DS620 Machine Learning and Deep Learning

HOS07 Exploding and Vanishing Gradients Problem

05/10/2021 Developed by Minh Nguyen

05/12/2021 Reviewed by Shanshan Yu

School of Technology & Computing @City University of Seattle (CityU)

Learning objectives

- Vanishing/Exploding Gradients
- Weight Initialization
- Non-saturating Activation Functions
- Batch Normalization
- Gradient Clipping

Resources

- Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Inc.
- Nielsen, M. A. (2015). Neural networks and deep learning. Neural Networks and Deep Learning. http://neuralnetworksanddeeplearning.com/chap2.html
- Batch Normalization ("batch norm") explained. (2018, January 18). [Video]. YouTube. https://www.youtube.com/watch?v=dXB-KQYkzNU

Introduction

In the previous module we learned about how a neural network learns while also learn how to build a Neural network with 2 layers on our own. When working with neural networks we can often find ourselves creating Neural networks with dozens of hidden layers. The more complex the network, the more problem may occur, one of which is the vanishing and exploding gradients problem. In this exercise, we will learn how to build a more complicated neural network while also learn how to tackle these problems.

I. The Vanishing/Exploding Gradient problem

Let's revise how the neural network started by assigning random weight and biases to the network. Then, it started its "learning" by adjusting its weight and bias so that it fits the underlying patterns of the data. These adjustments started from the last layers to the input layers through a process called Backpropagation. The adjustment to each weight and bias of the model is determined bases on the derivative of the loss function with respect to all the weights and biases of the model (the gradient). Then, through Gradient descent, the weight and bias are updated with each step.

However, when the number of hidden layers of the neural network increases it becomes more difficult to adjust the weight and biases of the layers of the layer further from the output layer. Some adjustments were so insignificant that it barely changes from the initial random assignment. This is what called the vanishing gradients problem. In some cases, the opposite happens where the weight experience drastic changes after each iteration, this is the result of unstable gradient.

The underlying problem that results in the vanishing/exploding gradient problem lies in how the data is transferred through each layer of the neural network. The computation of each neuron and the activation function led to the variance of the data become greater after each layer. There are 4 ways that this could be addressed.

1. Weight Initialization

Recalling the training process, the weight and bias of each layer were first randomly initialized. The input data get multiplied by the weight, added with the bias and passed through an activation function. However, even when the input data is standardized, its variance increase after this computation. This pose a problem as when all of this is passed through the activation function, the output becomes saturated. The gradient of this saturated output is extremely small thus resulting in the vanishing gradient problem.

Weight initialization goal is to keep the variance of each layer stable. It occurs on a per layer basis. It makes sures that when the random weights are initialized, they are created in a way that does not lead to extremely big or small weighted sum.

This can be done with the kernel_initializer parameter of the Dense class.

2. Non-Saturating Activation function

The choice of the activation function also led to the vanishing gradient problem as it makes the gradient extremely small. Choosing a less saturating activation function can resolve this problem.

3. Batch Normalization

When passing data to a model, the training process can be sped up if the data is on the same scale, this is what normalization do. However, in a neural network, the data is transferred through each layer, after each layer, the data is updated based on the weight and bias. This computation can cause instability in the scale of each neuron. Thus, creating the exploding gradient problem. To make sure this doesn't happen, Batch Normalization can normalize data at each layer.

4. Gradient Clipping

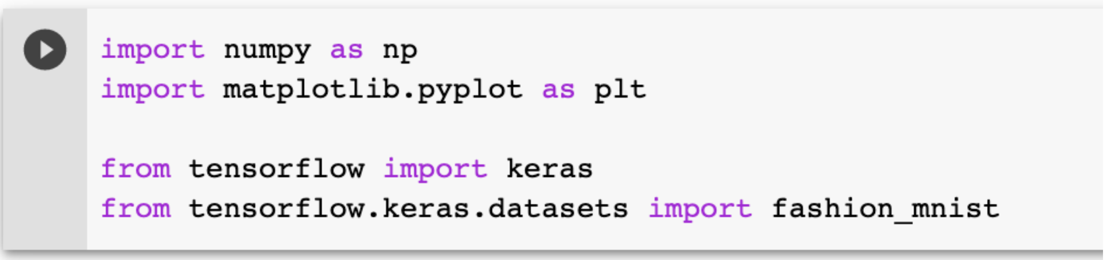
Exploding gradient is resulted from gradient being too large. Gradient clipping is a method for setting the gradient limit during Stochastic Gradient Descent so that it won't "explode"

II. Practice with Keras

In this exercise, we will work with the fashion mnist dataset, we will build a neural network adding Weight Initialization and Batch Normalization method to improve the model.

