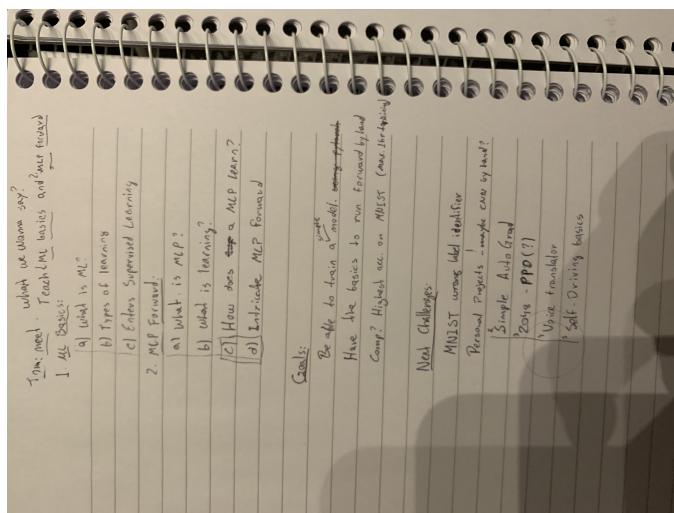




ML Intro Class

▼ Reference



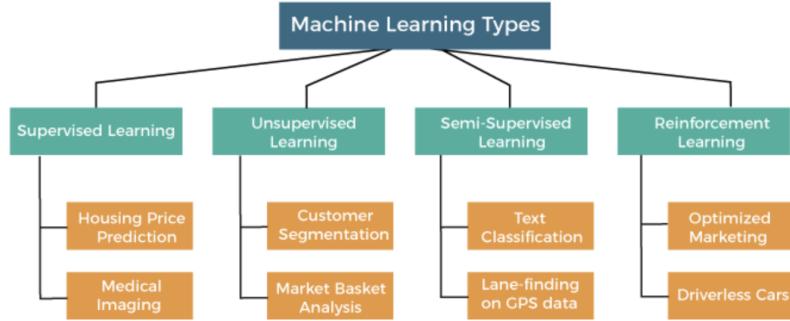
▼ What is ML

The act of receiving a correct output without explicitly programming it

Informal sketch of proof of convergence

- Each time the perceptron makes a mistake, the current weight vector moves to decrease its squared distance from every weight vector in the “generously feasible” region.
- The squared distance decreases by at least the squared length of the input vector.
- So after a finite number of mistakes, the weight vector must lie in the feasible region **if this region exists.**

▼ Types of ML



▼ Self-Supervised Learning

Is it part of semi-supervised learning? Nope

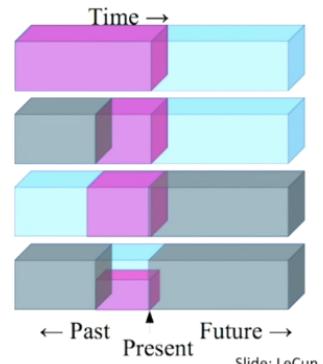
What is self-supervised learning?

Self-supervised learning is a machine learning process where the model trains itself to learn one part of the input from another part of the input. It is also known as predictive or pretext learning.

In this process, the unsupervised problem is transformed into a supervised problem by auto-generating the labels. To make use of the huge quantity of unlabeled data, it is crucial to set the right learning objectives to get supervision from the data itself.

The process of the self-supervised learning method is to identify any hidden part of the input from any unhidden part of the input.

- ▶ Predict any part of the input from any other part.
- ▶ Predict the future from the past.
- ▶ Predict the future from the recent past.
- ▶ Predict the past from the present.
- ▶ Predict the top from the bottom.
- ▶ Predict the occluded from the visible
- ▶ Pretend there is a part of the input you don't know and predict that.



