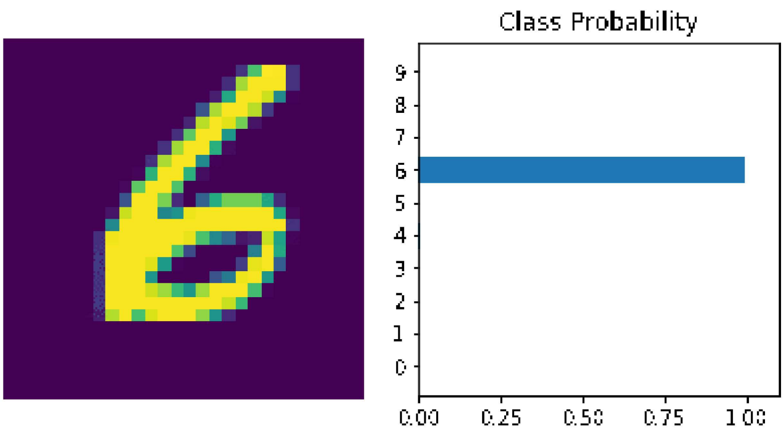
NEURAL NETWORK

Training Neural Network from Scratch using PyTorch in just 7 cells

MNIST Hand Digit Recognition using PyTorch in just 7 cells using Neural Network (Multi-layer perceptron) from Scratch

Khush Patel · Follow
Published in Towards Data Science
5 min read · Apr 26, 2020



MNIST Hand Digit Recognition using PyTorch

Content of the blog:

1. Installation and import modules
2. re-processing and loading the dataset
3. Designing the model
4. Training the model

5. Visualizing the Output

Installation:

First thing first. Regardless of the operating system just run below command which will install all the required modules to run the below code snippets. If you use anaconda then also you can install it with conda command.

```
pip install torch torchvision numpy matplotlib
```

```
conda install torch torchvision numpy matplotlib
```

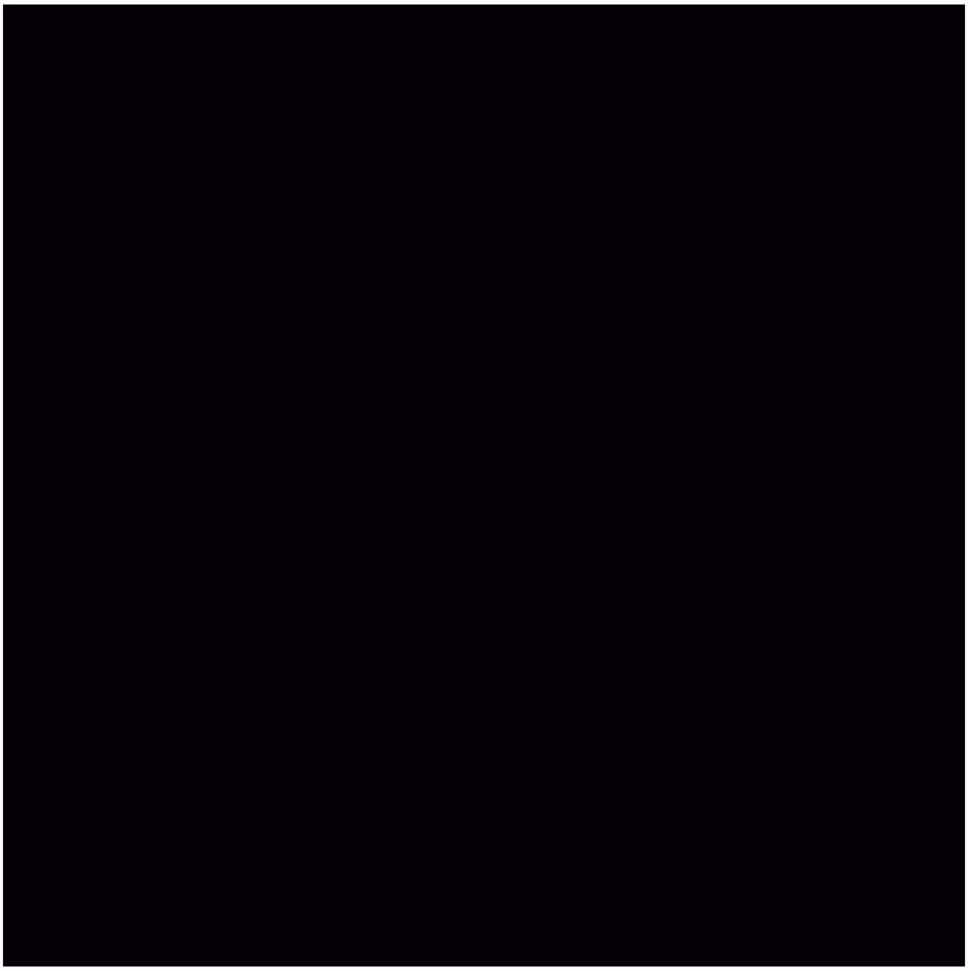
Import Statements:

`import torch` is used to add all the essential modules to build the neural network and `torchvision` is used to add other functionalities like pre-processing and transformation of the data. `numpy` is used to work with image arrays and `matplotlib` is to display the image.

```
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
from torch import optim
import numpy as np
%matplotlib inline
```

PyTorch has `transform` module to convert images to tensors and pre-process every image to normalize with a standard deviation 1. `torchvison` has build-in dataset `MNIST` hand digits which I am going to use for further explanation for all below code snippets. `DataLoader` is the PyTorch module to combine the image and its corresponding label in a package. So we can easily access both the things simultaneously. Just note that we are adding `batch_size` as 64 to create a batch of 64 images in one iteration.

```
transform = transforms.Compose([
    transforms.ToTensor(),
    # transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
trainset = torchvision.datasets.MNIST('~/.pytorch/MNIST_data/',
train=True, transform=transform, download=True)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
shuffle=True)
```



Source: Tenor

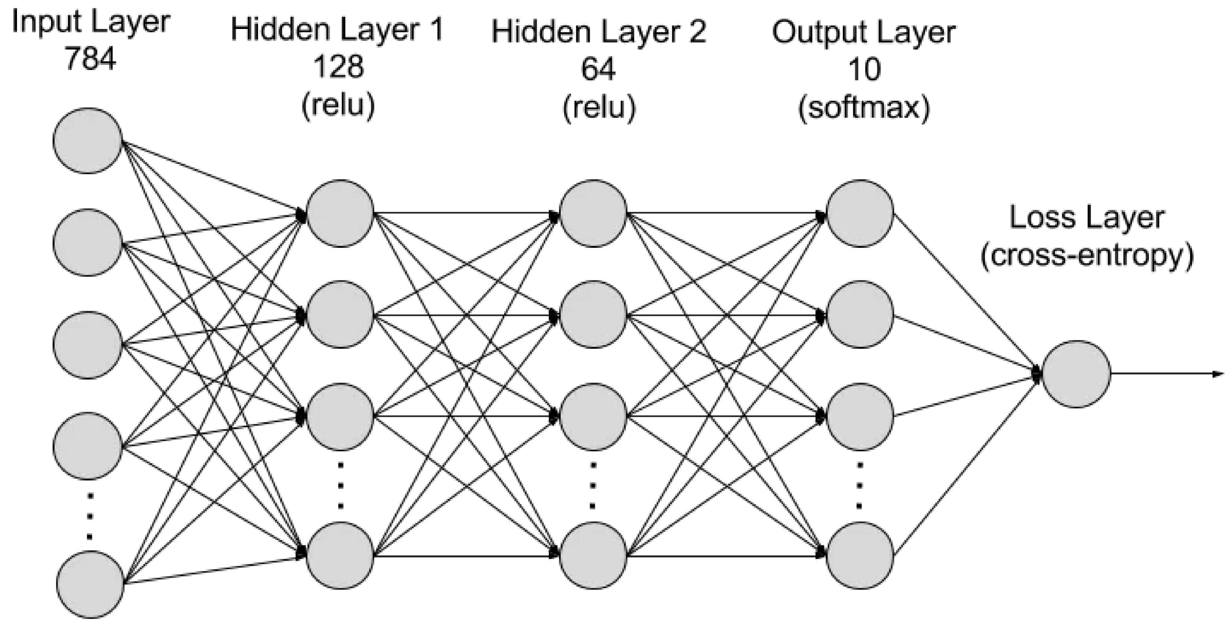
To train any neural network first we have to understand the input size of the images, # of output classes and hidden layers of the neural networks. Hereby checking `trainloader.dataset.train_data.shape` we will get 64,1,28,28 which denotes 64 images with height and width 28 and channel 1 as greyscale images.

