**Deep Learning Assignment**

**Y20AIT457**

**1.Explain about Adam Optimizer?**

Adam optimizer stands for Adaptive Moment Estimation and it is an iterative optimization algorithm used to minimize the loss function during the training of neural networks.

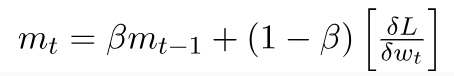
[Adam is an **adaptive learning rate algorithm designed to improve training speeds in deep neural networks and reach convergence quickly**](https://www.bing.com/ck/a?!&&p=780dfc4b59e206b6JmltdHM9MTcwNjc0NTYwMCZpZ3VpZD0wYTA2ZTgzYi1kZGQzLTY4YmEtMjI5MS1mODQ0ZGM3ZTY5YjgmaW5zaWQ9NTc2NQ&ptn=3&ver=2&hsh=3&fclid=0a06e83b-ddd3-68ba-2291-f844dc7e69b8&psq=about+adam+in+deep+learning&u=a1aHR0cHM6Ly9idWlsdGluLmNvbS9tYWNoaW5lLWxlYXJuaW5nL2FkYW0tb3B0aW1pemF0aW9u&ntb=1). [It customizes each parameter’s learning rate based on its gradient history, and this adjustment helps the neural network learn efficiently as a whole](https://www.bing.com/ck/a?!&&p=fd8a59ee28fd0398JmltdHM9MTcwNjc0NTYwMCZpZ3VpZD0wYTA2ZTgzYi1kZGQzLTY4YmEtMjI5MS1mODQ0ZGM3ZTY5YjgmaW5zaWQ9NTc2OQ&ptn=3&ver=2&hsh=3&fclid=0a06e83b-ddd3-68ba-2291-f844dc7e69b8&psq=about+adam+in+deep+learning&u=a1aHR0cHM6Ly9idWlsdGluLmNvbS9tYWNoaW5lLWxlYXJuaW5nL2FkYW0tb3B0aW1pemF0aW9u&ntb=1). [Adam is a powerful optimization algorithm that has become a default choice for training deep neural networks](https://www.bing.com/ck/a?!&&p=92f17a71f2714d19JmltdHM9MTcwNjc0NTYwMCZpZ3VpZD0wYTA2ZTgzYi1kZGQzLTY4YmEtMjI5MS1mODQ0ZGM3ZTY5YjgmaW5zaWQ9NTc3MQ&ptn=3&ver=2&hsh=3&fclid=0a06e83b-ddd3-68ba-2291-f844dc7e69b8&psq=about+adam+in+deep+learning&u=a1aHR0cHM6Ly9kZWVwYWkub3JnL21hY2hpbmUtbGVhcm5pbmctZ2xvc3NhcnktYW5kLXRlcm1zL2FkYW0tbWFjaGluZS1sZWFybmluZw&ntb=1). [It can handle sparse gradients and adapt its learning rate for each parameter, making it a versatile and effective tool](https://www.bing.com/ck/a?!&&p=8b5044b2a724aa64JmltdHM9MTcwNjc0NTYwMCZpZ3VpZD0wYTA2ZTgzYi1kZGQzLTY4YmEtMjI5MS1mODQ0ZGM3ZTY5YjgmaW5zaWQ9NTc3Mw&ptn=3&ver=2&hsh=3&fclid=0a06e83b-ddd3-68ba-2291-f844dc7e69b8&psq=about+adam+in+deep+learning&u=a1aHR0cHM6Ly9kZWVwYWkub3JnL21hY2hpbmUtbGVhcm5pbmctZ2xvc3NhcnktYW5kLXRlcm1zL2FkYW0tbWFjaGluZS1sZWFybmluZw&ntb=1).

The method is really efficient when working with large problem involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the ‘gradient descent with momentum’ algorithm and the ‘RMSP’ algorithm.

**Momentum:**

This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the ‘exponentially weighted average’ of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace.





mt = aggregate of gradients at time t [current] (initially, mt = 0)

mt-1 = aggregate of gradients at time t-1 [previous]

Wt = weights at time t

Wt+1 = weights at time t+1

αt = learning rate at time t

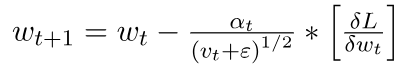
∂L = derivative of Loss Function

∂Wt = derivative of weights at time t

β = Moving average parameter (const, 0.9)

**Root Mean Square Propagation (RMSP):**

Root mean square prop or RMSprop is an adaptive learning algorithm that tries to improve AdaGrad. Instead of taking the cumulative sum of squared gradients like in AdaGrad, it takes the ‘exponential moving average.



