Building Neural Networks for Deep Learning Application

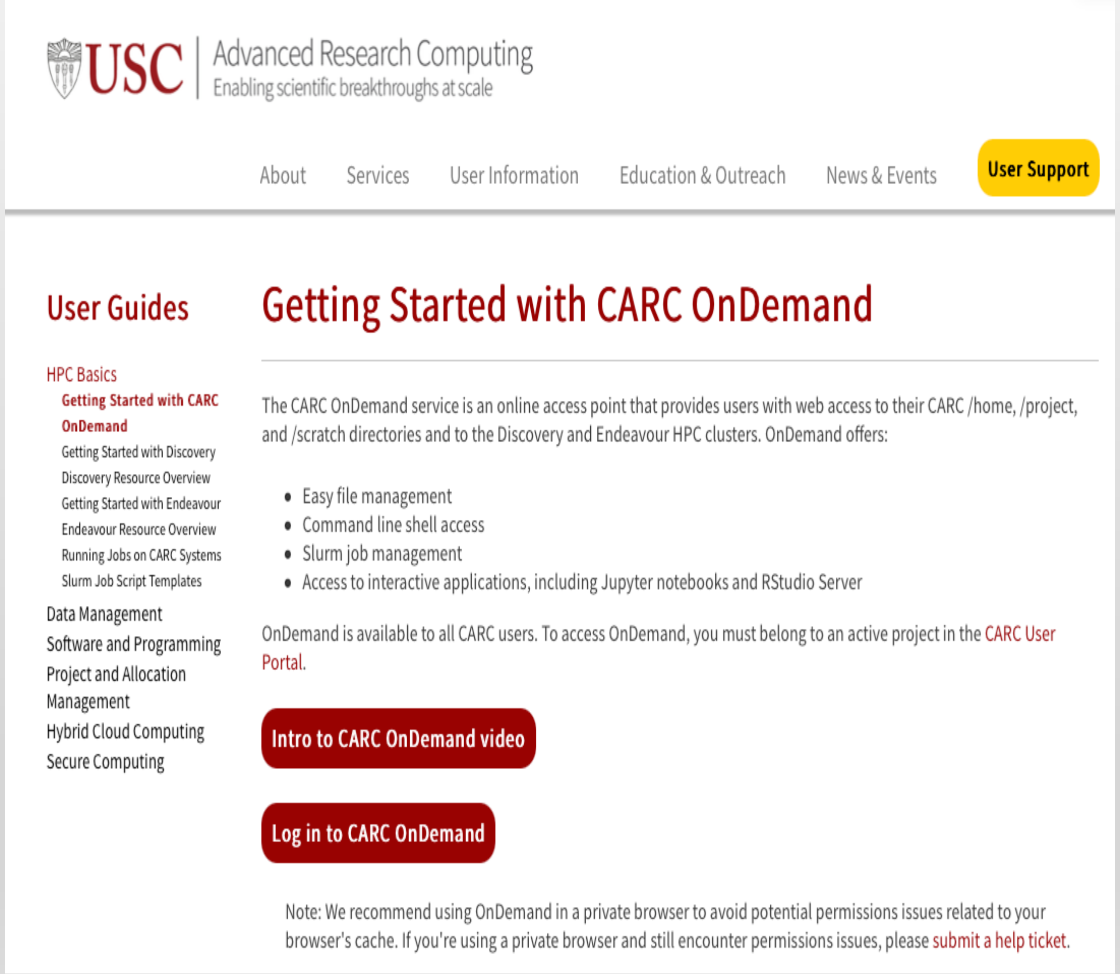
Hao Ji, Data Scientist

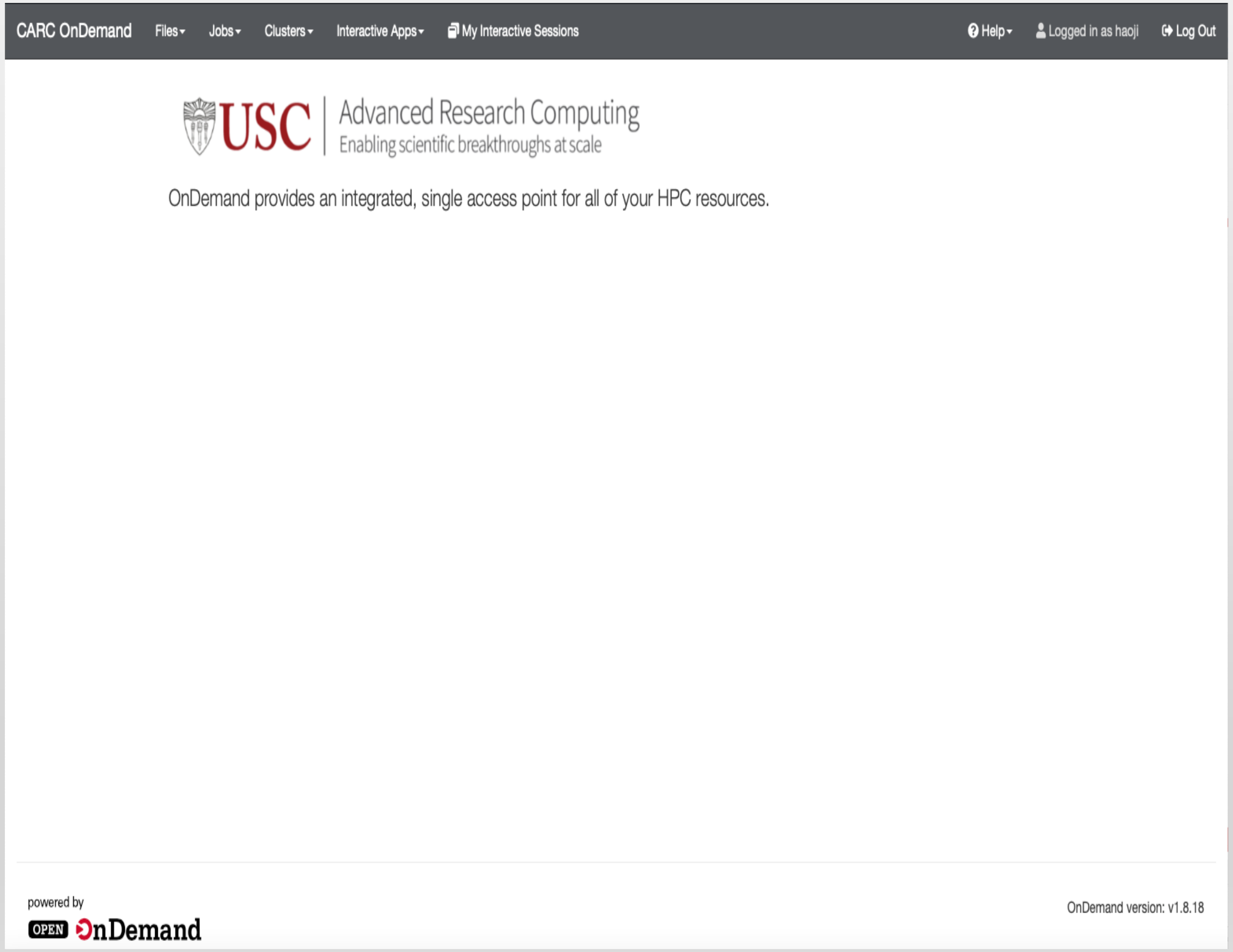
USC Center for Advanced Research Computing

Content

CARC OnDemand

Web Address: https://carc-ondemand.usc.edu





Using Anaconda on CARC

Anaconda: package and environment manager primarily used for open-source data science packages for the Python and R programming languages.

Using Anaconda

Anaconda is a package and environment manager primarily used for open-source data science packages for the Python and R programming languages. It also supports other programming languages like C, C++, FORTRAN, Java, Scala, Ruby, and Lua.

Using Anaconda on CARC systems

Begin by logging in. You can find instructions for this in the Getting Started with Discovery or Getting Started with Endeavour user guides.

To use Anaconda, first load the corresponding module:

```
module purge
module load conda
```

This module is based on the minimal Miniconda installer. Included in all versions of Anaconda, Conda is the package and environment manager that installs, runs, and updates packages and their dependencies. This module also provides Mamba, which is a drop-in replacement for most `conda` commands that enables faster package solving, downloading, and installing.

The next step is to initialize your shell to use Conda and Mamba:

```
mamba init bash
source ~/.bashrc
```

This modifies your `~/.bashrc` file so that Conda and Mamba are ready to use every time you log in (without needing to load the module).

If you want a newer version of Conda or Mamba than what is available in the module, you can also install them into one of your directories. We recommend installing either mambaforge or Miniconda.

Conda can also be configured with various options. Read more about Conda configuration here.

Installing Conda environments and packages

https://www.carc.usc.edu/user-information/user-guides/software-and-programming/anaconda

Using Anaconda on CARC

https://pytorch.org

INSTALL PYTORCH

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, 1.12 builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

Additional support or warranty for some PyTorch Stable and LTS binaries are available through the PyTorch Enterprise Support Program.

PyTorch Build	Stable (1.11.0)	Preview (Nightly)	LTS (1.8.2)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python	C++ / Java		
Compute Platform	CUDA 10.2	CUDA 11.3	ROCm 4.2 (beta)	CPU

Run this Command:

```
conda install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch
```

Creating Jupyter Kernel

A **Jupyter kernel** is a programming language-specific process that executes the code contained in a Jupyter notebook.

Installing Jupyter Kernels

This user guide provides instructions for installing Jupyter kernels when using CARC OnDemand. For more information about OnDemand and using Jupyter notebooks, see the Getting Started with CARC OnDemand user guide.

A **Jupyter kernel** is a programming language-specific process that executes the code contained in a Jupyter notebook. The following provides installation instructions for a few popular Jupyter kernels, which will be installed in your home directory at `~/.local/share/jupyter/kernels`. Install the kernels when logged in to CARC systems before accessing them via the Jupyter OnDemand interactive app. To learn more about installing software on CARC systems using the software module system, see the Software Module System user guide.