▼ Enters Supervised Learning

▼ The Process

1. Inputting DATA with respective labels (answers / truth values)

2. Through a **MODEL** that outputs a prediction
3. From prediction, get value for error of the prediction AKA: **LOSS**
4. **OPTIMIZE** adjustable parameters (weights) to improve performance on Loss
5. **TEST** your model performance on unseen data (test data)

▼ DATA

Get Data and Labels

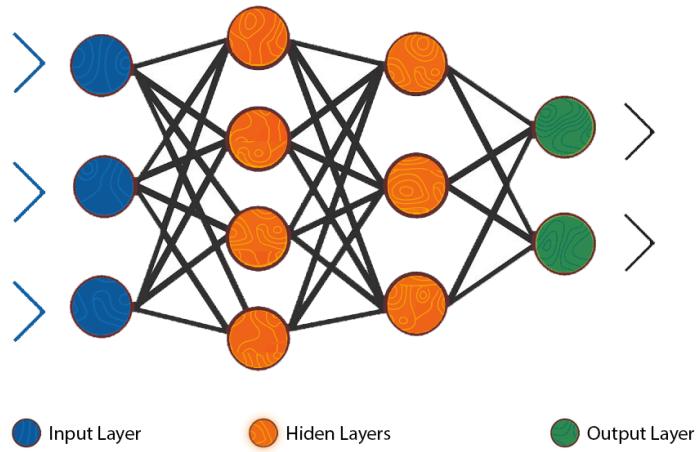
Prepare data

▼ MODEL (Structure)

▼ **Perceptron** (simplified model of biological neuron - initial goal binary classification): $W * X + b$

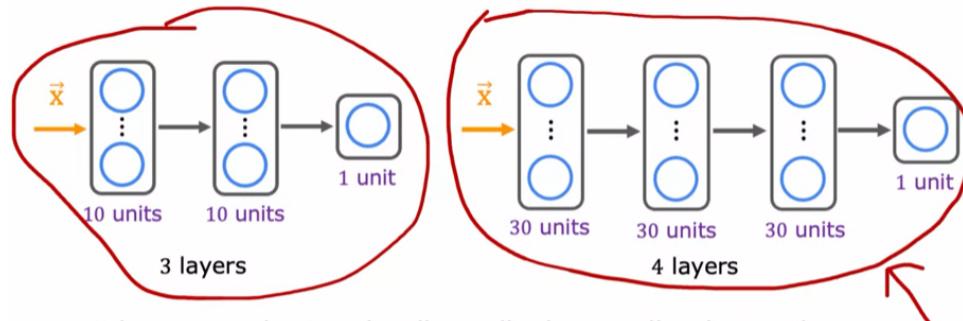


▼ **Multi Layer Perceptron (MLP)**: A fully connected multi-layer neural network



- ▼ **Deep Learning**: more than 2 hidden layers (nowadays way more than that)

Neural networks and regularization



- ▼ What do you have to choose

Output Layer + Layers: neurons, activation functions, other fancy shit

▼ LOSS

Depends on the type of problem you want to solve, there should be a selection (ChatGPT somewhat works) to check which one better fits your problem and data.

A function (good if its convex = differentiable) that outputs a value which considers the error for a given prediction, the closer the prediction to the truth, the better (lower loss).

- ▼ LOSS (COST) Functions

MSE, RMSE - Regression

Binary Cross Entropy - Classification (2 classes)

Cross Entropy - Classification (n classes)

Hinge Loss - Classification (2 classes)

KL Divergence - GAN's, Variational Autoencoder

▼ OPTIMIZE

This is how the *Learning* works.

Here lies the tasty **BACKPROP** - method introduced by Geoffrey Hinton in 1986, which resparked the interest in AI

- ▼ It consists in an algorithm that given a prediction and a loss function you can return a update to the parameters (weights) that improve the performance of the model on the training data.

```
def optim(pred, loss):  
    do computations  
    return updated_weights
```

- ▼ What do you have to do

```
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)  
  
optimizer.zero_grad()  
loss.backward()  
optimizer.step()
```

▼ Optimization algorithms

Gradient Descent

Momentum

RMSProp

ADAM

ADAGrad

▼ TEST

The goal is to see the model performance on the data, however, due to the high capabilities of deep learning, the model is able to “memorize” the training set, thus not performing on any other set of data. This is called **OVERFITTING**. To prevent this we separate a set of the data before even touching the model, called Test set.

A performance metric should be chosen and then tested on the test set (you only run Inference) - only run predictions, do not optimize.

The Famous forward Step

▼ For each Layer

Input - data at beginning

Weights - Learnable parameters

bias - Learnable parameters

$Z = W * \text{Input} + \text{bias}$

$A = \text{activation function}(Z)$

```
A = Layer(input, weight, bias, activation_function)
```

▼ Goals

Train a ML model (which dataset or which problem) - using PyTorch

Able to compute forward by hand

Competition? (problem/dataset) date? by hand?

▼ Next Challenges

MNIST wrong label identifier

Simple AutoGrad || CNN by hand (only forward)