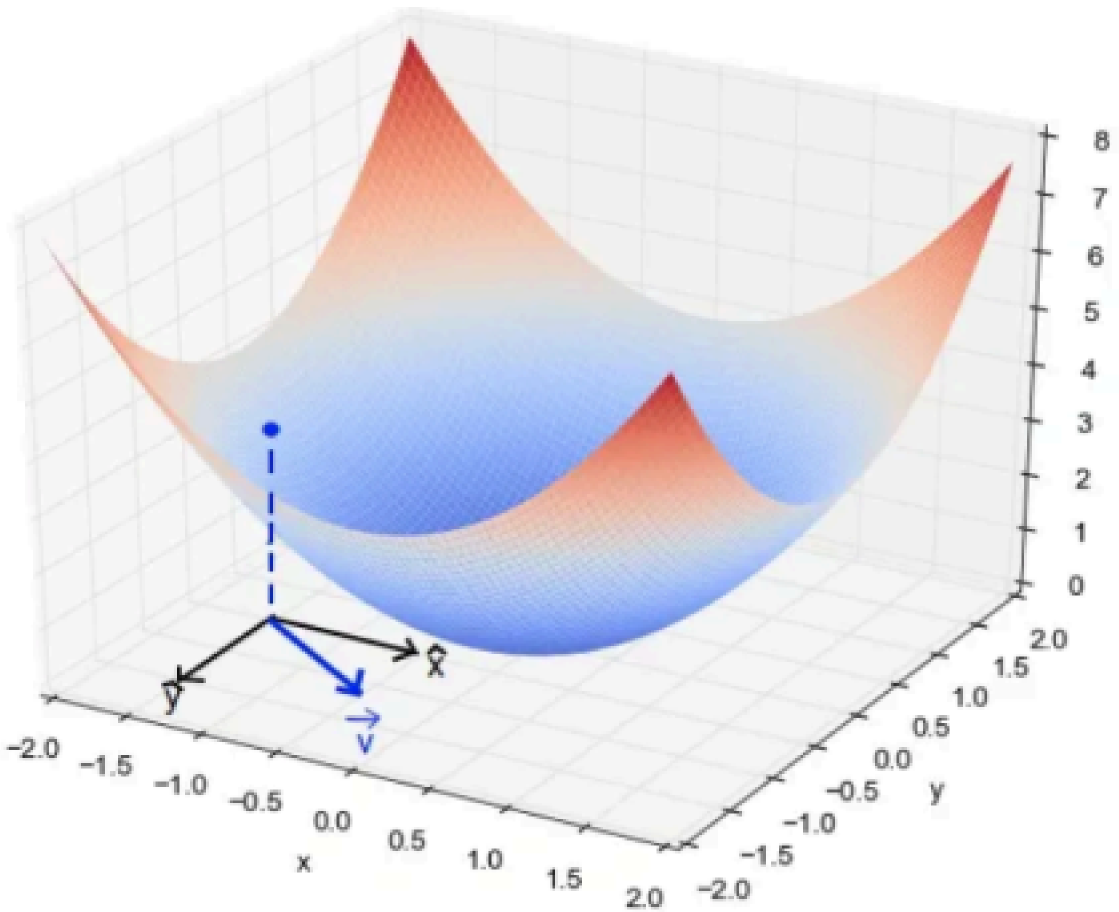
Gradient Descent

Gradient descent is one of the most popular algorithm to perform optimization and by far the most common way to optimize neural networks.
Gradient descent is a way to minimize an objective function j(θ) parameterized by a model's parameters
θ∈Rd by updating the parameter in the opposite direction of the gradient of the objective function ΔθJ(θ) w.r.t to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley.



Types of Gradient Descent

1. Batch gradient descent:

Also known as vanilla gradient descent, is the simplest variant of gradient descent. In batch gradient descent, ***the entire training dataset is used to compute the gradients of the cost function with respect to the model parameters in each iteration.*** This can be computationally expensive for large datasets, but it guarantees convergence to a local minimum of the cost function.