Preparing development environment

1. From Google Colab, create a new notebook, name it "DNN.ipynb"
2. Type the following codes to import libraries.

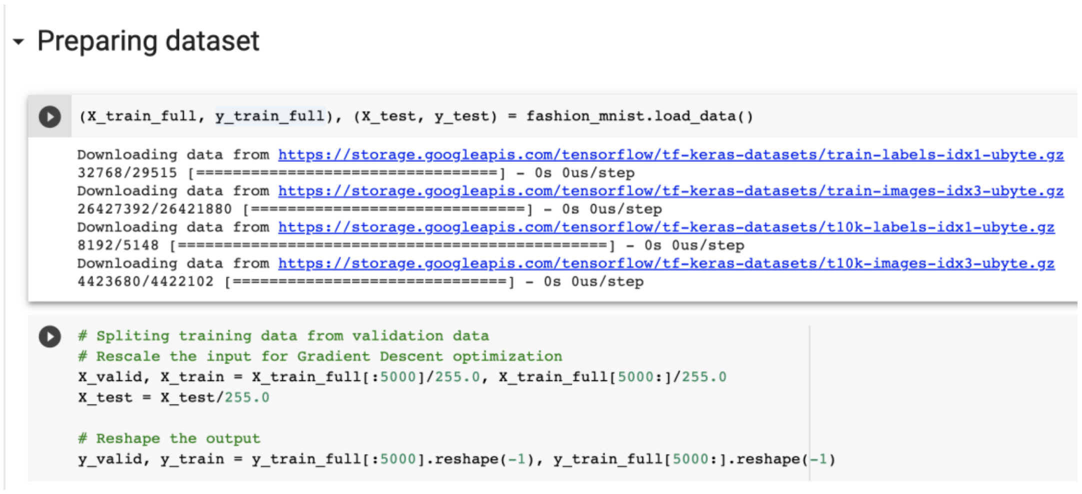


```
import numpy as np
import matplotlib.pyplot as plt

from tensorflow import keras
from tensorflow.keras.datasets import fashion_mnist
```

Preparing dataset

1. Add the following code to DNN.ipynb to prepare the fashion_mnist dataset.



Preparing dataset

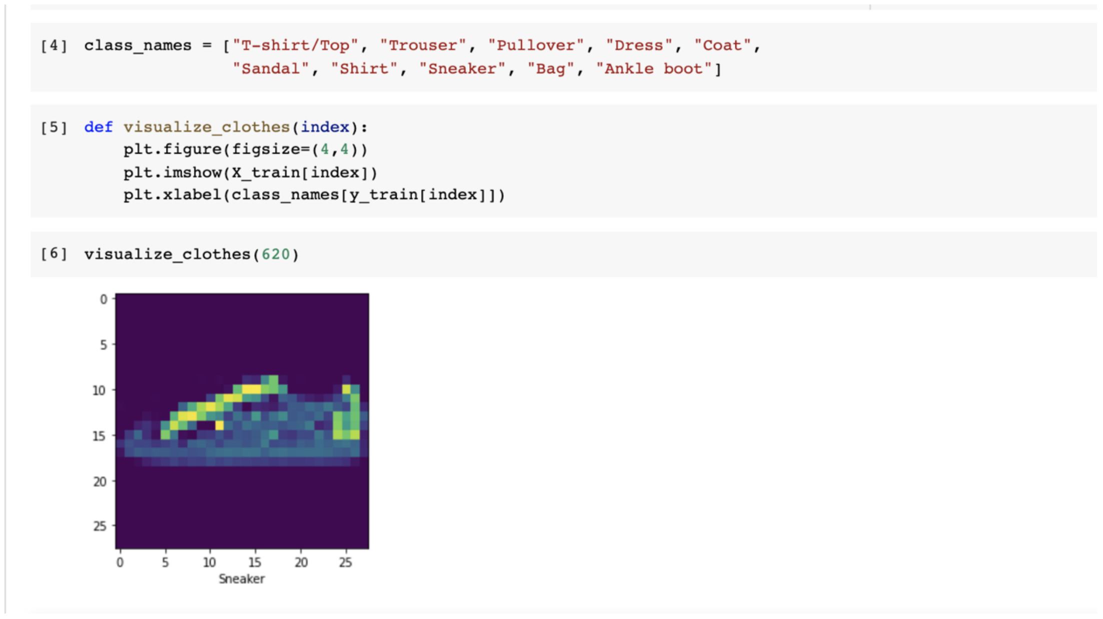
```
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
```

```
# Spliting training data from validation data
# Rescale the input for Gradient Descent optimization
X_valid, X_train = X_train_full[:5000]/255.0, X_train_full[5000:]/255.0
X_test = X_test/255.0

# Reshape the output
y_valid, y_train = y_train_full[:5000].reshape(-1), y_train_full[5000:].reshape(-1)
```

