Introduction to Artificial Neural Networks

Hao Ji, Data Scientist

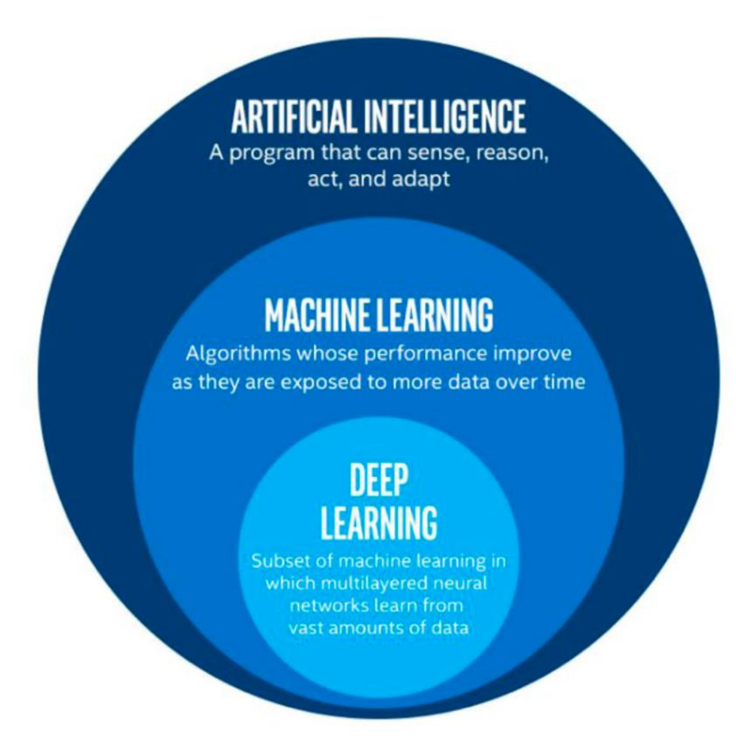
USC Center for Advanced Research Computing

Content

Introduction to Deep Learning

Deep Learning: subfield of traditional machine learning

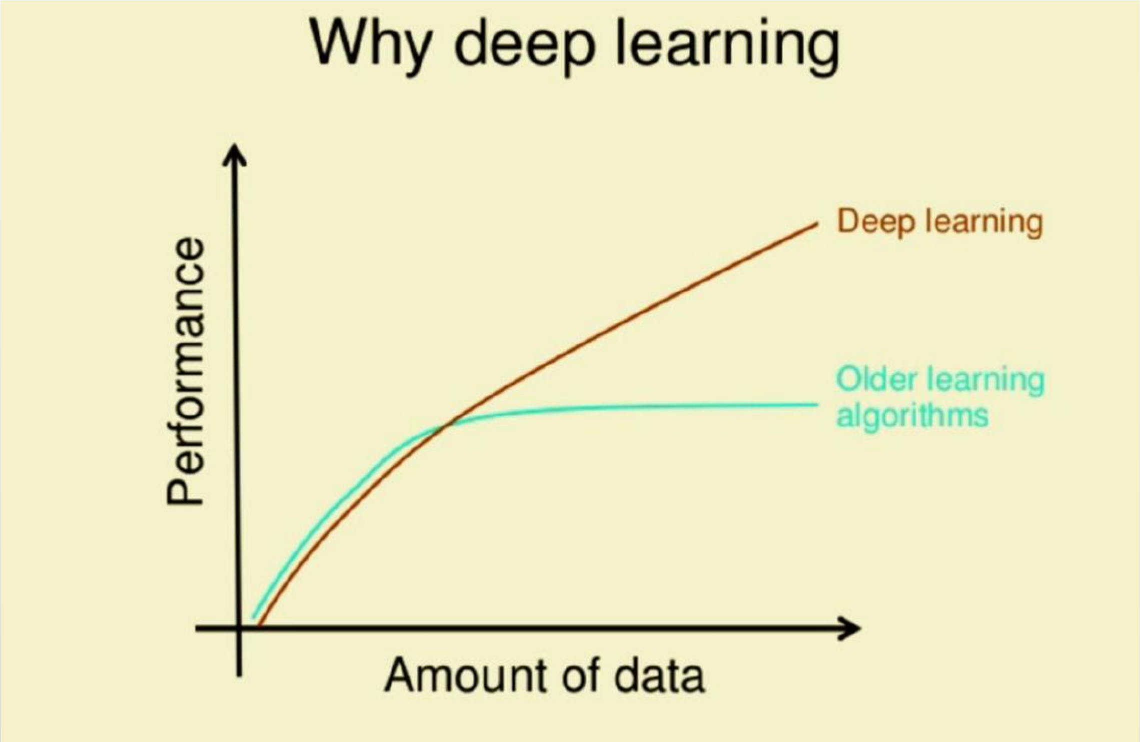
- Inspired by the structure and function of the brain: Artificial Neural Networks
- Computer vision: Tesla recognizing items on a street
- Text Generation: An algorithm trained to create a new Shakespeare piece
- Speech recognition
- Computer Games: AlphaGo



