Deep Learning with Pytorch v15

PyTorch

Trainer: Dr. Alfred Ang

TINFOTECH

Website:www.tertiarycourses.com.sg
Email: enquiry@tertiaryinfotech.com

About the Trainer

Dr. Alfred Ang is the founder of Tertiary Courses. He is a serial entrepreneur. He founded OSWeb2Design Singapore Pte Ltd in 1997 offering web development, e-commerce store development, graphics design, ebook publishing, mobile apps development, and digital marketing services. He established the first online gardening store in Singapore, Eco City Hydroponics Pte Ltd in 2000, offering a wide range of gardening products such as seeds, plant nutrients, hydroponics kits etc. Eco City Hydroponics has become the most popular and successful gardening store in Singapore. He founded Tertiary Infotech Pte Ltd in 2012 and transformed the business to a training platform, Tertiary Courses in 2014. Tertiary Courses offers a wide range of SkillsFuture courses for PMETs to upgrade their skills and knowledge. He also established Tertiary Courses Malaysia in 2016. He also founded Tertiary Robotics in 2015 offering Arduino, Raspberry Pi, Microbit and Robotics product

Dr. Alfred Ang earned his Ph.D. from National University of Singapore in 2000, majoring in Electrical and Electronics Engineering. He also completed an online MBA course with U21 Global based in Australia. He obtained his B.Sc (Hons) from National University of Singapore in 1992, majoring in Physics. He topped his Physics cohort for 3 consecutive years and funded his degree study with Book prizes, Study awards, bursaries and tuition. He has worked in Defence, Electronics and Semiconductor Industries. His current interests include Machine Learning, Deep Learning, Artificial Intelligence, Internet of Things, Robotics and Programming.

Dr. Alfred Ang is ACTA certified trainer and DACE certified developer. He is an associate adult educator with IAL, IBM Certified AI Practitioner and IBM Certified Design Thinking Practitioner. He was Distinguished Toastmasters (DTM) and Senior Member of IEEE. He has published more than 20 peer reviewed papers and co-inventors for more than 20 inventions.

Let's Know Each Other...

Say a bit about yourself
- Name
- What Industry you are from?
- Do you have any prior knowledge in Deep Learning or Pytorch?
- Why do you want to learn Deep Learning and Pytorch?
- What do you expect to learn from this course?

Ground Rules

- Set your mobile phone to silent mode
- Actively participate in the class. No question is stupid.
- Respect each other view
- Exit the class silently if you need to step out for phone call, toilet break

Ground Rules for Virtual Training

- Upon entering, mute your mic and turn on the video. Use a headset if you can
- Use the 'raise hand' function to indicate when you want to speak
- Participant actively. Feel free to ask questions on the chat whenever.
- Facilitators can use breakout rooms for private sessions.

Guidelines for Facilitators

1. Once all the participants are in and introduce themselves
2. Goto gallery mode, take a snapshot of the class photo - makes sure capture the date and time
3. Start the video recording (only for WSQ courses)
4. Continue the class
5. Before the class end on that day, take another snapshot of the class photo - makes sure capture the date and time
6. For NRIC verification, facilitator to create breakout room for individual participant to check (only for WSQ courses)
7. Before the assessment start, take another snapshot of the class photo - makes sure capture the date and time (only for WSQ courses)
8. For Oral Questioning assessment, facilitator to create breakout room for individual participant to OQ (only for WSQ courses)
9. End the video recording and upload to cloud (only for WSQ courses)
10. Assessor to send all the assessment records, assessment plan and photo and video to the staff (only for WSQ courses).

Prerequisite

- Basic Python
- Some knowledge of Machine Learning

Agenda

Topic 1 Overview of Deep Learning and Pytorch

- Overview of Deep Learning
- Introduction to Pytorch
- Install and Run Pytorch
- Basic Pytorch Tensor Operations
- Computation Graphs
- Compute Gradients with Autograd

Topic 2 Neural Network for Regression

- Introduction to Neural Network (NN)
- Activation Function
- Loss Function and Optimizer
- Machine Learning Methodology
- Build a NN Predictive Regression Model
- Load and Save Model

Topic 3 Neural Network for Classification

- Softmax
- Cross Entropy Loss Function
- Build a NN Classification Model

Agenda

Topic 4 Convolutional Neural Network (CNN)

- Overview of CNN
- Convolution, Max Pooling and Padding
- Build a CNN Model for Image Classificaiton
- Overfitting Issue with Small Dataset
- Techniques to overcome Overfitting Issue

Topic 5 Transfer Learning

- Introduction to Transfer Learning
- Pre-trained Models
- Feature Extraction & Fine Tuning for Small Dataset

Topic 6 Recurrent Neural Network (RNN)

- Overview of RNN
- Long Term Dependencies
- LSTM and GRU
- Apply LSTM to Time Series Forecasting

Download Course Material

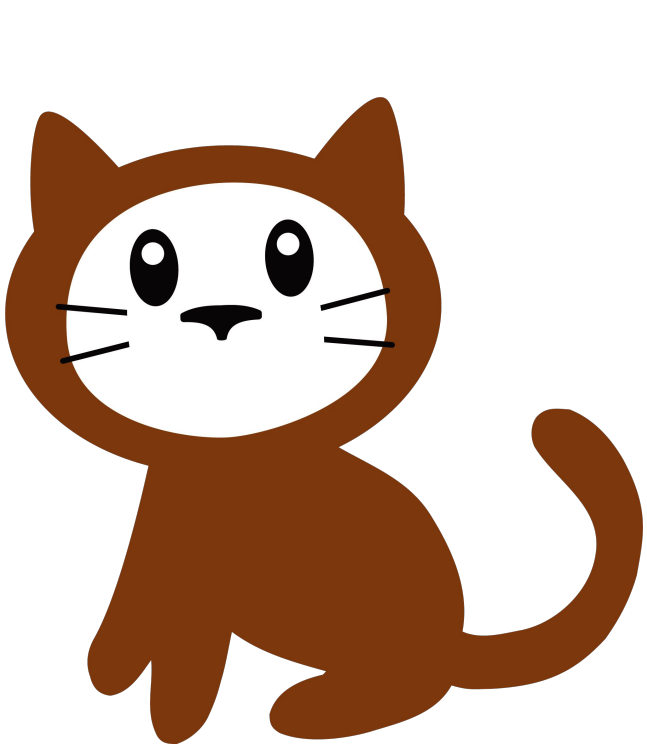
- You can download the course material from Google classroom https://classroom.google.com
- Enter the class code below to join the class on the top right.
- Goto Classwork>> Course material
- If you cannot access the Google Classroom, please inform trainer or staff.

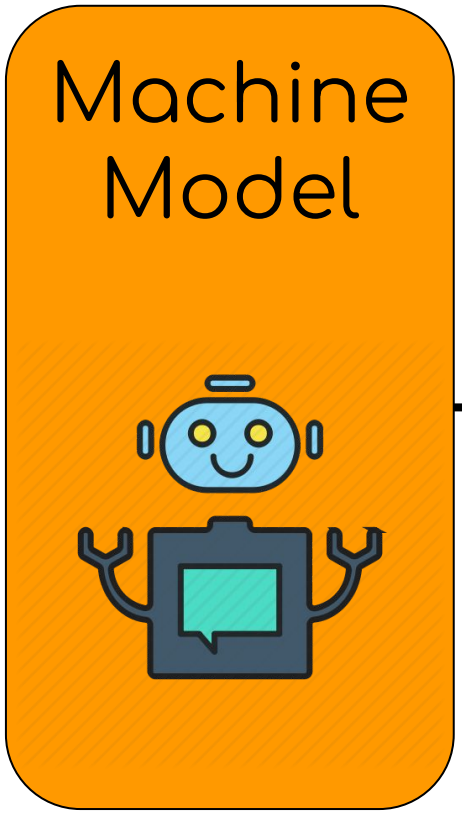
nn7x4ou

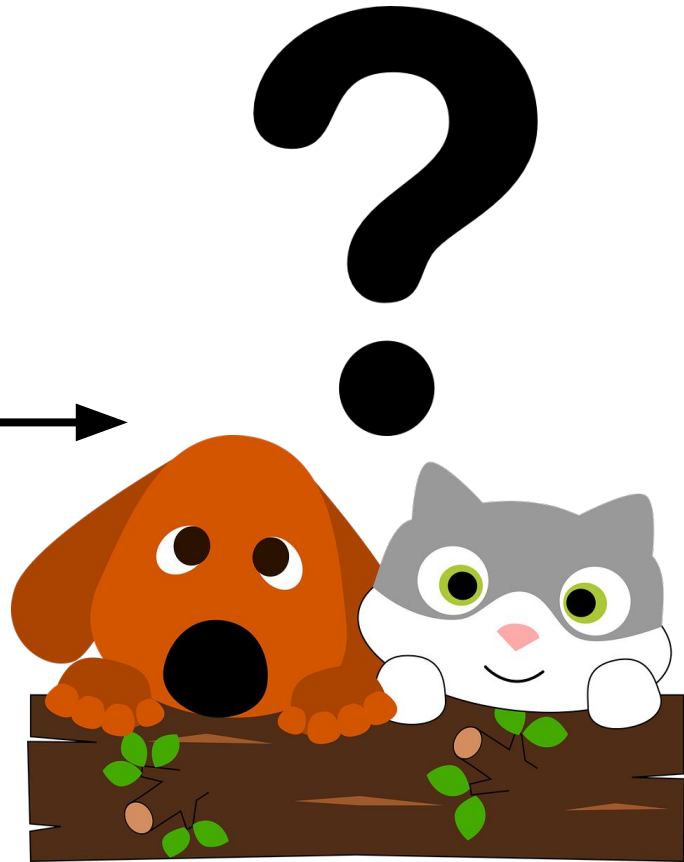
Topic 1
Overview of Deep Learning and Pytorch


So, do you
see a crow
...or a
cat?

After you trained the machine to recognize dog or cat, then machine is able to tell your the answer when you give it an image







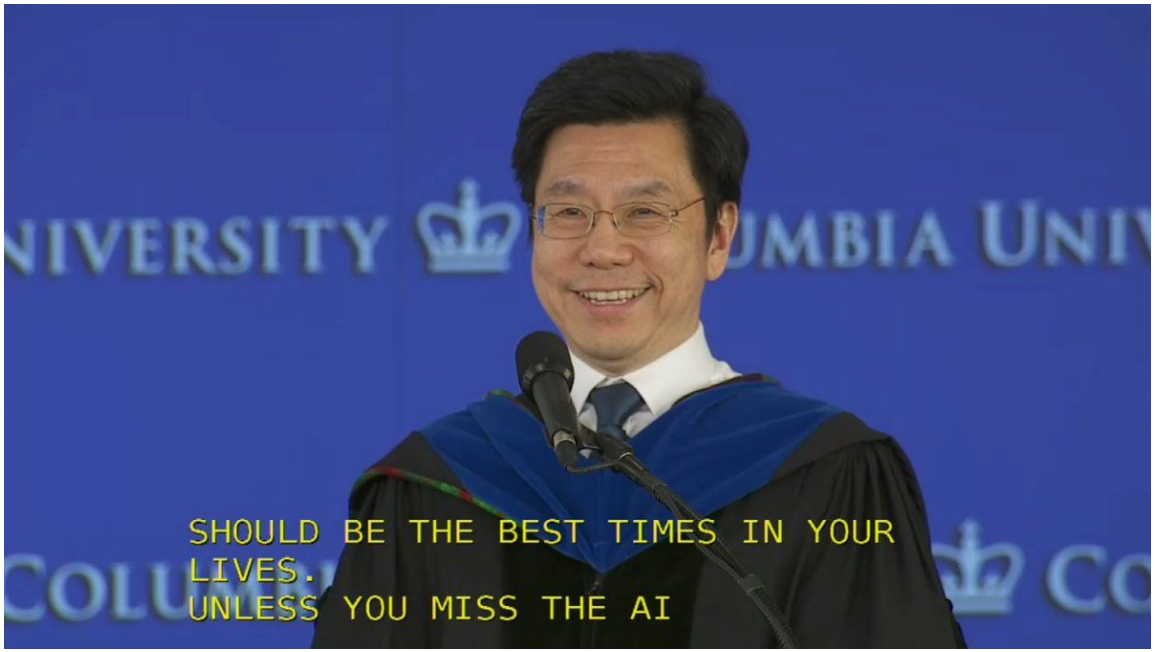
Why Machine Learning?

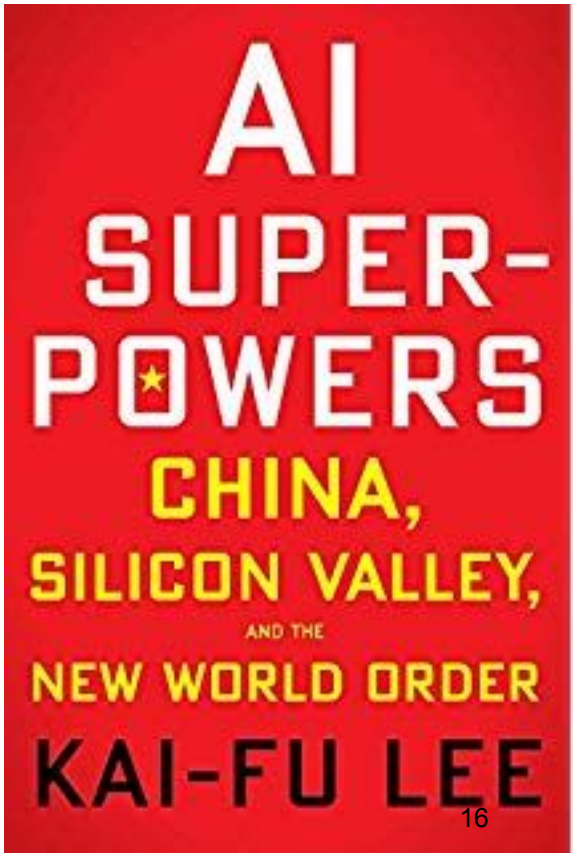
Google AlphaGO (A Deep NN) beat World GO Champion Ke Jia on May 2017


Deep Learning
“AI is the new electricity. It has the potential to transform every industry and to create huge economic value”
--Andrew Ng

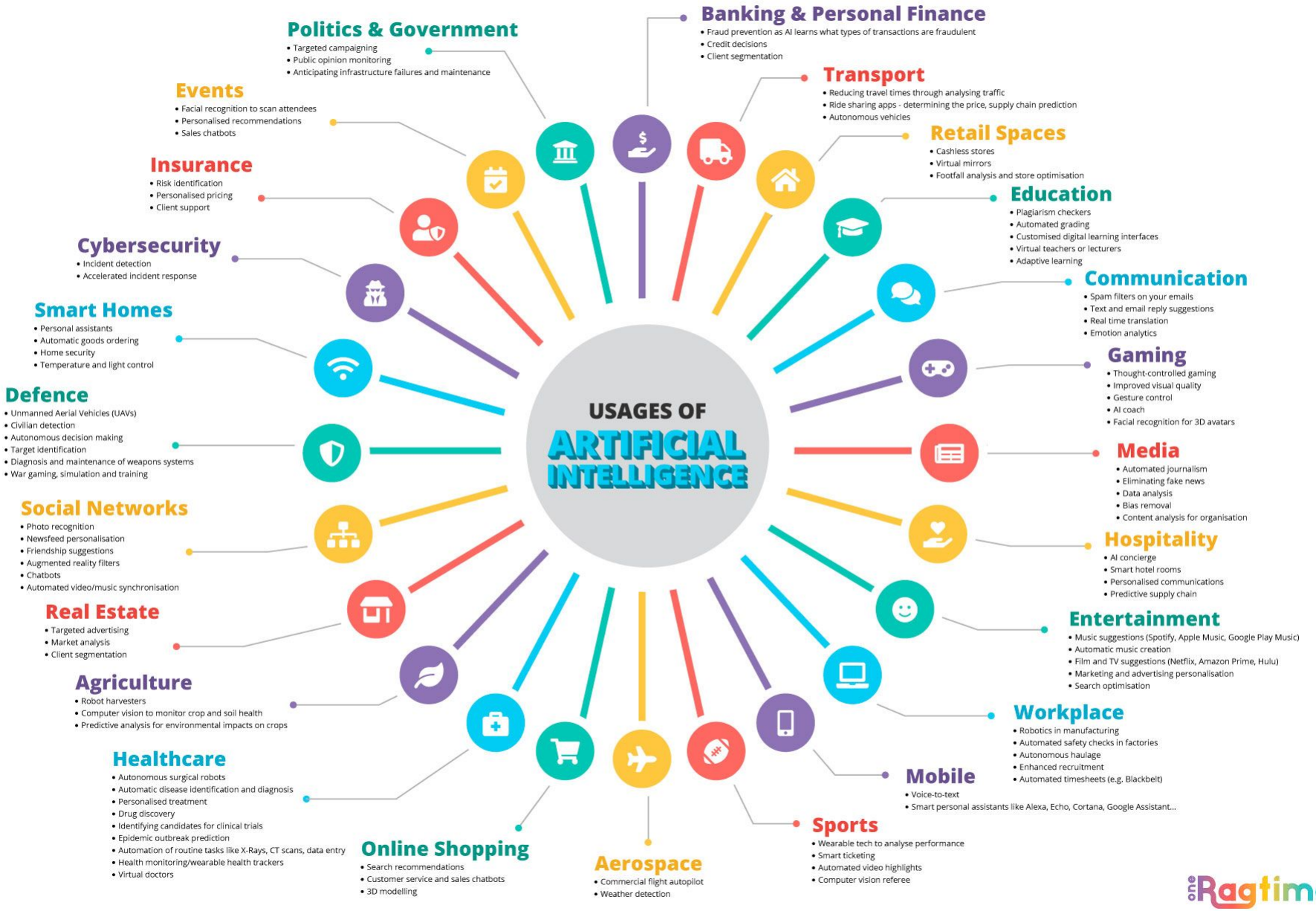
Will AI take over Human Jobs?

"Artificial intelligence will replace half of all jobs in the next decade" - Lee Kai Fu



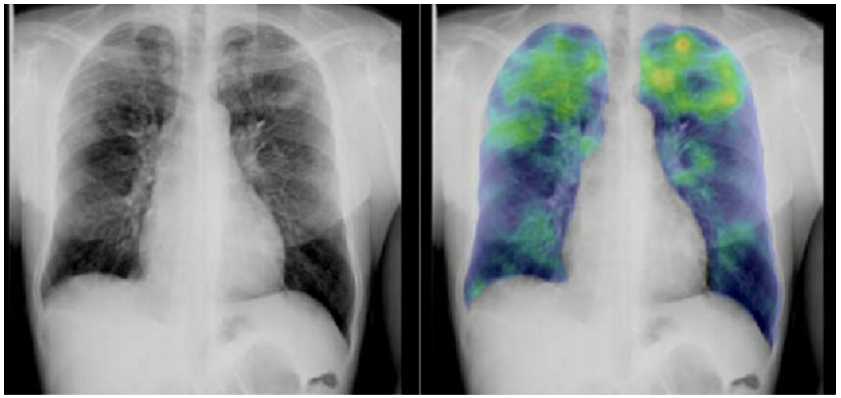


Machine Learning Applications

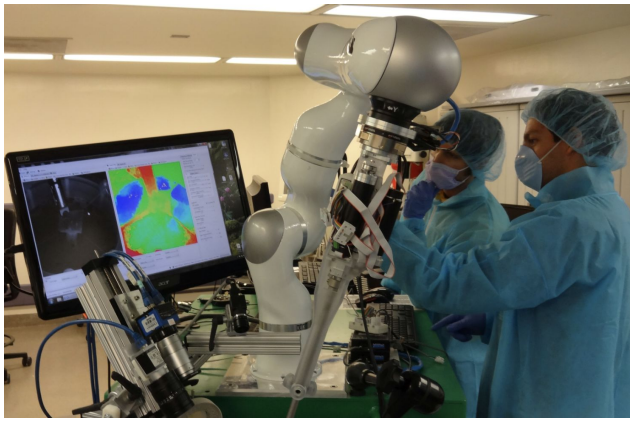


ML Applications on Health Care

Identifying tuberculosis

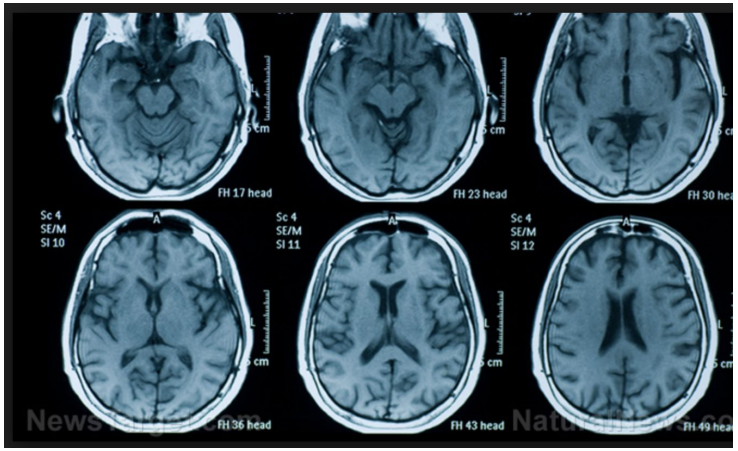


Robotics-assisted surgery



Detecting brain bleeds

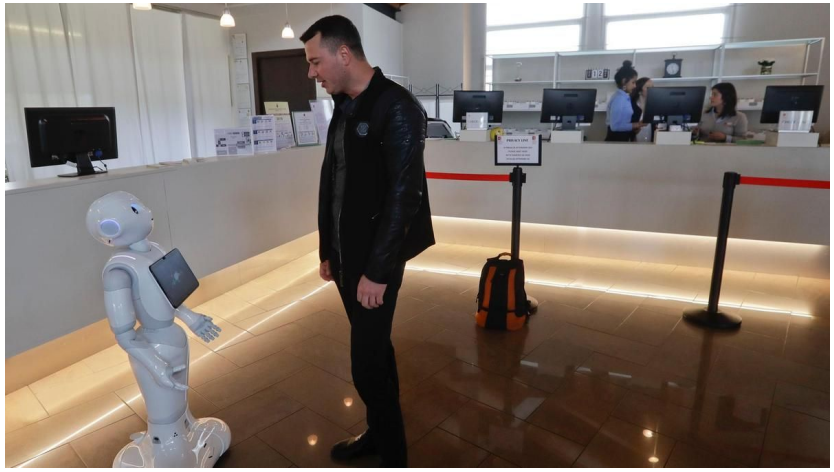
Detecting Alzheimer's disease



ML Applications on Retail/Logistics

Amazon Go

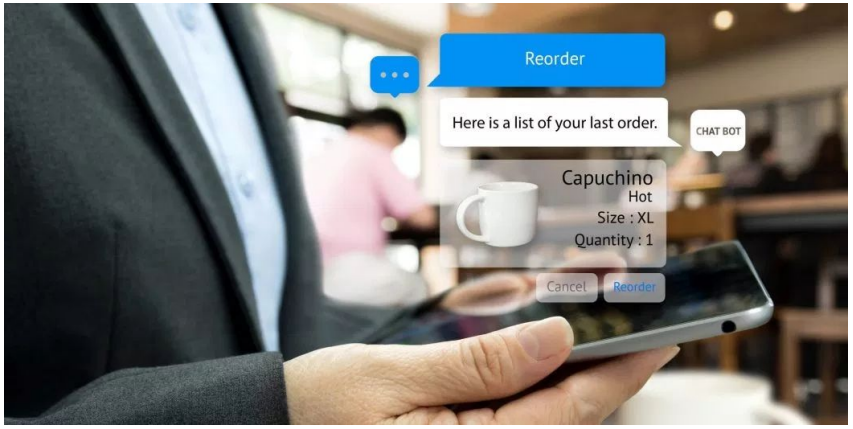
Frontdesk Robots



Alibaba Unmanned store

阿里巴巴无人超市

Chatbot for retail/services

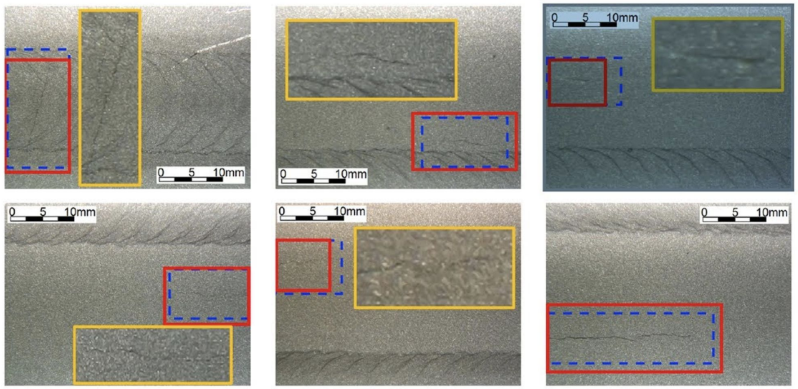


ML Applications on Security/Mfg

Face recognition

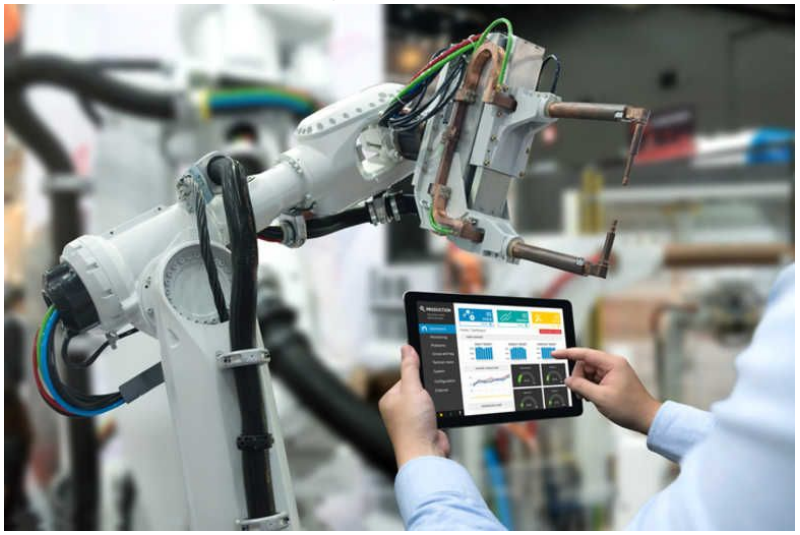
identifying criminals

Defect detection



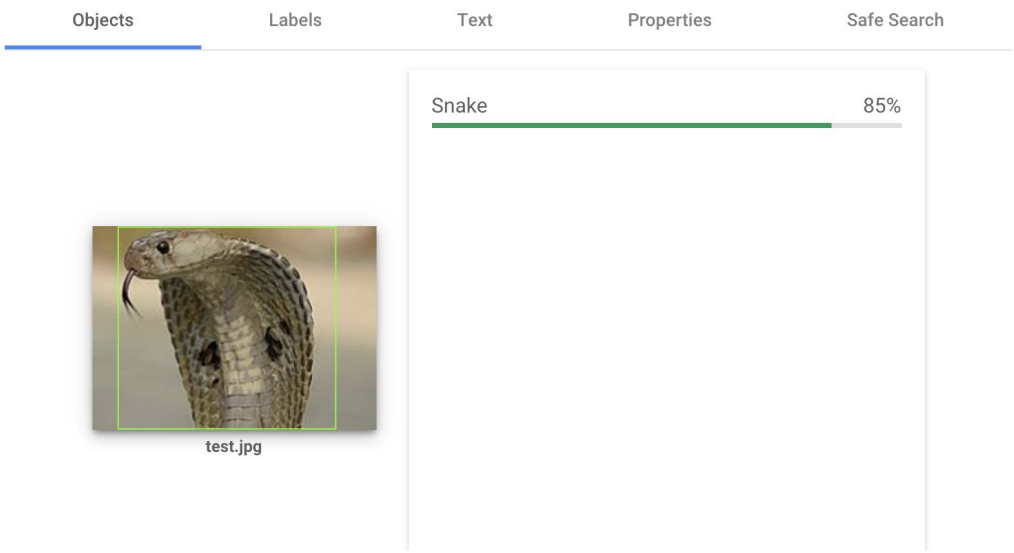
Parking Spotter

Robots in Mfg

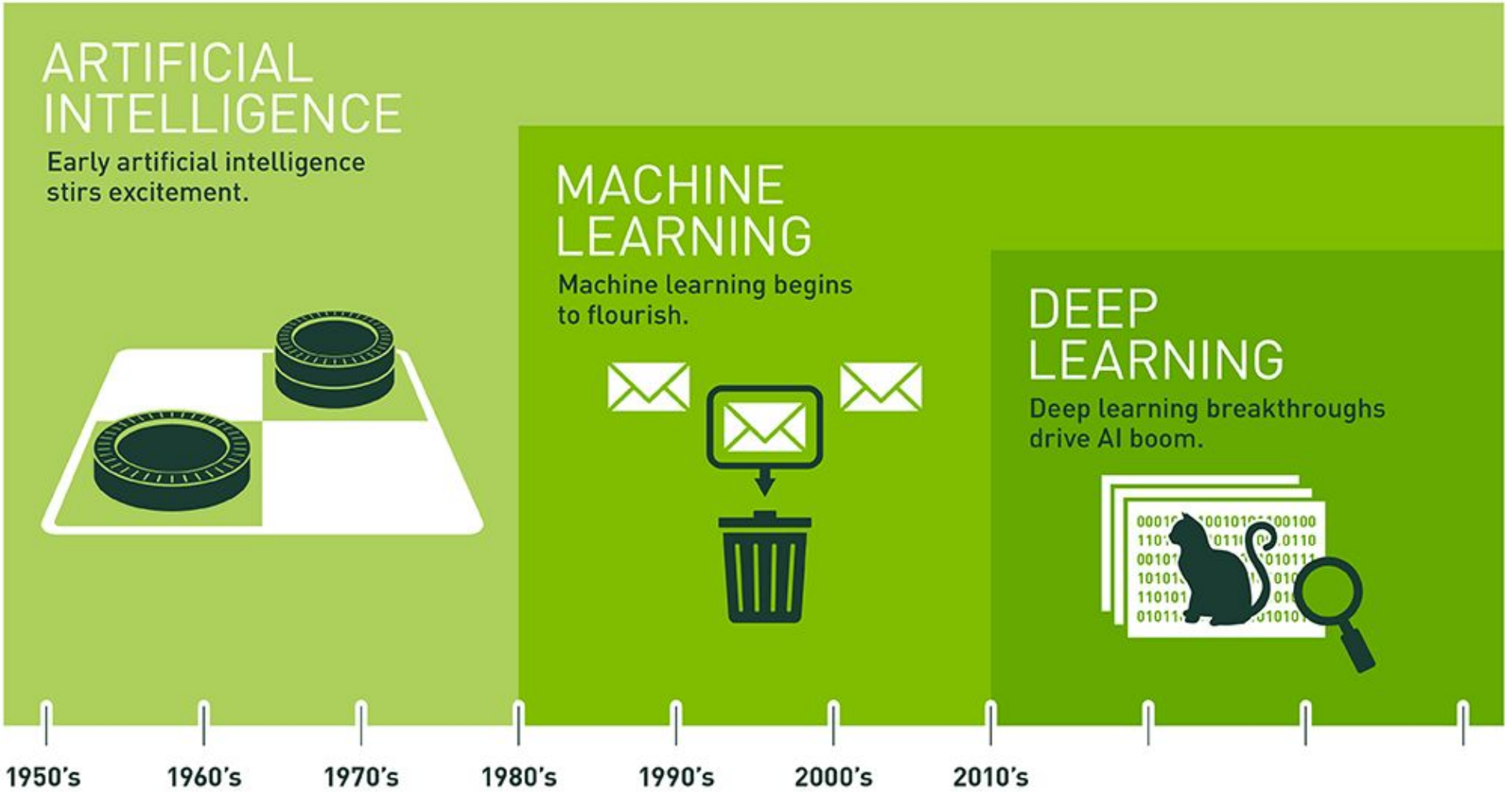


Activity : Vision

- Upload some images from internet
- Use the following Vision tools to classify the images
  - Google Vision Demo: https://cloud.google.com/vision
  - IBM Vision Demo: https://watson-visual-recognition-duo-dev.ng.bluemix.net/

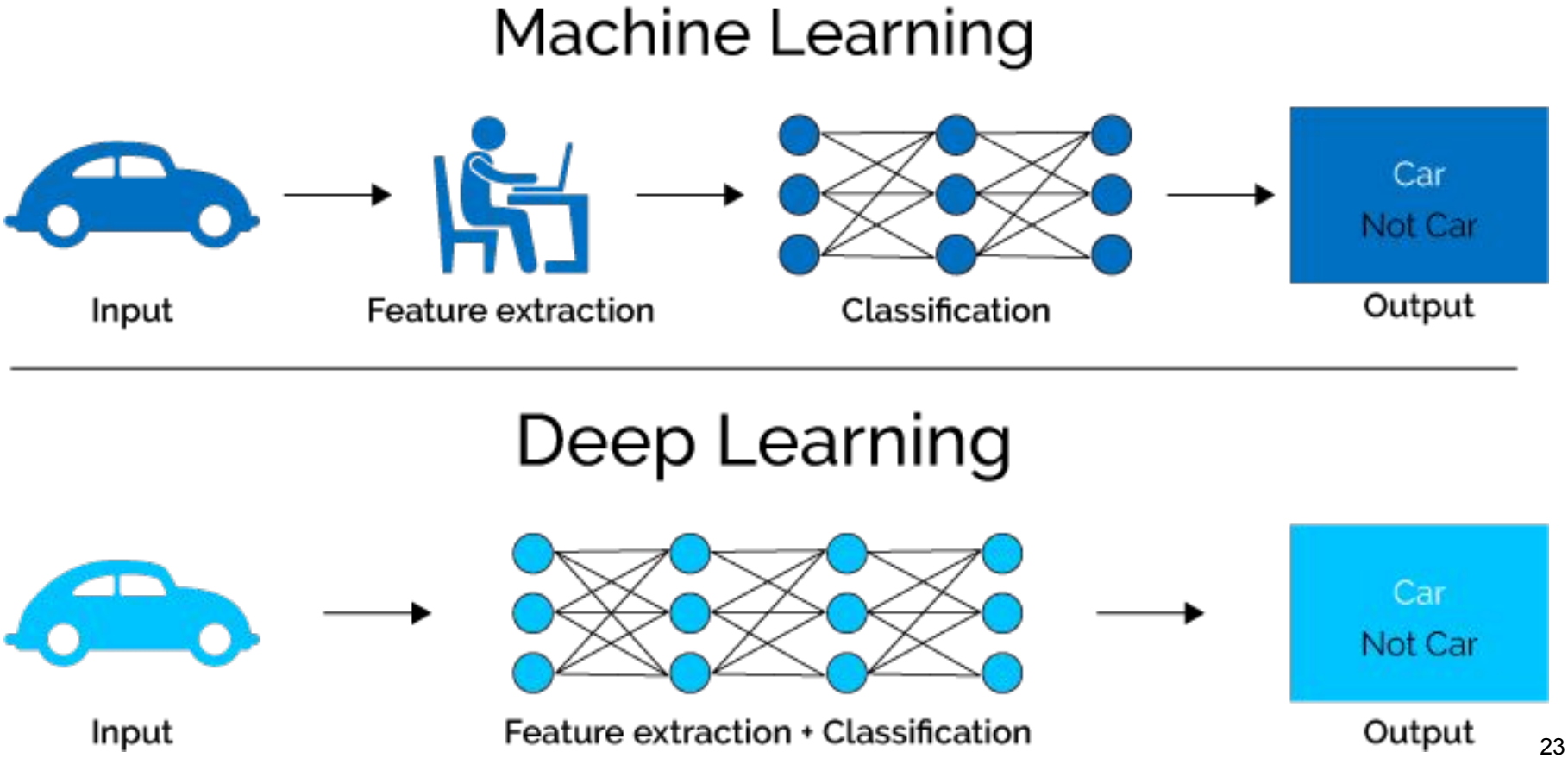


AI vs ML vs DL


ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

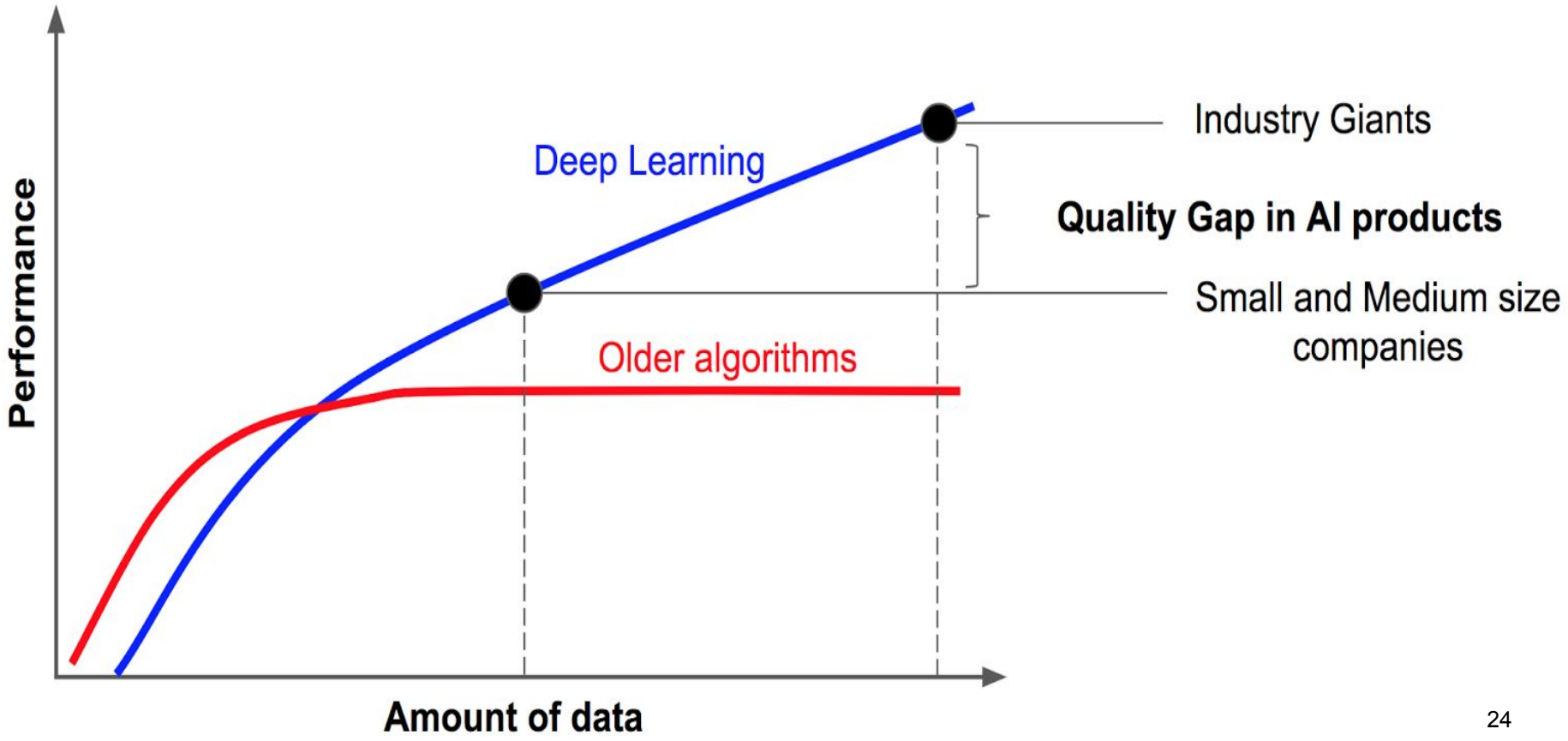
Machine Learning vs Deep Learning

Deep Learning are neural network based and consists of millions of parameters to train.



