Week 8 Discussion

Agenda

- Introduction to Deep Learning

- PyTorch

Machine Learning vs. Deep Learning

Machine Learning

- Algorithms that can learn from data and generalize to unseen data
- In ML, the programmer does not explicitly define the rules

Deep Learning

- Subset of machine learning that utilizes artificial neural networks
- “Deep” refers to the many layers that a neural network can have

Field of Artificial Intelligence

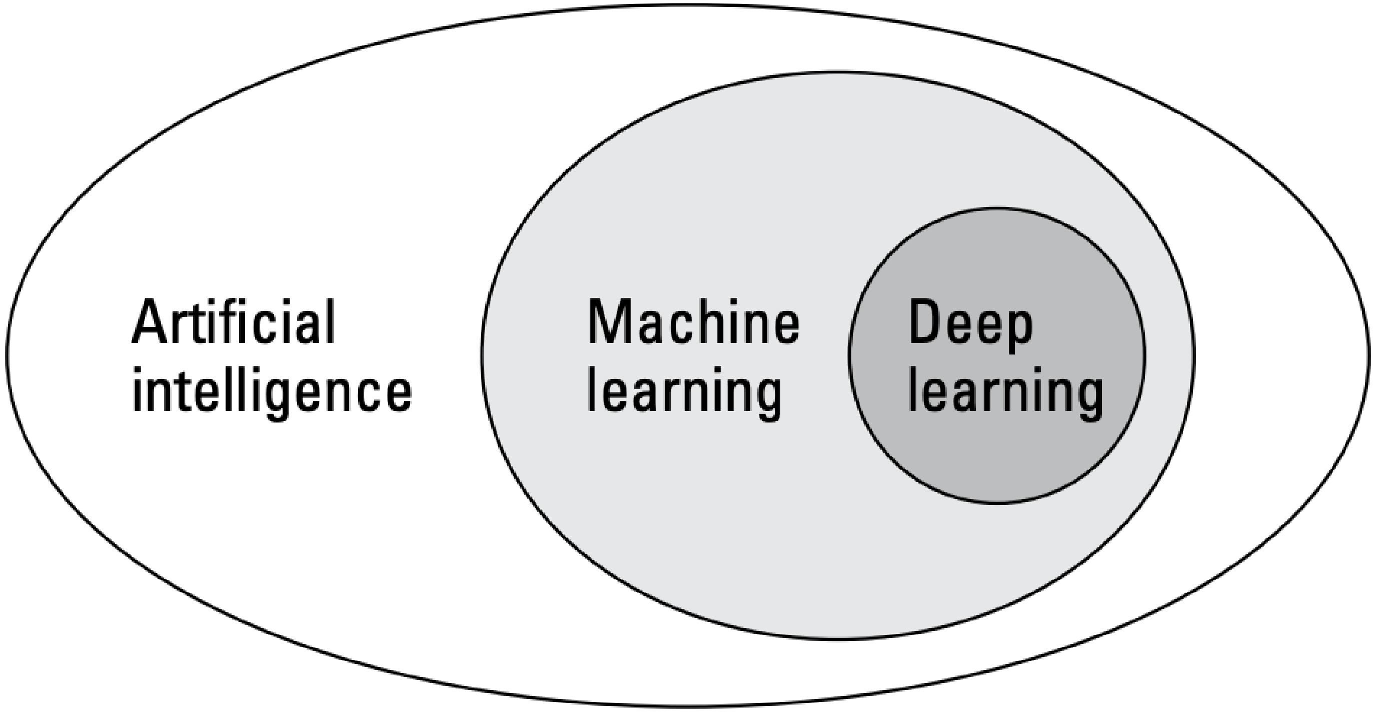
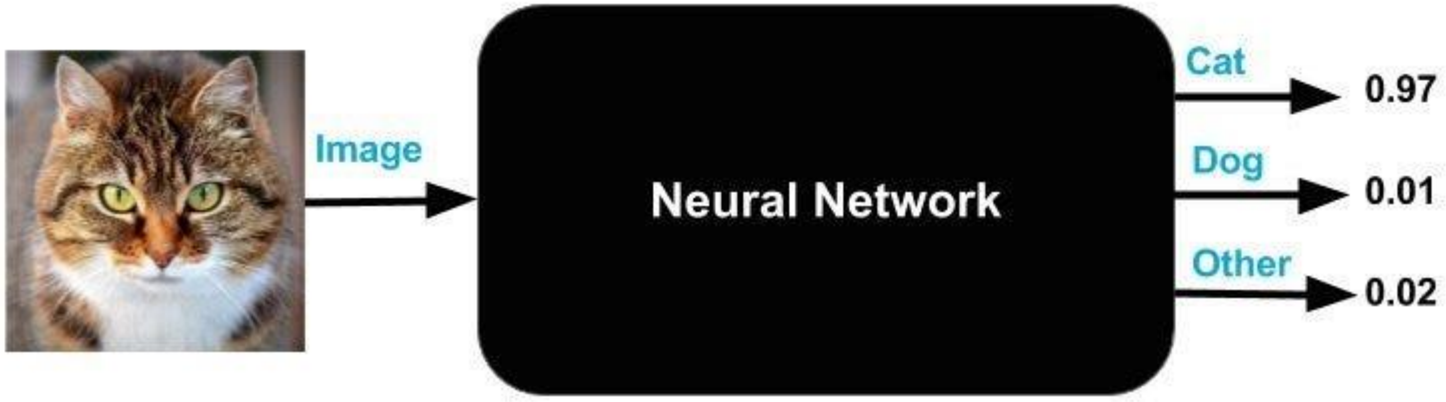


FIGURE 1-1: Deep learning, a subset of a subset of AI.