where,

A black and white math equation

Description automatically generated

Wt = weights at time t

Wt+1 = weights at time t+1

αt = learning rate at time t

∂L = derivative of Loss Function

∂Wt = derivative of weights at time t

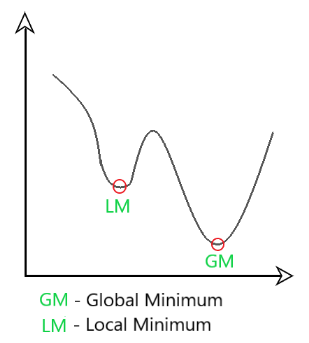
Vt = sum of square of past gradients. [i.e sum(∂L/∂Wt-1)] (initially, Vt = 0)

β = Moving average parameter (const, 0.9)

ϵ = A small positive constant (10-8)

**how Adam works?**

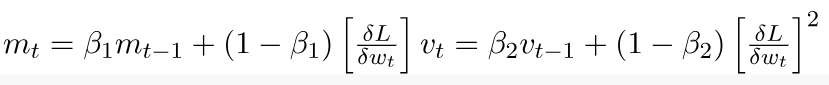
Adam Optimizer inherits the strengths or the positive attributes of the above two methods and builds upon them to give a more optimized gradient descent.



Here, we control the rate of gradient descent in such a way that there is minimum oscillation when it reaches the global minimum while taking big enough steps (step-size) so as to pass the local minima hurdles along the way. Hence, combining the features of the above methods to reach the global minimum efficiently.

**Mathematical Aspect of Adam Optimizer**

Taking the formulas used in the above two methods, we get



**Parameters Used :**

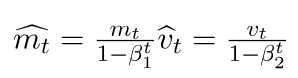
**1.ϵ =** a small +ve constant to avoid 'division by 0' error when

(vt -> 0). (10-8)

**2. β1 & β2 =** decay rates of average of gradients in the above two methods. (β1 = 0.9 & β2 = 0.999)

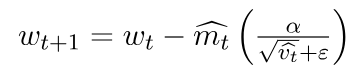
**3. α —** Step size parameter / learning rate (0.001)

Since mt and vthave both initialized as 0 (based on the above methods), it is observed that they gain a tendency to be ‘biased towards 0’ as both β1 & β2 ≈ 1. This Optimizer fixes this problem by computing ‘bias-corrected’ mtand vt. This is also done to control the weights while reaching the global minimum to prevent high oscillations when near it. The formulas used are:



Intuitively, we are adapting to the gradient descent after every iteration so that it remains controlled and unbiased throughout the process, hence the name Adam.

Now, instead of our normal weight parameters mtand vt, we take the bias-corrected weight parameters (m\_hat)tand (v\_hat)t. Putting them into our general equation, we get



#### **Steps Involved in the Adam Optimization Algorithm**

1. Initialize the first and second moments’ moving averages (m and v) to zero

2. Compute the gradient of the loss function to the model parameters.

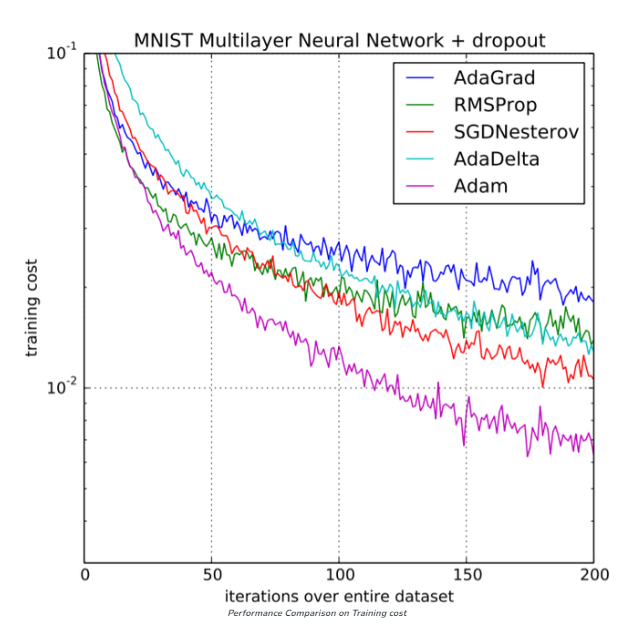
3. Update the moving averages using exponentially decaying averages. This involves calculating m\_t and v\_t as weighted averages of the previous moments and the current gradient.