Input Size: So the input size is 784 which is the product of height(28) and width(28) of the image. The image has only 1 channel so no need to add it in the input size.

Output Size: We have digits from 0–9 so a total of 10 possible class options. Therefor output size is 10

Hidden Layers: Layers are between the input layer and output layers are basically known as hidden layers. In our case, we have an input image of 784 nodes and the output size is 10 so in between, we are adding layers of 128 and 64. Thus our network will scale from 784 to 128 to 64 to 10.

```
input_size = trainloader.dataset.train_data.shape[1] *
trainloader.dataset.train_data.shape[2]
hidden_layers = [128,64]
output_size = 10
```



Simple Neural Network(Multi-Layer Perceptron) for Hand Digit MNIST Classification (Source: Udacity)

`torchvision` has `nn` module which has all the functionalities to build a neural network. Initially, adding input size to the first hidden layer which is 784 to 128 followed by ReLU (Activation function). From 128 to 64 with the same ReLU activation function and 64 to 10 in the very last layer. To getting probabilities distribution we are adding the final layer of `LogSoftmax` and dimension = 1 because we have 64 image batch so it will give 64x10 results in the output.

To compute the errors and mistakes of the neural network we are adding `NLLLoss` cross-entropy loss (Negative log-likelihood loss) as a criterion (error function) and optimizer as `SGD` (stochastic gradient descent) with a learning rate of 0.003.

> *Read more about Cross-Entropy Loss here and Stochastic gradient descent here*

```
model = nn.Sequential(
    nn.Linear(input_size, hidden_layers[0]),
    nn.ReLU(),
    nn.Linear(hidden_layers[0], hidden_layers[1]),
    nn.ReLU(),
    nn.Linear(hidden_layers[1], output_size),
    nn.LogSoftmax(dim=1)
)
print(model)
criterion = nn.NLLLoss()
optimizer = optim.SGD(model.parameters(), lr=0.003)
```

Training:

Now, we successfully defined the model and its time to train the model by passing the images and corresponding labels. But before that, we are flattening our images from 28x28 to 784x1 and setting all the gradient to zero to train the weights and bias for models.

Finally `model(images)` will train the model and `criterion` will calculate the loss. `loss.backward()` is used to backpropagation and `optimizer.step()` will update the weights as per backpropagated weights and bias.

We are printing loss as every epoch of training. Just make sure your training loss will decrease as epochs are increasing. If you are not getting less loss with every epoch, you made some mistakes in the code.

```
epochs = 5
for e in range(epochs):
    running_loss = 0
    for images, labels in trainloader:
        # Flatten the Image from 28*28 to 784 column vector
        images = images.view(images.shape[0], -1)

        # setting gradient to zeros
        optimizer.zero_grad()
        output = model(images)
        loss = criterion(output, labels)

        # backward propagation
        loss.backward()

        # update the gradient to new gradients
        optimizer.step()
        running_loss += loss.item()
```

```
    else:
        print("Training loss: ",(running_loss/len(trainloader)))
```

Visualization:

In the prediction phase of the neural network, we are passing images and their probability distribution to visualizing the image. `ax1` is the original image of any digit and `ax2` is a probability distribution. Just setting xlabel and ylabel between 0–9 and title of the plot.

```
def view_classify(img, ps):
    ps = ps.data.numpy().squeeze()
    fig, (ax1, ax2) = plt.subplots(figsize=(6,9), ncols=2)
    ax1.imshow(img.resize_(1, 28, 28).numpy().squeeze())
    ax1.axis('off')
    ax2.barh(np.arange(10), ps)
    ax2.set_aspect(0.1)
    ax2.set_yticks(np.arange(10))
    ax2.set_yticklabels(np.arange(10))
    ax2.set_title('Class Probability')
    ax2.set_xlim(0, 1.1)
    plt.tight_layout()
```

Prediction:

Turn of the gradient as we are using the same model which will start training so we are turning off all the gradient and getting the probability distribution of the testing image. All probability in logarithm so we are converting it to 0–1 and visualizing image using the function.

```
# Getting the image to test
images, labels = next(iter(trainloader))

# Flatten the image to pass in the model
img = images[0].view(1, 784)

# Turn off gradients to speed up this part
with torch.no_grad():
    logps = model(img)

# Output of the network are log-probabilities, need to take
exponential for probabilities
ps = torch.exp(logps)
view_classify(img, ps)
```

And you are done with training a neural network on the MNIST Hand digit recognition dataset from very scratch using PyTorch.