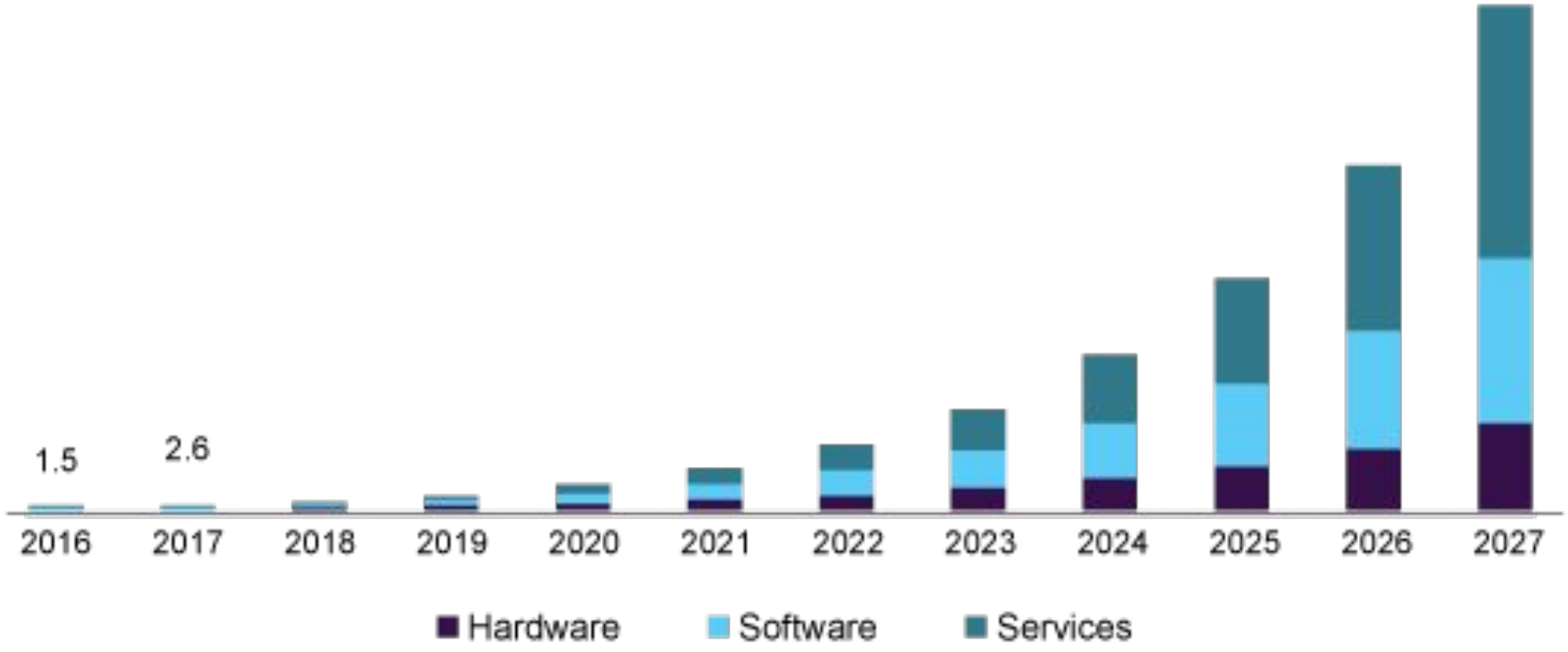
Why Deep Learning?

The performance of Deep Learning surpasses classical machine learning with millions and millions of data.



Projected AI Market Growth

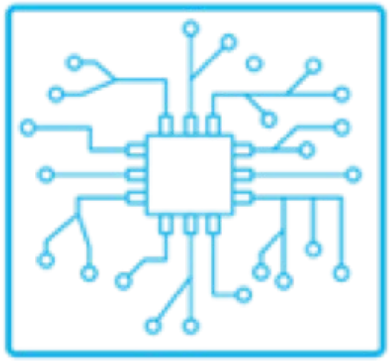
Projected Asia Pacific AI market size growth from 2016 to 2027 (USD Billion)



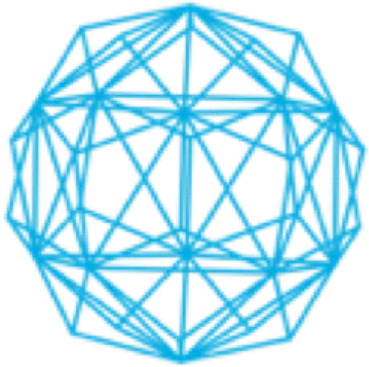
Source: www.grandviewresearch.com

Driving Forces of AI Growth

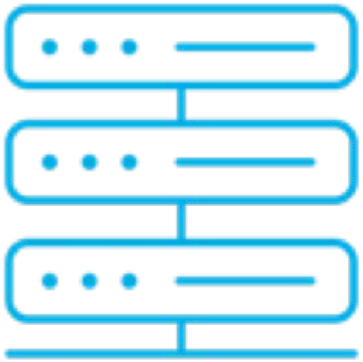
The three driving forces of AI growth is increasing computing power, algorithm and data availability



Computing Power



Algorithm Power

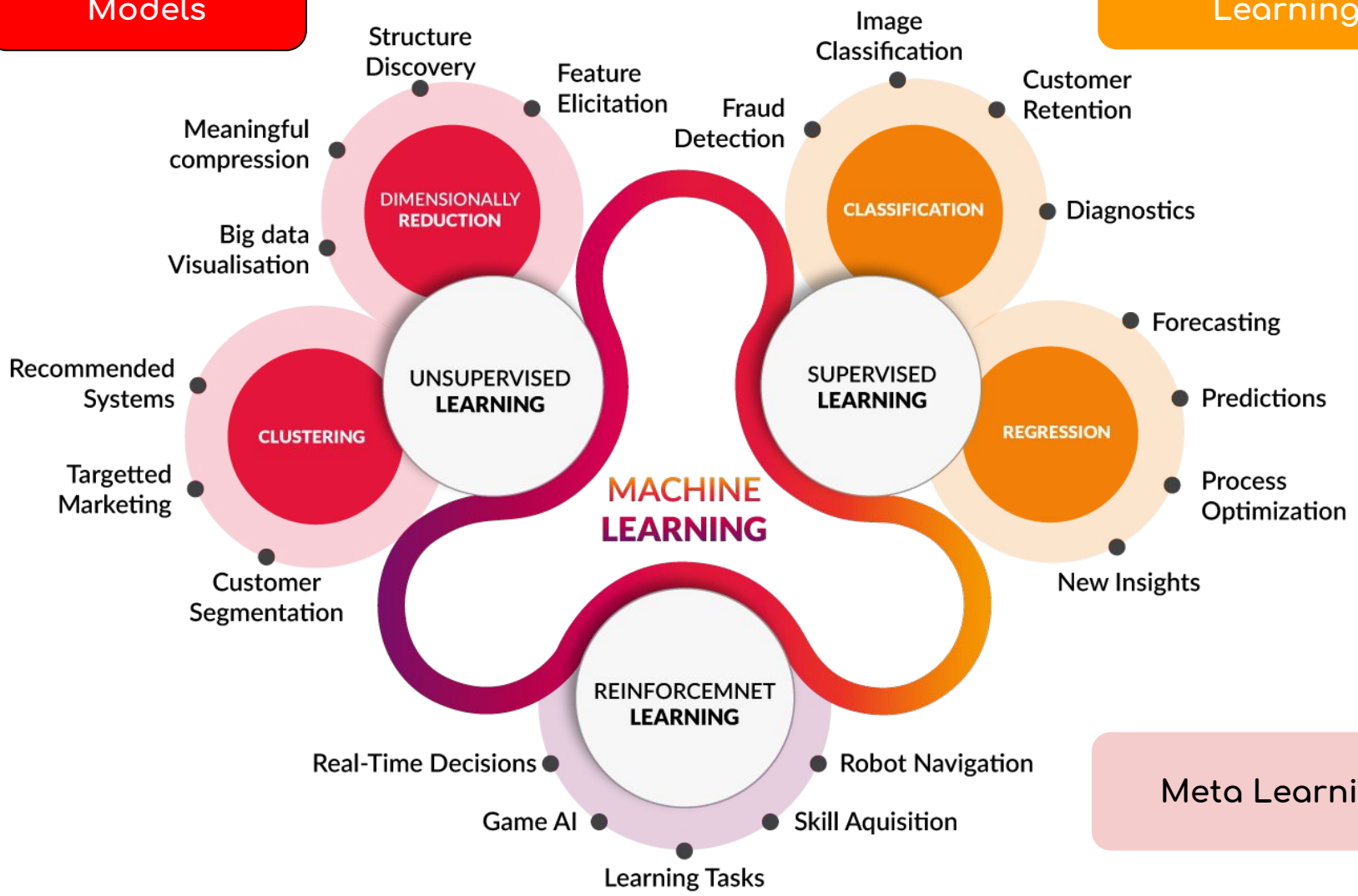


Data Availability

Types of Machine Learning

GAN Generative Models

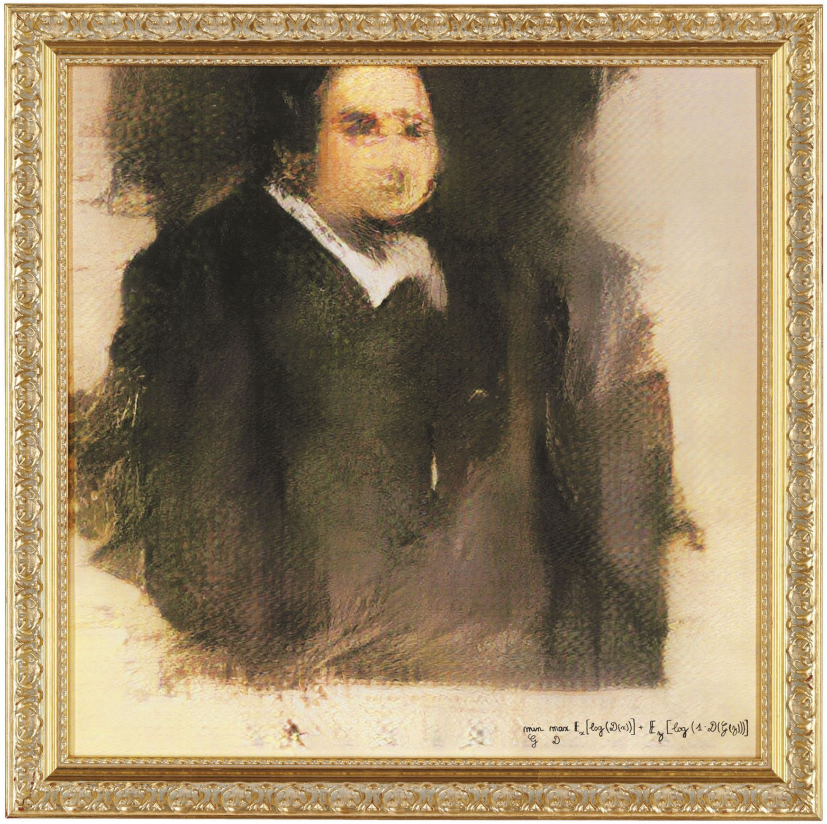
Semi Supervised Learning

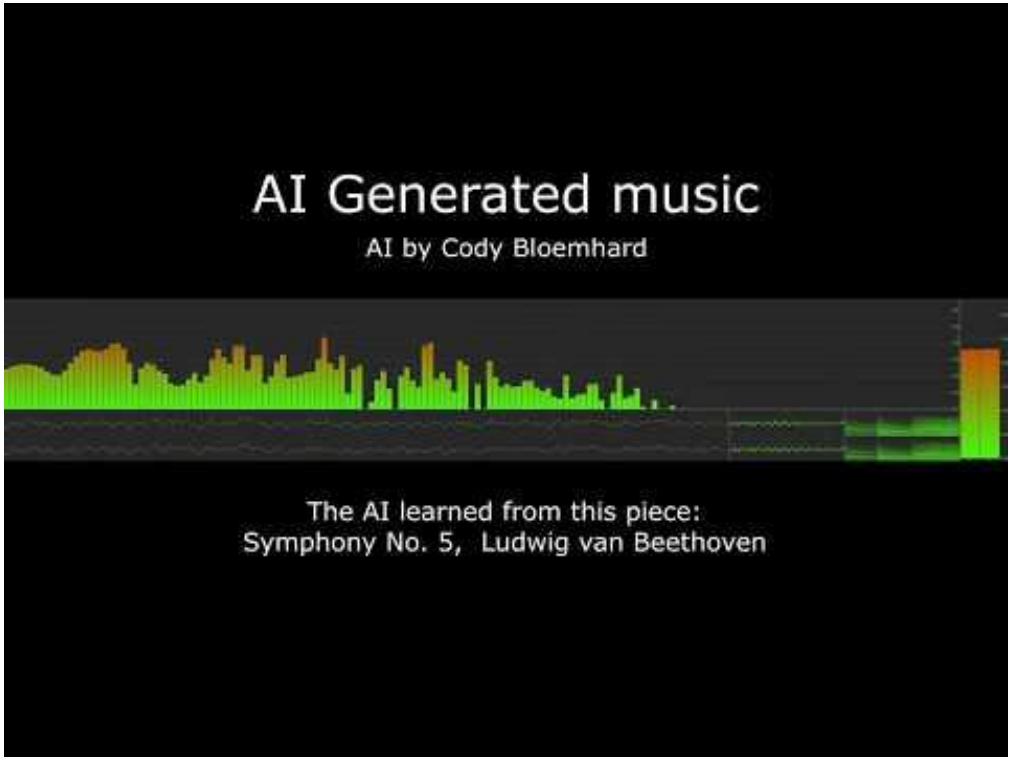


Meta Learning

AI Generated Content - Big Business

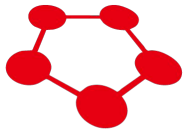
AI-Generated Portrait Sells for $432,500
https://www.cbsnews.com/news/ai-generated-portrait-sells-for-stunning-432500-portrait-of-edmond-de-belamy/



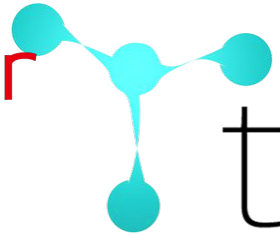


AI Generated Fake News

Machine Learning Frameworks



Chainer



Caffe

torch

theano

Machine Learning on Cloud

Google Cloud

IBM Cloud

Alibaba Cloud

What is Pytorch?

- PyTorch (http://pytorch.org/) is a python package developed by Facebook
- The key features of Pytorch:
  - Tensor computation with strong GPU acceleration
  - Deep Neural Networks built on a reverse-mode automatic differentiation
- Pytorch is a rival DL platform against Tensorflow developed by Google

Who use Pytorch?

Install Pytorch on Colab

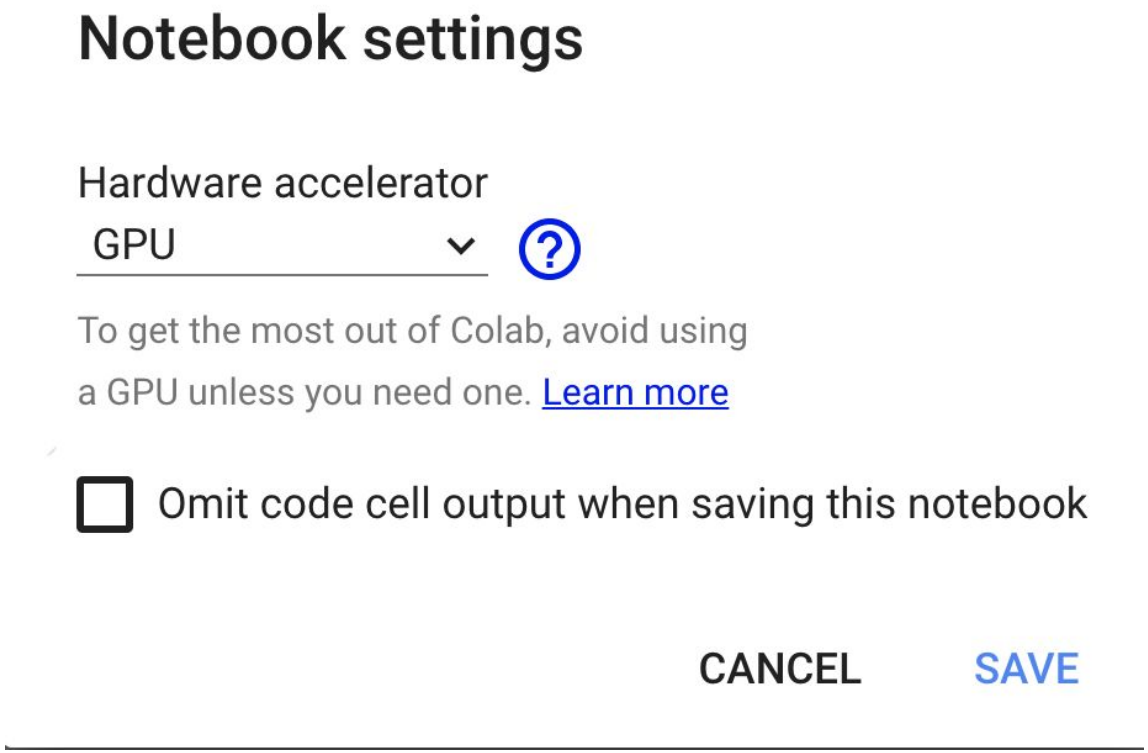
- Google Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.
- You can also access computing resources such as GPU and TPU for free
- You can get started with Google Colab from https://colab.research.google.com/

- Pytorch is pre-installed in Colab. You can re-install the latest version of using the command below in Colab

  !pip3 install torch torchvision torchaudio

Setup GPU on Colab

To use GPU, Goto Edit->Notebook Setting, set hardware accelerator to GPU



Check Pytorch Version

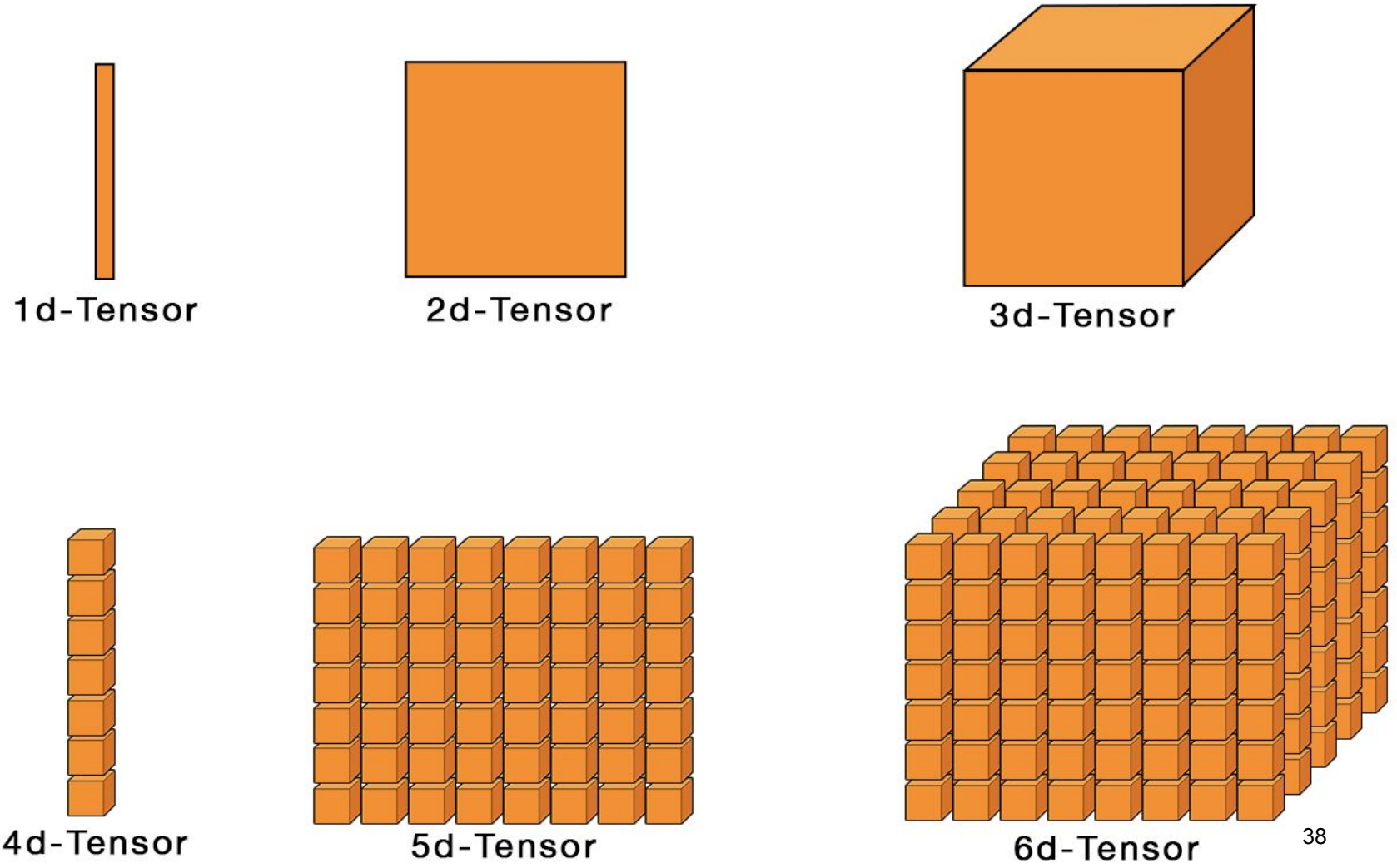
```
import torch
print(torch.__version__)
```

Check GPU Resource

```
print("Cuda Current Device: ", torch.cuda.current_device())
print("Cude Device Count: ", torch.cuda.device_count())
print("Cude Device Name: ", torch.cuda.get_device_name(0))
print("Cude Device Available : ", torch.cuda.is_available())
```

What is a Tensor?

Tensor is a multi-dimensional array (data)



Create Torch Tensor

- Torch tensors are quite similar to Numpy arrays
- However, unlike Numpy arrays, Torch tensors are required for computing gradient (backpropagation) and GPU.
- There are two ways to create Torch tensor
  - torch.Tensor(x)
  - torch.tensor(x)
- torch.tensor is recommended as it can infer the data types from x (input) even for complex numbers.

Create Tensors from Python List

Tensors can be created from Python lists with the torch.tensor() function

Create a Vector

a = [1, 2, 3]
b = torch.Tensor(a) -> always convert to float data type
b = torch.tensor(a) -> keep to same data type (integer)

Create a Matrix

a = [[1, 2, 3], [4, 5, 6]]
b = torch.Tensor(a)

Torch & Numpy Conversion

- To convert torch tensor to numpy array
  b = a.numpy()

- Convert numpy array to torch tensor:
  b = torch.from_numpy(a)

Torch vs Numpy Functions

Numpy:

```
a = [1,2,3,4]
b = np.array(a)
c = np.sum(b)
d = np.mean(b)
e = np.max(b)
```

Torch:

```
a = [1.,2.,3.,4.]
b = torch.tensor(a)
c = torch.sum(b)
d = torch.mean(b)
e = torch.max(b)
```

Torch Math Operations

- torch.abs(a)
- torch.pow(a, 2)
- torch.ceil(a)
- torch.floor(a)
- torch.add(a,b)
- torch.mul(a, b)
- torch.sum()
- torch.sqrt(a)
- torch.mean(a)
- torch.median(a)
- torch.eq(a,b)
- torch.gt(a,b)
- torch.le(a,b)
- torch.max(a)
- torch.min(a)
- torch.sort(a)

Compute with GPU

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

a = torch.tensor([1,1]).to(device)
b = torch.tensor([2,2]).to(device)
c = a+b
```

- To revert to cpu

```
c.cpu()
```

To convert to numpy, need to revert to cpu first

```
c.cpu().numpy()
```

Activity: Compute with GPU

Compute the following operations with GPU and convert to numpy array

a = 3
b = 4
c = 5
d = (a*b)+c

Ans: 17

Matrix Multiplication

- torch.mm performs a matrix multiplication of the matrices

Eg

```
mat1 = torch.randn(2, 3)
mat2 = torch.randn(3, 3)
torch.mm(mat1, mat2)
```

Activity: Matrix Multiplication

Given

x = [[1,1]]
w = [[1,2],[3,4]]
b = [[2,2]]

Compute the outcome of x*w+b

Ans: [6,8]

Generate Special Torch Tensors

```
torch.zeros(2, 3) # zero matrix
torch.eye(3) # diagonal matrix
torch.linspace(1,5,10) # sequence equal spaced
torch.range(0,10,2) # sequence with specific spacing
torch.rand(5, 3) # random matrix from 0 to 1
torch.randn(5, 3) # normal distributed random matrix
```

Torch Max

- torch.max(a,dim) returns the maximum value of each row of the input Tensor in the given dimension dim.
- The second return value is the index location of each maximum value found (argmax).

```
a = torch.Tensor([[1,0,0],[1,0,0],[0,1,0],[0,0,1]])
print(torch.max(a,1))
```

Reshape Tensor

You can use **view** or **reshape** functions to reshape a tensor

Examples
a = torch.linspace(1,10,10).view(2,5)
a = torch.linspace(1,10,10).reshape(2,5)

a = torch.linspace(1,10,10).view(-1,2)
a = torch.linspace(1,10,10).reshape(-1,2)

Squeeze and Unsqueeze Dim

```
x = torch.linspace(0, 5, 5)
print(x)
x = torch.unsqueeze(x, dim=0)  # row
print(x)

x = torch.linspace(0, 5, 5)
print(x)
x = torch.unsqueeze(x, dim=1) # column
print(x)
```

Concatenate

```
x = torch.tensor([1,2,3])
y = torch.cat((x,x,x))
```

Transpose

```
x = torch.tensor([[1,2],[3,4]])
y = torch.t(x)
```

Activity: Torch Operations

Perform the following torch operations in Pytorch

x = [1,1] => convert to 1x2 matrix using unsqueeze
w = [[1,2],[3,4]]
b = [[2],[2]] => transpose the vector

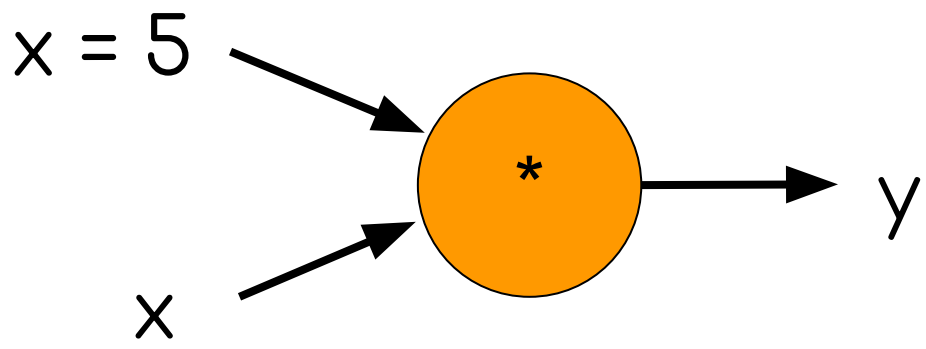
y = x*w+b

Ans: [6, 8]

Gradient

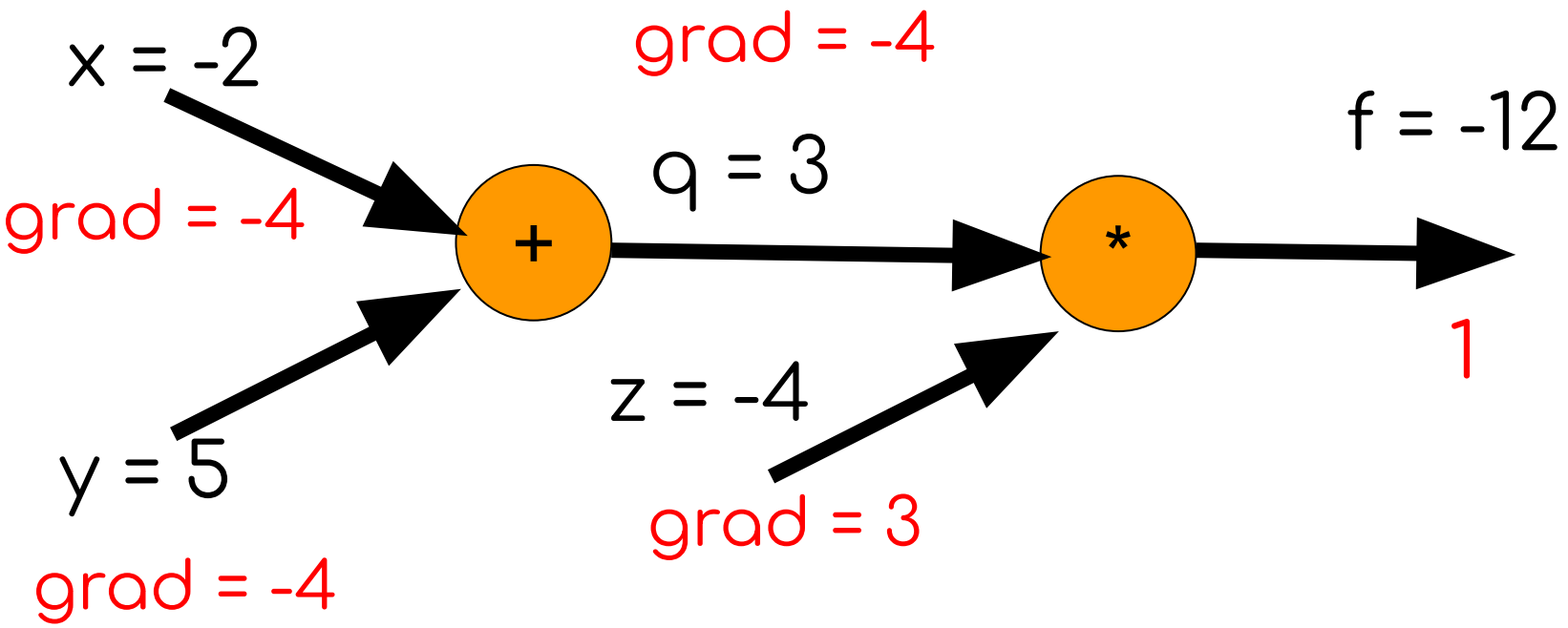
```
x = torch.tensor([5], requires_grad=True)
y = x*x
y.backward()
x.grad.item()
```

x.grad = 2*x = 2*5=10



Compute Gradients

Compute gradients from computation graph
f = (x+y)*z = q*z



Compute Gradients

```
x = torch.tensor([-2.], requires_grad=True)
y = torch.tensor([5.],requires_grad=True)
z = torch.tensor([-4.], requires_grad=True)
f = (x+y)*z

f.backward()

print('x gradient = ',x.grad.item())
print('y gradient = ',y.grad.item())
print('z gradient = ',z.grad.item())
```

Difference between .item and .data

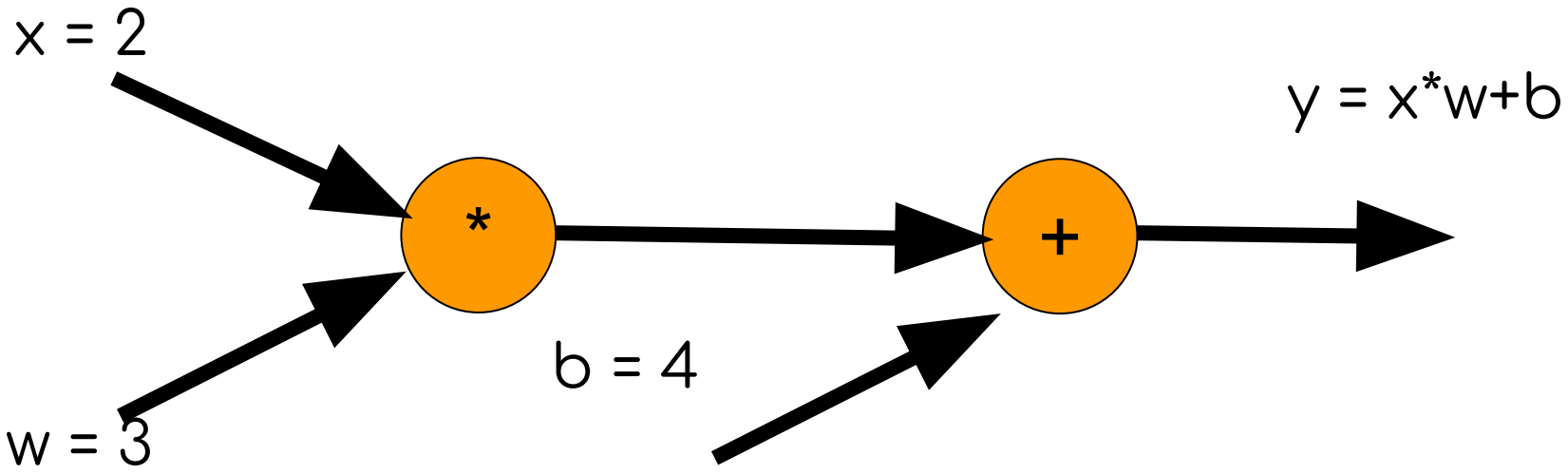
```
a = torch.randn(1)
print(a.item())
print(a.data)
print(a)
```

Output

