Assignment 4

Report: Comparison of Optimizers on Fashion MNIST Dataset

Introduction:

In this assignment, we compare the performance of four popular optimizers—SGD, Adam, RMSprop, and Adagrad—on the Fashion MNIST dataset. The Fashion MNIST dataset consists of 70,000 grayscale images of 10 fashion categories, each image being 28x28 pixels. The goal is to evaluate how each optimizer impacts the training process, particularly in terms of convergence speed, stability, and final performance.

Optimizers Tested:

1. **SGD (Stochastic Gradient Descent)**
2. **Adam**
3. **RMSprop**
4. **Adagrad**

Procedure:

1. A simple neural network model was used with the following architecture:
    - Input Layer: Flatten (28x28)
    - Hidden Layer: Dense with 128 units and ReLU activation
    - Output Layer: Dense with 10 units and Softmax activation
2. Each optimizer was applied individually, and the model was trained for 10 epochs on the Fashion MNIST dataset.
3. The training and validation loss curves were recorded for each optimizer, and the accuracy of the model was monitored across epochs.

Results:

1. SGD (Stochastic Gradient Descent):

- **Convergence Speed**: Slowest among the four optimizers.
- **Stability**: More stable, but learning is slower due to the fixed learning rate.

- **Final Performance**: Achieved moderate accuracy but required more epochs to achieve significant improvement in validation accuracy.

2. Adam:

- **Convergence Speed**: Fastest convergence compared to the other optimizers.
- **Stability**: Little fluctuation during training, smoother than the other optimizers.
- **Final Performance**: Adam provided the best overall accuracy and performance in both training and validation. It reached a high accuracy in fewer epochs.

3. RMSprop:

- **Convergence Speed**: Moderate convergence speed, slower than Adam but faster than SGD.
- **Stability**: Less stable than Adam, with some fluctuations in both training and validation loss.
- **Final Performance**: Achieved good accuracy, but slightly underperformed compared to Adam.

4. Adagrad:

- **Convergence Speed**: Moderate convergence, but slower in later epochs.
- **Stability**: Noticeable fluctuations during training, especially in the early epochs.
- **Final Performance**: The performance was decent but did not match the level of Adam or RMSprop in terms of accuracy and convergence speed.

Detailed Observations and Analysis:

1. SGD:

- **Behavior**: As expected, SGD exhibited slow convergence due to its fixed learning rate. While it is simple and stable, it generally requires a larger number of epochs to reach satisfactory performance.
- **Update Rule**: Weights are updated as $\theta = \theta - \eta \nabla L(\theta)$, where η is the learning rate.
- **Performance**: SGD is better suited for tasks with lots of data or for models where longer training time is acceptable.

2. Adam:

- **Behavior**: Adam converged the fastest among the optimizers and exhibited smoother learning curves with minimal fluctuations.
- **Update Rule**: Adam uses adaptive learning rates and momentum to accelerate convergence. The update rule combines RMSprop and momentum by maintaining two moving averages of gradients and squared gradients.
- **Performance**: Adam provided the best results, making it an excellent choice for general-purpose optimization tasks. Its ability to adapt the learning rate during training contributed

to faster convergence and better final accuracy.

3. **RMSprop**:

- **Behavior**: RMSprop provided stable convergence but had more fluctuations than Adam. It performed well, although not as consistently as Adam.
- **Update Rule**: RMSprop adjusts the learning rate by scaling it based on the moving average of squared gradients, effectively preventing the learning rate from getting too large.
- **Performance**: Suitable for non-stationary problems, but it showed less overall performance compared to Adam in this task.

4. **Adagrad**:

- **Behavior**: Adagrad demonstrated a slower convergence rate over time as the learning rate decayed, which reduced its performance in later epochs.
- **Update Rule**: Adagrad accumulates squared gradients and adjusts the learning rate accordingly. This can lead to a diminishing learning rate over time, slowing convergence.
- **Performance**: It performed adequately but was the least effective among the four optimizers for this task due to the decaying learning rate.