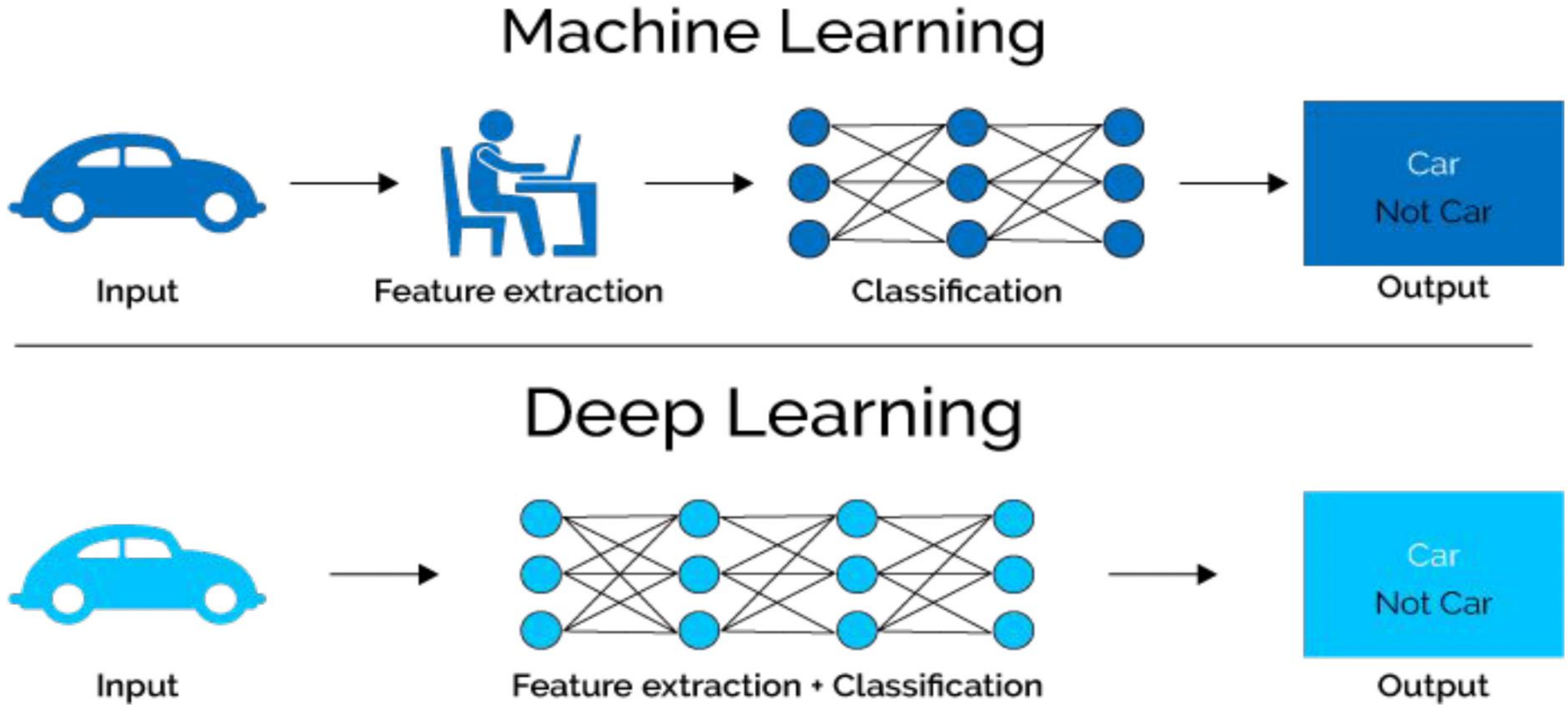
Introduction to Deep Learning

What drives the recent development of Deep Learning?

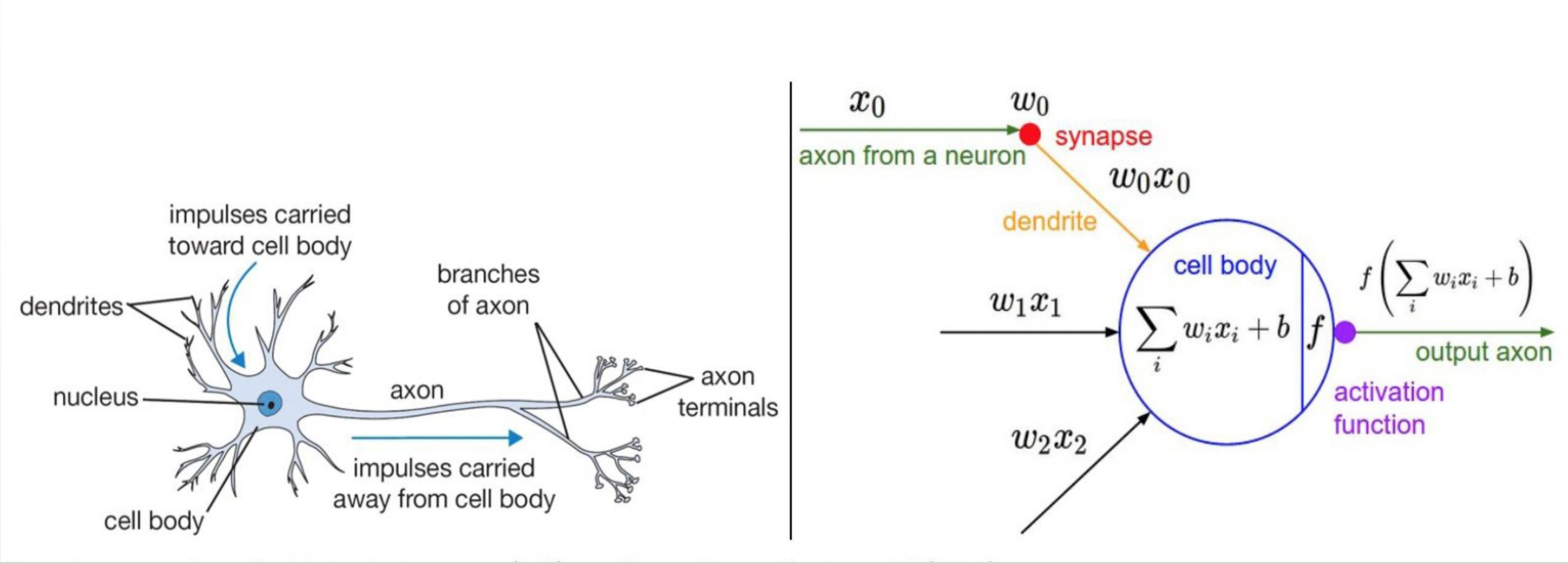
- Larger amounts of data available
- Data Storage
- Strong computation units such as GPU's



