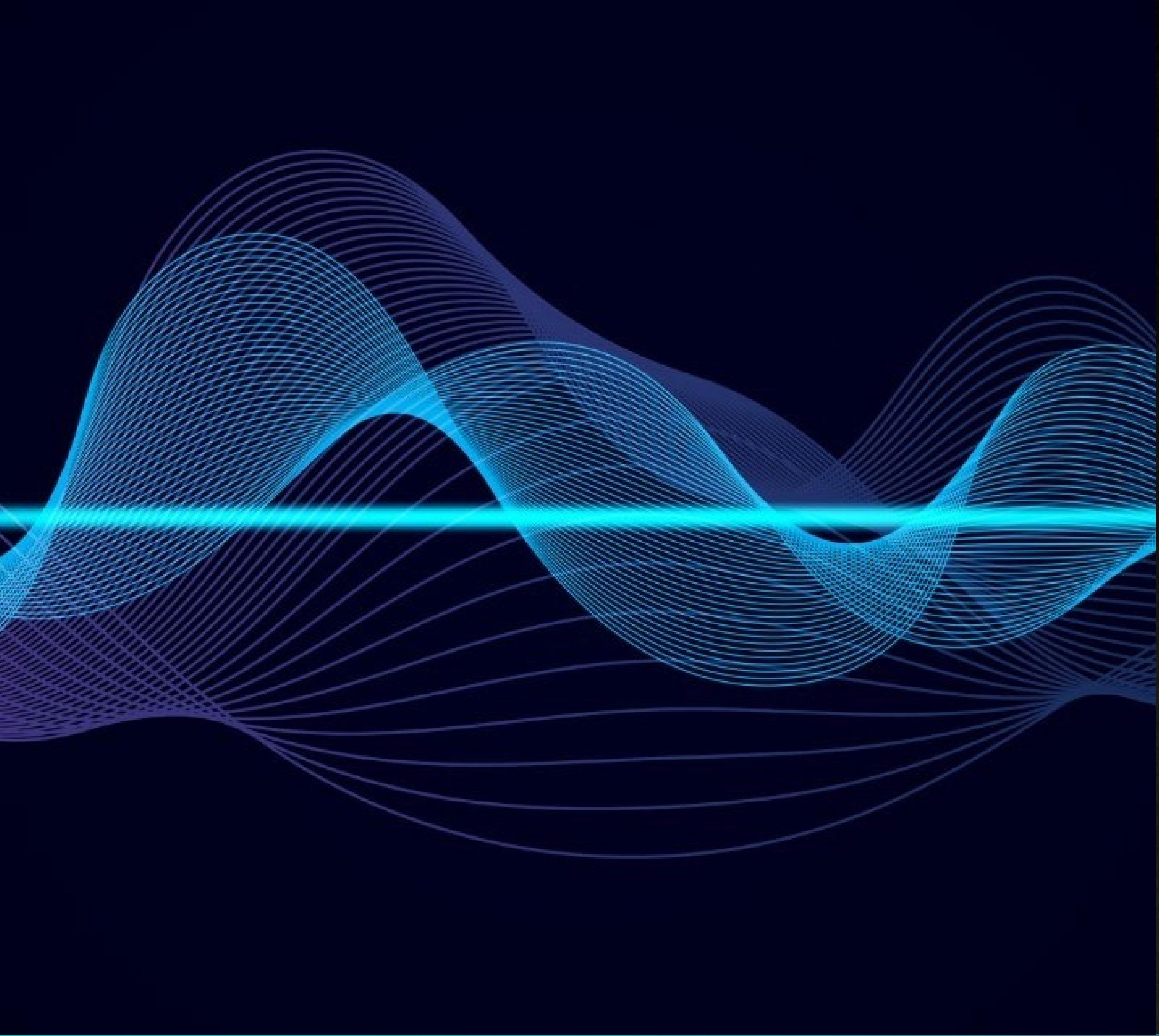


Computational Physics

PHYS 6260

Machine Learning: Neural Networks II

Announcements:

- Spring Break next week!
- Project progress report: Due Friday 3/28

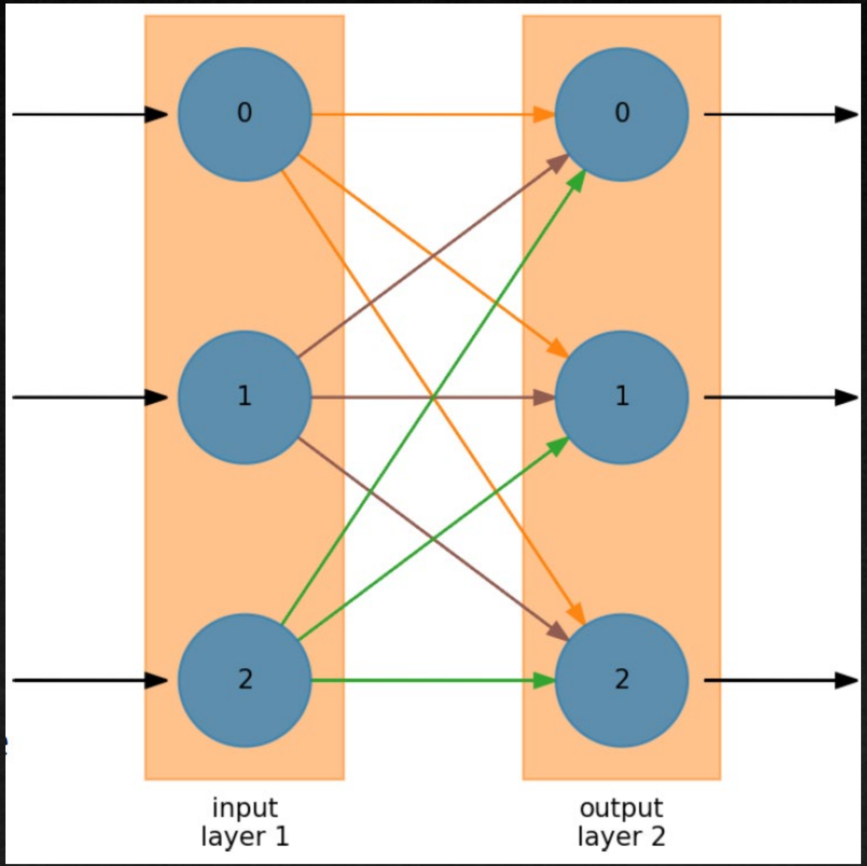
Lecture Outline

We will cover these topics

- Constructing a NN from scratch
- Hidden layers
- Image classification

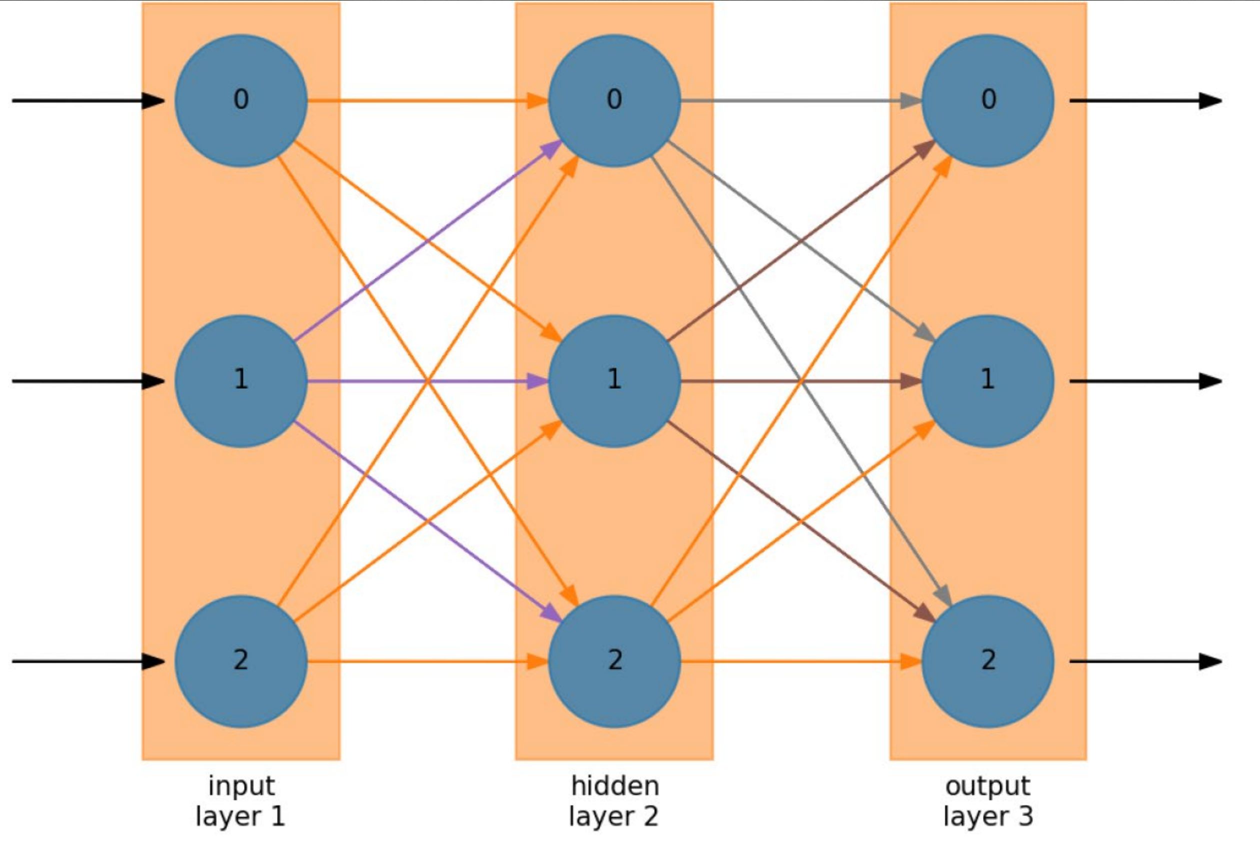
Neural network overview

- Neural networks are divided into layers
  - There's always an input layer – it doesn't do any processing – just accepts the input
  - There is always output layers
- Within a layer, there are neurons or nodes
  - For input, there will be one node for each input variable
- Every node in the first layer connects to every node in the next layer
  - The weight associated with the connection can vary – these are the matrix elements

