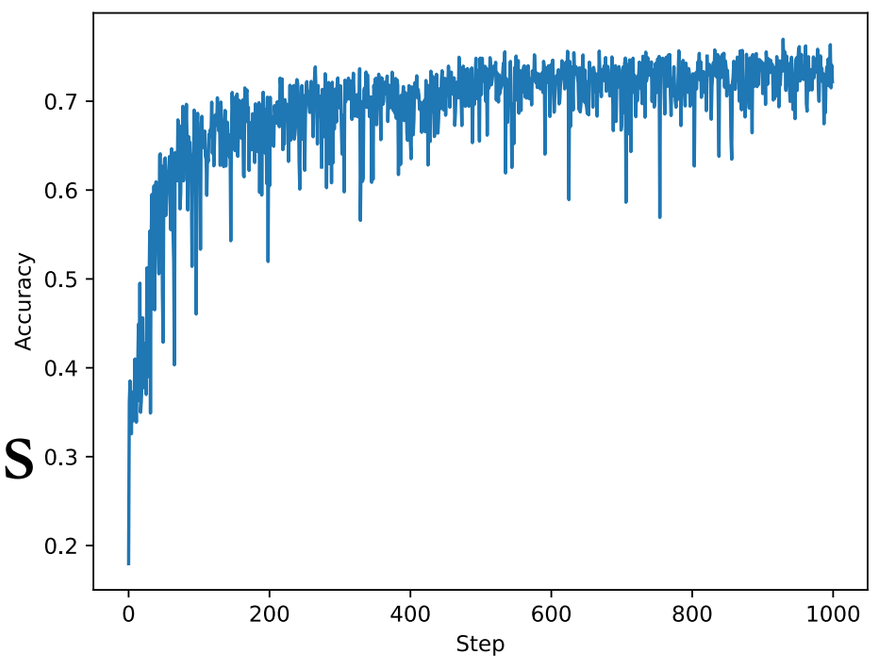
$W_1 : 5 \times 100$

$b_2 : 5$

Second derivative matrix would be 1.7 terabytes of memory.

Stochastic gradient descent:

1. Evaluate the gradient of the loss for ONE random image in the training set
2. Make a tiny change in parameters in the direction of the gradient.
3. Repeat many many times



The reason this works is pretty subtle. It's important that we do not optimize on all images in the set at once; otherwise we will overfit.

Python Pickle

General example

```python
import pickle

data = {'energy': [1.2, 3.4, 5.6], 'params': {'T': 300, 'μ': 0.1}}
with open('save.pkl', 'wb') as f:
    pickle.dump(data, f)

# Later:
with open('save.pkl', 'rb') as f:
    restored = pickle.load(f)

print(restored)
# {'energy': [1.2, 3.4, 5.6], 'params': {'T': 300, 'μ': 0.1}}
```

Converts python object →stream of bytes, writes to file “wb”

Reads in bytes, reconstructs original object “rb”

For class today:

```python
import gzip
(train_images_raw,train_labels,test_images_raw,test_labels) = pickle.load(gzip.open("dataG.pkl.gz",'rb'))
```

Using jax

Similar to numpy, but use jnp instead of np

You get:

1) just-in-time compilation (speed)
2) automatic execution on GPU
3) automatic differentiation
4) vmap

Requires a functional style

(no global variables, no side effects, can't modify arguments)

```
function = jit(function)

cpu=True

if cpu:

    jax.config.update('jax_platform_name', 'cpu')

#   jax.config.update("jax_enable_x64", True)

else:

    pass

gradient = jax.grad(function)
```

vmap

Loops over a on function $f(x)$ or "vectorizing"

Allows you to add dimensions to your arrays.

say:

```
def f(a, x):
    return a@x
```

takes in a (matrix) and x (vector) and returns a vector

```
fvmap = jax.vmap(f, in_axes=(None,0), out_axes=(0))
```

takes in a (matrix) and x (N vectors) and returns N vectors

More efficient for parallelization! GPUs...