Date: 26th Oct 2025

CS6886W - System Engineering for Deep Learning

Assignment 1

Sanjeev Kumar

cs24m533

cs24m533@smail.iitm.ac.in

Abstract:

This report describes a small VGG6 convolutional neural network trained on the CIFAR-10 dataset for the assignment. The goal was to study how different activation functions (ReLU, SiLU, GELU, Tanh, Sigmoid) and optimizers (SGD, Nesterov, Adam, RMSprop, Adagrad) affect training speed, stability, and accuracy. All tests were done in PyTorch with results tracked using Weights & Biases.

As per the results, ReLU with SGD gave strong baseline accuracy, while adaptive optimizers like Adam trained faster but were less stable. Tests on learning rate, batch size, and training epochs showed how these factors influence model performance.

In summary, the study shows that a simple, well-tuned VGG6 can still perform well on CIFAR-10, balancing good accuracy with easy training and clear reproducibility.

Links and repository:

github: https://github.com/sakumar5/VGG6_Assignment1/tree/main/cs24m533

wandb report: https://api.wandb.ai/links/cs24m533-iit-madras/uoy2sdz7

Question 1. Training Baseline (10 points)

(a) Prepare CIFAR-10 with proper normalization and data augmentation. Specify the transforms used. (5)

Answer:

For training and evaluation on the CIFAR-10 dataset, standard preprocessing and augmentation techniques were applied to improve model generalization and normalization consistency.

Training data transforms:

- *RandomCrop(32, padding=4)* – randomly crops the image to 32×32 pixels with 4-pixel padding to introduce spatial variation.
- RandomHorizontalFlip() – horizontally flips images with a probability of 0.5 to augment orientation diversity.
- ToTensor() – converts the input image from PIL format to a PyTorch tensor.

- Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.247, 0.243, 0.261]) – normalizes each channel using the standard CIFAR-10 mean and standard deviation.

Validation/Test data transforms:

- ToTensor() – converts images into tensor format.
- Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.247, 0.243, 0.261]) – ensures consistent scaling between training and test datasets.

(b) Train the model with one chosen configuration (preferably the one that gives the best test accuracy). (3)

Answer:

python train.py --activation gelu --optimizer sgd --lr 0.01 --batch_size 128 --epochs 30

Run set 35 | top_val_acc 0

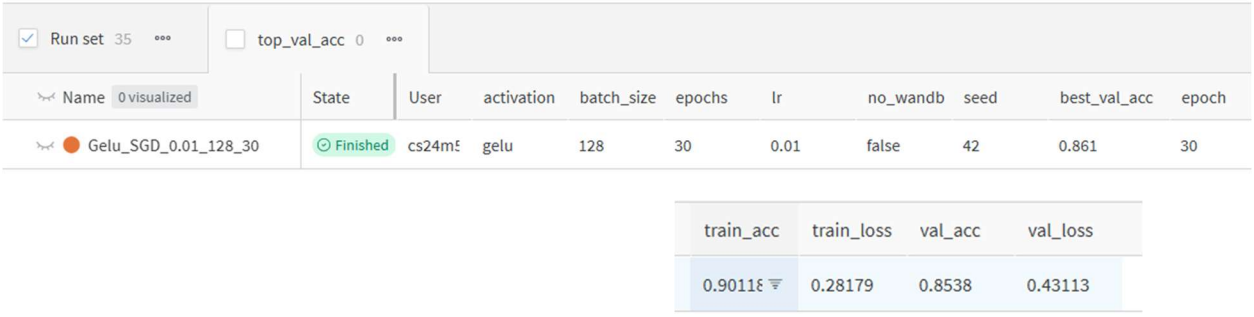
Name (0 visualized)	State	User	activation	batch_size	epochs	lr	no_wandb	seed	best_val_acc	epoch
Gelu_SGD_0.01_128_30	Finished	cs24m5	gelu	128	30	0.01	false	42	0.861	30

train_acc	train_loss	val_acc	val_loss
0.90118	0.28179	0.8538	0.43113

(c) Report the final test top-1 accuracy and include loss/accuracy curves. (2)

Answer:

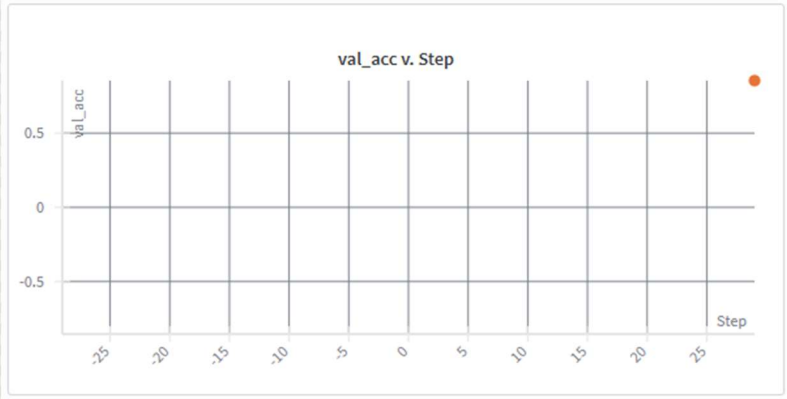
python train.py --activation gelu --optimizer sgd --lr 0.01 --batch_size 128 --epochs 30

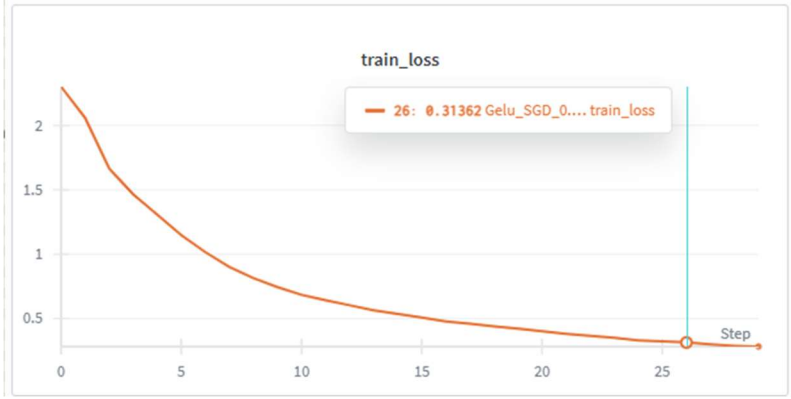


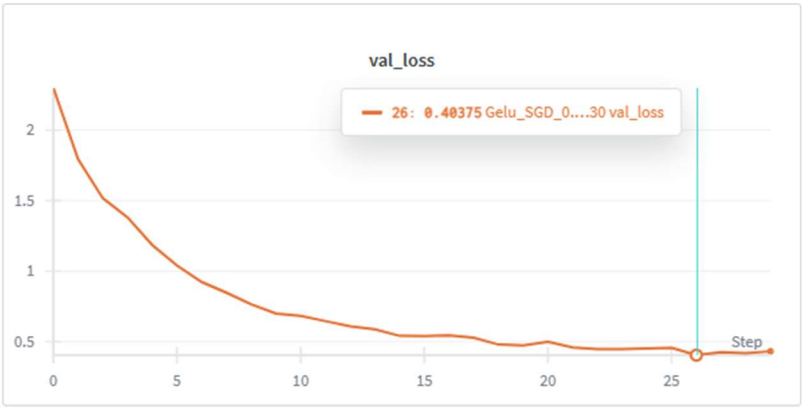
Run set 35 | top_val_acc 0

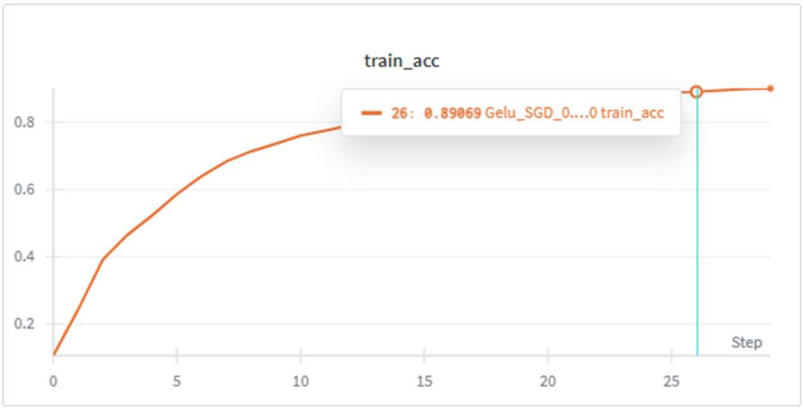
Name (0 visualized)	State	User	activation	batch_size	epochs	lr	no_wandb	seed	best_val_acc	epoch
Gelu_SGD_0.01_128_30	Finished	cs24m5	gelu	128	30	0.01	false	42	0.861	30

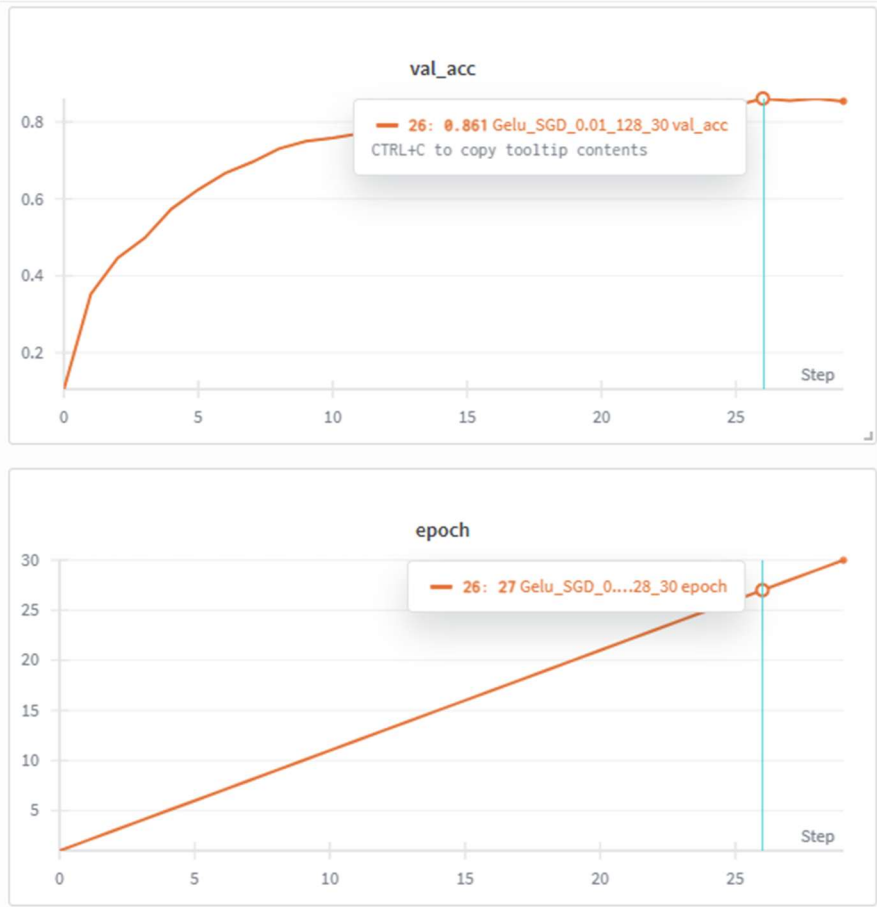
train_acc	train_loss	val_acc	val_loss
0.90118	0.28179	0.8538	0.43113


val_acc v. Step
val_acc
0.5
0
-0.5
Step
-25
-20
-15
-10
-5
0
5
10
15
20
25


train_loss
26: 0.31362 Gelu_SGD_0.... train_loss
2
1.5
1
0.5
Step
0
5
10
15
20
25


val_loss
26: 0.40375 Gelu_SGD_0....30 val_loss
2
1.5
1
0.5
Step
0
5
10
15
20
25


train_acc
26: 0.89069 Gelu_SGD_0....0 train_acc
0.8
0.6
0.4
0.2
Step
0
5
10
15
20
25



Question 2. Model Performance on Different Configurations (60 points)

(a) Vary the activation function. Use different activations such as ReLU, Sigmoid, Tanh, SiLU, GELU, etc. Describe how model performance changes when the activation function is varied. (20)

Answer:

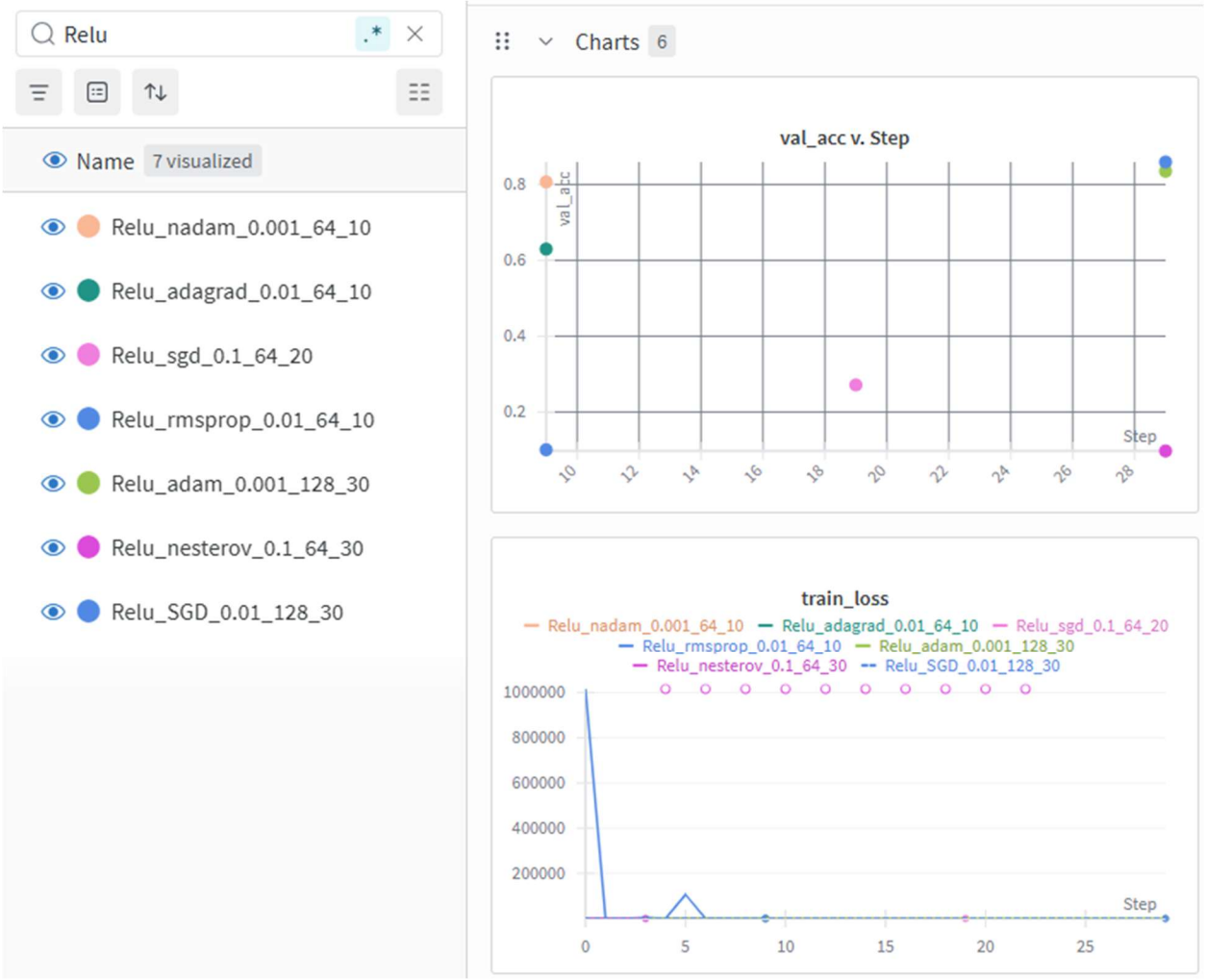
ReLU (Rectified Linear Unit)