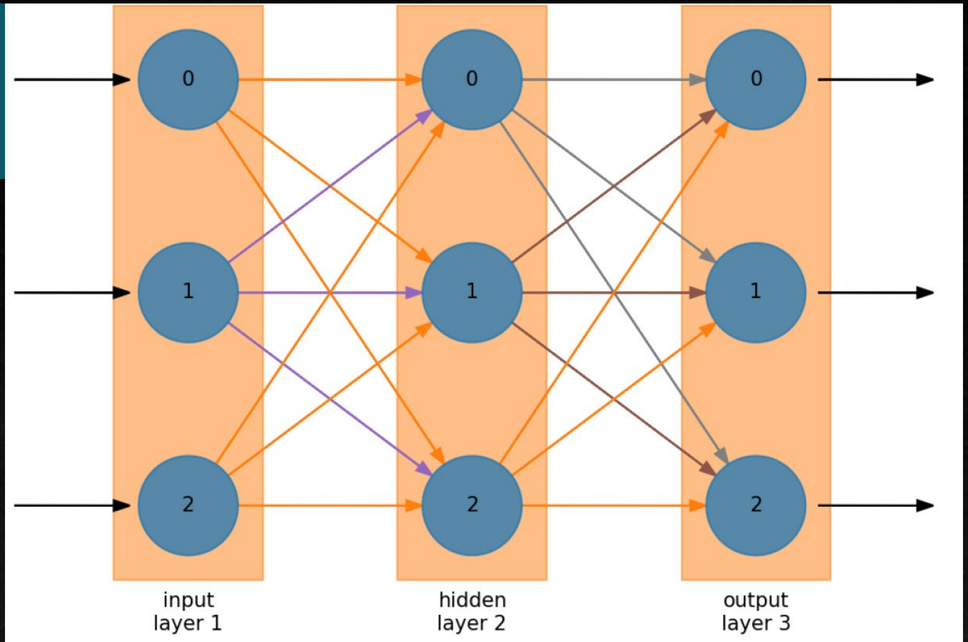


Hidden layers

- We can add more parameters by adding another layer of nodes
- Layers between the input and output are known as hidden layers
- They introduce non-linear combinations of the inputs into the model



Hidden layers

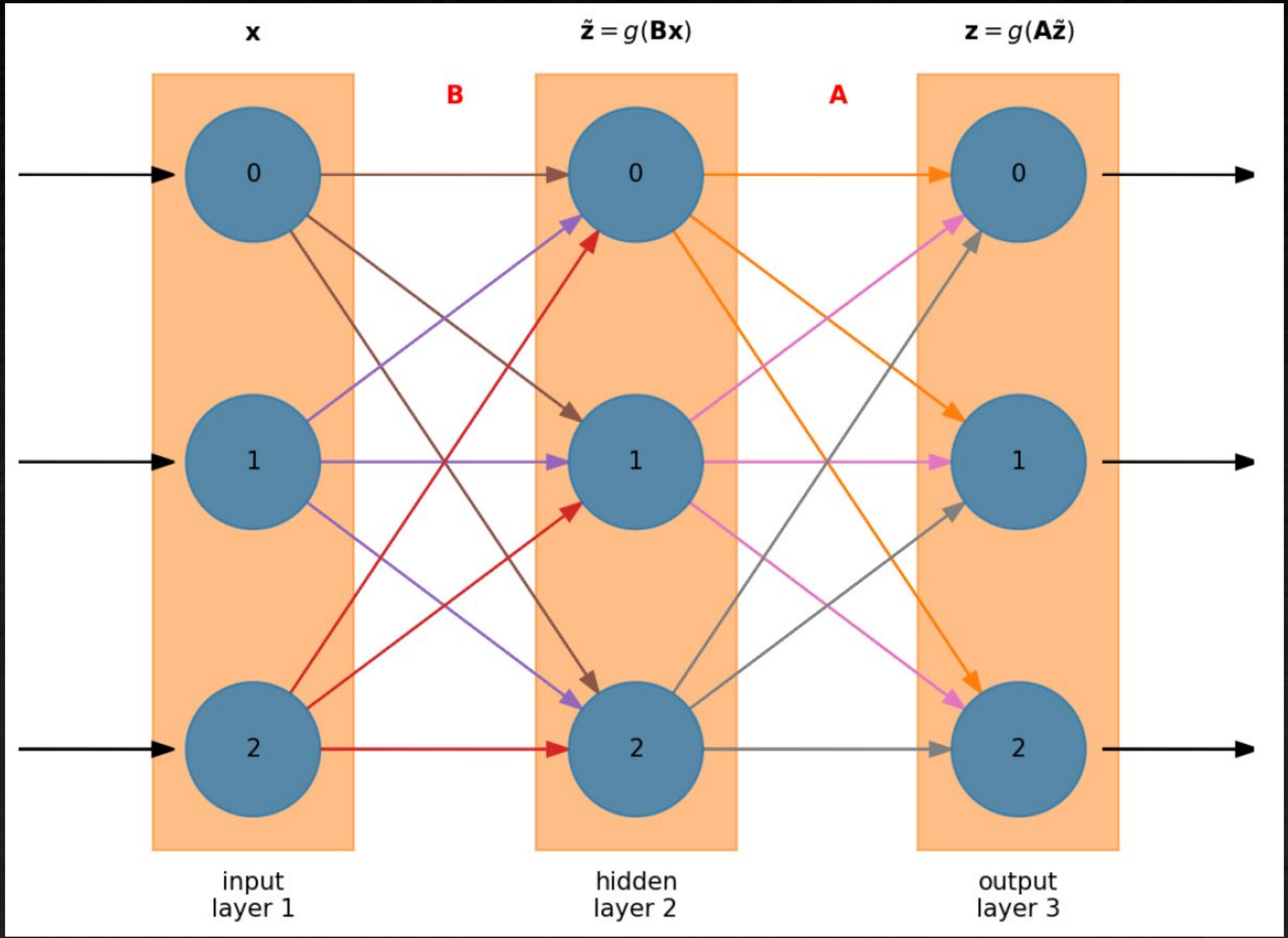


- For a hidden layer of dimension k:
  - Inputs: $x \in \mathbb{R}^n$
  - Outputs: $z \in \mathbb{R}^m$
  - **A** is an (m x k) matrix
  - **B** is a (k x n) matrix
  - The product **AB** is (m x n), as we had before without a hidden layer
- Universal approximation theorem: single layer networks can represent any continuous function
- From now on, we will not use an α, so the sigmoid functions are the same in each layer

Hidden layers

- We transform the input in two steps:

$$\tilde{z} = g(Bx)$$
$$z = g(A\tilde{z})$$



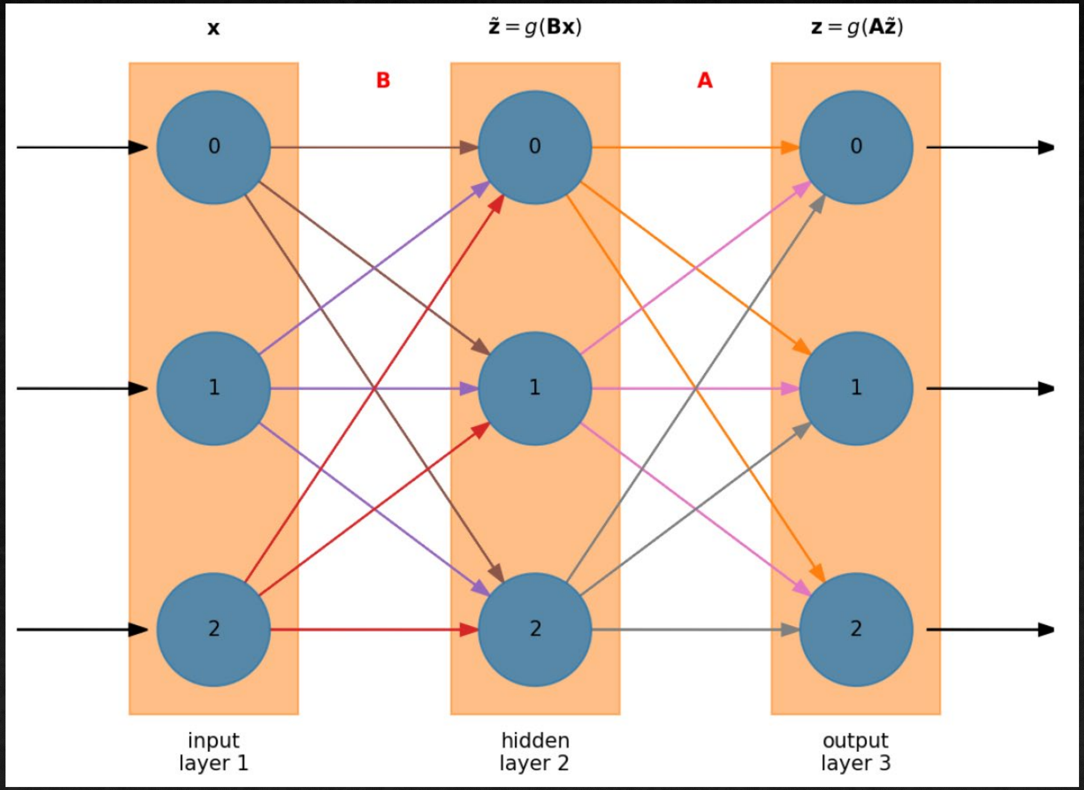
Hidden layers

- Minimize to find the A, B matrices

$$f\left(A_{ls}, B_{ij}\right) = \sum_{l=1}^{m} (z_l - y_l)^2$$

$$\tilde{z}_i = g\left(\sum_{j=1}^{n} B_{ij} x_j\right)$$

$$z_l = g\left(\sum_{s=1}^{k} A_{ls} \tilde{z}_s\right)$$



Hidden layers

- Minimize to find the A, B matrices

$$f\left(A_{ls}, B_{ij}\right) = \sum_{l=1}^{m} (z_l - y_l)^2$$

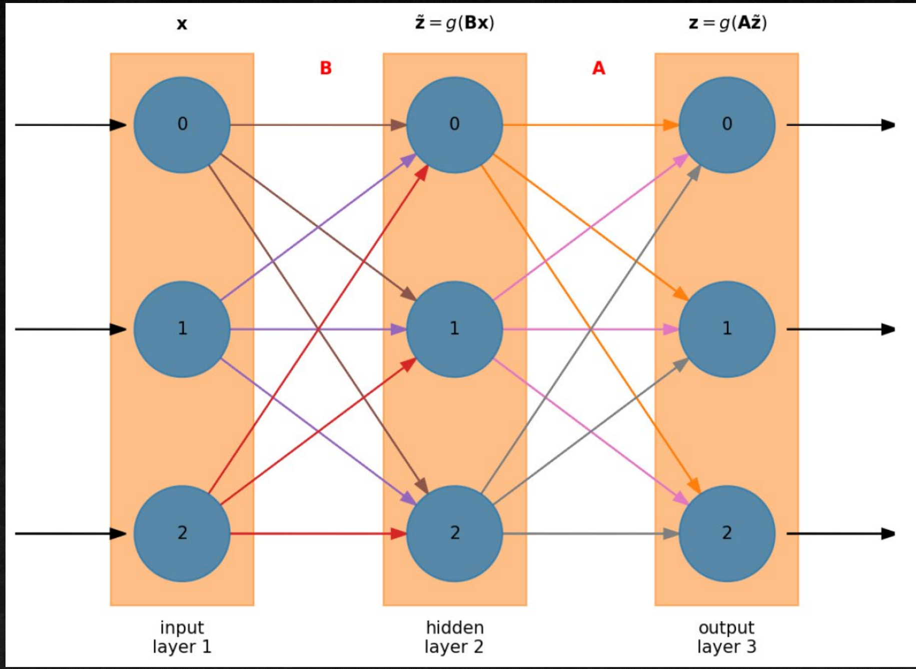
$$\tilde{z}_i = g\left(\sum_{j=1}^{n} B_{ij} x_j\right)$$