When installing kernels, make sure to use descriptive names in order to distinguish among them. Once installed, when launching Jupyter on OnDemand, the kernels will show up on a Launcher tab (File > New Launcher) and when selecting kernels through other methods.

Many software kernels are available for use with Jupyter. See a full list here: https://github.com/jupyter/jupyter/wiki/Jupyter-kernels.

Python

The default kernel is for Python 3.9.2, and this is ready to be used when Jupyter is launched. To use other versions of Python, enter a set of commands like the following:

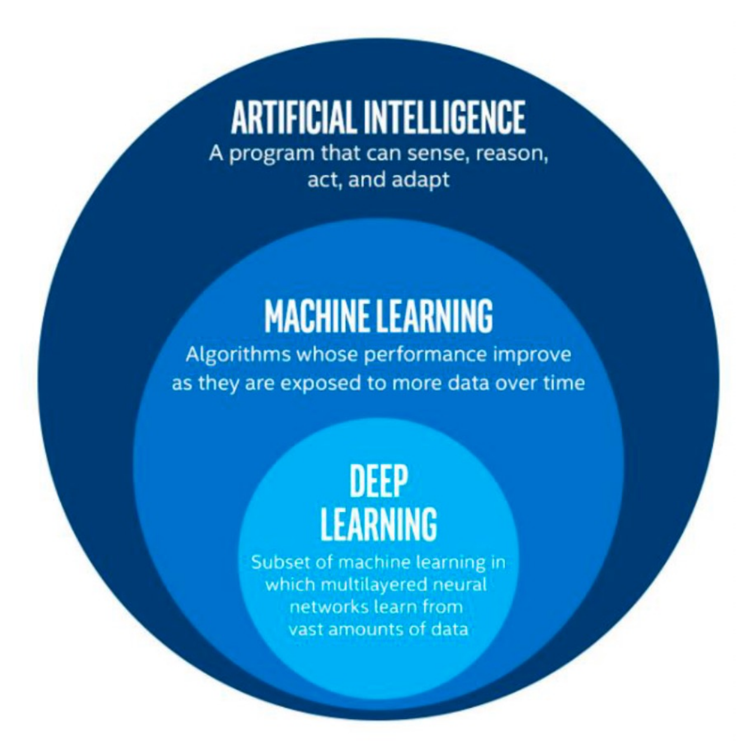
```
module load usc python/<version>
python -m ipykernel install --user --name py376 --display-name "Python 3.7.6"
```

https://www.carc.usc.edu/user-information/user-guides/software-and-programming/jupyter-kernels

Introduction to Deep Learning

Deep Learning: subfield of traditional machine learning

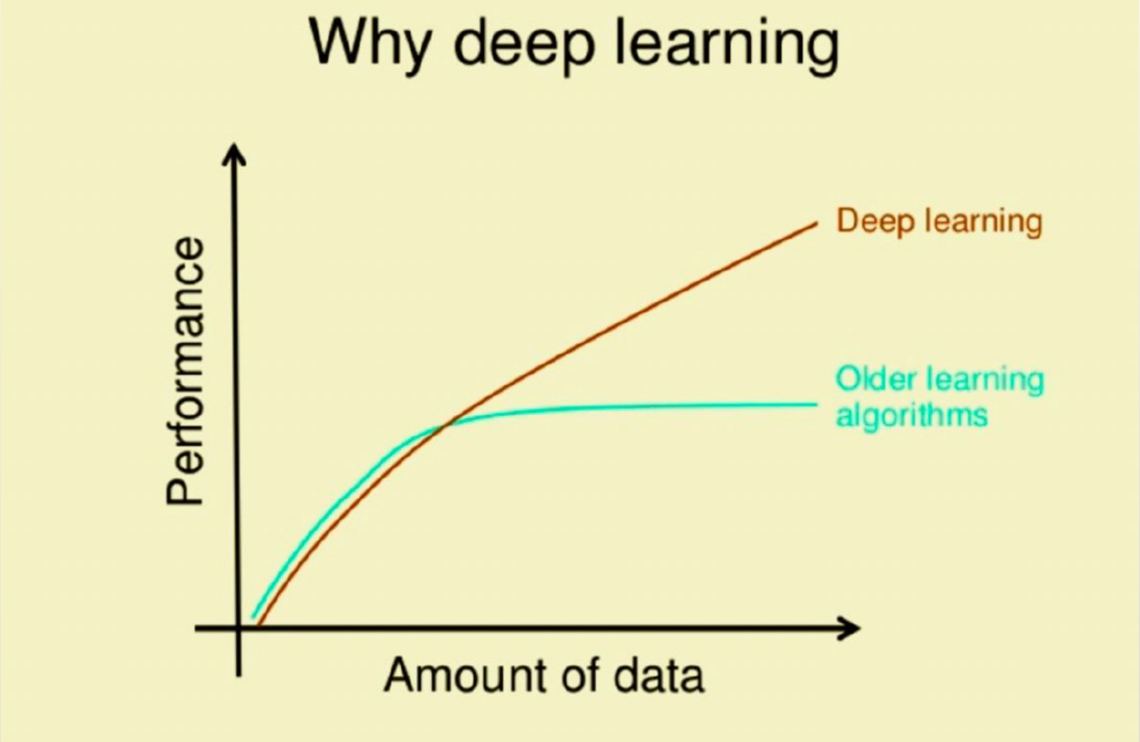
- Inspired by the structure and function of the brain: Artificial Neural Networks
- Computer vision: Tesla recognizing items on a street
- Text Generation: An algorithm trained to create a new Shakespeare piece
- Speech recognition
- Computer Games: AlphaGo



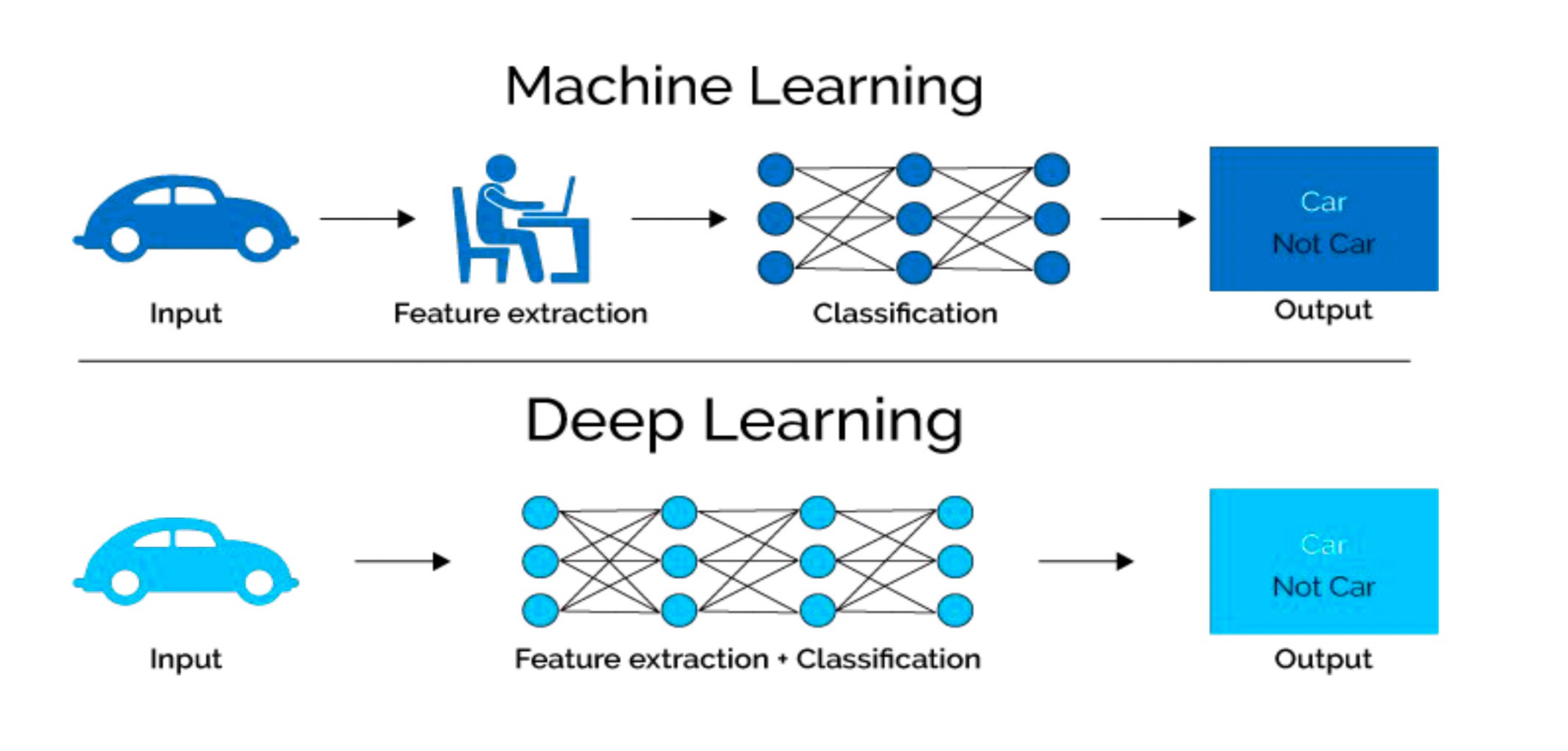
Introduction to Deep Learning

Why Deep Learning so hot recently?