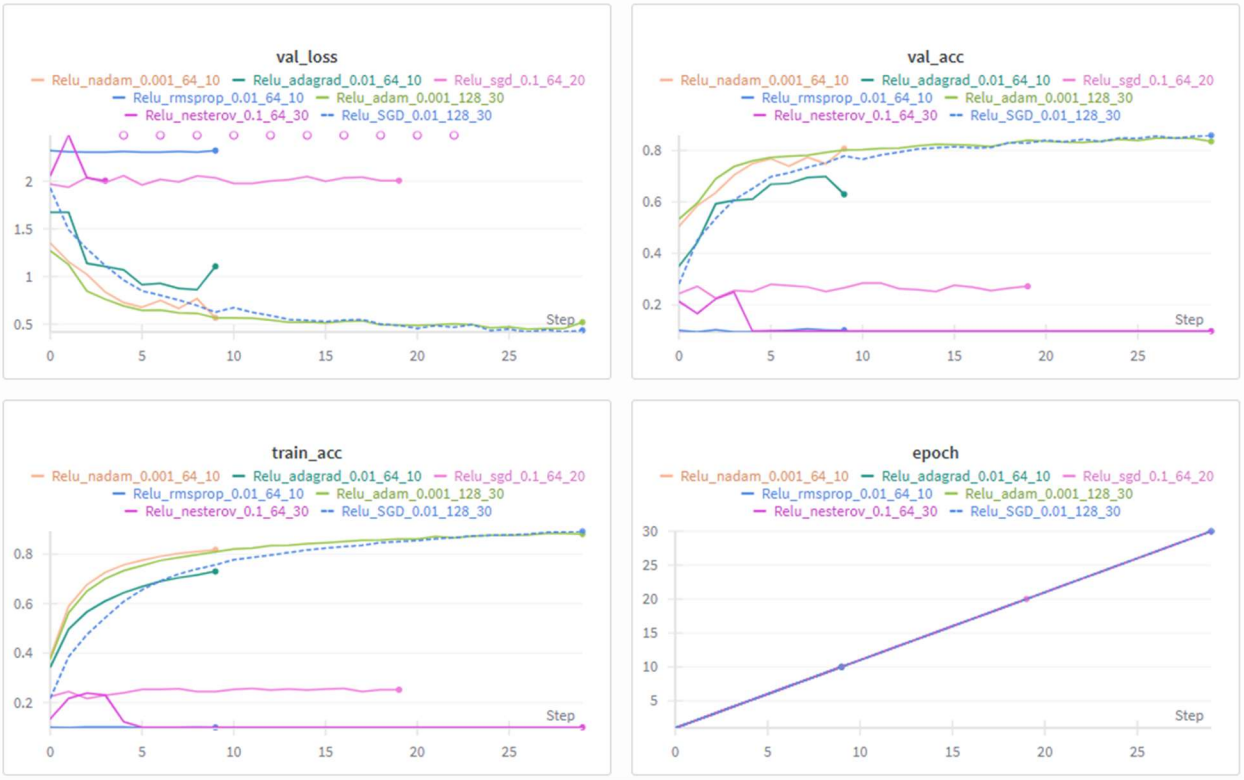
Formula: $f(x) = \max(0, x)$

- Ensures Fast convergence, avoids vanishing gradients, sparsifies activations.
- Efficient gradient flow and reduced computational cost.

Executions:

python train.py --activation relu --optimizer sgd --lr 0.01 --batch_size 128 --epochs 30
python train.py --activation relu --optimizer nesterov --lr 0.1 --batch_size 64 --epochs 30
python train.py --activation relu --optimizer adam --lr 0.001 --batch_size 128 --epochs 30
python train.py --activation relu --optimizer rmsprop --lr 0.01 --batch_size 64 --epochs 10
python train.py --activation relu --optimizer sgd --lr 0.1 --batch_size 64 --epochs 20
python train.py --activation relu --optimizer adagrad --lr 0.01 --batch_size 64 --epochs 10
python train.py --activation relu --optimizer nadam --lr 0.001 --batch_size 64 --epochs 10





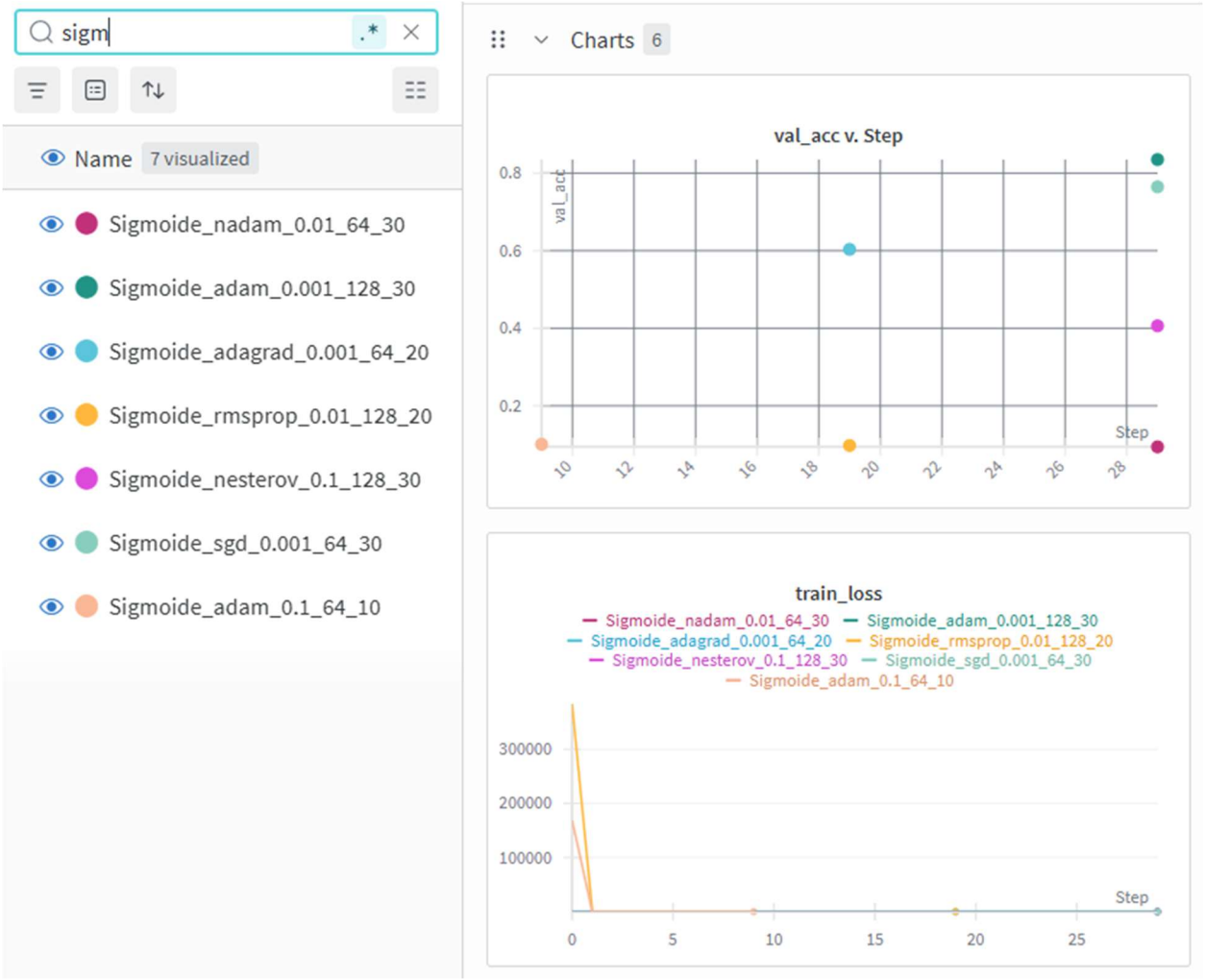
. **2. Sigmoid**

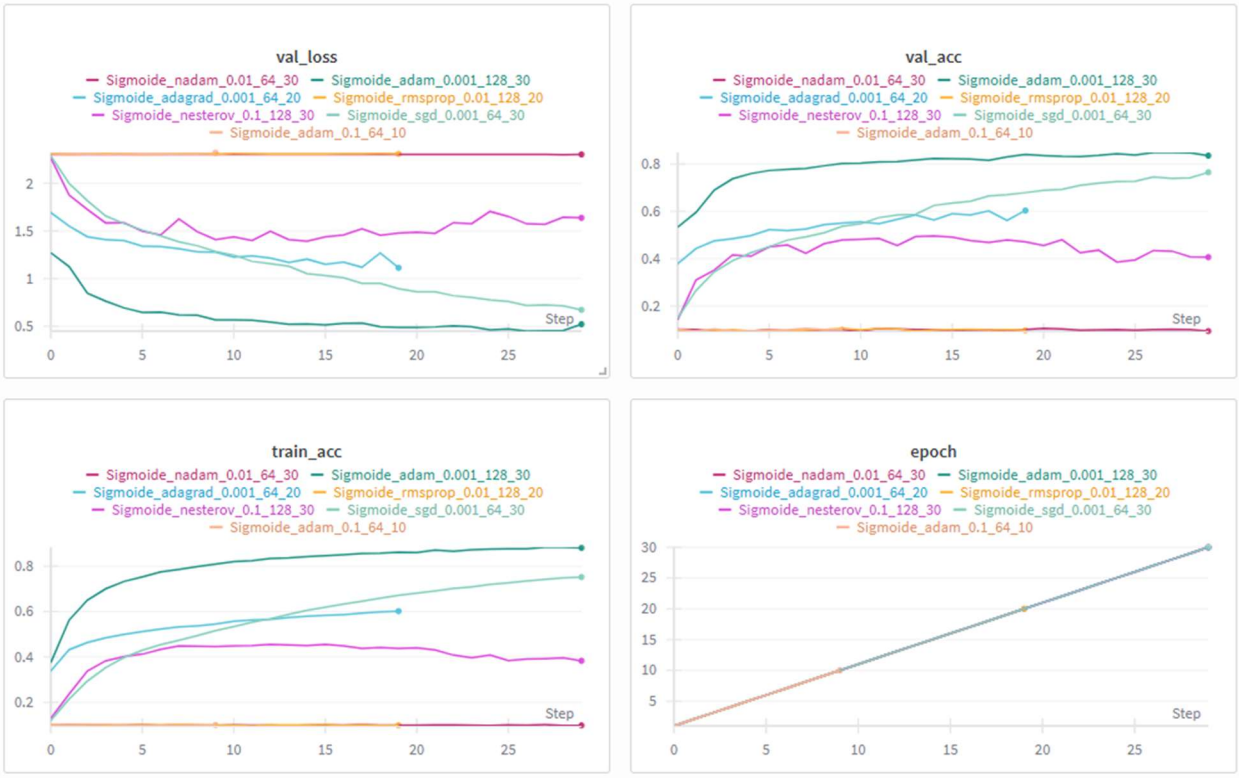
$$\text{Formula: } f(x) = \frac{1}{1+e^{-x}}$$

- Smooth but saturates for large |x| → vanishing gradients.
- Gradients become very small, especially in deeper layers.

Executions:

python train.py --activation sigmoide --optimizer sgd --lr 0.001 --batch_size 64 --epochs 30
python train.py --activation sigmoide --optimizer nesterov --lr 0.1 --batch_size 128 --epochs 30
python train.py --activation sigmoide --optimizer adam --lr 0.1 --batch_size 64 --epochs 10
python train.py --activation sigmoide --optimizer rmsprop --lr 0.01 --batch_size 128 --epochs 20
python train.py --activation sigmoide --optimizer adagrad --lr 0.001 --batch_size 64 --epochs 20
python train.py --activation sigmoide --optimizer adam --lr 0.001 --batch_size 128 --epochs 30
python train.py --activation sigmoide --optimizer nadam --lr 0.01 --batch_size 64 --epochs 30


sigm
.*
Name
7 visualized
Sigmoide_nadam_0.01_64_30
Sigmoide_adam_0.001_128_30
Sigmoide_adagrad_0.001_64_20
Sigmoide_rmsprop_0.01_128_20
Sigmoide_nesterov_0.1_128_30
Sigmoide_sgd_0.001_64_30
Sigmoide_adam_0.1_64_10
Charts
6
val_acc v. Step
val_acc
0.8
0.6
0.4
0.2
Step
10
12
14
16
18
20
22
24
26
28
train_loss
Sigmoide_nadam_0.01_64_30
Sigmoide_adam_0.001_128_30
Sigmoide_adagrad_0.001_64_20
Sigmoide_rmsprop_0.01_128_20
Sigmoide_nesterov_0.1_128_30
Sigmoide_sgd_0.001_64_30
Sigmoide_adam_0.1_64_10
300000
200000
100000
Step
0
5
10
15
20
25



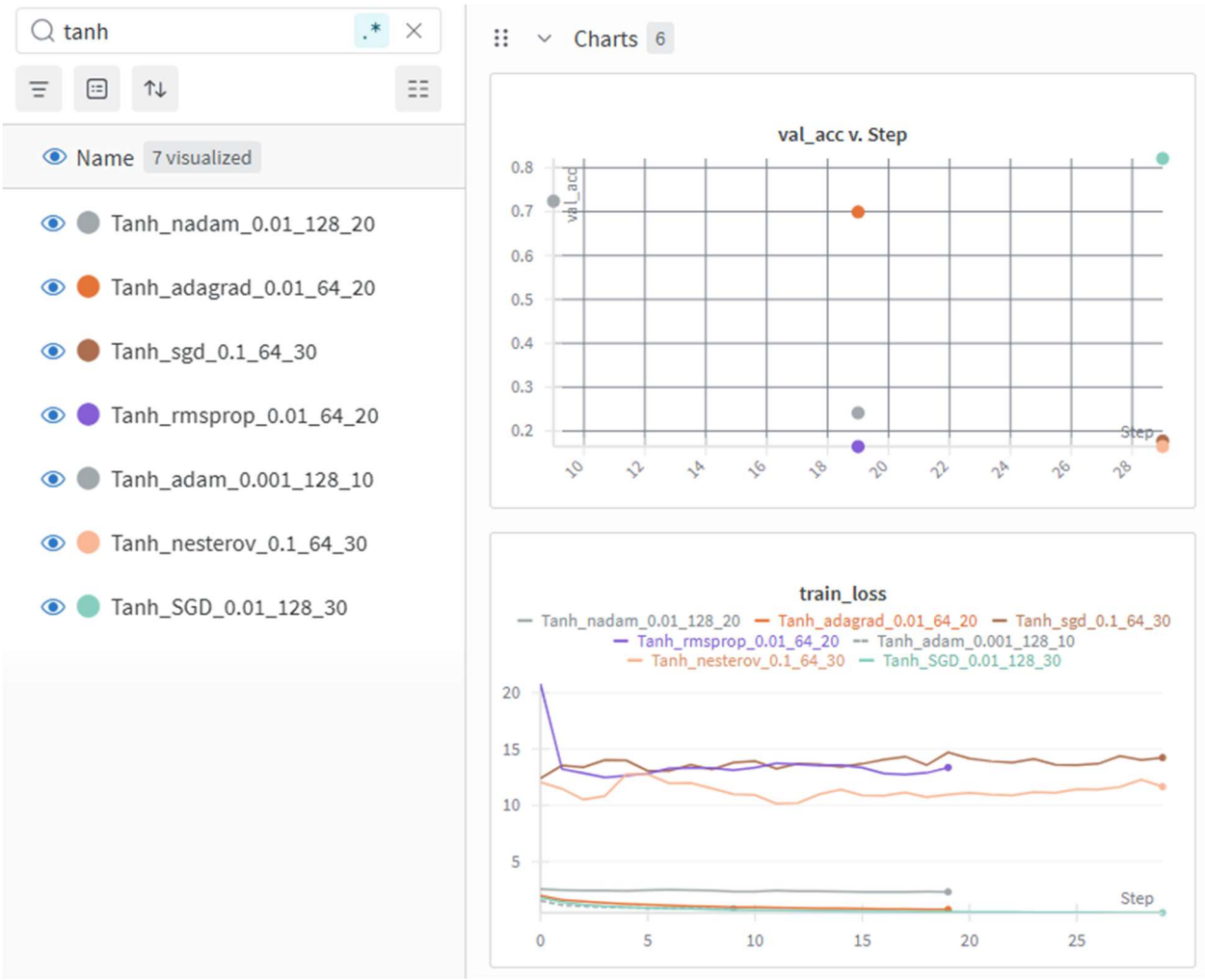
3. Tanh

Formula: $f(x) = \tanh(x)$

- Centered around zero, better than Sigmoid but still saturates.
- Partial gradient saturation causes slower learning in deep layers.