> Now, Its time for celebration, because you achieved it!!!

Source: Tenor

Thank you for reading the blog and appreciating my efforts. Feel free to comment and ask questions with your suggestions. You can connect with me on LinkedIn, Twitter and my check out my Website for more Deep Learning projects. Happy Learning!!!

Pytorch | Neural Networks | Deep Learning | Computer Vision | Machine Learning

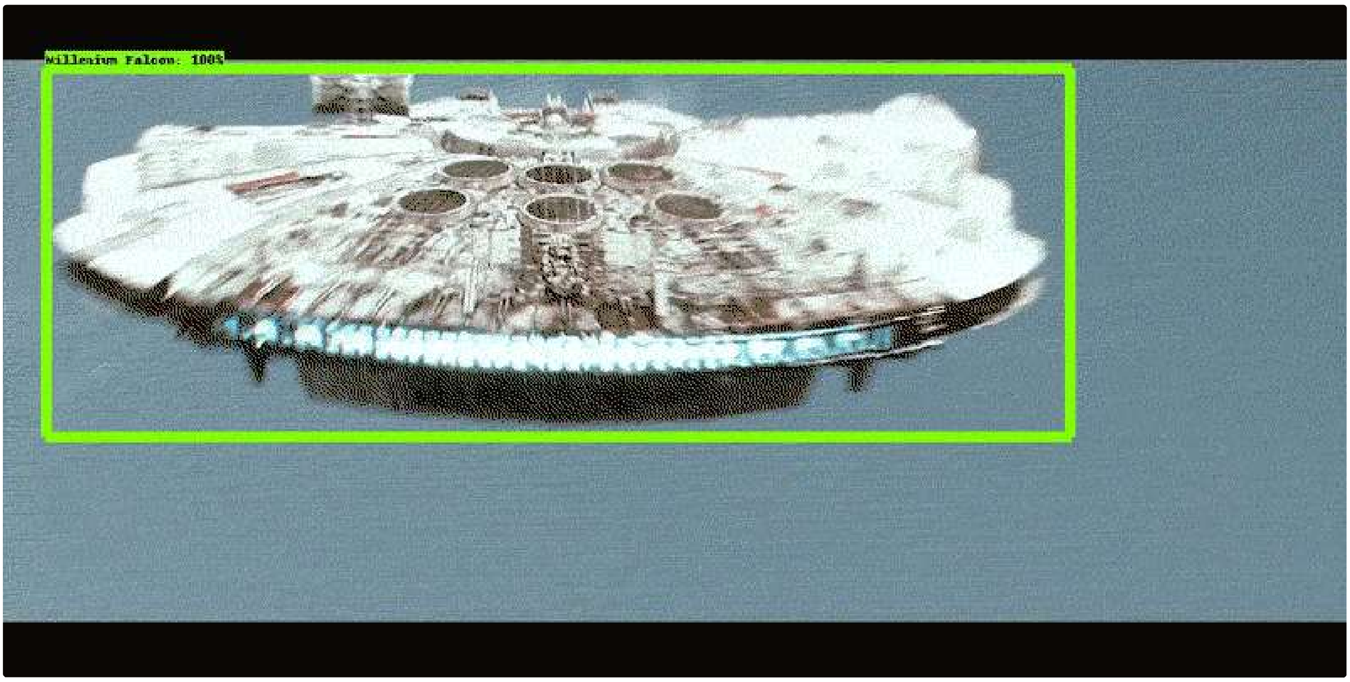
Follow

Written by Khush Patel

325 Followers · Writer for Towards Data Science

ML Engineer / Data Scientist | Master's in AI | DL & CV Nanodegree Holder | Facebook AI Scholar | Inventor | Int'l Medal Winner@USA | http://www.khushpatel.com