$$z_l = g\left(\sum_{s=1}^{k} A_{ls} \tilde{z}_s\right)$$

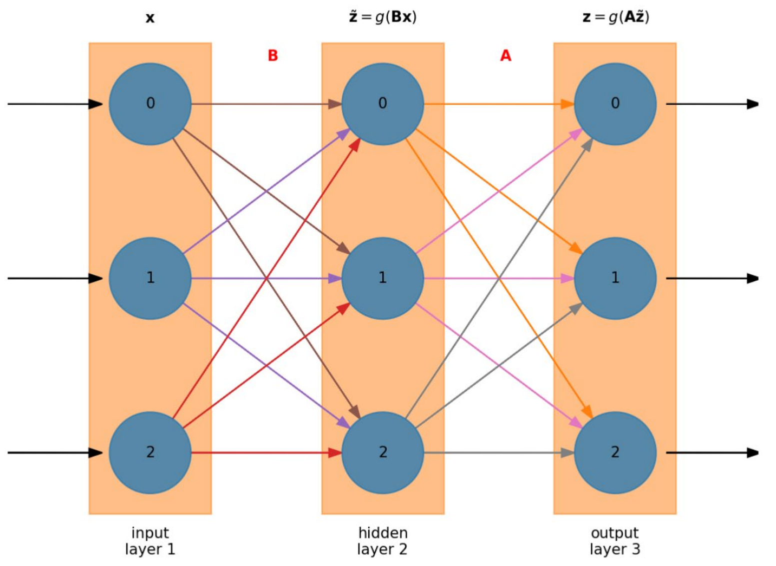
- We need to minimize both sets of weights (A and B matrices)
- In practice, we do them one at a time, with each seeing the result from its layer
- This process is called backpropagation

Backpropagation

- Backpropagation in NNs use the errors at the end to change the weights that came in earlier in the network
- In the evaluation step, we progress through the NN in a forward direction (input → hidden layers → output)
- Backpropagation is the process of taking the errors that we compute at the output layer and moving them backwards to the hidden layer



Backpropagation: gradient descent



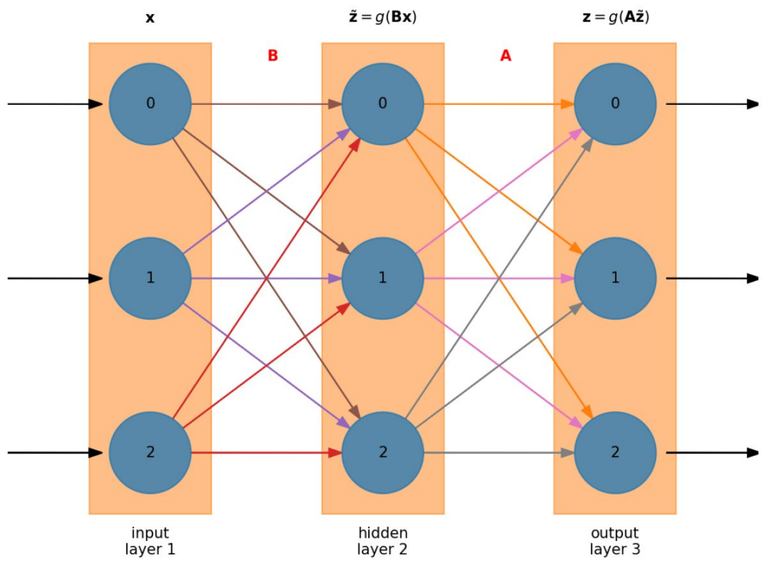
- Perform gradient descent on **A** and **B** separately
- This is the strength of backpropagation and descent versus some "canned" minimization routine – we are not optimizing the entire system
- Differentiating our error $\vec{e}$ through numerous chain rules gives:

$$\Delta A = -2\eta\, \vec{e} \circ \vec{z} \circ (1 - \vec{z}) \cdot \tilde{z}^T$$
$$\Delta B = -2\eta\, \tilde{e} \circ \tilde{z} \circ (1 - \tilde{z}) \cdot \vec{x}^T$$

where

$$\tilde{e} = A^T \vec{e} \circ \vec{z} \circ \left(1 - \vec{z}\right) \approx A^T \vec{e}$$

Hidden layers



- Usually only a single hidden layer is needed

- In general, you want fewer nodes in your hidden layer(s) than in your input layer
  - Reasonable choice: (n) inputs > (k) hidden nodes > (m) outputs

- Interactive hidden layers: https://playground.tensorflow.org/

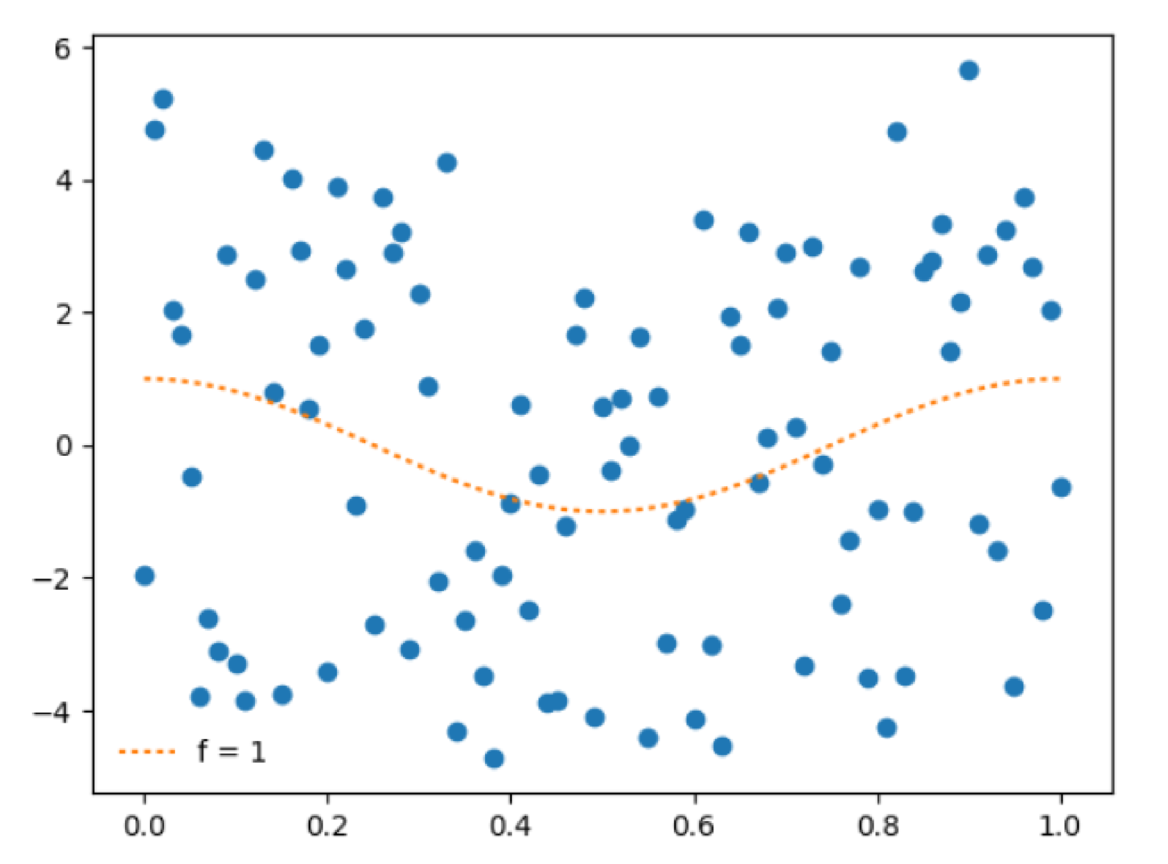
Another example: signal analysis (`signal_test_m4`)

- Consider a noisy signal that we expect to lie in 1 of 4 frequency bands: f = {1, 2, 3, 4}
- The clean signal should be: $s = \cos(2\pi f t)$
- We are given *n* points in the form
$$x_i = \cos(2\pi f t) + 5r_i$$
- $r_i \in [-1, +1]$ is a random number, making the signal-to-noise ratio small
- We will have 4 outputs with a 1 in the position corresponding to the frequency, e.g. 1 Hz: [1, 0, 0, 0]; 2 Hz: [0, 1, 0, 0]
- We'll train a NN on known input/output pairs and then test with unknown pairs – can we recover the frequency?

Another example: signal analysis

- Here's a single frequency (f=1) sample data set
- See the very low SNR

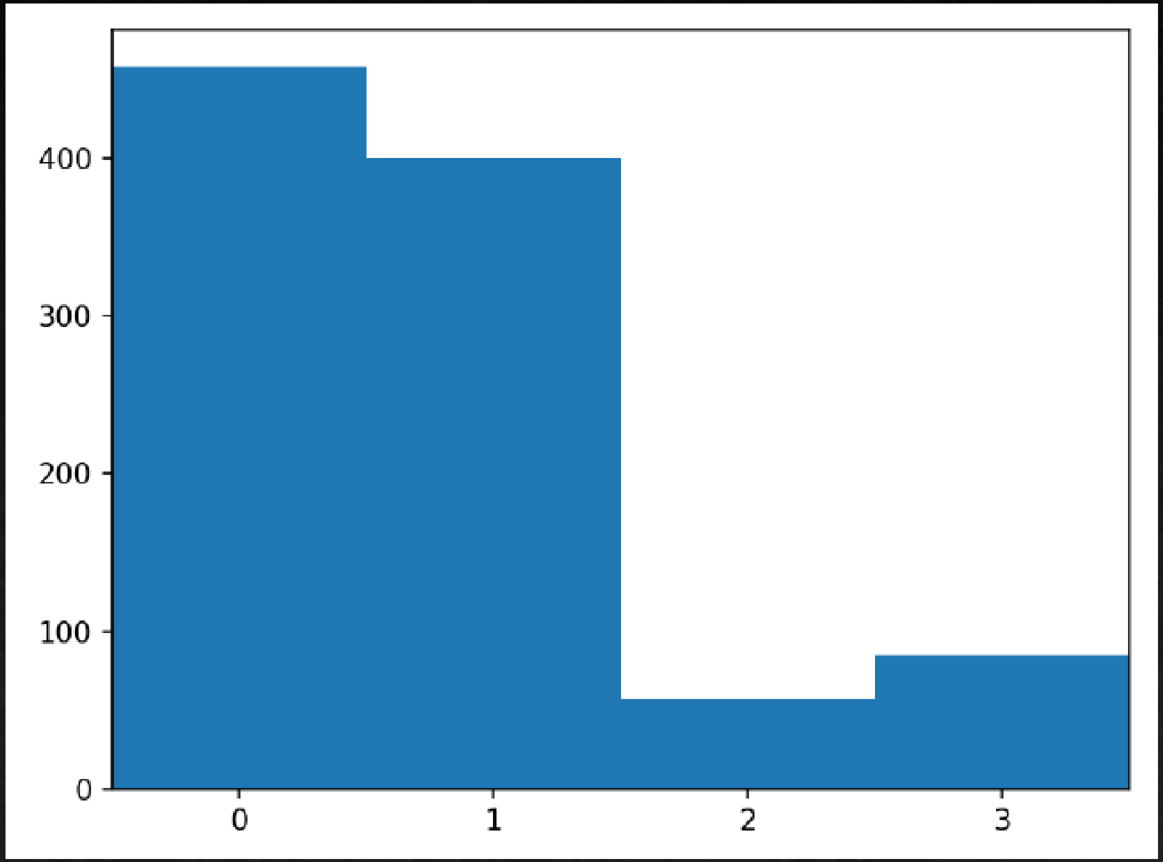
- We'll use 5 epochs to train the NN
- Learning rate $\eta = 0.05$



