Scientific Computing

Monday, April 27

Announcements

* Homework 6 due tonight!
* Final Exam: Monday, 5/4,
  1pm - 3pm
  Johnston Hall 417
* Final take home project assigned today, due Monday night, May 4.

Office Hours:

Mon, 9:30 - 10:30
Fri, 2:00 - 3:00

Cudahy 307

Topic 18 - Designing and Training a Neural Network

* Decide on the structure of your network.
  - How many hidden layers.
  - How big they are
  - Which activation functions (output layer will be different)
  - How many inputs/outputs?
    inputs for whatever data "features" you have (more on this later)
    outputs for whatever you're measuring

Topic 17 - Designing and Training a Neural Network

* Decide on the structure of your network.
  - How many hidden layers.
  - How big they are
  - Which activation functions (output layer will be different)
  - How many inputs/outputs?
    inputs for whatever data "features" you have (more on this later)
    outputs for whatever you're measuring

→ How do you know? Intuition, trial-and-error, general principles. e.g. smaller last hidden layer

Train/Test Split

* Use most of your data to train your network.

* Reserve some to test your network.

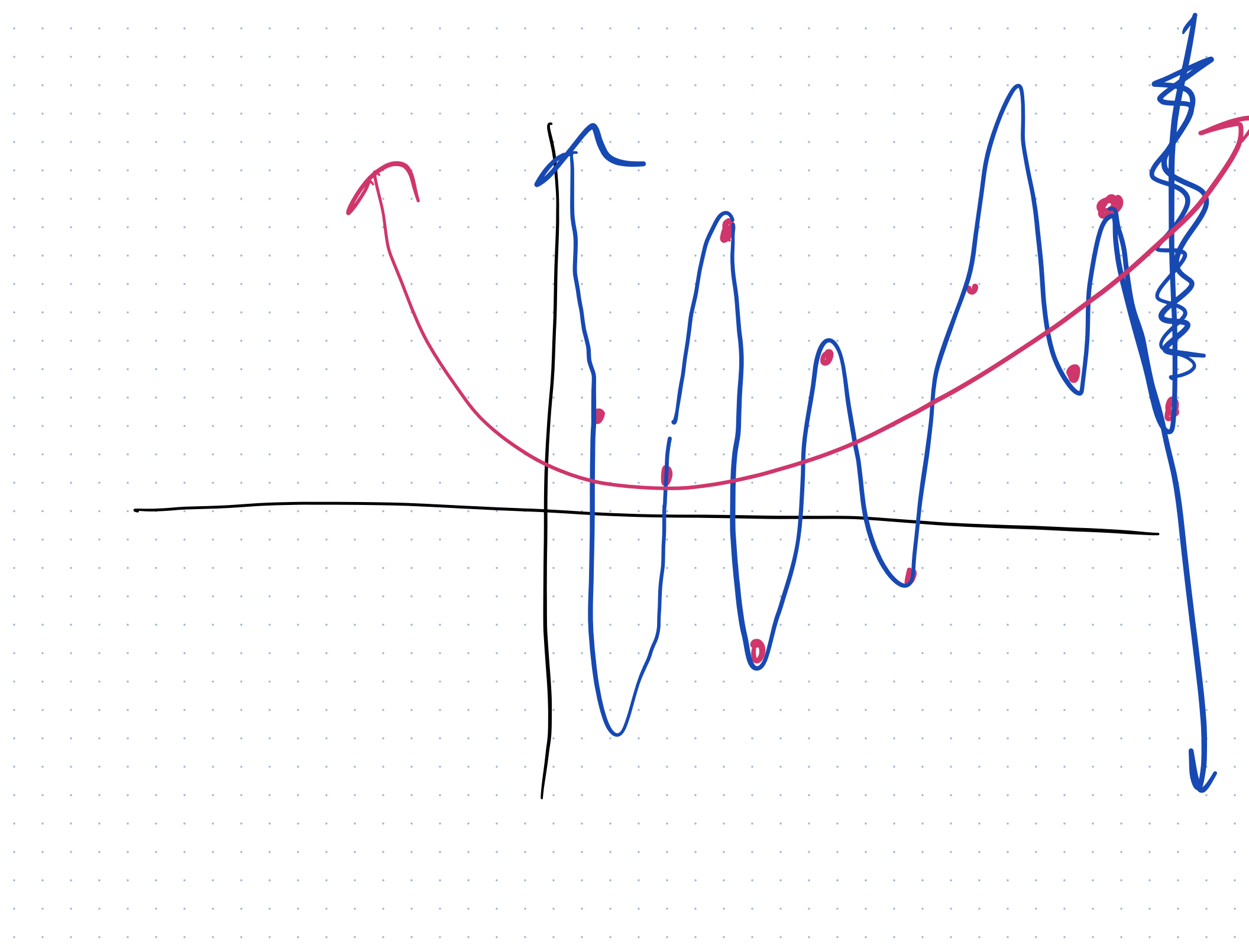
A reasonable split is 80% train / 20% test.