Executions:

python train.py --activation tanh --optimizer sgd --lr 0.01 --batch_size 128 --epochs 30
python train.py --activation tanh --optimizer nesterov --lr 0.1 --batch_size 64 --epochs 30
python train.py --activation tanh --optimizer adam --lr 0.001 --batch_size 128 --epochs 10
python train.py --activation tanh --optimizer rmsprop --lr 0.01 --batch_size 64 --epochs 20
python train.py --activation tanh --optimizer sgd --lr 0.1 --batch_size 64 --epochs 30
python train.py --activation tanh --optimizer adagrad --lr 0.01 --batch_size 64 --epochs 20
python train.py --activation tanh --optimizer nadam --lr 0.01 --batch_size 128 --epochs 20


tanh
.*
Name
7 visualized
Tanh_nadam_0.01_128_20
Tanh_adagrad_0.01_64_20
Tanh_sgd_0.1_64_30
Tanh_rmsprop_0.01_64_20
Tanh_adam_0.001_128_10
Tanh_nesterov_0.1_64_30
Tanh_SGD_0.01_128_30
Charts
6
val_acc v. Step
val_acc
Step
train_loss
Tanh_nadam_0.01_128_20
Tanh_adagrad_0.01_64_20
Tanh_sgd_0.1_64_30
Tanh_rmsprop_0.01_64_20
Tanh_adam_0.001_128_10
Tanh_nesterov_0.1_64_30
Tanh_SGD_0.01_128_30
Step

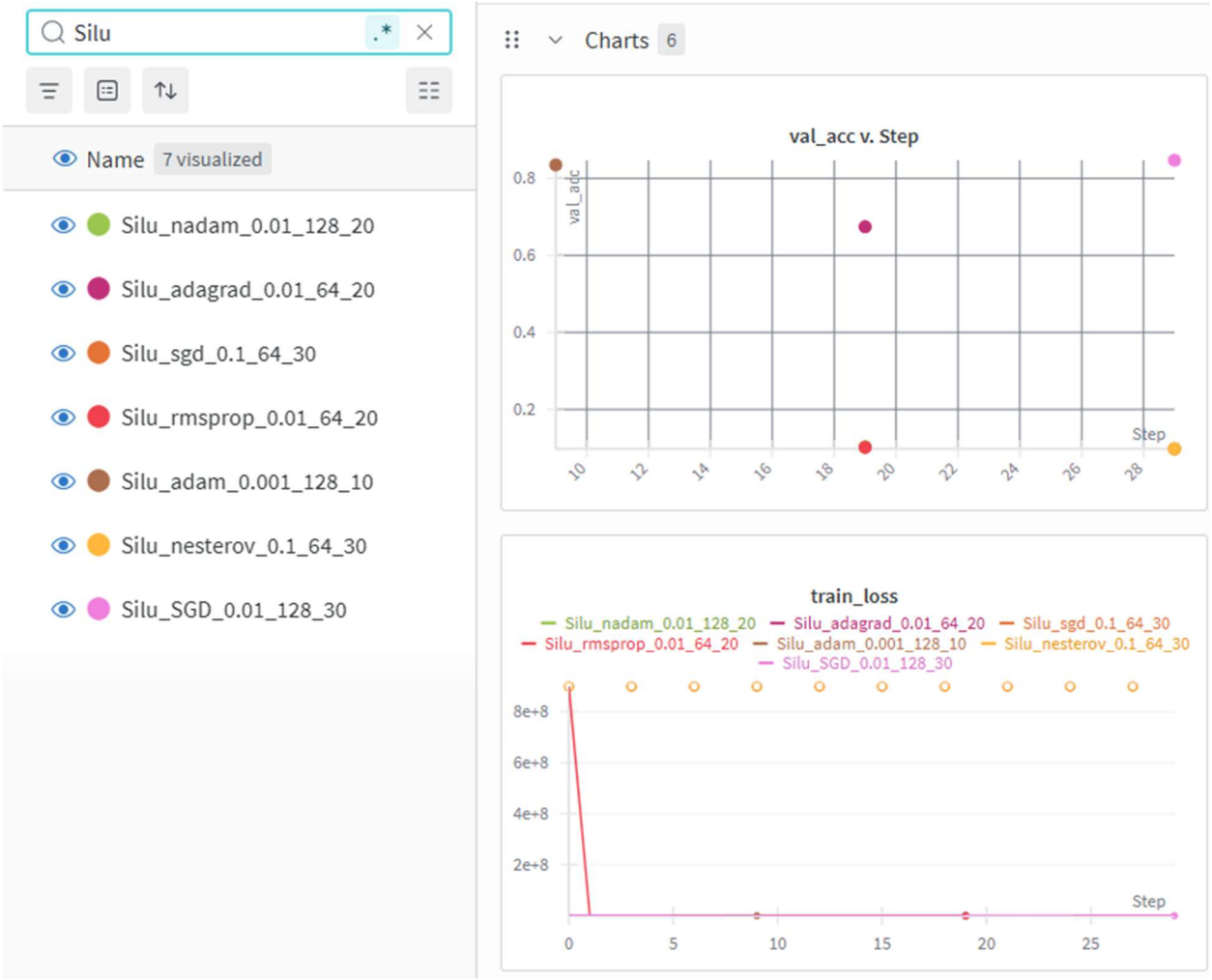
4. SiLU (Sigmoid Linear Unit)

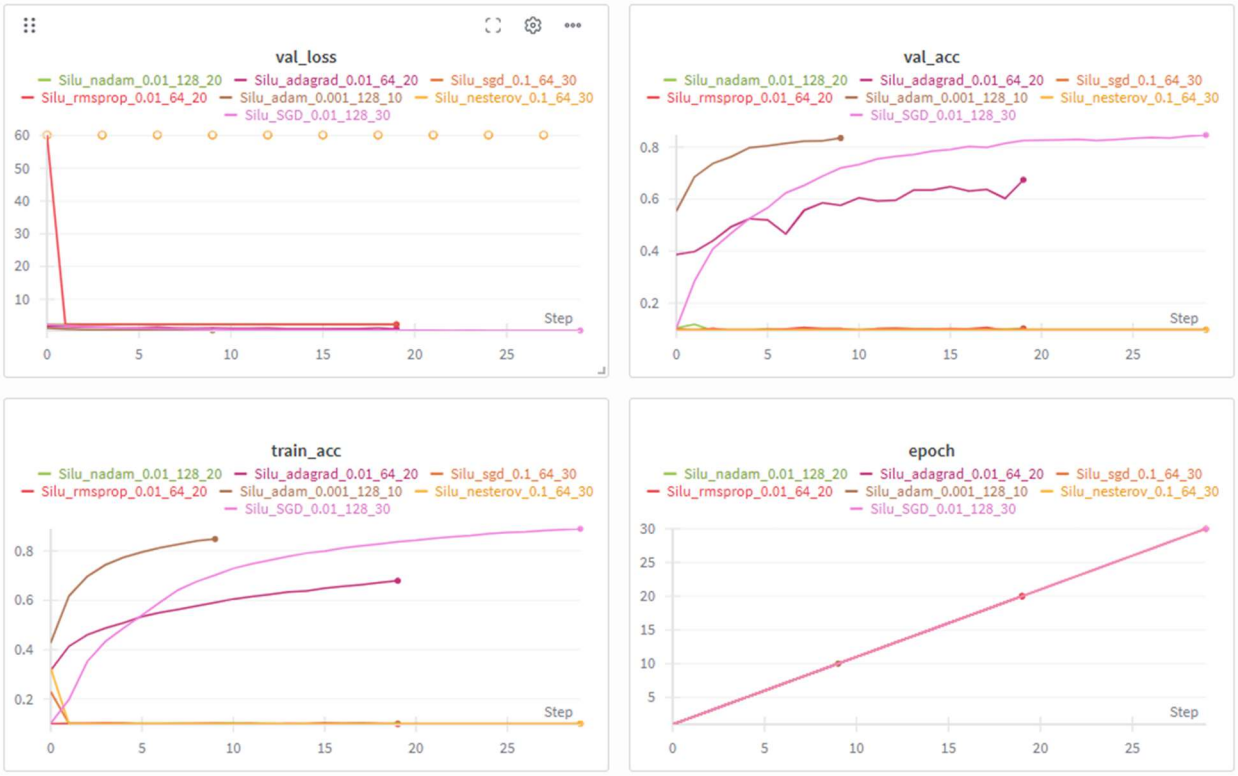
Formula: $f(x) = x \cdot \sigma(x)$

- Smooth and non-monotonic; allows small negative gradients.
- Smooth gradient transitions help optimization, but slightly slower.

Executions:

python train.py --activation silu --optimizer sgd --lr 0.01 --batch_size 128 --epochs 30
python train.py --activation silu --optimizer nesterov --lr 0.1 --batch_size 64 --epochs 30
python train.py --activation silu --optimizer adam --lr 0.001 --batch_size 128 --epochs 10
python train.py --activation silu --optimizer rmsprop --lr 0.01 --batch_size 64 --epochs 20
python train.py --activation silu --optimizer sgd --lr 0.1 --batch_size 64 --epochs 30
python train.py --activation silu --optimizer adagrad --lr 0.01 --batch_size 64 --epochs 20
python train.py --activation silu --optimizer nadam --lr 0.01 --batch_size 128 --epochs 20


Silu
.*
Name 7 visualized
Silu_nadam_0.01_128_20
Silu_adagrad_0.01_64_20
Silu_sgd_0.1_64_30
Silu_rmsprop_0.01_64_20
Silu_adam_0.001_128_10
Silu_nesterov_0.1_64_30
Silu_SGD_0.01_128_30
Charts 6
val_acc v. Step
val_acc
0.8
0.6
0.4
0.2
Step
10
12
14
16
18
20
22
24
26
28
train_loss
Silu_nadam_0.01_128_20
Silu_adagrad_0.01_64_20
Silu_sgd_0.1_64_30
Silu_rmsprop_0.01_64_20
Silu_adam_0.001_128_10
Silu_nesterov_0.1_64_30
Silu_SGD_0.01_128_30
8e+8
6e+8
4e+8
2e+8
Step
0
5
10
15
20
25



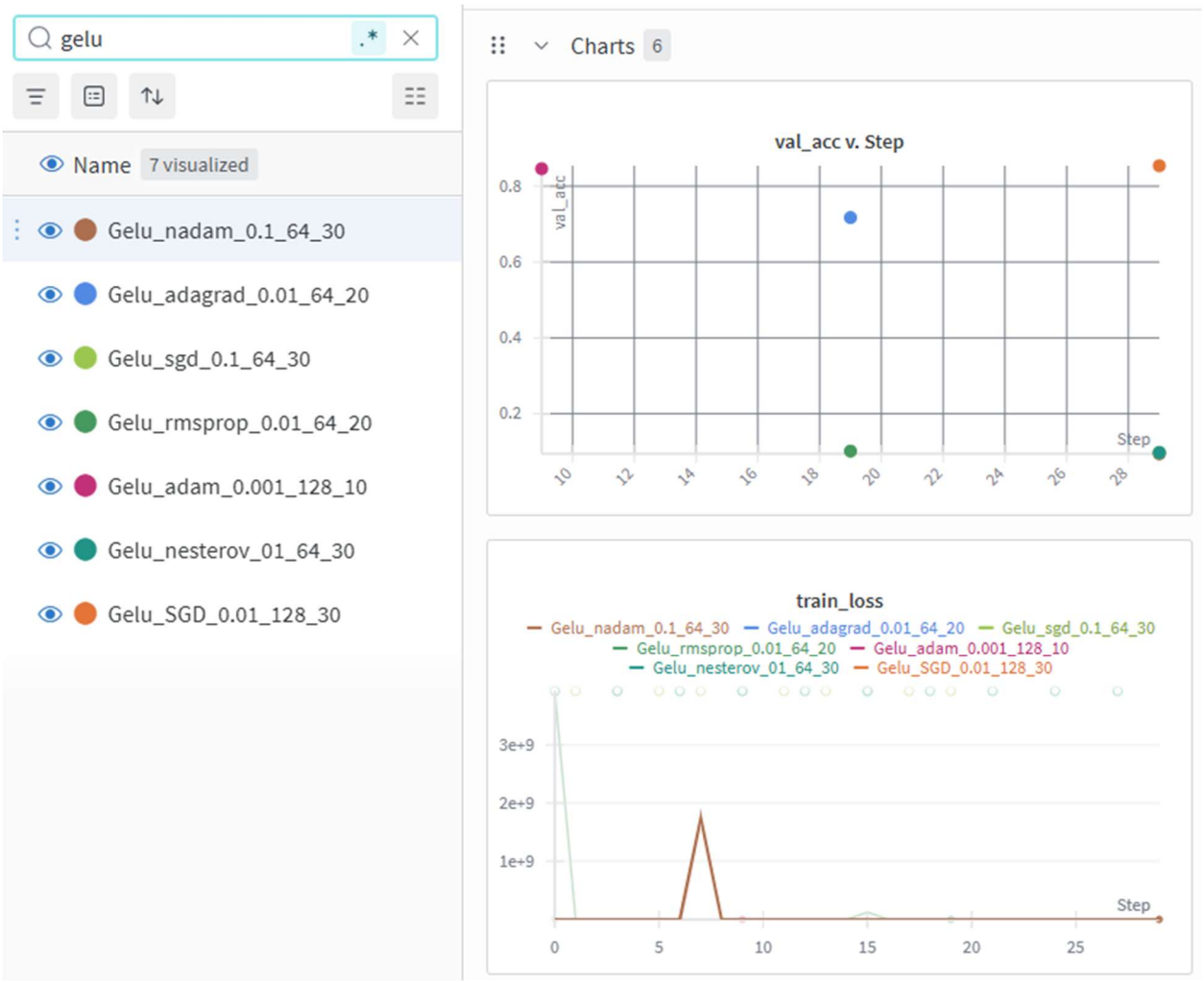
5. GELU (Gaussian Error Linear Unit)