More from Khush Patel and Towards Data Science

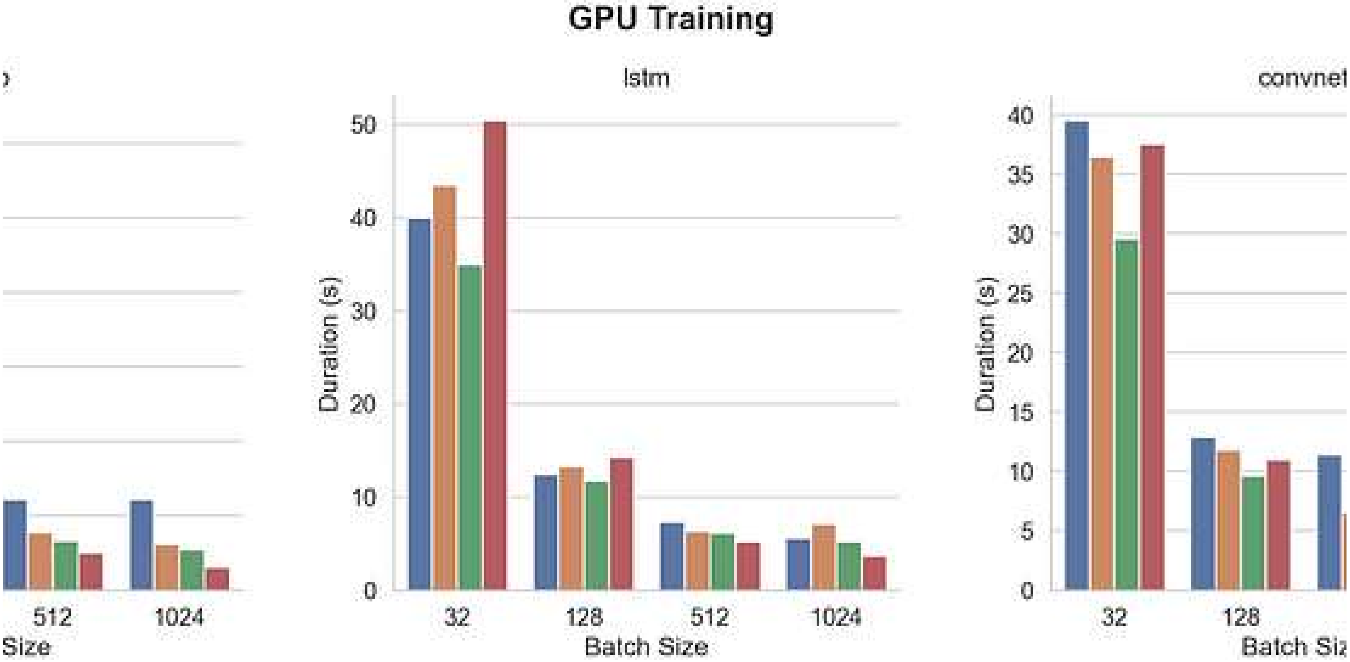


Khush Patel in Towards Data Science

Custom Object Detection using TensorFlow from Scratch

Custom Dataset Training for Object Detection using TensorFlow | Dog Detection in Real time Videos | Perfect Guide for Object Detection

· 8 min read · May 28, 2019



Fabrice Daniel in Towards Data Science

Apple M2 Max GPU vs Nvidia V100, P100 and T4

Compare Apple Silicon M2 Max GPU performances to Nvidia V100, P100, and T4 for training MLP, CNN, and LSTM models with TensorFlow.