https://www.deepinstinct.com/glossary/deep-learning

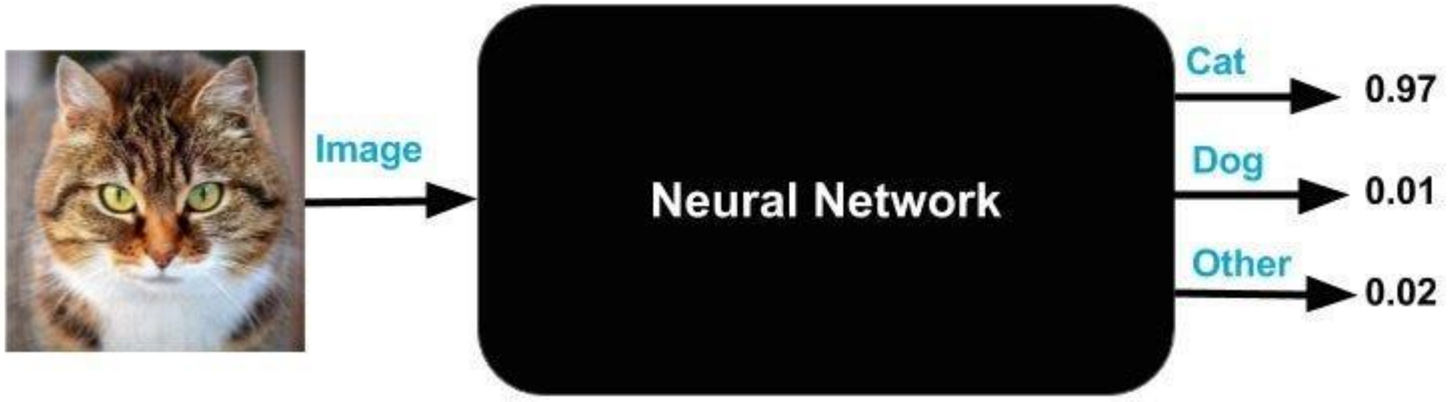
Neural Networks

- At first, let's imagine neural networks as a black box
- Here, we are trying to solve a classification problem
  - Given an image, can we predict the class label? (cat, dog, other)



Neural Networks

- You can first imagine that a neural network is just a (differentiable) function *f(x)*
  - x is the input data, and the function f is the network. What comes out is the prediction of the problem we want to solve



Neural Networks

- Neural networks are made up of connected neurons
- The network takes in some data and outputs a prediction

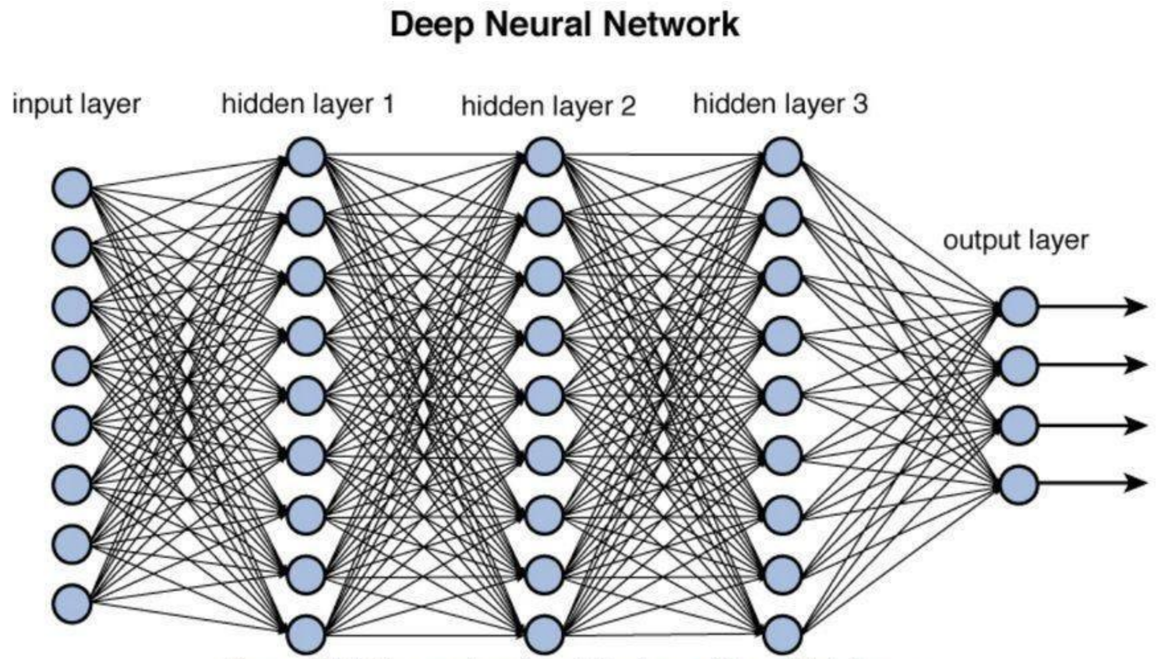


Figure 12.2 Deep network architecture with multiple layers.

https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Neural Networks

- The neurons are computational units that have one or more weighted inputs
    - They take in some data, perform a mathematical operation, and produce some output

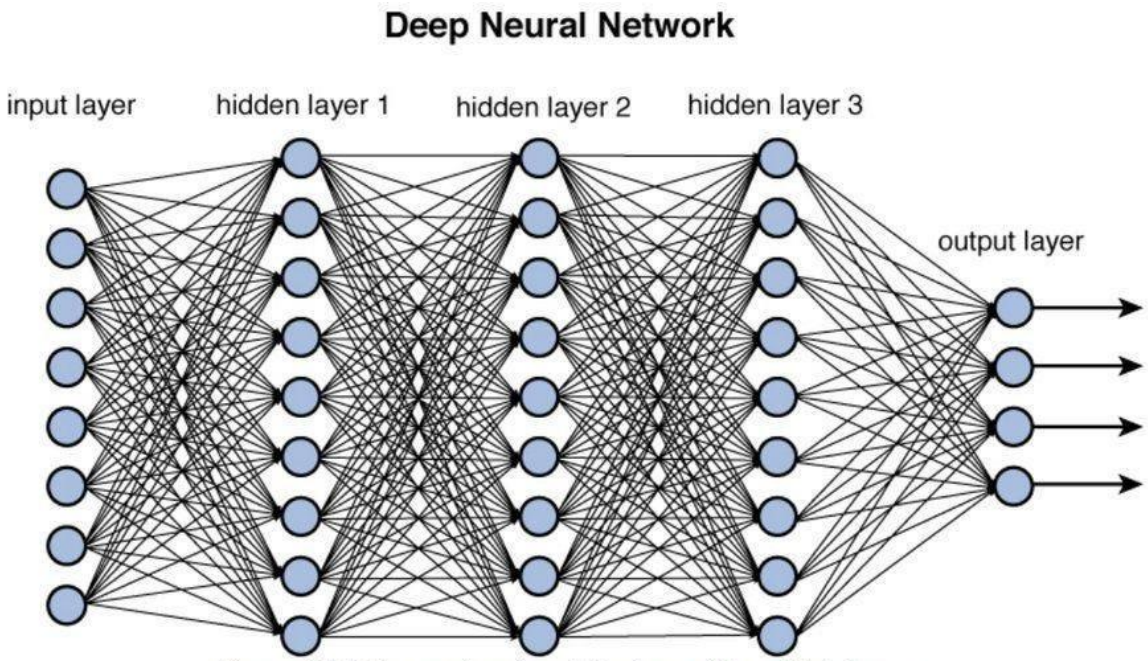


Figure 12.2 Deep network architecture with multiple layers.