Never use the test data for training!

↳ Shuffle your data before you split!

If your NN is overfitting, you'll see great (low) loss on the training data, but bad (high) loss on the test set.

over training
or
memorization



Loss / Accuracy / RMSE

Loss = measure of how close actual ouput is to expected output

Training process = make loss go down

80% → Train Loss, Test Loss ← 20%

More interpretable to see if the NN is working on test data

Classification: "Accuracy" = How many predicted classes (the one with the highest prob.) are the right one?

Regression: $(\hat{y} - y)^2$ $\$^2$

"Root Mean Squared Error": Just the square root of MSE. Matches the units of the output, unlike MSE which is scaled.

$\sqrt{\quad}$ unit $\$$

Training Loop:

"Epoch" = feed all training data through one time ✓ and learn from it

Different Ways — "Optimizers"

(1) Gradient Descent: Feed through all training data in one big batch, backpropagate back, update weights and biases by $L \cdot (-\nabla)$

no mini batches

L = "learning rate"

$L = \frac{1}{100} = .01$

or .001

or .0001

Smaller learning rates converge slower but can end up at better results.

Sometimes smaller is worse!

Training Loop:

"Epoch" = feed all training data through and learn from it

```python
class NeuralNetwork:

    def __init__(self):
        self.layers = []

    def add_layer(self, layer):
        self.layers.append(layer)

    def forward_pass(self, batch_data):
        """
        Perform a forward pass through the network by passing the data through
        each layer.
        """
        input_data = batch_data
        for layer in self.layers:
            input_data = layer.forward_pass(input_data)
        return input_data

    def backward_pass(self, dvalues):
        """
        Perform a backward pass through the network given gradient at output.
        """
        grad = dvalues
        for layer in reversed(self.layers):
            grad = layer.backward_pass(grad)
```

Training Loop:

"Epoch" = feed all training data through and learn from it

```
def train(self, X, y, loss_obj, epochs, learning_rate, batch_size=None, shuffle=True,
          verbose=False, X_val=None, y_val=None, metric=None):
    """
    Train the network using gradient descent.
    X: input data of shape (features, samples)
    y: true labels of shape (outputs, samples)
    loss_obj: instance of a loss class with forward_pass and backward_pass
    epochs: number of epochs to train
    learning_rate: step size for parameter updates
    batch_size: number of samples per batch (None for full-batch)
    shuffle: whether to shuffle data each epoch
    verbose: whether to print progress
    X_val: validation data of shape (features, samples), optional
    y_val: validation labels of shape (outputs, samples), optional
    metric: string, either "accuracy" or "rmse", to compute on validation data, optional

    Training can be interrupted at any time with Ctrl+C, and the method will return gracefully.
    """
```

train data

test data

(2) Stochastic Gradient Descent — The one our code uses

Feed data through in batches ("mini batches") and update gradients in the same way.

(3) "Adam Optimizer"

↳ Adaptive Momentum

Has "momentum"

Adjusts the learning rate of each weight + bias dynamically

Very good, and fast!