```
-1.1979358196258545
tensor([-1.1979])
tensor([-1.1979])
```

Activity: Compute Gradients

Compute gradients at w and b for y = x*w+b



Answer:
x gradient = 3.0
w gradient = 2.0
b gradient = 1.0

Activity: Compute Gradients

Compute gradients for the following functions for x=1

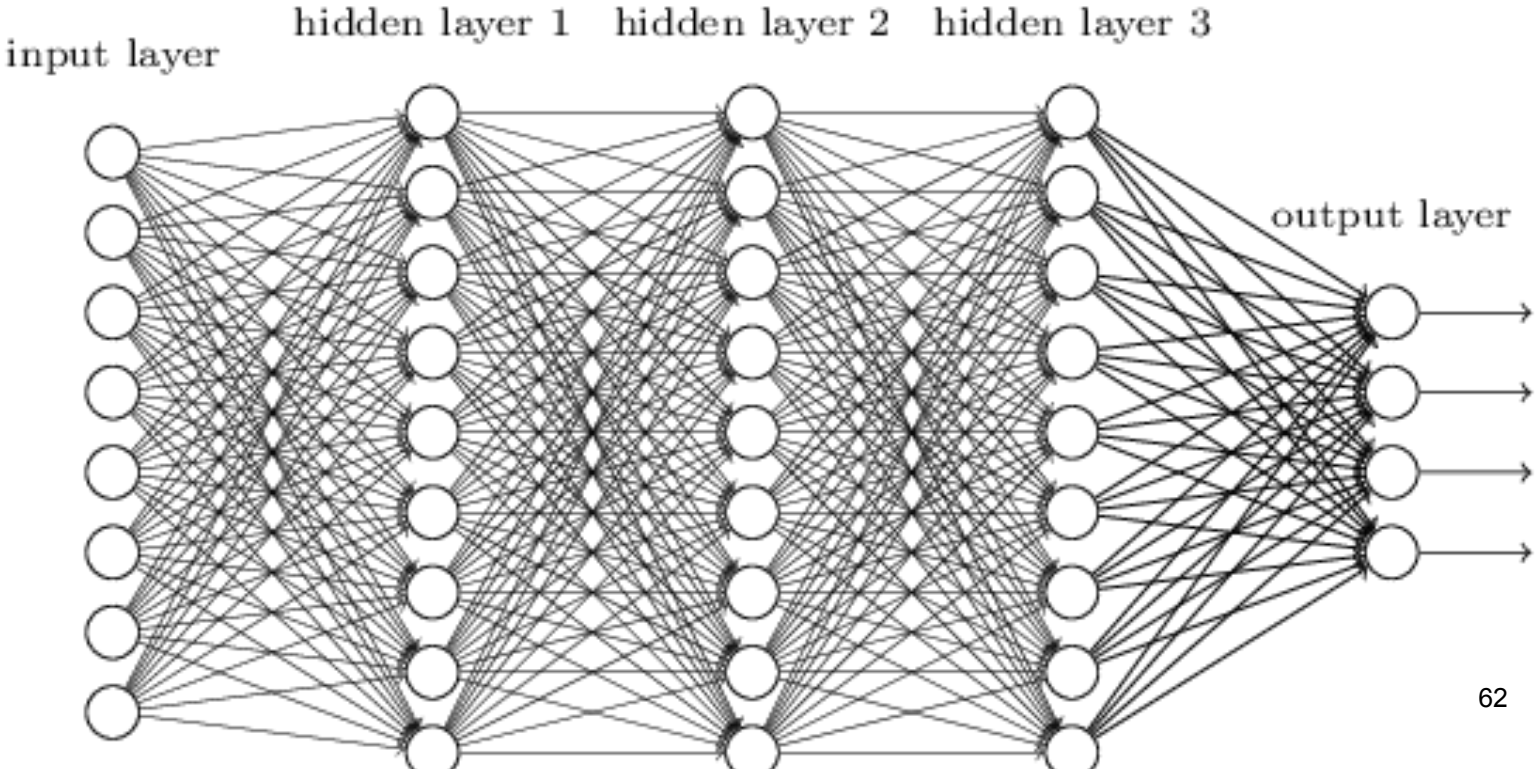
$$z = 2y^2$$
$$y = x^3$$

Answer: 48

Topic 2
Neural Network for Regression

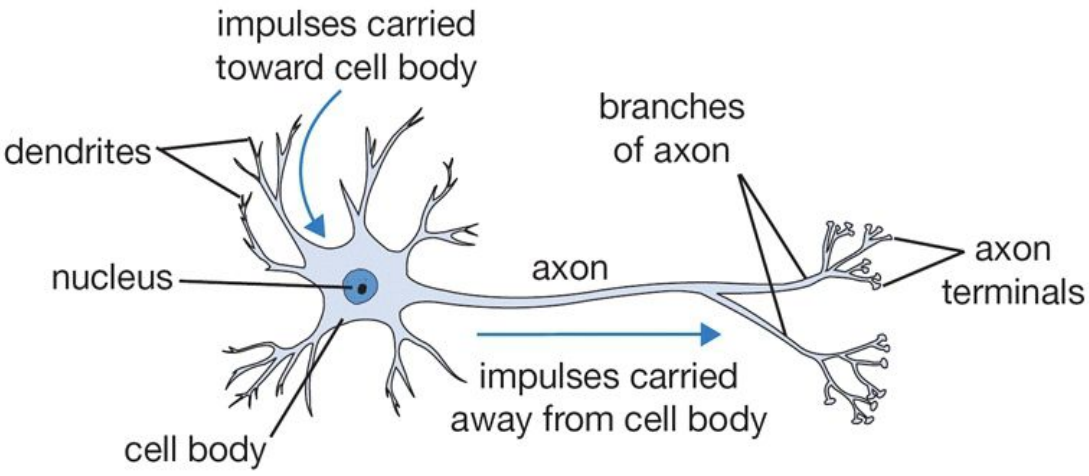
Neural Network (NN)

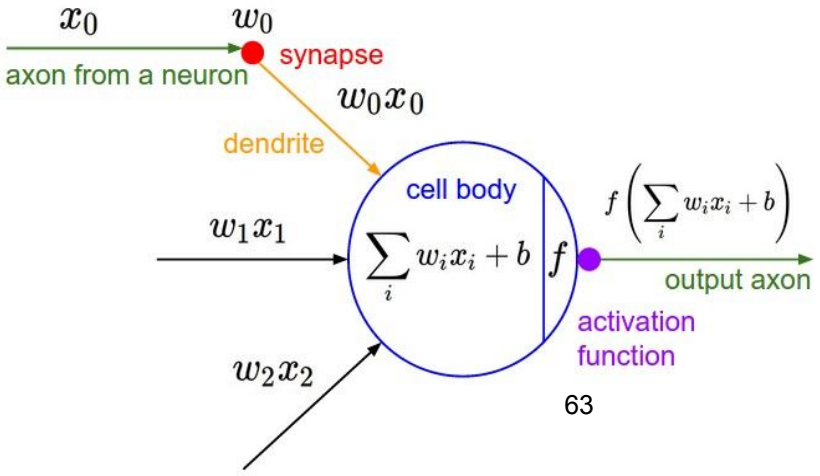
- Neural Networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns
- NN is made up of neurons. It consists of input layer, multiple hidden layers and output layers. The simplest NN is one hidden layer.



What is a Neuron?

- A biological neuron has dendrites to receive signals, a cell body to process them, and an axon to send signals out to other neurons
- An artificial neuron has a number of input channels, a processing stage, and one output that can fan out to multiple other artificial neurons.
- The weighted sum of inputs + bias is feed through a non-linear activation function to output a value to next neurons.



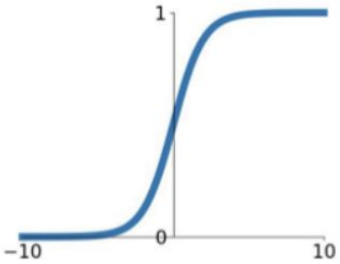


Activation Functions

Activation functions allow Neural Networks to learn complicated and Non-linear complex functional mappings between the inputs and outputs.

Sigmoid

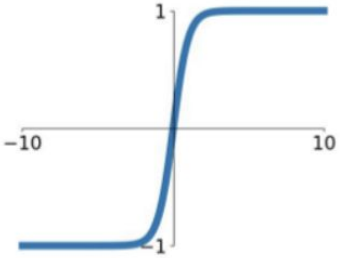
$\sigma(x) = \frac{1}{1+e^{-x}}$



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

tanh

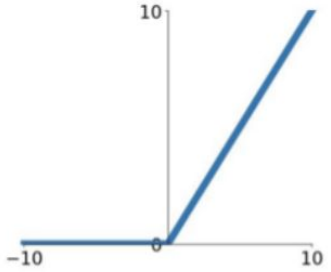
$\tanh(x)$



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

ReLU

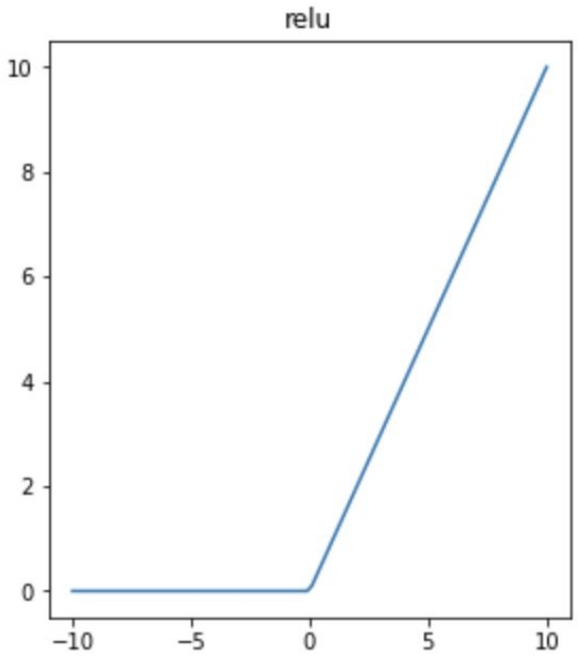
$\max(0, x)$

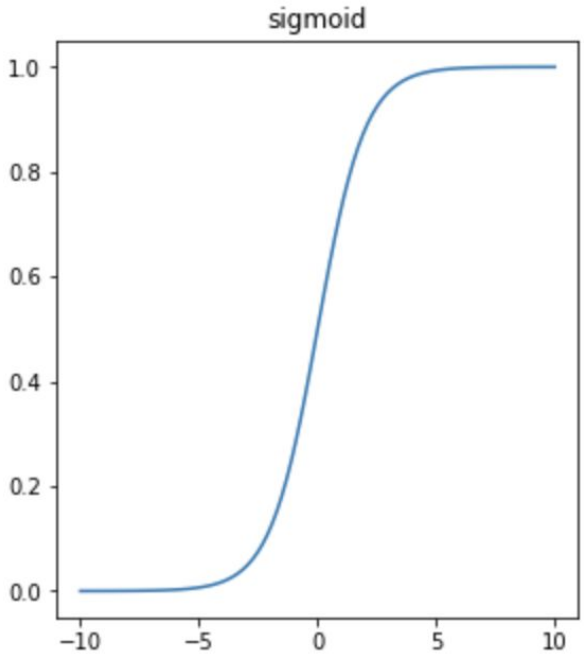


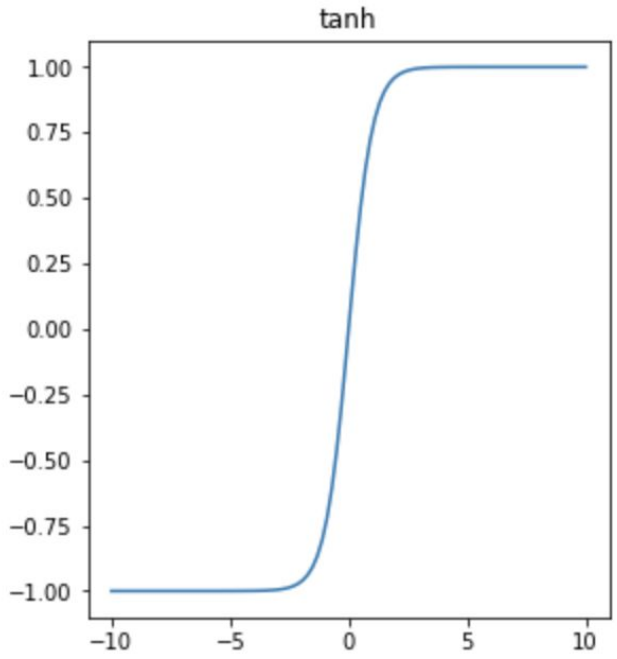
$$\sigma(z) = max(0, z)$$

Activation Functions

torch.relu(x)
torch.sigmoid(x)
torch.tanh(x)

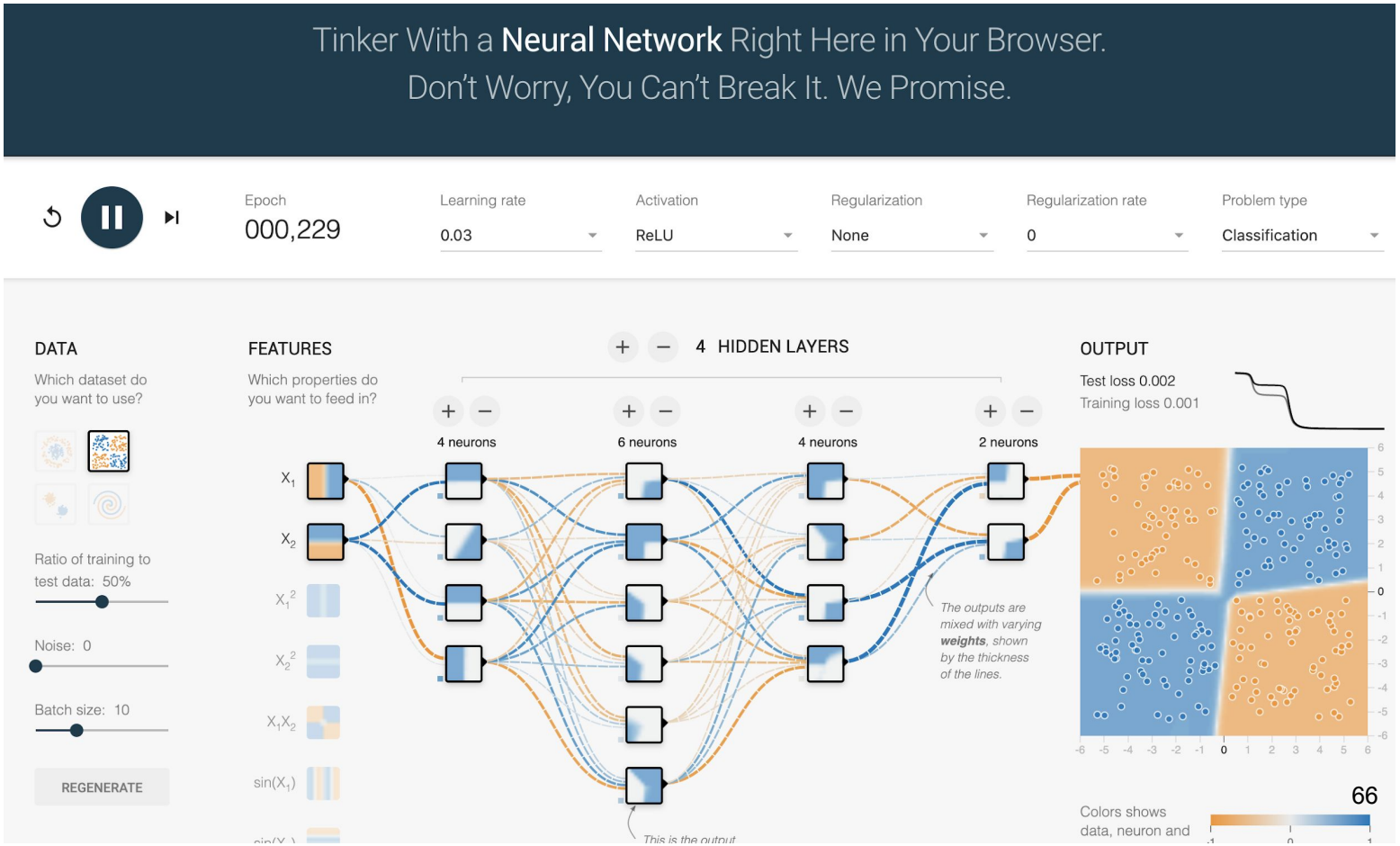




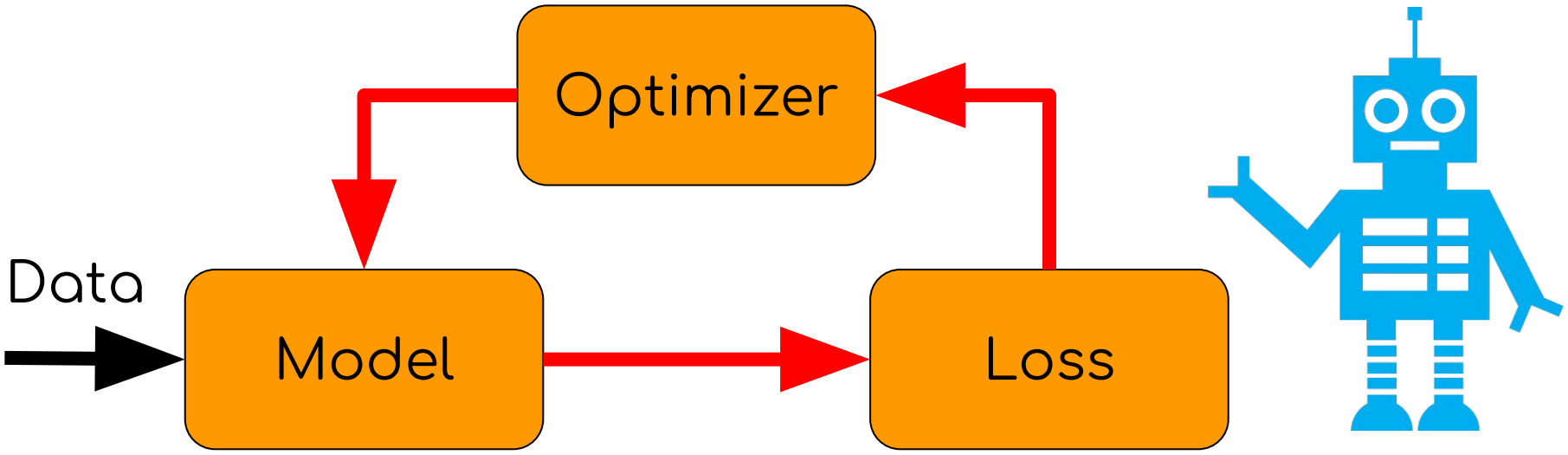


Activity: Tensorflow Playground

- Try different activation functions (sigmoid vs relu) on Tensorflow playground https://playground.tensorflow.org/
- Hover over the connections to check the weights



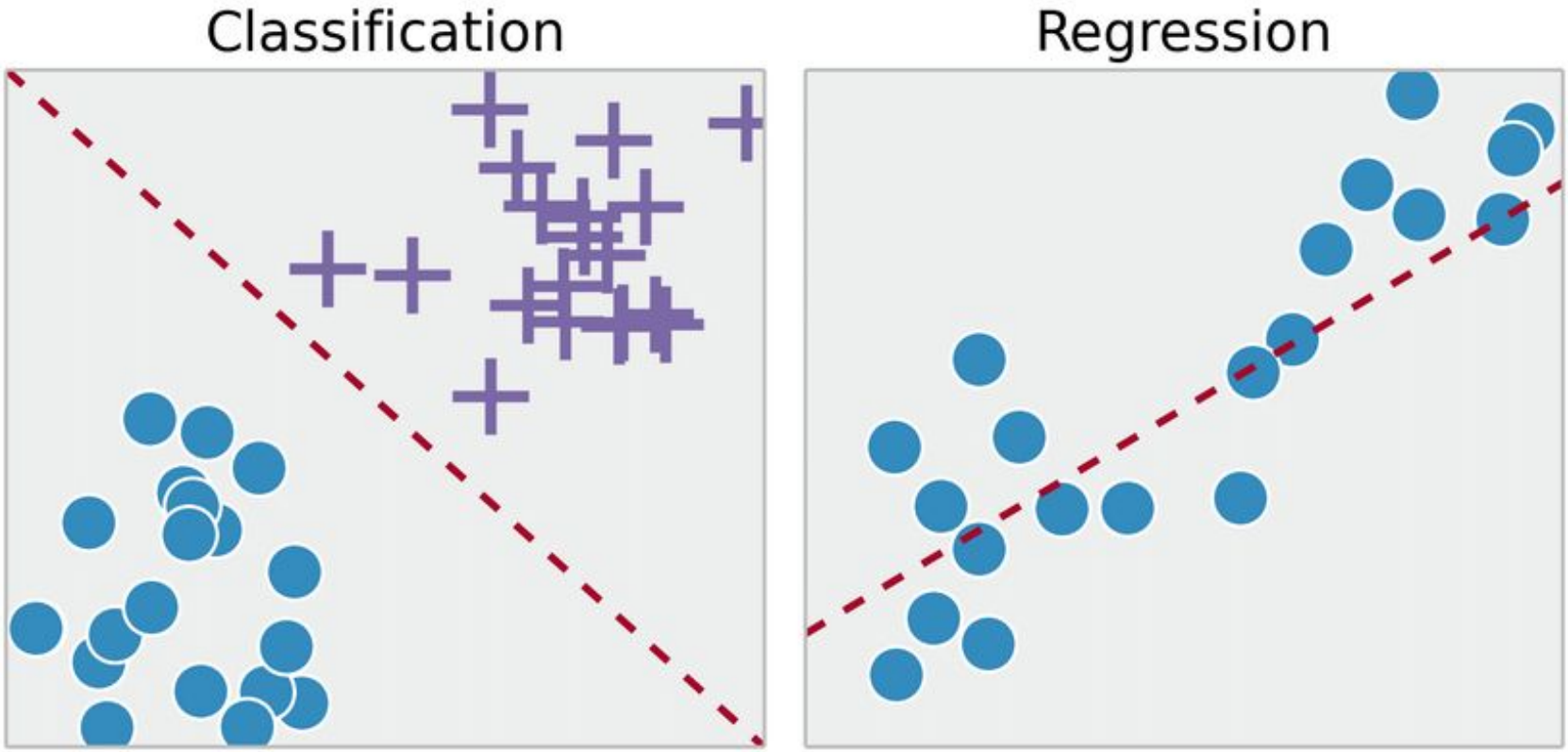
Machine Learning (Trial & Error) Approach



- Deep learning is a trial & error, brute force and blackbox approach to learn the patterns and make predictions
- The approach requires 3 components: **NN Model, Loss Function, and Optimizer**
- The learning (training) is driven by data to reduce the loss in successive iterations.
- The optimizer is to minimize the loss function of a model by adjusting the weights (parameters).

What is Regression?

- Regression it to predicting a continuous-valued attribute associated with an object.
- Regression is a supervised learning method. During training, actual values are provided.



Machine Learning Steps

1. Preprocessing Data
2. Define Model
3. Select Optimizer
4. Define Loss
5. Training Loop

Step 1: Preprocessing Data

```
import torch

X = torch.tensor([1.,2.,3,4,5])
y = torch.tensor([0,-1.1,-1.8,-3.1,-4.5])

W = torch.rand(1,requires_grad=True)
b = torch.rand(1,requires_grad=True)
```

Step 2: Select Optimizer

```
learning_rate = 0.001

optimizer = torch.optim.SGD([W,b], lr=learning_rate)
```

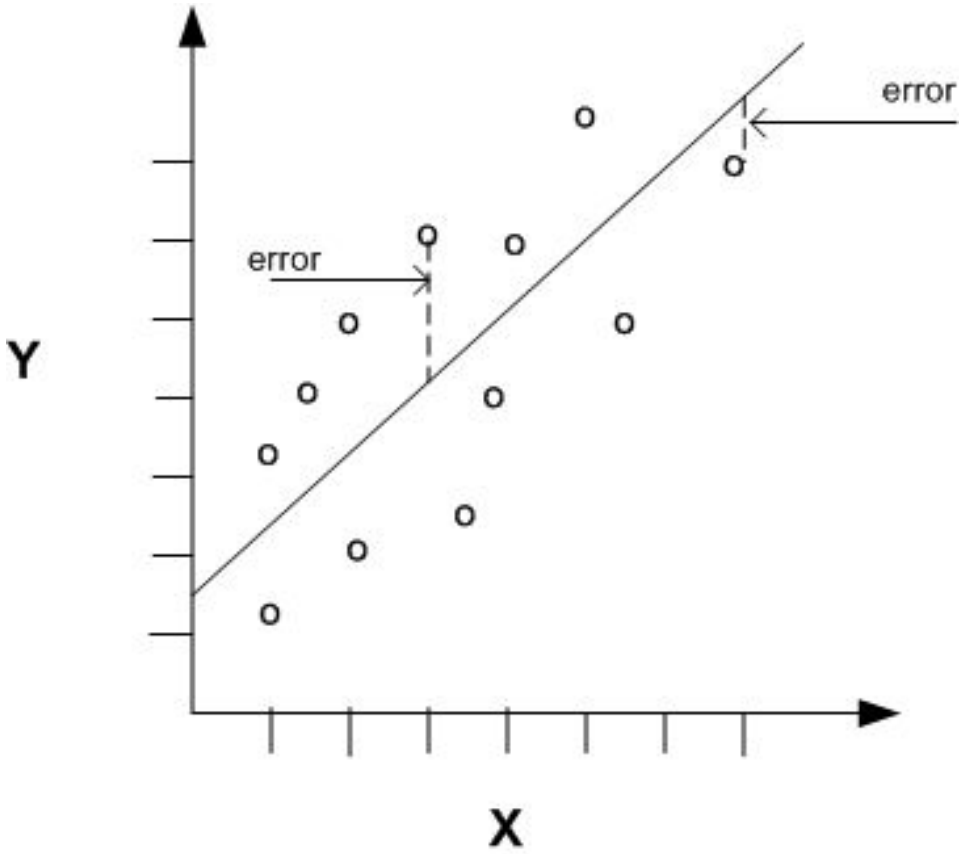
Step 3 to 4 Define Model and Loss

```
yhat = X*W+b # Linear Model

loss = (yhat-y).pow(2).sum()   #Mean Square Error
```

Alternatively, you can use the Pytorch MSELoss function as follows:

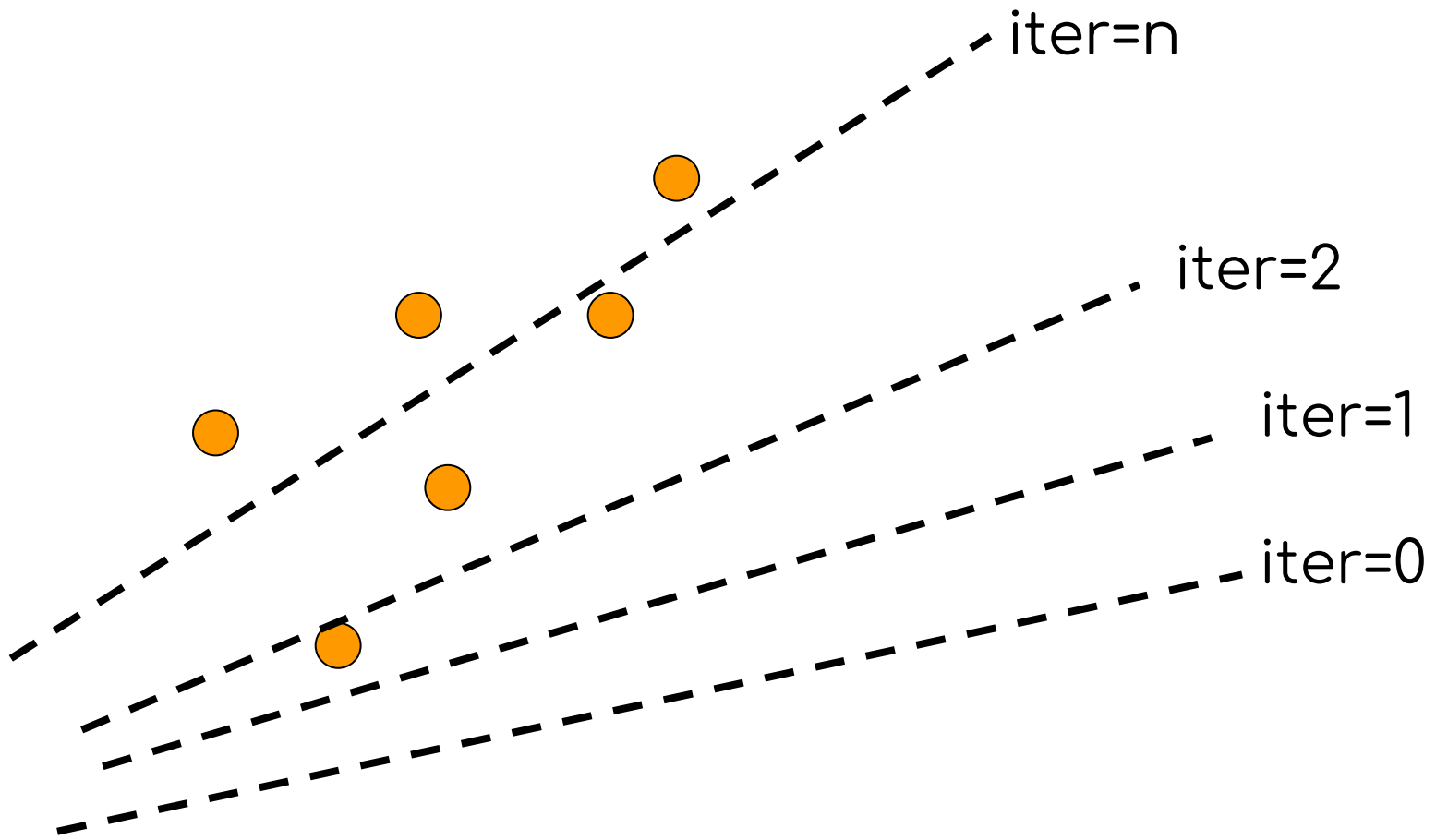
```
criterion = torch.nn.MSELoss()
loss = criterion(yhat,y)
```

Mean Square Error



$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

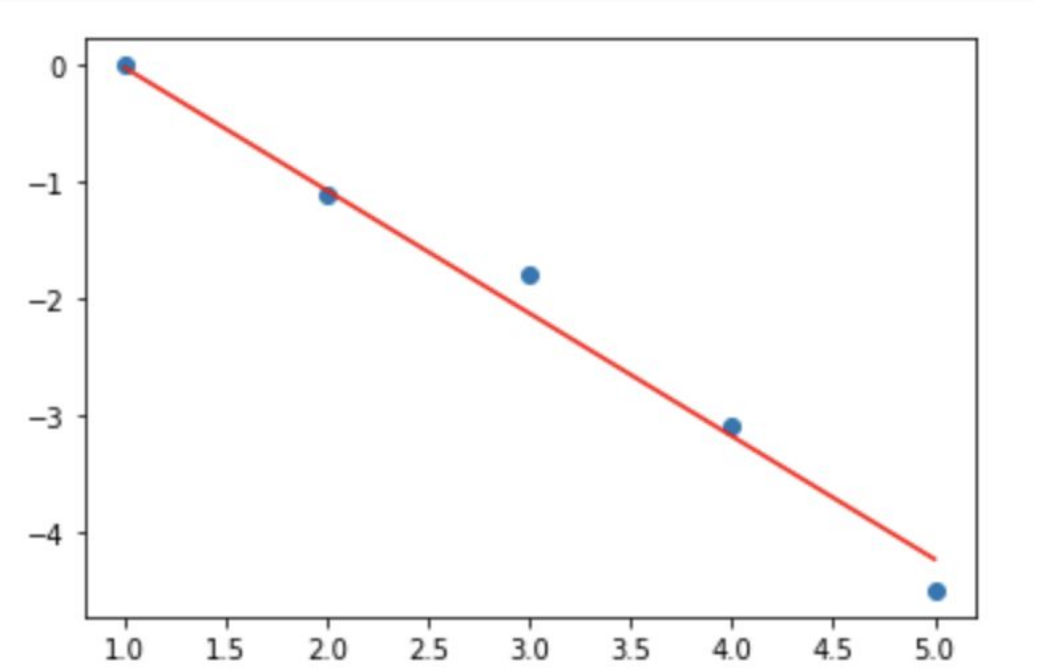
ML Approach to Linear Regression



ML determine the the best fit curve by reducing the MSE to minimum

Step 5: Training Loop

```
for i in range(1000):
    yhat = X*W+b

    loss = (yhat-y).pow(2).sum()

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```



Zero Grad Explanation

- In PyTorch, we need to set the gradients to zero before starting to do backpropagation because PyTorch accumulates the gradients on subsequent backward passes.
- This is convenient while training RNNs.Because of this, when you start your training loop, ideally you should zero out the gradients so that you do the parameter update correctly.
- Else the gradient would point in some other direction than the intended direction towards the minimum

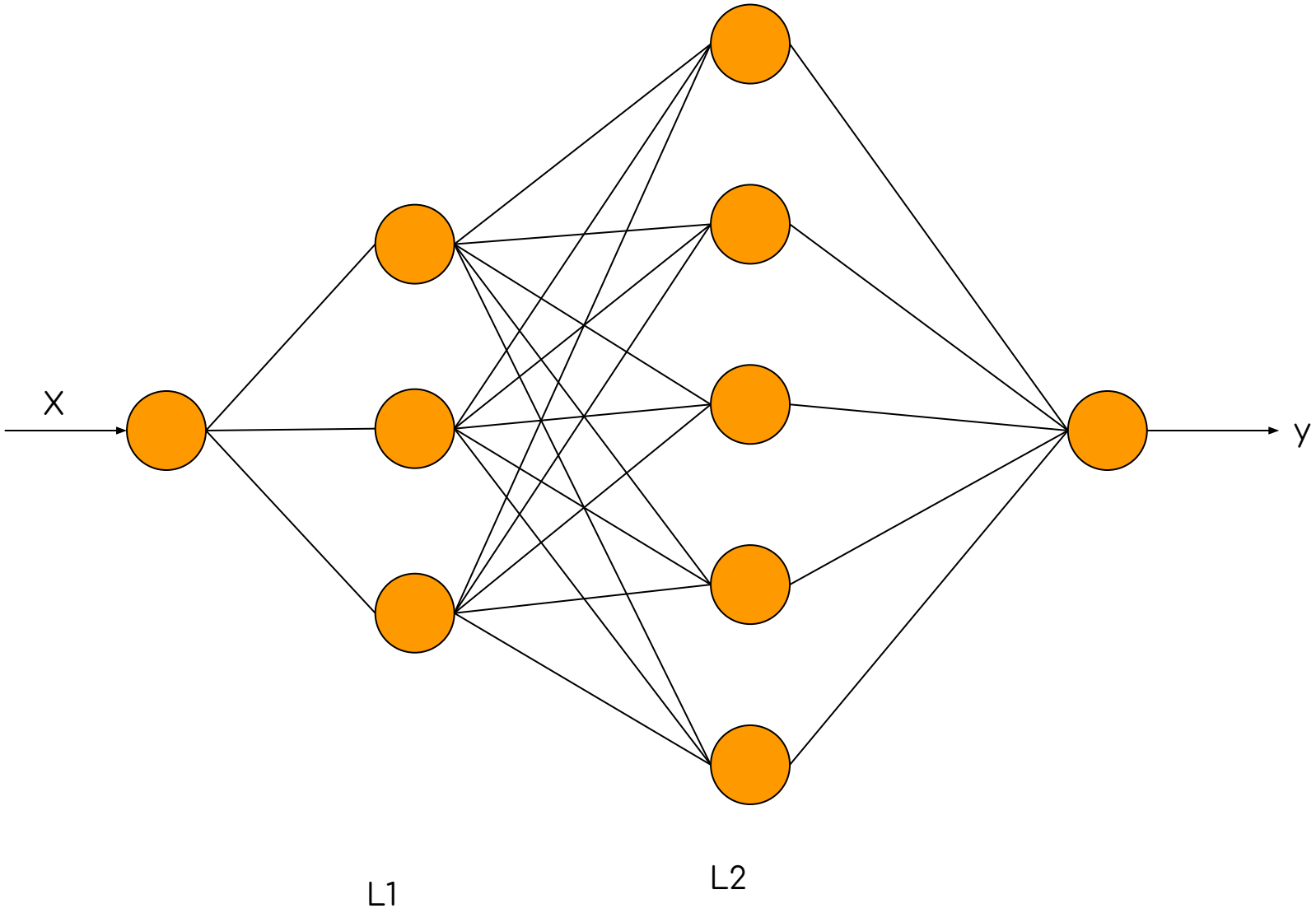
Explicit Zero Grad Demo

```
import torch
learning_rate = 0.001

X = torch.tensor([1.,2.,3,4,5])
y = torch.tensor([0,-1.1,-1.8,-3.1,-4.5])
W = torch.randn(1, requires_grad=True)
b = torch.randn(1, requires_grad=True)

for i in range(1000):
    yhat = X*W+b
    loss = (yhat - y).pow(2).sum()
    loss.backward()

    W.data -= learning_rate * W.grad.item()
    b.data -= learning_rate * b.grad.item()

    W.grad.data.zero_()
    b.grad.data.zero_()

    print(f'i:{i}, W:{W.item()}, b:{b.item()}, loss:{loss.item()}')
)
```

Similar to optimizer.step()

Similar to zero_grad()

Linear Regression NN Model



Step 1: Preprocessing Data (NN)

```
import torch

X = torch.tensor([1.,2.,3,4,5])
y = torch.tensor([0,-1.1,-1.8,-3.1,-4.5])

X = torch.unsqueeze(X, dim=1)
y = torch.unsqueeze(y, dim=1)
```

Step 2 Define Model (NN)

```
import torch
import torch.nn as nn
import torch.nn.functional as F

L1 = 3
L2 = 5
class Model(nn.Module):
    def __init__(self):
        super(Model,self).__init__()
        self.fc1 = nn.Linear(1,L1)
        self.fc2 = nn.Linear(L1,L2)
        self.fc3 = nn.Linear(L2,1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

model = Model()
print(model)
```

Alternate Model Definition

```
# Step 2: Define Model

import torch
import torch.nn as nn
import torch.nn.functional as F

L1 = 3
L2 = 5

model = nn.Sequential(nn.Linear(1, L1),
                      nn.ReLU(),
                      nn.Linear(L1, L2),
                      nn.ReLU(),
                      nn.Linear(L2, 1))

print(model)
```

Step 3 Select Optimizer

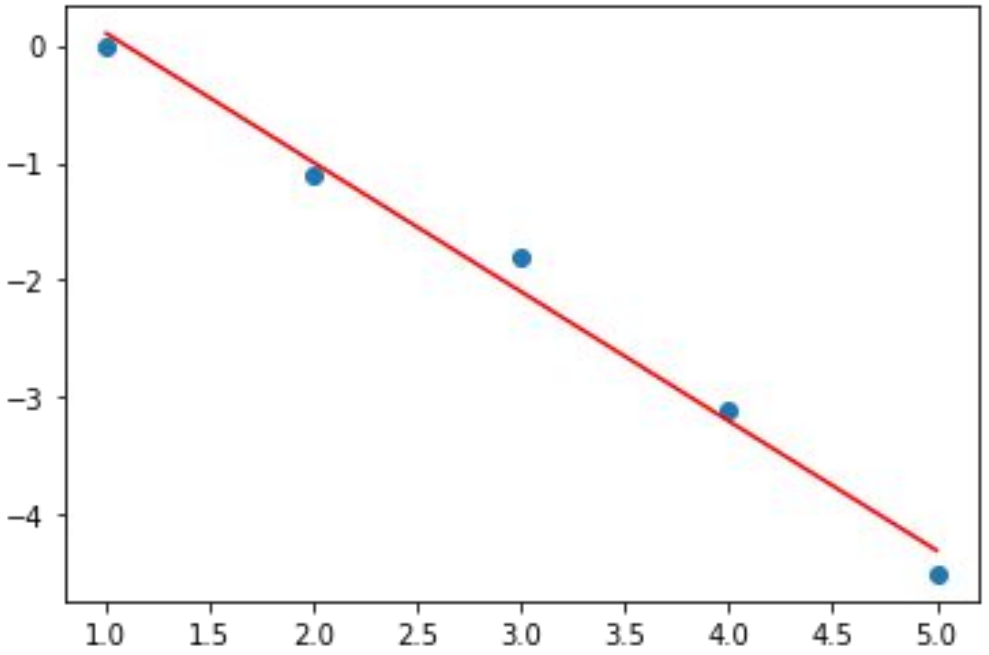
```
learning_rate = 1e-2

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

Step 4 Training