https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Neural Networks

- The neurons are organized into the layers of the network
- The hidden layers are “hidden” because their inputs and outputs are not directly understandable from outside of the neural network

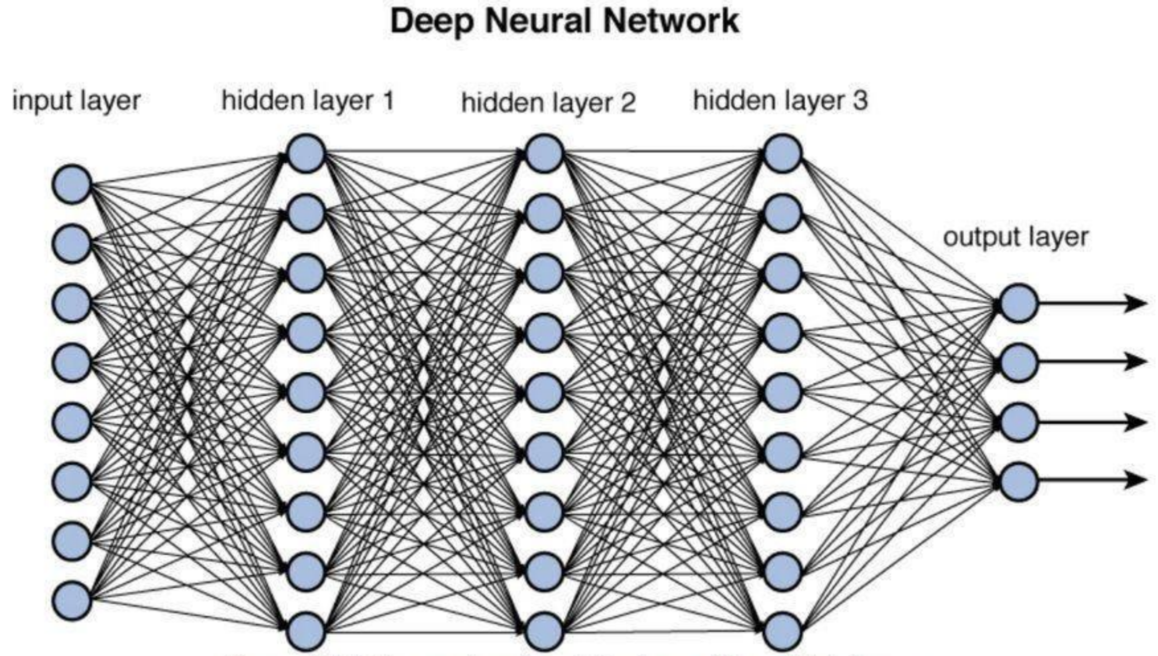
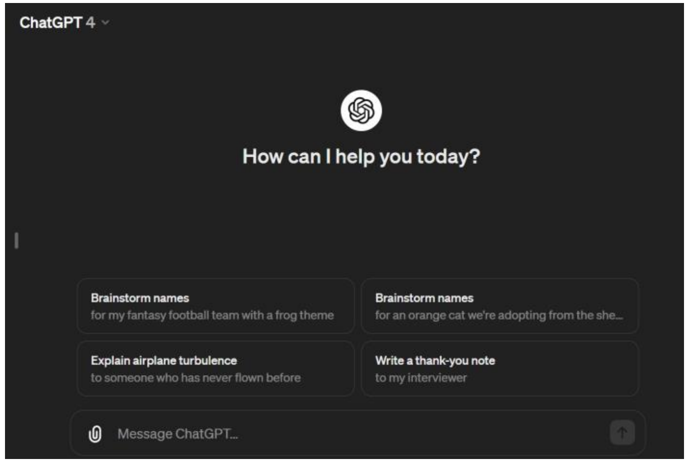


Figure 12.2 Deep network architecture with multiple layers.

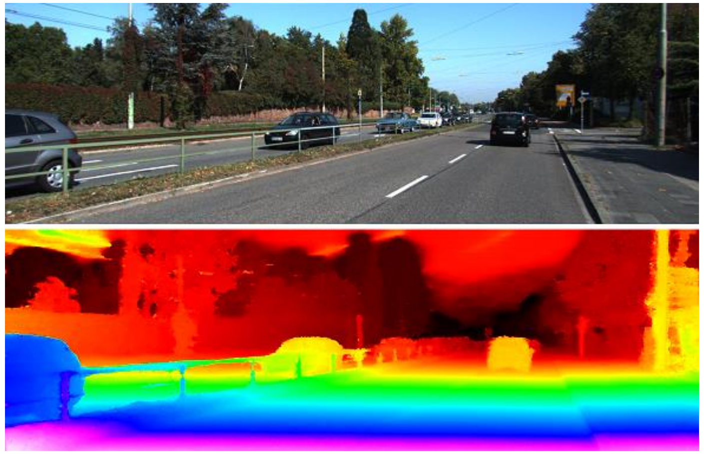
https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Neural Networks