- Performs better with larger amounts of data
- Requires strong computation units such as GPU’s
- Data Storage

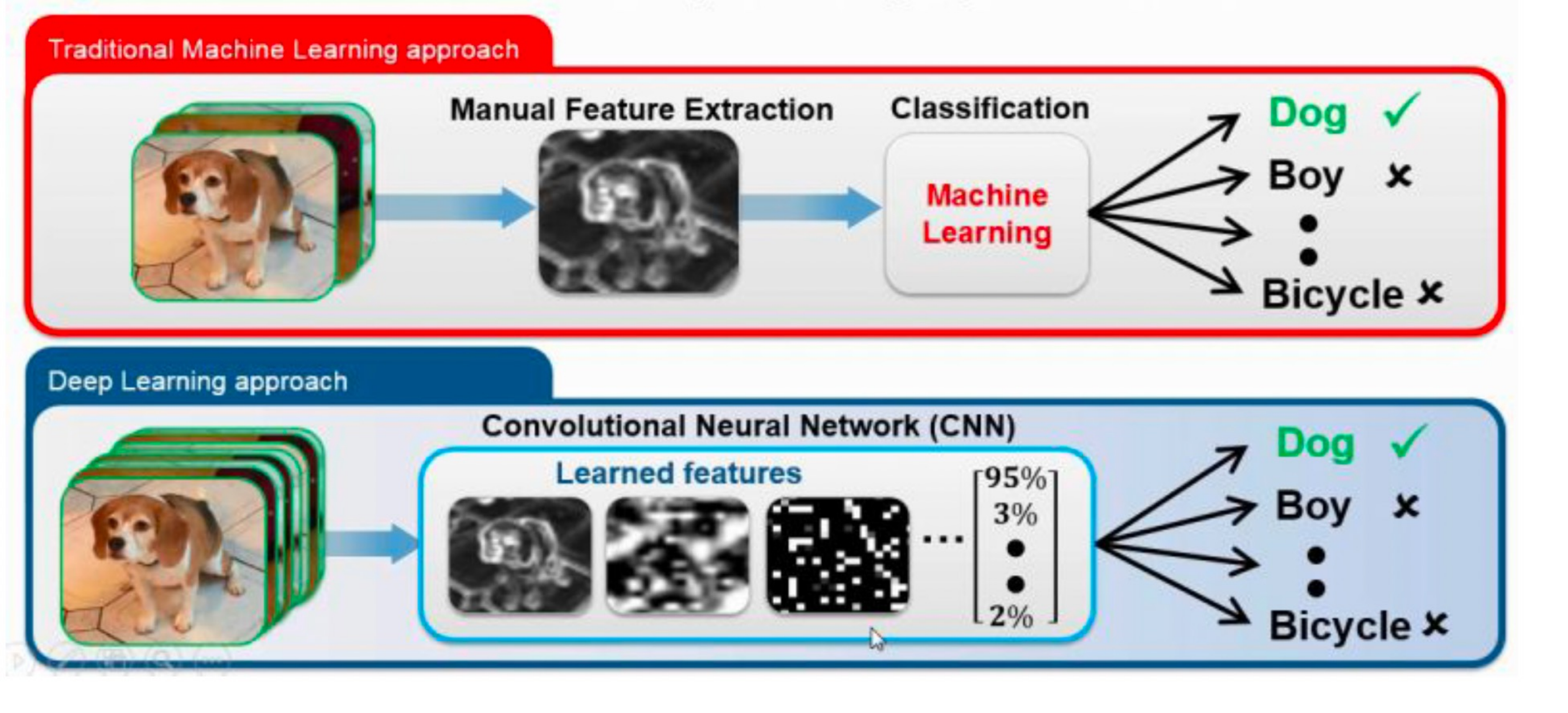


Introduction to Deep Learning

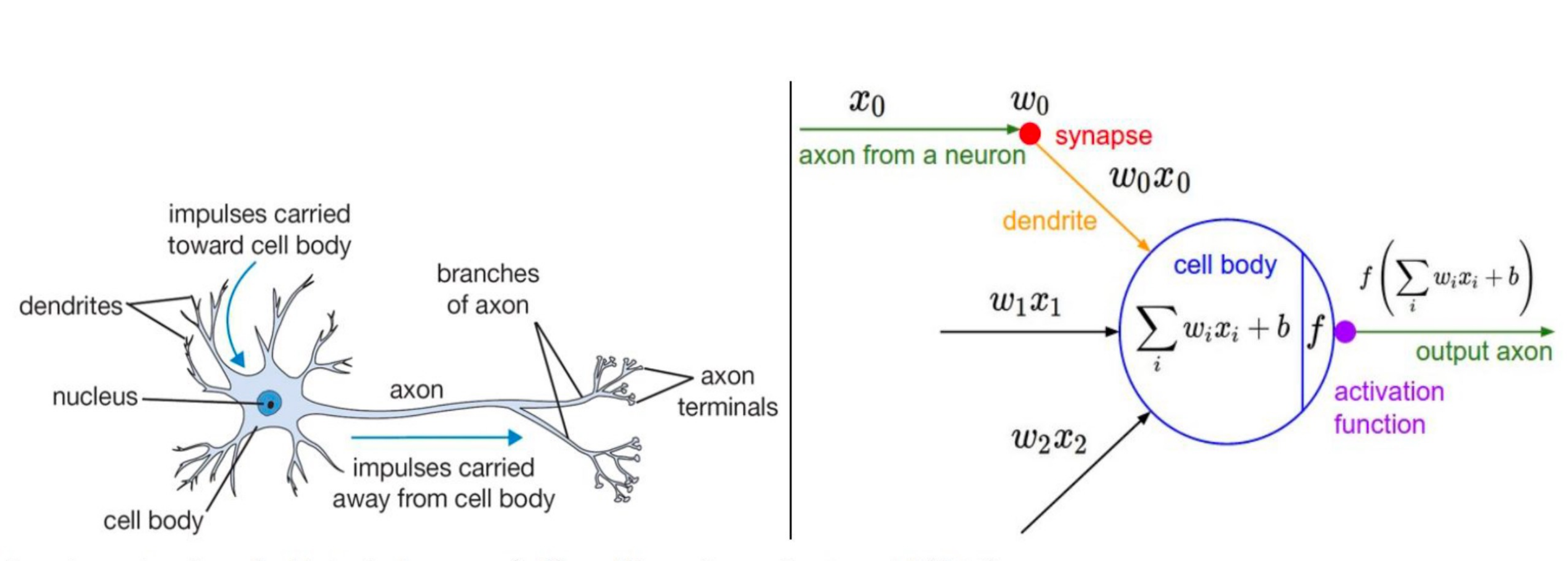


Introduction to Deep Learning

Deep Learning is a machine learning technique that can learn useful representations or features directly from images, text and sound



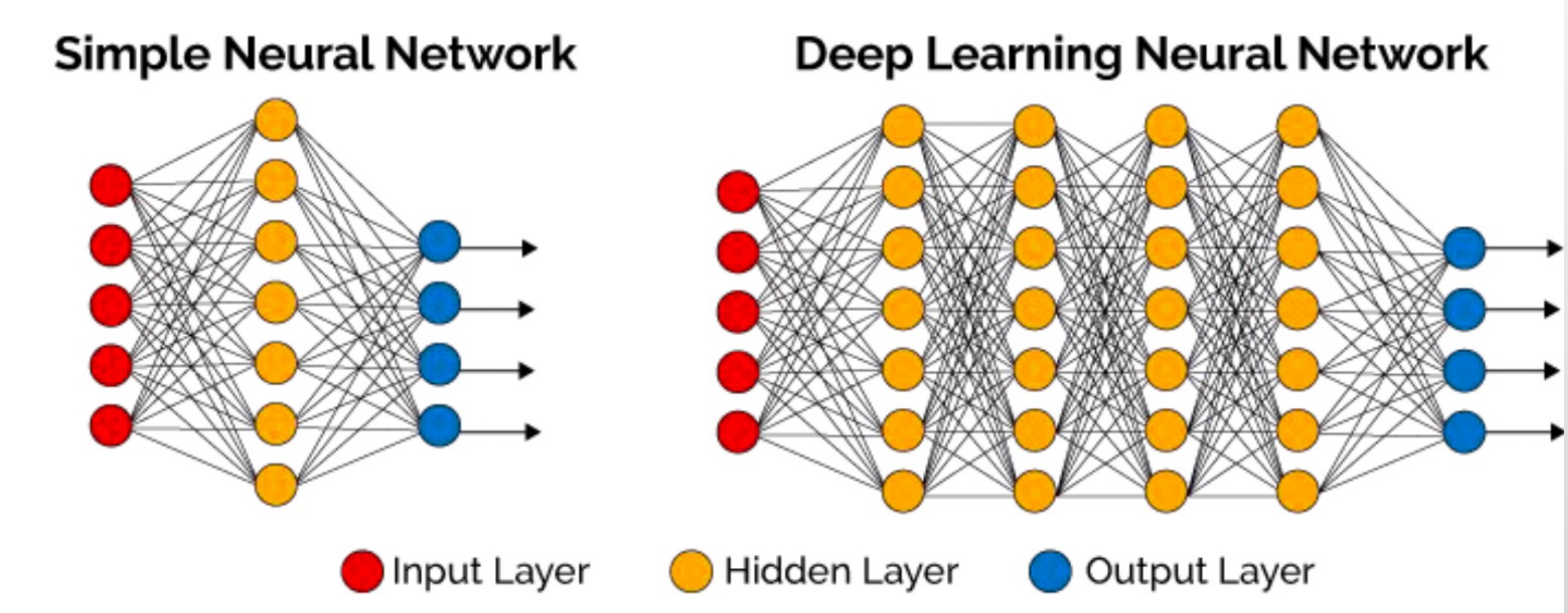
Neural Networks



biological neuron

artificial neural networks

Neural Networks



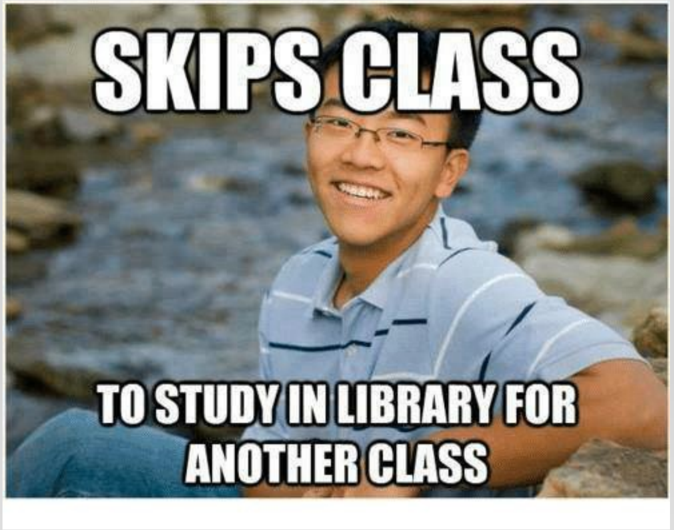
A neural network (NN) has 3 types of layers:
Input layer Hidden layer Output layer