Another example: signal analysis

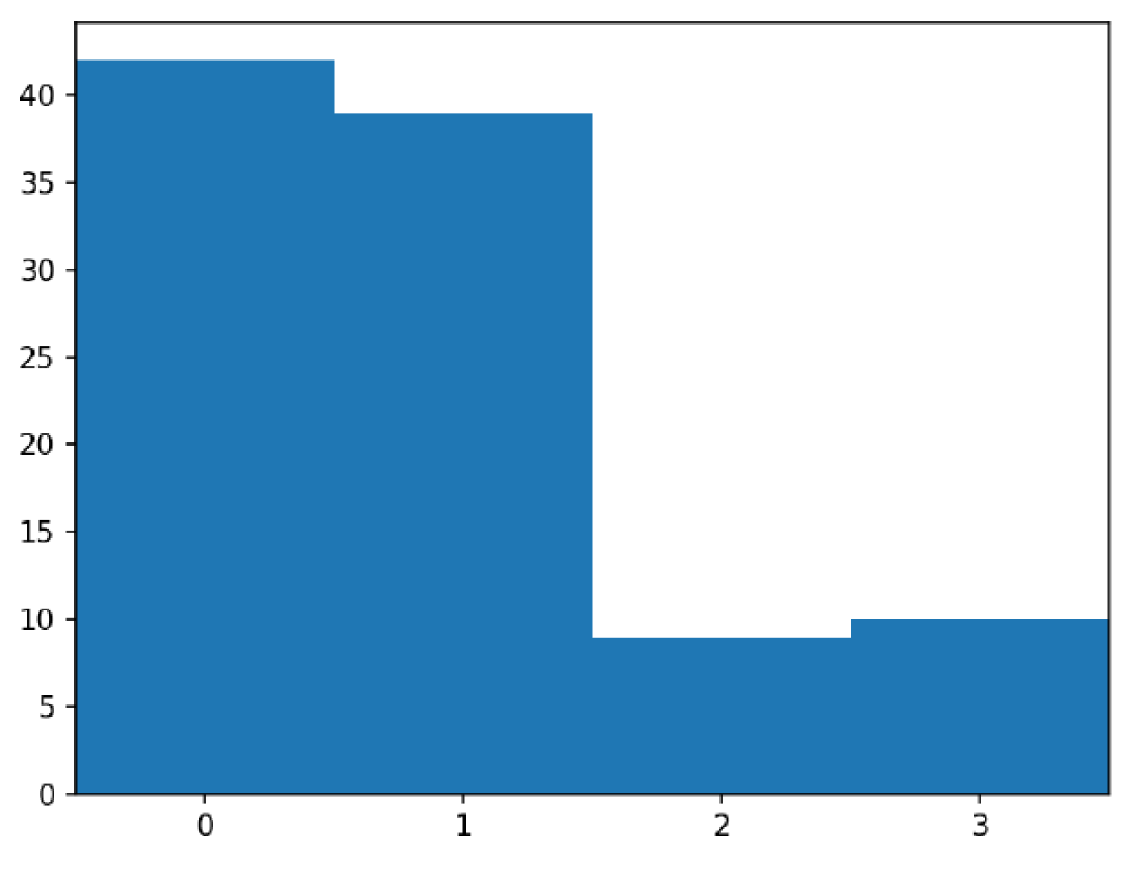
- k = 2 hidden layers
- Here is a histogram of Δf
  - Zero meaning that we predicted the frequency correctly

- 1000 random datasets in the training set



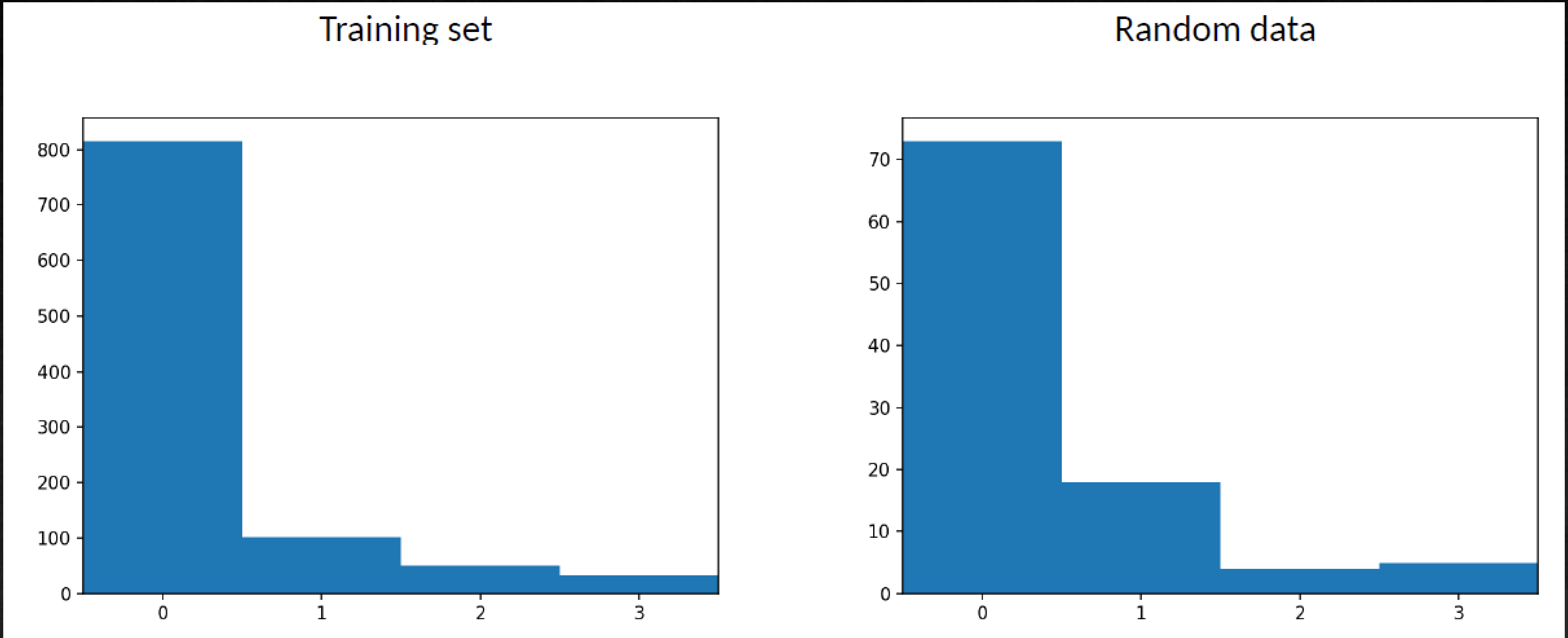
Another example: signal analysis

- k = 2 hidden layers
- Here is a histogram of Δf
  - Zero meaning that we predicted the frequency correctly
- Now on the testing datasets with 100 sets
- Let’s try to increase the hidden layers