Introduction to Deep Learning



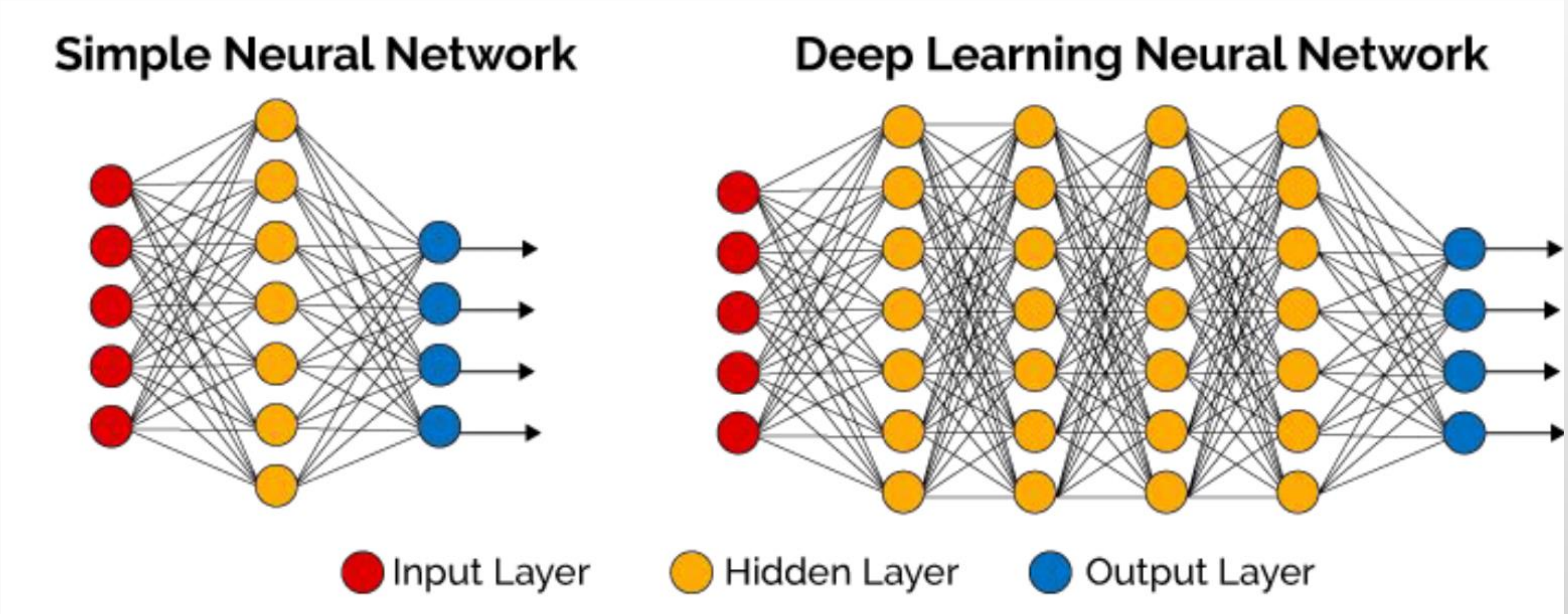
Neural Networks



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

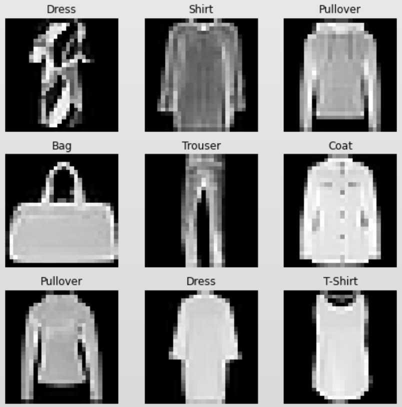
Building Neural Networks

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

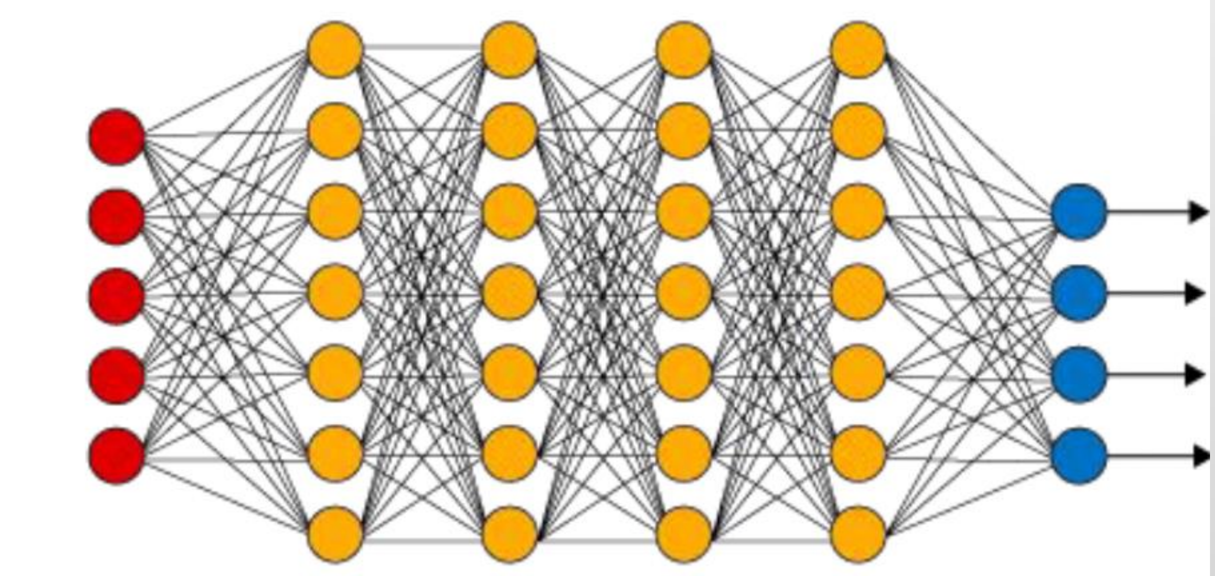
FasionMNIST Dataset

Fashion-MNIST is a dataset comprising of 28×28 grayscale images of 70,000 fashion products from 10 categories, with 7,000 images per category. The training set has 60,000 images and the test set has 10,000 images.

Building Neural Networks

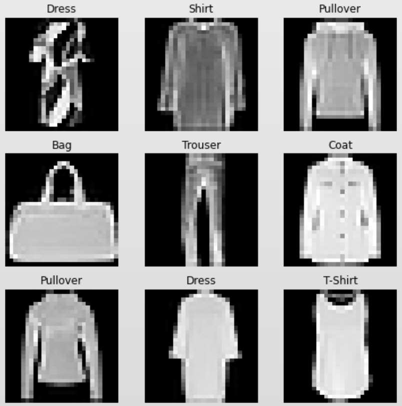


Input X: (28 * 28)