Deep neural networks (DNN) usually has more hidden layers
Still has same 3 types of layers

Building Neural Networks

Task: Predict if it worth going to class or not

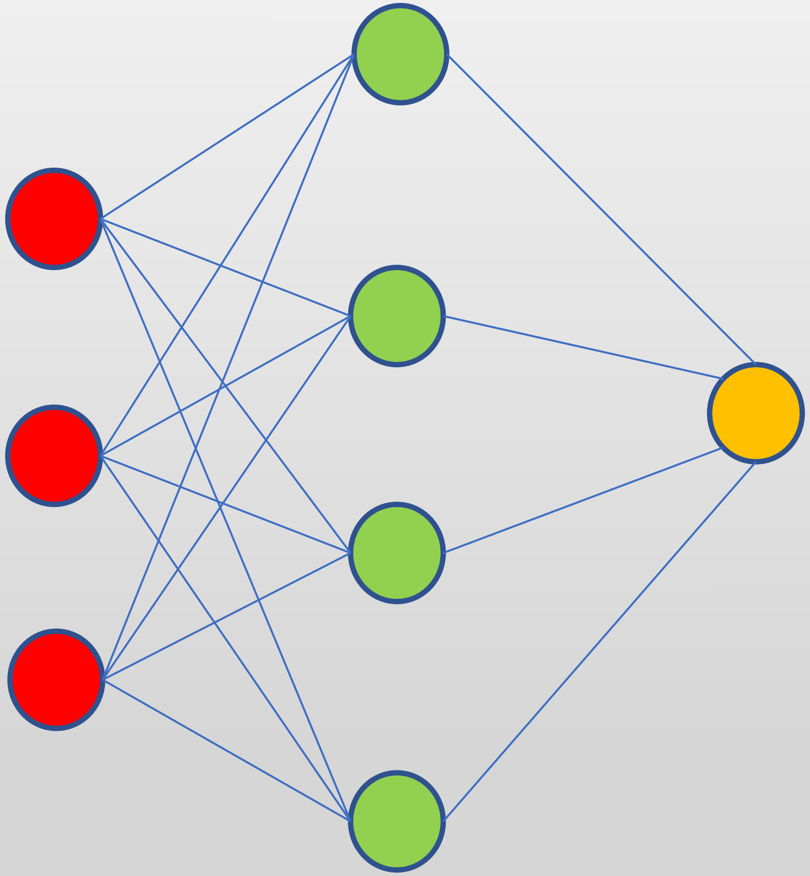
$$X = \begin{bmatrix} difficulty\ of\ material \\ number\ of\ assignments\ due \\ days\ until\ midterm \end{bmatrix} \quad Y = [probability\ of\ going\ to\ class]$$



Building Neural Networks

Task: Predict if it worth going to class or not

$$X = \begin{bmatrix} difficulty\ of\ material \\ number\ of\ assignments\ due \\ days\ until\ midterm \end{bmatrix}$$



$$Y = [probability\ of\ going\ to\ class]$$

Neural Networks

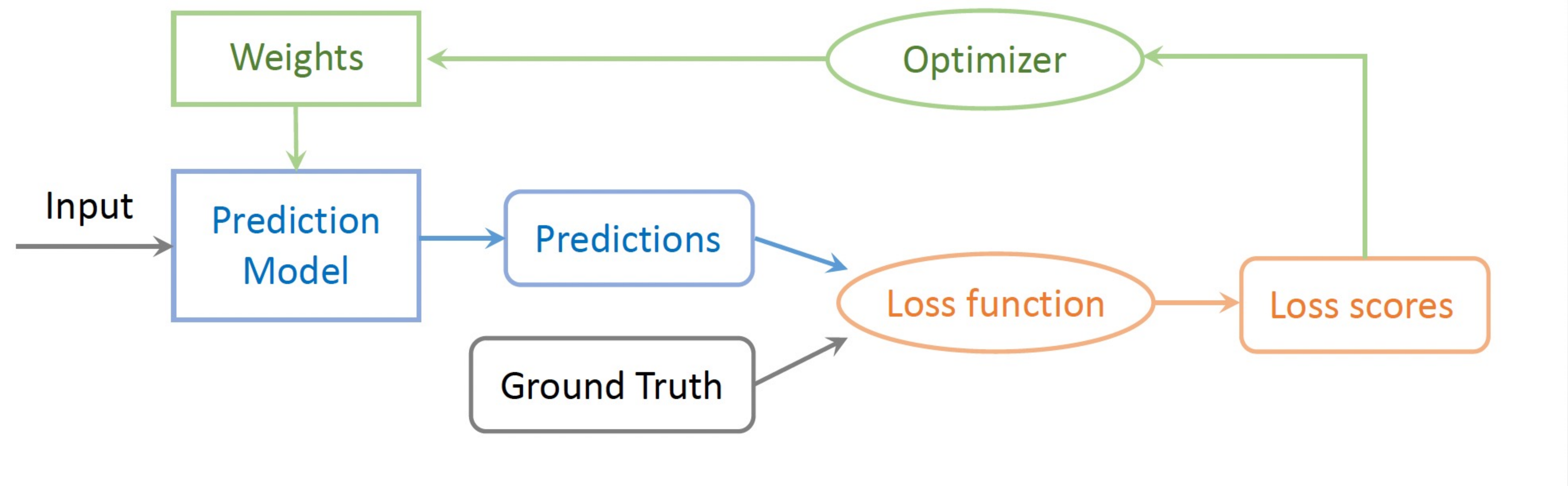
Three steps to training a neural network

1 Forward propagation: push example through the network to get a predicted output

2 Compute the cost: calculate the difference between predicted output and actual data

3 Backward propagation: push back the derivative of the error and apply to each weight, such that next time it will result in a lower error

Training Pipeline

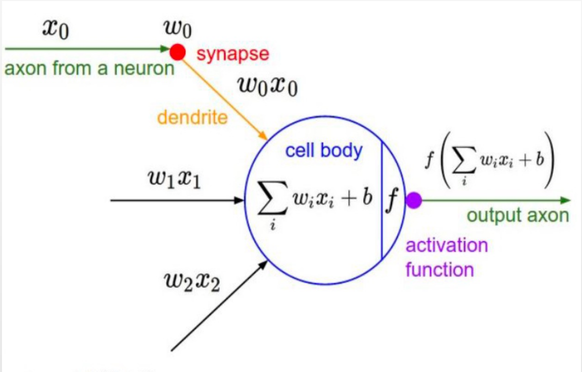


The training pipeline consists of choosing the prediction model, the loss function and the optimizer.

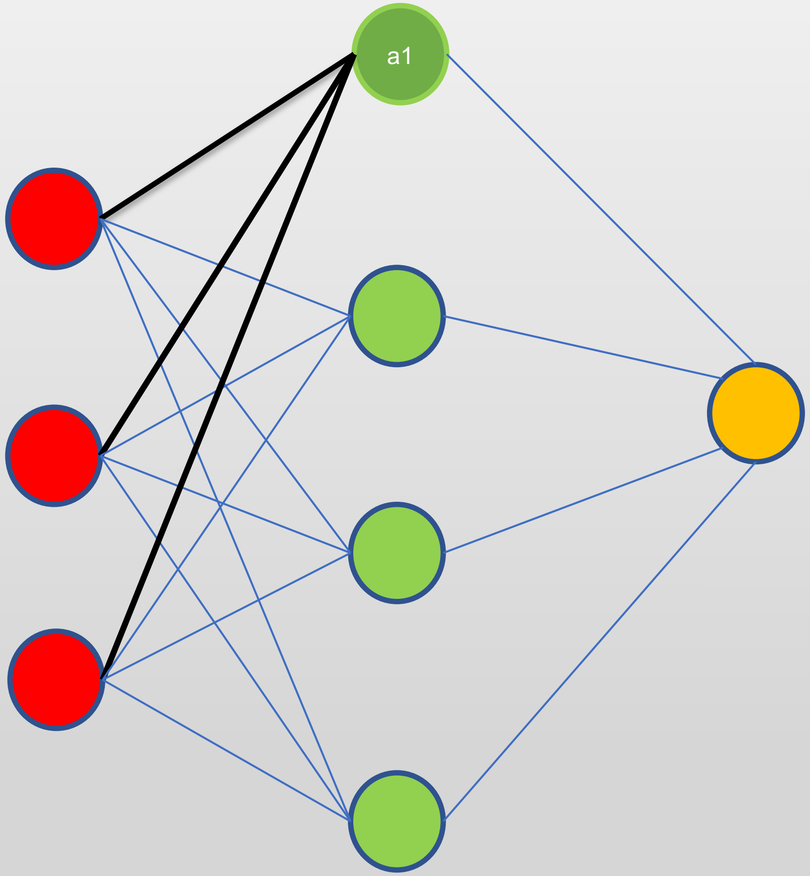
Once these choices are made, we can feed the input data and labels to start the training process.