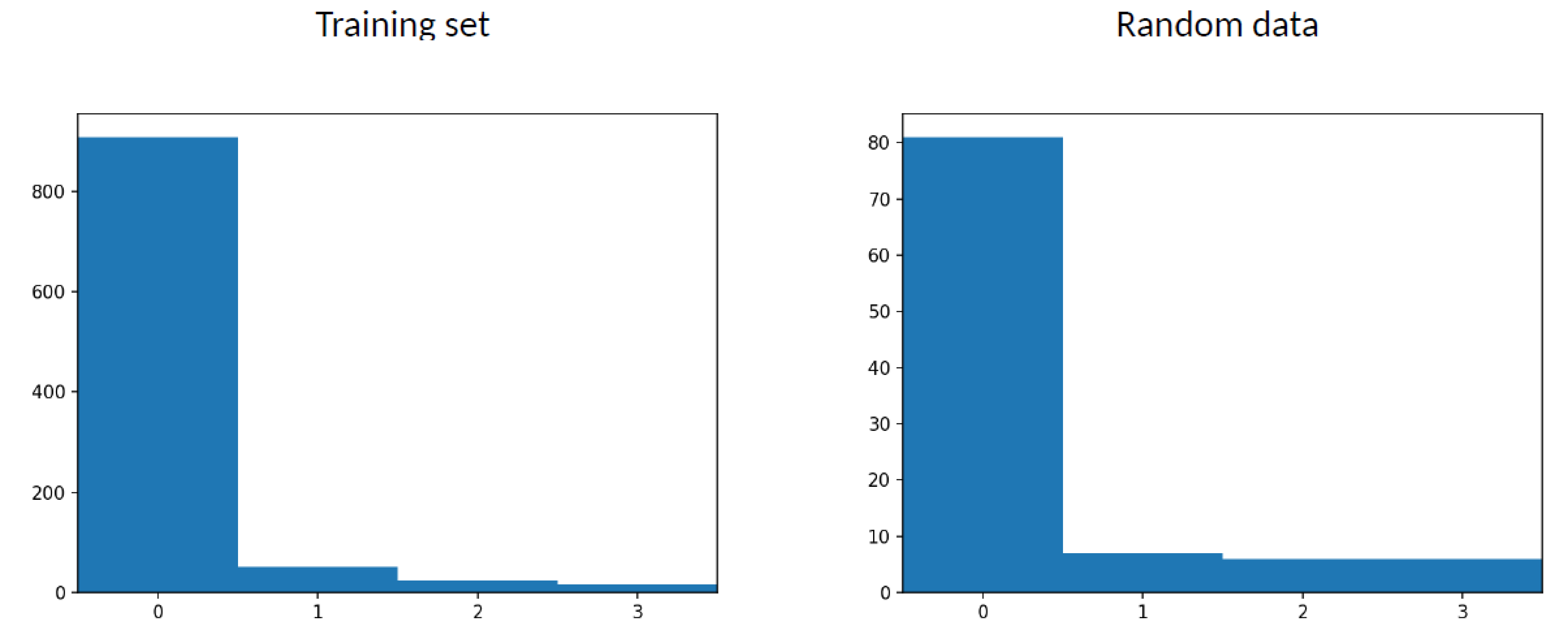
Another example: signal analysis

- k = 4 hidden layers



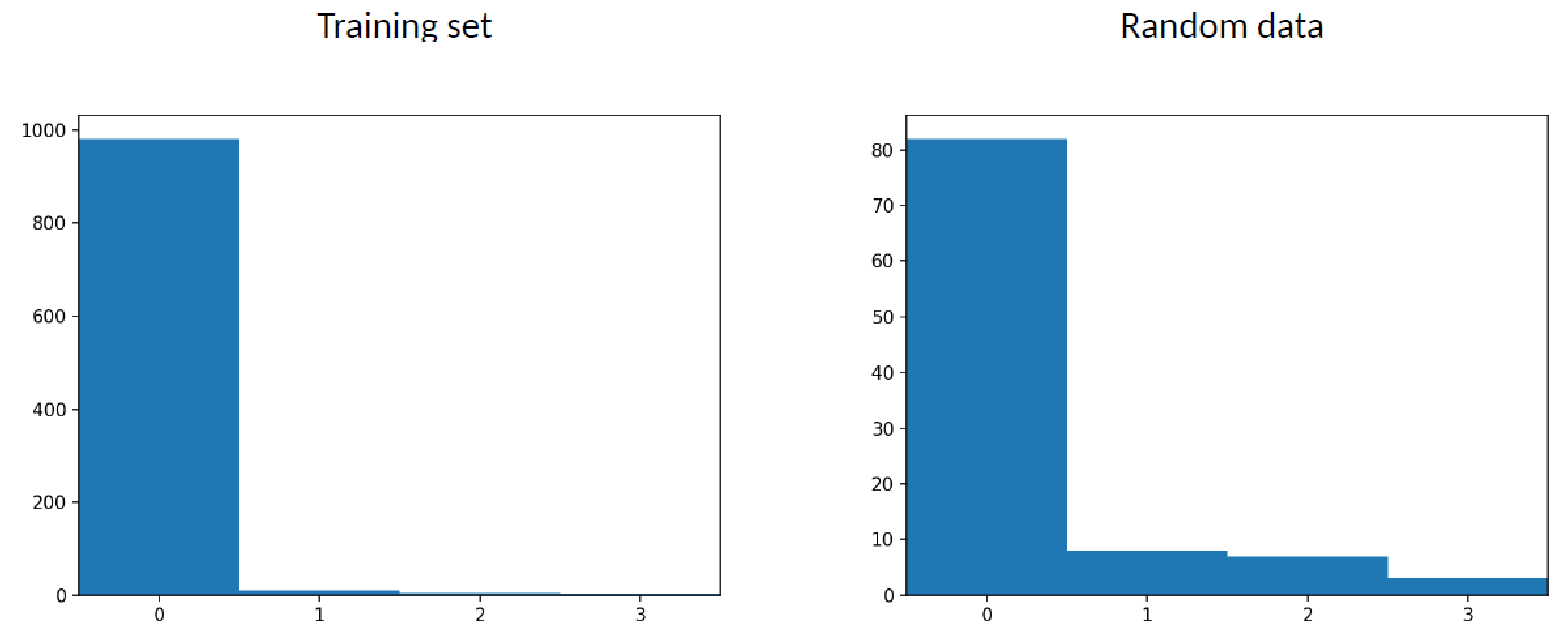
Another example: signal analysis

- k = 8 hidden layers



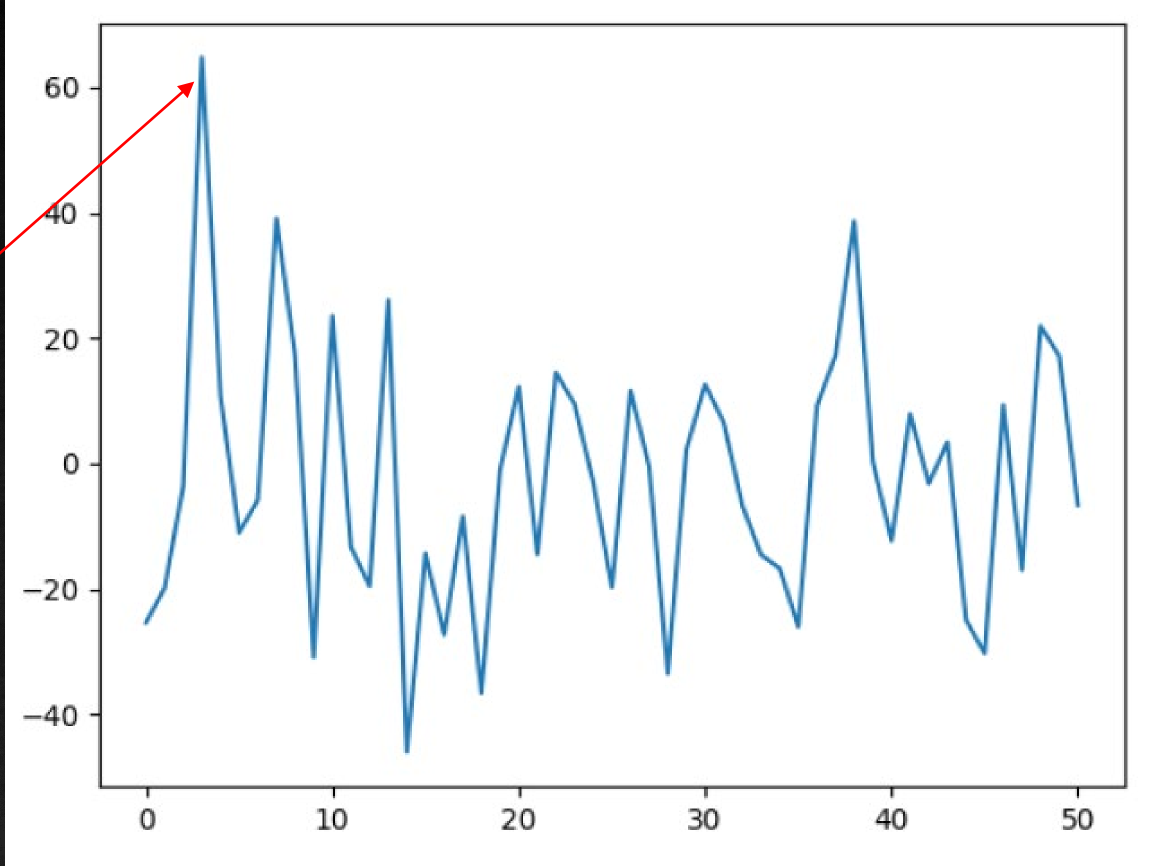
Another example: signal analysis

- k = 32 hidden layers. Much better!



Another example: signal analysis

- Is a NN the best choice for this problem?
  - ML isn't always the answer
- We could imagine doing this sample example with an FFT
- Take the FFT of the test signal and return the frequency with the maximum power



Another example: signal analysis (`fft_compare.py`)

- Perform this same FFT analysis 100 times

- It only has a 50% success rate
- But the error is not dominated by the adjacent frequency ($\Delta f = 1$)

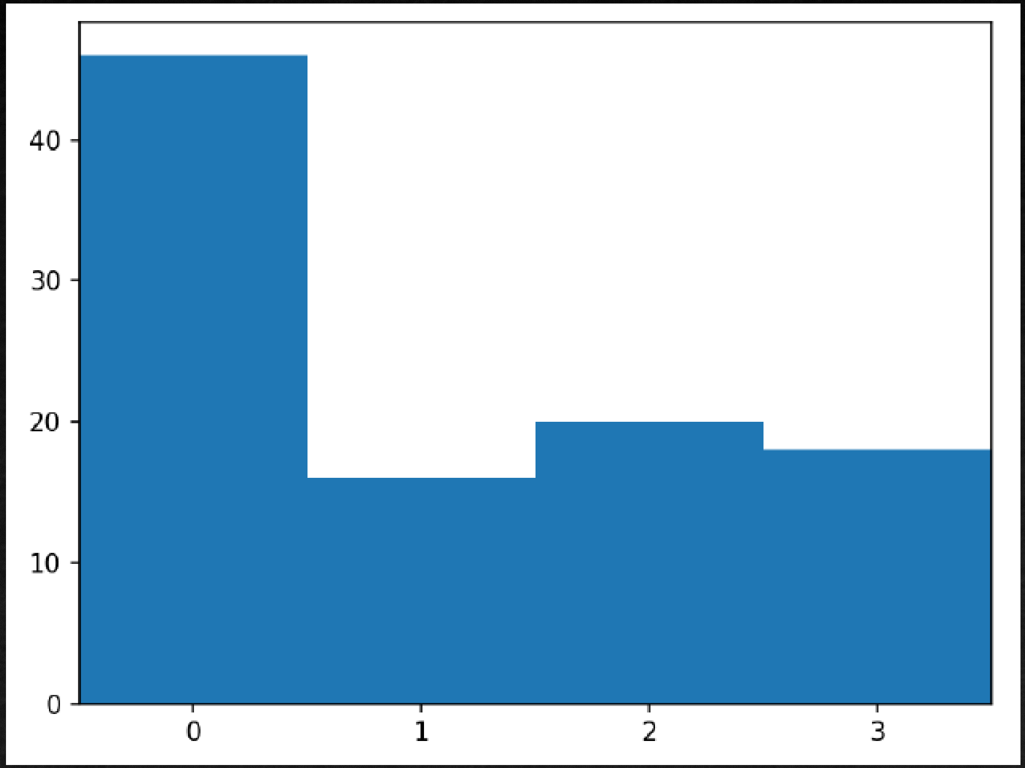
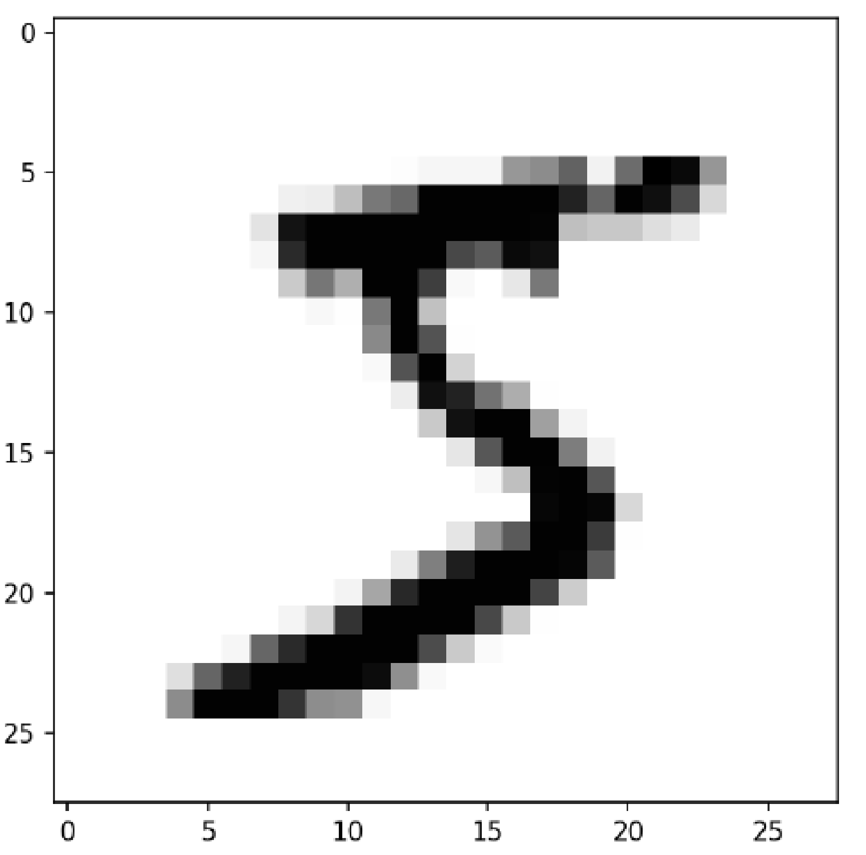