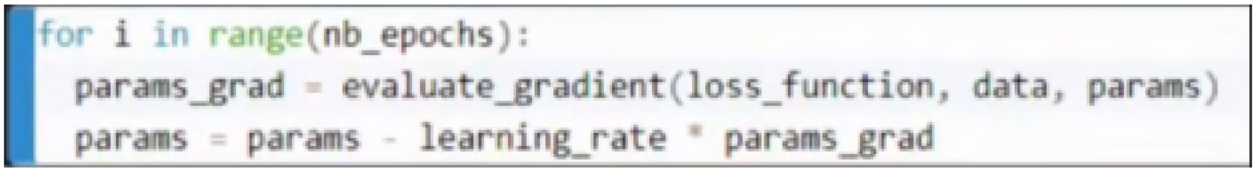
$\theta = \theta - \alpha \cdot \nabla J(\theta)$

where:

θ is the parameter vector

α is the learning rate

$\nabla J(\theta)$ is the gradient of the cost function J with respect to θ



```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(loss_function, data, params)
  params = params - learning_rate * params_grad
```

2. Stochastic Gradient Descent (SGD):

Stochastic gradient descent is a variant of gradient descent that updates the model parameters for each training example in the dataset. Unlike batch gradient descent, which uses the entire dataset to compute the gradients, ***SGD updates the parameters based on a randomly selected training example.*** *This can lead to faster convergence because the* *updates are more frequent and noisy, but it can also result in more oscillations in the cost function due to the randomness of the updates.*

$\theta = \theta - \alpha \cdot \nabla Ji(\theta)$

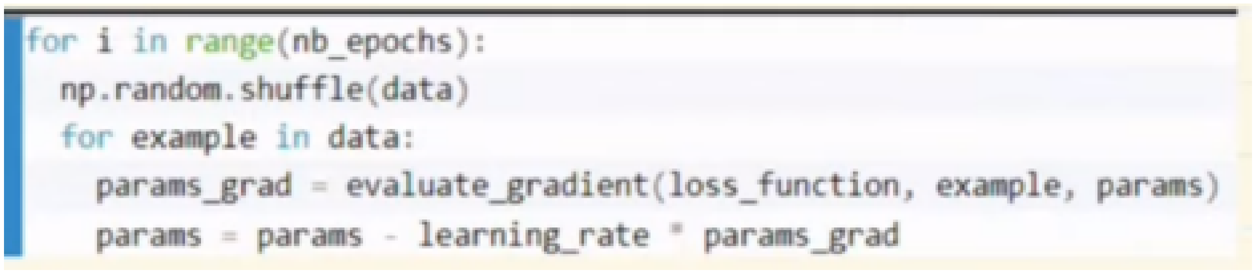
where:

θ is the parameter vector

α is the learning rate

$\nabla Ji(\theta)$ is the gradient of the cost function J with respect to θ computed on

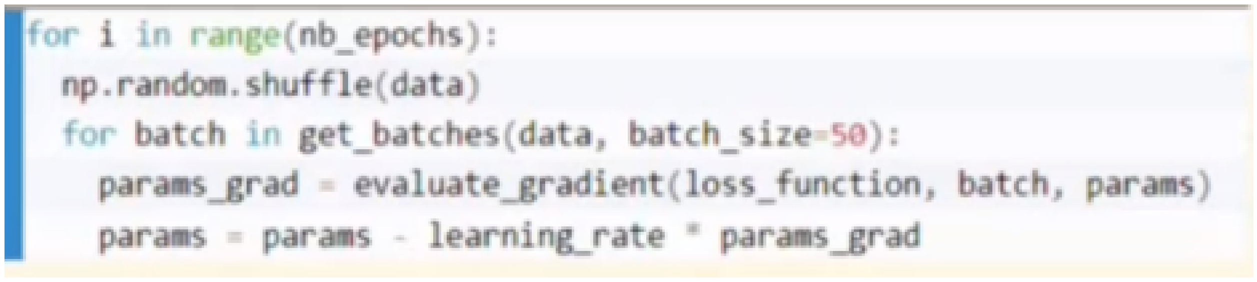
a single randomly selected training example i



```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function, example, params)
    params = params - learning_rate * params_grad
```