Building Neural Networks

Task: Predict if it worth going to class or not



$$X = \begin{bmatrix} difficulty\ of\ material \\ number\ of\ assignments\ due \\ days\ until\ midterm \end{bmatrix}$$



$$Y = [probability\ of\ going\ to\ class]$$

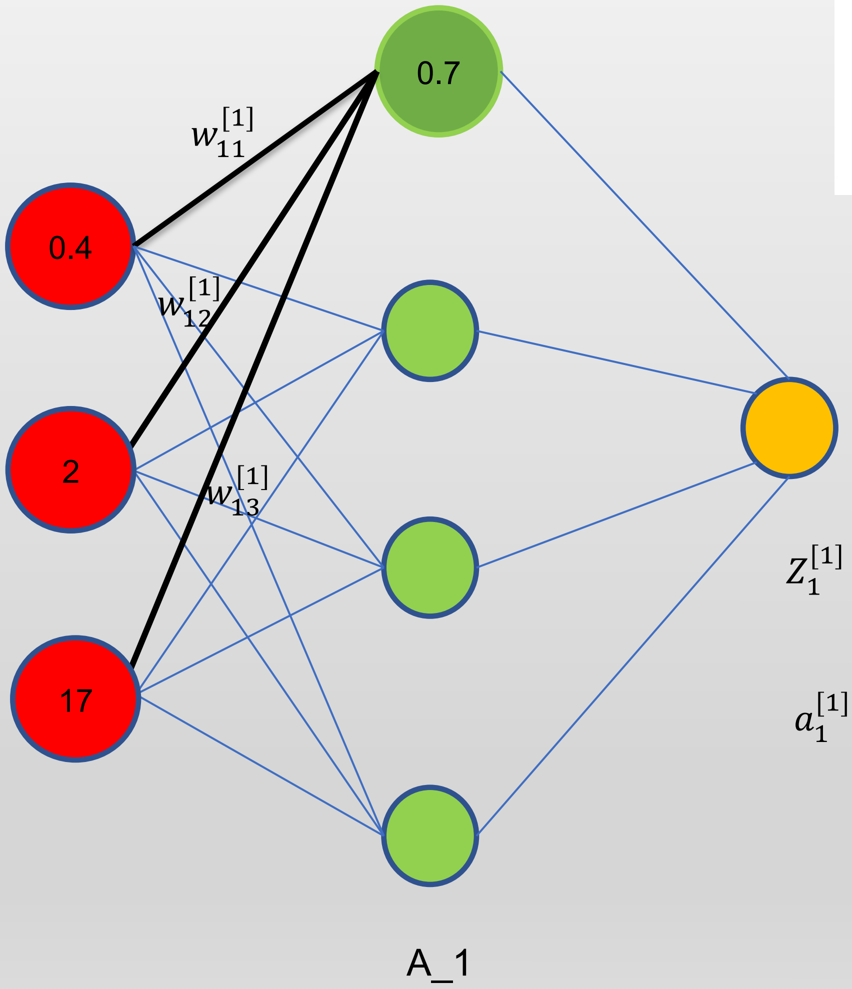
```
if y > 0.5:
    goto_class()
else:
    skip()
```

A_1

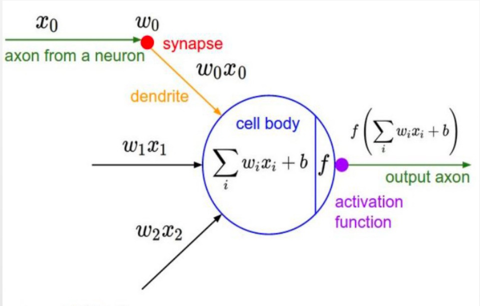
Building Neural Networks

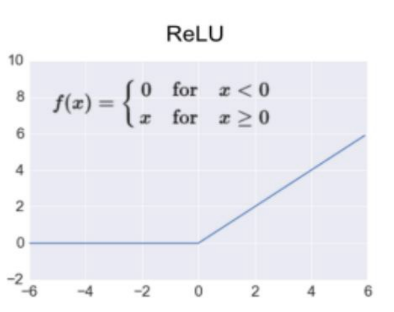
Task: Predict if it worth going to class or not

$$X = \begin{bmatrix} difficulty\ of\ material \\ number\ of\ assignments\ due \\ days\ until\ midterm \end{bmatrix}$$



$$Y = [probability\ of\ going\ to\ class]$$





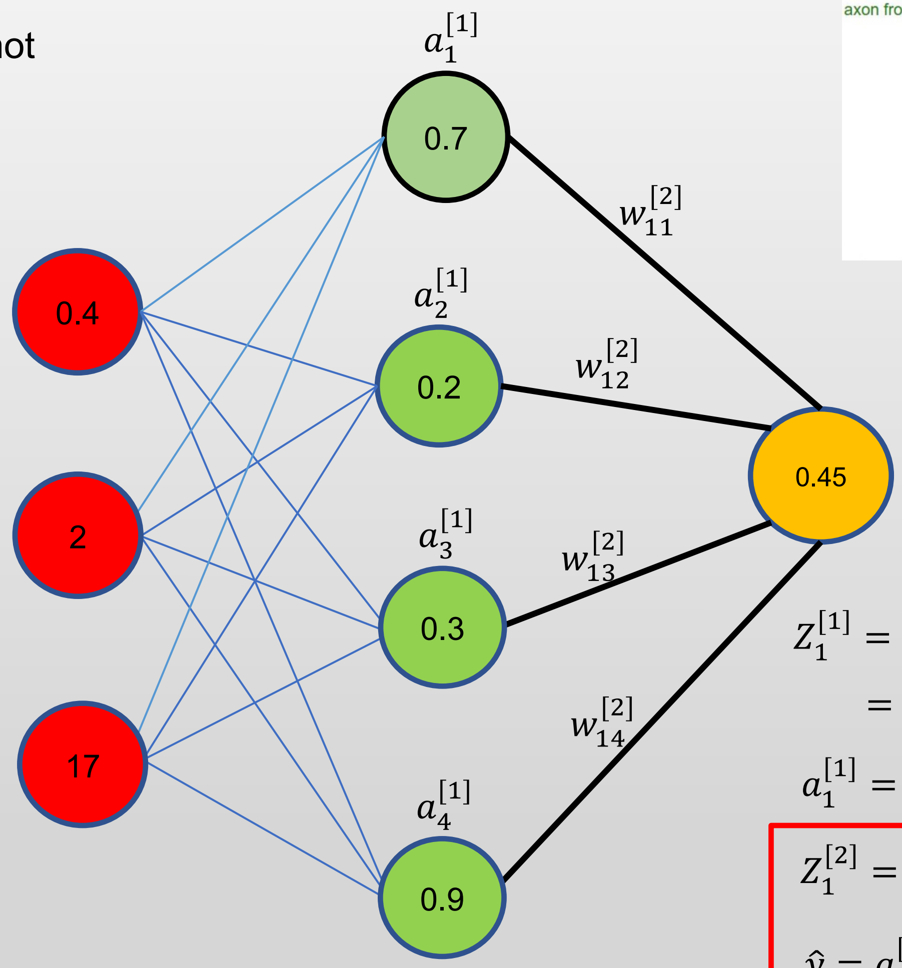
$$Z_1^{[1]} = w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2 + w_{13}^{[1]} x_3 + b_1^{[1]}$$
$$= 0.5 * 0.4 + 0.1 * 2 + 0.0058 * 17 + 0.2 = 0.7$$

$$a_1^{[1]} = f(0.7) = ReLU(0.7) = 0.7$$

Building Neural Networks

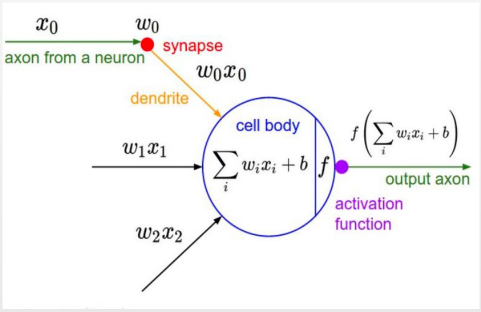
Task: Predict if it worth going to class or not

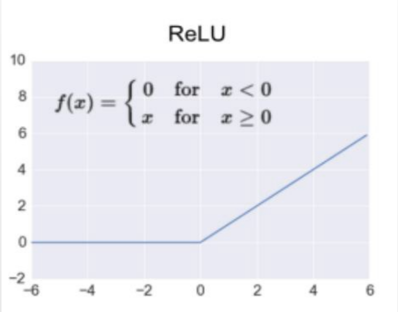
$$X = \begin{bmatrix} difficulty\ of\ material \\ number\ of\ assignments\ due \\ days\ until\ midterm \end{bmatrix}$$



A_1

$Y = [probability\ of\ going\ to\ class]$





$$Z_1^{[1]} = w_{11}^{[1]}x_1 + w_{12}^{[1]}x_2 + w_{13}^{[1]}x_3 + b_1^{[1]}$$

$$= 0.5 * 0.4 + 0.1 * 2 + 0.0058 * 17 + 0.2 = 0.7$$

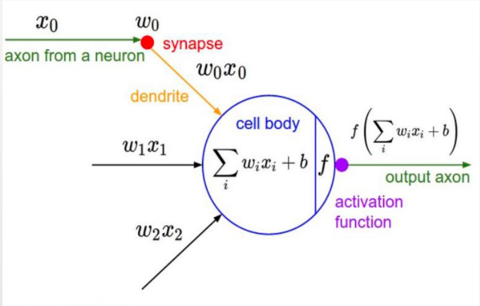
$$a_1^{[1]} = f(0.7) = ReLU(0.7) = 0.7$$

