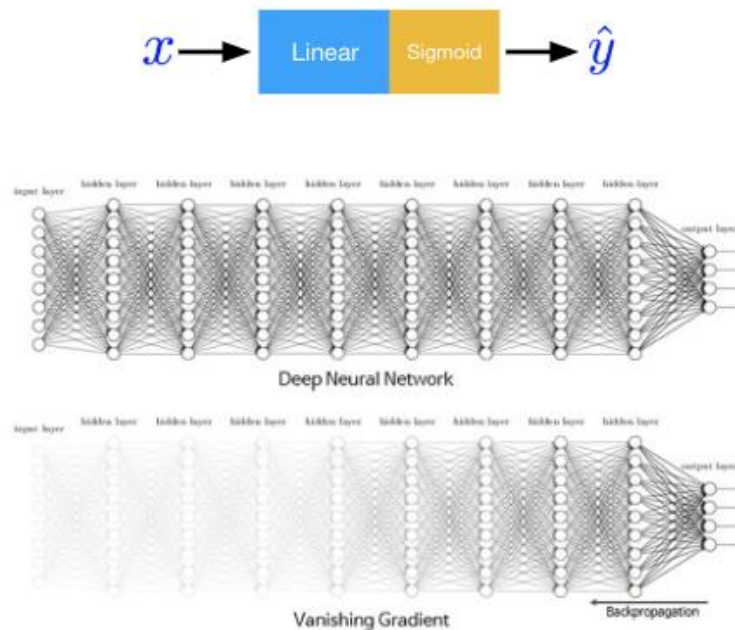


Vanishing Gradient Problem in Deep Learning: Understanding, Intuition, and Solutions

Sigmoid: Vanishing Gradient Problem



Introduction

Deep Learning has rapidly become an integral part of modern AI applications, such as computer vision, natural language processing, and speech recognition. The success of deep learning is attributable to its ability to automatically learn complex patterns from large amounts of data without explicit programming. Gradient-based optimization, which relies on backpropagation, is the primary technique used to train deep neural networks (DNNs).

Understanding the Vanishing Gradient Problem


One of the major roadblocks in training DNNs is the vanishing gradient problem, which occurs when the gradients of the loss function with respect to the weights of the early layers become vanishingly small. As a result, the early layers receive little or no updated weight information

during backpropagation, leading to slow convergence or even stagnation. The vanishing gradient problem is mostly attributed to the choice of activation functions and optimization methods in DNNs.

Vanishing Gradient Problem
Thursday, April 7, 2022 7:45 AM

In machine learning, the **vanishing gradient problem** is encountered when training artificial neural networks with gradient-based learning methods and backpropagation. In such methods, during each iteration of training each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training. A classic example of the problem is the sigmoid-tanh problem

1) $0.1 \times 0.1 \times 0.1 \times 0.1 = 0.0001 \rightarrow \text{VGP}$

2) Deep NN \rightarrow 

3) sigmoid / tanh \rightarrow Af \rightarrow

Vanishing gradient problems generally occurs when the value of partial derivative of loss function w.r.t. to weights are very small. The complexity of Deep Neural Networks also causes VD problem.

Role of Activation Functions and Backpropagation

Activation functions, such as sigmoid and hyperbolic tangent, are responsible for introducing non-linearity into the DNN model. However, these functions suffer from the saturation problem, where the gradients become close to zero for large or small inputs, contributing to the vanishing gradient problem. Backpropagation, which computes the gradients of the loss function with respect to the weights in a chain rule fashion, exacerbates the problem by multiplying these small gradients.

Backprop

$0 - 0$

x_{i1}, x_{i2}

$\hat{y} - y = L$

sigmoid derivative

$w_{11} = w_0 - \eta \left\{ \frac{\partial L}{\partial w_{11}} \right\}$ derivative of L wrt weight