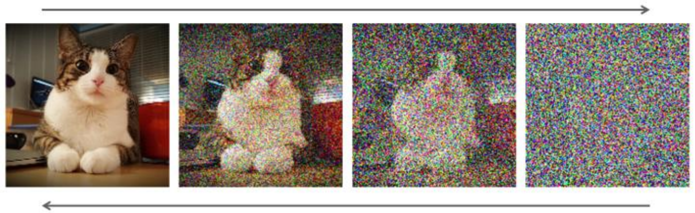
- Neural networks have enabled many applications



https://chat.openai.com/



https://research.nvidia.com/publication/2018-04_importance-stereo-accurate-depth-estimation-efficient-semi-supervised-deep



https://developer.nvidia.com/blog/improving-diffusion-models-as-an-alternative-to-gans-part-2/

Inside the Layers of a Neural Network

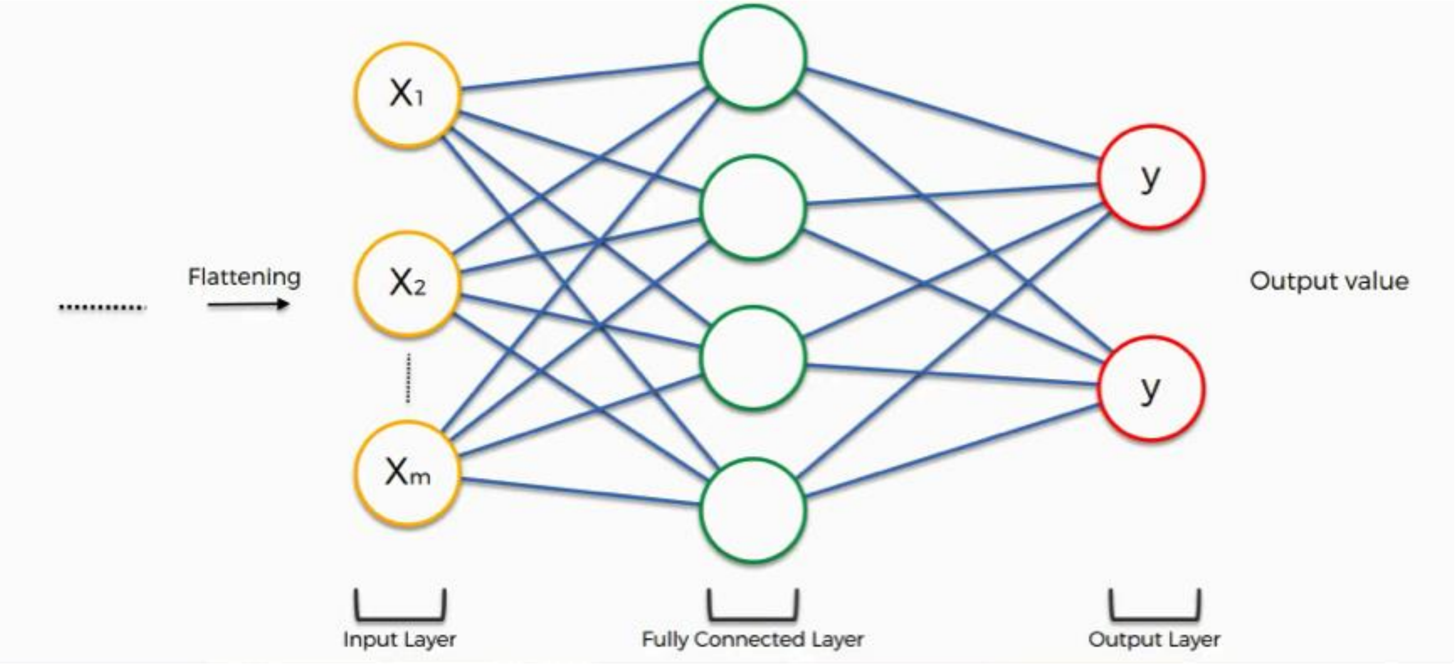
- There are a few examples of layers inside of a neural network

- Fully-connected layer
  - Performs a matrix-vector product Ax=b
  - Called *Linear* layers in PyTorch

- Convolutional Layers
  - Performs a convolution operation

Fully-Connected Layer

- Matrix-vector multiplication Ax=b
  - A is a matrix of parameters/weights that needs to be optimized
  - x is the input data from the previous layer



https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection

Parameters or Weights of a Network

- The parameters or weights of the network are the things inside the network that we want to optimize
  - This is what is “learned” from our data

- For a fully-connected layer, we want to find the values in our matrix A that gives our network the best performance
  - We can first initialize them randomly and optimize them through training

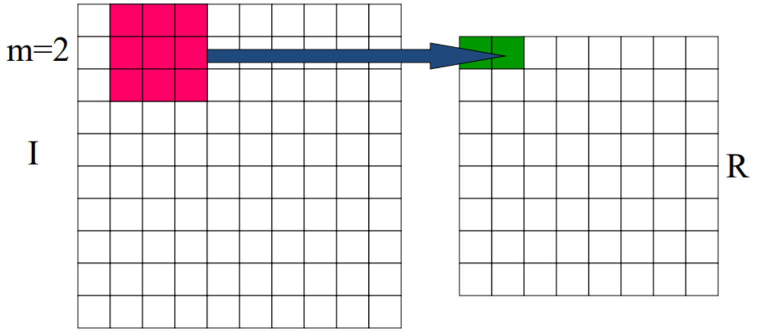
$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

https://bytepawn.com/pytorch-basics-solving-the-axb-matrix-equation-with-gradient-descent.html

Convolutional Layers

- In CV, we often work with images or 2D/3D tensors. Convolutions can take advantage of the structure of image data
- A network that utilizes convolutional layers are convolutional neural networks or CNNs
  - They have quickly become state-of-the-art in many problems

Convolution: R= K*I



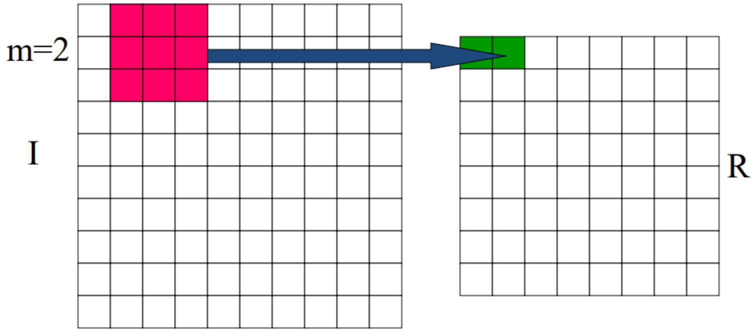
Kernel size
is m+1 by m+1

$$R(i,j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h,k) I(i-h, j-k)$$

Convolutional Layers

- We've seen convolutions many times in the course so far, but we used them with a specific goal
  - We needed to specify the kernel for a specific calculation
  - Calculating derivatives of an image, smoothing
- In a CNN, the weights in the kernel are optimized by the network

Convolution: R= K*I



Kernel size
is m+1 by m+1

$$R(i,j)=\sum_{h=-m/2}^{m/2}\sum_{k=-m/2}^{m/2}K(h,k)I(i-h,j-k)$$

Network Architecture Design

Question:

OK, We have defined the layers of a network. How do you know how to design one specifically?

