Scientific Computing

May 2, 2025

Announcements

→ Course Evaluations close Sunday (morning?)

→ Homework 6 due tonight

→ Final exam assigned today, due Fri, May 9, 11:59pm

→ I will have some office hours next week if you have questions. Not the normal times.

{ Tues, May 6: 1pm - 2pm
Wed, May 7: 11am - 12pm
Thurs, May 8: 2pm - 3pm }

Today

→ Backpropagation

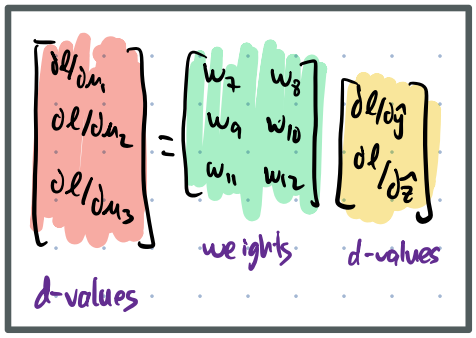
→ Designing and Training NNs

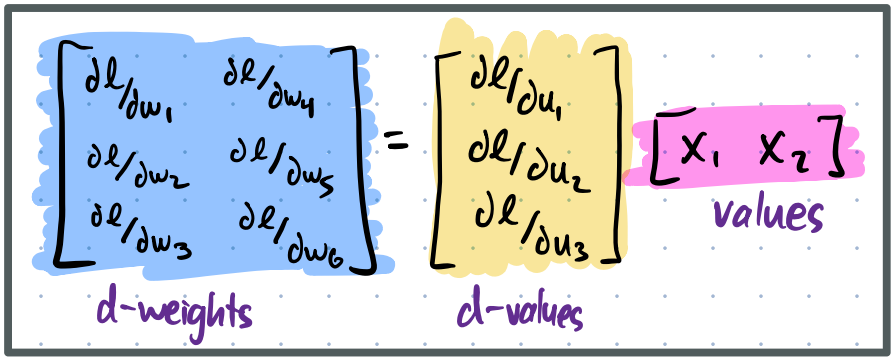
Office Hours:

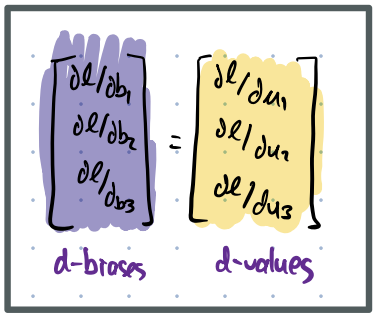
Mon + Fri

9:30am - 10:30am

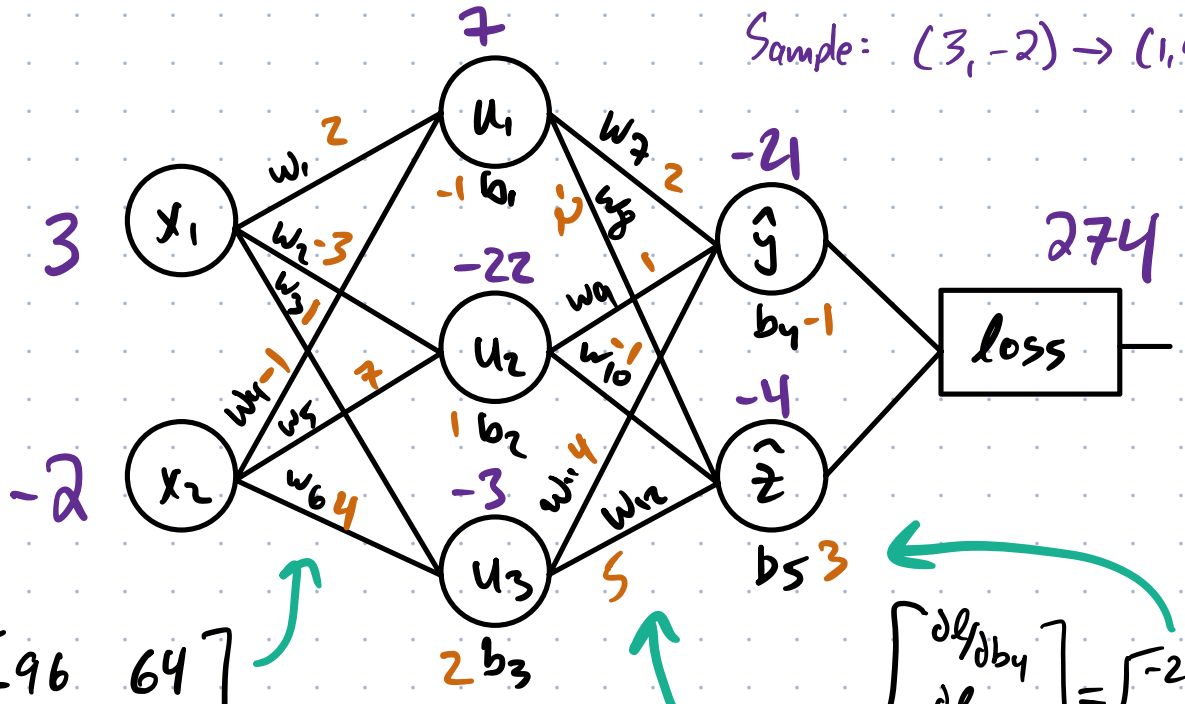
Cudahy 307







Sample: $(3, -2) \rightarrow (1, 4)$



$$\begin{bmatrix} \partial l/\partial w_1 & \partial l/\partial w_4 \\ \partial l/\partial w_2 & \partial l/\partial w_5 \\ \partial l/\partial w_3 & \partial l/\partial w_6 \end{bmatrix} = \begin{bmatrix} -96 & 64 \\ -48 & 32 \\ -354 & 236 \end{bmatrix}$$

$$\begin{bmatrix} \partial l/\partial b_1 \\ \partial l/\partial b_2 \\ \partial l/\partial b_3 \end{bmatrix} = \begin{bmatrix} -32 \\ -16 \\ -118 \end{bmatrix}$$

$$\begin{bmatrix} \partial l/\partial b_4 \\ \partial l/\partial b_5 \end{bmatrix} = \begin{bmatrix} -22 \\ -6 \end{bmatrix}$$

$$\begin{bmatrix} \partial l/\partial w_7 & \partial l/\partial w_9 & \partial l/\partial w_{11} \\ \partial l/\partial w_8 & \partial l/\partial w_{10} & \partial l/\partial w_{12} \end{bmatrix} = \begin{bmatrix} -154 & 484 & 66 \\ -42 & 132 & 18 \end{bmatrix}$$

Two more things in this topic:

* How does the previous example change with batching?
* Adding in activation functions.

Different topics:

* What do you do with the gradient?
  - Moving $-\frac{1}{100}\nabla$ is very simplistic.
  - Incorporate momentum
  - These are called "optimizers."
* How to structure a network for your particular task.

$$\begin{bmatrix} \partial l/\partial u_1 \\ \partial l/\partial u_2 \\ \partial l/\partial u_3 \end{bmatrix} = \begin{bmatrix} w_7 & w_8 \\ w_9 & w_{10} \\ w_{11} & w_{12} \end{bmatrix} \begin{bmatrix} \partial l/\partial \hat{y} \\ \partial l/\partial \hat{z} \end{bmatrix}$$

d-values = weights · d-values

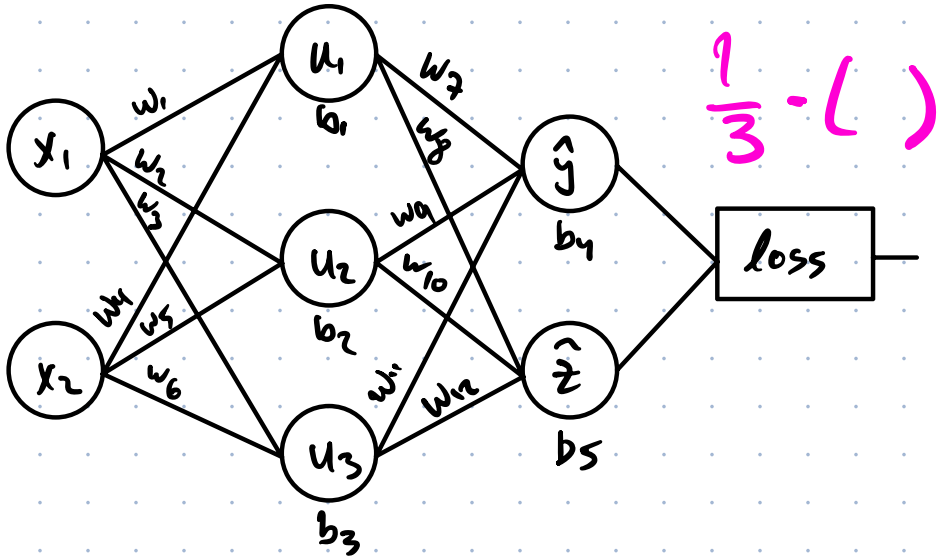
$$\begin{bmatrix} \partial l/\partial w_1 & \partial l/\partial w_4 \\ \partial l/\partial w_2 & \partial l/\partial w_5 \\ \partial l/\partial w_3 & \partial l/\partial w_6 \end{bmatrix} = \begin{bmatrix} \partial l/\partial u_1 \\ \partial l/\partial u_2 \\ \partial l/\partial u_3 \end{bmatrix} \begin{bmatrix} x_1 & x_2 \end{bmatrix}$$