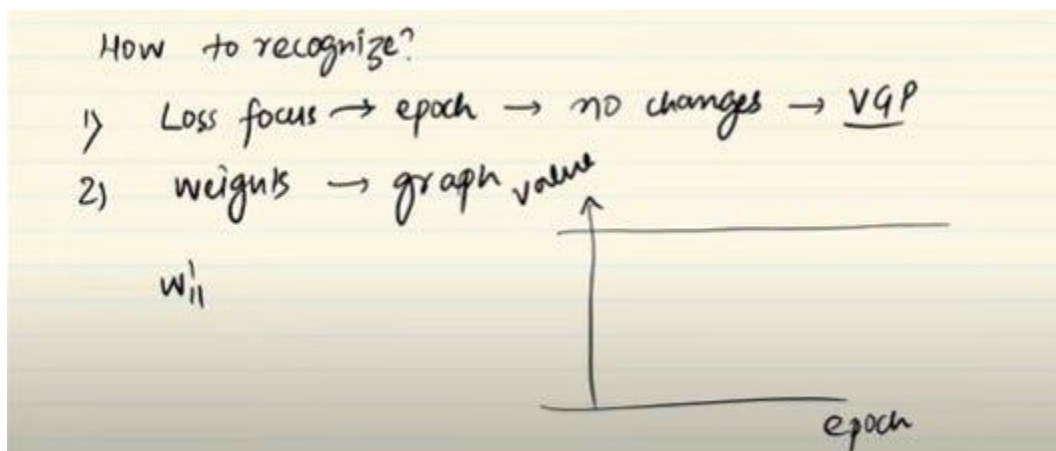
$\left[\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w_{11}} \right]$

During backpropagation we calculate loss ($y - \hat{y}$) and update our weights using partial derivatives of loss function but it follows chain rule and to update w_{11} there will be a sequence of chain with

multiplication of smaller values of gradient descent and learning rate which in result no change in weights .

How to recognise Vanishing Gradient Problem

1. Calculate loss using Keras and if its consistent during epochs that means Vanishing Gradient Problem.
2. Draw the graphs between weights and epochs and if it is constant that means weight has not changed and hence vanishing gradient problem.



```
#importing libraries
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
import keras
from sklearn.datasets import make_moons #classification datasets
from sklearn.model_selection import train_test_split
from keras.layers import Dense
from keras.models import Sequential
```

```
X,y = make_moons(n_samples=250, noise=0.05, random_state=42)
```

```
plt.scatter(X[:,0],X[:,1], c=y, s=100)
plt.show()
```

```
model = Sequential()
```

```
#constructing a complex neural network with two inputs an nine layers with 10 nodes
```

```
model.add(Dense(10,activation='sigmoid',input_dim=2))
```

```

model.add(Dense(10,activation='sigmoid'))
model.add(Dense(10,activation='sigmoid'))
model.add(Dense(10,activation='sigmoid'))
model.add(Dense(10,activation='sigmoid'))
model.add(Dense(10,activation='sigmoid'))
model.add(Dense(10,activation='sigmoid'))
model.add(Dense(10,activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'
])

model.get_weights()[0]

old_weights = model.get_weights()[0]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=42)

model.fit(X_train, y_train, epochs = 100)

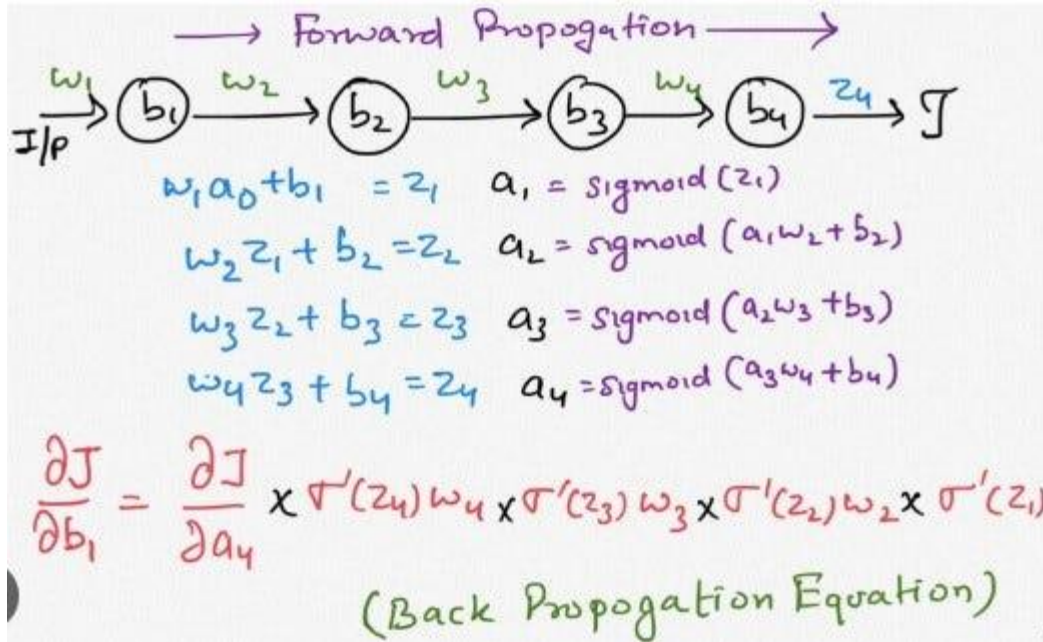
new_weights = model.get_weights()[0]

#Similarly we can check the changes in weights using relu function