There is no single correct answer to this. It will depend on a variety of factors such as:

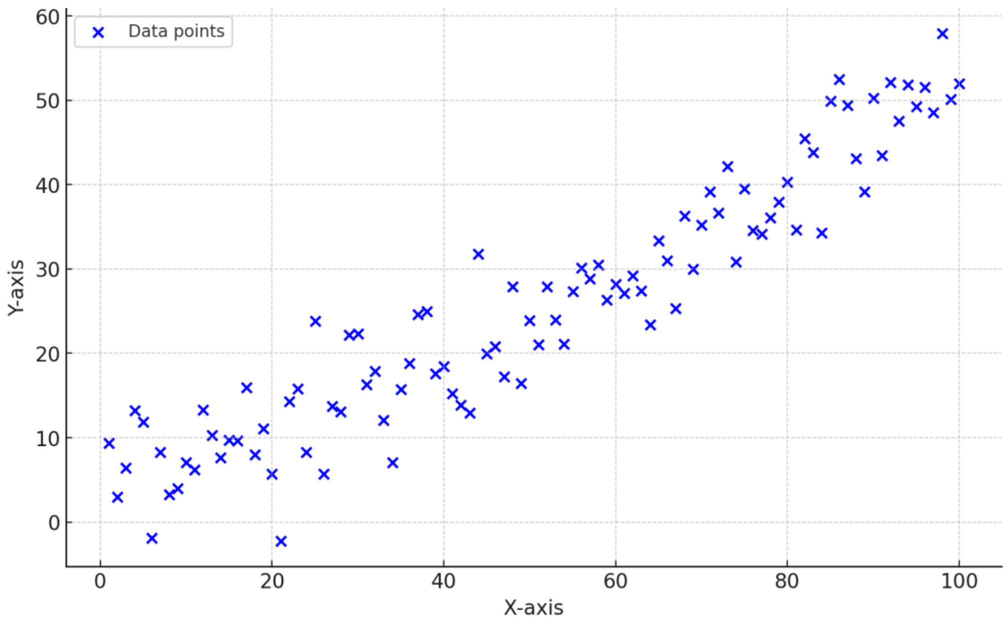
- The problem you want to solve and the data you have
- Problem complexity/difficulty
- Hardware limitations

You could often just start with a well-known architecture and tweak it for your needs.

In the homework, it will be provided to you.

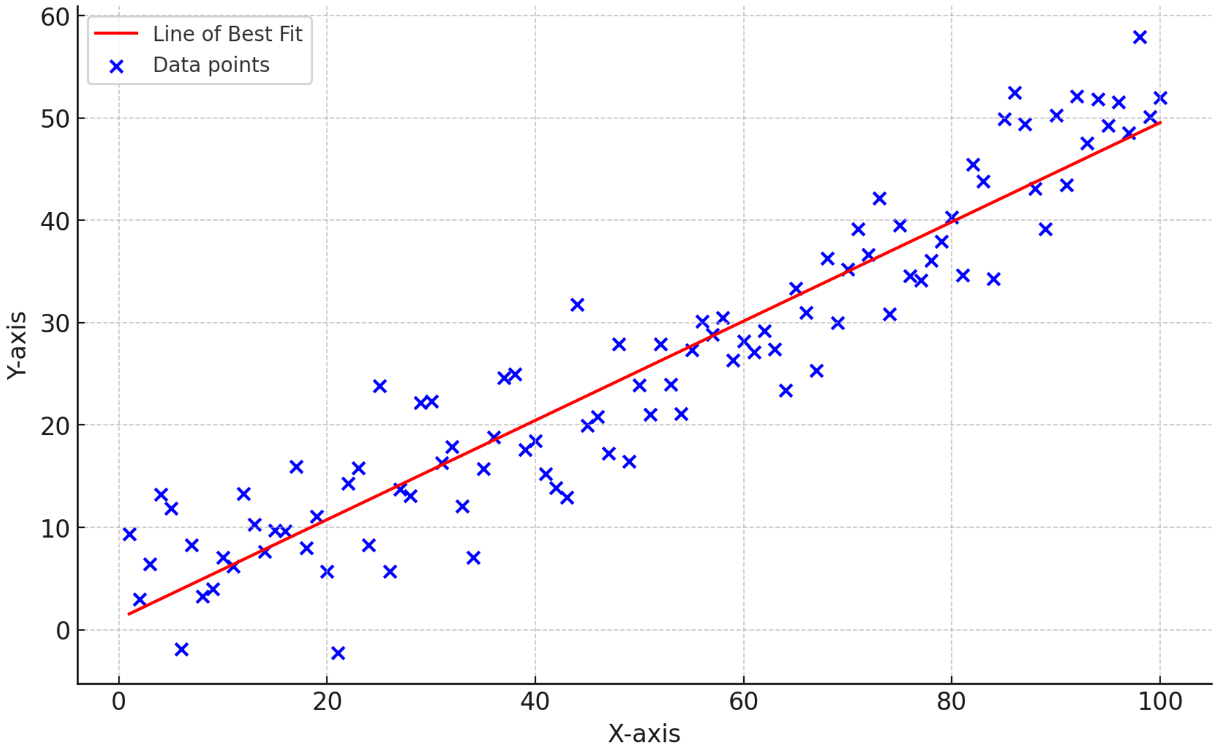
Taking a step back

- We’ve discussed the main components of a deep neural network and general applications, but what exactly do they do?
  - Simply put, they are functions that model the underlying patterns in our data
- Taking a step backwards, lets look at some data