2. This dataset is quite similar to the Handwritten digit dataset but the label is drastically different. Therefore, visualizing the image with label need a little bit of extra work.



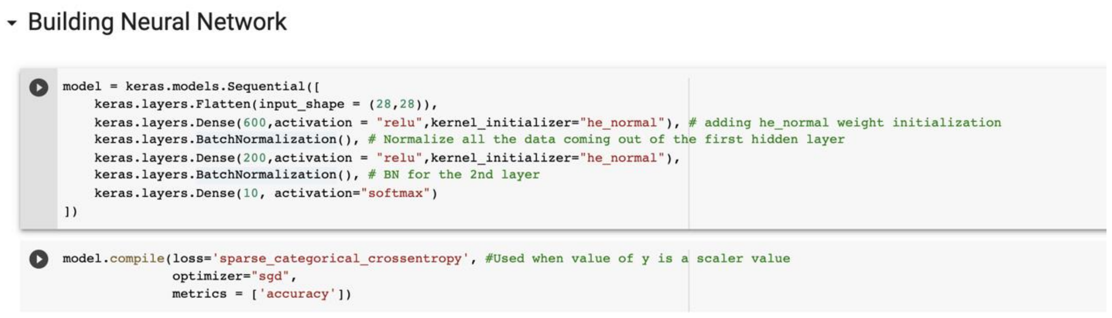
```
[4] class_names = ["T-shirt/Top", "Trouser", "Pullover", "Dress", "Coat",
                   "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

```
[5] def visualize_clothes(index):
        plt.figure(figsize=(4,4))
        plt.imshow(X_train[index])
        plt.xlabel(class_names[y_train[index]])
```

```
[6] visualize_clothes(620)
```

Modeling

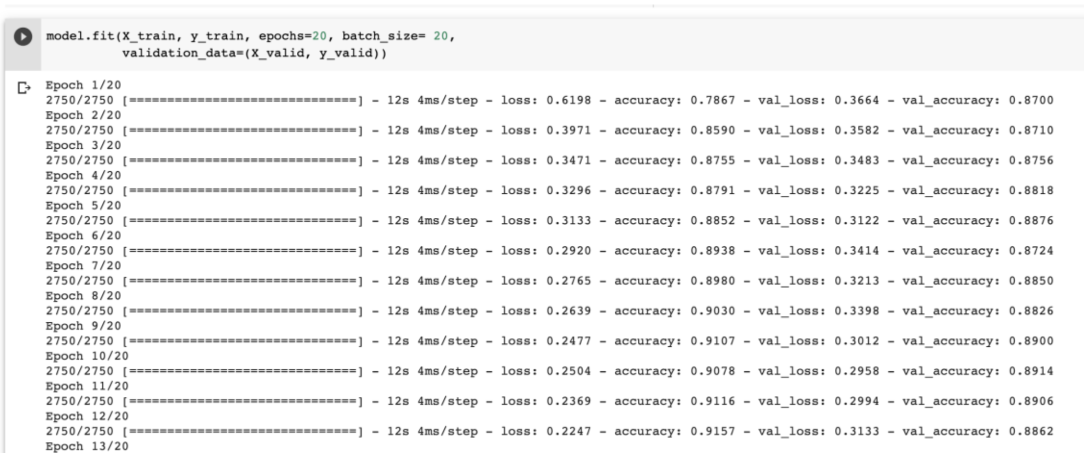
1. Type the following code to build the neural network with Batch Normalization and He weight initialization.