d-weights = d-values · values

$$\begin{bmatrix} \partial l/\partial b_1 \\ \partial l/\partial b_2 \\ \partial l/\partial b_3 \end{bmatrix} = \begin{bmatrix} \partial l/\partial u_1 \\ \partial l/\partial u_2 \\ \partial l/\partial u_3 \end{bmatrix}$$

d-biases = d-values

What changes when batching inputs?



The values of the layers are matrices instead of vectors. So d-values are also matrices.

d-biases are then row sums of d-values (averaging)

d-weights is a matrix times a matrix (more averaging)

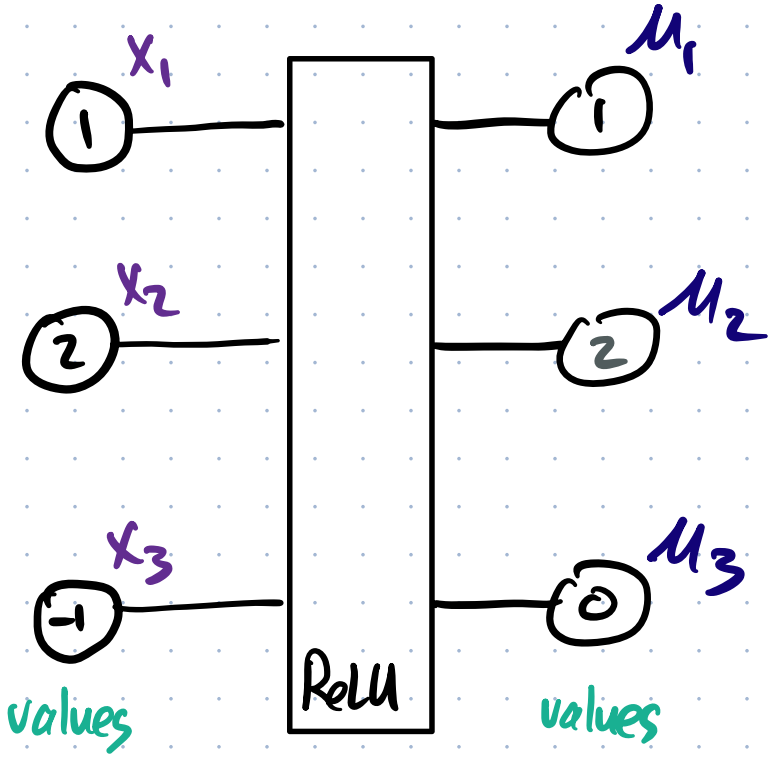
Activation Functions

* Easy derivatives that go between the layers.

$$ReLU(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

$$\frac{\partial ReLU}{\partial x} = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

(ignore the discontinuity)



Activation Functions

* Easy derivatives that go between the layers.