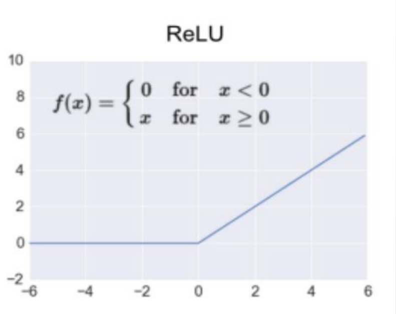
$$Z_1^{[2]} = w_{11}^{[2]}a_1^{[1]} + w_{12}^{[2]}a_2^{[1]} + w_{13}^{[2]}a_3^{[1]} + w_{14}^{[2]}a_4^{[1]} + b_1^{[2]}$$

$$\hat{y} = a_1^{[2]} = f\left(Z_1^{[2]}\right) = 0.45$$

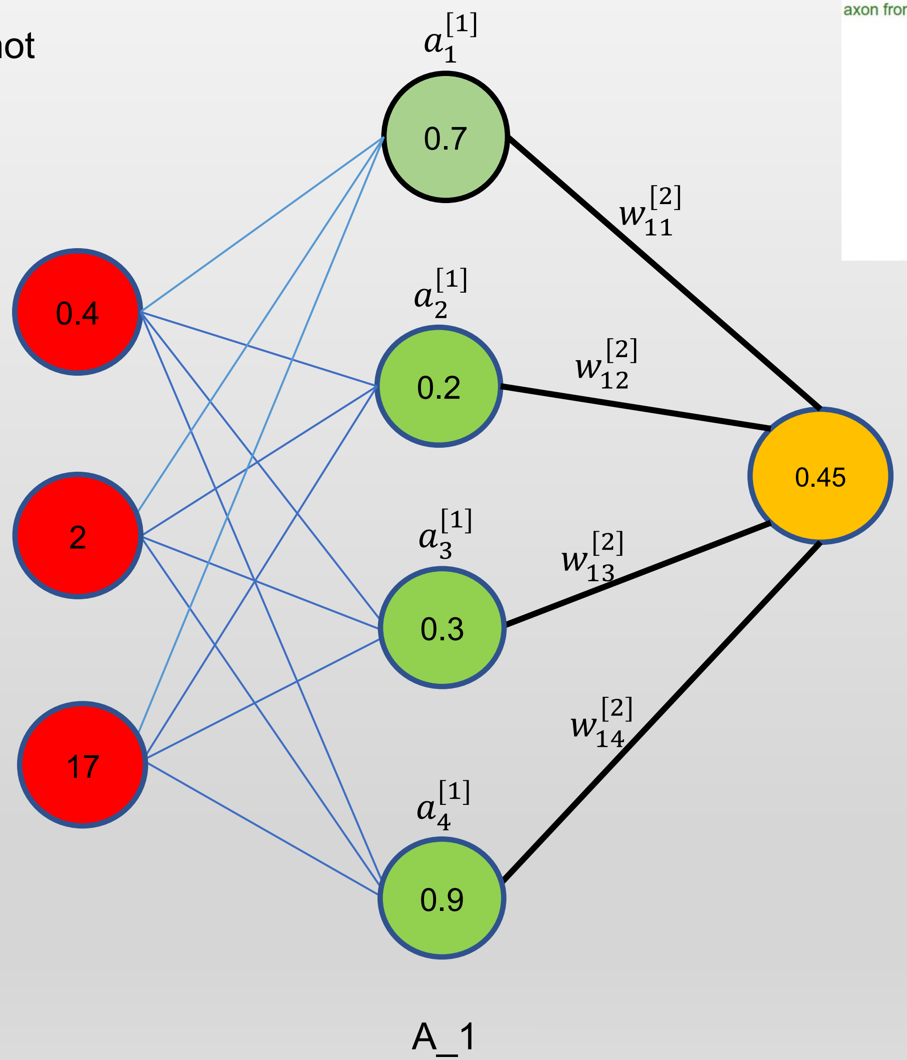
Building Neural Networks

Task: Predict if it worth going to class or not





$$X = \begin{bmatrix} difficulty\ of\ material \\ number\ of\ assignments\ due \\ days\ until\ midterm \end{bmatrix}$$



$$Y = [probability\ of\ going\ to\ class]$$

$$\hat{y} = a_1^{[2]} = f\left(Z_1^{[2]}\right) = 0.45$$

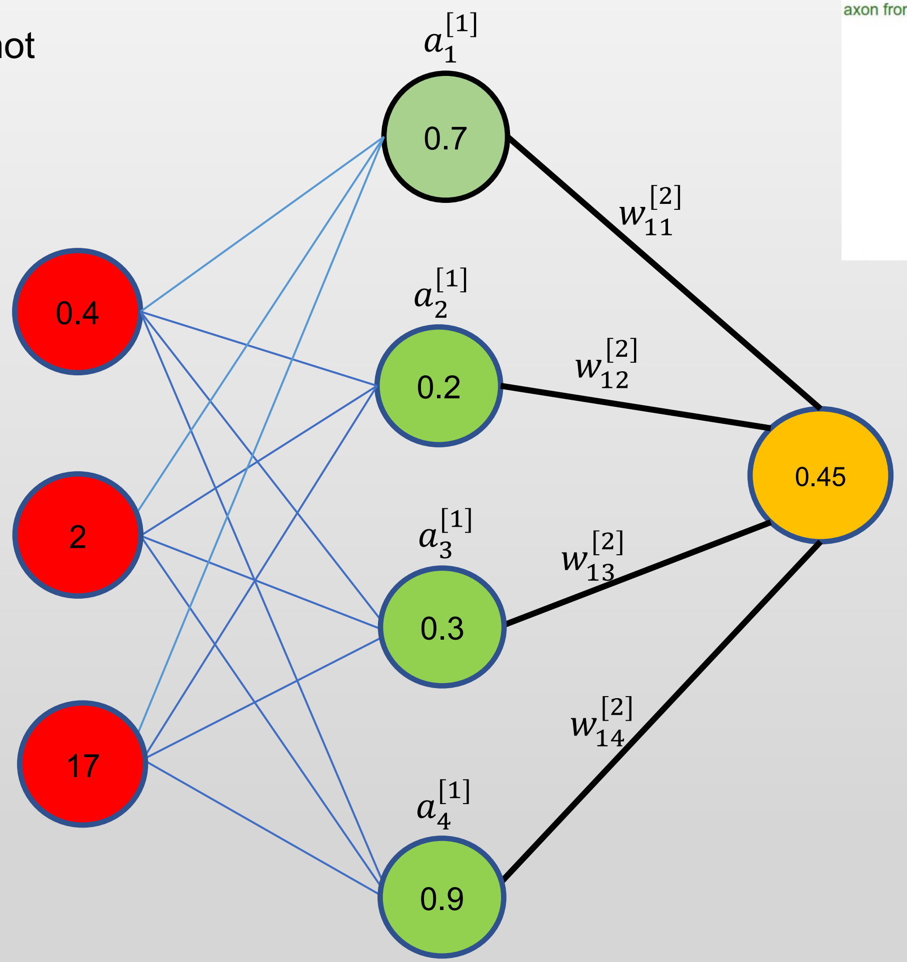
But it turns out midterm material was covered

$$y = 1$$

Building Neural Networks

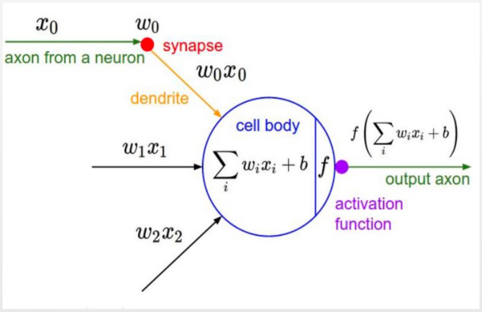
Task: Predict if it worth going to class or not

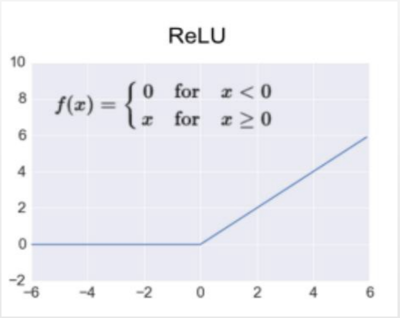
$$X = \begin{bmatrix} difficulty\ of\ material \\ number\ of\ assignments\ due \\ days\ until\ midterm \end{bmatrix}$$



A_1

$$Y = [probability\ of\ going\ to\ class]$$





$$\hat{y} = a_1^{[2]} = f\left(Z_1^{[2]}\right) = 0.45, \qquad y = 1$$

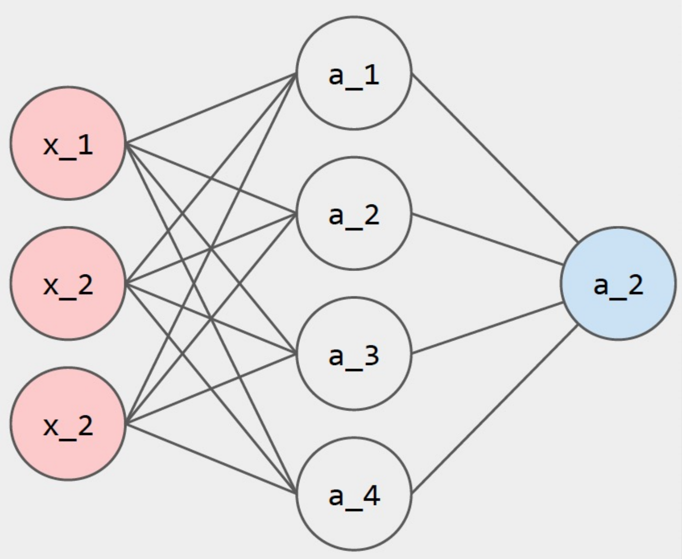
$$L(\hat{y}, y) = -y \log \hat{y} \ - (1-y) \log(1-\hat{y})$$

$$= -1 * \log(0.45) \ - (1-1) \log(1-0.45)$$

$$= 0.798508$$

Binary Cross Entropy

Neural Networks



Forward Propagation:

for each sample i:

for each layer l:

for each activation j :

$$a_j^{[l]} = f^{[l]}\left(\sum_k w_{jk}^{[l]}\, a_k^{[l-1]} + b_j^{[l]}\right) = f^{[l]}\left(z_j^{[l]}\right)$$