Image classification

- We'll try to recognize a digit (0—9) from an image of a handwritten digit
- Use the MNIST dataset (see Canvas since the original location is password protected now)
  - Popular dataset for testing ML techniques
  - Training (testing) set is 60k (10k) images from ~250 different people
  - Correct answer is known for both sets so we can test our performance
- Image details: 28x28 pixels, grayscale (0—255 intensity)
- The best ML algorithms can get accuracy >99%

Image classification: NN details

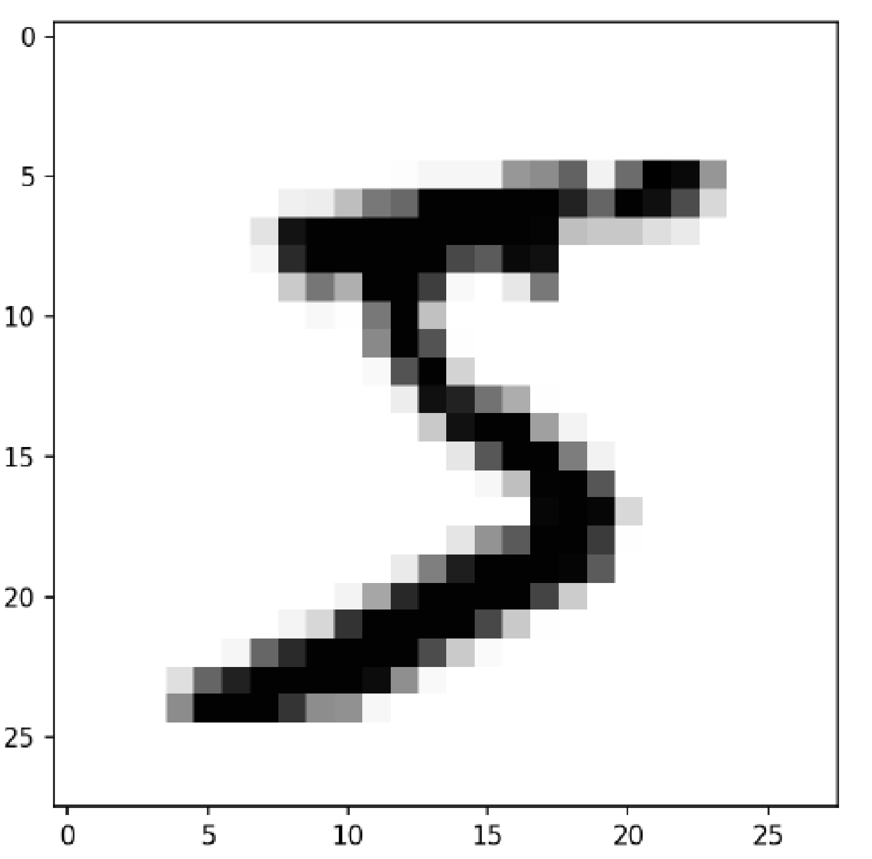
- Input layer will have 784 nodes (pixels)
- Output layer will have 10 nodes
  - Array with an entry for each possible digit
  - E.g. “3” would be represented as [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
  - We’ll start with a hidden layer size of 100
- Train on all 60k training datasets
- Test on all 10k testing datasets



First digit MNIST in the training set

Image classification: NN details

- Let's look through the code `19_char_recognition.py`
- With the default configuration in the code, we achieve 95-96% accuracy!



First digit MNIST in the training set

Image classification: some failures

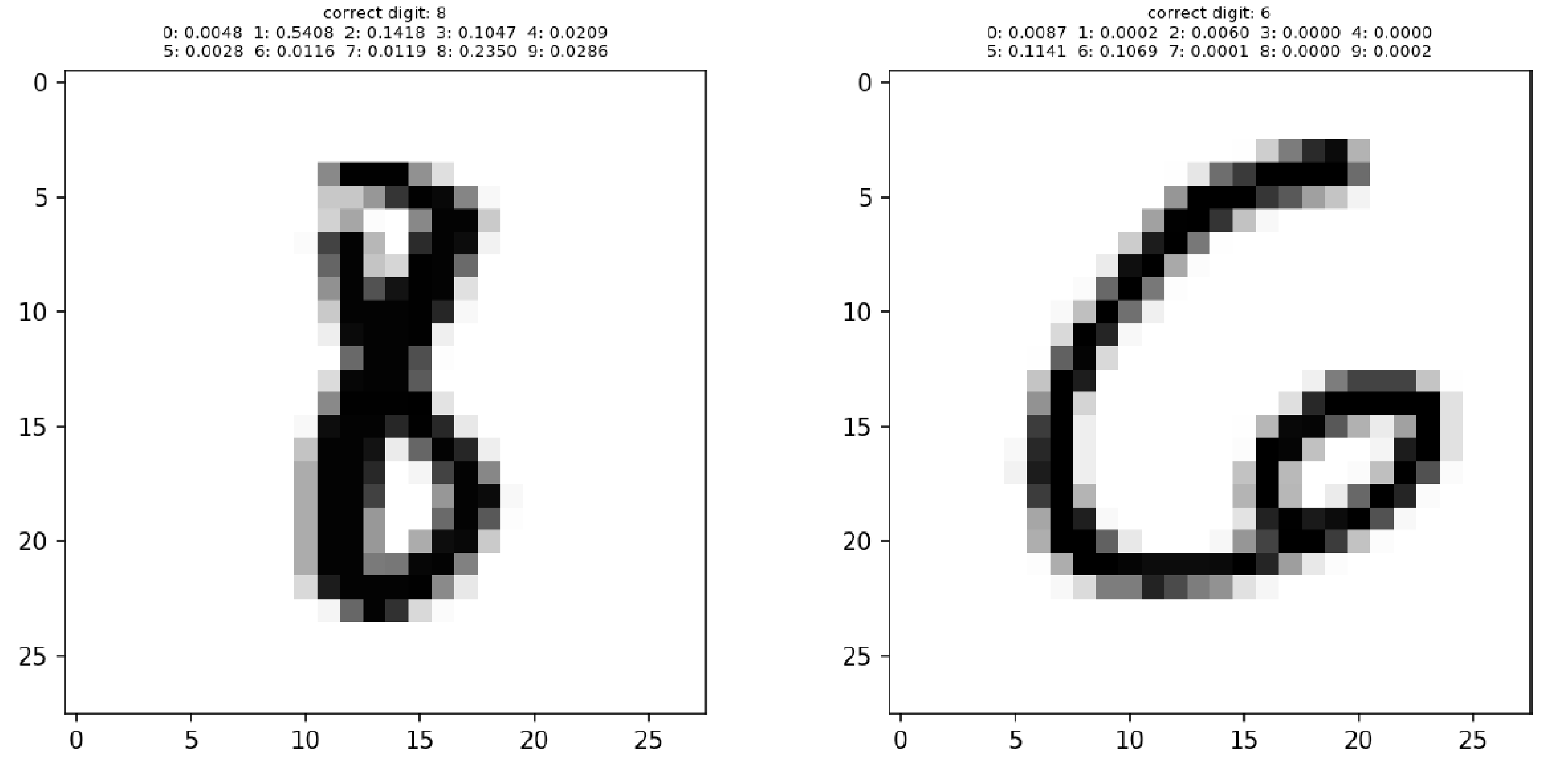
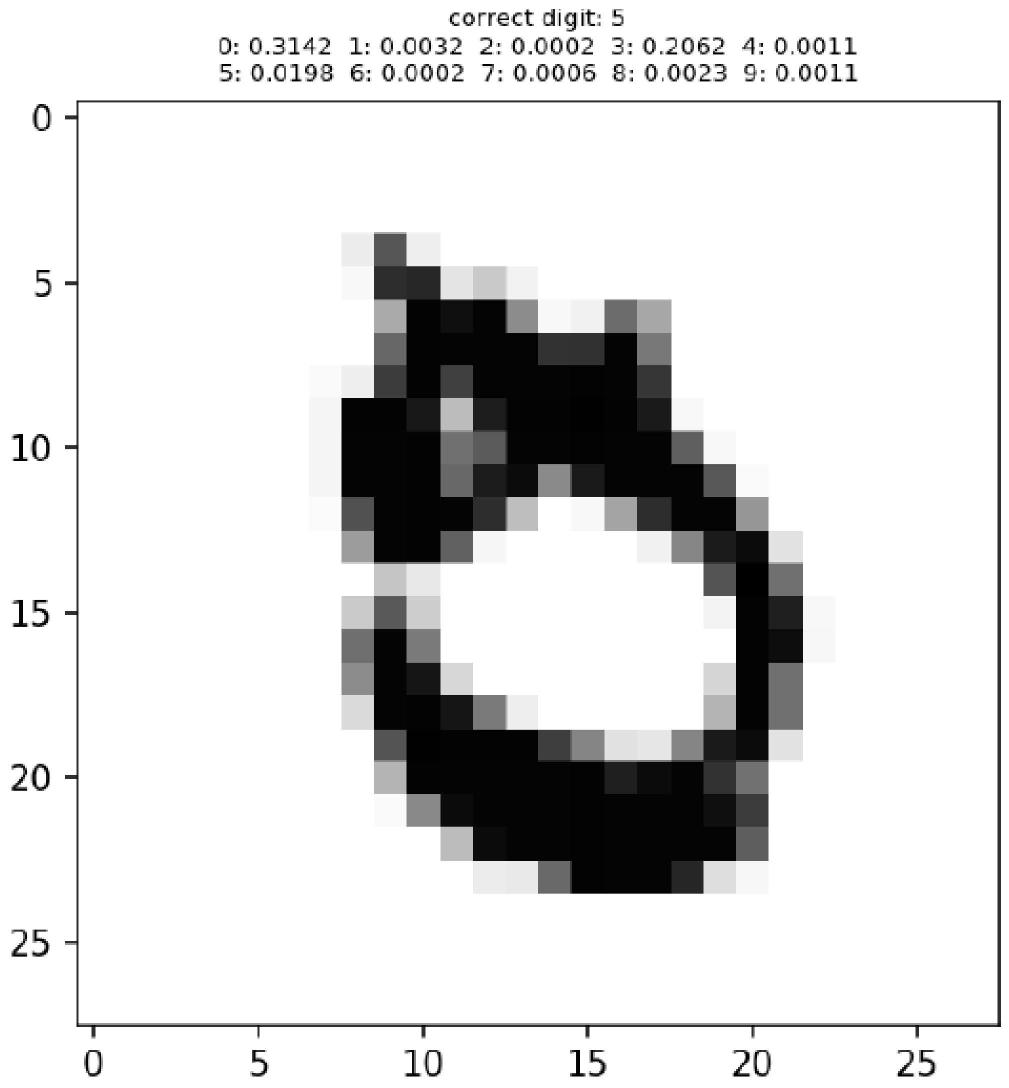