3. Mini-batch Gradient Descent:

Mini-batch gradient descent is a compromise between batch gradient descent and stochastic gradient descent. In mini-batch gradient descent, ***the gradients are computed on a small random subset of the training dataset, typically between 10 and 1000 examples, called a mini-batch****. This reduces the computational cost of the algorithm compared to batch gradient descent, while also reducing the variance of the updates compared to SGD. Mini-batch gradient descent is widely used in deep learning because it* *strikes a good balance between convergence speed and stability.*

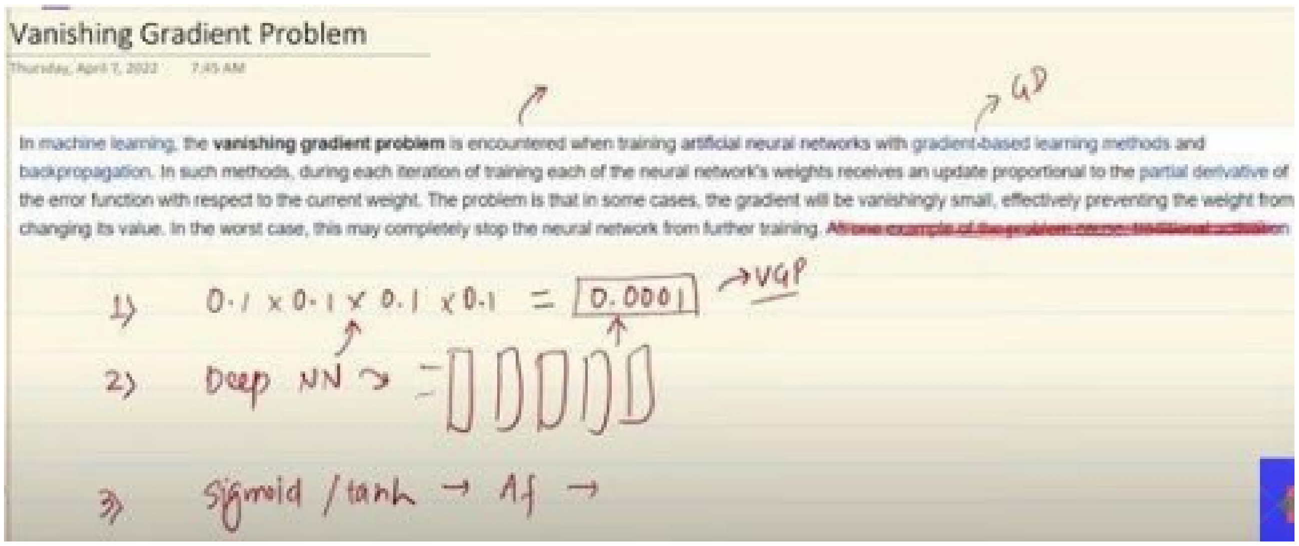


```python
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data, batch_size=50):
    params_grad = evaluate_gradient(loss_function, batch, params)
    params = params - learning_rate * params_grad
```

Gradient_descent.ipynb

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

GD

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. ~~As one example of the problem cause, traditional activation~~

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

2) Deep NN $\rightarrow$

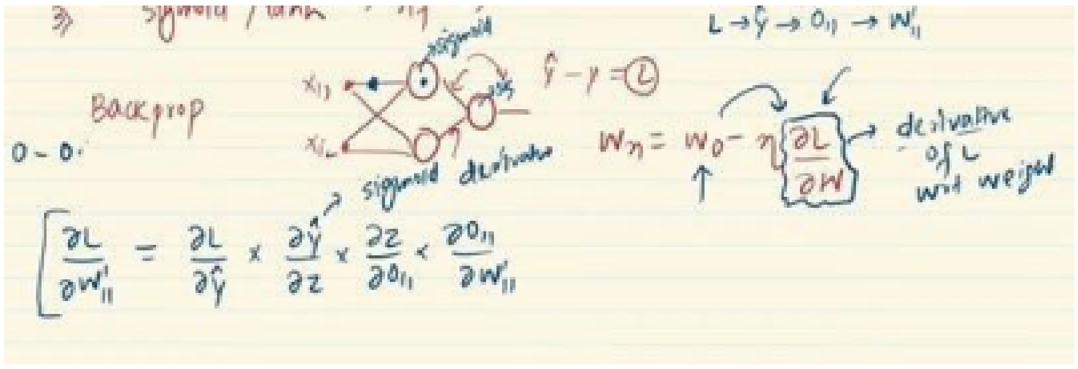
3) Sigmoid / tanh $\rightarrow$ Af $\rightarrow$