Neural Networks

Compute Cost: In order to train our neural network, we need some way to tell us how far off its estimate was from the actual value

We define the cost function, $J(\hat{y}, y)$ as the sum of losses

$$J(\hat{y}, y) = \sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right) \qquad m: number\ of\ examples$$

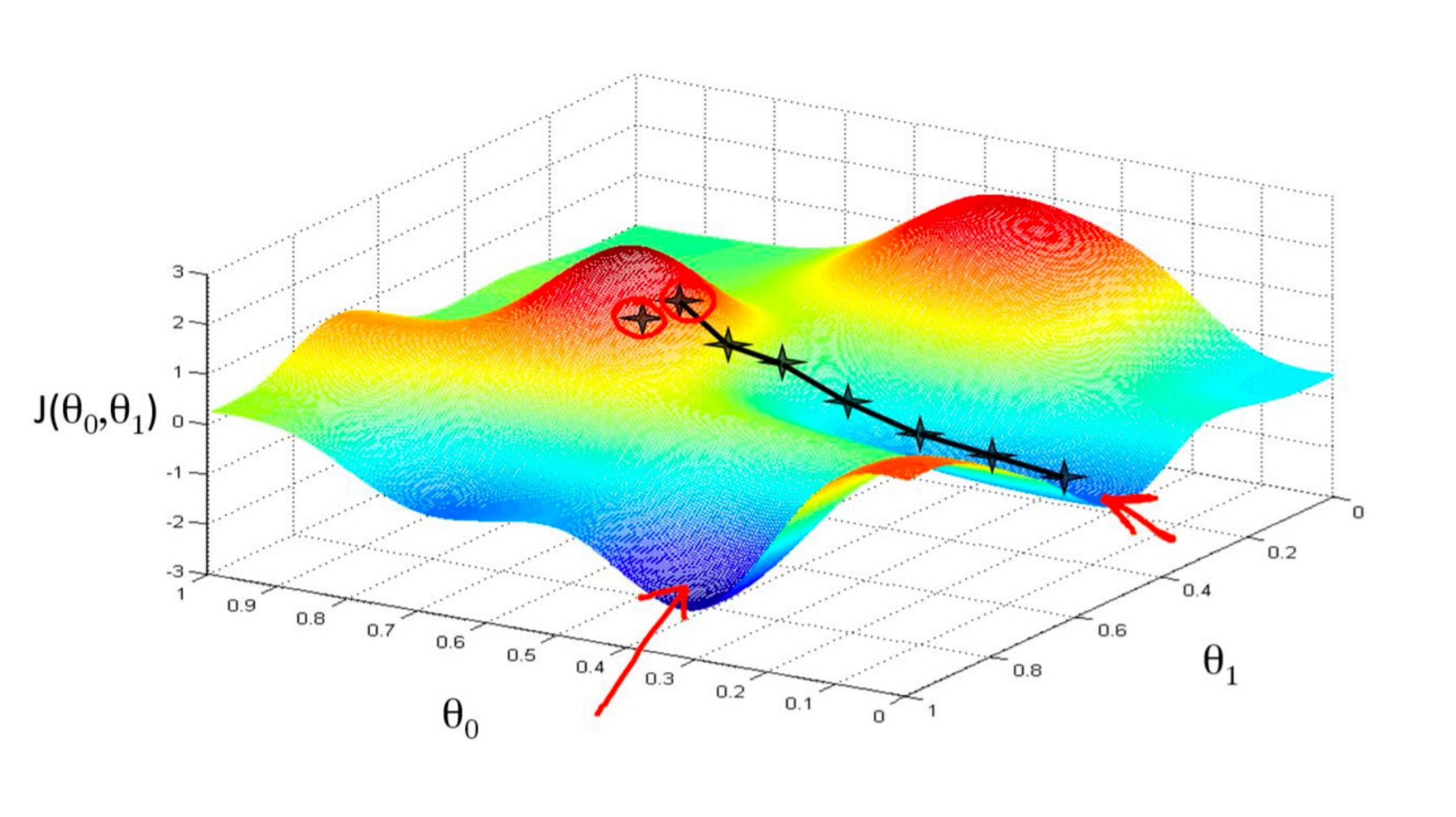
$$J(\theta) = Cost\ Function$$

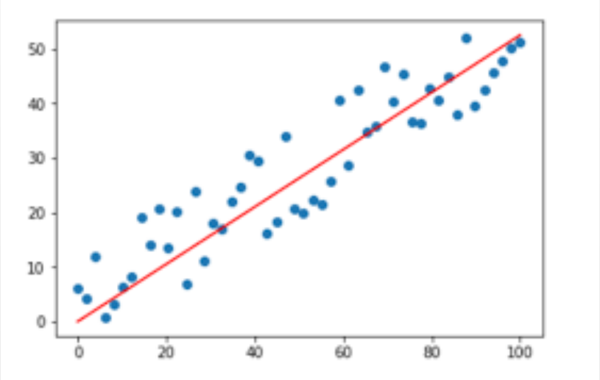
Loss = Error for a single training example