```

Mathematical Intuition behind the Vanishing Gradient Problem

The vanishing gradient problem arises from the product of the Jacobian matrices of the activation functions and the weights of the DNN layers. Each layer contributes a Jacobian matrix to the product, and the product becomes rapidly small as the number of layers increases. The problem is more severe when the Jacobian matrices have small eigenvalues, which happens when the activation functions are near-saturated or the weights are poorly initialized.



Chain rule applied when finding the partial derivative of loss function w.r.to. to b_1

Reducing Complexity

The problem of vanishing gradients can be mitigated by reducing the complexity of the DNN model. Complexity reduction can be achieved by reducing the number of layers or the number of neurons in each layer. While this can alleviate the vanishing gradient problem to some extent, it would also lead to reduced model capacity and performance.

Impact of Network Architecture

To preserve model capacity while mitigating the vanishing gradient problem, researchers have explored various network architectures, such as skip connections, convolutional neural networks (CNNs), and recurrent neural networks (RNNs). These architectures are designed to facilitate better gradient flow and information propagation across layers, thereby reducing the vanishing gradient problem.

Trade-off between Model Complexity and the Vanishing Gradient Problem

There is a trade-off between model complexity and the vanishing gradient problem. While a more complex model might lead to better performance, it is also more prone to the vanishing gradient problem due to the increased depth and non-linearity. Finding a balance between model complexity and gradient flow is critical in building successful DNN models.

Using ReLU Activation Function

Rectified Linear Unit (ReLU) is a popular activation function that has gained popularity due to its simplicity and effectiveness in DNNs. ReLU has a non-saturating activation function, which ensures that the gradients flow freely across the network.

Benefits of ReLU

ReLU addresses the vanishing gradient problem by preventing gradient saturation. ReLU has been shown to significantly improve the convergence speed and accuracy of DNNs. Additionally, ReLU has computational advantages over other activation functions, making it a popular choice in practice.

Examples and Empirical Evidence of ReLU's Effectiveness

ReLU has been widely used in state-of-the-art DNN models, such as ResNet, Inception, and VGG. Empirical evidence shows that ReLU achieves better performance than other activation functions, such as sigmoid and hyperbolic tangent.

Batch Normalization

Batch normalization is another technique that has shown great success in mitigating the vanishing gradient problem and improving the training of DNNs.

Definition and Role of Batch Normalization

Batch normalization involves normalizing the input to each layer to have zero mean and unit variance. This improves the gradient flow and allows faster convergence of the optimization algorithm. Additionally, batch normalization acts as a regularization technique, enabling the model to generalize better.

Benefits of Batch Normalization

Research has shown that batch normalization leads to faster convergence and improved generalization of DNN models. Batch normalization is also robust to changes in hyperparameters and improves the stability of the training process.

Proper Weight Initialization

Weight initialization plays a crucial role in training DNNs, even before the optimization algorithm optimizes the weights.

Impact of Weight Initialization

Poor weight initialization can result in gradient explosion or vanishing, making the network difficult or impossible to learn. Well-chosen weight initialization methods can alleviate the vanishing gradient and enable faster convergence without sacrificing model capacity.

Techniques for Weight Initialization

Researchers have proposed various weight initialization techniques, such as Xavier and He initialization, for different activation functions. These methods are designed to ensure the proper scaling of the gradients during backpropagation, thereby facilitating better gradient flow.

Residual Networks (ResNets)

Residual Networks, commonly referred to as ResNets, are a popular type of neural network that have shown remarkable performance in deep learning.

Introduction and Architecture of ResNets

The architecture of ResNets introduces skip connections, where the output of one layer is added to the input of another layer, facilitating better gradient flow and avoiding the vanishing gradient problem. The skip connections also enable the network to learn residual (or residual error) functions, making training deeper models easier.

Advantages of ResNets

ResNets have been used to achieve state-of-the-art performance in various domains, such as image recognition, object detection, and natural language processing. The skip connections in ResNets have shown to effectively alleviate the vanishing gradient problem and make training easier.