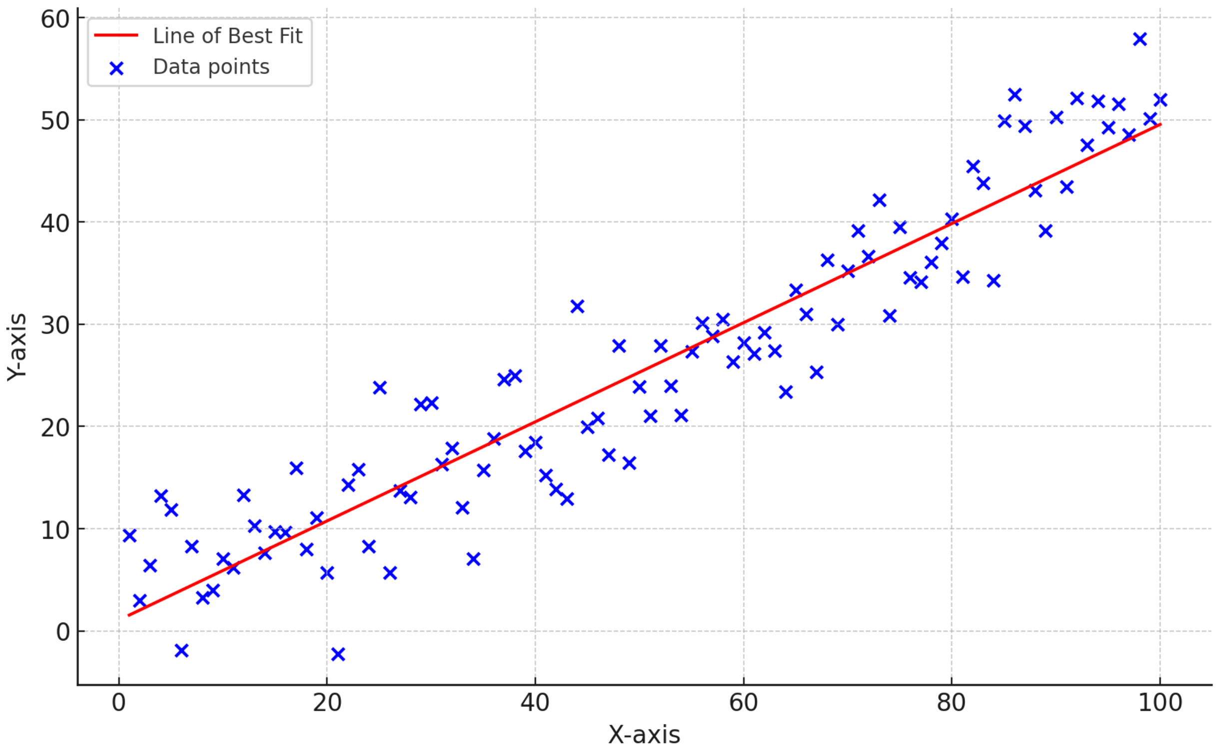
Taking a step back

- What kind of model fits this data best?
  - Probably a line! $f(x)=mx+b$
  - m and b would be the weights that we want to optimize!



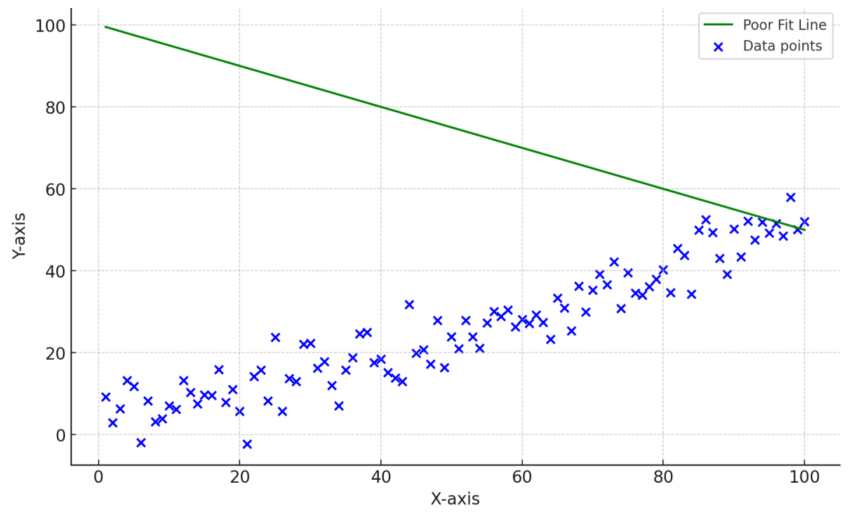
Taking a step back

- Looking at the function *f(x)=mx+b*, we can describe this line with a neural network.
- This network would have a single fully-connected layer!
    - *f(x)=Ax*



So, we have a network. How do we train it?

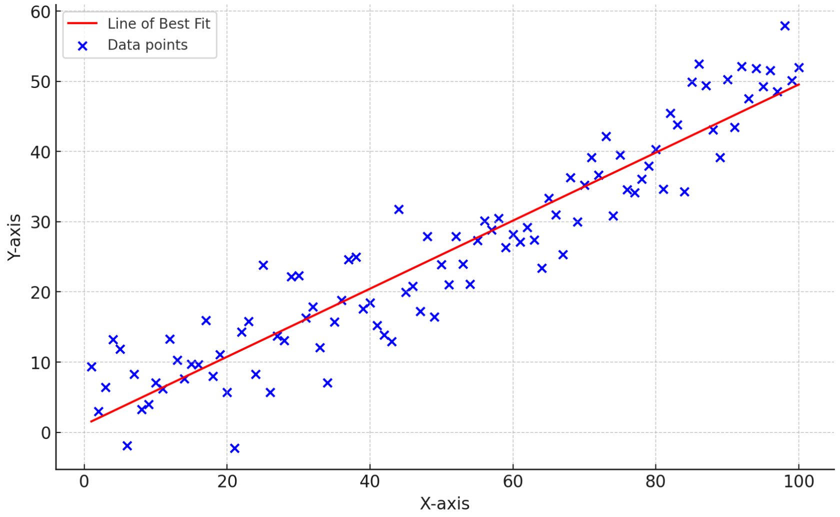
- At this point, we have our model, and we have our data
  - Say we initialize the weights randomly
- So, how do we optimize or train it?