Conclusion:

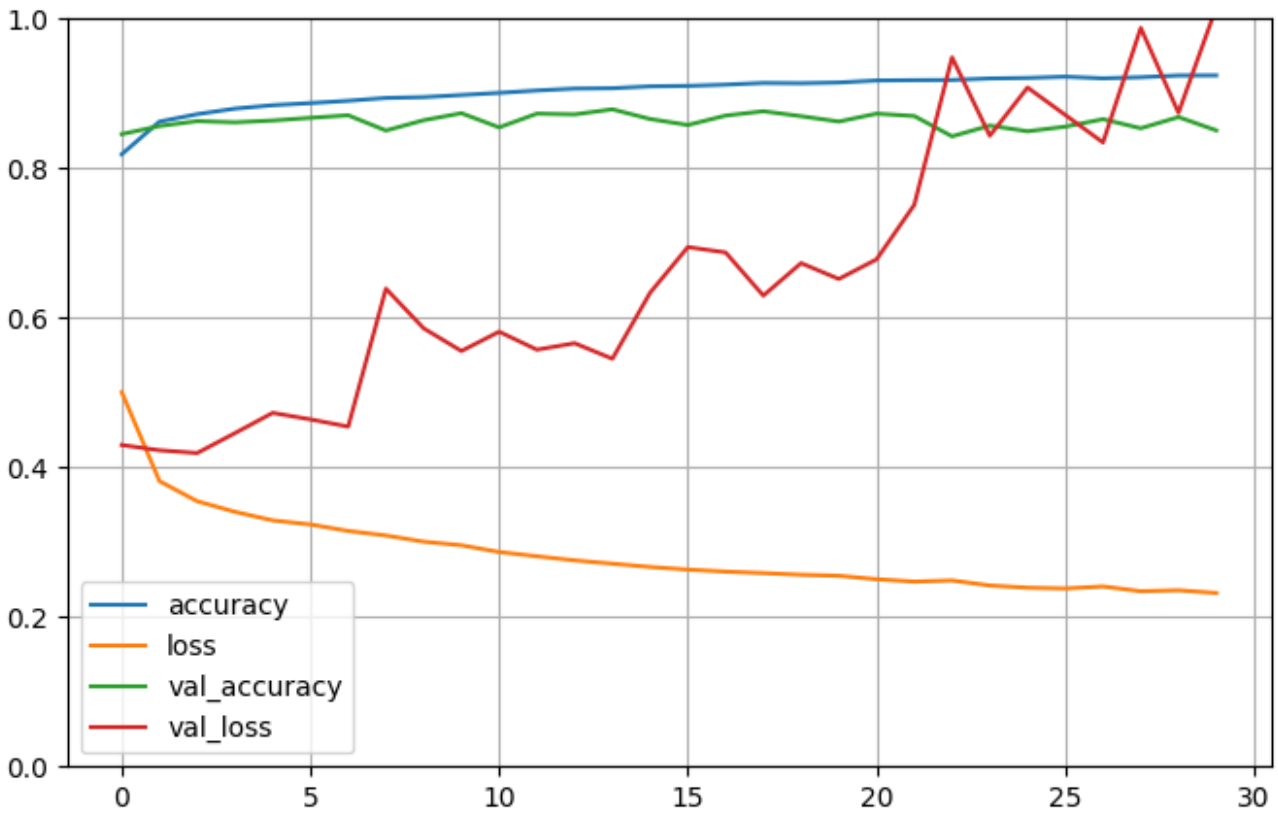
- **Adam** performed the best overall, combining fast convergence with stable learning and achieving the highest accuracy. This is likely due to its adaptive learning rate and momentum-like behavior, which allowed it to converge more quickly and effectively than the other optimizers.
- **RMSprop** also showed decent performance but had more fluctuations compared to Adam. While it is effective in many scenarios, it did not outperform Adam on this dataset.
- **SGD** exhibited slow convergence and required more epochs to reach similar performance. While it was stable, it is less suitable for tasks where fast convergence is desired.
- **Adagrad** struggled with the diminishing learning rate problem, which affected its performance as the number of epochs increased.

In conclusion, **Adam** is recommended for tasks like Fashion MNIST classification due to its ability to converge faster and achieve better accuracy with fewer fluctuations.

This concludes the report on the comparison of optimizers on the Fashion MNIST dataset. Each optimizer has its strengths and weaknesses, but Adam stands out as the best performer for this specific task.