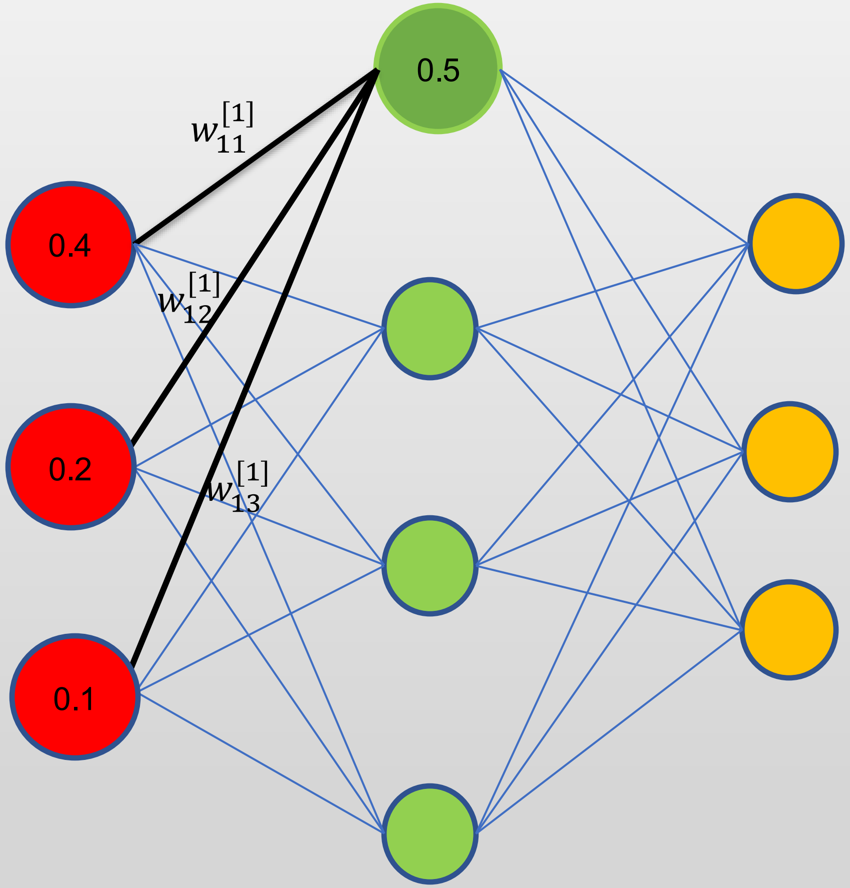


Make Predictions based on Logits

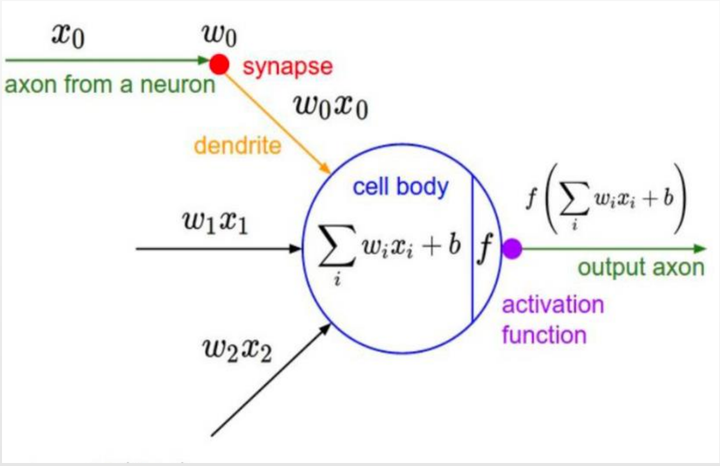
Building Neural Networks



28 * 28 = 784



A_1



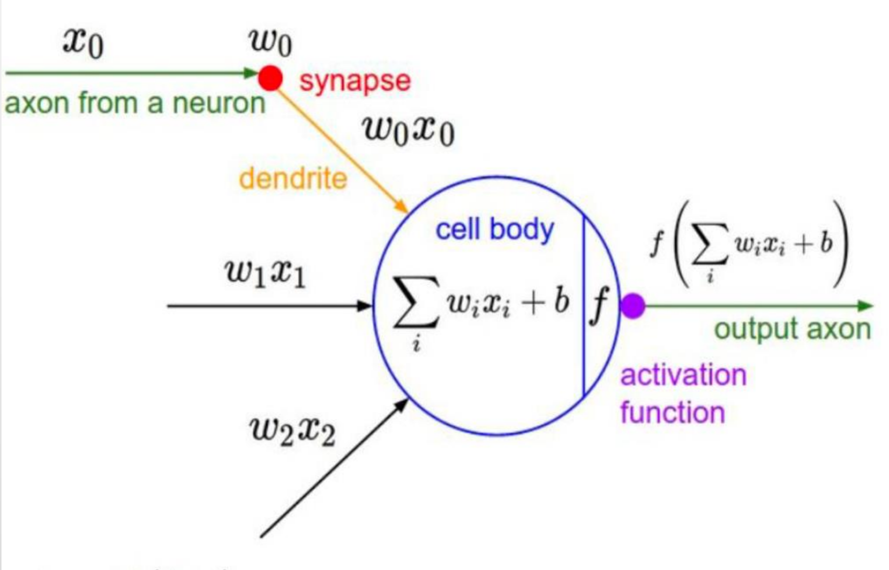
$$Z_1^{[1]} = w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2 + w_{13}^{[1]} x_3 + b_1^{[1]}$$

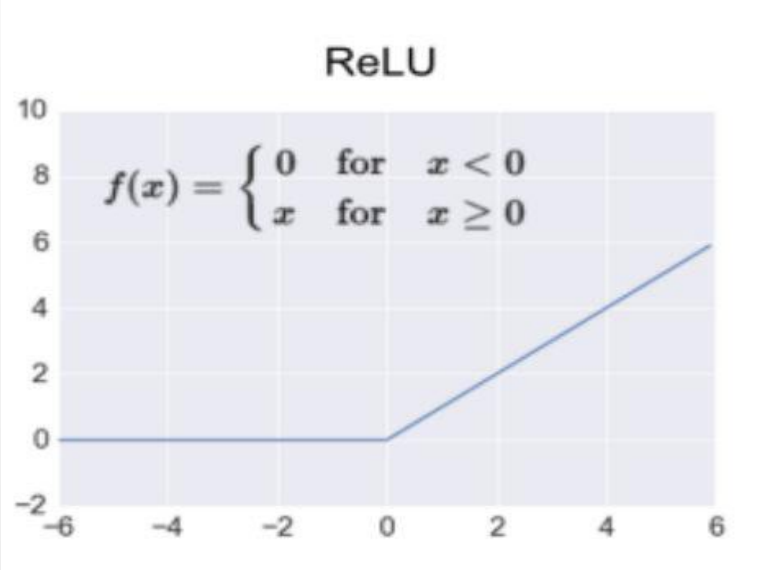
$$= 0.5 * 0.4 + 0.1 * 0.2 + 0.8 * 0.1 + 0.2 = 0.5$$