Random weights produce a line with a poor fit

f(x)=mx+b

Training



After training, we obtain a line with a much better fit

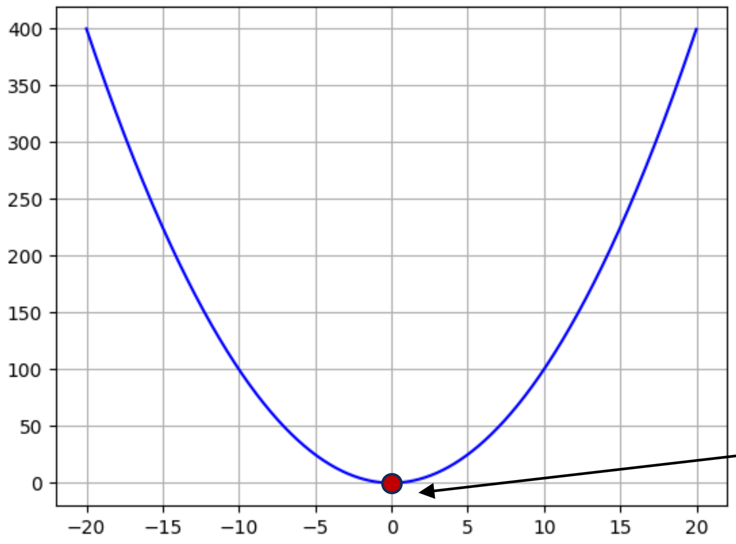
So, we have a network. How do we train it?

- First, we need to define a loss function
  - For a regression problem, we can use something like mean squared error (MSE)
- Loss functions are a quantitative evaluation of how well your model is performing
- By minimizing the loss on the training data, we are modifying the parameters in the network that gives us the best performance

Ground-truth label

$$\text{MSE} = \overbrace{\boxed{\frac{1}{n}\sum_{i=1}^{n}}}^{\textbf{Mean}}\Big(\overbrace{\boxed{Y_i - \hat{Y}_i}}^{\text{Error}}\Big)^{\overbrace{\boxed{2}}^{\textbf{Squared}}}$$

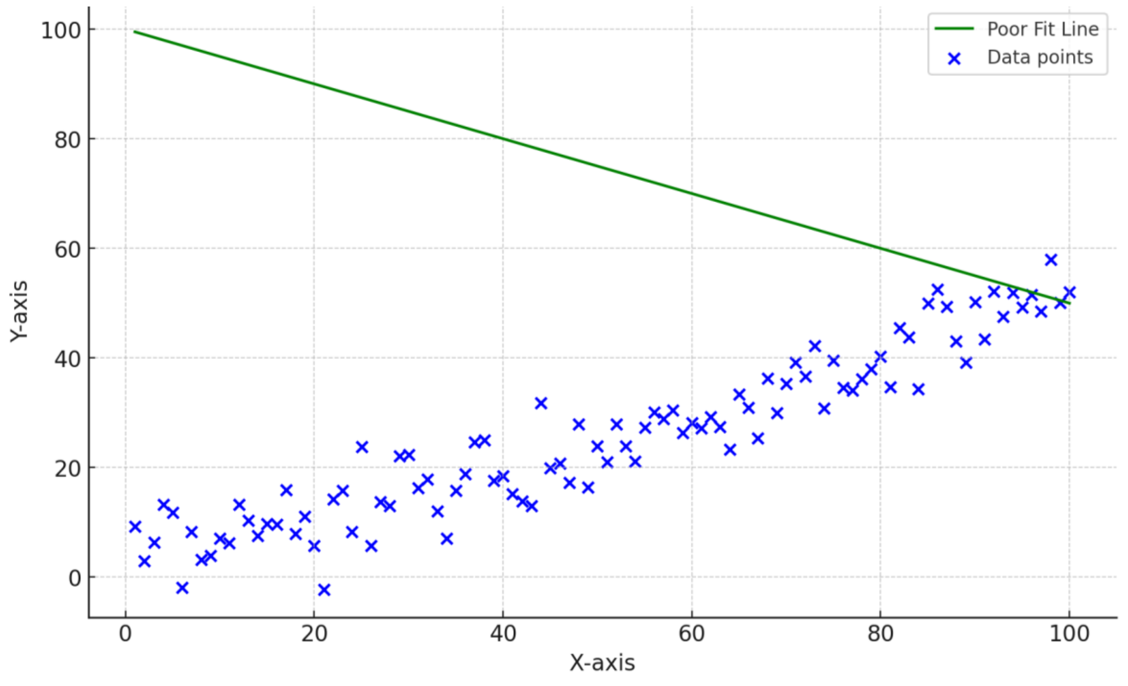
f(x)=mx+b
(prediction)



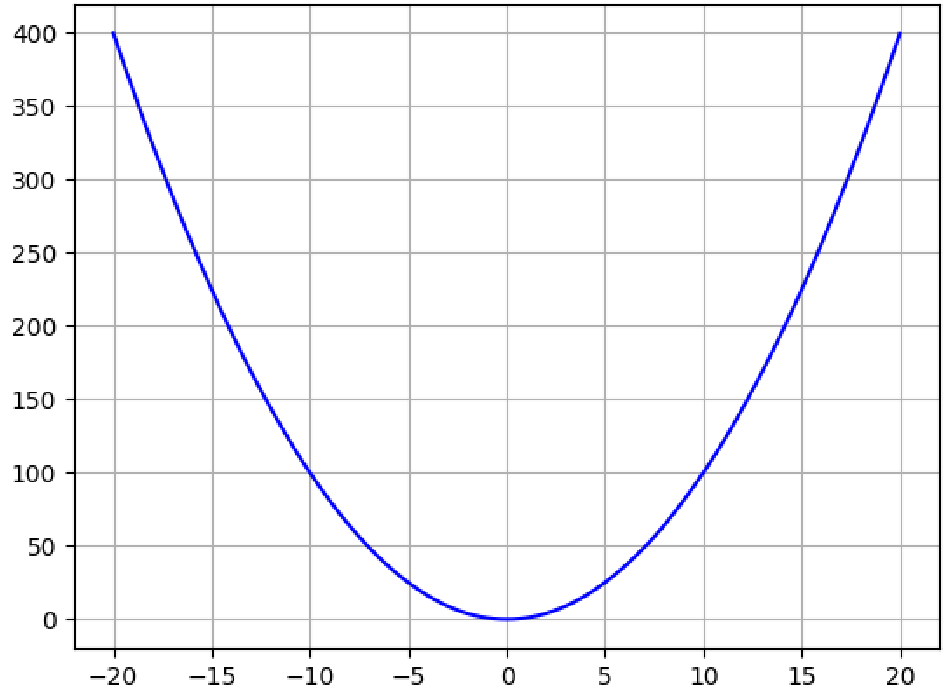
MSE plot might look something like this

A model with a perfect fit would produce 0 error on the training data

So, we have a network. How do we train it?