Image classification: some failures



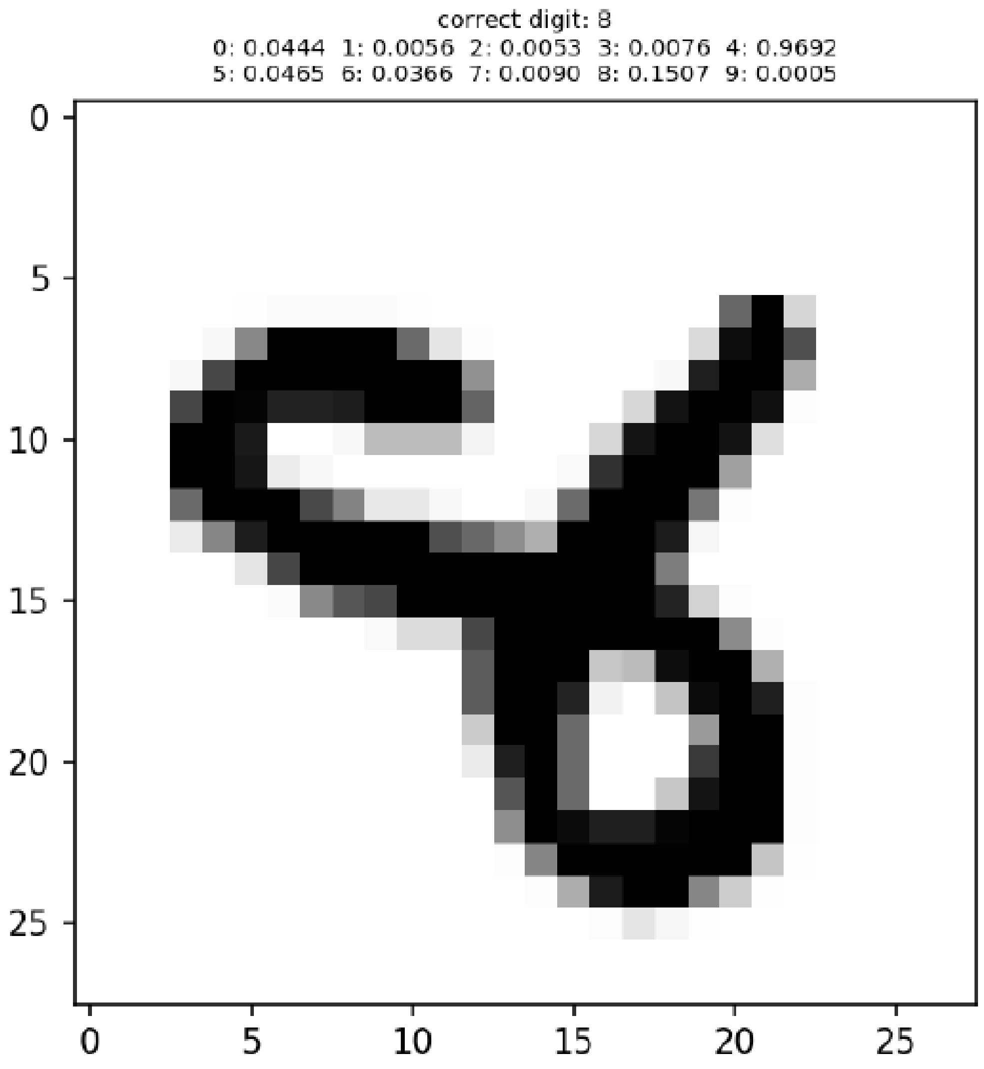


Image classification: some failures

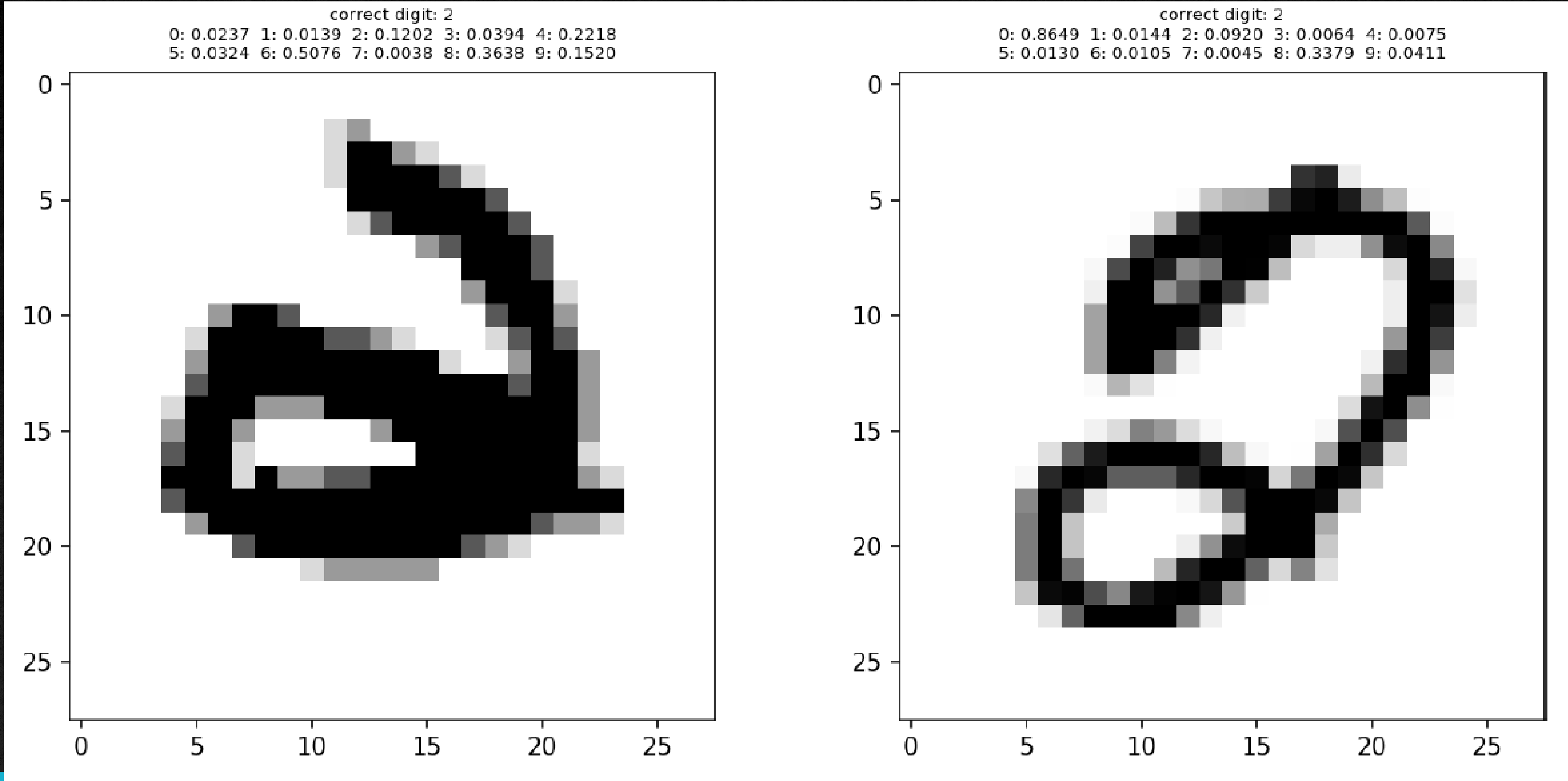


Image classification: image classification matrices

- The weights (A and B matrices) seem symmetric around 0
- Interestingly with more training, the width of the distribution seems to grow

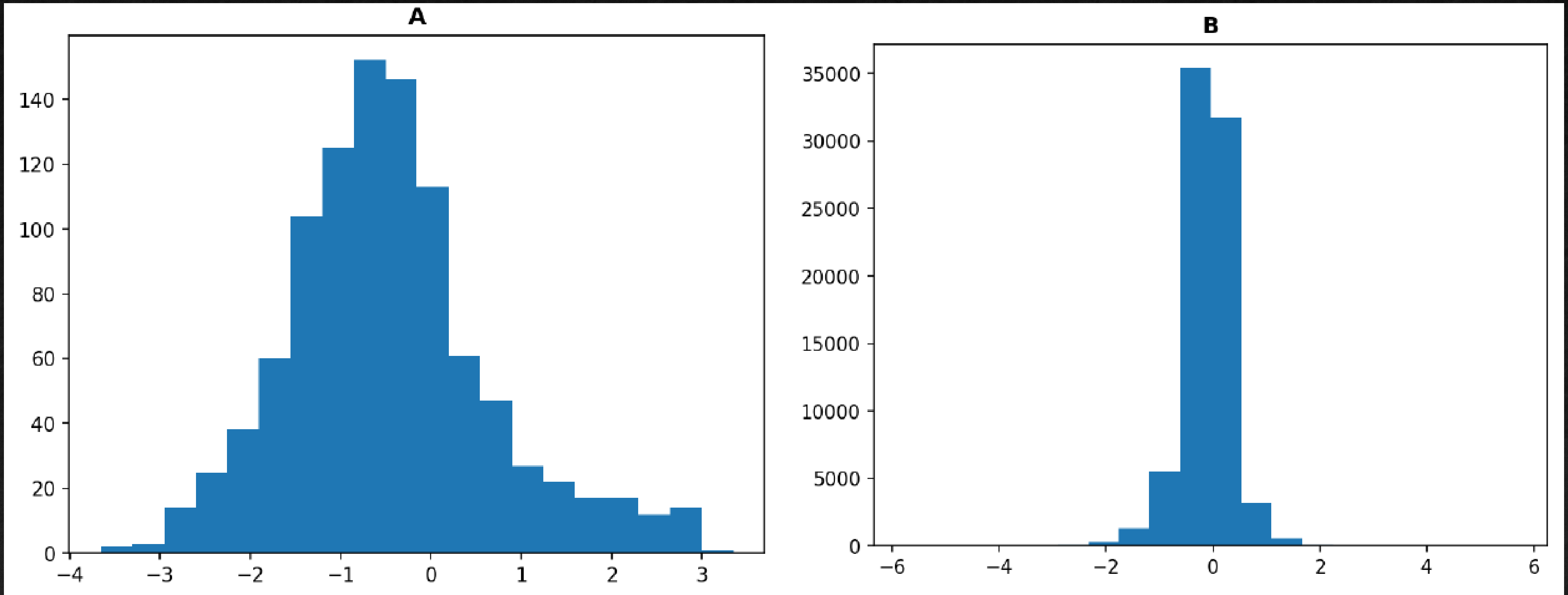


Image classification: Number of epochs

- When we use the full training set (60k images), the number of epochs (passes through the training data) doesn't matter much