Cost = Sum of all losses

Neural Networks

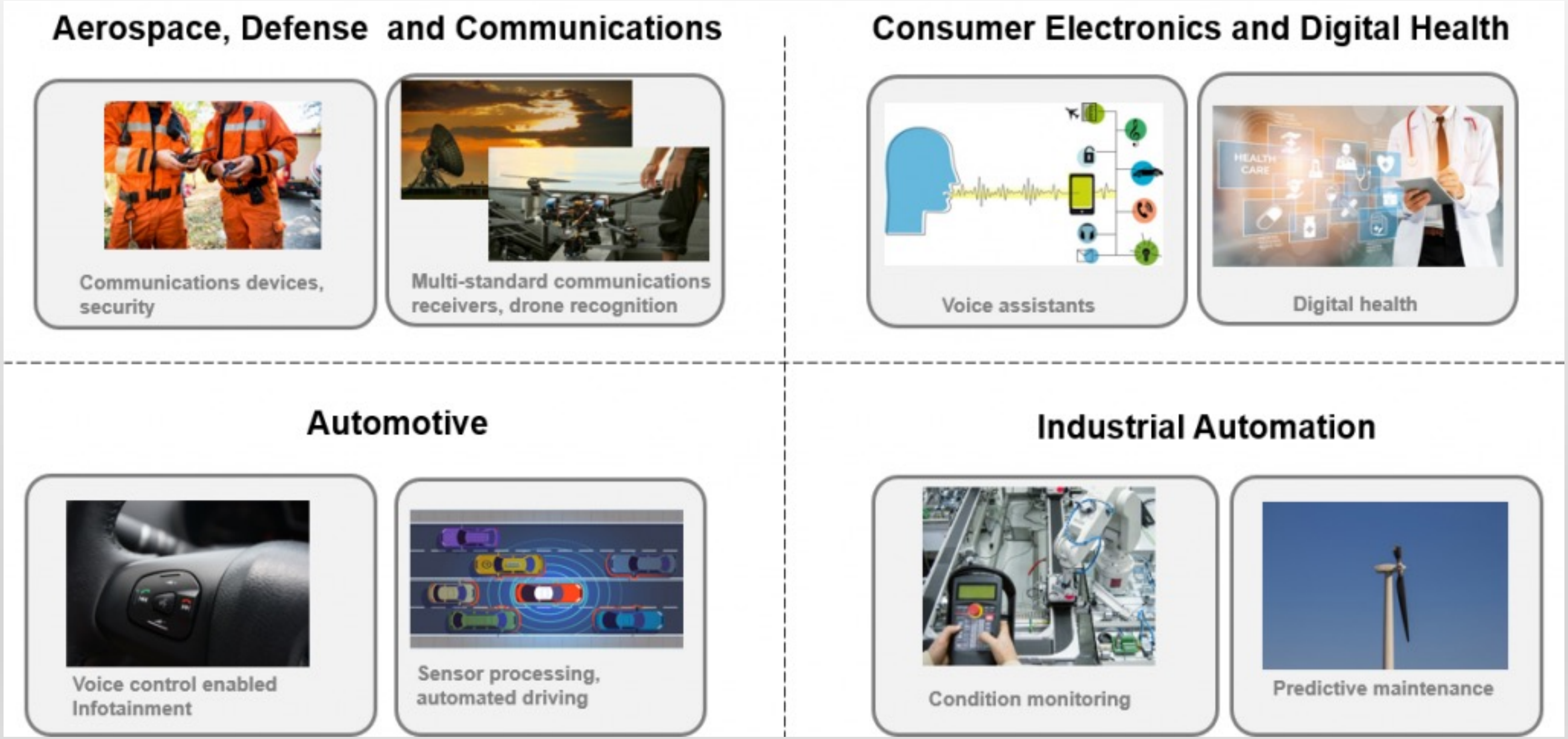
Train with gradient descent





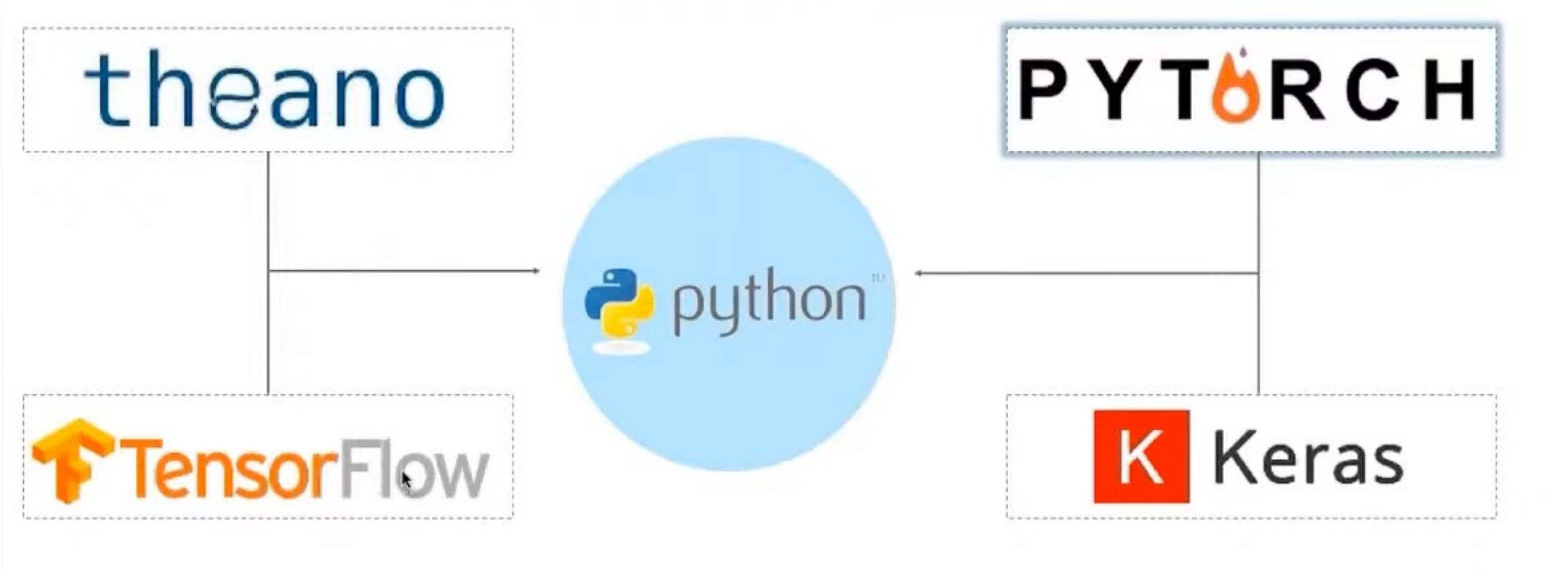
Example: $y = mx + b$

Deep Learning Applications



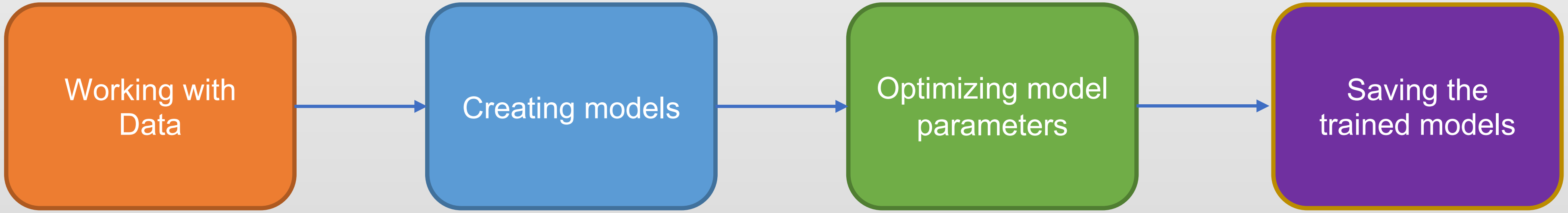
Deep Learning Applications

Neural Networks Packages



Pytorch Tutorial

Machine Learning Workflows



Pytorch Tutorial

Predict if an input image belongs to one of the following classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, or Angle boot.

FasionMNIST Dataset

Pytorch Tutorial: Working with Data

PyTorch

Dataset: stores the samples and their corresponding labels
TorchText, TorchVision, and TorchAudio

torchvision.datasets: CIFAR, COCO, FashionMNIST

transform and target_transform to modify the samples and lables respectively

DataLoader: wraps an iterable around the Dataset

Working with Data

```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
```

Importing Modules

```
# Download training data from open datasets.
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)

# Download test data from open datasets.
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)
```

Define Training and Test Dataset

Working with Data

Pass the Dataset as an argument to DataLoader

Wraps an iterable over dataset and supports automatic batching, sampling, shuffling and multiprocess data loading

```
batch_size = 64

# Create data loaders.
train_dataloader = DataLoader(training_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)

for X, y in test_dataloader:
    print(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"Shape of y: {y.shape} {y.dtype}")
    break
```

Define DataLoader

Creating Models

```
# Get cpu or gpu device for training.
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")

# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)
print(model)
```

Check if GPU is Available

Create Neural Networks Model

Optimizing the Model Parameters

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
```

Define Training Loop

```
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

Define Test Dataset

```
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
```

Training Loop

```
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
print("Done!")
```

Saving Models

```
torch.save(model.state_dict(), "model.pth")
print("Saved PyTorch Model State to model.pth")
```

Loading Models

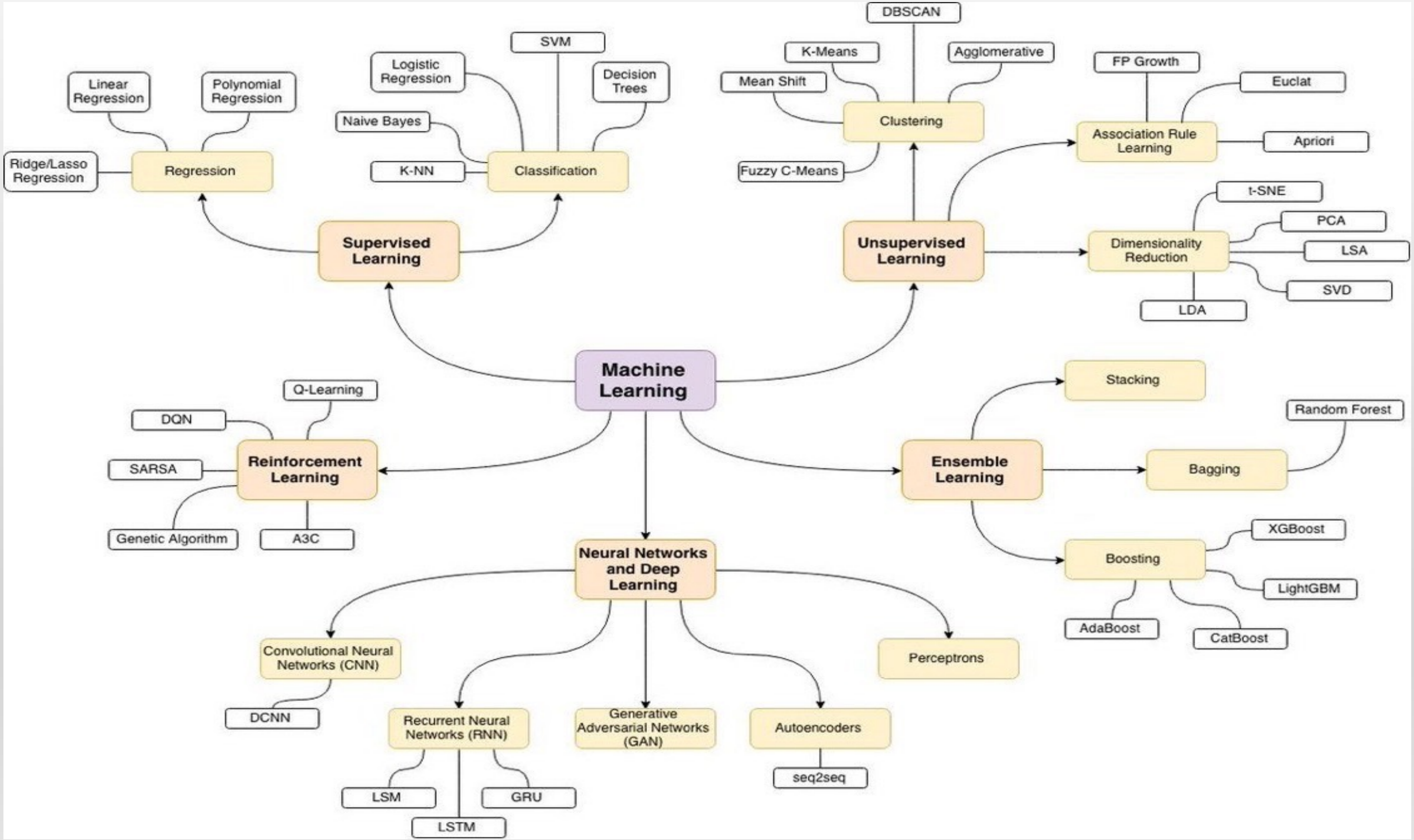
```
model = NeuralNetwork()
model.load_state_dict(torch.load("model.pth"))
```

Make Predictions

```
classes = [
    "T-shirt/top",
    "Trouser",
    "Pullover",
    "Dress",
    "Coat",
    "Sandal",
    "Shirt",
    "Sneaker",
    "Bag",
    "Ankle boot",
]

model.eval()
x, y = test_data[0][0], test_data[0][1]
with torch.no_grad():
    pred = model(x)
    predicted, actual = classes[pred[0].argmax(0)], classes[y]
    print(f'Predicted: "{predicted}", Actual: "{actual}"')
```

Summary



Machine Learning
Supervised Learning
Regression
Linear Regression
Polynomial Regression
Ridge/Lasso Regression
Classification
Logistic Regression
SVM
Decision Trees
Naive Bayes
K-NN
Unsupervised Learning
Clustering
DBSCAN
K-Means
Agglomerative
Mean Shift
Fuzzy C-Means
Association Rule Learning
FP Growth
Euclat
Apriori
Dimensionality Reduction
t-SNE
PCA
LSA
SVD
LDA
Reinforcement Learning
Q-Learning
DQN
SARSA
Genetic Algorithm
A3C
Ensemble Learning
Stacking
Bagging
Random Forest
Boosting
XGBoost
LightGBM
AdaBoost
CatBoost
Neural Networks and Deep Learning
Convolutional Neural Networks (CNN)
DCNN
Recurrent Neural Networks (RNN)
LSM
LSTM
GRU
Generative Adversarial Networks (GAN)
Autoencoders
seq2seq
Perceptrons