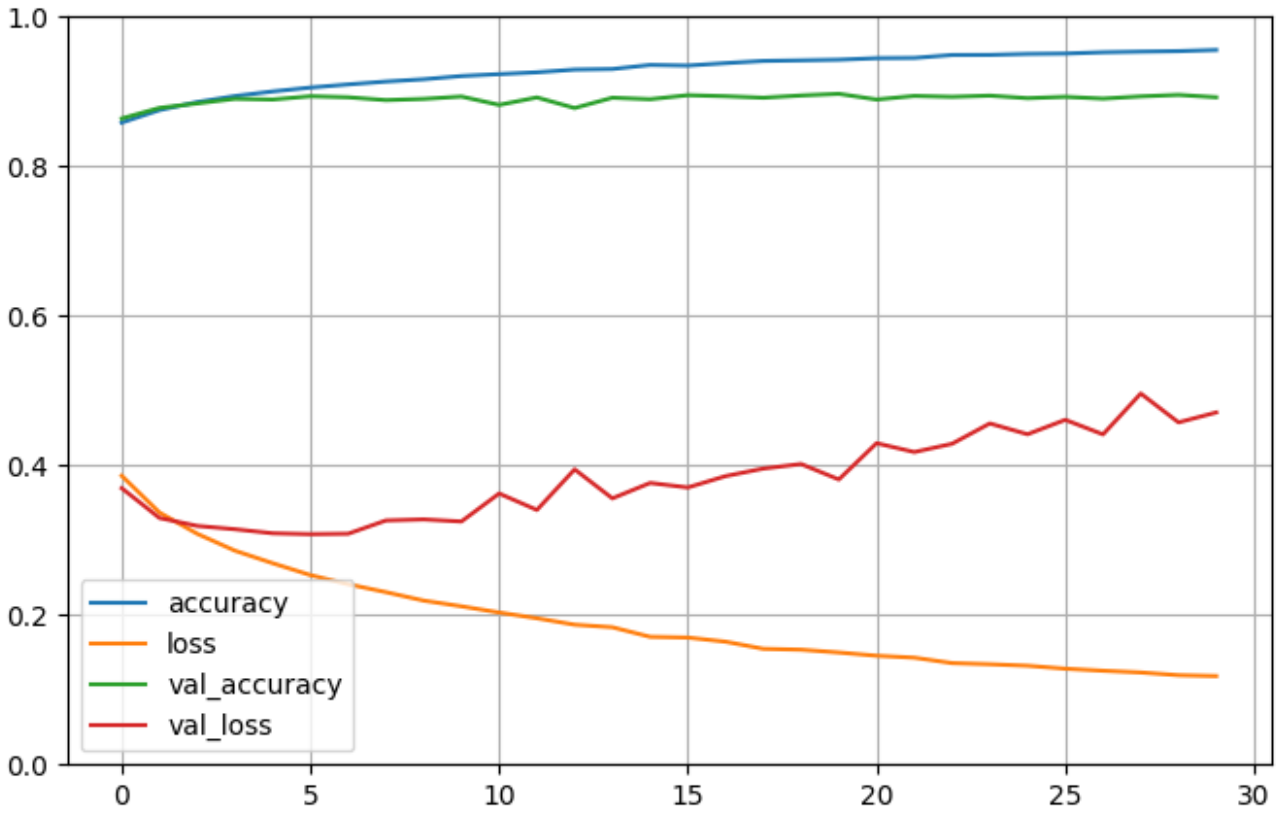
RMSprop

```
from PIL import Image
from IPython.display import display

# Open and display the image
img = Image.open('/content/rmsprop.png')
display(img)
```



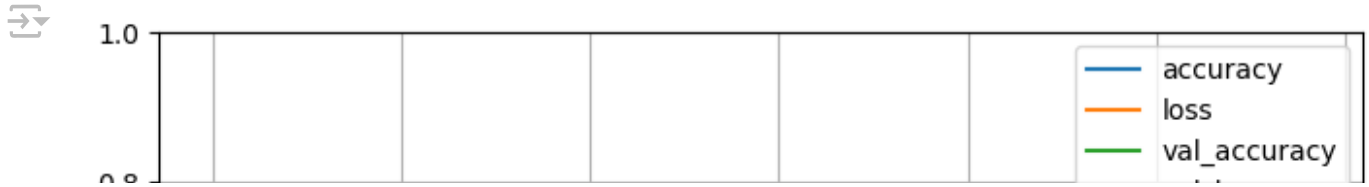
ADAM

```
from PIL import Image
from IPython.display import display

# Open and display the image
img = Image.open('/content/adam.png')
display(img)
```