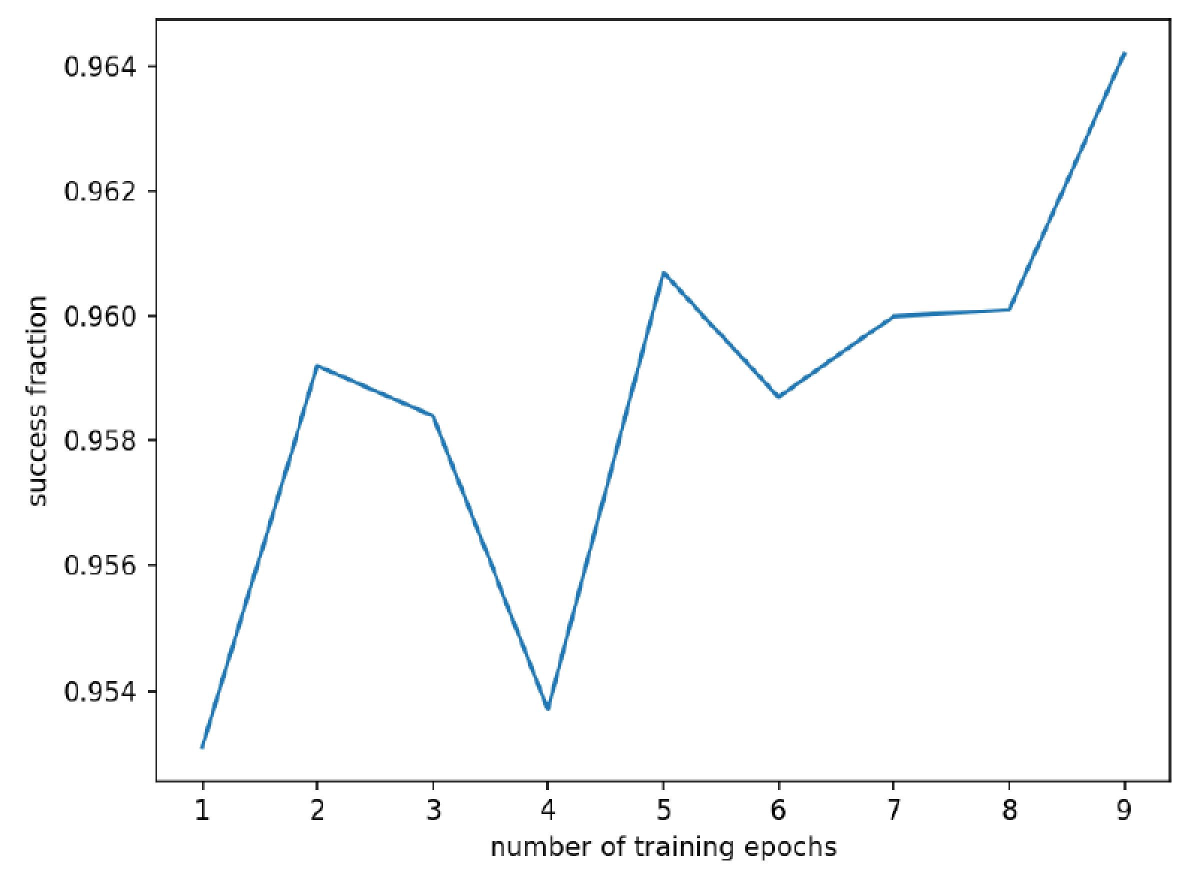


Image classification: Number of epochs

- No surprise: the larger the training set, the better the NN performs

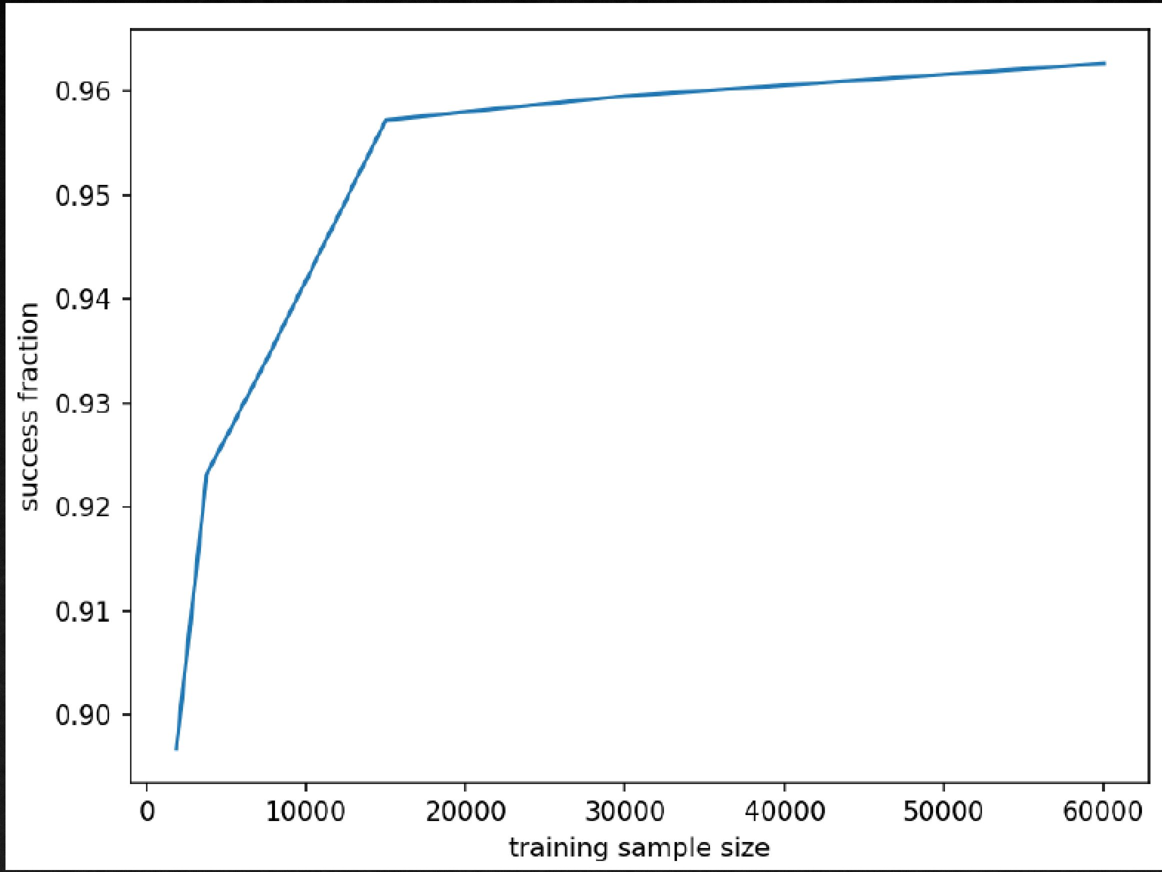
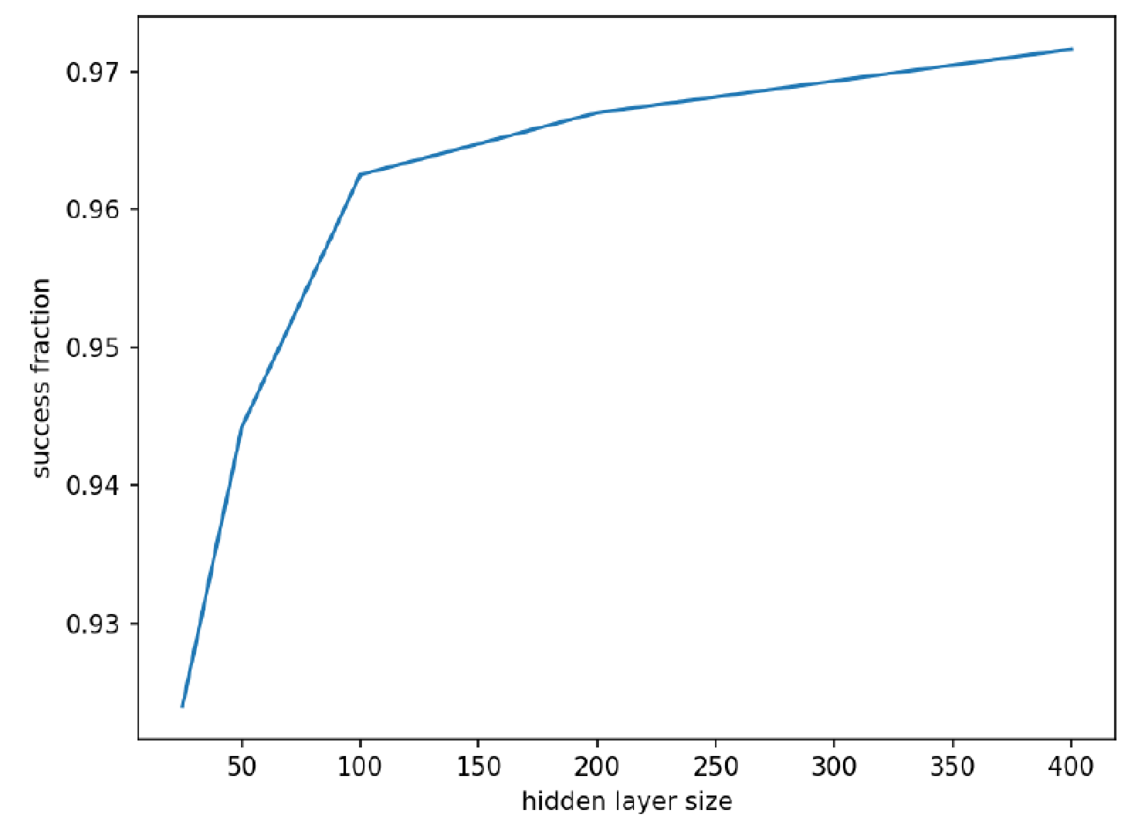


Image classification: Hidden nodes / layer

- The larger the hidden layer, the better the performance



Deep Learning (after Spring Break)

- A deep neural network is one with many hidden layers (certainly >1 hidden)
- Surprisingly nice discussion on StackExchange
- Free textbook
- Another free textbook

- There are other ML algorithms aside from neural networks
  - 2019 review of ML and physical sciences