Demos:

(1) MNIST digits

(2) MNIST fashion

(3) Bike rentals

(4) Diabetes

Classification

Regression

→ Has the most comments

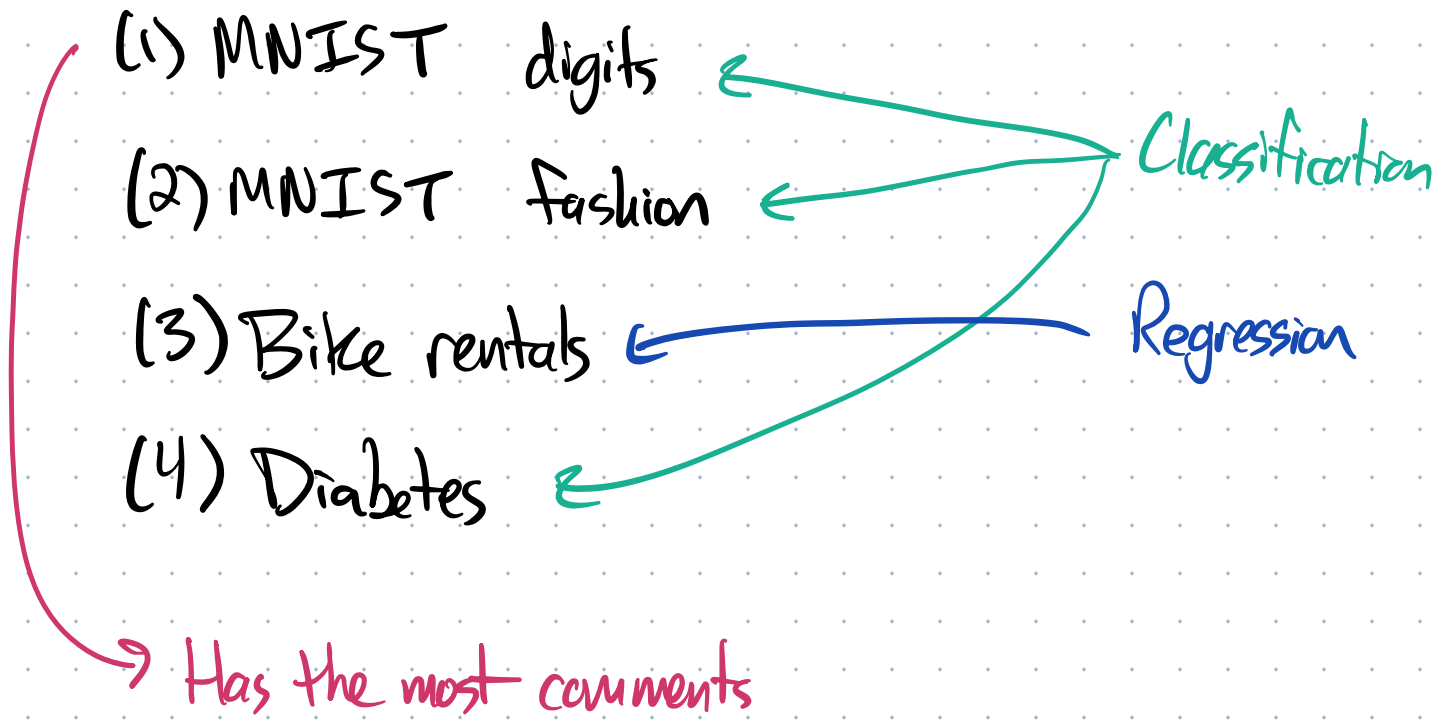
To run them, you need to be in the "network-classes-and-demos" folder, and run a command like

```
python3.13 -m demos.mnist_digits.demo_mnist_digits
```

folder

file

Demos:



You should feel free to discuss the demos with an LLM!

Paste in the code and ask "what are the lines... doing?"

(1) MNIST digits

- demo
- how we're using the classes we built
- how the "train function works"

(2) Fashion MNIST

- harder!

One-hot encoding

expected answer in the data: 7

change it in pre-processing

to: [0 0 0 0 0 0 0 1 0 0]