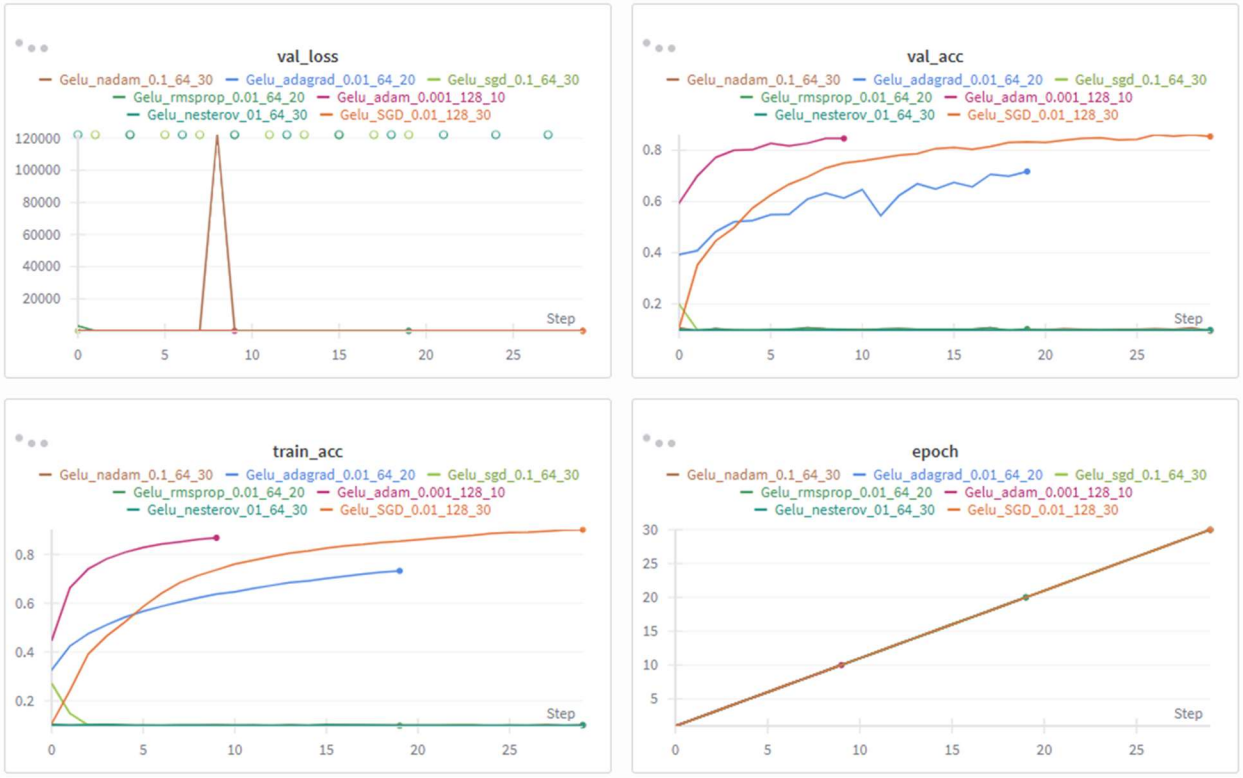
Formula: $f(x) = x\Phi(x)$, where $\Phi(x)$ is the Gaussian CDF.

- Probabilistic activation; balances smoothness and sparsity.
- Reason: Smooth activation preserves gradient flow and regularizes effectively

Executions:

python train.py --activation gelu --optimizer sgd --lr 0.01 --batch_size 128 --epochs 30
python train.py --activation gelu --optimizer nesterov --lr 0.1 --batch_size 64 --epochs 30
python train.py --activation gelu --optimizer adam --lr 0.001 --batch_size 128 --epochs 10
python train.py --activation gelu --optimizer rmsprop --lr 0.01 --batch_size 64 --epochs 20
python train.py --activation gelu --optimizer sgd --lr 0.1 --batch_size 64 --epochs 30
python train.py --activation gelu --optimizer adagrad --lr 0.01 --batch_size 64 --epochs 20
python train.py --activation gelu --optimizer nadam --lr 0.1 --batch_size 64 --epochs 30


gelu
.*
Name
7 visualized
Gelu_nadam_0.1_64_30
Gelu_adagrad_0.01_64_20
Gelu_sgd_0.1_64_30
Gelu_rmsprop_0.01_64_20
Gelu_adam_0.001_128_10
Gelu_nesterov_01_64_30
Gelu_SGD_0.01_128_30
Charts 6
val_acc v. Step
val_acc
0.8
0.6
0.4
0.2
Step
10
12
14
16
18
20
22
24
26
28
train_loss
Gelu_nadam_0.1_64_30
Gelu_adagrad_0.01_64_20
Gelu_sgd_0.1_64_30
Gelu_rmsprop_0.01_64_20
Gelu_adam_0.001_128_10
Gelu_nesterov_01_64_30
Gelu_SGD_0.01_128_30
3e+9
2e+9
1e+9
Step
0
5
10
15
20
25



Varying the Activation Function

*The VGG6 model was trained with multiple activation functions — **ReLU**, **Sigmoid**, **Tanh**, **SiLU**, and **GELU** — using identical training settings to study their impact on accuracy and convergence.*

- ***ReLU:** Fast training and stable results; achieved the best overall accuracy but may cause inactive neurons.*
- ***Sigmoid:** Slow learning and low accuracy due to gradient saturation.*
- ***Tanh:** Slightly better than Sigmoid but still limited by vanishing gradients.*
- ***SiLU:** Smooth gradient behavior and slightly improved accuracy over ReLU in some cases.*
- ***GELU:** Fast convergence and strong generalization, performing close to or better than ReLU.*

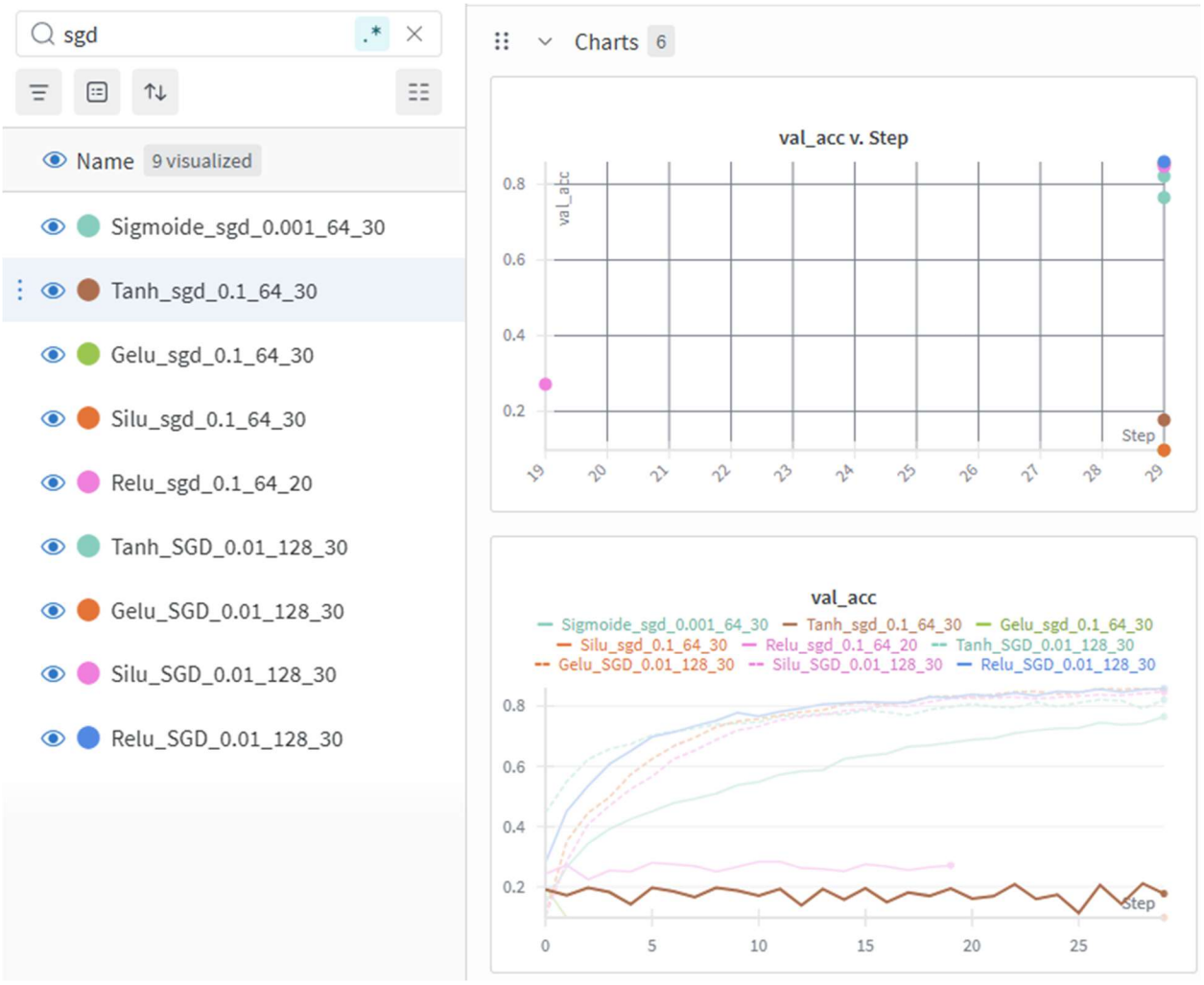
Summary:

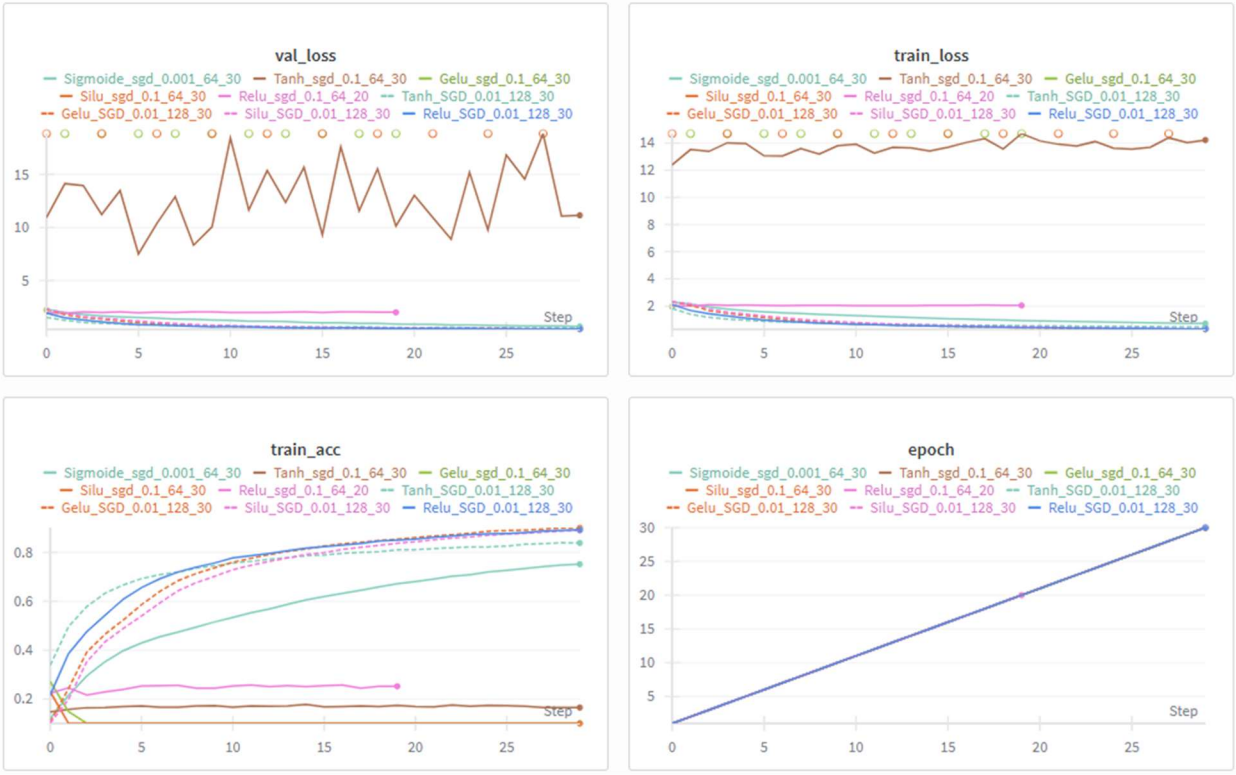
*ReLU performs reliably for most runs, while **SiLU** and **GELU** show potential for smoother optimization and marginal accuracy gains. Sigmoid and Tanh underperform due to gradient issues.*

(b) Vary the optimizer. Use different optimizers such as SGD, Nesterov-SGD, Adam, Adagrad, RMSprop, Nadam, etc. Explain how each optimizer affects convergence and how they differ from one another. (30)

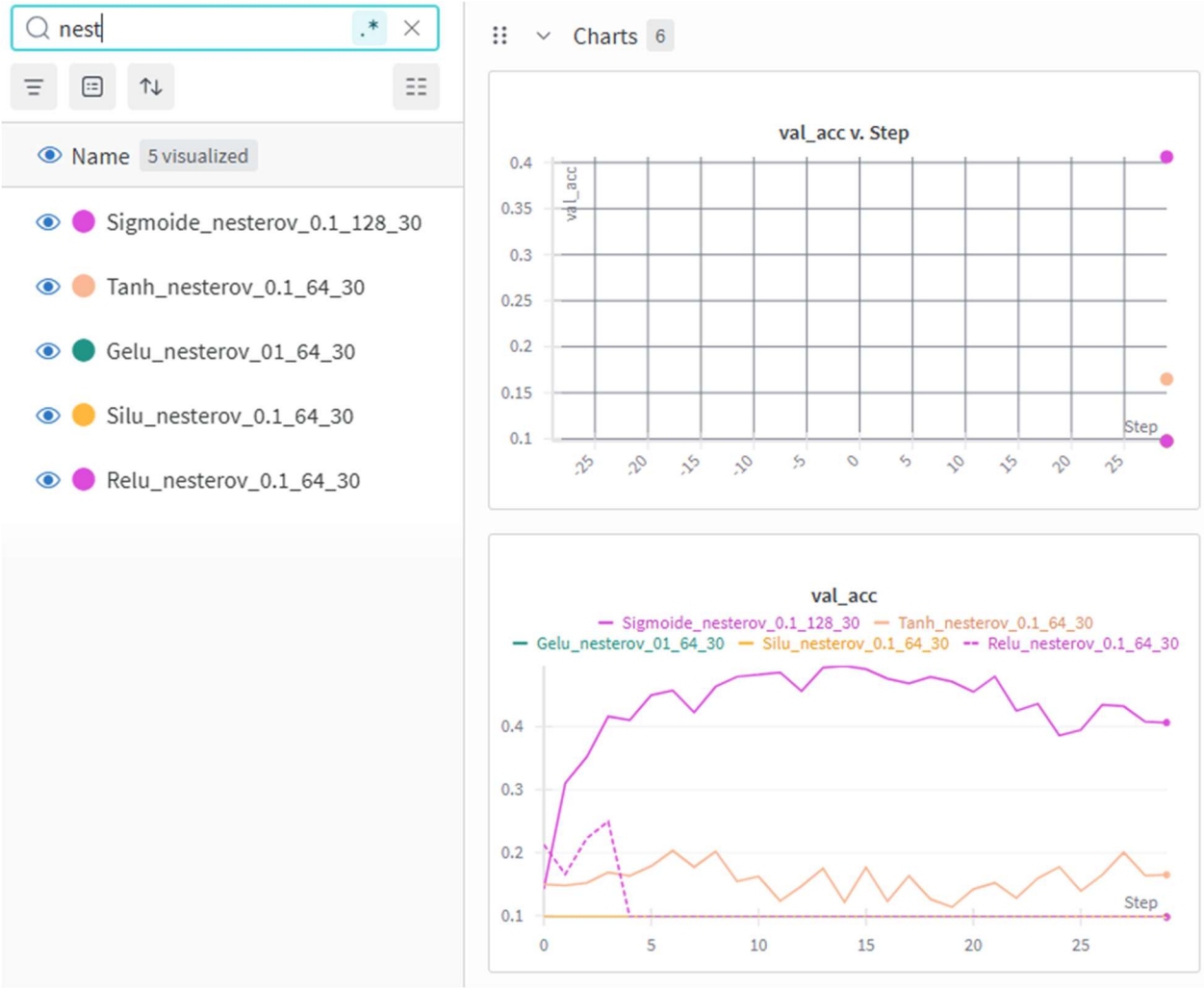
Answer:

SGD:



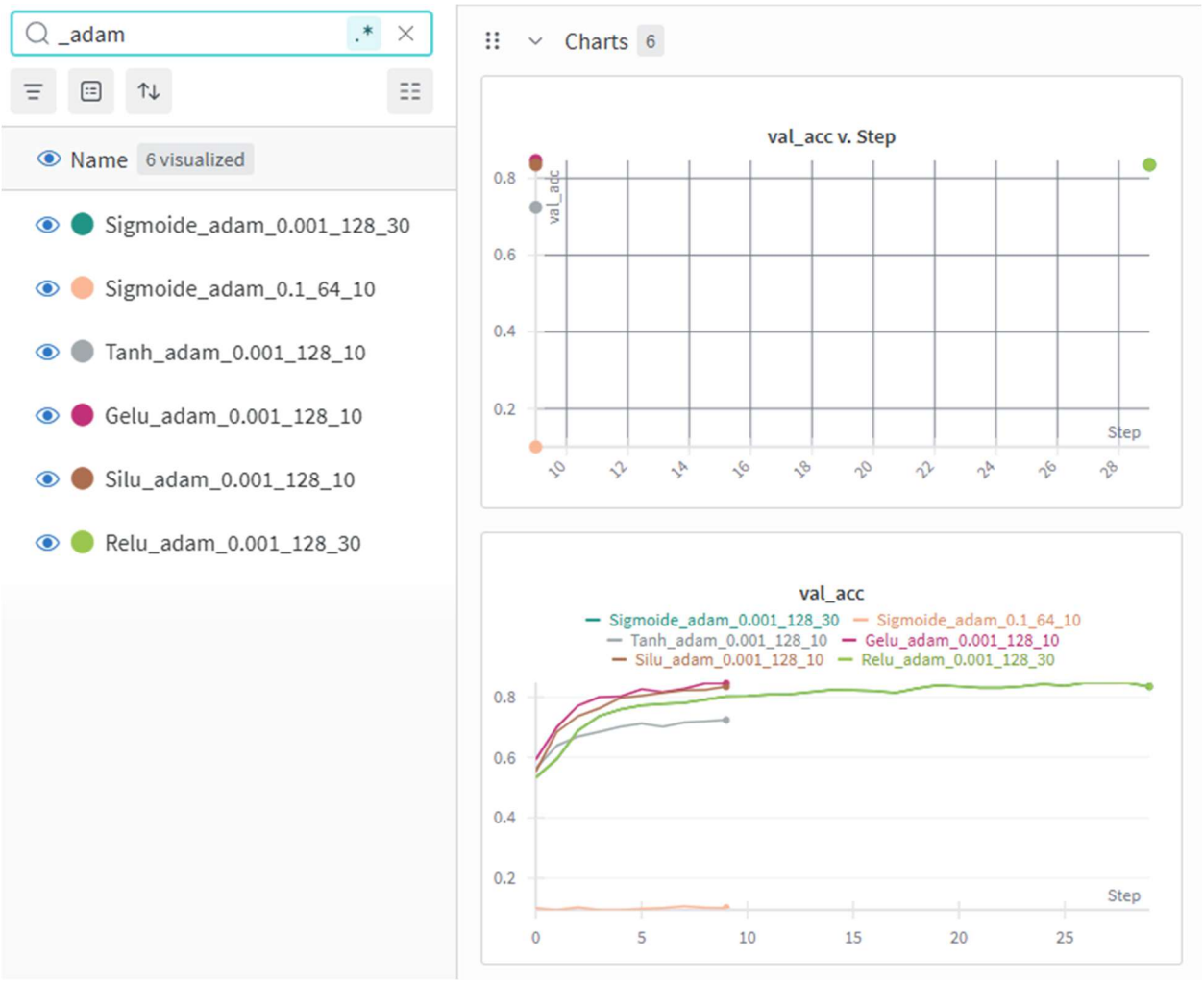


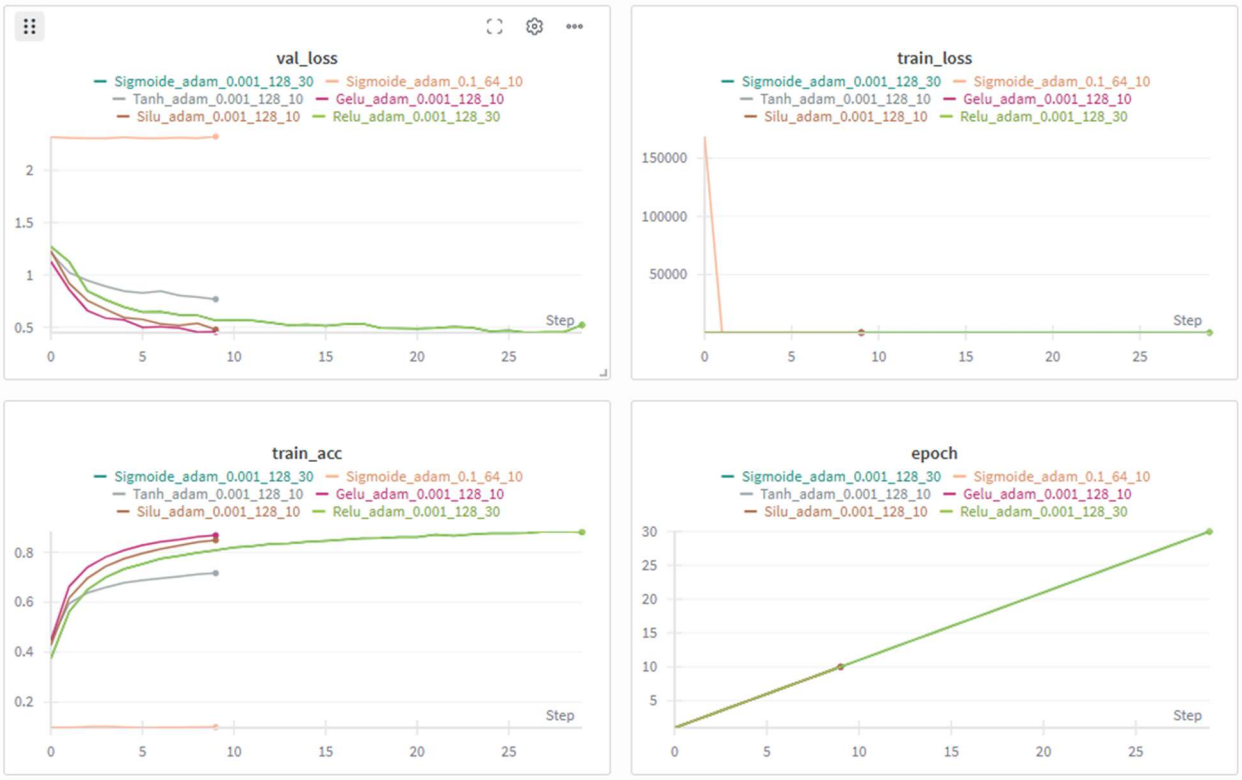
Nesterov-SGD,


nest
.*
Name
5 visualized
Sigmoide_nesterov_0.1_128_30
Tanh_nesterov_0.1_64_30
Gelu_nesterov_01_64_30
Silu_nesterov_0.1_64_30
Relu_nesterov_0.1_64_30
Charts
6
val_acc v. Step
val_acc
Step
0.4
0.35
0.3
0.25
0.2
0.15
0.1
-25
-20
-15
-10
-5
0
5
10
15
20
25
val_acc
Sigmoide_nesterov_0.1_128_30
Tanh_nesterov_0.1_64_30
Gelu_nesterov_01_64_30
Silu_nesterov_0.1_64_30
Relu_nesterov_0.1_64_30
Step

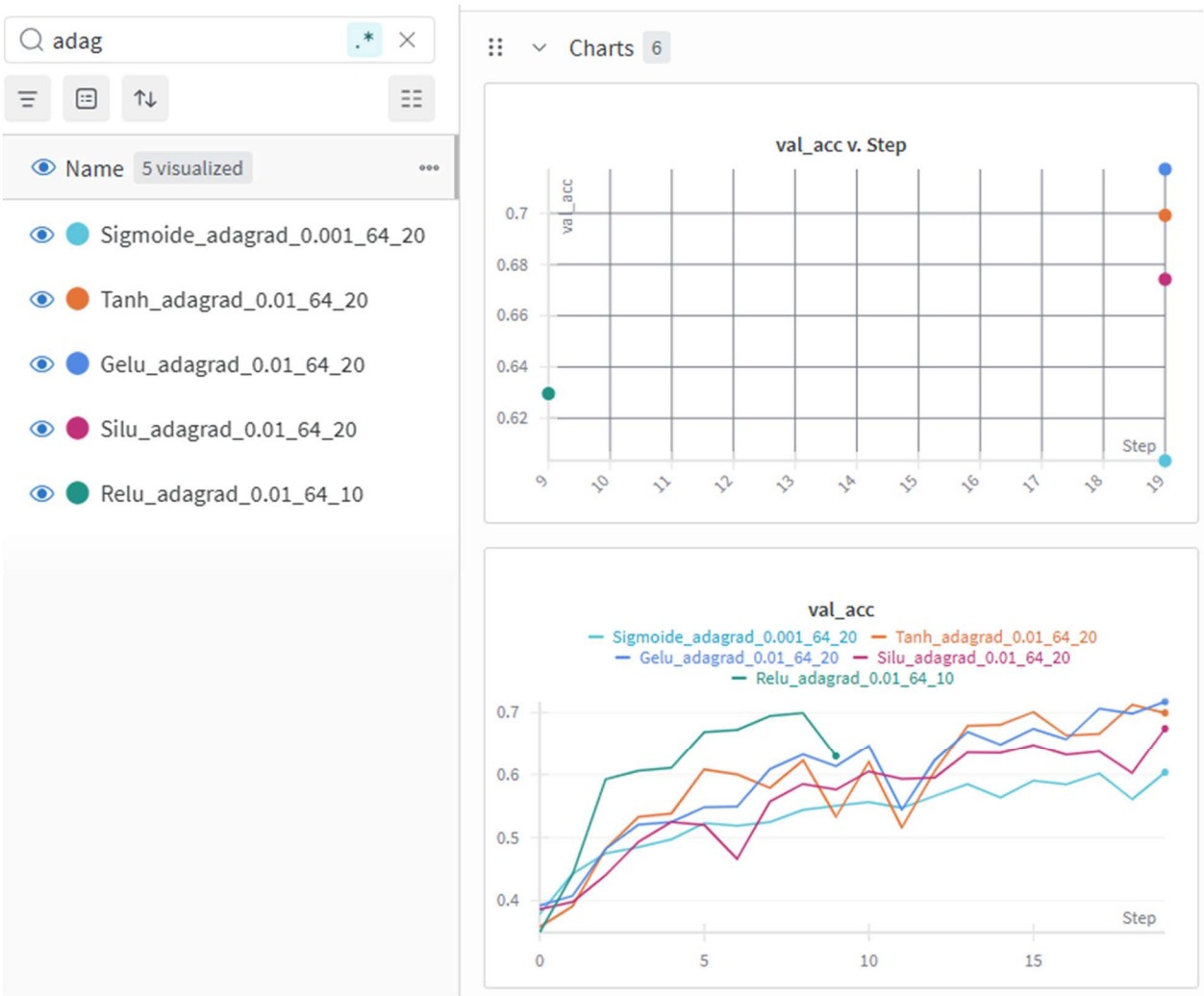

val_loss
Sigmoide_nesterov_0.1_128_30
Tanh_nesterov_0.1_64_30
Gelu_nesterov_01_64_30
Silu_nesterov_0.1_64_30
Relu_nesterov_0.1_64_30
Step
train_loss
Sigmoide_nesterov_0.1_128_30
Tanh_nesterov_0.1_64_30
Gelu_nesterov_01_64_30
Silu_nesterov_0.1_64_30
Relu_nesterov_0.1_64_30
Step
train_acc
Sigmoide_nesterov_0.1_128_30
Tanh_nesterov_0.1_64_30
Gelu_nesterov_01_64_30
Silu_nesterov_0.1_64_30
Relu_nesterov_0.1_64_30
Step
epoch
Sigmoide_nesterov_0.1_128_30
Tanh_nesterov_0.1_64_30
Gelu_nesterov_01_64_30
Silu_nesterov_0.1_64_30
Relu_nesterov_0.1_64_30
Step