· 6 min read · Nov 2

1K 17

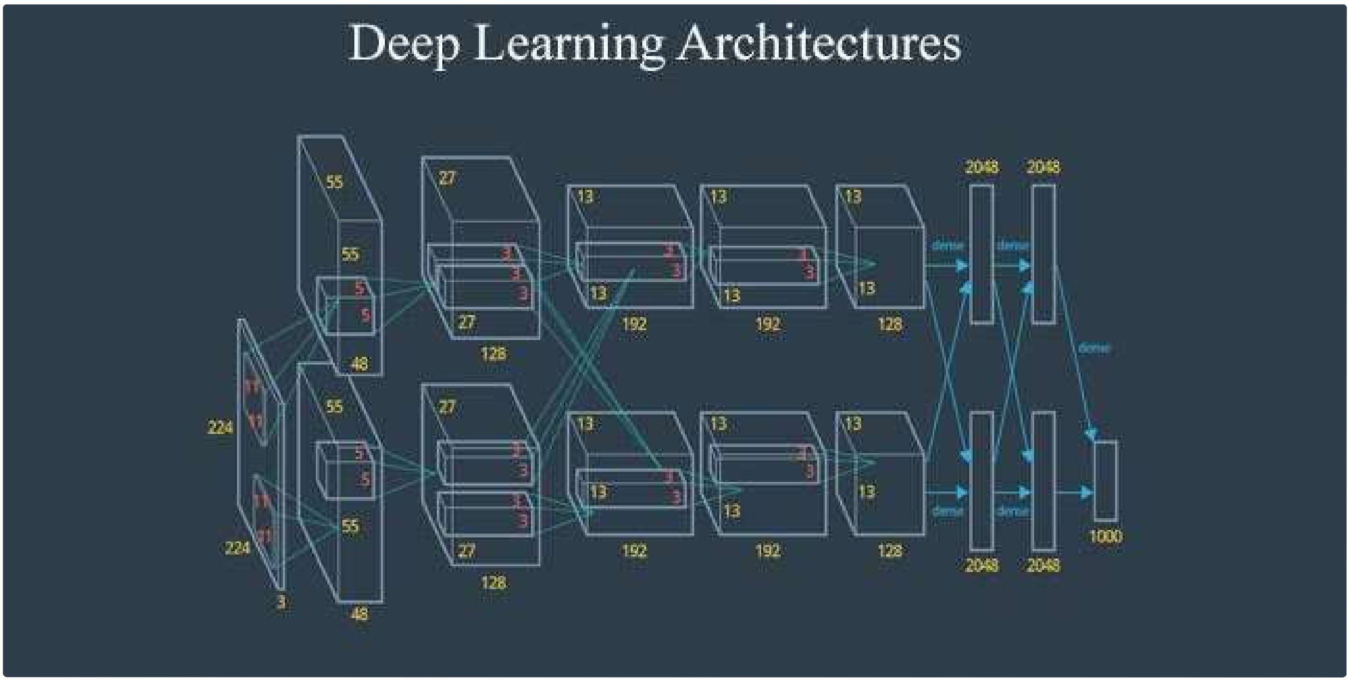
Khouloud El Alami in Towards Data Science

Don’t Apply to Tech Without Mastering These 6 Must-Have Data Science Skills—A Spotify Data…

A glimpse into the wizarding world of Tech & what you need to make your way in and succeed as a Data Wizard-ist