```
for i in range(1000):

    yhat = model(X)

    loss = (yhat - y).pow(2).sum()

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Step 5: Evaluate Model

```
import matplotlib.pyplot as plt

plt.plot(X,y,'o')
plt.plot(X,model(X).data,'r')
plt.show()
```



Demo: Predictive Model for Housing Price

This is a demo of how to build a predictive regression model for housing price. The steps are as follows:

- **Prepare the data**
  - Import raw data
  - Clean data eg remove missing data
  - Normalize the data
  - Split raw data to train and test dataset
  - Create input and output
- **Build the model**
  - Create a neural network model
  - Compile the model (optimizer, loss function)
  - Inspect the model
- **Train the model**
- **Evaluate the model**
- **Make prediction**

Import the Data

We will get some raw data from github Eg boston hosting price

import pandas as pd

dataset_path = "https://raw.githubusercontent.com/tertiarycourses/datasets/master/boston.csv"

Prepare the Data for Training

- **Split the data to train and test dataset**

```
x_train = dataset.sample(frac=0.7,random_state=0)
x_test = dataset.drop(x_train.index)
```

- **Create input and output**

```
y_train = x_train.pop('medv')
y_test = x_test.pop('medv')
```

Clean the Data

- **Remove missing data**
  dataset = dataset.dropna()

- **Normalize the data**
  x_train = (x_train - x_train.mean())/x_train.std()
  x_test = (x_test - x_test.mean())/x_test.std()

Convert Data to Tensors

```
import torch

x_train = torch.Tensor(x_train.values)
y_train = torch.Tensor(y_train.values)

x_test = torch.Tensor(x_test.values)
y_test = torch.Tensor(y_test.values)
```

Increase Tensor Dimension

```
y_train = torch.unsqueeze(y_train, dim=1)
y_test = torch.unsqueeze(y_test, dim=1)
```

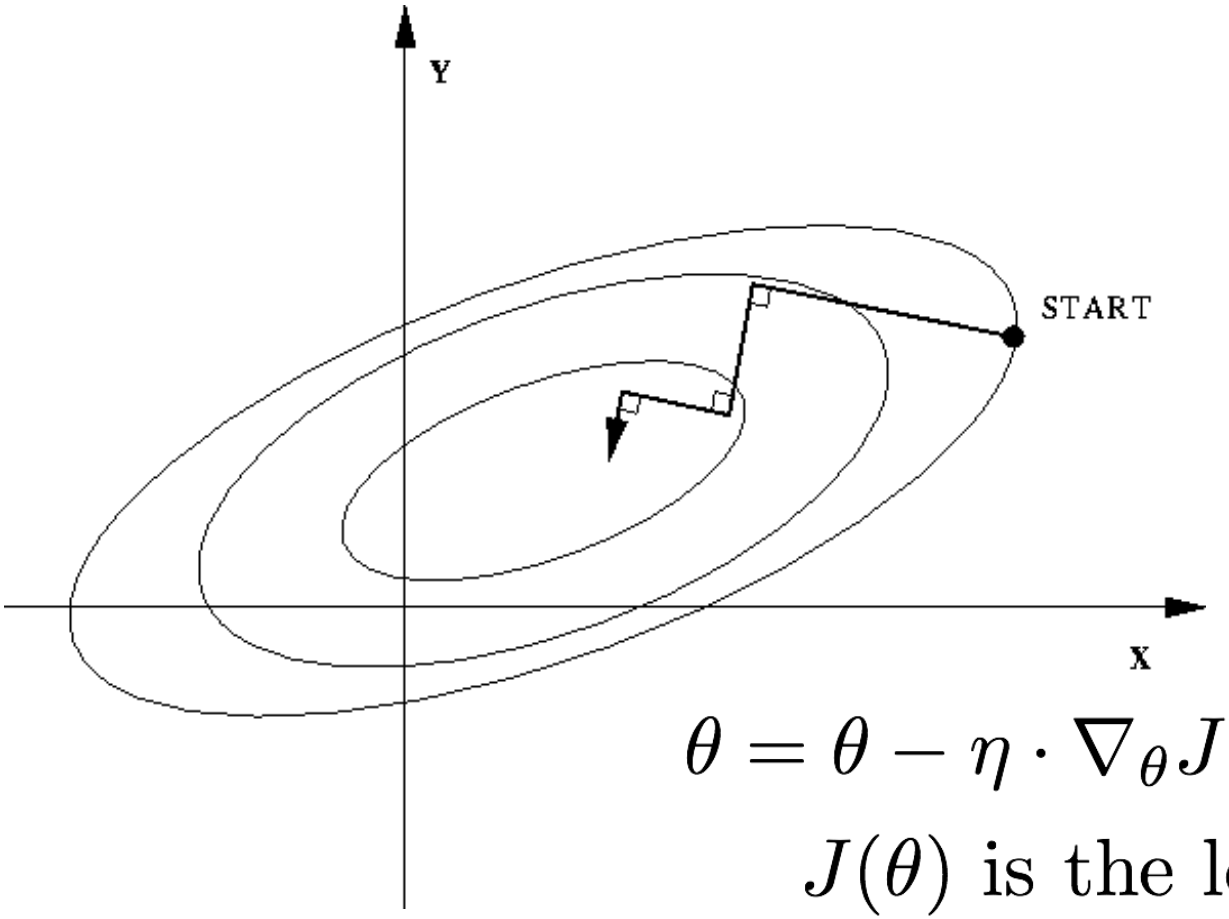
Build the Model

```
class Model(nn.Module):

    def __init__(self):
        super(Model,self).__init__()
        self.fc1 = nn.Linear(13,L1)
        self.fc2 = nn.Linear(L1,L2)
        self.fc3 = nn.Linear(L2,1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

model = Model()
```

Define the Loss and Optimizer

```
criterion = nn.MSELoss()

learning_rate = 0.001
optimizer = torch.optim.SGD(model.parameters(),
lr=learning_rate)
```

GradientDescentOptimizer (SGD)



$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

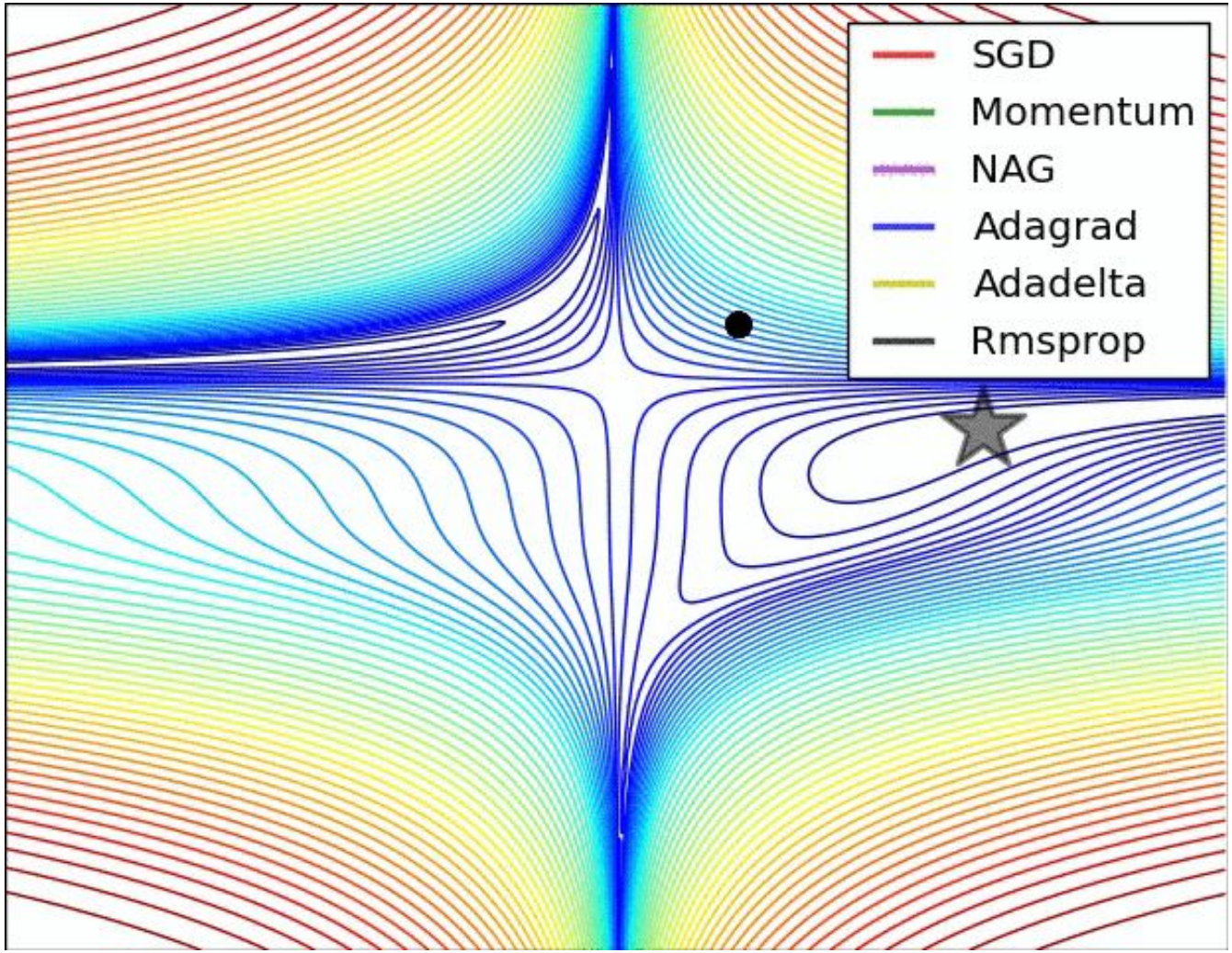
$J(\theta)$ is the loss function

η is the learning rate

θ are the model parameters

Other Popular Optimizers

- Adam
- RMSProp
- Adagrad
- Adadelta
- Momentum
- NAG



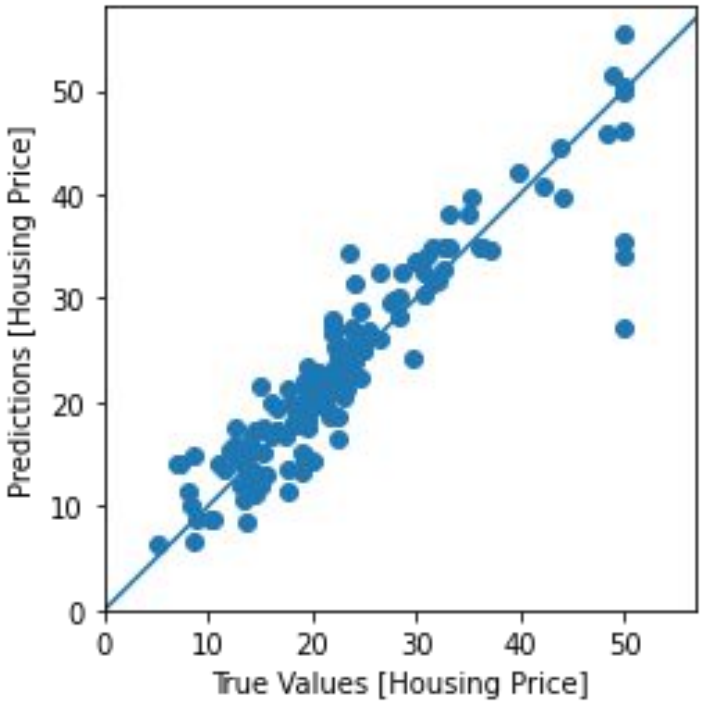
Train the Model

```
for i in range(1000):

    # Model prediction
    yhat = model(x_train)

    # Compute loss
    loss = criterion(yhat, y_train)

    # Compute gradients and update parameters
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

we need to manually set the gradients to zero using optimizer.zero_grad() since PyTorch by default accumulates gradients.

Evaluate the Model

```
import matplotlib.pyplot as plt

plt.scatter(y_test, model(x_test).data)
plt.xlabel('True Values [Housing Price]')
plt.ylabel('Predictions [Housing Price]')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
plt.ylim([0,plt.ylim()[1]])
plt.plot([0, 100], [0, 100])
```



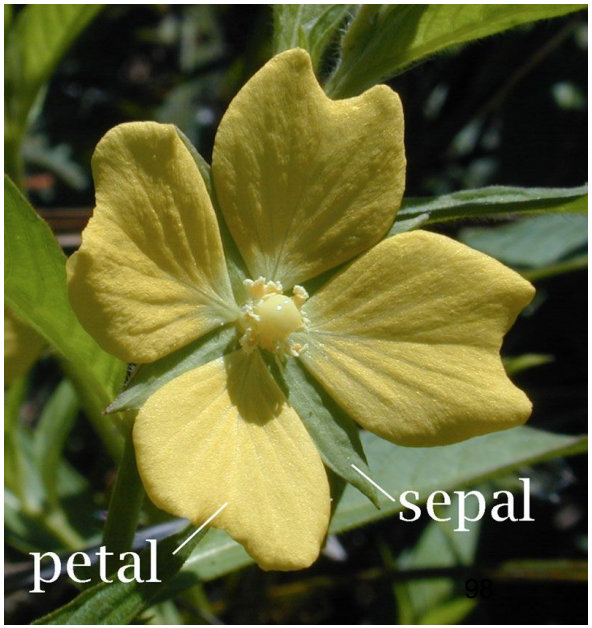
Save and Load the Model

```
# Save the model
torch.save(model,'./regression.pkl')

# Load the model
new_model=torch.load('./regression.pkl')
```

Activity: Predictive Regression Model

- Import the iris.dataset from github https://raw.githubusercontent.com/tertiarycourses/datasets/master/iris.csv
- Create a NN model is with 3 hidden layers (64,32,32) to predict the sepal width from sepal length, petal length and petal width
- Perform all the steps as shown in the demo
  - Prepare the data
  - Build the model
  - Train the model
  - Make prediction
  - Save the model

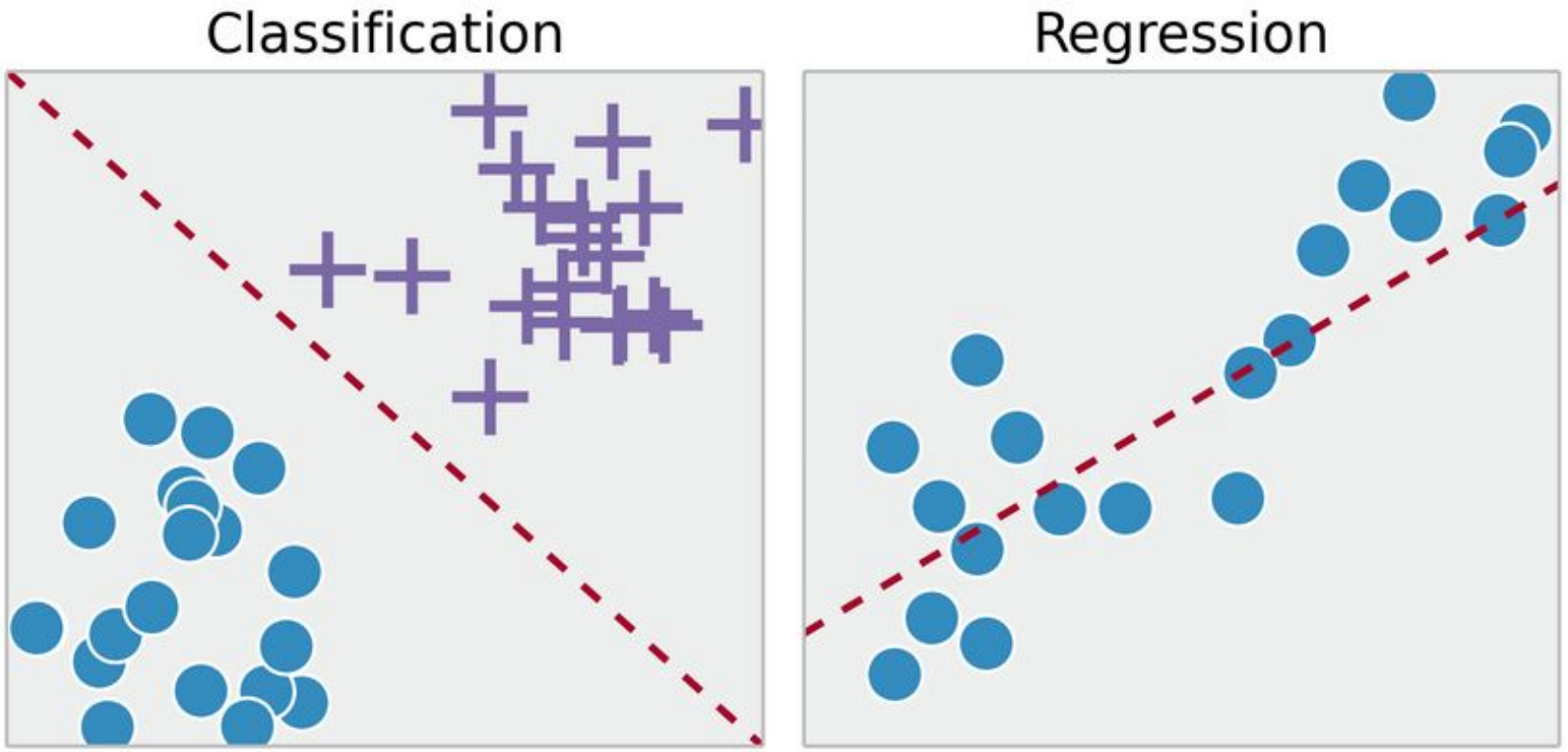
Time: 30 mins



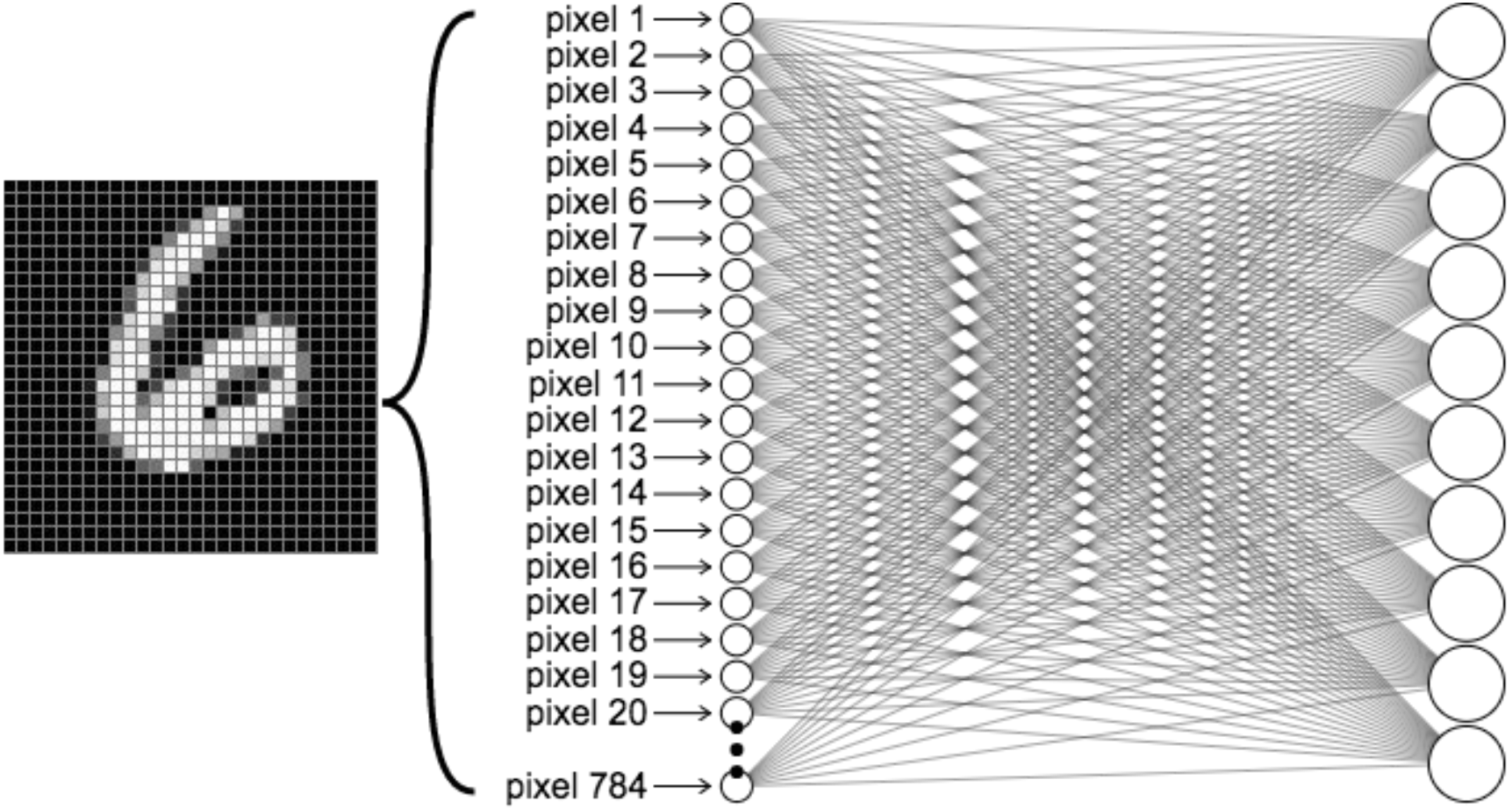
Topic 3
Neural Network for Classification

What is Classification?

- Classification is to identifying to which category an object belongs to.
- Classification is a supervised learning method. During training, labels/classes are provided.

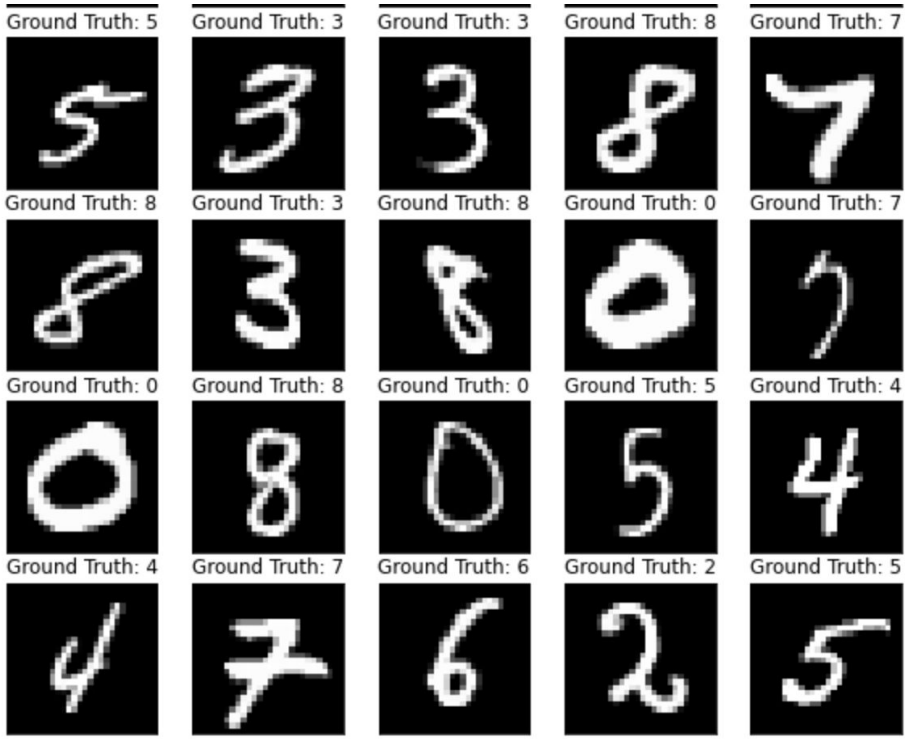


NN for HandWritten Digit Classification



MNIST Handwritten Digits

- The MNIST data is split into three parts:
  - 55,000 training data
  - 10,000 test data
  - 5,000 validation data
- Each image is 28 pixels by 28 pixels and serialize to a 1D vector of 784 numbers
- The label is one hot encoded.



Download MNIST Datasets

```
mean, std = (0.5,), (0.5,)

# Create a transform and normalise data
# Download MNIST test dataset and load test data
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize(mean, std)
                               ])
trainset = datasets.MNIST('~/.pytorch/MNIST/', download=True, train=True,
transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)


# Download MNIST test dataset and load test data
testset = datasets.MNIST('~/.pytorch/MNIST/', download=True, train=False,
transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False)
```

Use datasets to:
Download dataset
Transform data

Use DataLoader function for Mini-batch Training

One Hot Encoding for Digits

0	[1 0 0 0 0 0 0 0 0 0]
1	[0 1 0 0 0 0 0 0 0 0]
2	[0 0 1 0 0 0 0 0 0 0]
3	[0 0 0 1 0 0 0 0 0 0]
4	[0 0 0 0 1 0 0 0 0 0]
5	[0 0 0 0 0 1 0 0 0 0]
6	[0 0 0 0 0 0 1 0 0 0]
7	[0 0 0 0 0 0 0 1 0 0]
8	[0 0 0 0 0 0 0 0 1 0]
9	[0 0 0 0 0 0 0 0 0 1]

Probabilistics Output

- Classification neural network model always gives a probabilistic output, for example

Probiliity

0.8

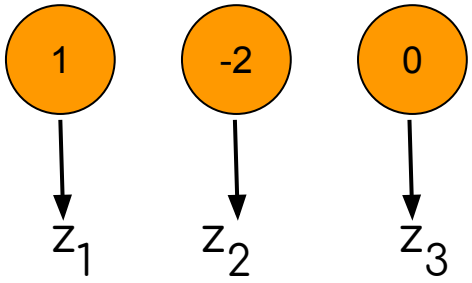
0.15

0.05

Softmax

- Softmax extends logistics regression (produce 0 to 1 for a single output) into a multi-class problem.
- Softmax assigns probabilities to each class in a multi-class problem using the formula below

$$p_j = \sigma(z)_j = \frac{e^{z_j}}{\sum_{i=1}^{k} e^{z_i}}$$



$$[1, -2, 0] \rightarrow e^1, e^{-2}, e^0 = [2.71, 0.14, 1] \rightarrow 0.7, 0.04, 0.26$$

Define the NN Model

```
L1 = 32
L2 = 64

class Model(nn.Module):
    def __init__(self):
        super(Model,self).__init__()
        self.fc1 = nn.Linear(784,L1)
        self.fc2 = nn.Linear(L1,L2)
        self.fc3 = nn.Linear(L2,10)

    def forward(self, x):
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

model = Model()

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

Alternative NN Model Definition

```
import torch
import torch.nn as nn
import torch.nn.functional as F

L1 = 32
L2 = 64
model = nn.Sequential(nn.Flatten(),
                      nn.Linear(784, L1),
                      nn.ReLU(),
                      nn.Linear(L1, L2),
                      nn.ReLU(),
                      nn.Linear(L2, 10))
```

Add Model to GPU

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

Loss Function and Optimizer

```
# Loss Function
criterion = nn.CrossEntropyLoss()

# Optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

Cross Entropy

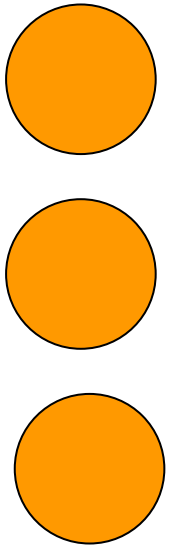
Cross entropy is use to measure the closeness of two probability distribution

$$H_{y'}(y) = -\sum_i y'_i \log(y_i)$$

Example of Cross Entropy

Cross entropy = 1*log(0.6) + 0*log(0.3) + 0*log(0.1) = 0.51

Compute Cross Entropy Example



Softmax Prediction	Case 1	Case 2
0.73	0	1
0.02	0	0
0.25	1	0

Loss for Case 1 = -log(0.25) = 1.46
Loss for Case 2 = -log(0.73) = 0.31

Activity: Cross Entropy

Compute the Cross Entropy for the following 2 cases

Correct Label (y)	Predicted Softmax (yhat) case 1	Predicted Softmax (yhat) case 2
0	0.8	0.1
1	0.1	0.8
0	0.1	0.1

Ans: Case 1 = 1, Case 2 = 0.1

Cross Entropy Loss

torch.nn.CrossEntropyLoss()

- Combines LogSoftMax and NLLLoss
- useful when training a classification problem with n classes
- The input is expected to contain scores for each class
- input has to be a 2D Tensor of size (minibatch,n).

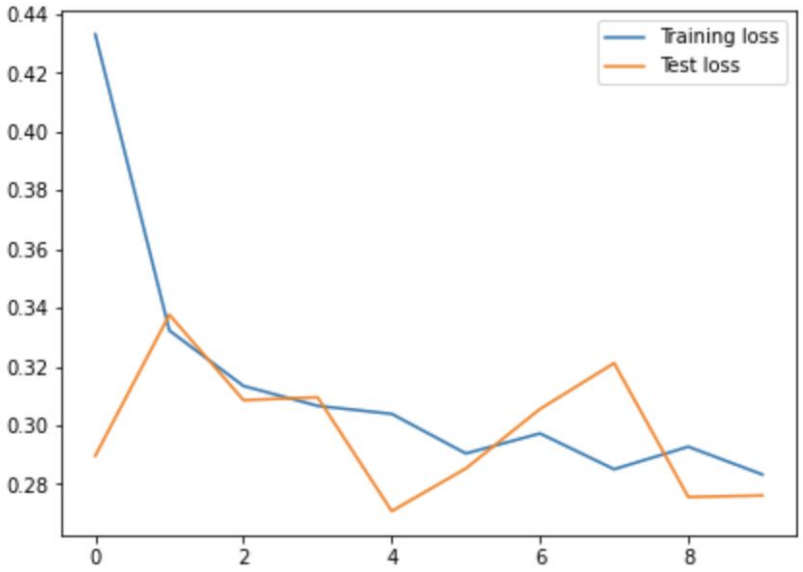
Train the Model

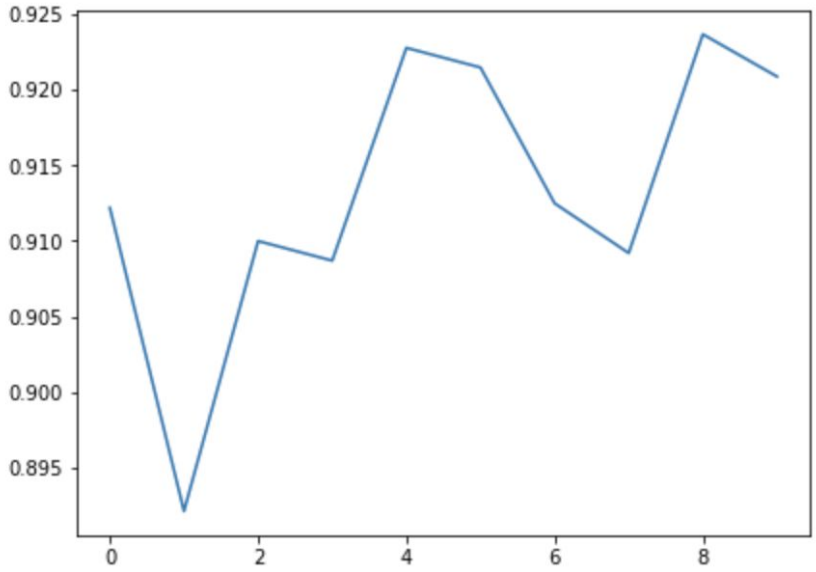
```
for i in range(num_epochs):
    train_loss = 0

    for batch, (X, y) in enumerate(trainloader):
        X = X.to(device)
        y = y.to(device)

         yhat = model(X)

        optimizer.zero_grad()
        loss = criterion(yhat, y)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    train_tracker.append(train_loss/len(trainloader))
    print(f"Epoch({i+1}/{num_epochs}) | Training loss: {train_loss/len(trainloader)} | ",end='')
```

```
Epoch(1/10) | Training loss: 0.493915855884552 | Test loss: 0.36642441153526306 | Accuracy : 0.8849999904632568
Epoch(2/10) | Training loss: 0.3544181287288666 | Test loss: 0.27184629440307617 | Accuracy : 0.9181999564170837
Epoch(3/10) | Training loss: 0.3391174077987671 | Test loss: 0.38928401470184326 | Accuracy : 0.8816999793052673
Epoch(4/10) | Training loss: 0.3254077732563019 | Test loss: 0.2915478050708771 | Accuracy : 0.9121999740600586
Epoch(5/10) | Training loss: 0.31116983294487 | Test loss: 0.29876407980918884 | Accuracy : 0.911300003528595
Epoch(6/10) | Training loss: 0.32121512293815613 | Test loss: 0.3007839322090149 | Accuracy : 0.9145999550819397
Epoch(7/10) | Training loss: 0.3109281361103058 | Test loss: 0.2833910584449768 | Accuracy : 0.9146999716758728
Epoch(8/10) | Training loss: 0.3096688389778137 | Test loss: 0.33763203024864197 | Accuracy : 0.8985999822616577
Epoch(9/10) | Training loss: 0.3109353184700012 | Test loss: 0.39147722721099854 | Accuracy : 0.8889999985694885
Epoch(10/10) | Training loss: 0.31191393733024597 | Test loss: 0.31013721227645874 | Accuracy : 0.910099983215332

Number correct : 9101, Total : 10000
Accuracy of the model after 30 epochs on the 10000 test images: 91.00999450683594%
```

Evaluate the Model