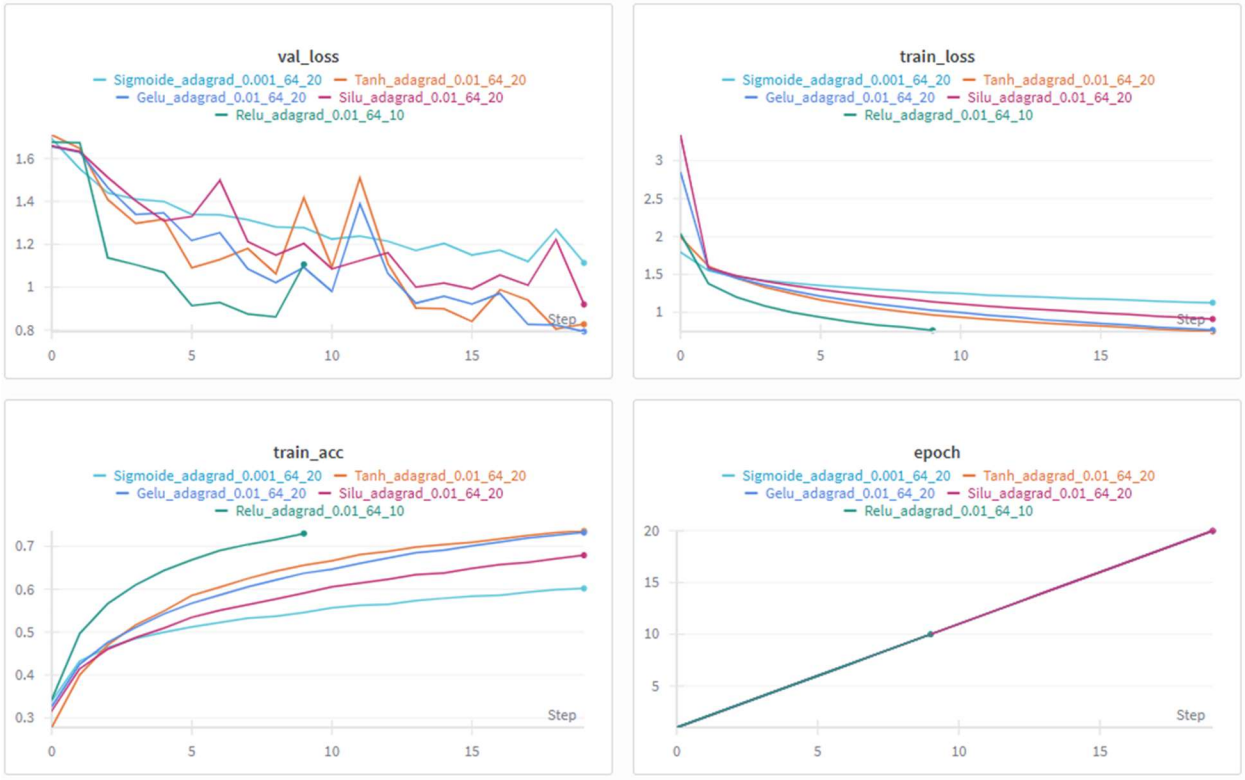
Adam


_adam
.*
Name
6 visualized
Sigmoide_adam_0.001_128_30
Sigmoide_adam_0.1_64_10
Tanh_adam_0.001_128_10
Gelu_adam_0.001_128_10
Silu_adam_0.001_128_10
Relu_adam_0.001_128_30
Charts
6
val_acc v. Step
val_acc
0.8
0.6
0.4
0.2
Step
10
12
14
16
18
20
22
24
26
28
val_acc
Sigmoide_adam_0.001_128_30
Sigmoide_adam_0.1_64_10
Tanh_adam_0.001_128_10
Gelu_adam_0.001_128_10
Silu_adam_0.001_128_10
Relu_adam_0.001_128_30
0.8
0.6
0.4
0.2
Step
0
5
10
15
20
25

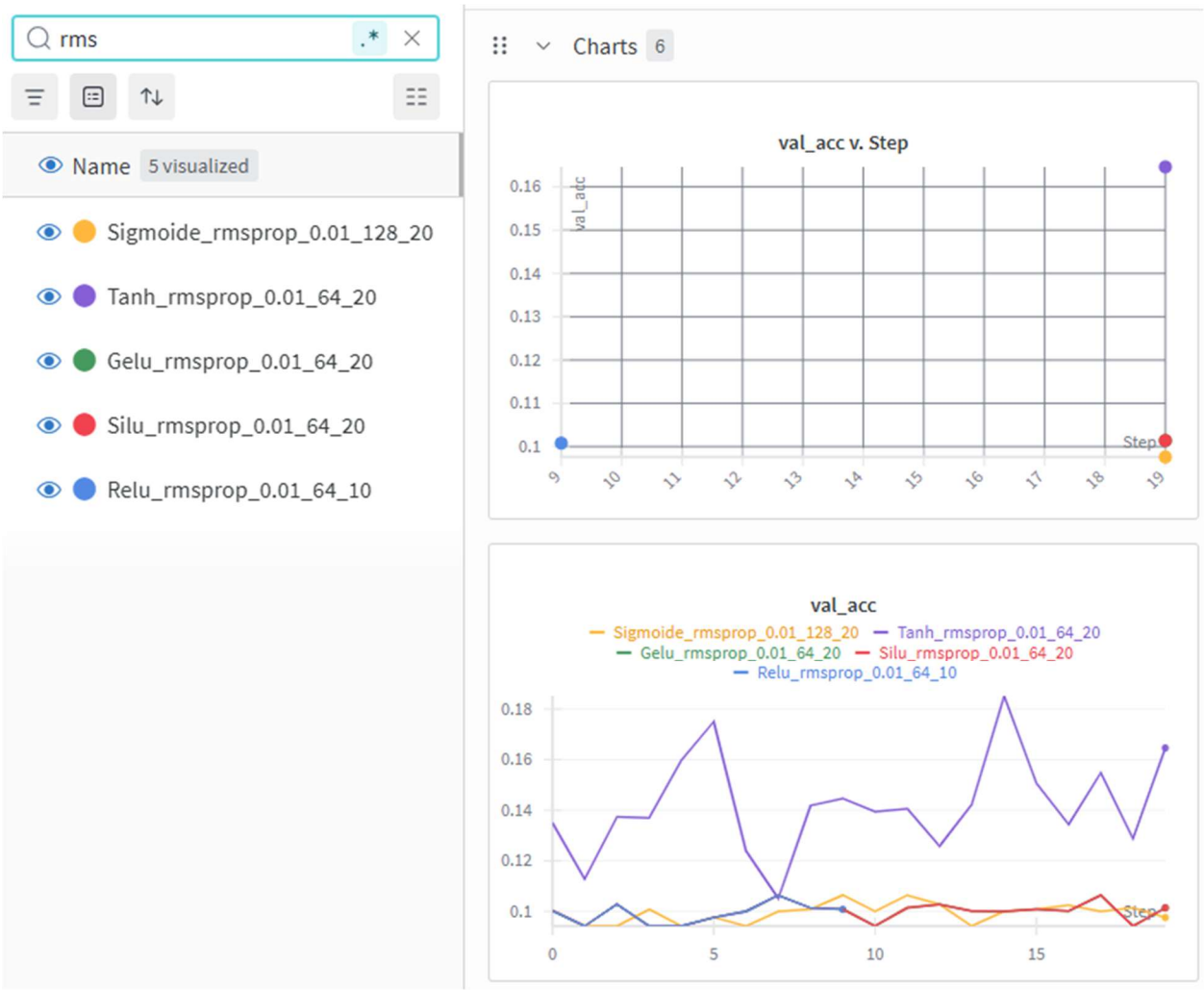

val_loss
Sigmoide_adam_0.001_128_30
Sigmoide_adam_0.1_64_10
Tanh_adam_0.001_128_10
Gelu_adam_0.001_128_10
Silu_adam_0.001_128_10
Relu_adam_0.001_128_30
Step
train_loss
Sigmoide_adam_0.001_128_30
Sigmoide_adam_0.1_64_10
Tanh_adam_0.001_128_10
Gelu_adam_0.001_128_10
Silu_adam_0.001_128_10
Relu_adam_0.001_128_30
Step
train_acc
Sigmoide_adam_0.001_128_30
Sigmoide_adam_0.1_64_10
Tanh_adam_0.001_128_10
Gelu_adam_0.001_128_10
Silu_adam_0.001_128_10
Relu_adam_0.001_128_30
Step
epoch
Sigmoide_adam_0.001_128_30
Sigmoide_adam_0.1_64_10
Tanh_adam_0.001_128_10
Gelu_adam_0.001_128_10
Silu_adam_0.001_128_10
Relu_adam_0.001_128_30
Step

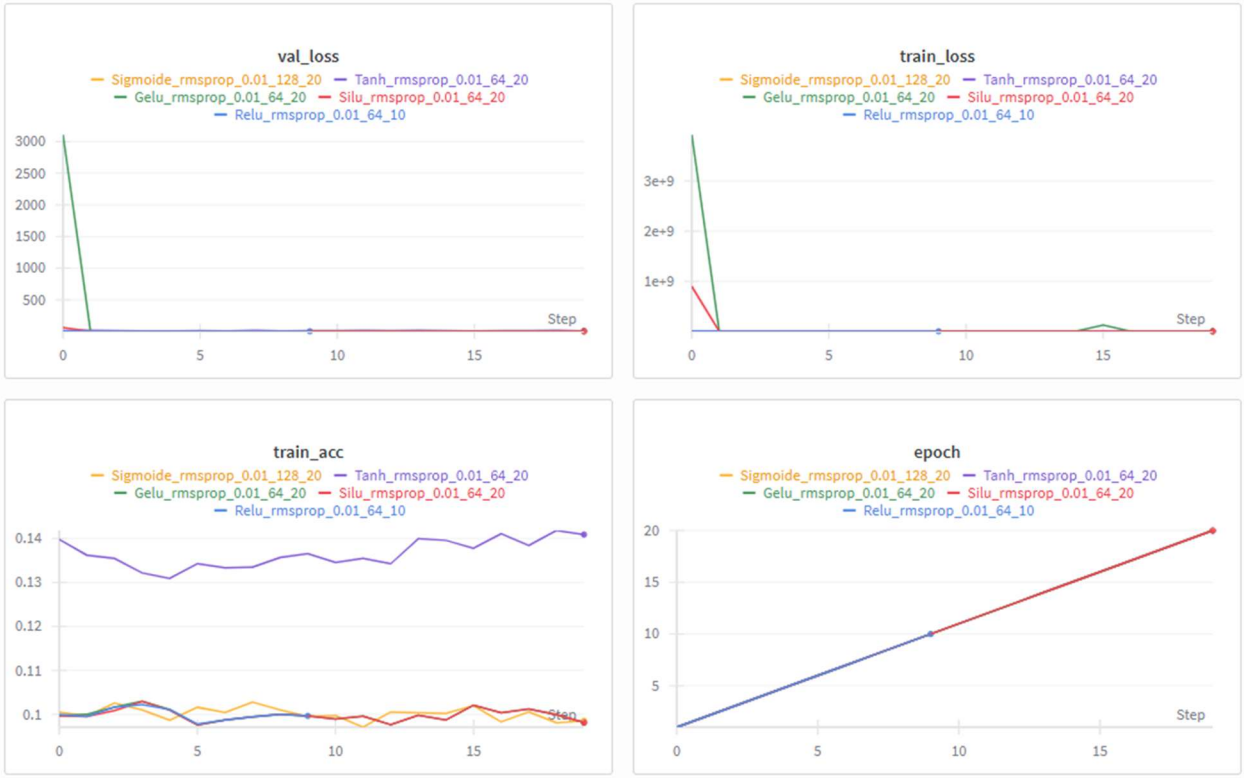
Adagrad,


adag
.*
Name
5 visualized
Sigmoide_adagrad_0.001_64_20
Tanh_adagrad_0.01_64_20
Gelu_adagrad_0.01_64_20
Silu_adagrad_0.01_64_20
Relu_adagrad_0.01_64_10
Charts
6
val_acc v. Step
val_acc
0.7
0.68
0.66
0.64
0.62
Step
9
10
11
12
13
14
15
16
17
18
19
val_acc
Sigmoide_adagrad_0.001_64_20
Tanh_adagrad_0.01_64_20
Gelu_adagrad_0.01_64_20
Silu_adagrad_0.01_64_20
Relu_adagrad_0.01_64_10
0.7
0.6
0.5
0.4
Step
0
5
10
15

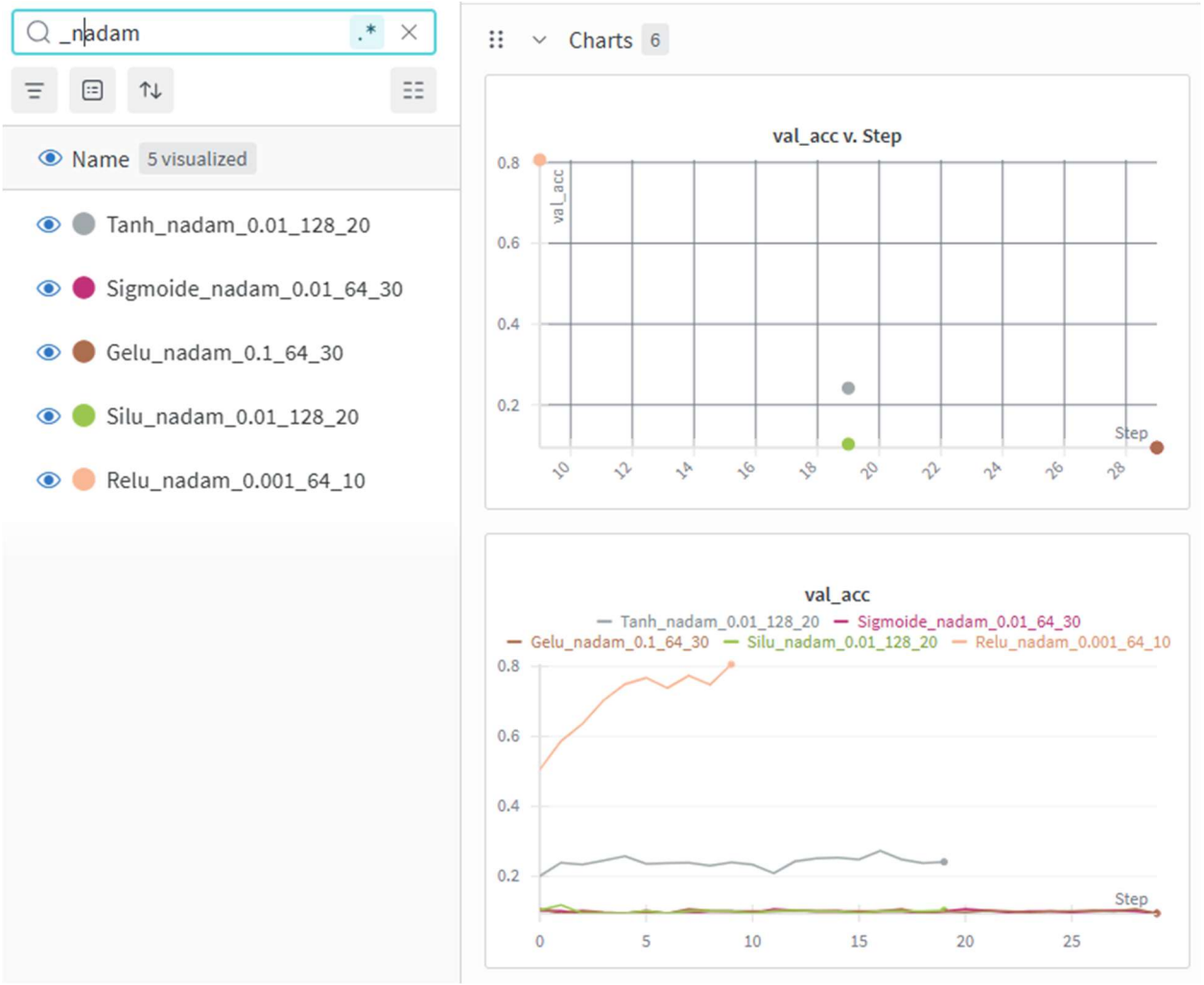

val_loss
Sigmoide_adagrad_0.001_64_20
Tanh_adagrad_0.01_64_20
Gelu_adagrad_0.01_64_20
Silu_adagrad_0.01_64_20
Relu_adagrad_0.01_64_10
Step
train_loss
Sigmoide_adagrad_0.001_64_20
Tanh_adagrad_0.01_64_20
Gelu_adagrad_0.01_64_20
Silu_adagrad_0.01_64_20
Relu_adagrad_0.01_64_10
Step
train_acc
Sigmoide_adagrad_0.001_64_20
Tanh_adagrad_0.01_64_20
Gelu_adagrad_0.01_64_20
Silu_adagrad_0.01_64_20
Relu_adagrad_0.01_64_10
Step
epoch
Sigmoide_adagrad_0.001_64_20
Tanh_adagrad_0.01_64_20
Gelu_adagrad_0.01_64_20
Silu_adagrad_0.01_64_20
Relu_adagrad_0.01_64_10
Step