$$a_1^{[1]} = f(0.5) = ReLU(0.5) = 0.5$$

Weights are initialized randomly

Building Neural Networks



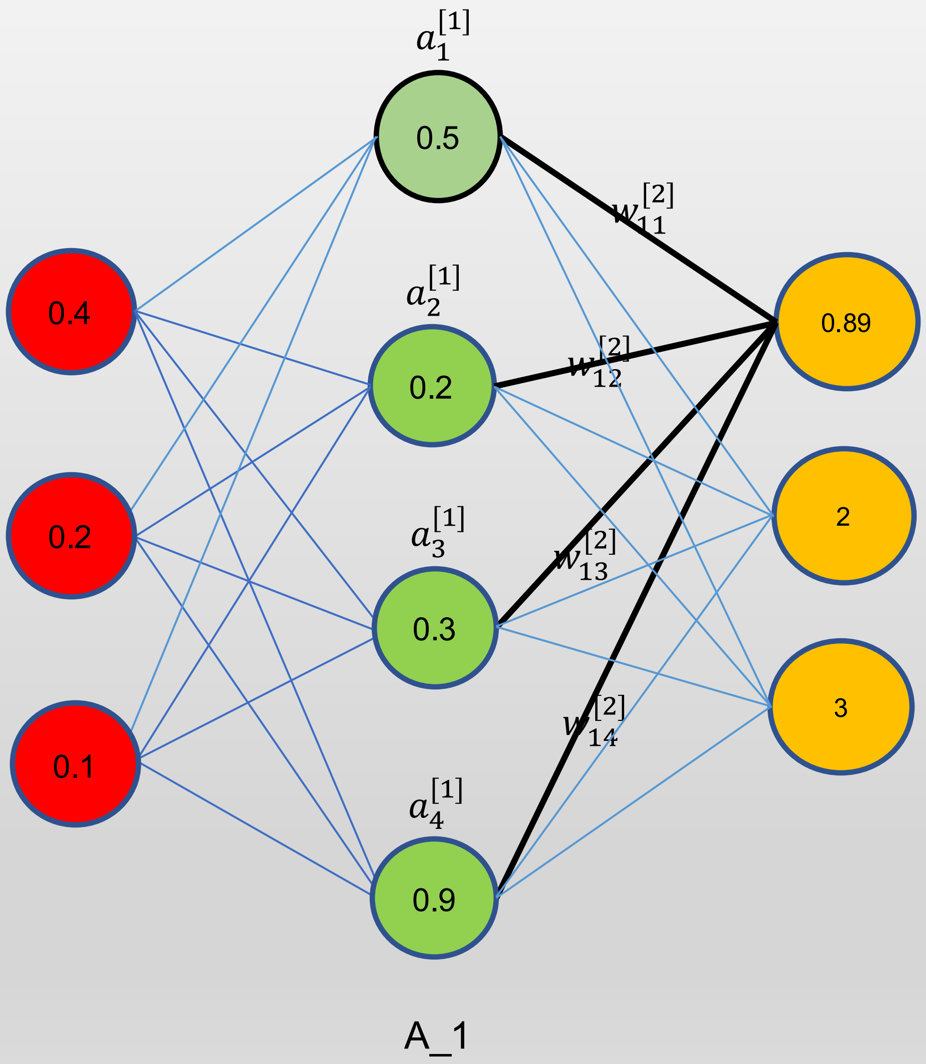


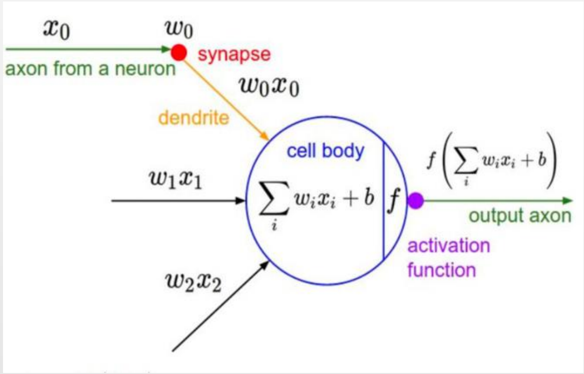
$$Z_1^{[1]} = w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2 + w_{13}^{[1]} x_3 + b_1^{[1]}$$

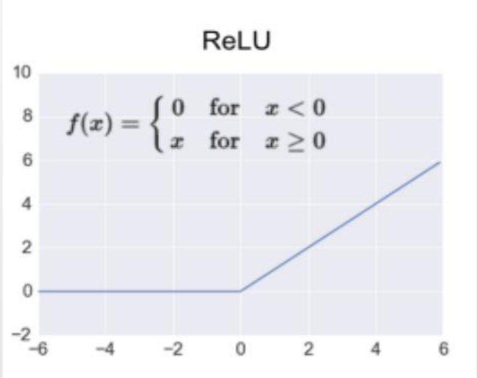
$$= 0.5 * 0.4 + 0.1 * 0.2 + 0.8 * 0.1 + 0.2 = 0.5$$

$$a_1^{[1]} = f(0.5) = ReLU(0.5) = 0.5$$