$$\text{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

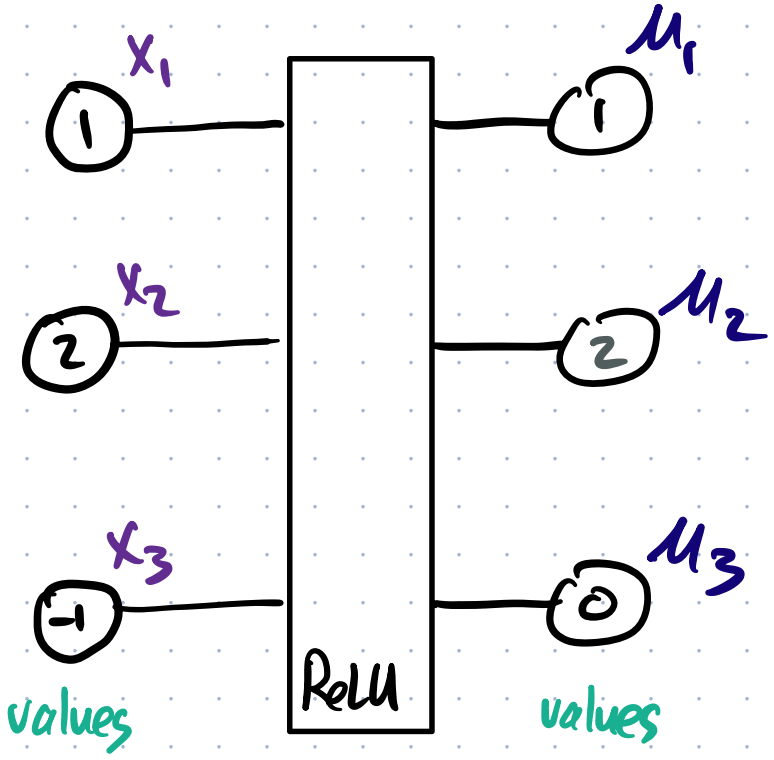
$$\frac{\partial \text{ReLU}}{\partial x} = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

(ignore the discontinuity)

$\frac{\partial u_1}{\partial x_1} = 1$

$\frac{\partial u_2}{\partial x_2} = 1$

$\frac{\partial u_3}{\partial x_3} = 0$



Activation Functions

* Easy derivatives that go between the layers.

Sigmoid, $\sigma(x) = \frac{1}{1+e^{-x}}$

$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \sigma(x) \cdot (1 - \sigma(x))$$

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

you can compute the derivative of $\sigma(x)$ using only its value!

Activation Functions

* Tanh

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = \dots = 1 - g(z)^2.$$

Another trick!

Activation Functions

* Categorical Cross-Entropy Loss – classification
* Softmax

There's a great trick here where we do the softmax activation and then CCE loss in as one big function instead of 2, and their joint derivative is much nicer than the individual ones.

→ Details in the code and in the Neural Networks From Scratch book.

Layer Class:

```python
def forward_pass(self, batch_inputs):
    """
    Compute the value for the neurons in this layer given the input
    from the previous layer.
    """
    self.inputs = batch_inputs
    return np.dot(self.weights, batch_inputs) + self.biases
```

```python
def backward_pass(self, dvalues):
    """
    Compute the gradient of the loss with respect to the weights and biases
    feeding into this layer and the neuron values of previous layer (aka, the
    inputs into this layer). Input is the gradient with respect to neuron values
    in this layer. Think of this like we are pushing out the gradients from us
    into the previous layer.
    """
    # Compute the gradient of the loss with respect to the weights and biases
    self.dweights = np.dot(dvalues, self.inputs.T)
    self.dbiases = np.sum(dvalues, axis=1, keepdims=True)

    # Compute the gradient of the loss with respect to the inputs
    self.dinputs = np.dot(self.weights.T, dvalues)
    return self.dinputs
```

ReLU Activation

```python
def forward_pass(self, batch_values):
    """
    Compute the ReLU activation for the input values.
    """
    self.inputs = batch_values
    return np.maximum(0, batch_values)

def backward_pass(self, dvalues):
    """
    Compute the gradient of the loss with respect to the input values.
    """
    # Gradient of ReLU is 1 for positive values, 0 for negative values
    self.dvalues = dvalues * (self.inputs > 0)
    return self.dvalues
```

gradients from layer after

{ 1, x>0 ; 0, x<0 } · prev. gradient

Sigmoid Activation

$\sigma' = \sigma \cdot (1 - \sigma)$

```python
def forward_pass(self, batch_values):
    """
    Compute the sigmoid activation for the input values.
    Sigmoid function: f(x) = 1 / (1 + e^(-x))
    """
    self.inputs = batch_values
    # in this case we save the outputs because we can reuse them for
    # the backward pass
    self.outputs = 1 / (1 + np.exp(-batch_values))
    return self.outputs

def backward_pass(self, dvalues):
    """
    Compute the gradient of the loss with respect to the input values.
    The derivative of the sigmoid function is f'(x) = f(x) * (1 - f(x))
    """
    self.dvalues = dvalues * (self.outputs * (1 - self.outputs))
    return self.dvalues
```

Neural Network Training Loop

```
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)
```

input neurons

Plenty more details we're skipping for now.

Done!

* We know how to make a NN match the training data a little bit better.