```
import matplotlib.pyplot as plt

plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(train_tracker, label='Training loss')
plt.plot(test_tracker, label='Test loss')
plt.legend()

plt.subplot(1,2,2)
plt.plot(accuracy_tracker, label='Test accuracy')
plt.show()
```





Fashion MNIST

- Similar to Handwritten Digits, the fashion MNIST data is split into three parts:
  - 55,000 training data
  - 10,000 test data
  - 5,000 validation data
- Each image is 28 pixels by 28 pixels and serialize to a 1D vector of 784 numbers
- The label is one hot encoded.

Fashion MNIST Classes

T-shirt/top	[1 0 0 0 0 0 0 0 0 0]
Trouser	[0 1 0 0 0 0 0 0 0 0]
Pullover	[0 0 1 0 0 0 0 0 0 0]
Dress	[0 0 0 1 0 0 0 0 0 0]
Coat	[0 0 0 0 1 0 0 0 0 0]
Sandal	[0 0 0 0 0 1 0 0 0 0]
Shirt	[0 0 0 0 0 0 1 0 0 0]
Sneaker	[0 0 0 0 0 0 0 1 0 0]
Bag	[0 0 0 0 0 0 0 0 1 0]
Ankle boot	[0 0 0 0 0 0 0 0 0 1]

Activity: Fashion MNIST Classification

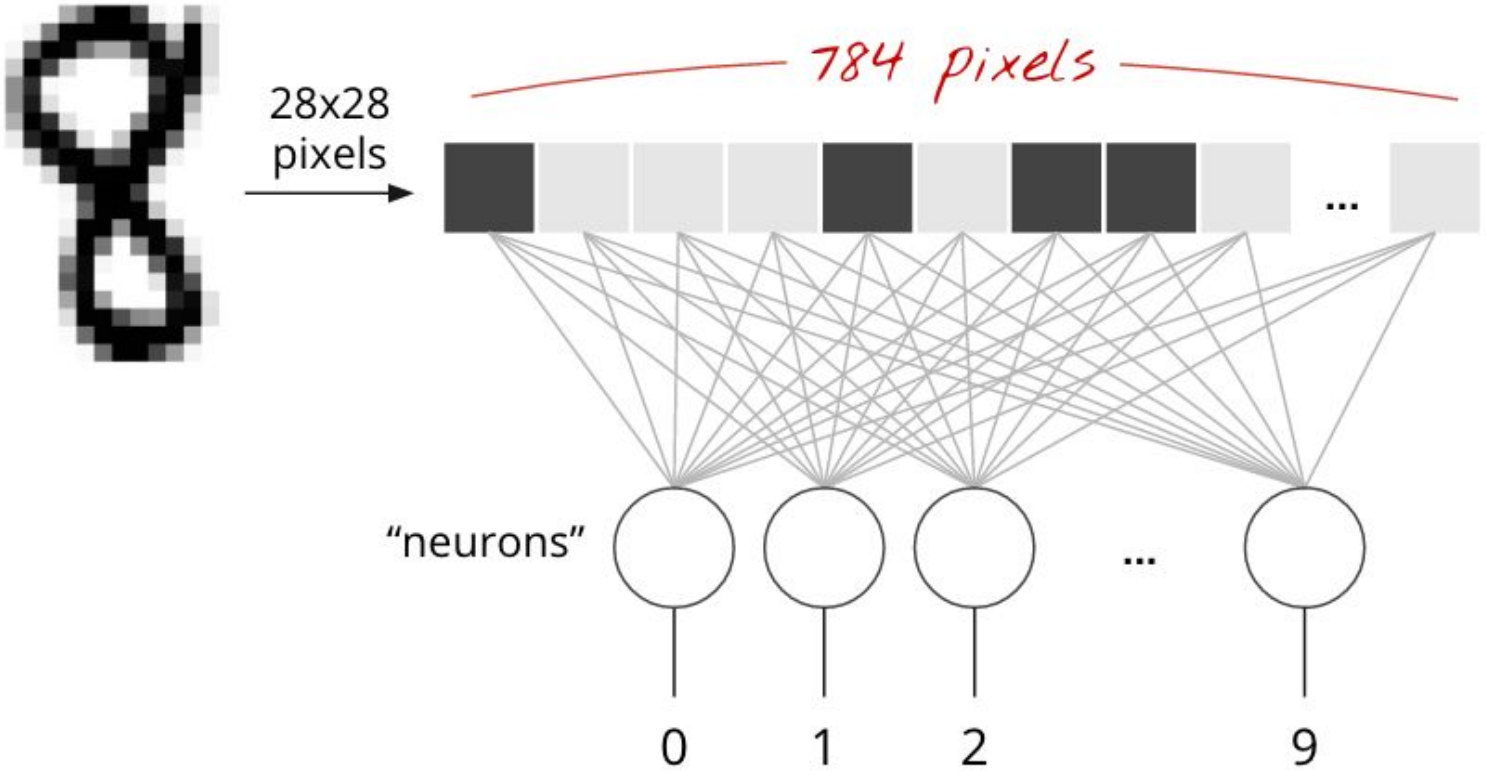
- Import the Fashion MNIST data
- Create a NN model is with 3 hidden layers (128,64,32) to classify the fashion
- Perform all the steps as shown in the demo
  - Prepare the data
  - Build the model
  - Train the model
  - Evaluate the model

Time: 30 mins

Topic 4
Convolutional Neural Network (CNN)

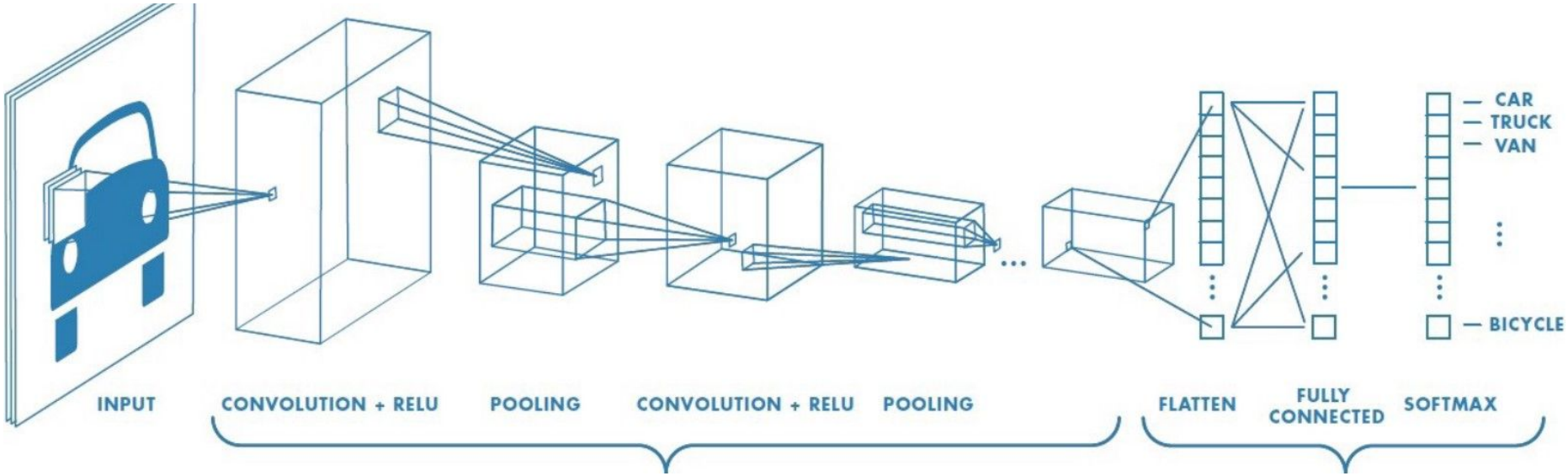
Limitation of DNN

When we vectorize a 28x28 mnist image to a 784x1 vector, we lose the spatial information of the image, and thus reduce the accuracy of the prediction



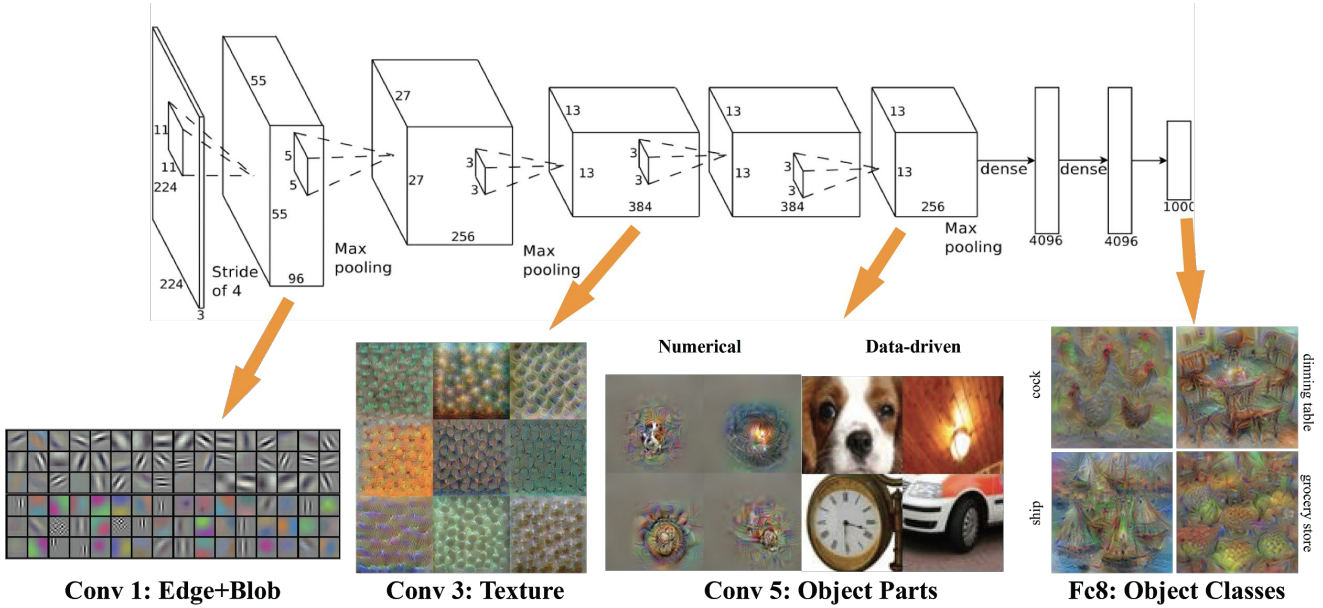
Convolutional Neural Network (CNN)

- The key technology behind computer vision is convolutional neural network (CNN)
- Convolution networks are generally composed of Convolution layers, pooling layers and a last component which is the fully connected that will be used for an appropriate task like classification or detection.



Why CNN?

- CNN preserves the spatial relationship between pixels by learning localized image features using small filters (kernels).
- The first conv layer extracts general features like edges, and subsequent conv layers extract more detailed features



AlexNet
8 layers:
5 conv layers +
3 FC layers

Brief History of CNN

- 1959 Hubel & Wiesel found two types of cells in the visual primary cortex, and proposed a cascading model of these two types of cells for use in pattern recognition task
- 1980 - Inspired by Hubel and Wiesel work, Kunihiko Fukushima introduced neocognition neutral network, similar to CNN concept (not using BP yet)
- 1982-1985 Geoff Hinton, Yann LeCun and others successfully used BP on NN with sigmoid activation function.
- 1990 - Yann LeCun invented CNN with BP for handwritten digits recognition
- 2012 - Alex Krizhevsky won the ImageNet LSVRC competition with AlexNet (deep CNN +ReLU+Data Augmentation+Dropout+GPU)

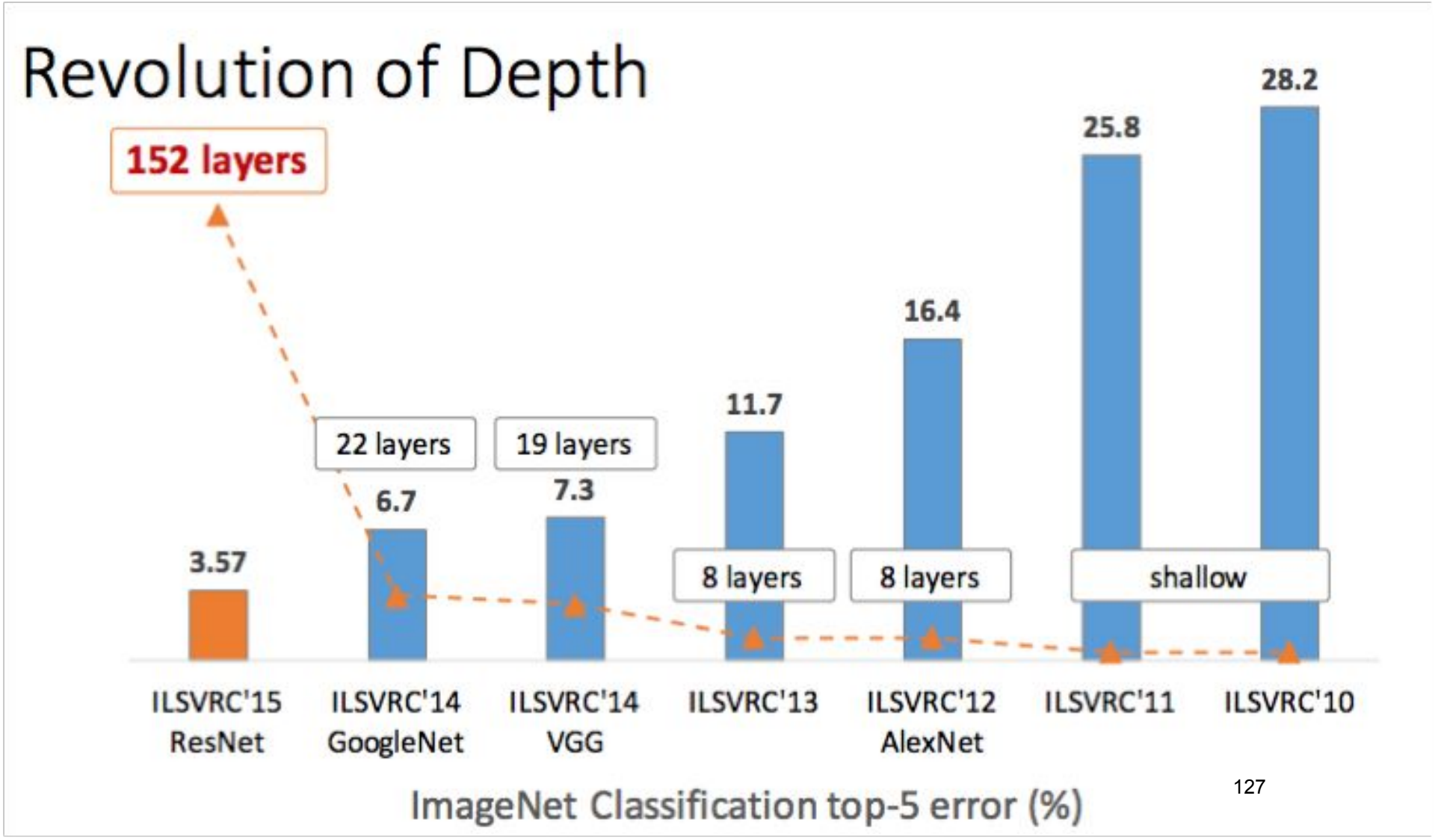
Inventor of CNN

- CNN was invented by Yann LeCun, currently the Director of AI Research, Facebook (http://yann.lecun.com/)
- LeNet-5 is the latest convolutional network designed for handwritten and machine-printed character recognition (http://yann.lecun.com/exdb/lenet/index.html)

ImageNET ILSVRC

- ImageNet (image-net.org) was initiated by Li Fei Fei in 2007 with more than 15 Million images
- Held ILSVRC (International Large Scale Visual Recognition Challenge) since 2010

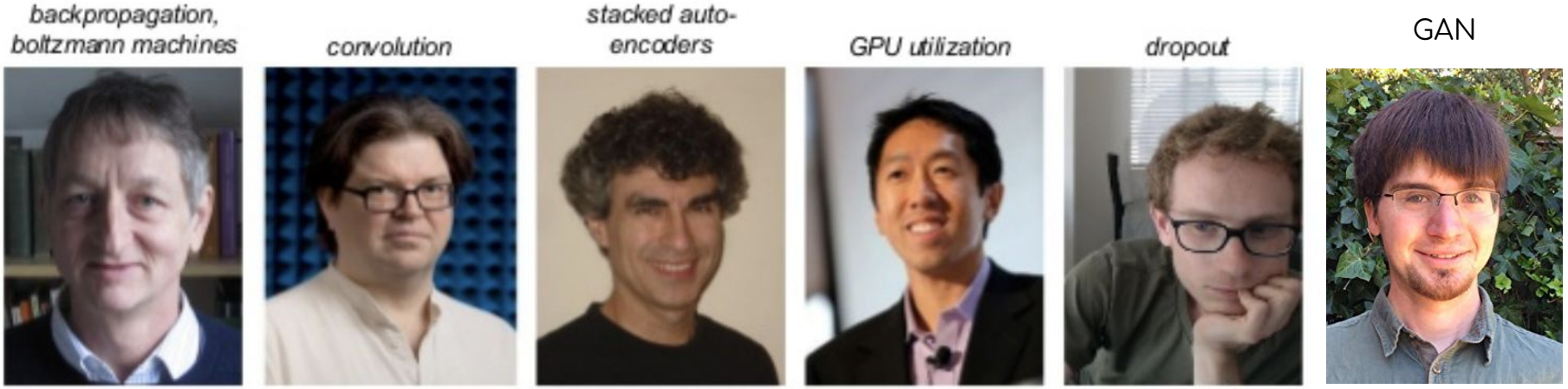
ILSVRC Winners



Heroes of Machine Learning

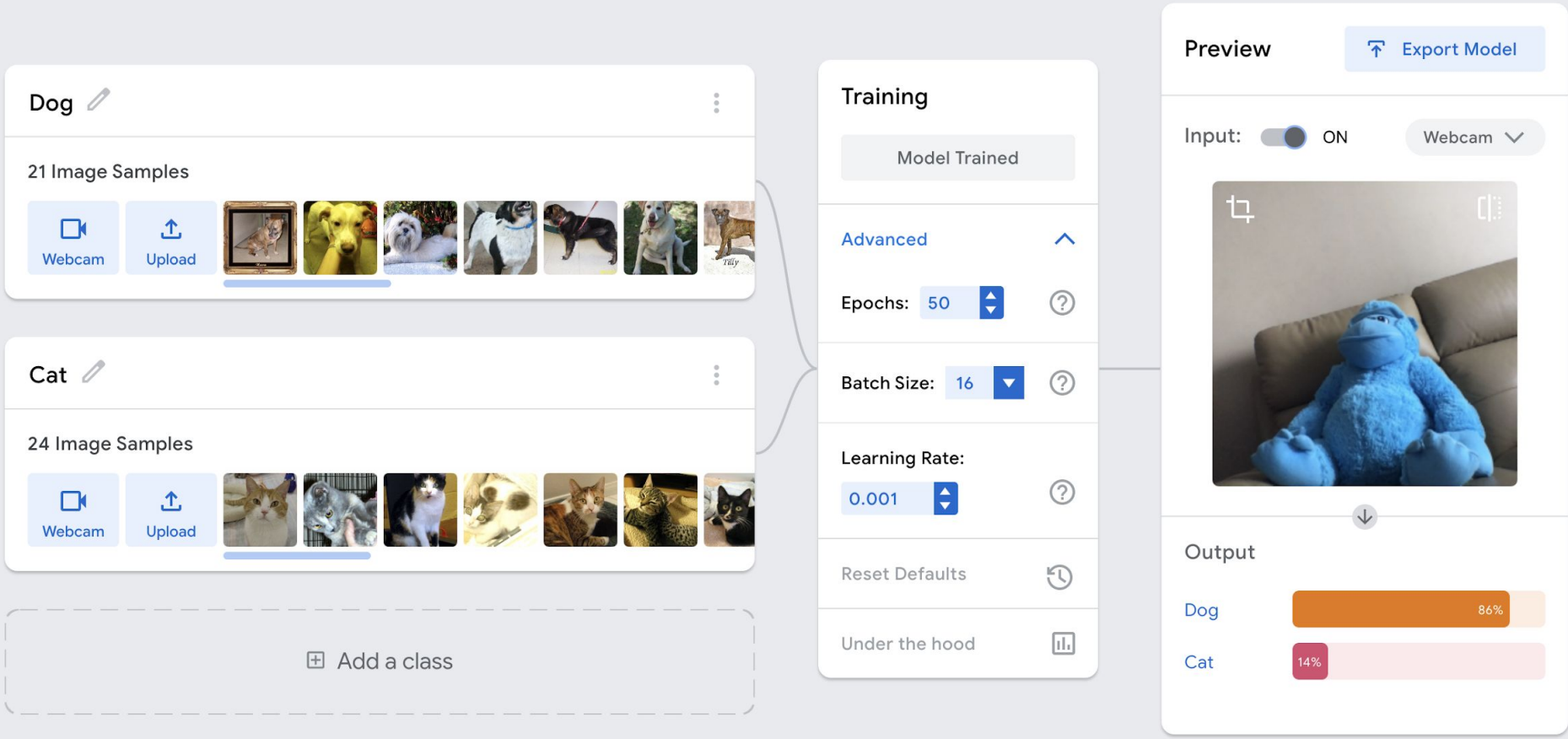
There are many heroes in Machine Learning. The Turing Award 2019 is award to

- Geoff Hinton
- Yann Lecun
- Yoshua Bengio



Activity : Image Classification

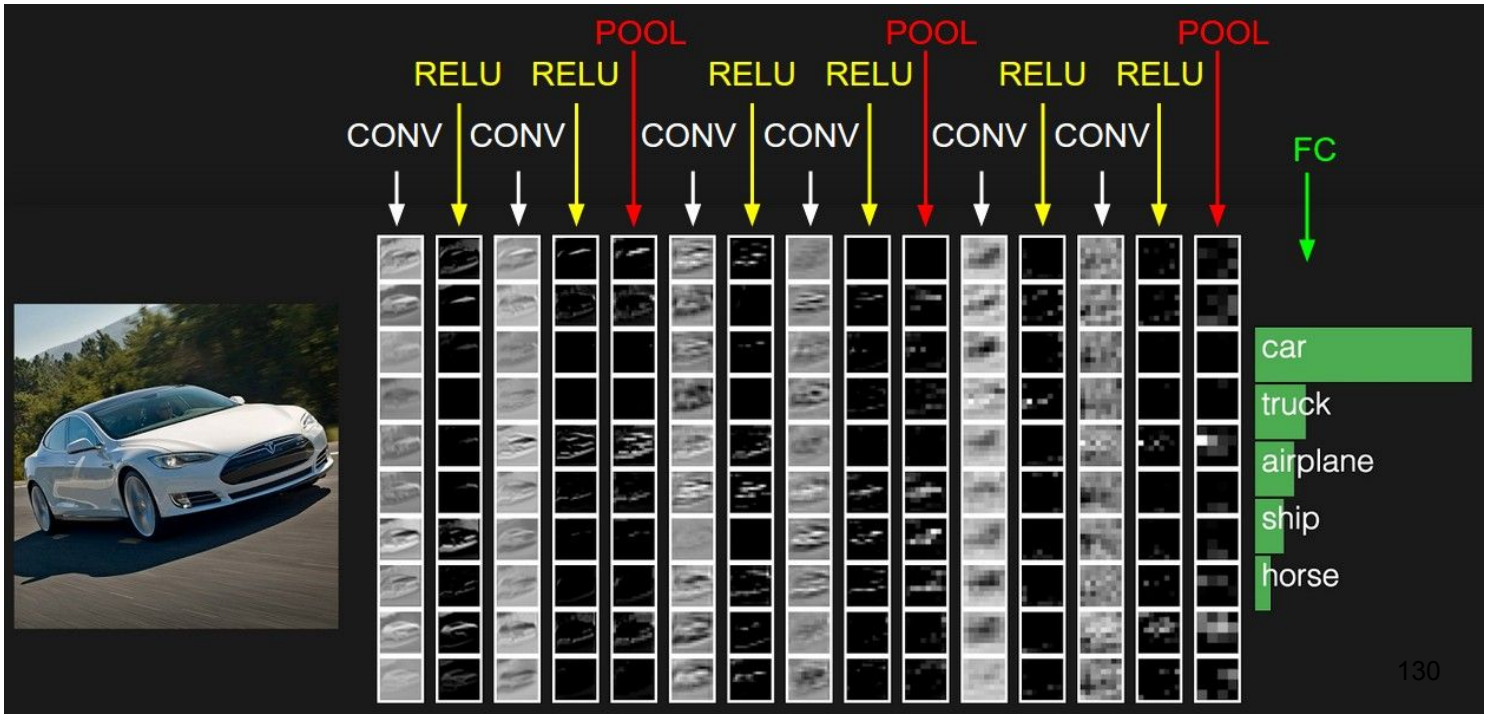
- Create an image classification model and training https://teachablemachine.withgoogle.com/train/image
- You can use the cat and dog datasets here
- https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip



ConvNET Architecture

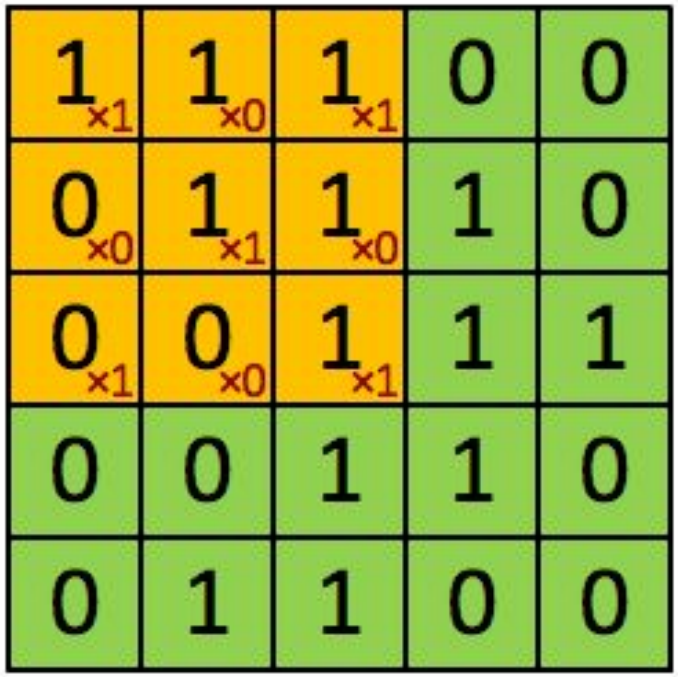
ConvNET architecture consists the follow layers

- Convolutional layer
- Activation layer
- Max Pooling layer
- Dropout layer
- Fully Connected layer

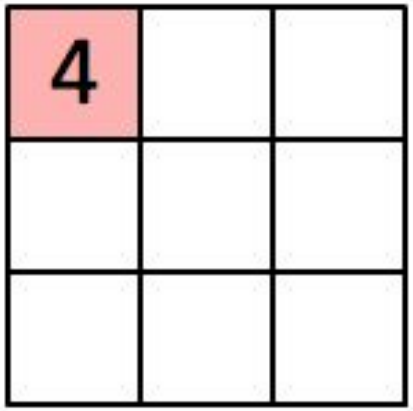


Convolution

- Convolution is the process of sliding a filter across the image
- The filter act as a feature detector



Image



Convolved Feature

Output Image Size

The output image size can be computed by this formula

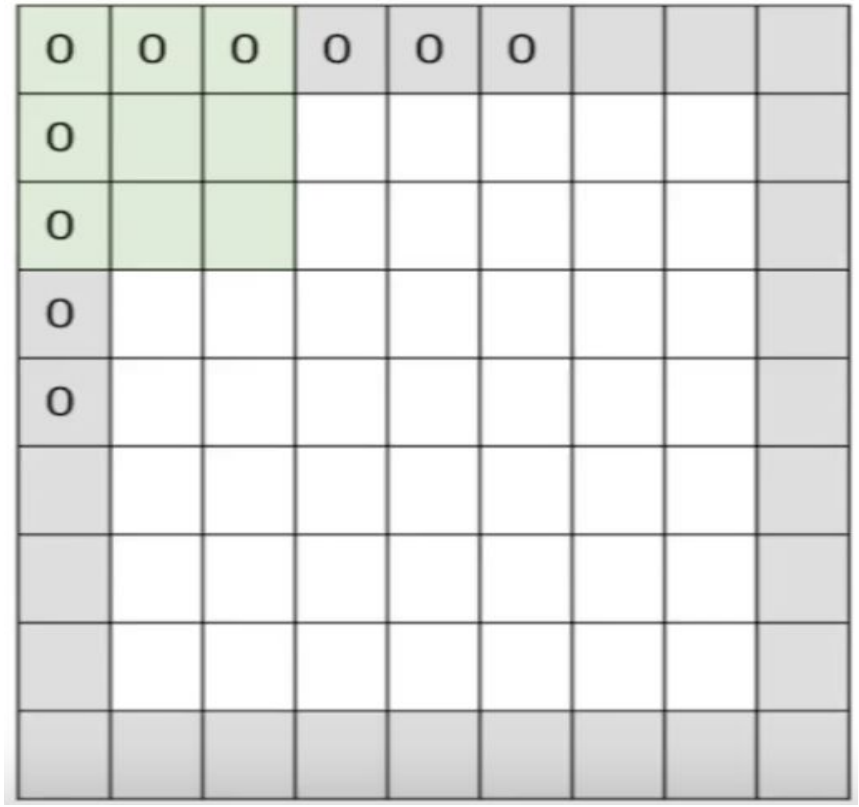
(N-F)/stride + 1
N: input size
F: filter size
S: stride

Eg (5-3)/1+1 = 3
(5-3)/2+1 = 2

Stride is the step taken when sliding the filter

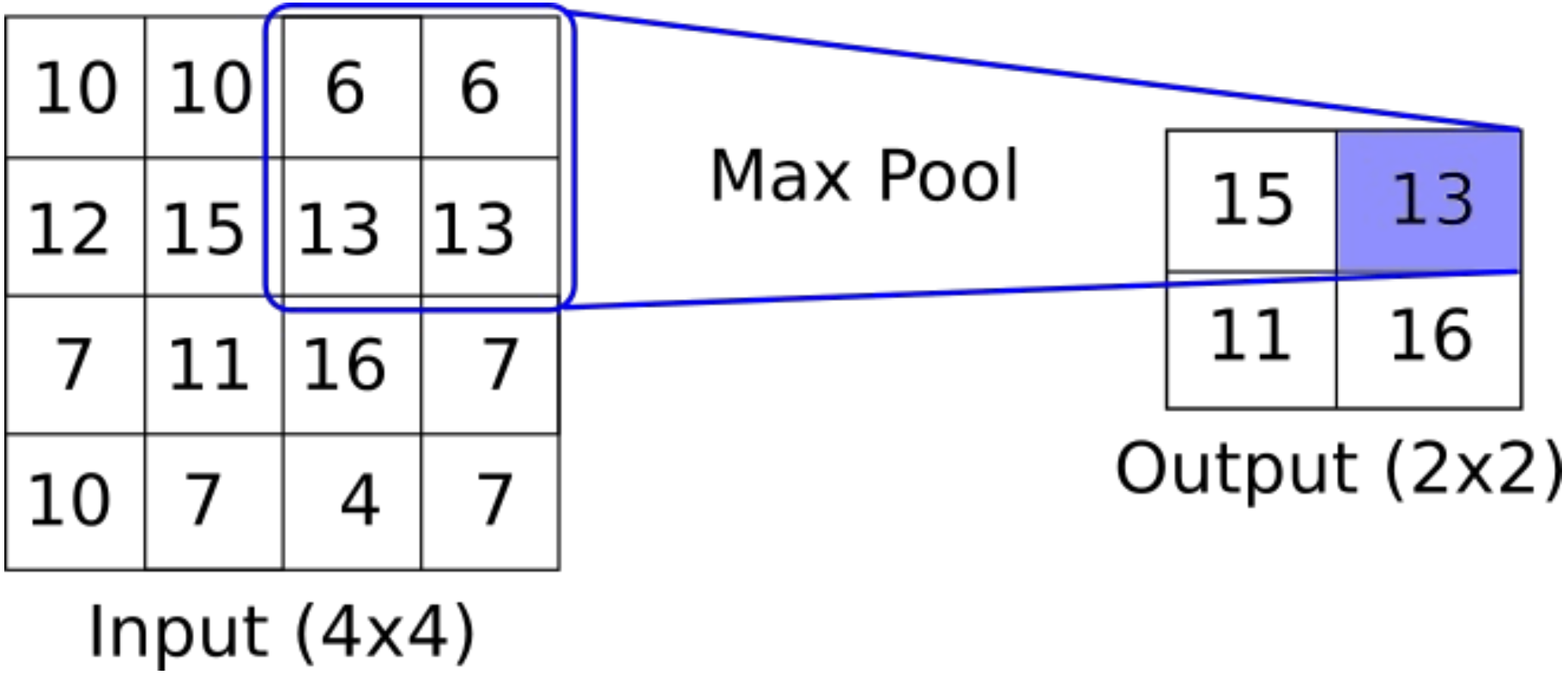
Padding

- Zero padding is used to preserve the output image size and reduce edge information loss.
- There are two types of padding
  - Valid padding - no padding
  - Same padding - keep the same output image size



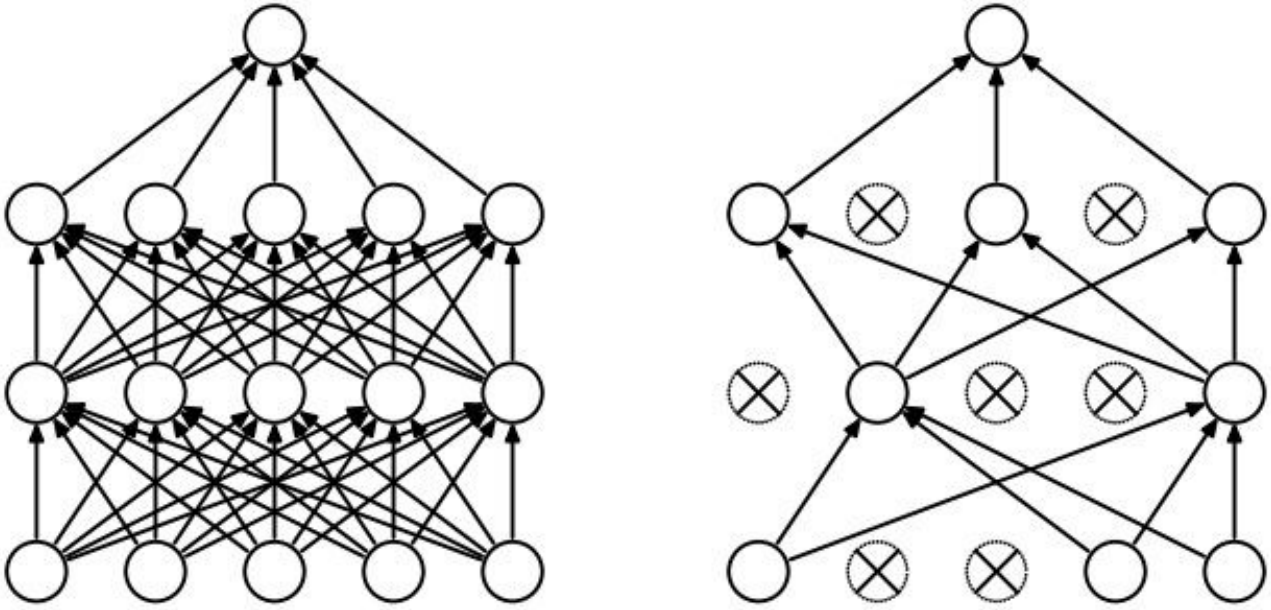
Pooling

- Similar to stride, the purpose of a pooling layer is to downsample the output size. and extract the salient features.
- Maximum or average pooling layers



Dropout

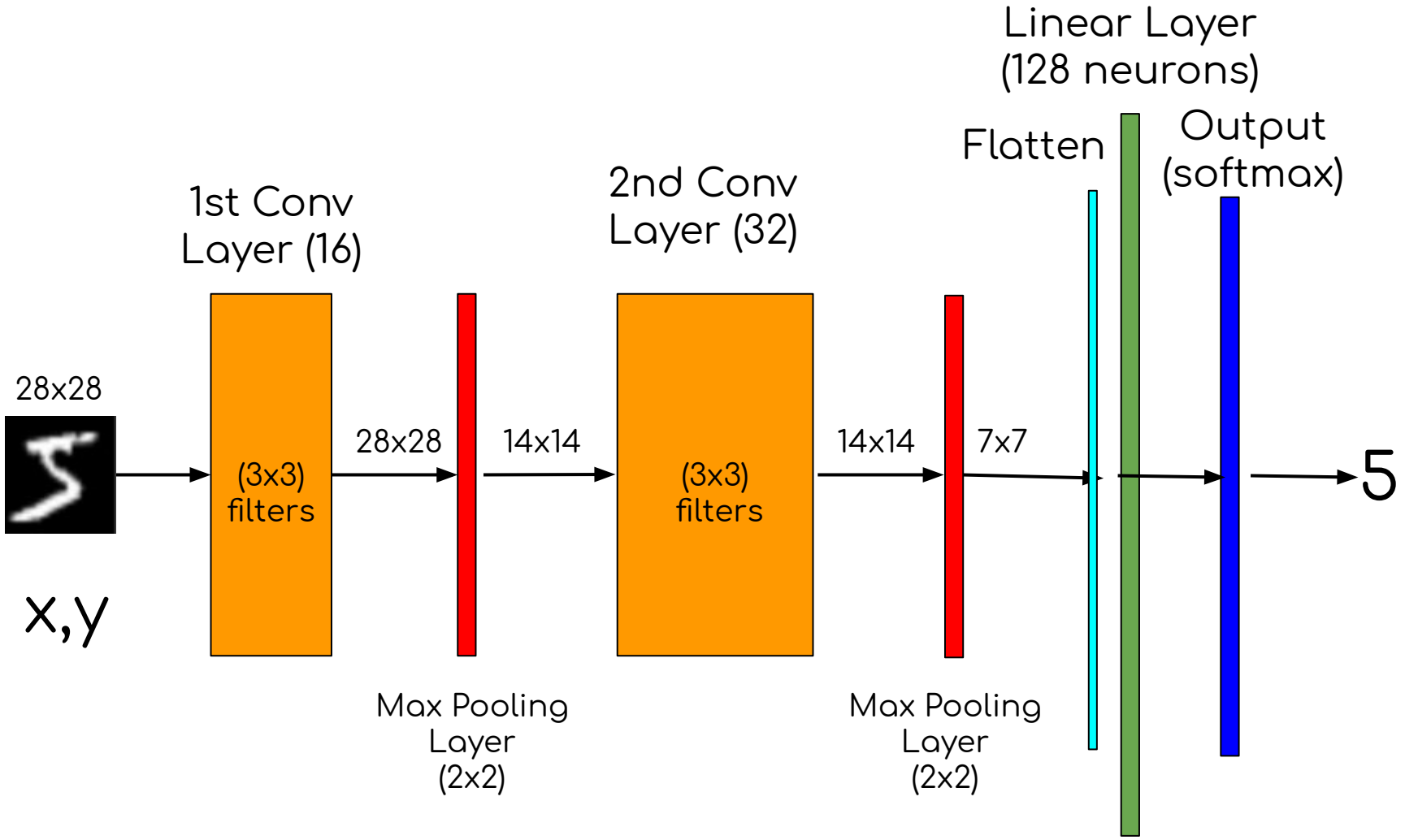
- Dropout is a way regularize the parameters, thus reduce over-fitting and general fit better.
- During training, we randomly select activation functions and make the output equal to zero.
- It is generally apply to the fully connected layer.



(a) Standard Neural Net

(b) After applying dropout.

CNN Demo for MNIST



Define the CNN Model

```
L1 = 16
L2 = 32
L3 = 128

class Model(nn.Module):
    def __init__(self):
        super(Model,self).__init__()
        self.conv1 = nn.Conv2d(1,L1,3,1,1)
        self.conv2 = nn.Conv2d(L1,L2,3,1,1)
        self.fc1 = nn.Linear(L2*7*7,L3)
        self.fc2 = nn.Linear(L3, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = Model()
```

Alternative CNN Model Definition