· 22 min read · Oct 27

966 8



Khush Patel in Towards Data Science

Architecture comparison of AlexNet, VGGNet, ResNet, Inception, DenseNet

Layers Description with Hyperparameter and accuracy in ILSVRC Challenge results

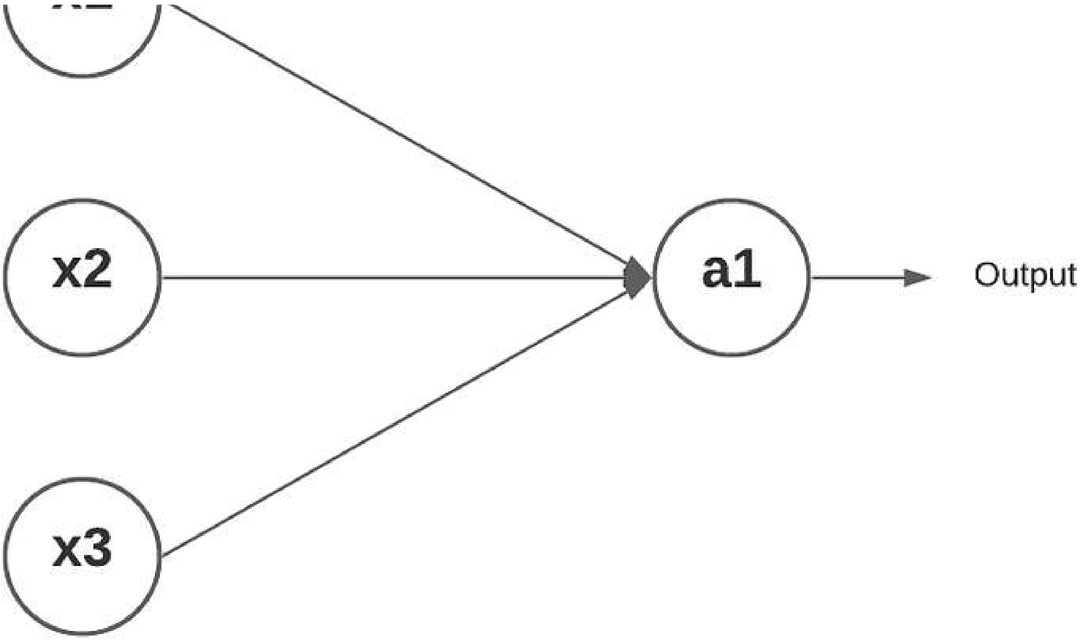
· 9 min read · Mar 8, 2020

331 1

See all from Khush Patel

See all from Towards Data Science

Recommended from Medium



Shashank Ravi

Introduction to Neural Networks : Build a Single Layer Perceptron in PyTorch

A neural network is a set of neuron nodes that are interconnected with one another. These connections extend not only to neighboring...

8 min read · Jul 24

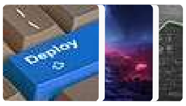
75 1

Ruman

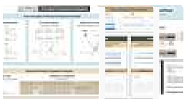
Part 1: Ultimate Guide to Fine-Tuning in PyTorch : Pre-trained Model and Its Configuration

Master model fine-tuning: Define pre-trained model, Modifying model head, loss functions, learning rate, optimizer, layer freezing, and...

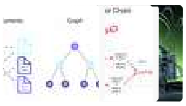
Lists



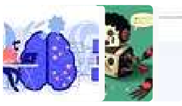
Predictive Modeling w/ Python



Practical Guides to Machine Learning



Natural Language Processing



The New Chatbots: ChatGPT, Bard, and Beyond

Anisatrop

In-depth Review: TensorFlow vs PyTorch

TensorFlow and PyTorch are two of the most popular machine learning libraries today, each with its own strengths and weaknesses. They both...