* Pass it all through in a forward pass ⟶
  then compute the gradients going backward ⟵

* Update weights + biases based on those gradients ("optimizers")
  then repeat.

Well, one more thing: mini batching

* Usually you don't send all the training data through in one big batch.
* Shuffle the order of the data. Then do forward and backward passes of 64 (or whatever) input/output pairs at a time until all have been used.
* Shuffle again and repeat.

- Faster, less memory usage, and often trains better!

1 "epoch" is one pass through the training data

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

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.

Loss / Accuracy / RMSE

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

Train Loss, Test Loss

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

Training Loop:

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

Different Ways — "Optimizers"

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

↑ 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!

(2) Stochastic Gradient Descent

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!

Training Loop:

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

```
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.
    """
```

Demos:

(1) MNIST digits

(2) MNIST fashion

(3) Bike rentals

(4) Diabetes

Classification

Regression

→ Has the most comments

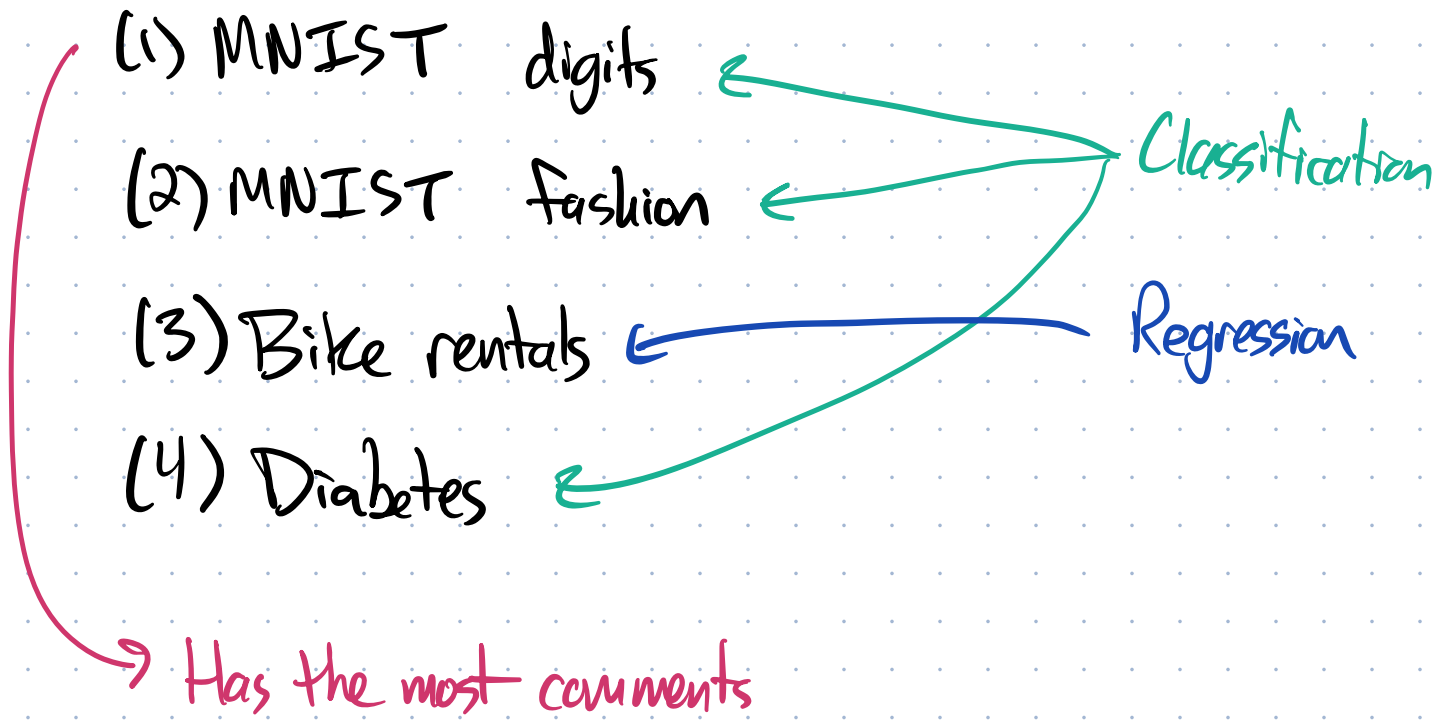
To run them, you need to be in the "network-classes-and-demos" folder, and run a command like

```
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!

13) Bike sharing (there is a "bike-share-explanation.md" markdown file

Data from a bike rental service in Washington, D.C.

- Date + Time
- Season
- Holiday
- Workday
- Weather
- Temperature
- "Feels like" temp.
- Humidity
- Wind Speed

- # rentals per day (registered / casual)

(3) Bike sharing (there is a "bike-share-explanation.md" markdown file

Data from a bike rental service in Washington, D.C.

Data Fields

datetime - hourly date + timestamp
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
workingday - whether the day is neither a weekend nor holiday
weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp - temperature in Celsius
atemp - "feels like" temperature in Celsius
humidity - relative humidity
windspeed - wind speed
casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals

How do we set these up as numerical inputs?

All inputs should be in the range $[-1, 1]$ (or at least close)