4. Apply bias correction to the moving averages, particularly during the early iterations.

5. Calculate the parameter update by dividing the bias-corrected first moment by the square root of the bias-corrected second moment, with an added small constant (epsilon) for numerical stability.

6. Update the model parameters using the calculated updates

7. Repeat steps 2-6 for a specified number of iterations or until convergence.

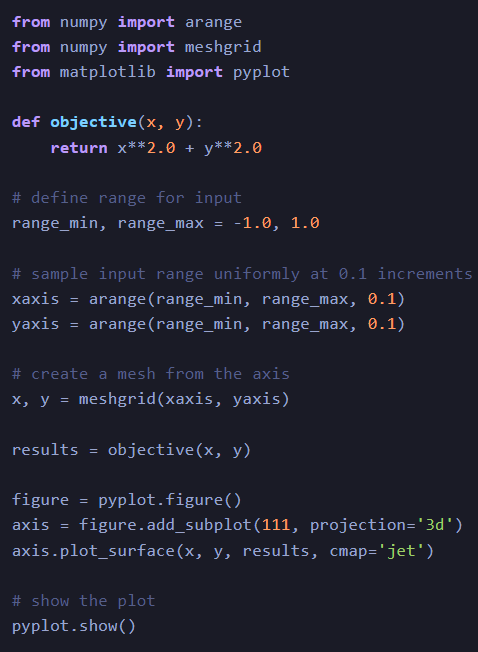
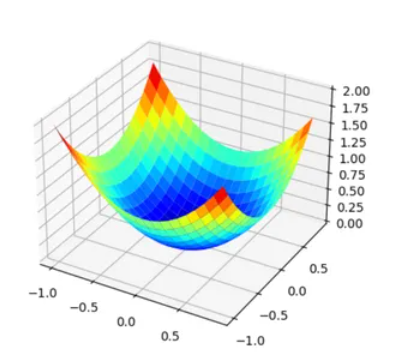


## Practical Implementation:

#### **Gradient Descent With Adam**

First, let’s define an optimization function. We will use a simple two-dimensional function that squares the input of each dimension and defines the range of valid inputs from -1.0 to 1.0.

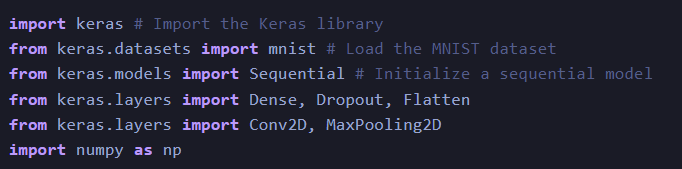
The objective() function below implements this function.

#### **Adam in Neural Network**

#### Here’s a simplified Python code example demonstrating how to use the Adam optimizer in a neural network training scenario using the popular deep learning library TensorFlow. In this example, we’ll use TensorFlow’s Keras API for creating and training a simple neural network for image classification:

1. **Importing Library**:

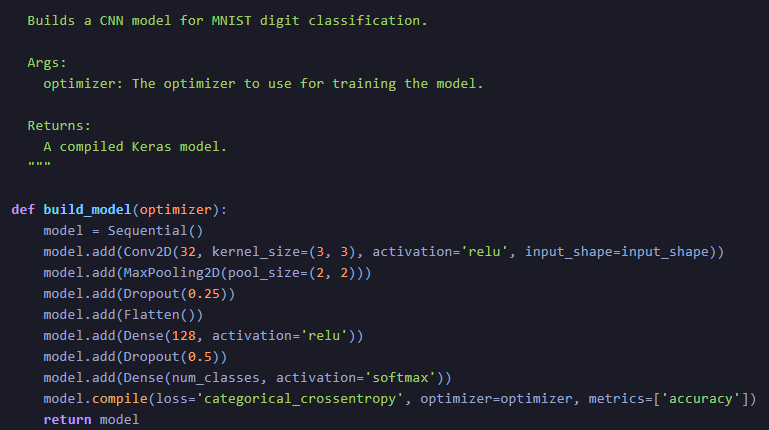


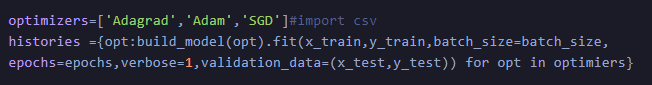
**2.Split Data in train and test**:

A screenshot of a computer program

Description automatically generated

**3.Define model function**:



4. **Optimization in Neural Network**:

## Key Features of Adam Optimizer:

**1. Adaptive Learning Rates**: Adam adjusts the learning rates for each parameter individually. It calculates a moving average of the first-order moments (the mean of gradients) and the second-order moments (the uncentered variance of gradients) to scale the learning rates adaptively. This makes it well-suited for problems with sparse gradients or noisy data.