```
import torch
import torch.nn as nn
import torch.nn.functional as F

L1 = 16
L2 = 32
L3 = 128

model = nn.Sequential(nn.Conv2d(1,L1,3,1,1),
                      nn.ReLU(),
                      nn.MaxPool2d(2),
                      nn.Conv2d(L1,L2,3,1,1),
                      nn.ReLU(),
                      nn.MaxPool2d(2),
                      nn.Flatten(),
                      nn.Linear(L2*7*7,L3),
                      nn.Linear(L3, 10))
```

Add Model to GPU

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

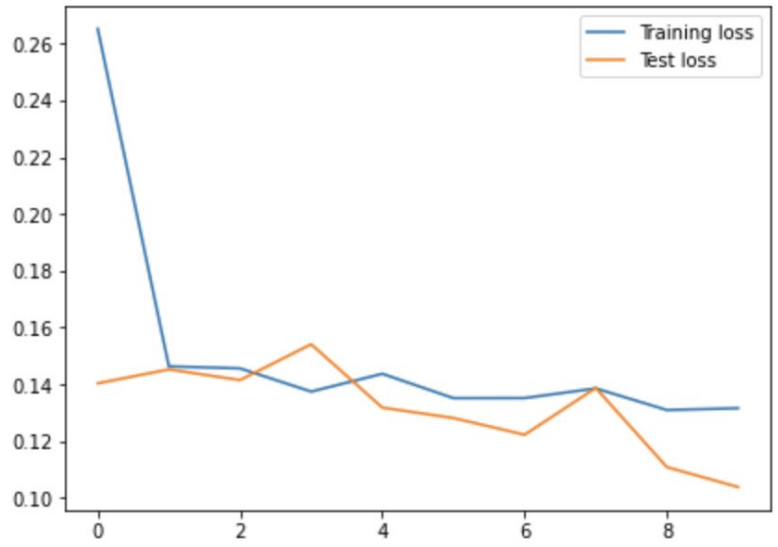
Train the Model

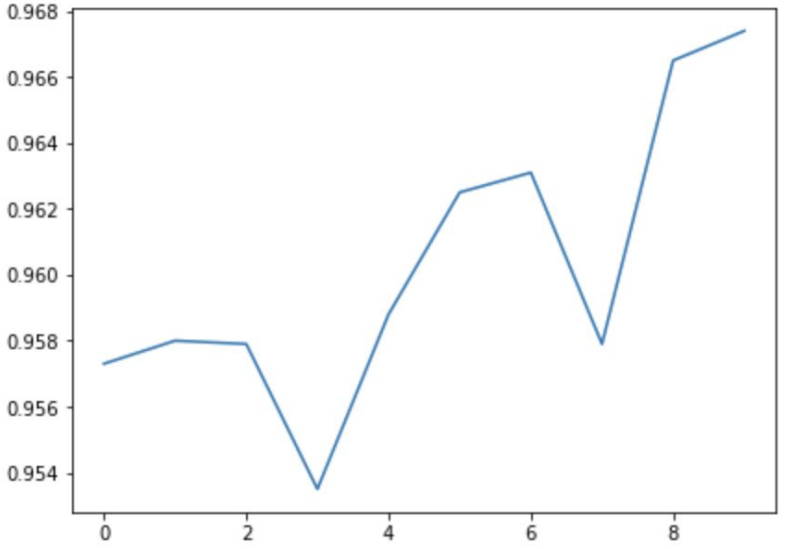
```
for i in range(num_epochs):
    train_loss = 0

    for batch, (X, y) in enumerate(trainloader):
        X = X.to(device)
        y = y.to(device)

        yhat = model(X)

        optimizer.zero_grad()
        loss = criterion(yhat, y)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    train_tracker.append(train_loss/len(trainloader))
    print(f"Epoch({i+1}/{num_epochs}) | Training loss: {train_loss/len(trainloader)} | ",end='')
```

```
Epoch(1/10) | Training loss: 0.1815527715781795 | Test loss: 0.07517748720911302 | Accuracy : 0.9764999747276306
Epoch(2/10) | Training loss: 0.08686092886530153 | Test loss: 0.06301685600368392 | Accuracy : 0.9801999926567078
Epoch(3/10) | Training loss: 0.07878129867931653 | Test loss: 0.07113557542655892 | Accuracy : 0.9785999655723572
Epoch(4/10) | Training loss: 0.07240879847404066 | Test loss: 0.07097163051218507 | Accuracy : 0.9775999784469604
Epoch(5/10) | Training loss: 0.06776289350617164 | Test loss: 0.06310828684380197 | Accuracy : 0.9818999767303467
Epoch(6/10) | Training loss: 0.06799703942527295 | Test loss: 0.0705306206628807 | Accuracy : 0.9801999926567078
Epoch(7/10) | Training loss: 0.06663249518403581 | Test loss: 0.08165361432873594 | Accuracy : 0.9777999520301819
Epoch(8/10) | Training loss: 0.06236315957801568 | Test loss: 0.06675530408520106 | Accuracy : 0.983299970626831
Epoch(9/10) | Training loss: 0.06283533013741194 | Test loss: 0.0810895188391511 | Accuracy : 0.9786999821662903
Epoch(10/10) | Training loss: 0.06212432895201757 | Test loss: 0.07680490346007038 | Accuracy : 0.979699969291687

Number correct : 9797, Total : 10000
Accuracy of the model after 30 epochs on the 10000 test images: 97.97000122070312%
```

Evaluate the Model

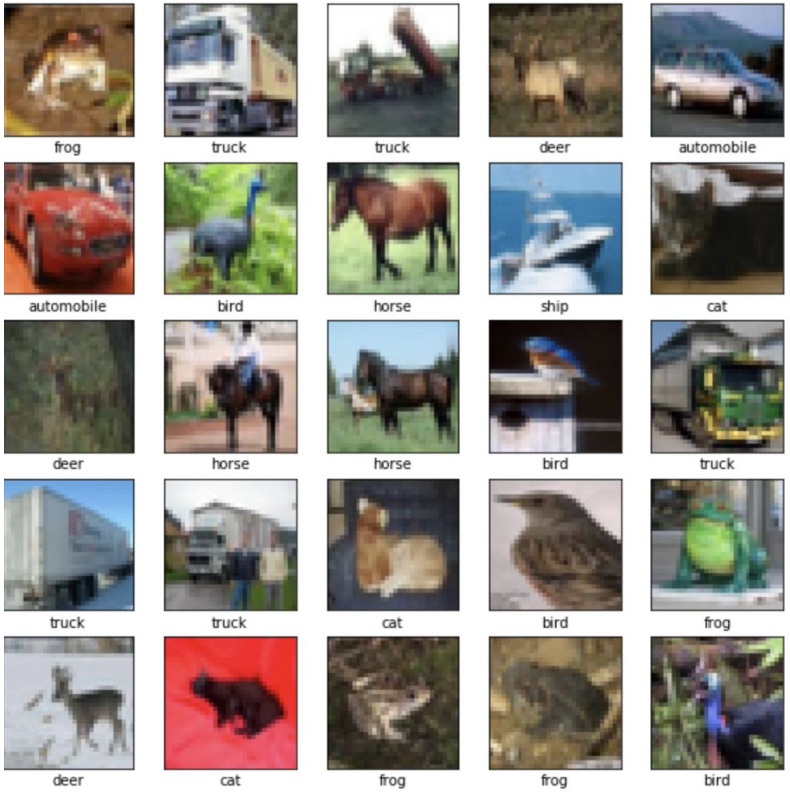
```
import matplotlib.pyplot as plt

plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(train_tracker, label='Training loss')
plt.plot(test_tracker, label='Test loss')
plt.legend()

plt.subplot(1,2,2)
plt.plot(accuracy_tracker, label='Test accuracy')
plt.show()
```





CIFAR-10 Image Dataset

- The CIFAR-10 dataset (https://www.cs.toronto.edu/~kriz/cifar.html)
- 60000 32x32 colour images
- 10 classes, with 6000 images per class.
- 50000 training images and 10000 test images

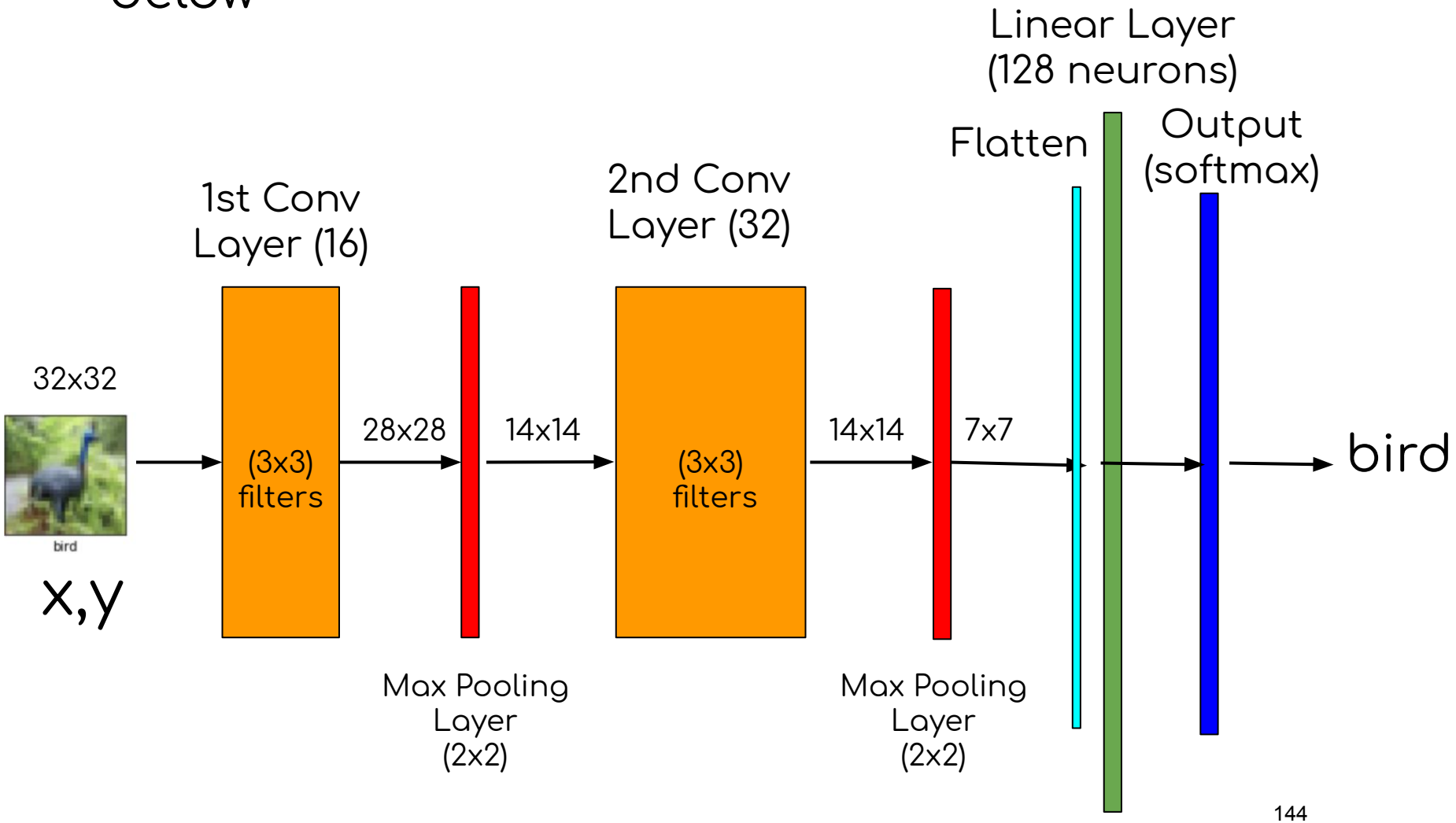


CIFAR10 Classes

airplane	[1 0 0 0 0 0 0 0 0 0]
automobile	[0 1 0 0 0 0 0 0 0 0]
bird	[0 0 1 0 0 0 0 0 0 0]
cat	[0 0 0 1 0 0 0 0 0 0]
deer	[0 0 0 0 1 0 0 0 0 0]
dog	[0 0 0 0 0 1 0 0 0 0]
frog	[0 0 0 0 0 0 1 0 0 0]
horse	[0 0 0 0 0 0 0 1 0 0]
ship	[0 0 0 0 0 0 0 0 1 0]
truck	[0 0 0 0 0 0 0 0 0 1]

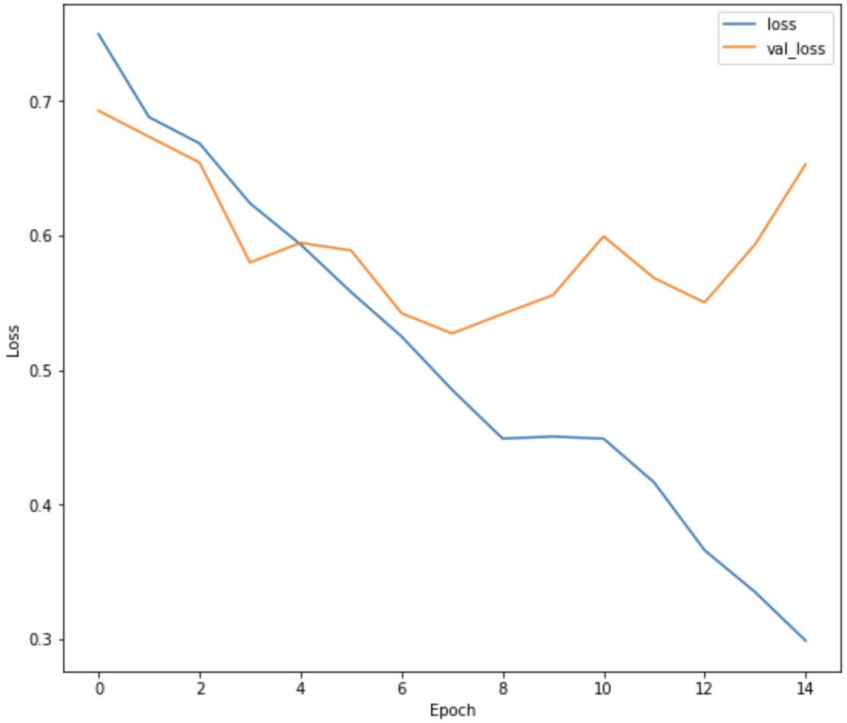
Activity: CNN on CIFAR10 Dataset

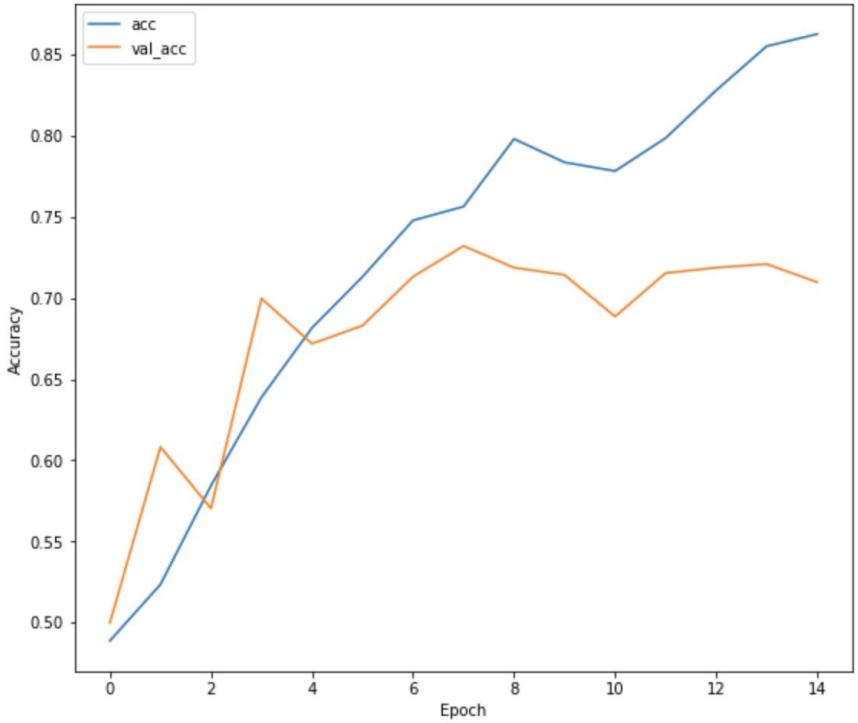
- Create a CNN model for CIFAR10 dataset as shown below



Issues with Small Dataset

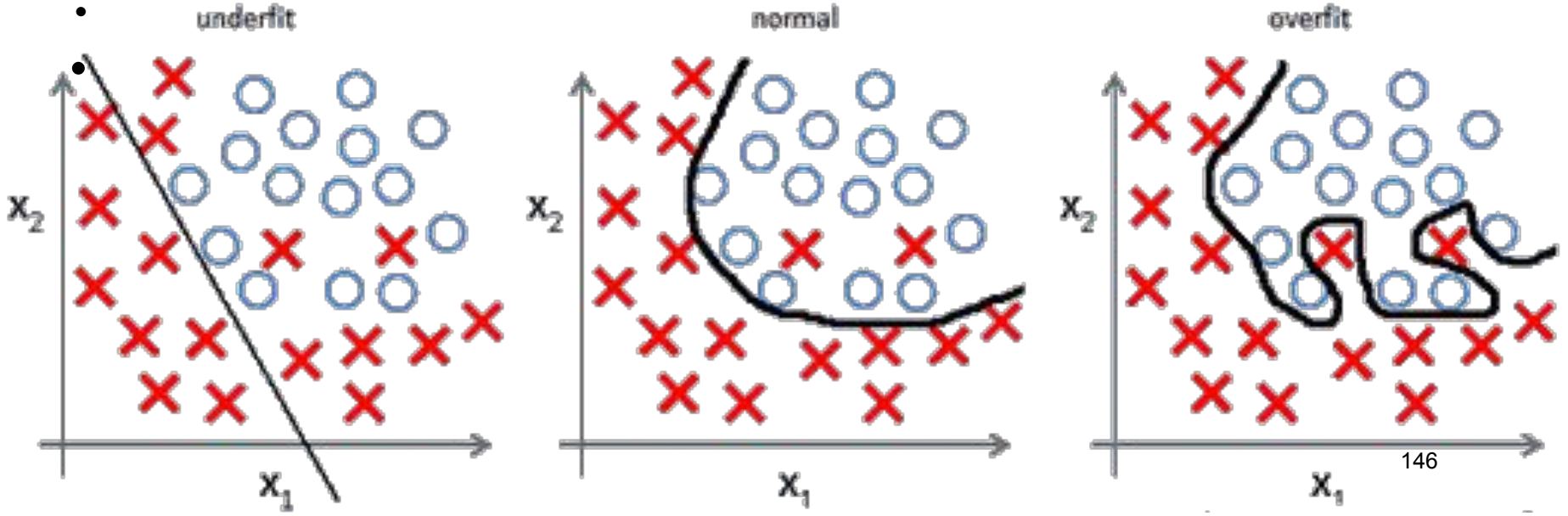
For small dataset, a general issue is overfitting. While the training loss reduce with training epoch, the testing loss has U-turned and increased with training epoch.





Overfitting and Underfitting

- **Overfitting** occurs when a statistical model or machine learning algorithm captures the noise of the data. Intuitively, overfitting occurs when the model or the algorithm fits the data too well. Specifically, overfitting occurs if the model or algorithm shows low bias but **high variance**.
- **Underfitting** occurs when a statistical model or machine learning algorithm cannot capture the underlying trend of the data. Intuitively, underfitting occurs when the model or the algorithm does not fit the data well enough. Specifically, underfitting occurs if the model or algorithm shows low variance but **high bias**
  -
-
-



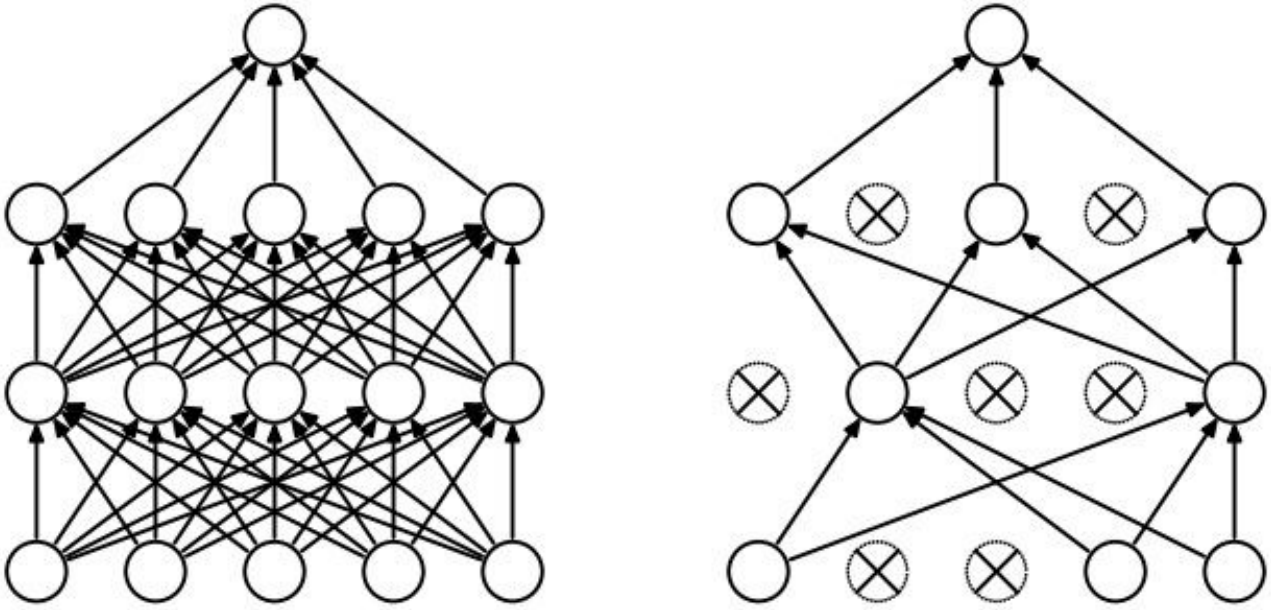
Methods to Solve Overfitting

Overfitting issues are commonly solved by the following techniques:

- Increase training data (expensive)
- Reduce model capacity (reduce learning)
- L1 or L2 weight regularization
- Early stopping (reduce learning)
- Data augmentation
- Dropout

Dropout

- Dropout is a way regularize the parameters, thus reduce over-fitting and general fit better.
- During training, we randomly select activation functions and make the output equal to zero.
- It is generally apply to the fully connected layer.



(a) Standard Neural Net

(b) After applying dropout.

Load Small Datasets

- Download the hymenoptera_data dataset from Google classroom and upload to your Google Drive or local computer.
- That data has about 120 training images each for ants and bees.
- There are 75 validation images for each class. Usually, for a very small dataset, the model is prompt to over-hitting.

```
data_dir = '/content/drive/MyDrive/dataset/hymenoptera_data'

trainset = datasets.ImageFolder(os.path.join(data_dir, 'train'),
transforms['train'])
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
shuffle=True)

testset = datasets.ImageFolder(os.path.join(data_dir, 'val'),
transforms['val'])
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
shuffle=True)
```

Mount Google Drive

- If you use Google Drive for storing data, you need to mount the Google Drive to access your data

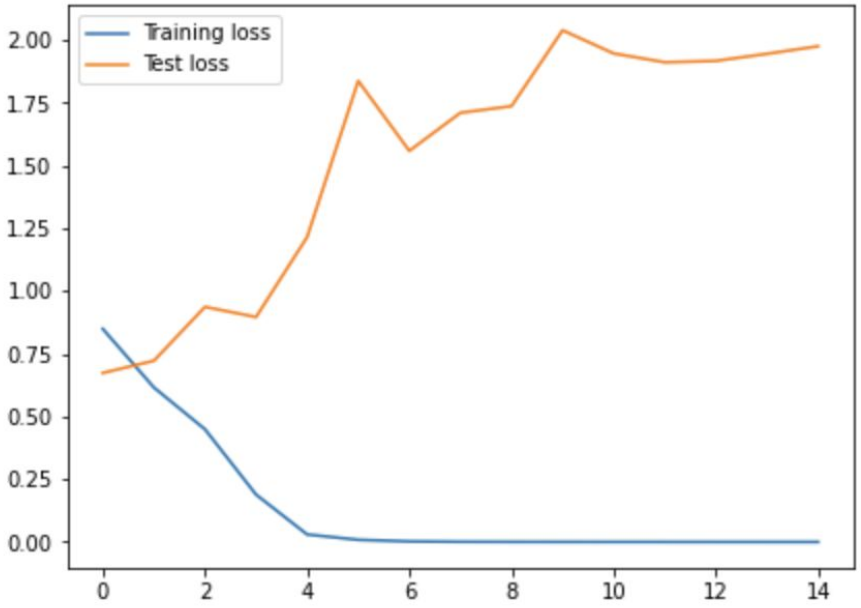
```
from google.colab import drive
drive.mount('/content/drive')
```

Resize the Images

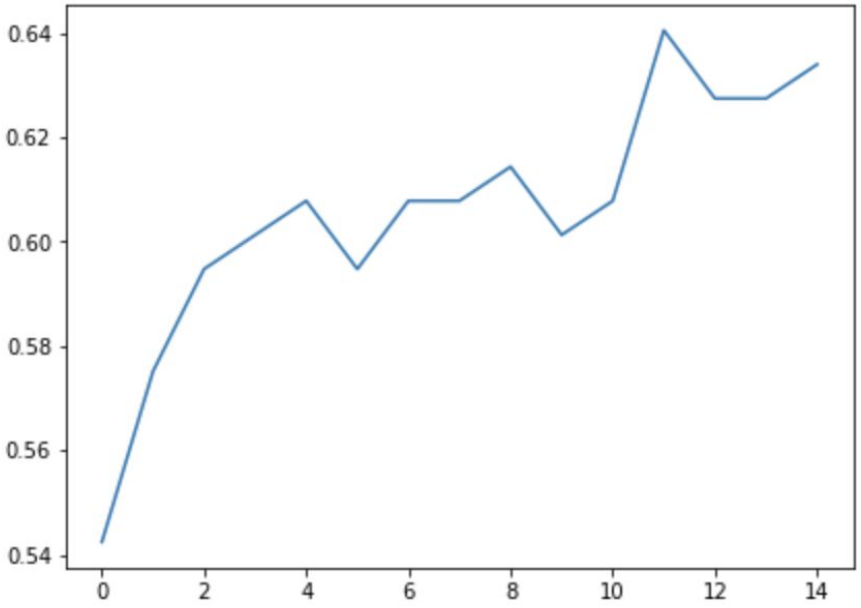
```
mean = [0.5, 0.5, 0.5]
std = [0.5, 0.5, 0.5]

transforms = {
    'train': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=mean, std=std)
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=mean, std=std)
    ]),
}
```

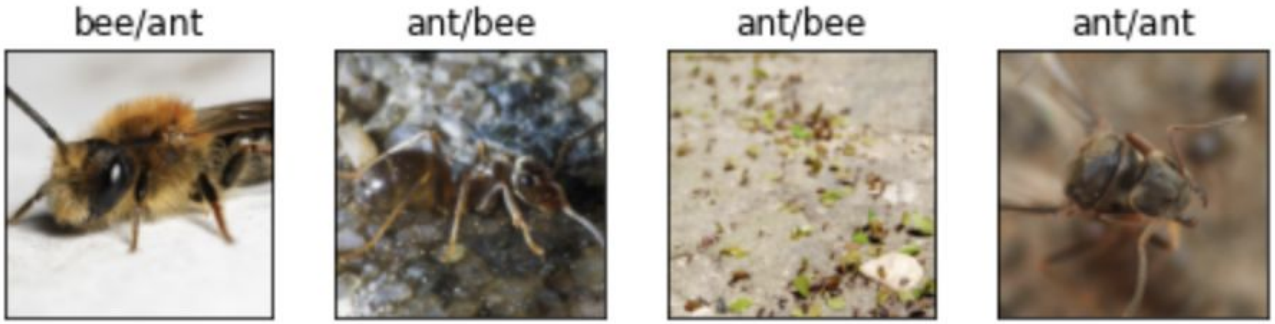
Result (Baase line)

Loss



Accuracy



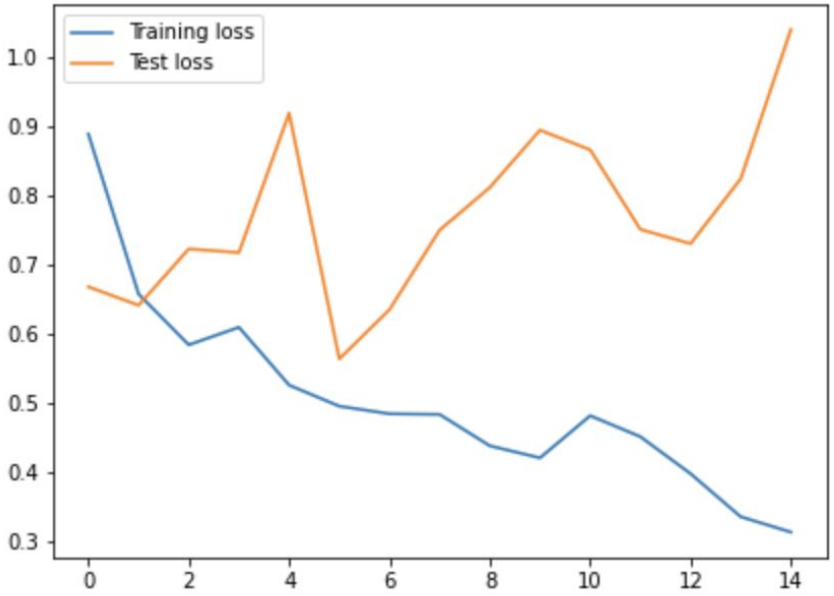


Apply Data Augmentation

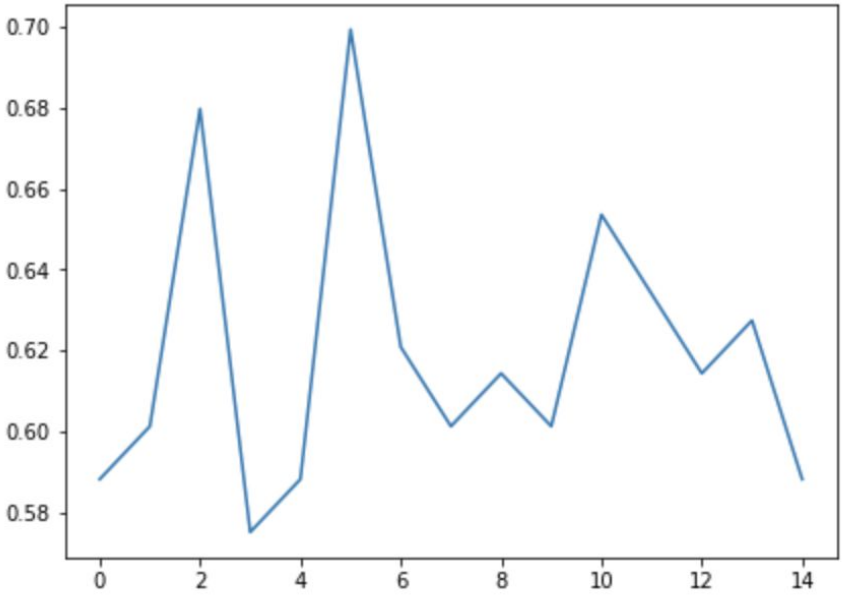
```
mean = [0.5, 0.5, 0.5]
std = [0.5, 0.5, 0.5]

transforms = {
    'train': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.RandomRotation(20),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=mean, std=std)
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=mean, std=std)
    ]),
}
```

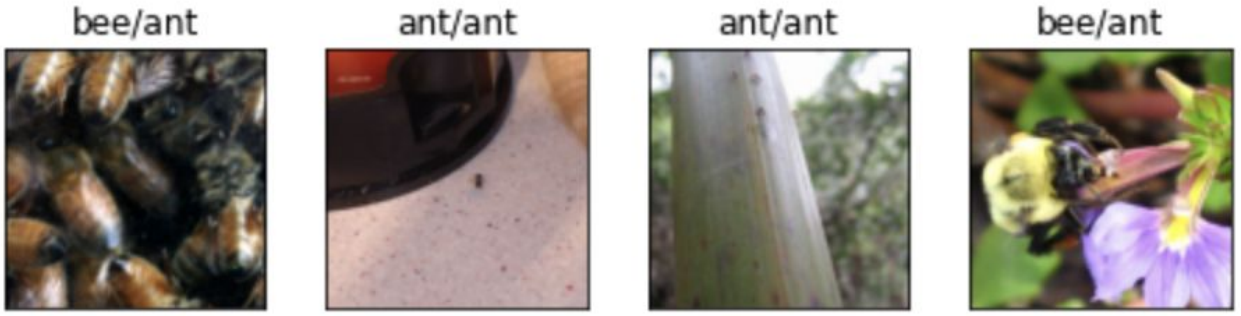
Result (Data Augmentation)

Loss



Accuracy



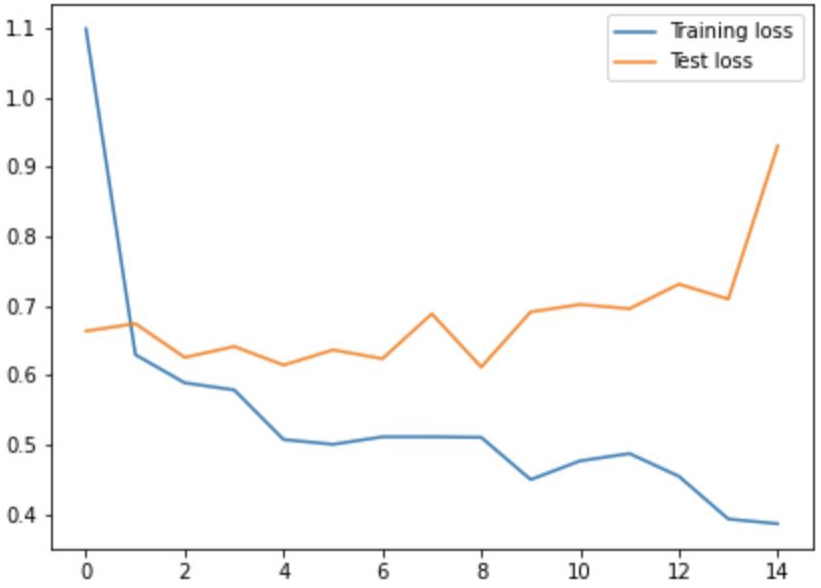


Apply DA and Dropout

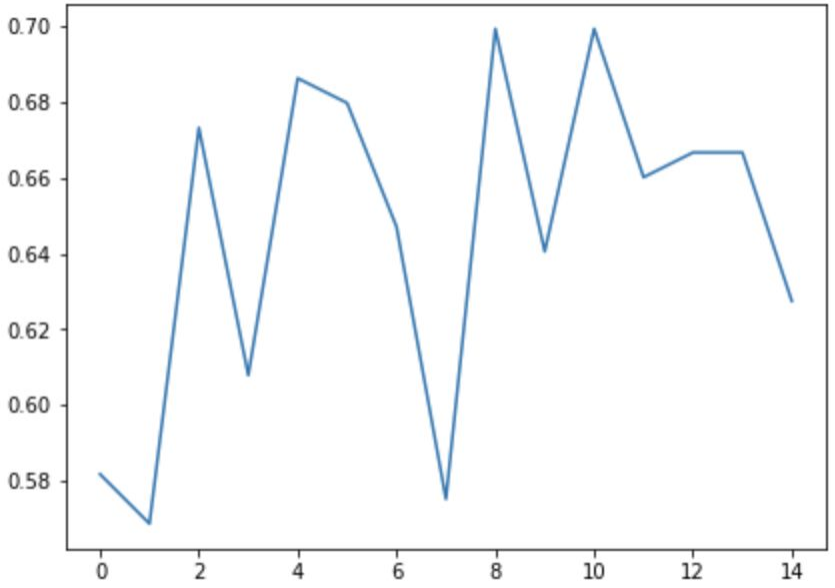
```
class Model(nn.Module):
    def __init__(self):
        super(Model,self).__init__()
        self.conv1 = nn.Conv2d(3,L1,3,1,1)
        self.d1 = nn.Dropout(0.2)
        self.conv2 = nn.Conv2d(L1,L2,3,1,1)
        self.fc1 = nn.Linear(L2*56*56,L3)
        self.fc2 = nn.Linear(L3, 2)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = self.d1(x)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view( x.size(0),-1)