(3) Bike sharing (there is a "bike-share-explanation.md" markdown file

Data from a bike rental service in Washington, D.C.

* Some data is already numeric (takes values in a range)

Scale data down into the range $[-1, 1]$

Option 1: Scale it linearly.
Option 2: Scale by $\frac{x - \text{mean}}{\text{st. dev.}}$

Data Fields

datetime - hourly date + timestamp
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
workingday - whether the day is neither a weekend nor holiday
weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp - temperature in Celsius
atemp - "feels like" temperature in Celsius
humidity - relative humidity
windspeed - wind speed
casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals = redundant

} output

Scale output too! just make sure you unscale when looking at your predictions later

(3) Bike sharing (there is a "bike-share-explanation.md" markdown file

Data from a bike rental service in Washington, D.C.

binary variables: holiday is yes or no
workingday is yes or no

Each gets one neuron,
0 input = no
1 input = yes

Data Fields

datetime - hourly date + timestamp
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
workingday - whether the day is neither a weekend nor holiday
weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp - temperature in Celsius
atemp - "feels like" temperature in Celsius
humidity - relative humidity
windspeed - wind speed
casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals

} output

= redundant

(3) Bike sharing (there is a "bike-share-explanation.md" markdown file

Data from a bike rental service in Washington, D.C.

Season + Weather are encoded as #s, but not #s that mean anything.

Bad: divide "weather" by 4 to get 1 neuron in [0,1].

Good: Use four neurons, one for each option. The correct option is a 1, the rest are 0.

Data Fields

datetime - hourly date + timestamp
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
workingday - whether the day is neither a weekend nor holiday
weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp - temperature in Celsius
atemp - "feels like" temperature in Celsius
humidity - relative humidity
windspeed - wind speed
casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals

} output

← redundant

(3) Bike sharing (there is a "bike-share-explanation.md" markdown file

Data from a bike rental service in Washington, D.C.

First split into month of the year, day of the week, hour of the day
1-12 0-6 0-23