9 min read · Jul 7

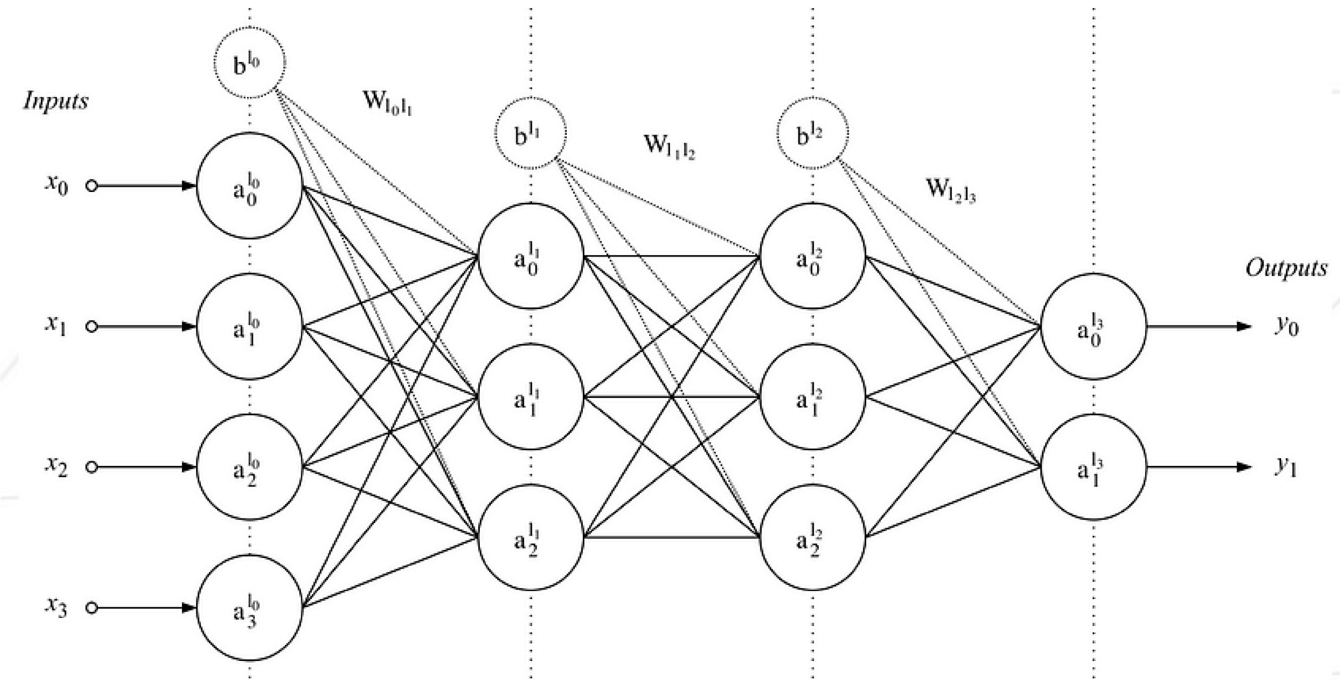
Lefteris Charteros

Scalable Project Structure for Machine Learning Projects with PyTorch and PyTorch Lightning

When embarking on a machine learning project, especially one that entails multiple components, it's easy to get caught up in the excitement

6 min read · Jun 14

18 1



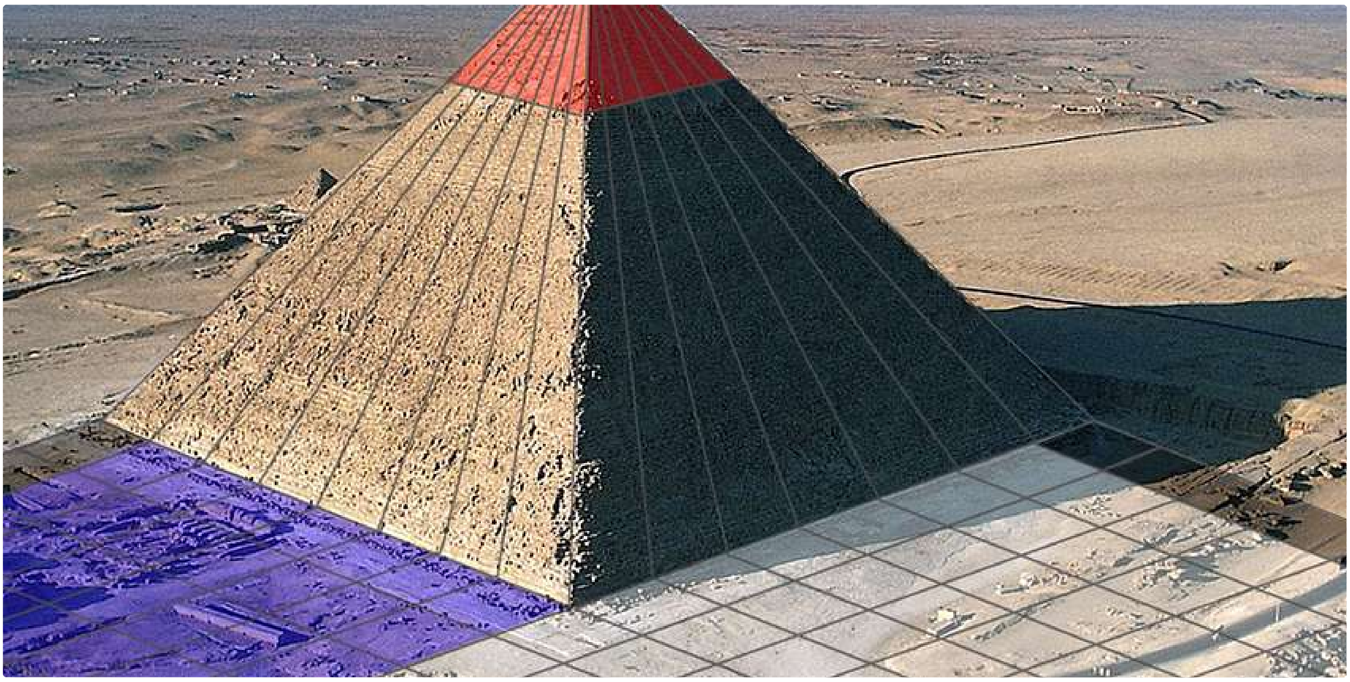
Hiroaki Kubo

Building a Multi Layer Perceptron from Scratch

I will share the code and description of how I built a multi-layer perceptron from scratch. The code is here.

5 min read · Aug 20

90



Edward Roe

Convolution: Image Filters, CNNs and Examples in Python & Pytorch

Introduction

12 min read · Jun 7

1

See more recommendations