Building Neural Networks







$$Z_1^{[2]} = w_{11}^{[2]} a_1^{[1]} + w_{12}^{[2]} a_2^{[1]} + w_{13}^{[2]} a_3^{[1]} + w_{14}^{[2]} a_4^{[1]} + b_1^{[2]}$$
$$= 0.3 * 0.5 + 0.1 * 0.2 + 0.2 * 0.3 + 0.4 * 0.9 + 0.3$$
$$= 0.89$$
$$a_1^{[2]} = f\left(Z_1^{[2]}\right) = f(0.89) = ReLU(0.89) = 0.89$$

Neural Networks

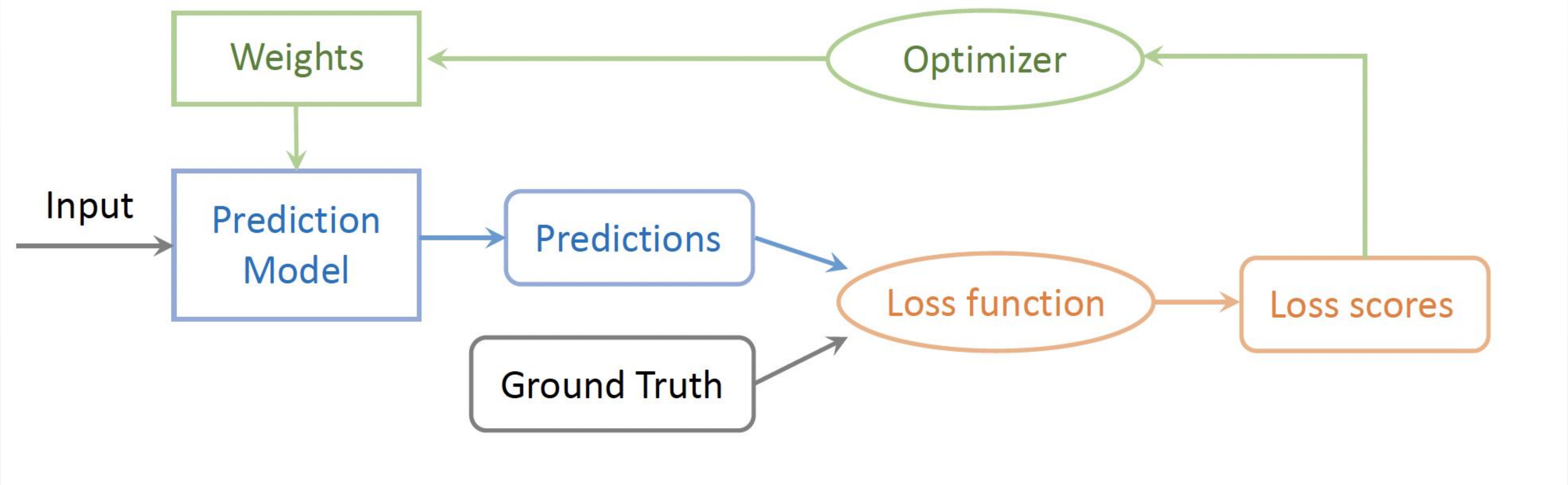
Three steps to training a neural network