Again, we don't want to use these as #s. Hour 0 and 23 should be "close", right?

"Cyclical Encoding"

Data Fields

datetime - hourly date + timestamp
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
workingday - whether the day is neither a weekend nor holiday
weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp - temperature in Celsius
atemp - "feels like" temperature in Celsius
humidity - relative humidity
windspeed - wind speed
casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals

} output

← redundant

(3) Bike sharing (there is a "bike-share-explanation.md" markdown file

Data from a bike rental service in Washington, D.C.

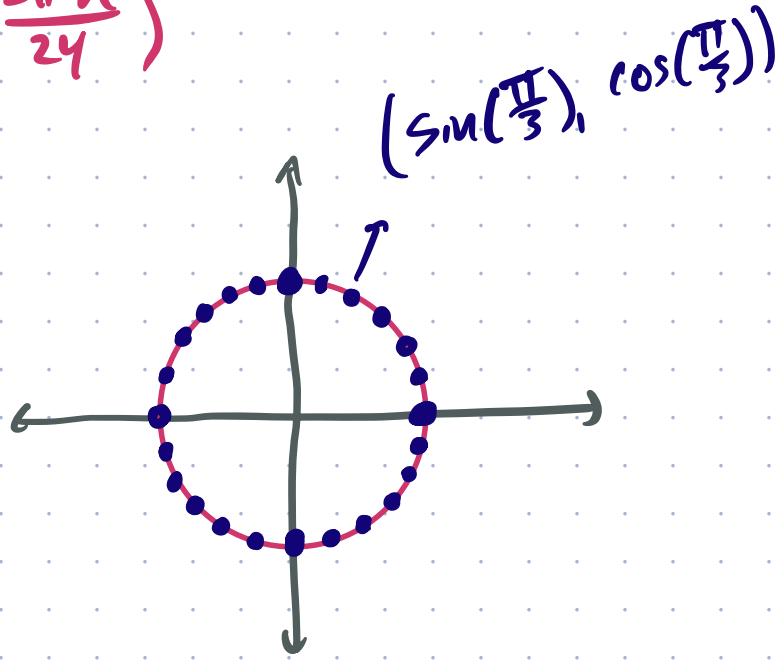
Cyclical Encoding Let h = hour of the day. Encode with two neurons. $n_1 = \sin\left(\frac{2\pi \cdot h}{24}\right)$

$n_2 = \cos\left(\frac{2\pi \cdot h}{24}\right)$

Data Fields

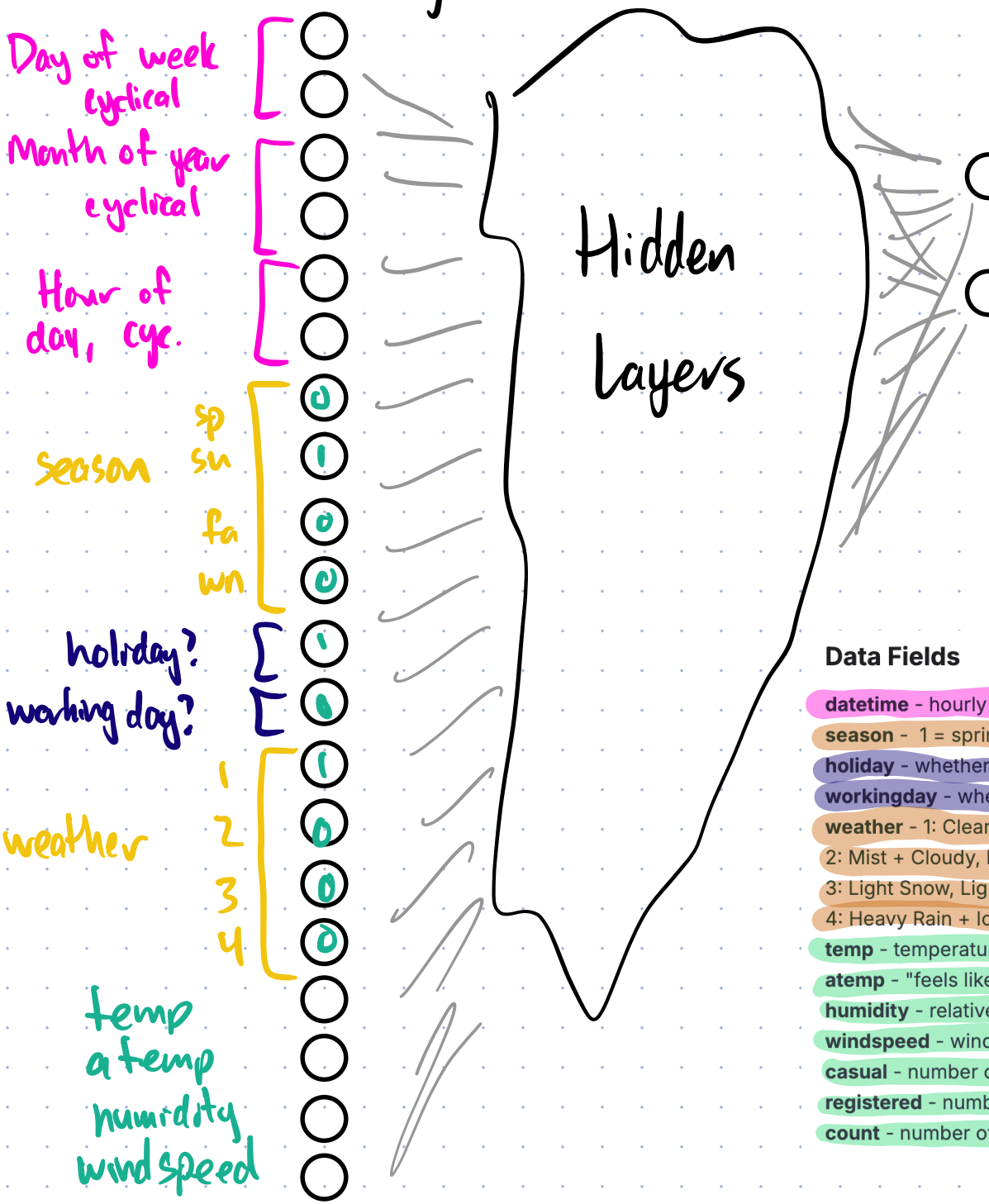
datetime - hourly date + timestamp
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
workingday - whether the day is neither a weekend nor holiday
weather - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp - temperature in Celsius
atemp - "feels like" temperature in Celsius
humidity - relative humidity
windspeed - wind speed
casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals } output

} redundant



Imparts a cyclical feel. 0 and 23 are close now

[3] Bike sharing



Data Fields

- **datetime** - hourly date + timestamp
- **season** - 1 = spring, 2 = summer, 3 = fall, 4 = winter
- **holiday** - whether the day is considered a holiday
- **workingday** - whether the day is neither a weekend nor holiday
- **weather** - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
  2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
  3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
  4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- **temp** - temperature in Celsius
- **atemp** - "feels like" temperature in Celsius
- **humidity** - relative humidity
- **windspeed** - wind speed
- **casual** - number of non-registered user rentals initiated
- **registered** - number of registered user rentals initiated
- **count** - number of total rentals ← redundant

} output

13) Bike sharing "bike-share-explanation.md"

- Demo
- There are some numpy parts that look complicated but are doing simple things. Use Claude or ChatGPT as your coach!

(4) Diabetes prediction

- Demo, investigate and run on your own!
- "diabetes-explanation.md"