        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = Model()
```

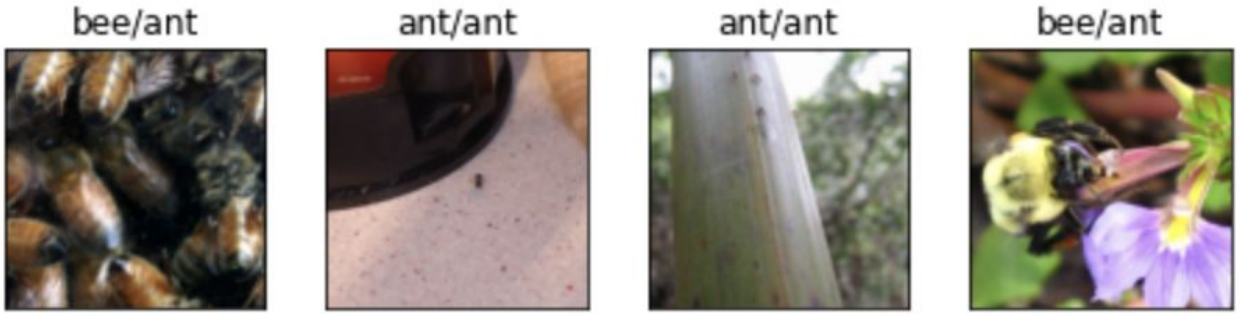
Result (DA+Dropout)

Loss



Accuracy



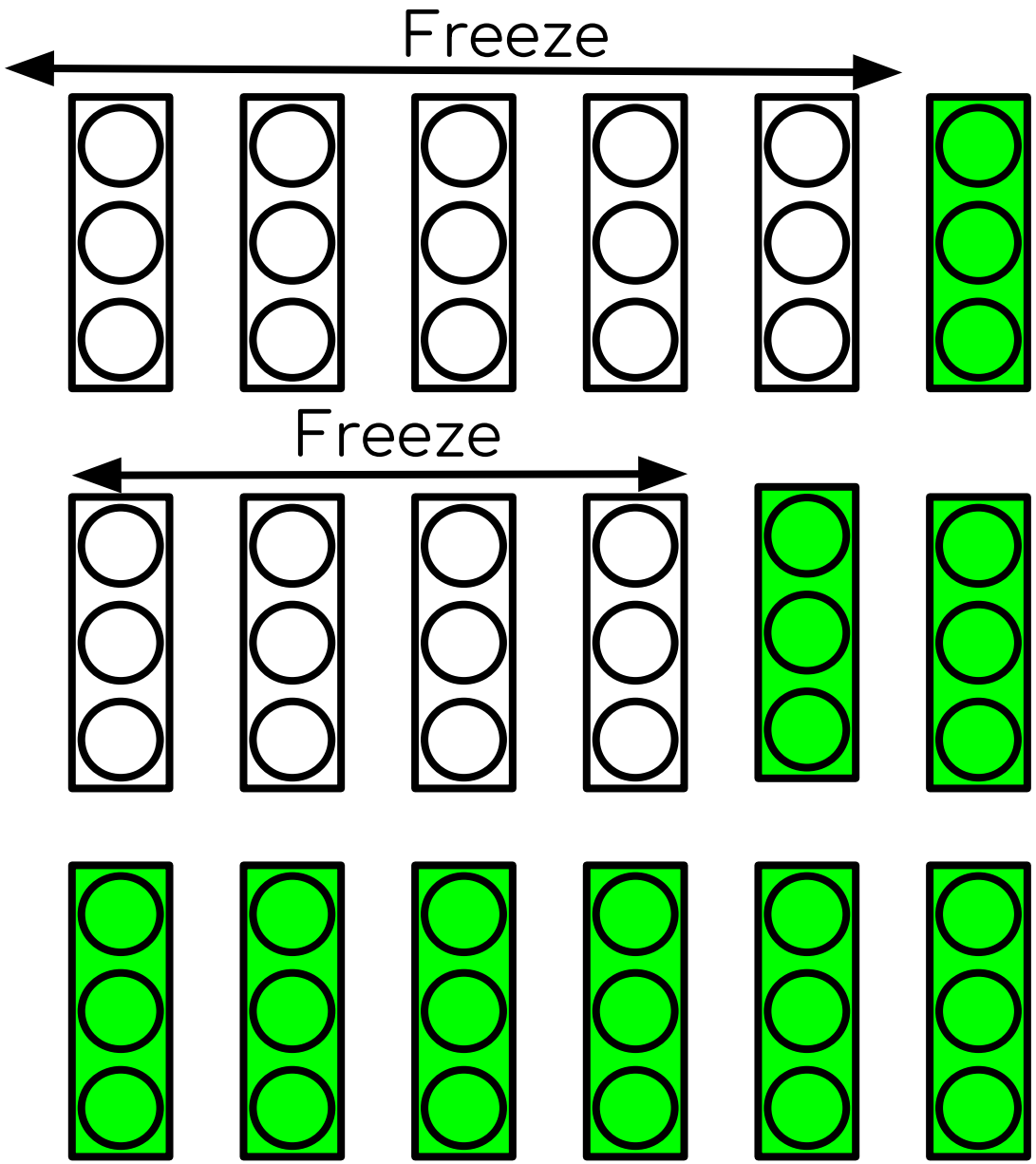


Topic 5
Transfer Learning

Transfer Learning

- A **pre-trained model** is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task.
- You either use the pretrained model as it is, or use transfer learning to customize this model to a given task.
- The intuition behind transfer learning is that if a model trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world.
- You can then take advantage of these learned feature maps without having to start from scratch training a large model on a large dataset.

Transfer Learning Approaches



Feature Extraction:

- Freeze feature extraction layers
- Replace the classifier layers
- Train the classifier layers

Fine Tuning:

- Freeze the bottom layers.
- Train the top few layers and classifier layers

Initialization

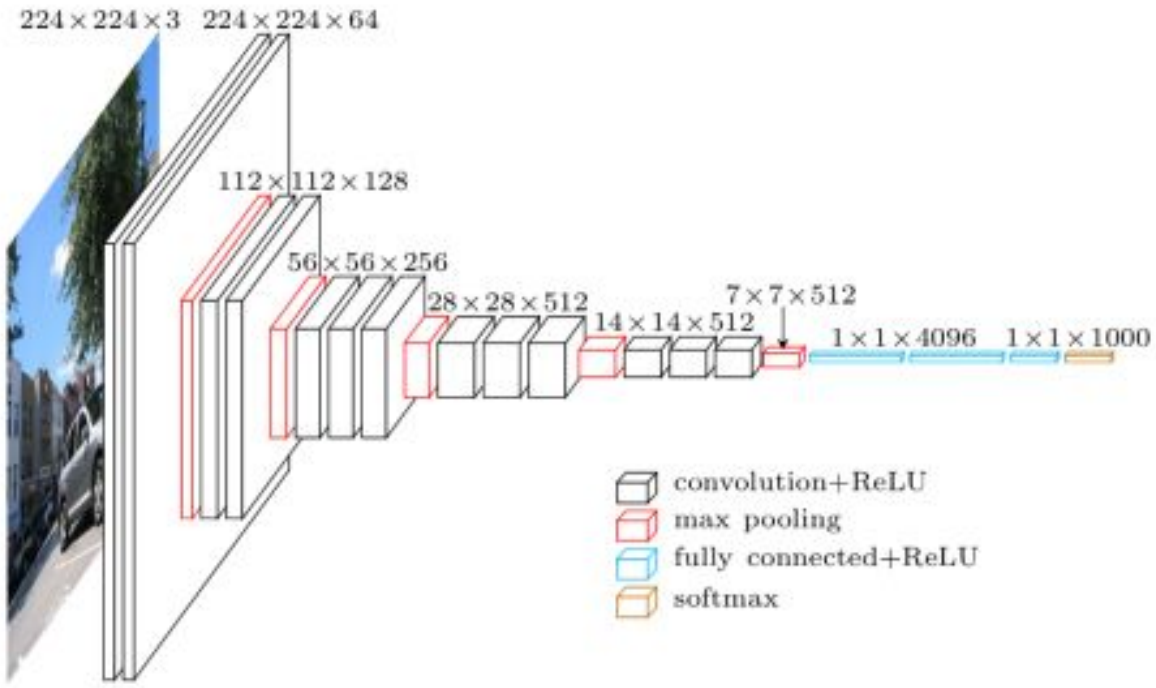
- Retrain the entire layers with preloaded weights as initialization

Pre-Trained TorchVison Models

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNet v2
- ResNeXt
- Wide ResNet
- MNASNet

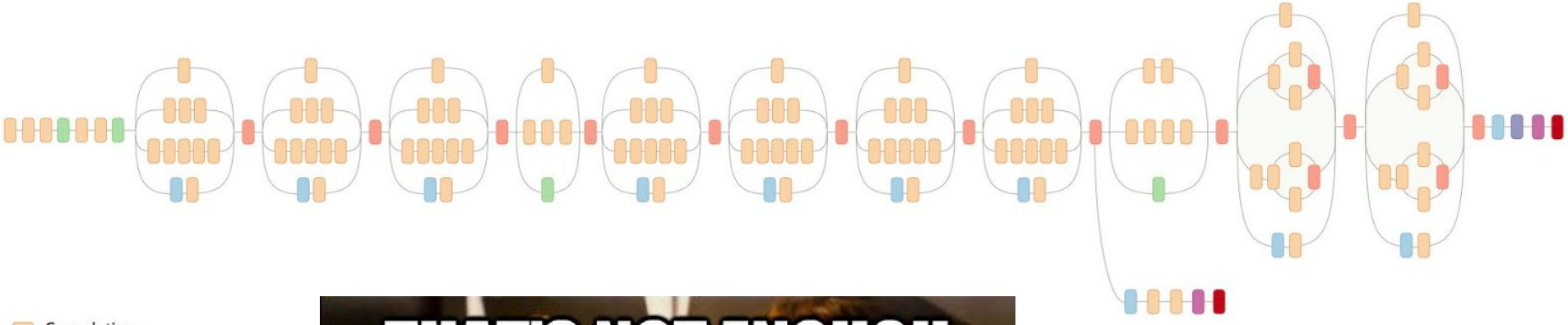
VGG16 Model

- Runner of ILSVRC 2014
- Pre-trained on ImageNet. Input image size is 224x224px
- Simple architecture - 3x3 filters, 2x2 max pooling, double filter numbers after each max pooling



Inception Model

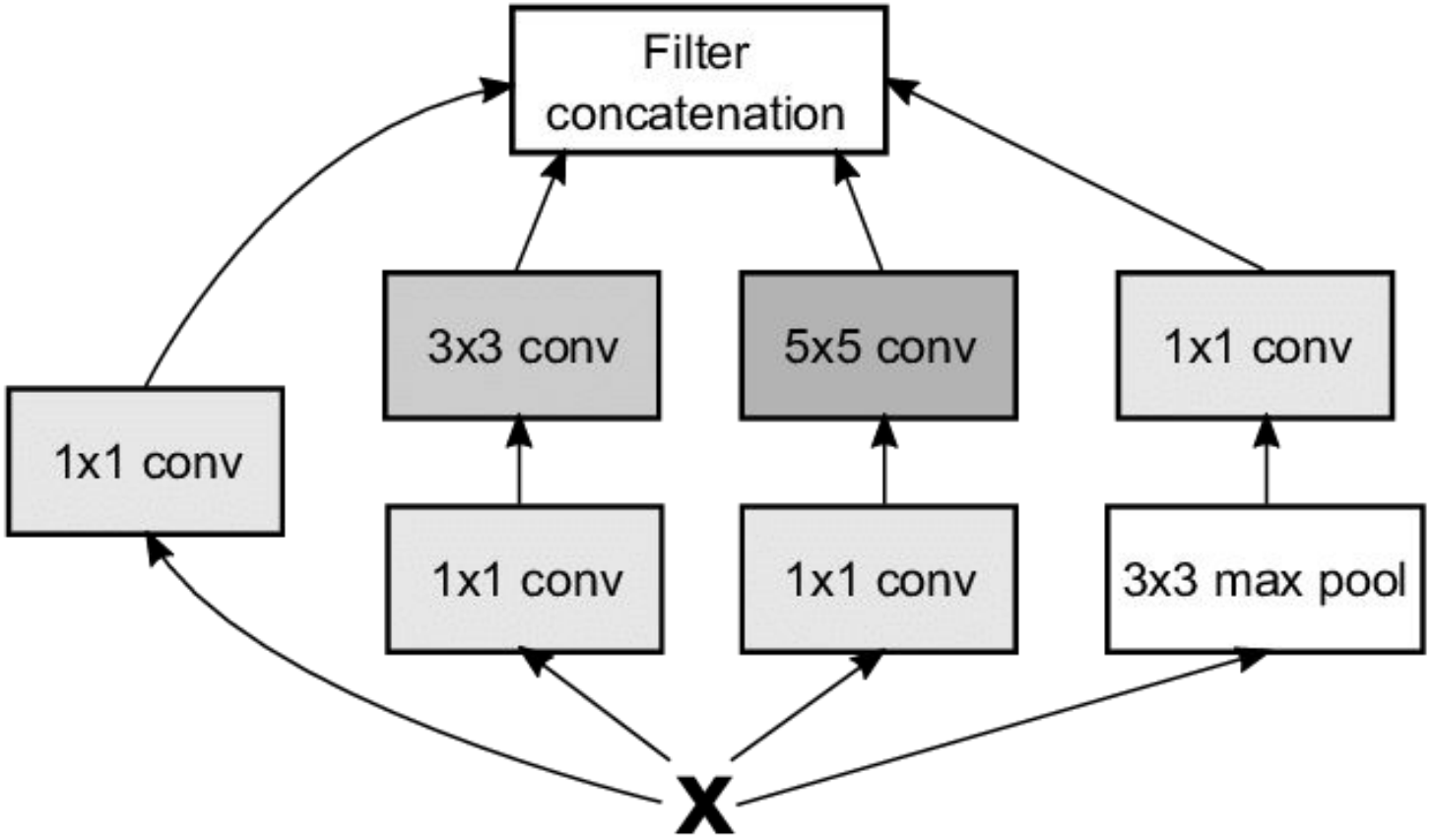
- Winner of ILSVRC 2014, also known as GoogleNet
- Pre-trained on ImageNet. Input Image size is 299x299px
- Consist of many inception modules



Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax

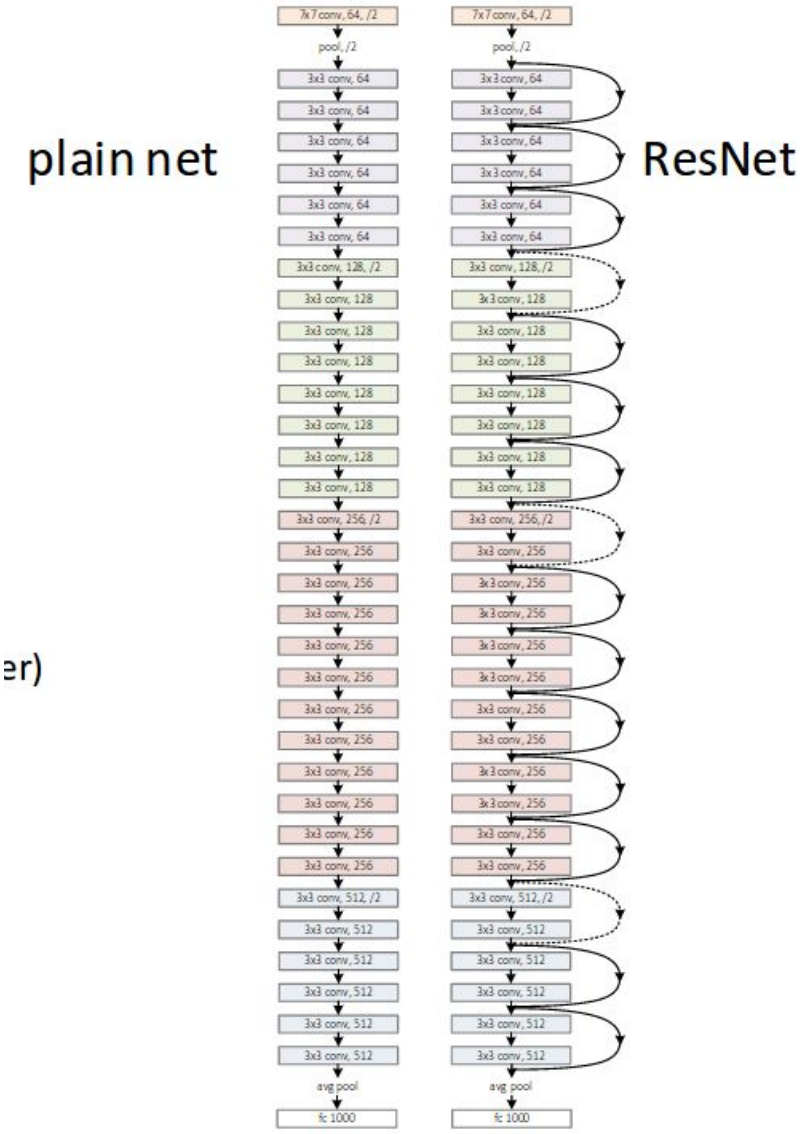
Single Inception Module

The basic idea of inception is to apply different filter sizes eg 1x1, 3x3 and 5x5



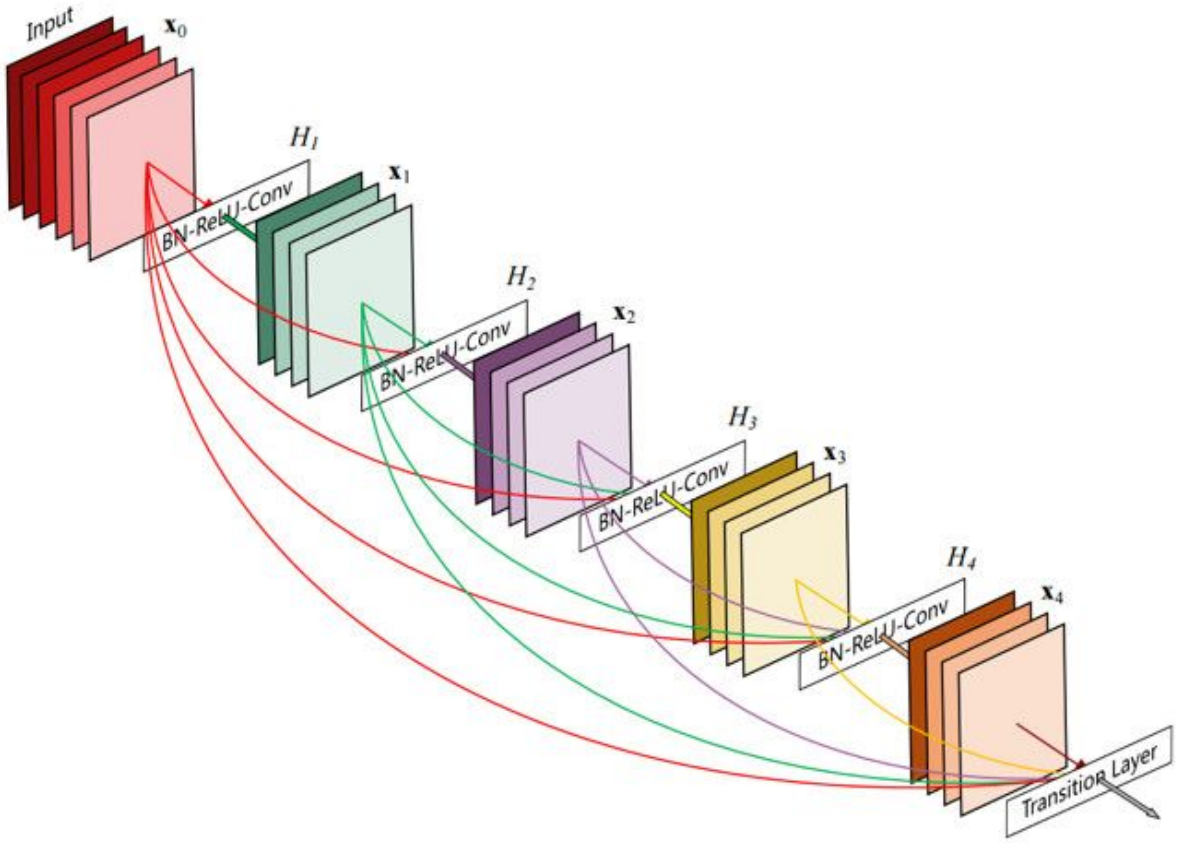
ResNet

- Residual network - winner of ILSVRC 2015 from Microsoft
- Pre-trained on ImageNet. Input image size is 224x224px
- With more layers, the NN performance might degrade due to vanishing gradient issue.
- ResNet allow NN to go deeper by skipping the bottleneck layers



DenseNet

Dense Convolutional Network (DenseNet) is an extension of ResNET, which connects each layer to every other layer in a feed-forward fashion.



Import Pretrained Model

```
import torchvision.models as models

model = models.vgg16(pretrained=True)
```

Model Features

model.features

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace=True)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace=True)
  (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (18): ReLU(inplace=True)
  (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace=True)
  (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): ReLU(inplace=True)
  (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (25): ReLU(inplace=True)
  (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (27): ReLU(inplace=True)
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): ReLU(inplace=True)
  (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

Import and Preprocess Data

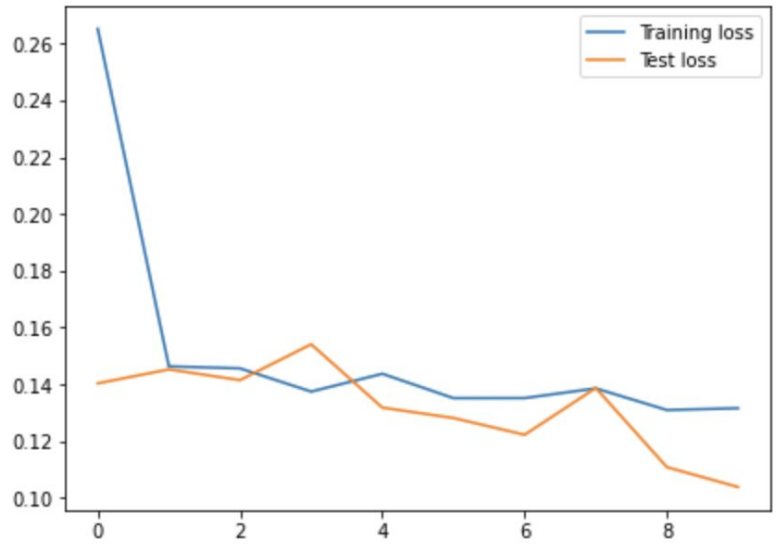
```
from PIL import Image

image_path = '/content/cat.jpg'

preprocess = transforms.Compose([
    transforms.Resize(224),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229,
0.224, 0.225))
])

img = Image.open(image_path)
img_tensor = preprocess(img)
img_tensor.unsqueeze_(0)
predict = model(img_tensor)
```

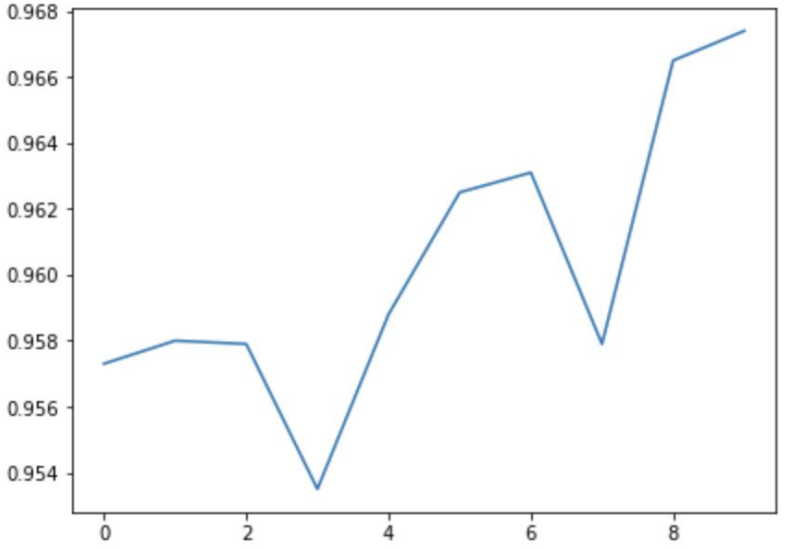
Download ImageNet Labels

```
import requests

LABELS_URL =
'https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json'
labels = {int(key):value for (key, value) in
requests.get(LABELS_URL).json().items()}
```

Evaluate the Model

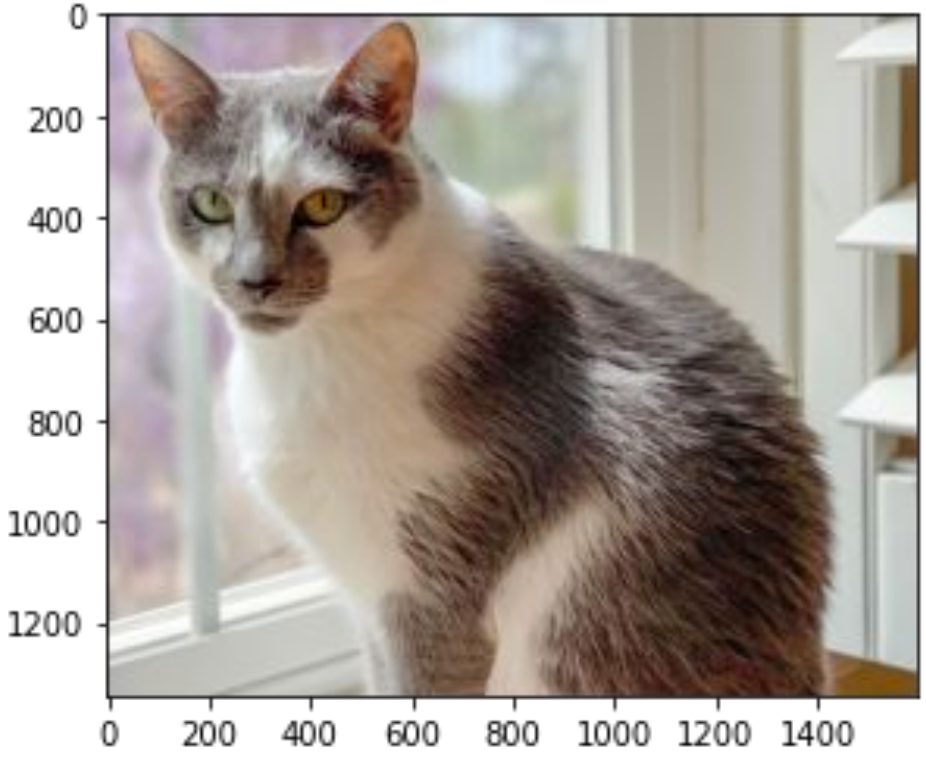
```
import matplotlib.pyplot as plt

plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(train_tracker, label='Training loss')
plt.plot(test_tracker, label='Test loss')
plt.legend()

plt.subplot(1,2,2)
plt.plot(accuracy_tracker, label='Test accuracy')
plt.show()
```





Prediction Using Pre-trained Model

```
import matplotlib.pyplot as plt

predict = model(img_tensor)

print(labels[predict.data.numpy().argmax()])
plt.imshow(img)
```



['n02124075', 'Egyptian_cat']

Activity: Pre-trained Model

- Load the RESNET pre-trained model
- Test out an image downloaded from internet.

Feature Extraction- Import Pretrained Model

```
import torchvision.models as models

model = models.vgg16(pretrained=True)
```

Replace Classifier

```
for param in model.parameters():
  param.requires_grad = False

model.classifier[-1] = nn.Sequential(
                    nn.Linear(in_features=4096,
out_features=10)
```

```
Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace=True)
  (2): Dropout(p=0.5, inplace=False)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace=True)
  (5): Dropout(p=0.5, inplace=False)
  (6): Sequential(
    (0): Linear(in_features=4096, out_features=10, bias=True)
  )
)
```

Loss Function & Optimizer

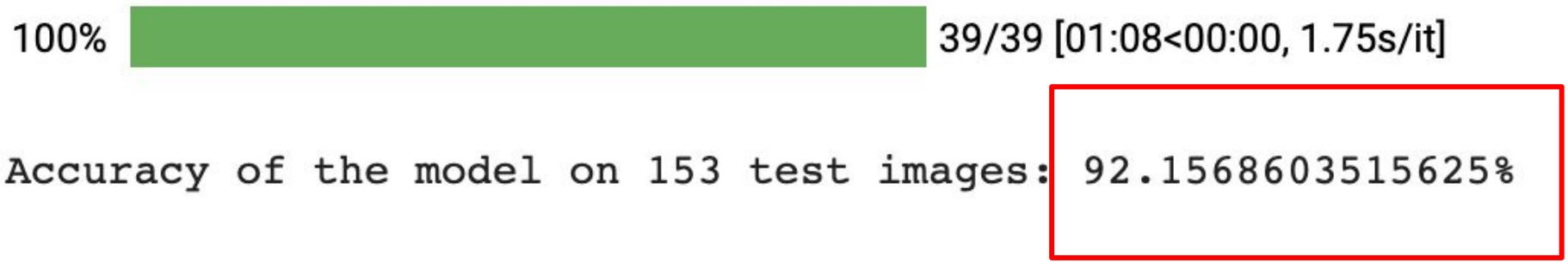
```
from torch.optim import Adam

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(),
lr=0.001)
```

Retrain the Model

```
from tqdm.notebook import trange,tqdm
num_epochs = 2

for i in trange(num_epochs):
    train_loss = 0

    for (X, y) in tqdm(trainloader):
        X = X.to(device)
        y = y.to(device)

        yhat = model(X)

        optimizer.zero_grad()
        loss = criterion(yhat, y)
        loss.backward()
        optimizer.step()

        train_loss  += loss.item()

    print(f"Epoch({i+1}/{num_epochs}) | Training loss:
{train_loss/len(trainloader)} | ",end='')
```

Evaluate the Model

```
from tqdm.notebook import trange,tqdm

model.to('cpu')
model.eval()
with torch.no_grad():
    num_correct = 0
    total = 0

    for (X,y) in tqdm(testloader):
        yhat = model(X)
        _, pred = torch.max(yhat.data, 1)
        total += y.size(0)
        num_correct += (pred == y.data).sum()
    print(f'Accuracy of the model on {total} test images: {num_correct * 100 / total}% ')
```



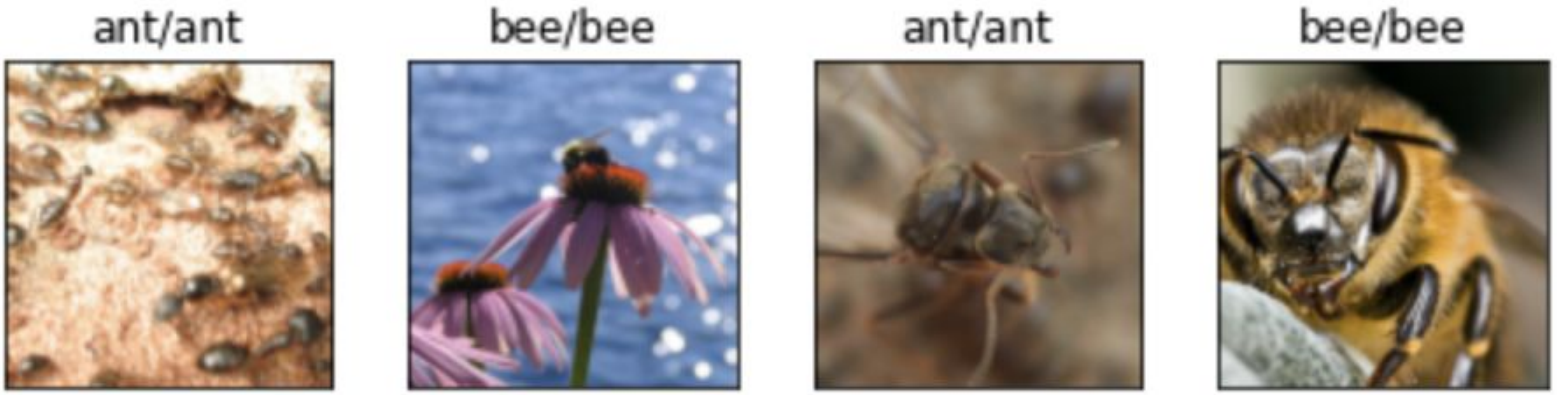
Model.eval()

- model.eval() is a kind of switch for some specific layers/parts of the model that behave differently during training and inference (evaluating) time. For example, Dropouts Layers, BatchNorm Layers etc.
- You need to turn off them during model evaluation, and .eval() will do it for you.
- In addition, the common practice for evaluating/validation is using torch.no_grad() in pair with model.eval() to turn off gradients computation.

```
# evaluate model:
model.eval()
with torch.no_grad():
    ...
    out_data = model(data)
```

Test the Model

```
import matplotlib.pyplot as plt
images,labels = next(iter(testloader))
output = model(images)
_,pred = torch.max(output,1)

fig = plt.figure(figsize=(10,10))
for i in range(4):
  plt.subplot(5,5,i+1)
  show_img(images[i])
  plt.title(get_class(labels[i])+"/"+get_class(pred[i]))
  plt.xticks([])
  plt.yticks([])
```



Fine Tuning - Load Pre-trained Model

```
import torchvision.models as models

model = models.vgg16(pretrained=True)
```

UnFreeze Layers and Replace Classifier

```
for param in model.parameters():
  param.requires_grad = False

for i in range(24,31):
  model.features[i].requires_grad = True

model.classifier[-1] = nn.Sequential(
                    nn.Linear(in_features=4096,
out_features=10)
                    )
```

Loss Function & Optimizer

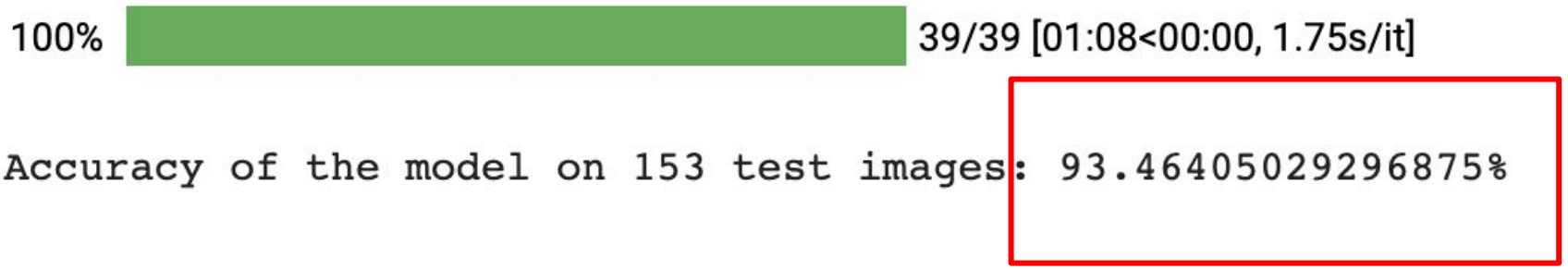
```
from torch.optim import Adam

loss_func = nn.CrossEntropyLoss()

model = model.to(device)
optimizer = Adam(filter(lambda p: p.requires_grad, model.parameters()))
```

Re-train the Model

```
from tqdm.notebook import trange,tqdm

num_epochs = 2

for i in trange(num_epochs):
    train_loss = 0

    for (X, y) in tqdm(trainloader):
        X = X.to(device)
        y = y.to(device)

        yhat = model(X)

        optimizer.zero_grad()
        loss = criterion(yhat, y)
        loss.backward()
        optimizer.step()

        train_loss  += loss.item()

    print(f"Epoch({i+1}/{num_epochs}) | Training loss: {train_loss/len(trainloader)} | ",end='')
```

Evaluate the Model

```
from tqdm.notebook import trange,tqdm

model.to('cpu')
model.eval()
with torch.no_grad():
    num_correct = 0
    total = 0