Plot of data *x* versus label *y*

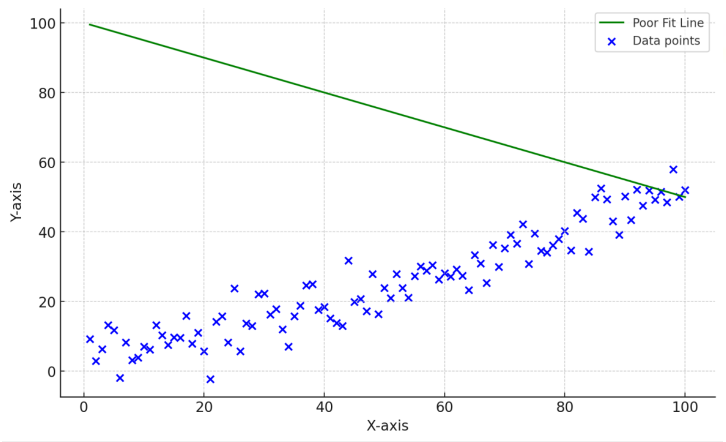


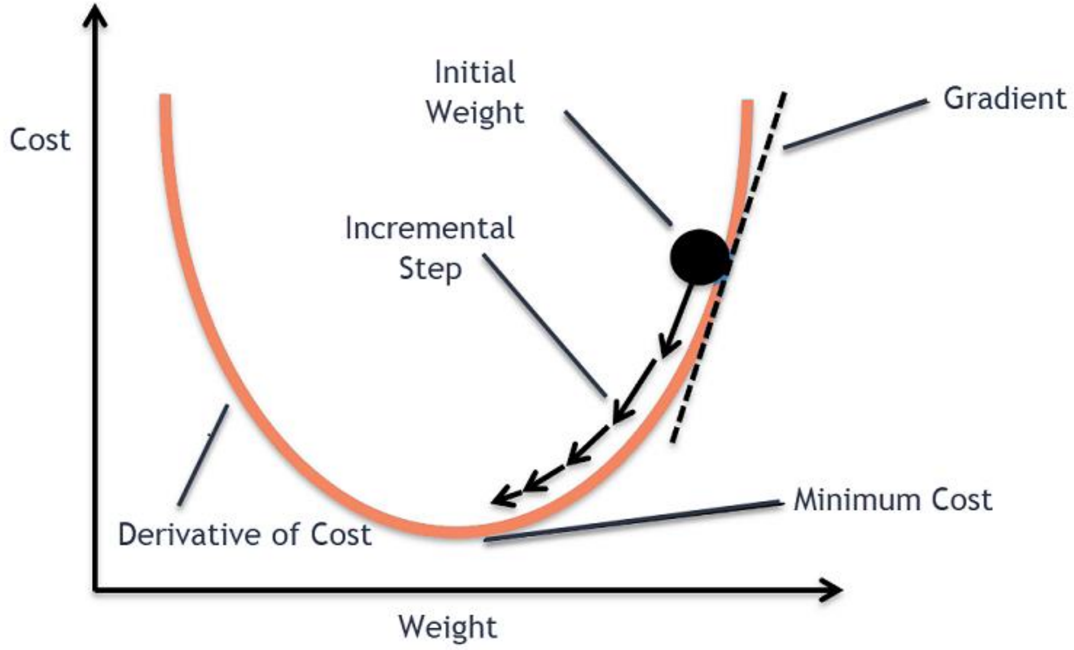
Plot of weights versus MSE

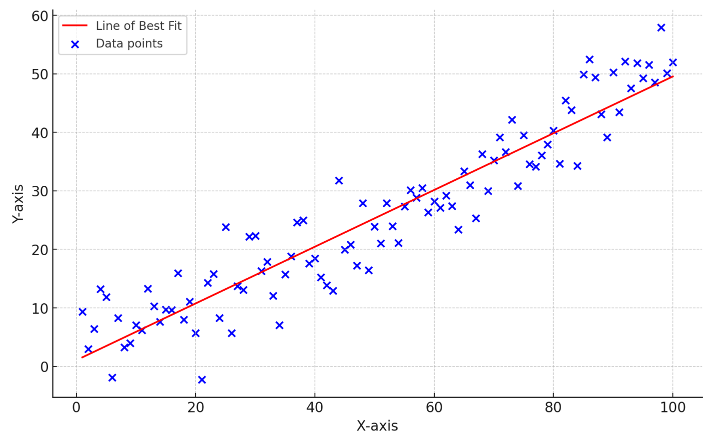
$$\text{MSE} = \overset{\text{Mean}}{\boxed{\frac{1}{n}\sum_{i=1}^{n}}}\Big(\overset{\text{Error}}{\boxed{Y_i - \hat{Y}_i}}\Big)^{\overset{\text{Squared}}{\boxed{2}}}$$

So, we have a network. How do we train it?

- We will train the network through some flavor of *gradient descent*
- We want move towards the minima of our loss function by adjusting the parameters of the network iteratively
- This is done by calculating the gradient of the loss function w.r.t. each parameter that you want to optimize







https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-

PyTorch

PyTorch

- PyTorch is a machine learning framework primarily used in deep learning applications and research

- Has a well-supported Python interface and provides the ability to utilize hardware (GPU) acceleration

- Has a very similar syntax and feel as NumPy
- Can go back and forth between PyTorch and NumPy easily!