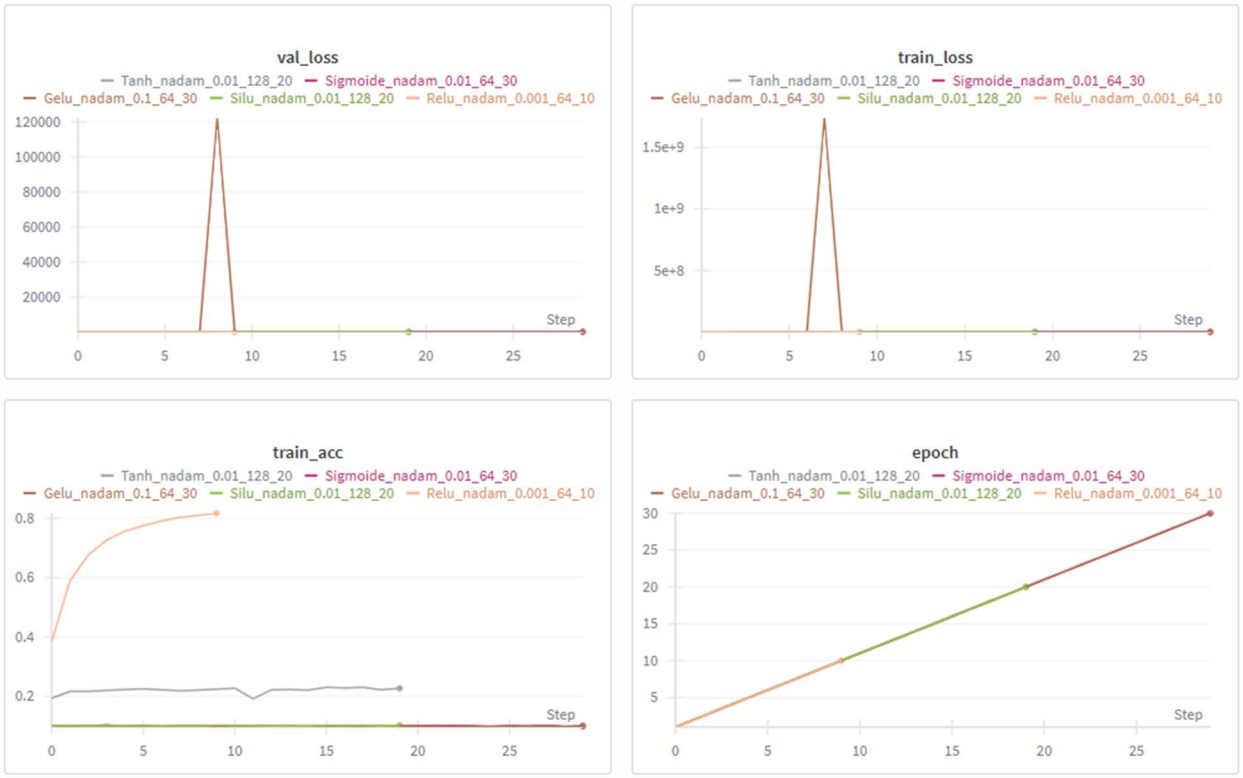
RMSprop


rms
.*
Name
5 visualized
Sigmoide_rmsprop_0.01_128_20
Tanh_rmsprop_0.01_64_20
Gelu_rmsprop_0.01_64_20
Silu_rmsprop_0.01_64_20
Relu_rmsprop_0.01_64_10
Charts
6
val_acc v. Step
val_adc
0.16
0.15
0.14
0.13
0.12
0.11
0.1
Step
9
10
11
12
13
14
15
16
17
18
19
val_acc
Sigmoide_rmsprop_0.01_128_20
Tanh_rmsprop_0.01_64_20
Gelu_rmsprop_0.01_64_20
Silu_rmsprop_0.01_64_20
Relu_rmsprop_0.01_64_10
0.18
0.16
0.14
0.12
0.1
Step
0
5
10
15


val_loss
Sigmoide_rmsprop_0.01_128_20
Tanh_rmsprop_0.01_64_20
Gelu_rmsprop_0.01_64_20
Silu_rmsprop_0.01_64_20
Relu_rmsprop_0.01_64_10
3000
2500
2000
1500
1000
500
Step
0
5
10
15
train_loss
Sigmoide_rmsprop_0.01_128_20
Tanh_rmsprop_0.01_64_20
Gelu_rmsprop_0.01_64_20
Silu_rmsprop_0.01_64_20
Relu_rmsprop_0.01_64_10
3e+9
2e+9
1e+9
Step
0
5
10
15
train_acc
Sigmoide_rmsprop_0.01_128_20
Tanh_rmsprop_0.01_64_20
Gelu_rmsprop_0.01_64_20
Silu_rmsprop_0.01_64_20
Relu_rmsprop_0.01_64_10
0.14
0.13
0.12
0.11
0.1
Step
0
5
10
15
epoch
Sigmoide_rmsprop_0.01_128_20
Tanh_rmsprop_0.01_64_20
Gelu_rmsprop_0.01_64_20
Silu_rmsprop_0.01_64_20
Relu_rmsprop_0.01_64_10
20
15
10
5
Step
0
5
10
15

Nadam


_nadam
.*
Name
5 visualized
Tanh_nadam_0.01_128_20
Sigmoide_nadam_0.01_64_30
Gelu_nadam_0.1_64_30
Silu_nadam_0.01_128_20
Relu_nadam_0.001_64_10
Charts
6
val_acc v. Step
val_acc
Step
0.8
0.6
0.4
0.2
10
12
14
16
18
20
22
24
26
28
val_acc
Tanh_nadam_0.01_128_20
Sigmoide_nadam_0.01_64_30
Gelu_nadam_0.1_64_30
Silu_nadam_0.01_128_20
Relu_nadam_0.001_64_10
Step
0
5
10
15
20
25



Varying the Optimizer

The model was trained using **SGD**, **Nesterov-SGD**, **Adam**, **Adagrad**, **RMSprop**, and **Nadam** to observe differences in convergence speed and stability.

- **SGD:** Simple and consistent but converges slowly.
- **Nesterov-SGD:** Faster and smoother than SGD due to momentum look-ahead.
- **Adam:** Rapid convergence with adaptive learning rates but may overfit.
- **Adagrad:** Learns quickly initially but slows as learning rates shrink.
- **RMSprop:** Maintains balanced updates, effective for non-stationary data.
- **Nadam:** Combines Adam's adaptivity with Nesterov momentum for stable, fast convergence.

Summary:
Momentum-based and adaptive optimizers (**Adam**, **Nadam**) train faster, while **SGD** and **Nesterov-SGD** give more stable generalization

(c) Vary the batch size, number of epochs, and learning rate. Explain how the convergence speed and performance vary with these changes. (10)

Answer:

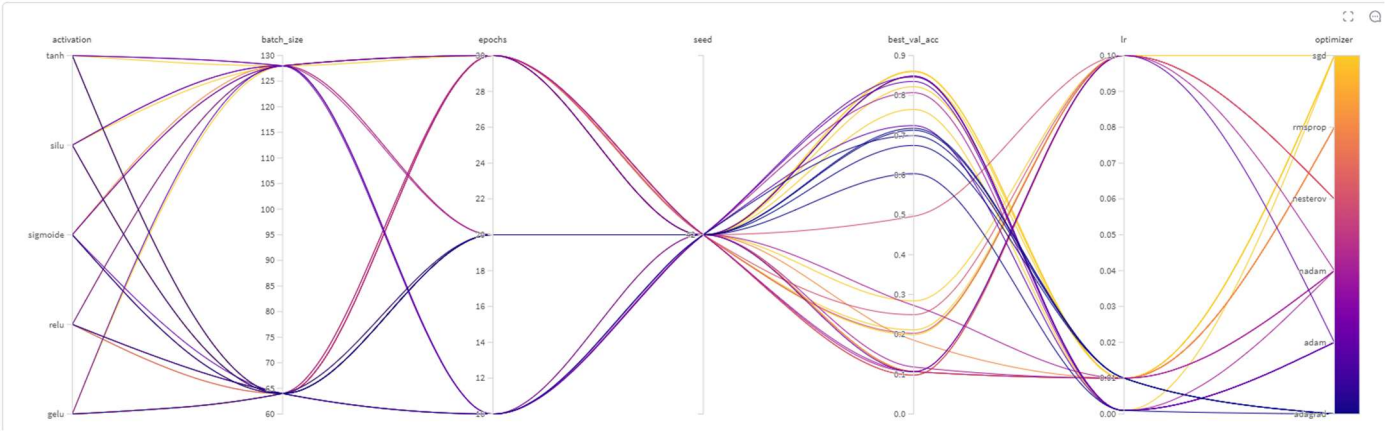
Model behavior was analyzed by varying batch size, number of epochs, and learning rate.

Hyperparameter	effect	Optimal Choice
Batch Size	Small batches improved generalization but increased training noise; large batches gave smoother updates but sometimes reduced accuracy	128
Epochs	More epochs improved accuracy until overfitting appeared; around 20–30 epochs gave balanced results	30
Learning Rate	High values caused unstable training, while very low values slowed convergence; moderate rates achieved stable and faster learning	0.01

Question 3. Plots (10 points)

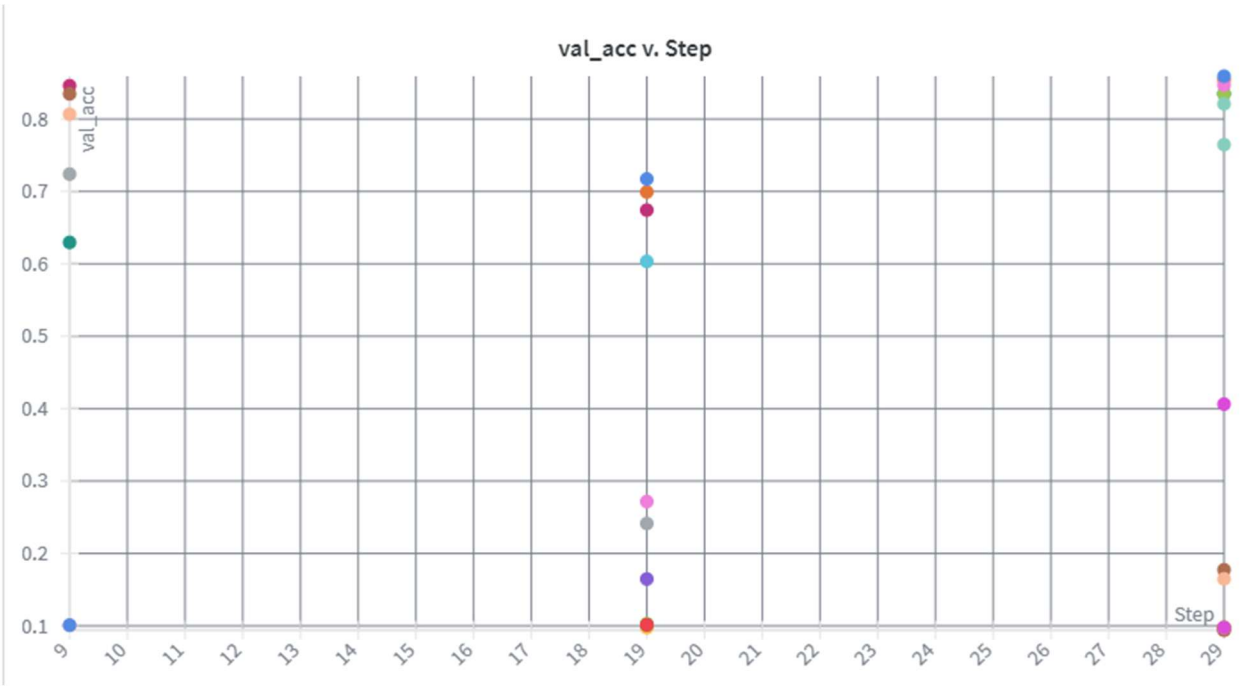
(a) Provide the W&B parallel-coordinate plot that shows which configuration achieves what accuracy (a sample plot is given below).

Answer:



(b) Provide the validation accuracy vs. step (scatter) plot.

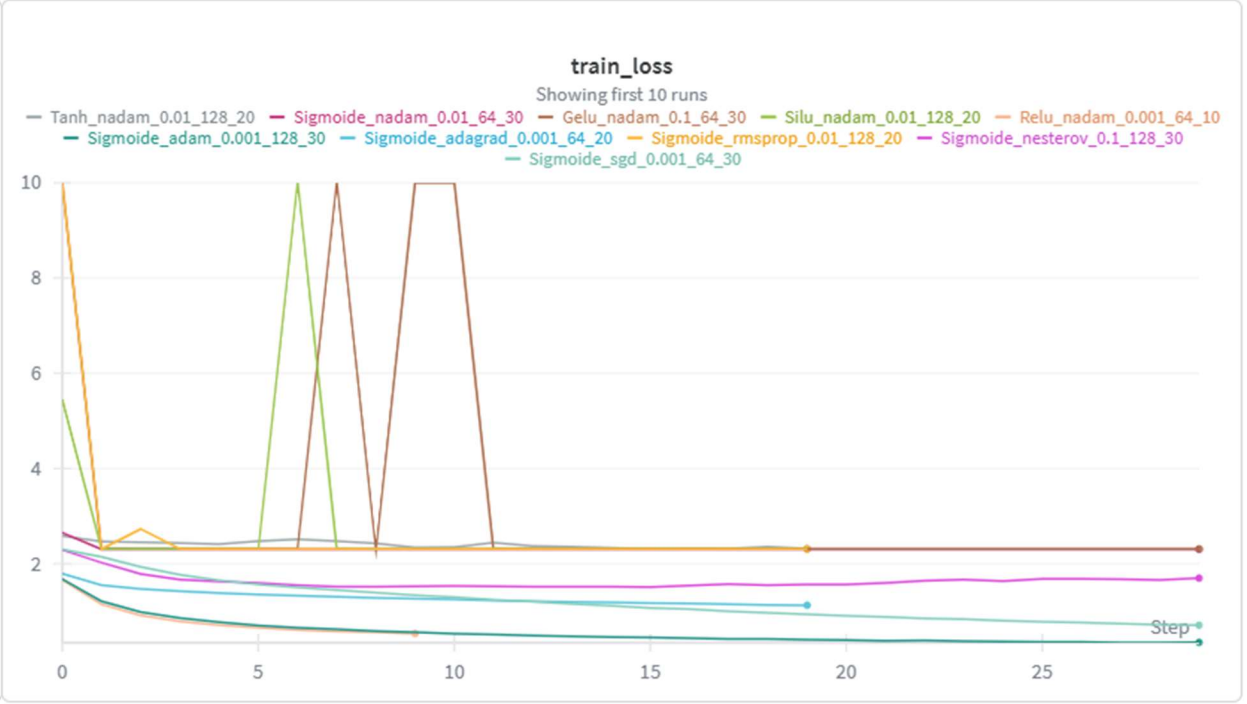
Answer:



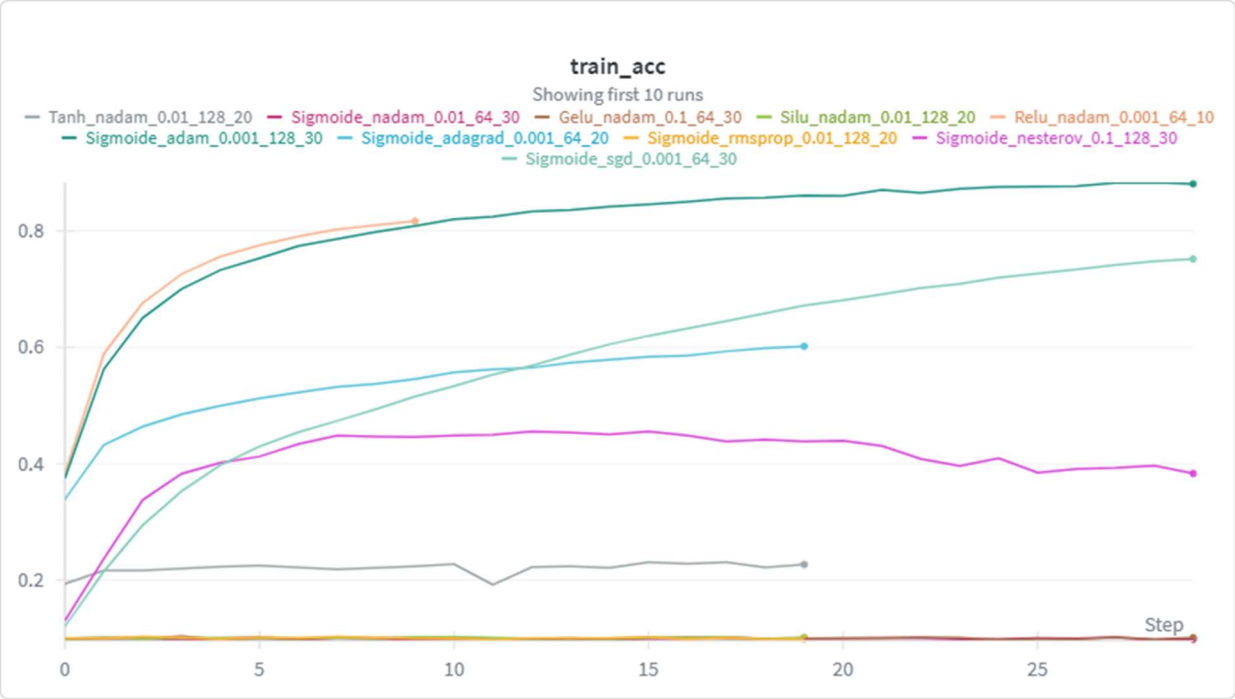
(c) Provide the plots for training loss, training accuracy, validation loss, and validation accuracy respectively.