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



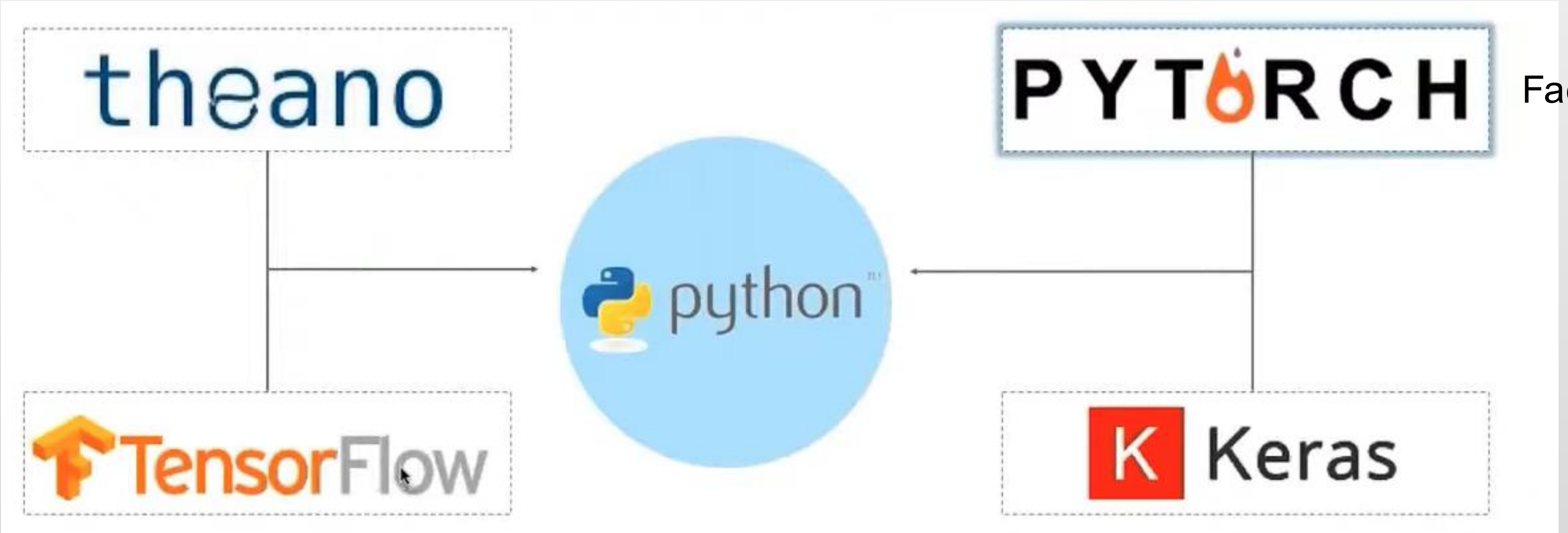
The training pipeline consists of choosing the prediction model, the loss function and the optimizer.

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

Deep Learning Applications

[Deep Learning Applications]

Neural Networks Packages



Facebook

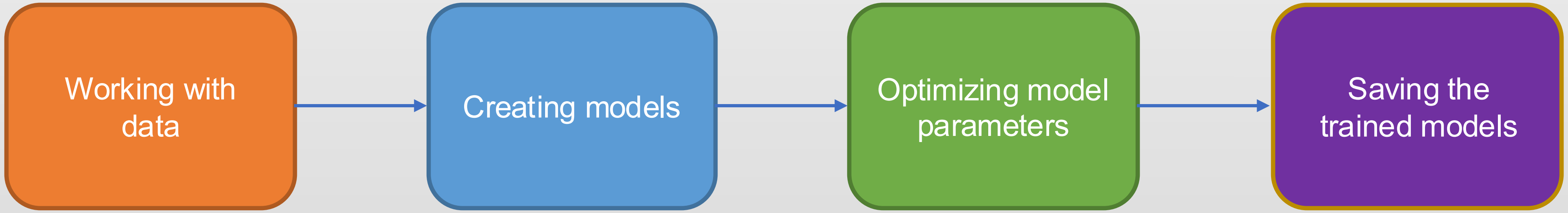
Google

Pytorch Tensors

- Tensors are a specialized data structure similar to arrays and matrices
- PyTorch uses tensors to encode the inputs and outputs of a model, as well as model’s parameters
- Can run on GPUs or other hardware accelerators
- Optimized for automatic differentiation

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.