    #set_trace()
    for (X,y) in tqdm(testloader):

        yhat = model(X)
        _, pred = torch.max(yhat.data, 1)
        total += y.size(0)
        num_correct += (pred == y.data).sum()

    print(f'Accuracy of the model on {total} test images: {num_correct * 100 / total}%
```



Test the Model

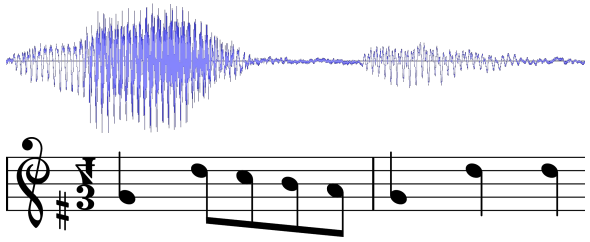
```
import matplotlib.pyplot as plt
images,labels = next(iter(testloader))
output = model(images)
_,pred = torch.max(output,1)
fig = plt.figure(figsize=(10,10))
for i in range(4):
  plt.subplot(5,5,i+1)
  show_img(images[i])
  plt.title(get_class(labels[i])+"/"+get_class(pred[i]))
  plt.xticks([])
  plt.yticks([])
```

Topic 6
Recurrent Neural Networks (RNN)

Sequential Data

Conventional NN and CNN are not able to model sequential data such as

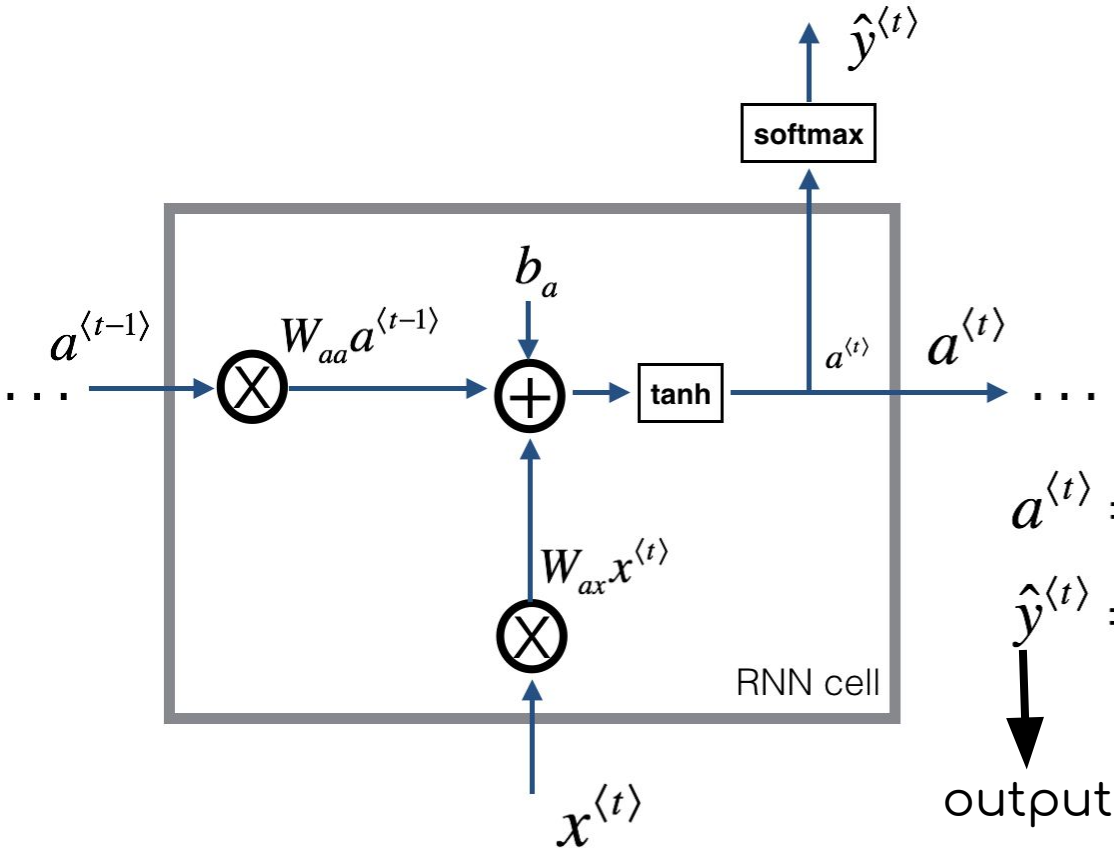
- Speech
- Music
- DNA sequence
- Video
- Machine translation
- Name entity detection



AGCCCCTGAGAATCTAG

What is RNN

A RNN cell can be seen as a simple one hidden layer NN (a) with input layer (a+x) and output layer (y)



input vector hidden vector

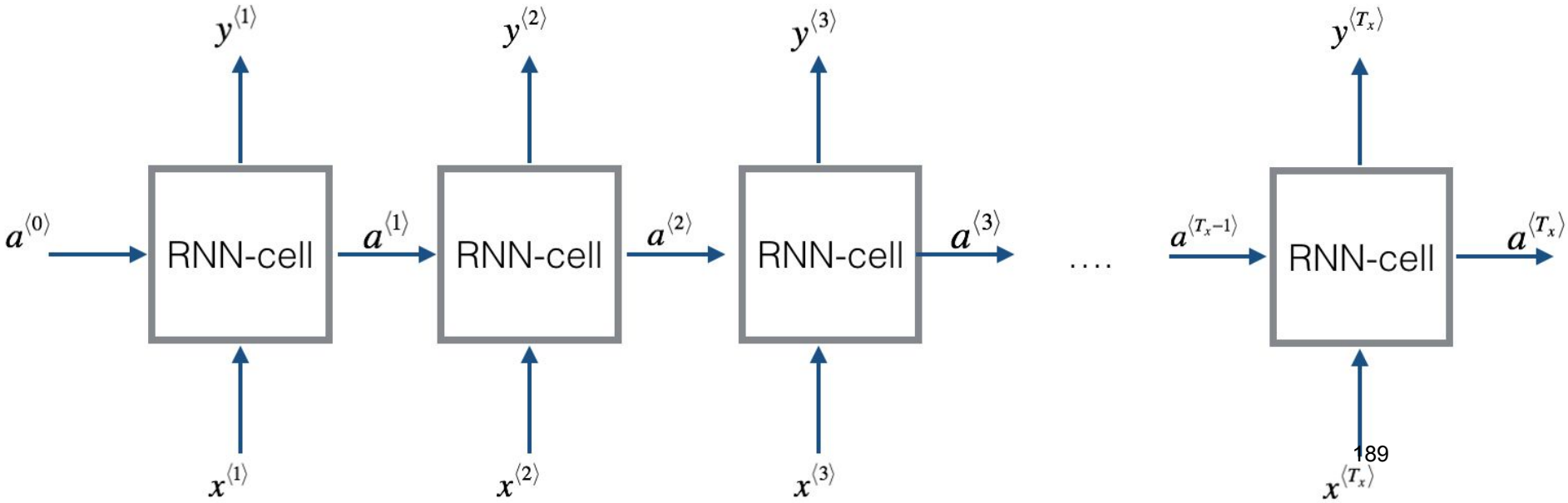
$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

output vector

Rollout RNN vs Time

- To input the sequential data to the RNN, RNN will roll out with time.
- RNN will read inputs x⟨t⟩ one at a time, and remember some info through the hidden layer activations that get passed from one time-step to the next.
- This allows a uni-directional RNN to take info from the past to process later inputs



Long Term Dependencies Issue

"Michel C. was born in Paris, France.....His mother tongue is ??????"

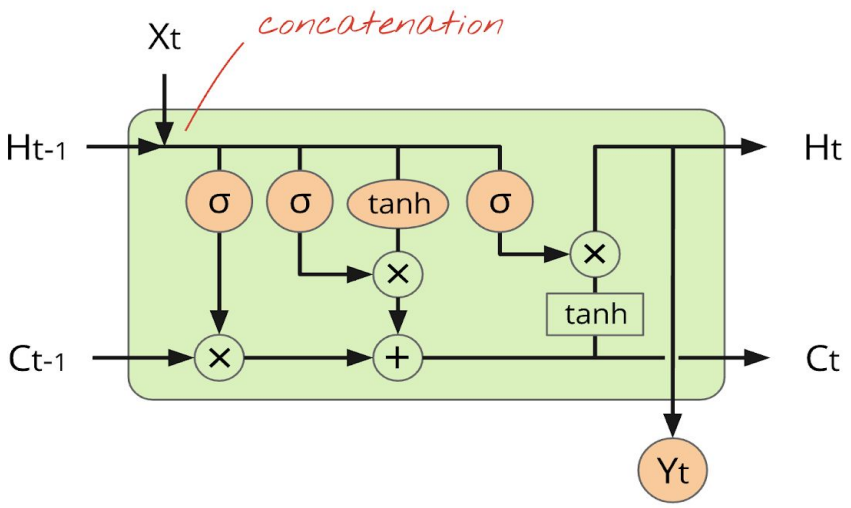
"The cat, which already ate.....was ...."

"The cats, which already ate .....were ...."

It is difficult to predict "French" or was/were if the sentences were very long due to vanishing gradients

LSTM Cell

- There are two internal states , C_t and H_t.
- C_t is the cell state (memory) to retain via forget and update gates
- H_t is the hidden state fed to the softmax via the result gate.

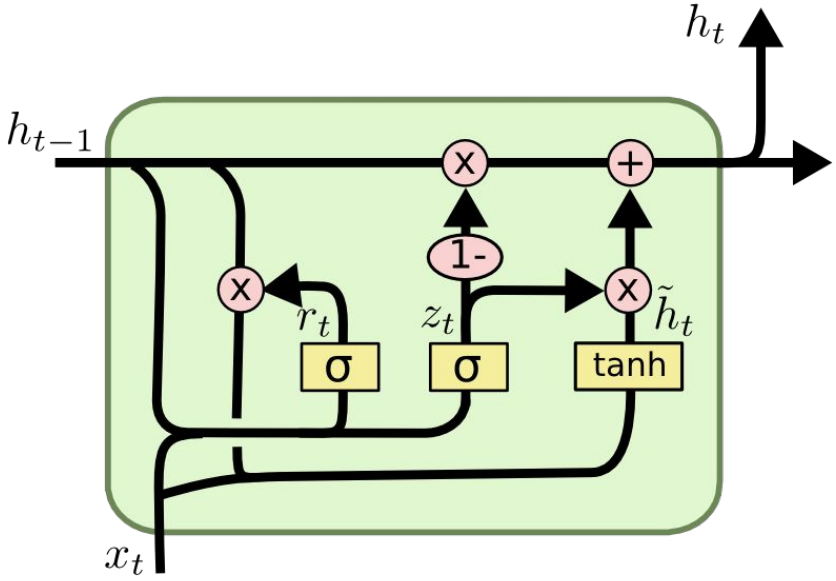


$$f = \sigma(X.W_f + b_f)$$
$$u = \sigma(X.W_u + b_u)$$
$$r = \sigma(X.W_r + b_r)$$
$$X' = \tanh(X.W_c + b_c)$$
$$C_t = f * C_{t-1} + u * X'$$
$$H_t = r * \tanh(C_t)$$

GRU Cell

GRU: Gated Recurrent Unit

Only two gates instead of 3 gates



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

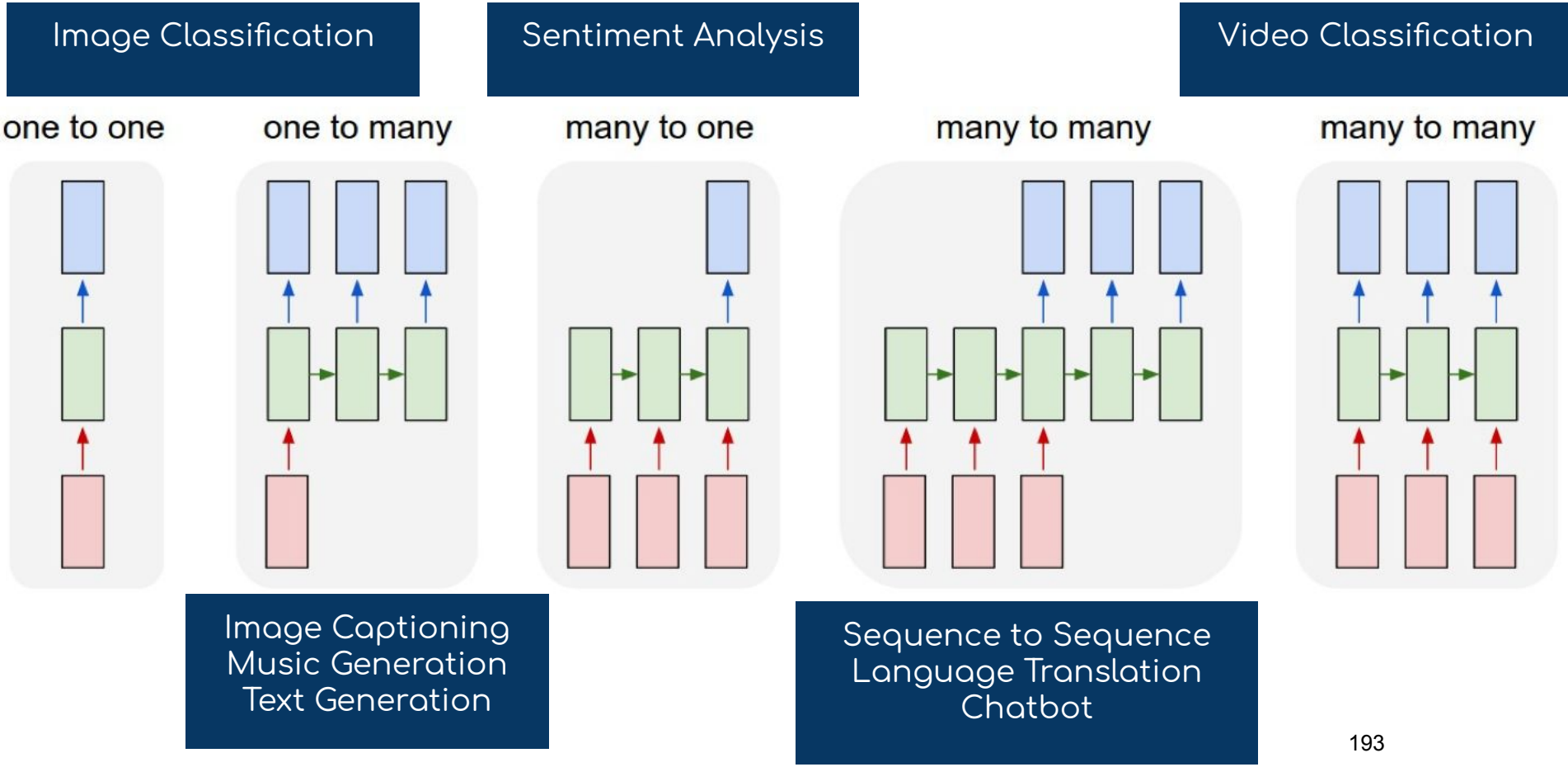
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

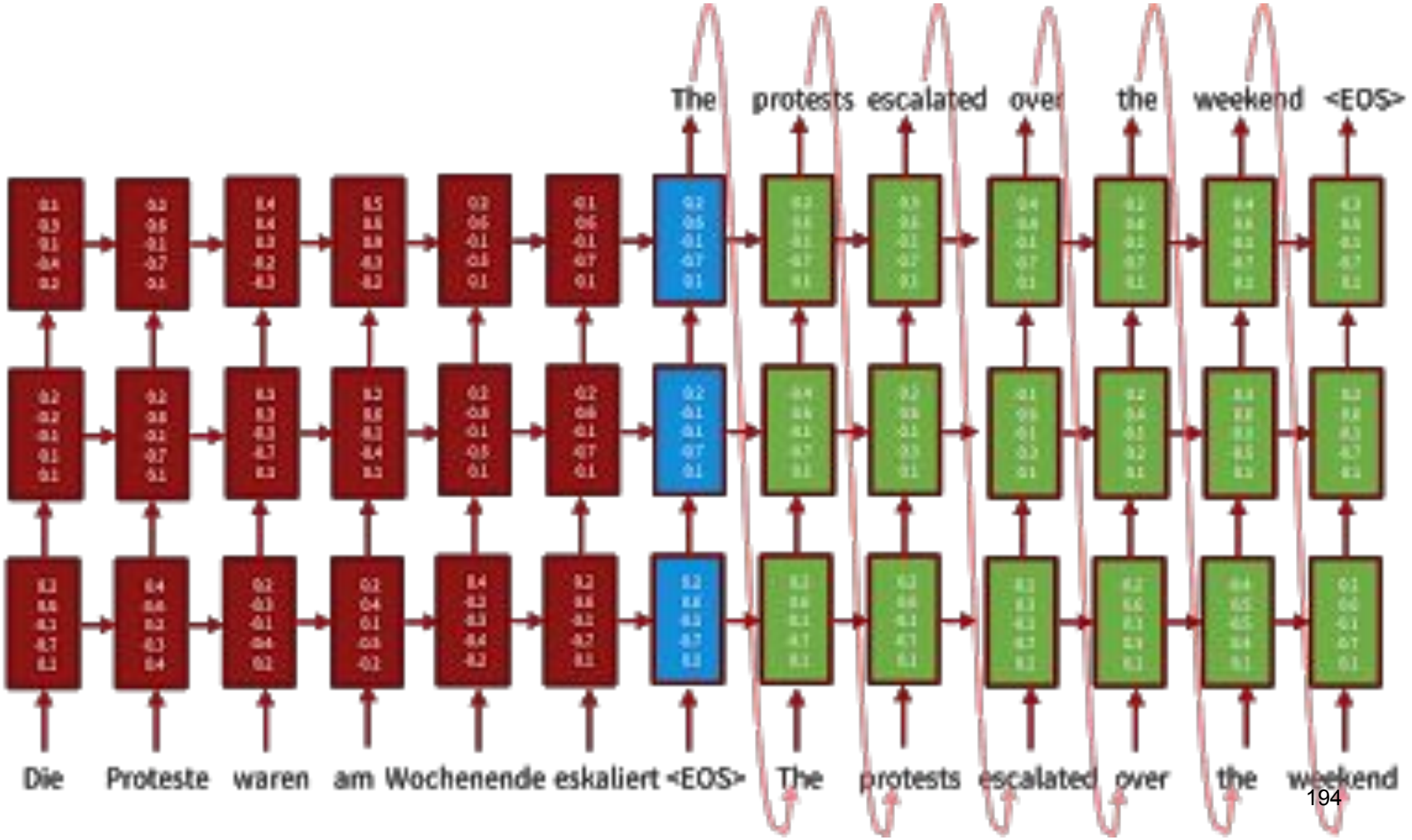
Types of RNN Architecture

RNN can have a variety of architecture



Sequence to Sequence RNN

Sequence to Sequence RNN have been used successfully for chatbot or machine translation..



Brief History of RNN

- 1982 - Hopfield Network
- 1997 - Hochreiter invented LSTM
- 2009 - Alex Graves demo cursive writing recognition with LSTM
- 2014 - Baidu demo speech recognition with LSTM
- 2016 - Google deployed seq2seq for machine translation

RNN Parameters

torch.nn.RNN(....) - output, h_n

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two RNNs together to form a *stacked RNN*, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1
- **nonlinearity** – The non-linearity to use. Can be either `'tanh'` or `'relu'`. Default: `'tanh'`
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each RNN layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional RNN. Default: `False`

LSTM Parameters

torch.nn.LSTM(....) - output, h_n,c_n

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two RNNs together to form a *stacked RNN*, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1
- **nonlinearity** – The non-linearity to use. Can be either `'tanh'` or `'relu'`. Default: `'tanh'`
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each RNN layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional RNN. Default: `False`

LSTM Input and Output Sizes

source: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

- **input**: tensor of shape (L, N, H_{in}) when `batch_first=False` or (N, L, H_{in}) when `batch_first=True` containing the features of the input sequence. The input can also be a packed variable length sequence. See `torch.nn.utils.rnn.pack_padded_sequence()` or `torch.nn.utils.rnn.pack_sequence()` for details.
- **h_0**: tensor of shape $(D * \text{num\_layers}, N, H_{out})$ containing the initial hidden state for each element in the batch. Defaults to zeros if (h_0, c_0) is not provided.
- **c_0**: tensor of shape $(D * \text{num\_layers}, N, H_{cell})$ containing the initial cell state for each element in the batch. Defaults to zeros if (h_0, c_0) is not provided.

where:

$$
\begin{aligned}
N &= \text{batch size} \\
L &= \text{sequence length} \\
D &= 2 \text{ if bidirectional=True otherwise } 1 \\
H_{in} &= \text{input\_size} \\
H_{cell} &= \text{hidden\_size} \\
H_{out} &= \text{proj\_size if proj\_size} > 0 \text{ otherwise hidden\_size}
\end{aligned}
$$

GRU Parameters

torch.nn.GRl(....) - output, h_n

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two GRUs together to form a *stacked GRU*, with the second GRU taking in outputs of the first GRU and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each GRU layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional GRU. Default: `False`

Parameters Description Examples

Consider the case of text classification:

- Each batch consist of a sequence of sentences
- Each sentence consist of a sequence of word tokens
- Each token is represented by an embedding vector

For example:

batch_size = 5 sentences or 100 days of stock price
sequence_length = 80 word tokens or 3 days of stock price
input_size = word embedding dimension or 1 for stock price

Batch-First (True)

```
import torch
import torch.nn as nn

number_layers = 1
batch_size = 1
input_size = 10
hidden_size = 20
sequence_length = 3

rnn = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
num_layers=number_layers, batch_first=True)
input = torch.randn(batch_size, sequence_length, input_size)
h0 = torch.randn(number_layers, batch_size, hidden_size)
c0 = torch.randn(number_layers, batch_size, hidden_size)
_,(hn, cn) = rnn(input, (h0, c0))
```

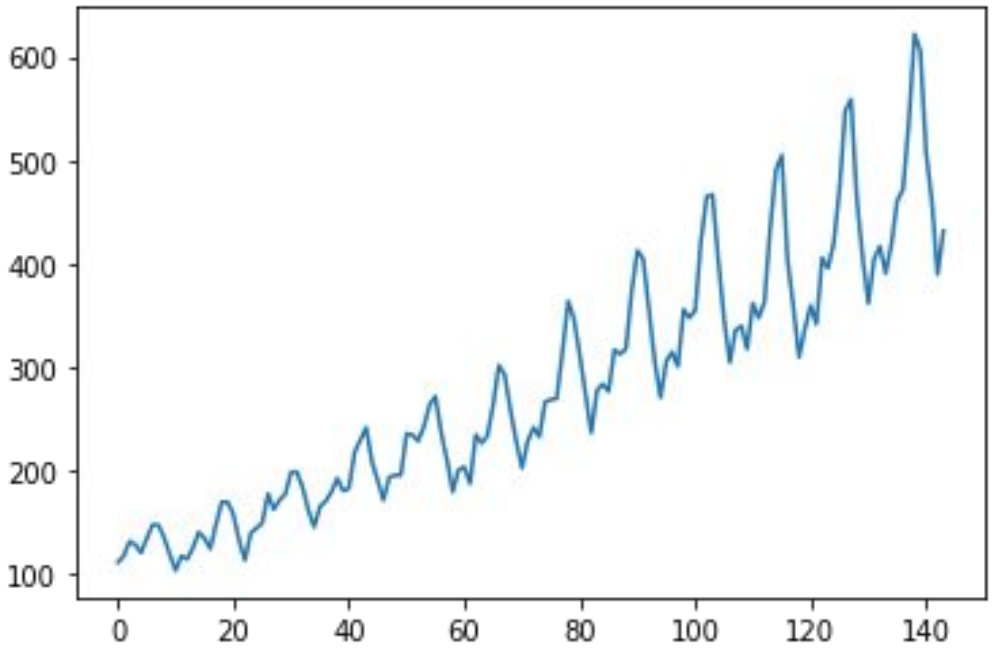
Batch First (False)

```
import torch
import torch.nn as nn

number_layers = 1
batch_size = 1
input_size = 10
hidden_size = 20
sequence_length = 3

rnn = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
num_layers=number_layers, batch_first=False)
input = torch.randn(sequence_length, batch_size, input_size)
h0 = torch.randn(number_layers, batch_size, hidden_size)
c0 = torch.randn(number_layers, batch_size, hidden_size)
_,(hn, cn) = rnn(input, (h0, c0))
```

Load the Data

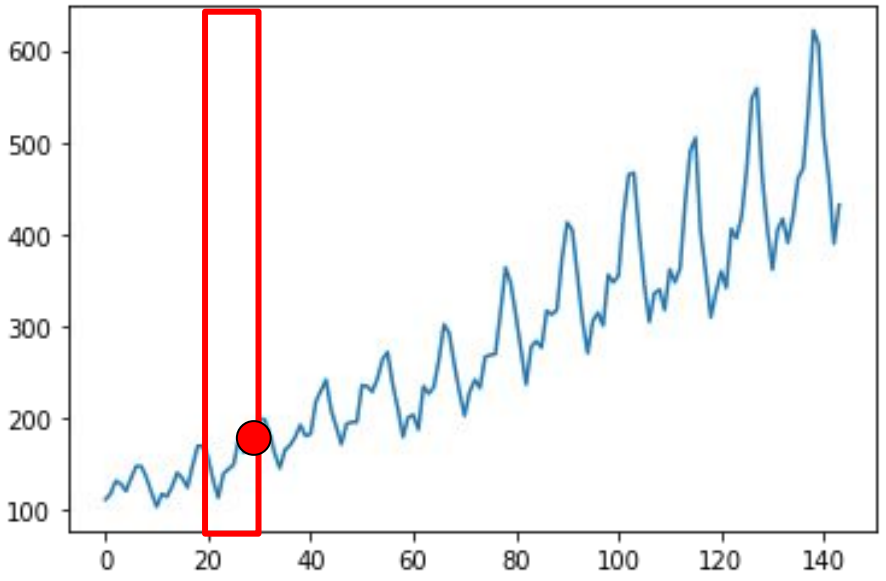
```
training_set = pd.read_csv('airline-passengers.csv')
training_set.head(10)
```

	Month	Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132
3	1949-04	129
4	1949-05	121
5	1949-06	135
6	1949-07	148
7	1949-08	148
8	1949-09	136
9	1949-10	119



Sliding Window

```
def sliding_window(data, seq_length):
    x = []
    y = []

    for i in range(len(data)-seq_length-1):
        _x = data[i:(i+seq_length)]
        _y = data[i+seq_length]
        x.append(_x)
        y.append(_y)

    return np.array(x),np.array(y)


seq_length = 4
x, y = scaling_window(training_data, seq_length)
```



Scale the Data

```
sc = MinMaxScaler()
training_data = sc.fit_transform(training_set)
```

Split the Data

```
train_size = int(len(y) * 0.67)
test_size = int(len(y)) - train_size

dataX = torch.Tensor(np.array(x))
dataY = torch.Tensor(np.array(y))

trainX = torch.Tensor(np.array(x[0:train_size]))
trainY = torch.Tensor(np.array(y[0:train_size]))

testX = torch.Tensor(np.array(x[train_size:len(x)]))
testY = torch.Tensor(np.array(y[train_size:len(y)]))
```

Define the LSTM Model

```
class LSTM(nn.Module):

    def __init__(self, num_classes, input_size, hidden_size, layer1, layer2):
        super(LSTM, self).__init__()

        self.num_classes = num_classes
        self.num_layers = num_layers
        self.input_size = input_size
        self.hidden_size = hidden_size

        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True)

        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h_0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)
        c_0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)

        ula, (h_out, _) = self.lstm(x, (h_0, c_0))
        h_out = h_out.view(-1, self.hidden_size)
        out = self.fc(h_out)

        return out
```

the batch size is x.size(0)

Loss Function and Optimizer

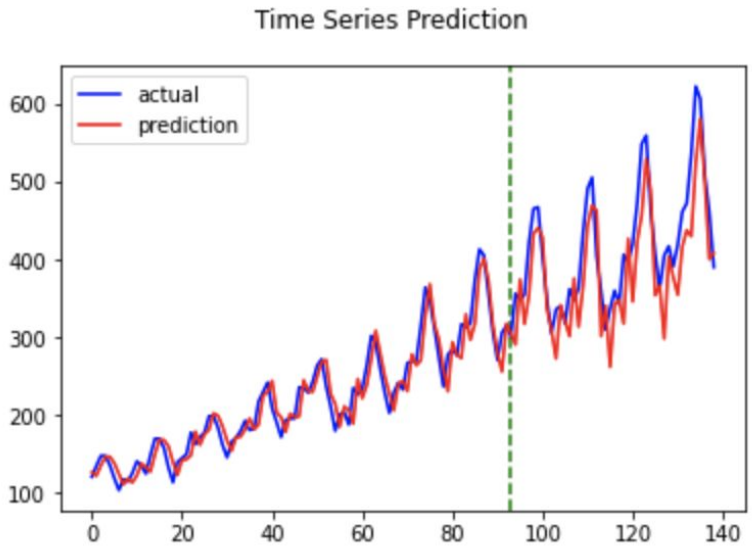
```
learning_rate = 0.01

criterion = torch.nn.MSELoss()
optimizer = torch.optim.Adam(lstm.parameters(),
lr=learning_rate)
```

Train the LSTM Model

```
num_epochs = 2000

for epoch in range(num_epochs):
    outputs = lstm(trainX)

    optimizer.zero_grad()
    loss = criterion(outputs, trainY)
    loss.backward()
    optimizer.step()

    if epoch % 100 == 0:
      print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))
```

Evaluate the LSTM Model

```
lstm.eval()
train_predict = lstm(dataX)

data_predict = train_predict.data.numpy()
dataY_plot = dataY.data.numpy()

data_predict = sc.inverse_transform(data_predict)
dataY_plot = sc.inverse_transform(dataY_plot)

plt.axvline(x=train_size, c='g', linestyle='--')

plt.plot(dataY_plot,'b',label='actual')
plt.plot(data_predict,'r',label='prediction')
plt.suptitle('Time Series Prediction')
plt.legend()
plt.show()
```



Define the GRU Model

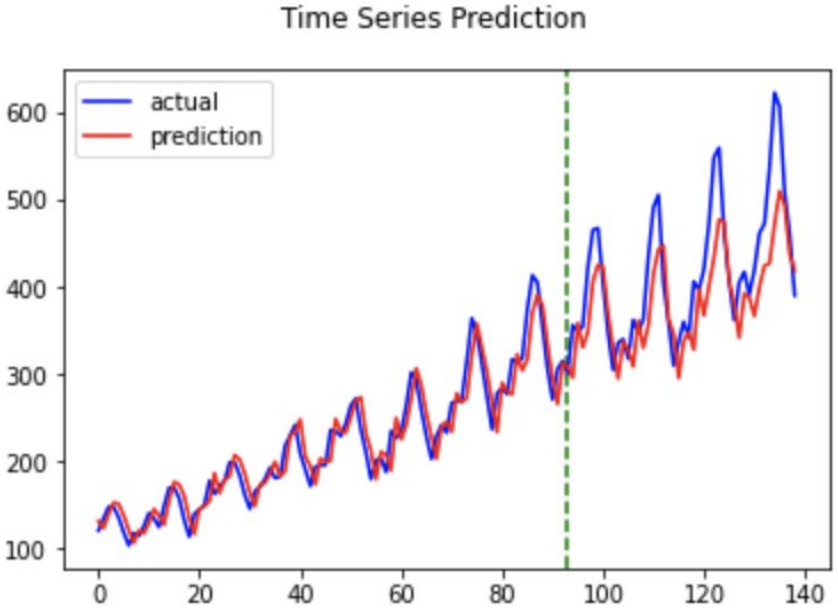
```
class GRU(nn.Module):

    def __init__(self, num_classes, input_size, hidden_size, num_layers):
        super(GRU, self).__init__()

        self.num_classes = num_classes
        self.num_layers = num_layers
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.seq_length = seq_length

        self.gru = nn.GRU(input_size=input_size, hidden_size=hidden_size,
                          num_layers=num_layers, batch_first=True)

        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h_0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)

        # Propagate input through LSTM
        _, h_out = self.gru(x, h_0)
        h_out = h_out.view(-1, self.hidden_size)
        out = self.fc(h_out)

        return out
```

Define the RNN Model

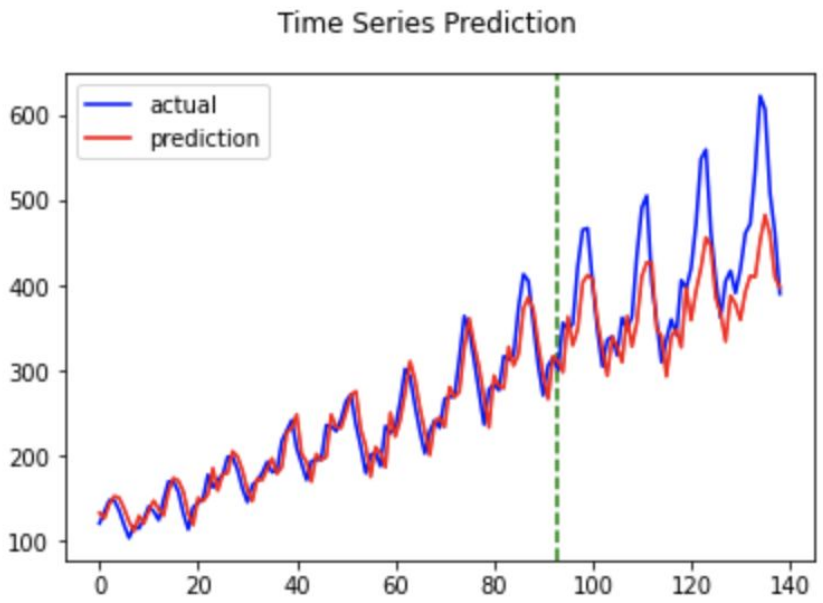
```
class RNN(nn.Module):

    def __init__(self, num_classes, input_size, hidden_size, num_layers):
        super(RNN, self).__init__()

        self.num_classes = num_classes
        self.num_layers = num_layers
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.seq_length = seq_length

        self.rnn = nn.RNN(input_size=input_size, hidden_size=hidden_size,
                          num_layers=num_layers, batch_first=True)

        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h_0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size)

        # Propagate input through LSTM
        _, h_out = self.rnn(x, h_0)
        h_out = h_out.view(-1, self.hidden_size)
        out = self.fc(h_out)

        return out
```

Comparison LSTM, GRU & RNN

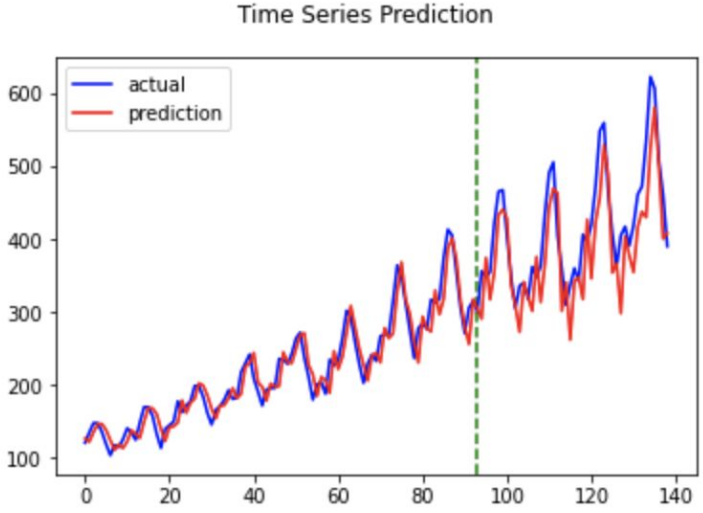
GRU



RNN



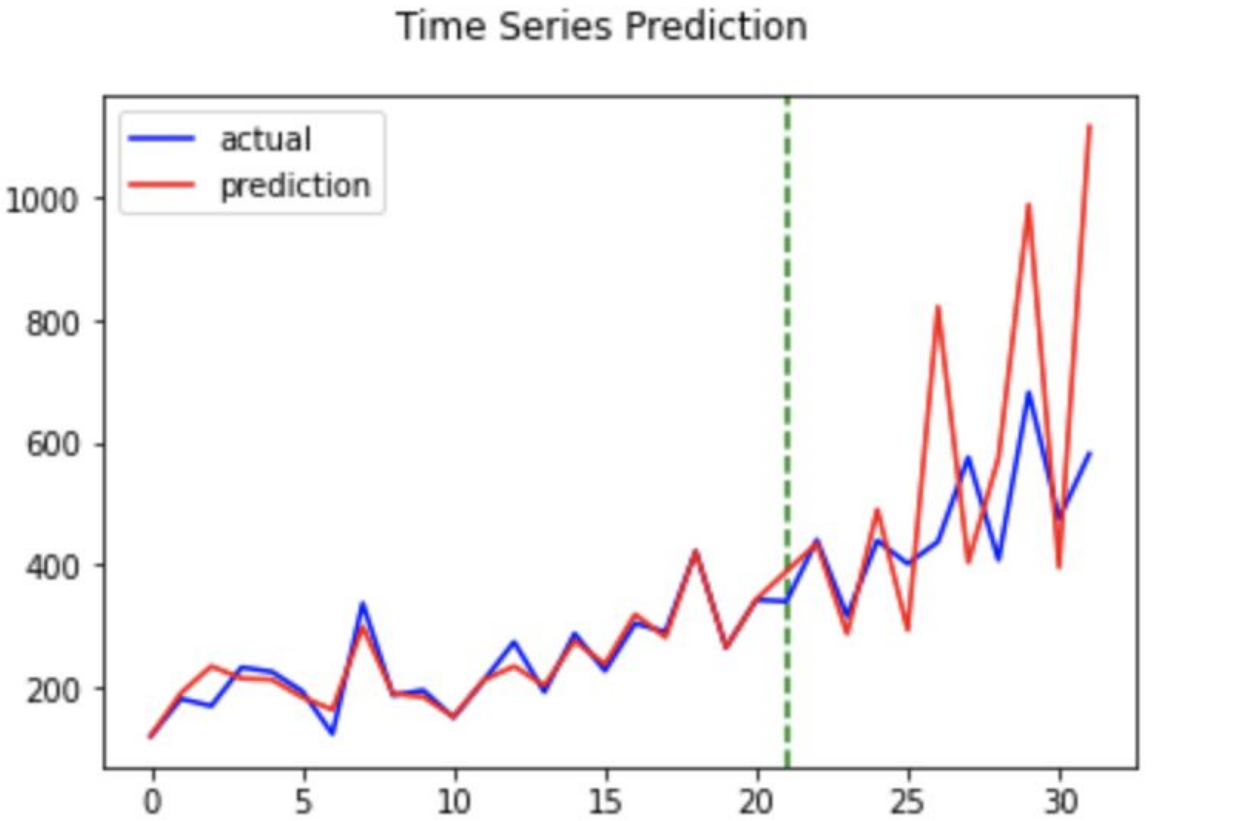
LSTM



- LTSM shows slightly better result than GRU and RNN.

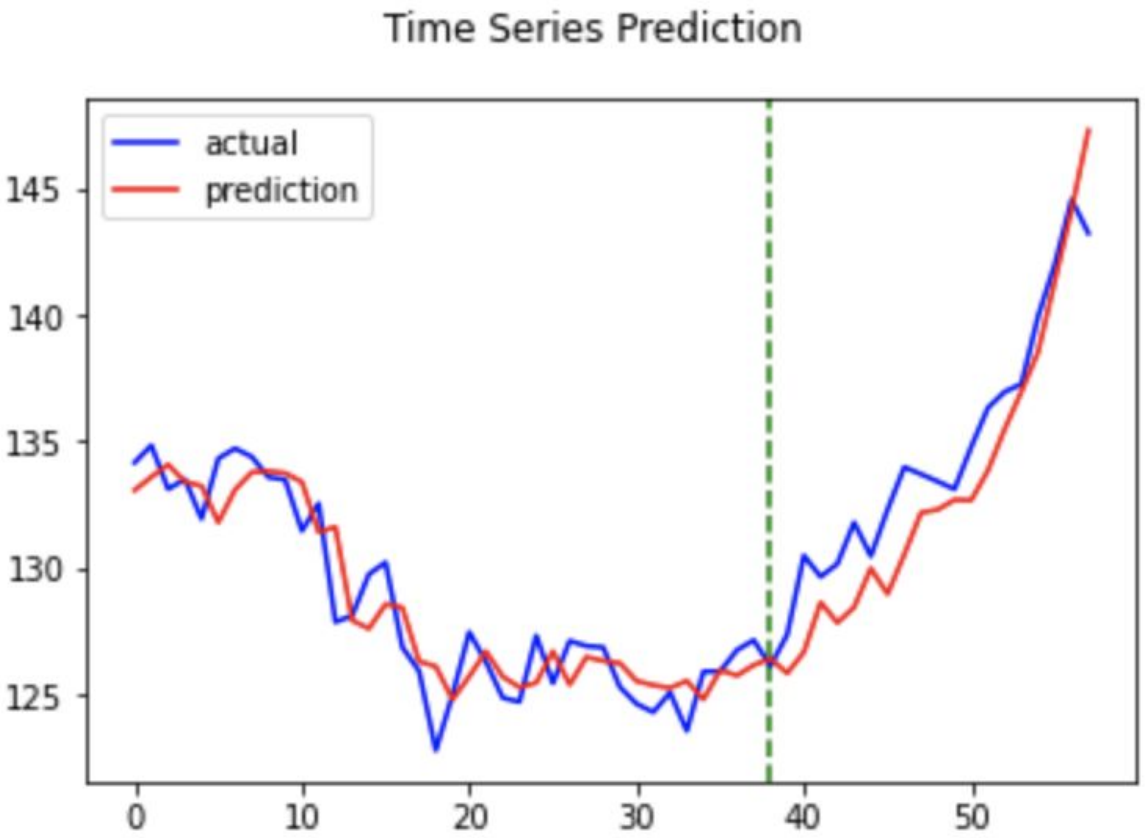
Activity: Sales Prediction

- Load the shampoo.csv dataset
- Create a LSTM model to predict the sales based on historical data.



Activity: Stock Price Prediction

- Goto Yahoo Finance to download at least 3 months of a stock price for a particular stock
- Create a LSTM model to predict the stock price based on historical data.



Summary Q&A

Course Feedback

https://goo.gl/R2eumq

TINFOTECH

CERTIFICATE

of ACCOMPLISHMENT

You will receive a digital certificate in your email
after the completion of the class
If you did not receive the digital certificate,
please send your request to
enquiry@tertiaryinfotech.com

Thank You!

Dr. Alfred Ang
Tel: 96983731
Email: angch@tertiaryinfotech.com
Facebook:
https://www.facebook.com/angchewhoe
Linkedin:
https://www.linkedin.com/in/angchewhoe/