**2.** **Bias Correction**: To counteract the initialization bias in the first moments, Adam applies bias correction during the early iterations of training. This ensures faster convergence and stabilizes the training process.

**3. Low Memory Requirements**: Unlike some optimization algorithms that require storing a history of gradients for each parameter, Adam only needs to maintain two moving averages per parameter. This makes it memory-efficient, especially for large neural networks.

## Advantages of Using Adam Optimizer:

1. **Fast Convergence**: Adam often converges faster than traditional gradient descent-based optimizers, especially on complex loss surfaces.

2. **Adaptive Learning Rates**: The adaptive learning rates make it suitable for various machine learning tasks, including natural language processing, computer vision, and reinforcement learning.

3. **Low Memory Usage**: Low memory requirements allow training large neural networks without running into memory constraints.

4. **Robustness**: Adam is relatively robust to hyperparameter choices, making it a good choice for practitioners without extensive hyperparameter tuning experience

**Disadvantages of Adam Optimizer:**

**1.Memory requirements**: Adam requires storing the first and second moments of the gradients for each parameter, which can be memory-intensive for large models with many parameters.

**2.Susceptible to noise**: Adam’s adaptive learning rate can make it sensitive to noise in the gradient estimates, especially for sparse data. This can lead to suboptimal convergence or even divergence in some cases.

**3.Biased estimates**: Adam’s first and second-moment estimates are biased towards zero, especially during the early stages of training. This can affect the convergence of the optimizer and may require more iterations to reach the optimal solution.

**4.Hyperparameter sensitivity**: While Adam is relatively insensitive to the choice of hyperparameters compared to other optimization algorithms, it still requires careful tuning of the learning rate, beta parameters, and epsilon to ensure optimal performance.

**5.Not guaranteed to converge**: Like other optimization algorithms, Adam is not guaranteed to converge to the global optimum and may get stuck in local optima or saddle points.

**2.Explain about Dataset Curation**

Dataset curation is a crucial aspect of deep learning that significantly influences the generalization performance of a model. Generalization refers to a model's ability to perform well on unseen data, and an appropriately curated dataset plays a key role in achieving this. Here's a detailed overview of dataset curation for improving the generalization of a deep learning model:

**Data Collection:**

Diverse Sources: Collect data from diverse sources to ensure the model learns a wide range of patterns and features.

Relevance: Ensure the data is relevant to the problem at hand. Irrelevant or noisy data can hinder generalization.

**Data Cleaning:**

Missing Values: Handle missing values appropriately, either by imputation or removal.

Outlier Detection: Identify and handle outliers to prevent them from unduly influencing the model.

Noise Reduction: Remove or mitigate noise in the data, such as errors or inconsistencies.

**Data Augmentation:**

Image Data: For image datasets, techniques like rotation, flipping, scaling, and cropping can be used to create augmented versions of existing images, increasing the dataset size and variability.

Text Data: For text datasets, techniques like paraphrasing, word substitution, and insertion of synonyms can be employed.

**Balancing Classes:**

Ensure a balanced representation of classes to prevent the model from favoring the majority class. Techniques include oversampling the minority class or undersampling the majority class.

**Normalization and Standardization:**

Normalize numerical features to a standard scale to prevent certain features from dominating others. This helps the model generalize well across different ranges of values.

**Feature Engineering:**

Create new features that may provide additional information to the model, enhancing its ability to generalize.

Reduce dimensionality using techniques like Principal Component Analysis (PCA) or feature selection.

**Cross-Validation:**

Implement cross-validation to assess the model's generalization performance on different subsets of the data. This helps identify potential overfitting or underfitting issues.

**Train-Validation-Test Split:**

Split the dataset into training, validation, and test sets. The model is trained on the training set, tuned on the validation set, and evaluated on the test set for unbiased performance assessment.

**Temporal and Spatial Considerations:**

For time-series or spatial data, ensure the temporal or spatial structure is maintained in the dataset split. This helps the model generalize well to unseen time periods or spatial regions.

**Continuous Monitoring and Updating:**

Regularly update and monitor the dataset to account for changes in the real-world distribution. This helps the model adapt to evolving patterns and maintain good generalization performance over time.

By paying careful attention to dataset curation, practitioners can enhance the robustness and generalization capabilities of deep learning models, ultimately improving their performance on new and unseen data.