Vanishing / Exploding Gradient Problem [Habiba Shera]

- **Vanishing Gradient**
    - If the weights initialized are very small then in case of deep networks, for any activation function, (dW) will get smaller and smaller as we go backwards with every layer during back propagation. **“ leaves the weights of the initial or lower layers nearly unchanged. “**

- **Exploading Gradient**
    - the gradients keep on getting larger and larger as the backpropagation algorithm progresses. This, in turn, **causes very large weight updates and causes the gradient descent to diverge.**
    - *This problem happens because of weights, not because of the activation function*

How to know if our model is suffering from the Exploding/Vanishing gradient problem?

- **<u>For Vanishing</u>**
    - The parameters of the higher layers change significantly whereas the **parameters of lower layers would not change much (or not at all).**
    - The model **weights may become 0 during training.**

- **<u>For Exploding</u>**
    - There is an **exponential growth in the model parameters.**
    - The model **weights may become NaN during training.**

Solutions

Initialization	Activation functions	σ^2 (Normal)
Glorot	None, Tanh, Logistic, Softmax	$1 / fan_{avg}$
He	ReLU & variants	$2 / fan_{in}$
LeCun	SELU	$1 / fan_{in}$

Initialize weights with these techniques, For exmaple :

- while using T**anh, softmax, sigmoid**, use **glorot technique** (called **Xavier initialization**)to initialize weights.
- while using **ReLU**, use **He technique**

Coding Examples

```
tf.keras.layers.Dense(25, activation = "relu", kernel_initializer="he_normal")
```

```
tf.keras.layers.Dense(25, activation = "relu", kernel_initializer="he_uniform"
```

```
# He Uniform Initialization
from tensorflow.keras import layers
from tensorflow.keras import initializers

initializer = tf.keras.initializers.HeUniform()
layer = tf.keras.layers.Dense(3, kernel_initializer=initializer)
```

> **ReLU** activation function can reduce the chances of **vanishing/exploding problems** at the beginning. However, it does not guarantee that the problem won't reappear during training.

So, there is another technique known as **Batch Normalization** to address the **problem of vanishing/exploding gradients.**

- **Batch Normalization**
    - It’s **adding an operation** in the model just **before** or **after** the activation function of each hidden layer.
    - This operation is **zero-centers and normalizes each input**, then scales and shifts the result using two new parameter vectors per layer: one for scaling, the other for shifting.

```
tf.keras.layers.Dense(300, activation="relu"),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Dense(100, activation="relu"),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Dense(10, activation="softmax")])
```

- **Gradient Clipping**
    - is a technique for **preventing exploding gradients** in **RNN**
    - let the original gradient vector be **[0.9, 100.0], once we clip it by some value, we get [0.9, 1.0] which now points somewhere around the diagonal between the two axes.**

```
optimizer = keras.optimizers.SGD(clipvalue = 1.0)
```

```
optimizer = keras.optimizers.SGD(clipnorm = 1.0)
```

While using SGD without clipvalue

```
model.compile(
    # inside the optimizer we are doing clipping
    optimizer=tf.keras.optimizers.SGD())
```

```
Epoch 1/10
500/500 [==============================] - 6s 8ms/step - loss: 2.2427 - sparse_categorical_accuracy: 0.2153
Epoch 2/10
500/500 [==============================] - 4s 8ms/step - loss: 2.0552 - sparse_categorical_accuracy: 0.2972
Epoch 3/10
500/500 [==============================] - 4s 8ms/step - loss: 1.8755 - sparse_categorical_accuracy: 0.2979
Epoch 4/10
500/500 [==============================] - 4s 8ms/step - loss: 1.6283 - sparse_categorical_accuracy: 0.4078
Epoch 5/10
500/500 [==============================] - 4s 8ms/step - loss: 1.6672 - sparse_categorical_accuracy: 0.3919
Epoch 6/10
500/500 [==============================] - 4s 8ms/step - loss: 1.5099 - sparse_categorical_accuracy: 0.4269
Epoch 7/10
500/500 [==============================] - 4s 7ms/step - loss: 1.4032 - sparse_categorical_accuracy: 0.4574
Epoch 8/10
500/500 [==============================] - 4s 8ms/step - loss: 1.3121 - sparse_categorical_accuracy: 0.5122
Epoch 9/10
500/500 [==============================] - 4s 8ms/step - loss: 1.2185 - sparse_categorical_accuracy: 0.5649
Epoch 10/10
500/500 [==============================] - 4s 8ms/step - loss: 1.1252 - sparse_categorical_accuracy: 0.5928
```

While using SGD with clipvalue

```
optimizer=tf.keras.optimizers.SGD(clipvalue=0.5)
```

```
Epoch 1/10
500/500 [==============================] - 6s 8ms/step - loss: 2.2395 - sparse_categorical_accuracy: 0.1893
Epoch 2/10
500/500 [==============================] - 4s 8ms/step - loss: 2.0502 - sparse_categorical_accuracy: 0.3863
Epoch 3/10
500/500 [==============================] - 4s 8ms/step - loss: 1.7738 - sparse_categorical_accuracy: 0.4928
Epoch 4/10
500/500 [==============================] - 4s 8ms/step - loss: 1.4923 - sparse_categorical_accuracy: 0.5601
Epoch 5/10
500/500 [==============================] - 4s 8ms/step - loss: 1.2820 - sparse_categorical_accuracy: 0.5875
Epoch 6/10
500/500 [==============================] - 4s 8ms/step - loss: 1.1151 - sparse_categorical_accuracy: 0.6326
Epoch 7/10
500/500 [==============================] - 4s 8ms/step - loss: 0.9977 - sparse_categorical_accuracy: 0.6755
Epoch 8/10
500/500 [==============================] - 4s 8ms/step - loss: 0.9157 - sparse_categorical_accuracy: 0.7023
Epoch 9/10
500/500 [==============================] - 4s 8ms/step - loss: 0.8679 - sparse_categorical_accuracy: 0.7109
Epoch 10/10
500/500 [==============================] - 4s 8ms/step - loss: 0.8187 - sparse_categorical_accuracy: 0.7203
```

Conclusion

- **Vanishing gradients** :
  - ReLU as the activation function
  - Reduce the model complexity
  - Weight initializer with variance (**He technique**)

- Better optimizer with a well-tuned learning rate

- **Exploding gradients**
    - Gradients clipping
    - Proper weight initializer
    - L2 norm regularization

References

- Vanishing and Exploding Gradients in Deep Neural Networks (analyticsvidhya.com)
- How can gradient clipping help avoid the exploding gradient problem? (analyticsindiamag.com)