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,

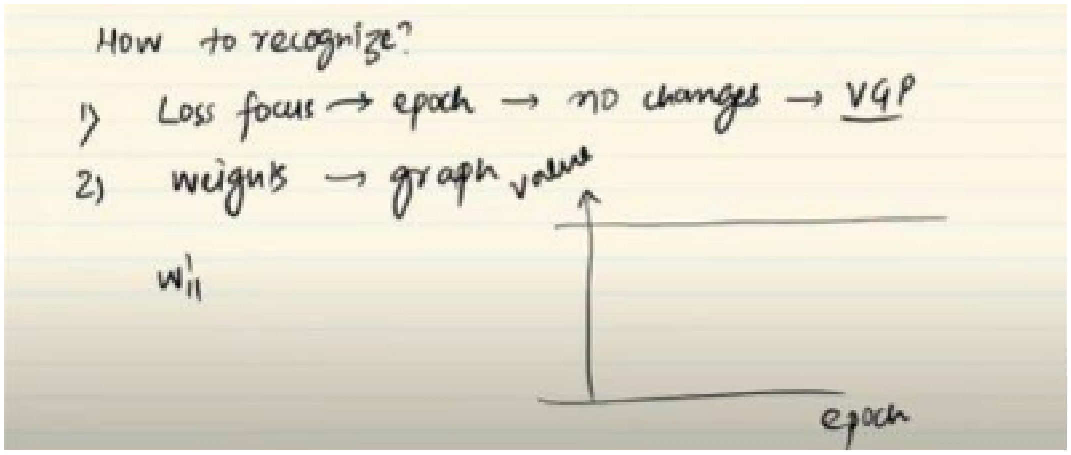
exacerbates the problem by multiplying these small gradients.



During backpropagation we calculate loss (y-y_hat) and update our weights using partial derivatives of loss function but it follows chain rule and to update w11 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.



How to identify Vanishing Gradient Problem.��

vanishing_gradient_descent.ipynb

What is Exploding Gradient?

The exploding gradient problem is a challenge encountered during training deep neural networks. It occurs when the gradients of the network's loss function with respect to the weights (parameters) become excessively large.

Why Exploding Gradient Occurs?

The issue of exploding gradients arises when, during backpropagation, the derivatives or slopes of the neural network's layers grow progressively larger
as we move backward. This is essentially the opposite of the vanishing gradient problem.

The root cause of this problem lies in the weights of the network, rather than the choice of activation function. High weight values lead to correspondingly high derivatives, causing significant deviations in new weight values from the previous ones. As a result, the gradient fails to converge and can lead to the network oscillating around local minima, making it challenging to reach the global minimum point.

In summary, exploding gradients occur when weight values lead to excessively large derivatives, making convergence difficult and potentially preventing the neural network from effectively learning and optimizing its parameters.

The exploding gradient problem occurs when the gradients become very large during backpropagation. This is often the result of gradients greater than 1, leading to a rapid increase in values as you propagate them backward through the layers.

How can we identify the problem?

Identifying the presence of exploding gradients in deep neural network requires careful observation and analysis during training. Here are some key indicators:

- The loss function exhibits erratic behavior, oscillating wildly instead of steadily decreasing suggesting that the network weights are being updated excessively by large gradients, preventing smooth convergence.
- The training process encounters "NaN" (Not a Number) values in the loss function or other intermediate calculations..
- If network weights, during training exhibit significant and rapid increases in their values, it suggests the presence of exploding

gradients.

- Tools like TensorBoard can be used to visualize the gradients flowing through the network.