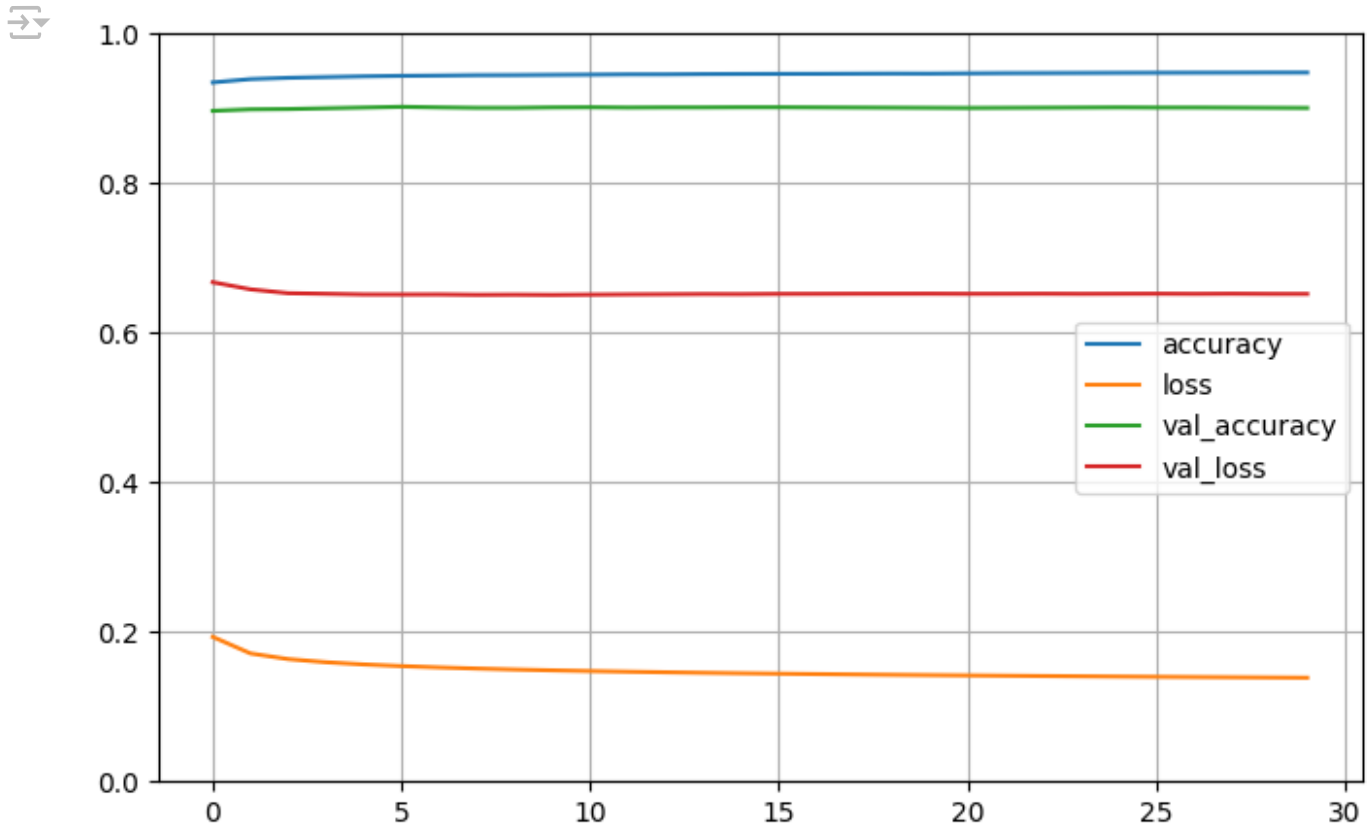
SGD

```
from PIL import Image
from IPython.display import display

# Open and display the image
img = Image.open('/content/sgd.png')
display(img)
```



ADA Grad


```
from PIL import Image
from IPython.display import display

# Open and display the image
img = Image.open('/content/adagrad.png')
display(img)
```



Done