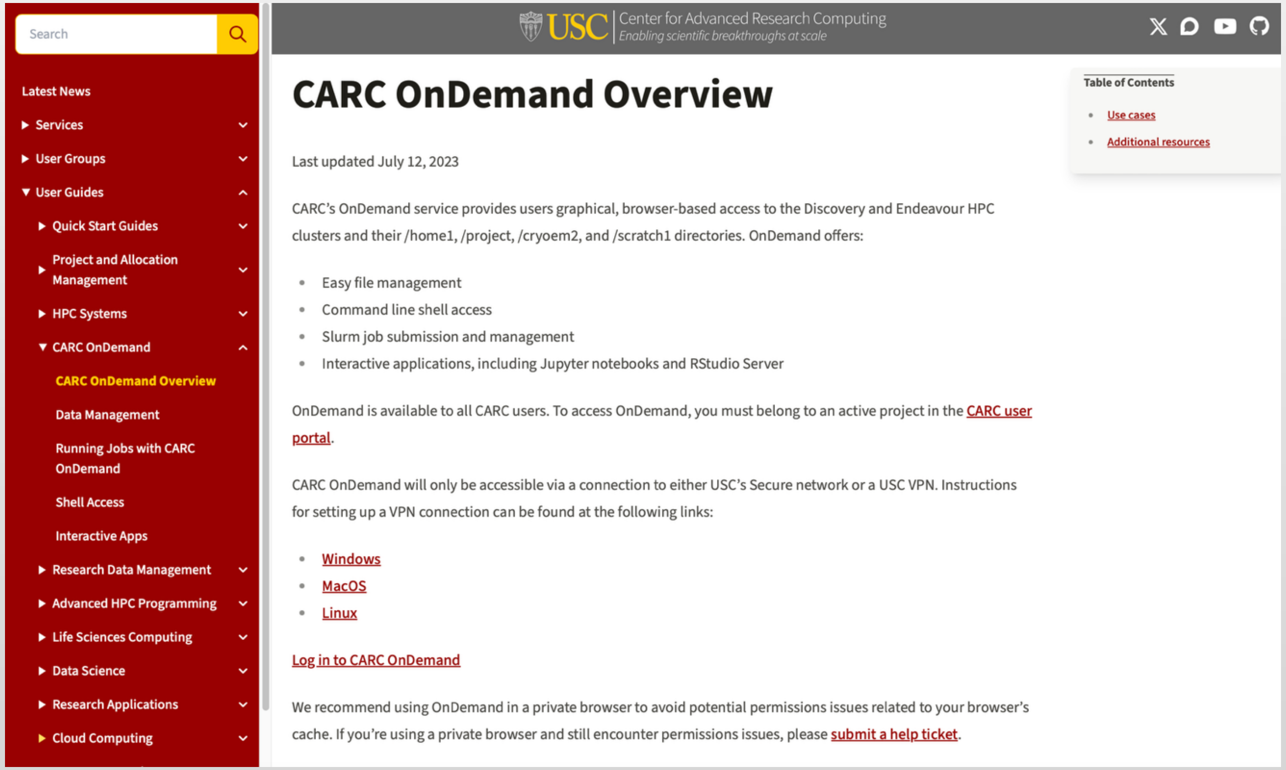
FasionMNIST Dataset

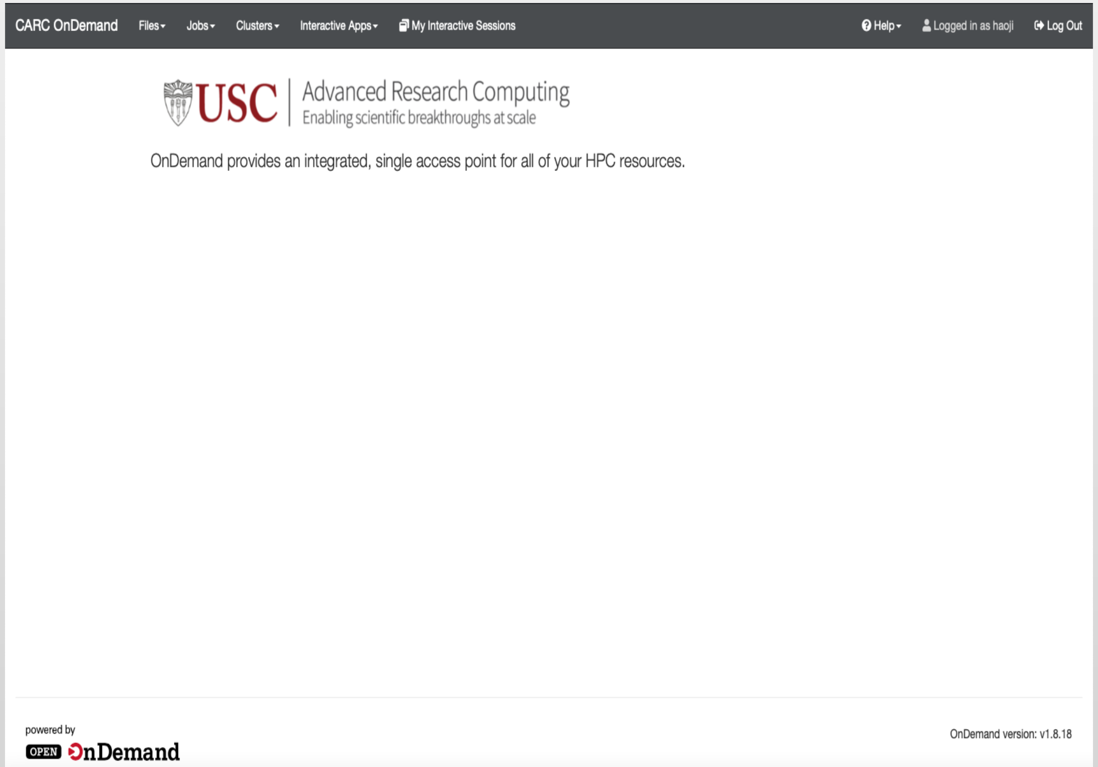
Using Anaconda & Juypyter Kernel

Let’s build our Conda Environment First: https://github.com/uschpc/Building-NeuralNetworks.git

CARC OnDemand

https://www.carc.usc.edu





Pytorch Tutorial: Working with Data

PyTorch

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

torchvision.datasets: CIFAR, COCO, FashionMNIST

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

```
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

```python
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}"')
```