Answer:

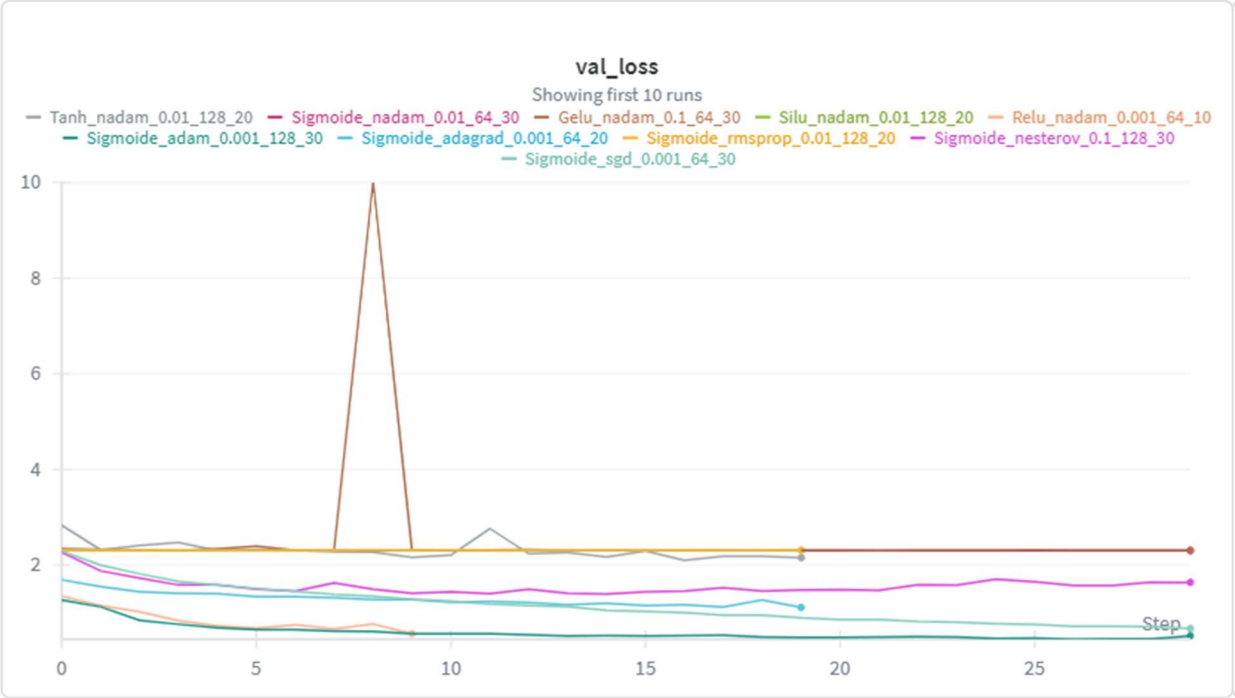
training loss



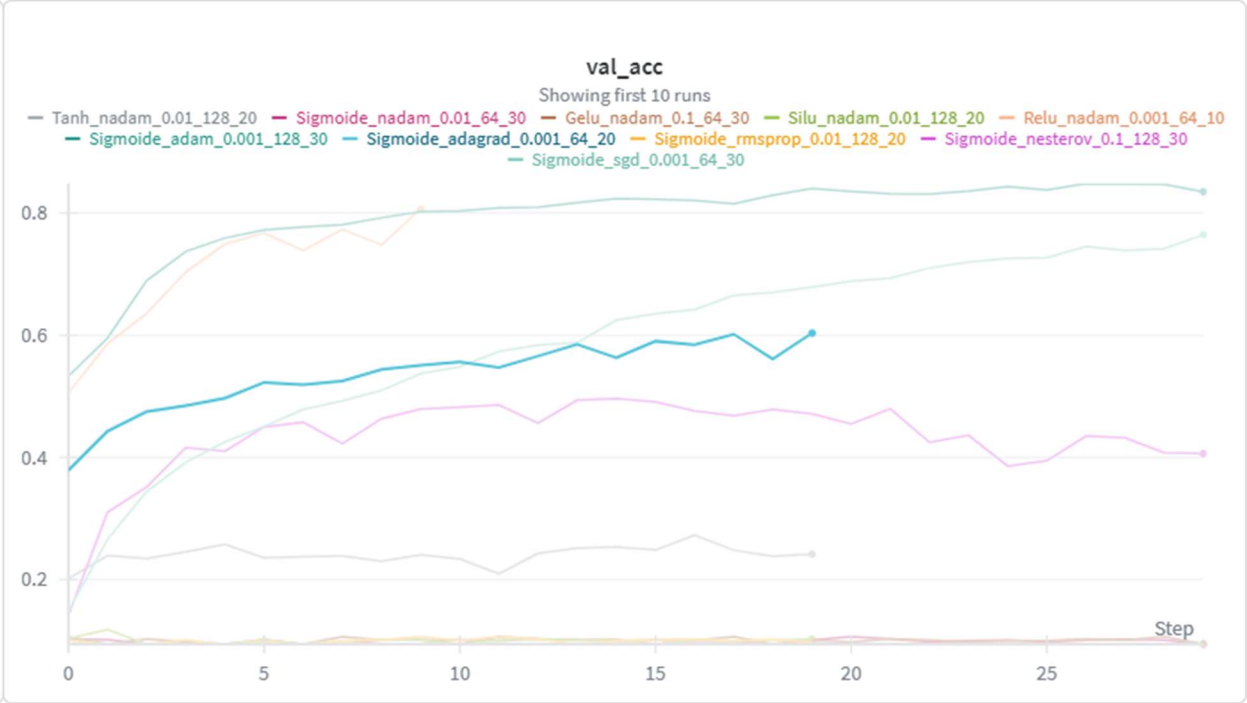
training accuracy



validation loss



Validation accuracy



Question 4. Final Model Performance (10 points)

Based on the W&B parallel plot, provide the configuration that achieved the best validation accuracy. Do not list multiple configurations. The configuration mentioned here will be verified by re-running your model. Ensure that this configuration indeed reproduces the reported best accuracy.

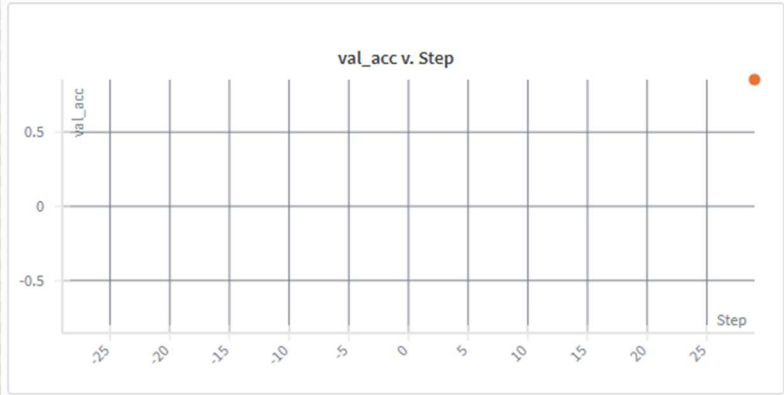
Answer:

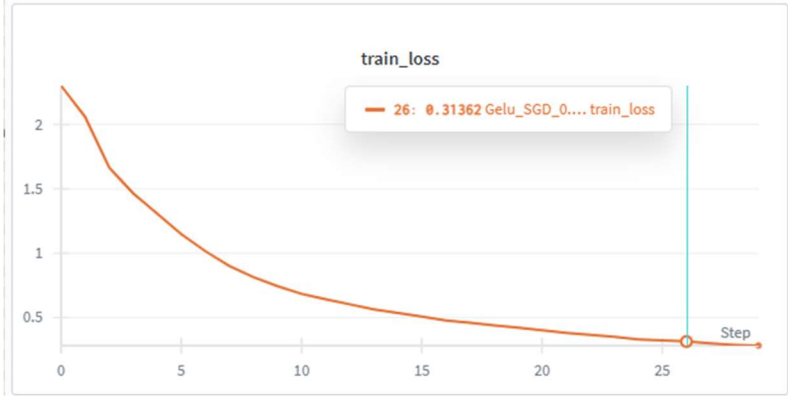
python train.py --activation gelu --optimizer sgd --lr 0.01 --batch_size 128 --epochs 30

Run set 35 | top_val_acc 0

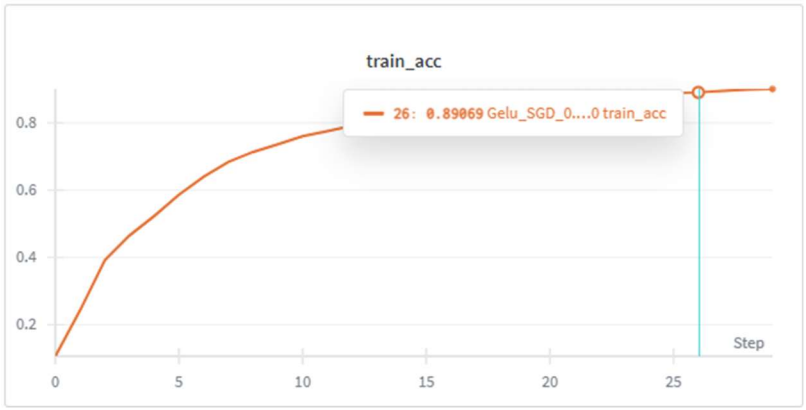
Name (0 visualized)	State	User	activation	batch_size	epochs	lr	no_wandb	seed	best_val_acc	epoch
Gelu_SGD_0.01_128_30	Finished	cs24m5	gelu	128	30	0.01	false	42	0.861	30

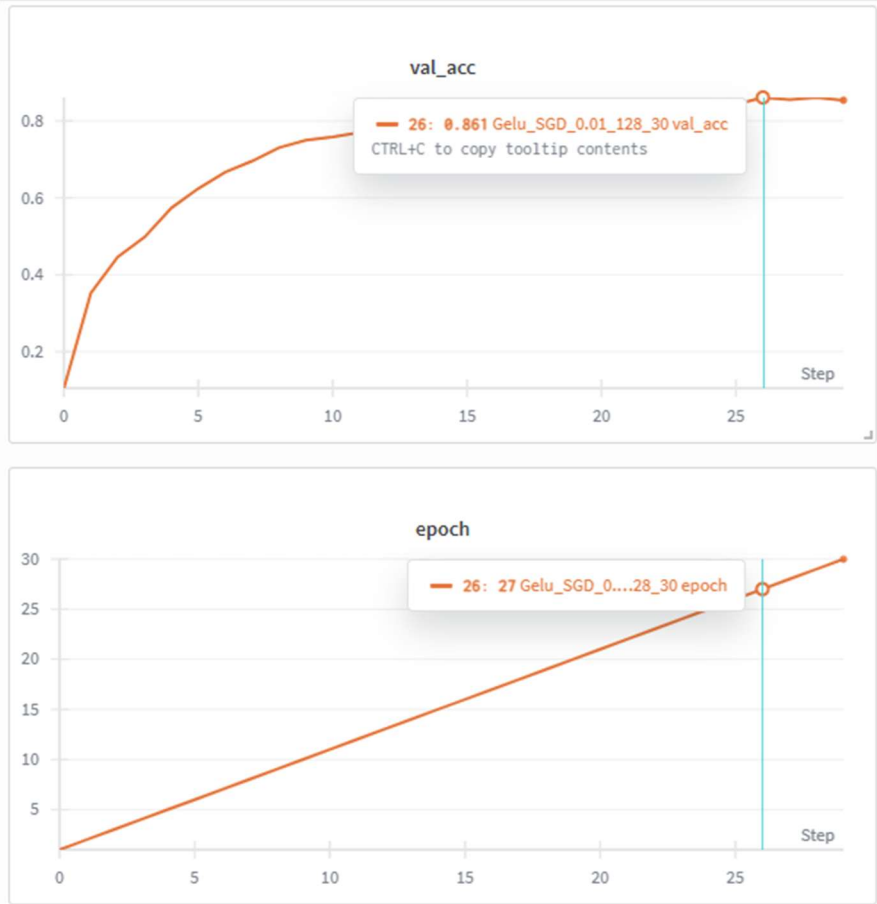
train_acc	train_loss	val_acc	val_loss
0.90118	0.28179	0.8538	0.43113






val_loss
26: 0.40375 Gelu_SGD_0....30 val_loss
2
1.5
1
0.5
Step
0
5
10
15
20
25


train_acc
26: 0.89069 Gelu_SGD_0....0 train_acc
0.8
0.6
0.4
0.2
Step
0
5
10
15
20
25



Question 5. Reproducibility and Repository (10 points)

(a) Provide clean, modular, and well-commented code with clear separation of training, evaluation, and compression modules. (4)

Answer:

github: https://github.com/sakumar5/VGG6_Assignment1/tree/main/cs24m533

(b) Include a README with exact commands, environment details, and dependency versions. Also include seed configuration. (4)

Answer:

github:https://github.com/sakumar5/VGG6_Assignment1/blob/main/cs24m533/README.md

(c) Upload the trained model to the GitHub repository and provide the GitHub repository link inside the pdf (2)

Answer:

github: https://github.com/sakumar5/VGG6_Assignment1/tree/main/cs24m533

wandb report: https://api.wandb.ai/links/cs24m533-iit-madras/uoy2sdz7