Field	Description	Type/Range	Preprocessing Approach
HighBP	High blood pressure (0 = no, 1 = yes)	Binary (0/1)	Use as-is
HighChol	High cholesterol (0 = no, 1 = yes)	Binary (0/1)	Use as-is
CholCheck	Cholesterol check in last 5 years (0 = no, 1 = yes)	Binary (0/1)	Use as-is
BMI	Body Mass Index	Continuous	Standardize (zero mean, unit variance)
Smoker	Smoked at least 100 cigarettes (0 = no, 1 = yes)	Binary (0/1)	Use as-is
Stroke	Ever told had a stroke (0 = no, 1 = yes)	Binary (0/1)	Use as-is
HeartDisease	Coronary heart disease or MI (0 = no, 1 = yes)	Binary (0/1)	Use as-is
PhysActivity	Physical activity in past 30 days (0 = no, 1 = yes)	Binary (0/1)	Use as-is
Fruits	Consume fruit 1+ times/day (0 = no, 1 = yes)	Binary (0/1)	Use as-is
Veggies	Consume vegetables 1+ times/day (0 = no, 1 = yes)	Binary (0/1)	Use as-is
HvyAlcohol	Heavy drinker (0 = no, 1 = yes)	Binary (0/1)	Use as-is
AnyHealthcare	Any health care coverage (0 = no, 1 = yes)	Binary (0/1)	Use as-is
NoDocbcCost	Could not see doctor due to cost (0 = no, 1 = yes)	Binary (0/1)	Use as-is
GenHlth	General health (1 = excellent, 5 = poor)	Ordinal (1–5)	Standardize
MentHlth	Days mental health not good (1–30)	Ordinal (1–30)	Standardize
PhysHlth	Days physical health not good (1–30)	Ordinal (1–30)	Standardize
DiffWalk	Difficulty walking/climbing stairs (0 = no, 1 = yes)	Binary (0/1)	Use as-is
Sex	0 = female, 1 = male	Binary (0/1)	Use as-is
Age	Age group (1 = 18–24, ..., 13 = 80+)	Ordinal (1–13)	Standardize
Education	Education level (1 = none/kindergarten, ..., 6 = college grad)	Ordinal (1–6)	Standardize
Income	Income scale (1 = < $10k, ..., 8 = $75k+)	Ordinal (1–8)	Standardize

You now have all the fundamental knowledge of how Neural Networks work!

There are many more things to learn about them.

Ex: Specialized networks for some kinds of tasks

Convolutional Neural Networks → Grid Data like images

Graph Neural Networks → Drug Design

Ex: Preventing overfitting, which is when the NN memorizes the training data in a way that doesn't extrapolate to the test data.

Ex: Preventing overfitting, which is when the NN memorizes the training data in a way that doesn't extrapolate to the test data.

* Use a smaller network.

* Dropout - each training loop, randomly turn off 20% of neurons (they don't pass data forward and don't get adjusted by gradient descent)

* Regularization - Try to stop the weights from getting too big (2, 5, 10, etc).

  Adds a component to the loss function to penalize for big weights.