```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape = (28,28)),
    keras.layers.Dense(600,activation = "relu",kernel_initializer="he_normal"), # adding he_normal weight initialization
    keras.layers.BatchNormalization(), # Normalize all the data coming out of the first hidden layer
    keras.layers.Dense(200,activation = "relu",kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(), # BN for the 2nd layer
    keras.layers.Dense(10, activation="softmax")
])
```

```
model.compile(loss='sparse_categorical_crossentropy', #Used when value of y is a scaler value
              optimizer="sgd",
              metrics = ['accuracy'])
```

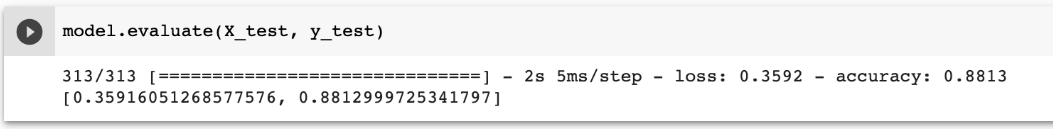
2. Train the model with 20 epochs and batch size of 20.



```
model.fit(X_train, y_train, epochs=20, batch_size= 20,
          validation_data=(X_valid, y_valid))
```

```
Epoch 1/20
2750/2750 [==============================] - 12s 4ms/step - loss: 0.6198 - accuracy: 0.7867 - val_loss: 0.3664 - val_accuracy: 0.8700
Epoch 2/20
2750/2750 [==============================] - 12s 4ms/step - loss: 0.3971 - accuracy: 0.8590 - val_loss: 0.3582 - val_accuracy: 0.8710
Epoch 3/20
2750/2750 [==============================] - 12s 4ms/step - loss: 0.3471 - accuracy: 0.8755 - val_loss: 0.3483 - val_accuracy: 0.8756
Epoch 4/20
2750/2750 [==============================] - 12s 4ms/step - loss: 0.3296 - accuracy: 0.8791 - val_loss: 0.3225 - val_accuracy: 0.8818
Epoch 5/20
2750/2750 [==============================] - 12s 4ms/step - loss: 0.3133 - accuracy: 0.8852 - val_loss: 0.3122 - val_accuracy: 0.8876
Epoch 6/20
2750/2750 [==============================] - 12s 4ms/step - loss: 0.2920 - accuracy: 0.8938 - val_loss: 0.3414 - val_accuracy: 0.8724
Epoch 7/20
2750/2750 [==============================] - 12s 4ms/step - loss: 0.2765 - accuracy: 0.8980 - val_loss: 0.3213 - val_accuracy: 0.8850
Epoch 8/20
2750/2750 [==============================] - 12s 4ms/step - loss: 0.2639 - accuracy: 0.9030 - val_loss: 0.3398 - val_accuracy: 0.8826
Epoch 9/20
2750/2750 [==============================] - 12s 4ms/step - loss: 0.2477 - accuracy: 0.9107 - val_loss: 0.3012 - val_accuracy: 0.8900
Epoch 10/20
2750/2750 [==============================] - 12s 4ms/step - loss: 0.2504 - accuracy: 0.9078 - val_loss: 0.2958 - val_accuracy: 0.8914
Epoch 11/20
2750/2750 [==============================] - 12s 4ms/step - loss: 0.2369 - accuracy: 0.9116 - val_loss: 0.2994 - val_accuracy: 0.8906
Epoch 12/20
2750/2750 [==============================] - 12s 4ms/step - loss: 0.2247 - accuracy: 0.9157 - val_loss: 0.3133 - val_accuracy: 0.8862
Epoch 13/20
```

3. Evaluate the model on testing set.



```
model.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.3592 - accuracy: 0.8813
[0.35916051268577576, 0.8812999725341797]
```

Push Your Work to GitHub

Download the notebook from Colab:

File -> DNN.ipynb

Move the downloaded file into your **Module7** working folder.

Open terminal and Navigate to the GitHub folder of this week HOS.

Make sure the assignment files on the subfolder Module7 of hos07a_YouGithubUserName folder, enter the following command to upload your work:

>>>> git add .

>>>> git commit -m "Submission for HOS07"

>>>> git push origin master