

Link to Github repo: <https://github.com/alyssahuang02/CS197-Pset4>

Exercise 1

Added import wandb lines at the top

```
import wandb
wandb.login()
```

Initializing project and config information for wandb

```
wandb.init(
    # Set the project where this run will be logged
    project="cs197-pset4",
    # Track hyperparameters and run metadata
    config=config_dict)
```

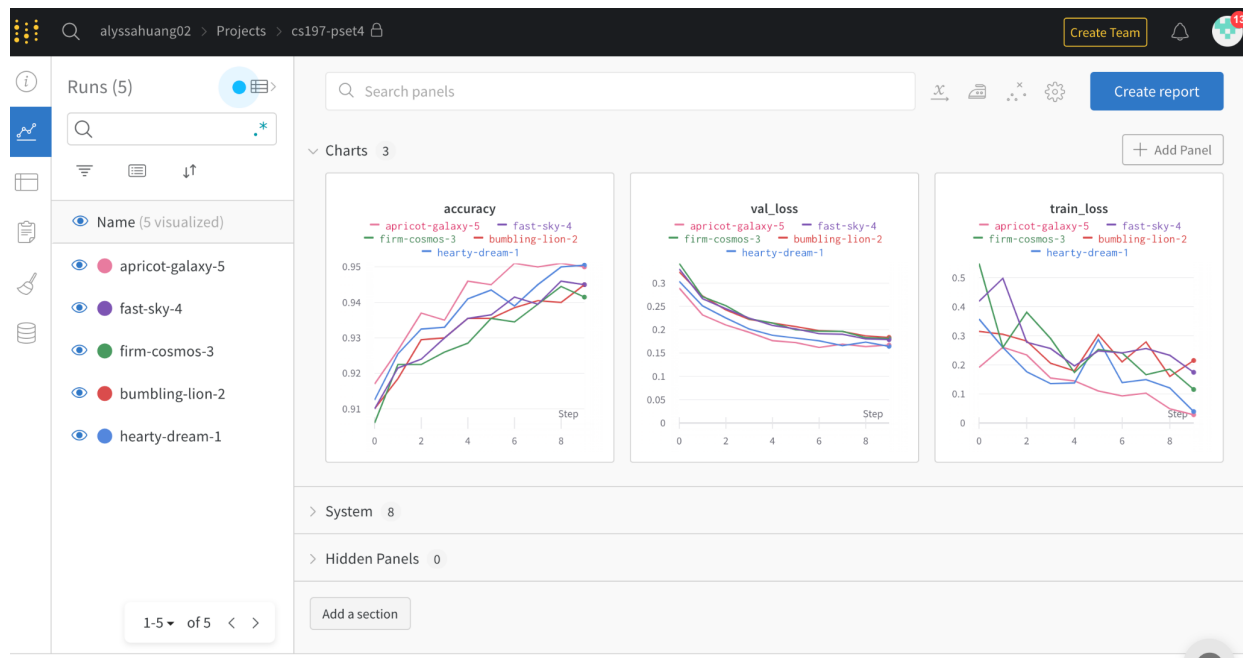
Logging train loss, val loss, and accuracy for wandb

```
wandb.log({"train_loss": train_loss, "val_loss": val_loss, "accuracy": accuracy})
```

Finishing the wandb run

```
wandb.finish()
```

Here are the charts in wandb:



Exercise 2

Added function to create the table with images, predictions, and labels

```
def log_image_table(images, predicted, labels):  
    "Log a wandb.Table with (image, prediction, label)"  
    table = wandb.Table(  
        columns=["image", "prediction", "label"]  
    )  
    for img, pred, label in \  
        zip(images.to("cpu"),  
            predicted.to("cpu"),  
            labels.to("cpu")):  
        table.add_data(  
            wandb.Image(img[0].numpy()*255), pred, label)  
    wandb.log({"predictions_table": table}, commit=False)
```

Modified validate model to call the log_image_table if we're in the last epoch for images in a given batch id

```
if i == batch_idx and log_images:  
    log_image_table(  
        images,  
        predicted,  
        labels)
```

Here are the tables in wandb:

The screenshot shows the wandb interface for a project named 'cs197-pset4'. On the left, a sidebar lists 11 runs, including 'valiant-frost-11', 'silver-flower-10', 'ruby-fog-9', 'misunderstood-sun-8', 'icy-dew-7', 'sage-darkness-6', 'apricot-galaxy-5', 'fast-sky-4', and 'humblime-lion-2'. The main panel displays a table titled 'runs.summary["predictions_table"]' with columns 'image', 'prediction', and 'label'. The table contains 6 rows of data, showing handwritten digits and their corresponding predictions and labels. The interface also includes a search bar, a 'Create report' button, and a 'My Workspace' section at the bottom.

	image	prediction	label
1		7	7
2		1	1
3		0	0
4		5	5
5		9	9
6		0	0

Exercise 3

Import necessary packages

```
import os
import numpy as np
import logging
logging.getLogger().setLevel(logging.INFO)
```

Create CheckpointSaver class

```
class CheckpointSaver:
    def __init__(self, dirpath, decreasing=True, top_n=5):
        """
        dirpath: Directory path where to store all model weights
        decreasing: If decreasing is `True`, then lower metric is better
        top_n: Total number of models to track based on validation metric value
        """
        if not os.path.exists(dirpath): os.makedirs(dirpath)
        self.dirpath = dirpath
        self.top_n = top_n
        self.decreasing = decreasing
        self.top_model_paths = []
        self.best_metric_val = np.Inf if decreasing else -np.Inf

    def __call__(self, model, epoch, metric_val):
        model_path = os.path.join(self.dirpath, model.__class__.__name__ +
f'_epoch{epoch}.pt')
        save = metric_val < self.best_metric_val if self.decreasing else
metric_val > self.best_metric_val
        if save:
            logging.info(f"Current metric value better than {metric_val} better than
best {self.best_metric_val}, saving model at {model_path}, & logging model weights to
W&B.")
            self.best_metric_val = metric_val
            torch.save(model.state_dict(), model_path)
            self.log_artifact(f'model-ckpt-epoch-{epoch}.pt', model_path, metric_val)
            self.top_model_paths.append({'path': model_path, 'score': metric_val})
            self.top_model_paths = sorted(self.top_model_paths, key=lambda o:
o['score'], reverse=not self.decreasing)
            if len(self.top_model_paths) > self.top_n:
                self.cleanup()

    def log_artifact(self, filename, model_path, metric_val):
```

```

        artifact = wandb.Artifact(filename, type='model', metadata={'Validation score':
metric_val})
        artifact.add_file(model_path)
        wandb.run.log_artifact(artifact)

    def cleanup(self):
        to_remove = self.top_model_paths[self.top_n:]
        logging.info(f"Removing extra models.. {to_remove}")
        for o in to_remove:
            os.remove(o['path'])
        self.top_model_paths = self.top_model_paths[:self.top_n]

```

Initializing CheckpointSaver

```

checkpoint_saver = CheckpointSaver(dirpath='./model_weights', decreasing=True,
top_n=3)

```

Updating CheckpointSaver

```

checkpoint_saver(model, epoch, val_loss)

```

Logs show that model checkpoints are being updated properly

```

wandb: currently logged in as: bty35sundung021 Use 'wandb login --relogin' to force relogin
INFO:root:Current metric value better than 0.20425148379802704 better than best 0.21572053372859956, saving model at ./model_weights/Sequential_epoch5.pt, & logging model weights to W&B.
INFO:root:Removing extra models.. [{'path': './model_weights/Sequential_epoch2.pt', 'score': 0.23770914578437805}]
Train Loss: 0.214, Valid Loss: 0.204251, Accuracy: 0.94

```

Exercise 4

Initializing the sweep configuration

```
sweep_configuration = {
    "name": "Hyperparameter Sweep",
    "method": "grid",
    "metric": {"name": "val_loss", "goal": "minimize"},
    "parameters": {
        "batch_size": {
            "values": [100, 150, 200]
        },
        "epochs": {
            "values": [5, 10, 15]
        },
        "lr": {
            "values": [1e-2, 1e-3, 1e-4]
        }
    }
}
```

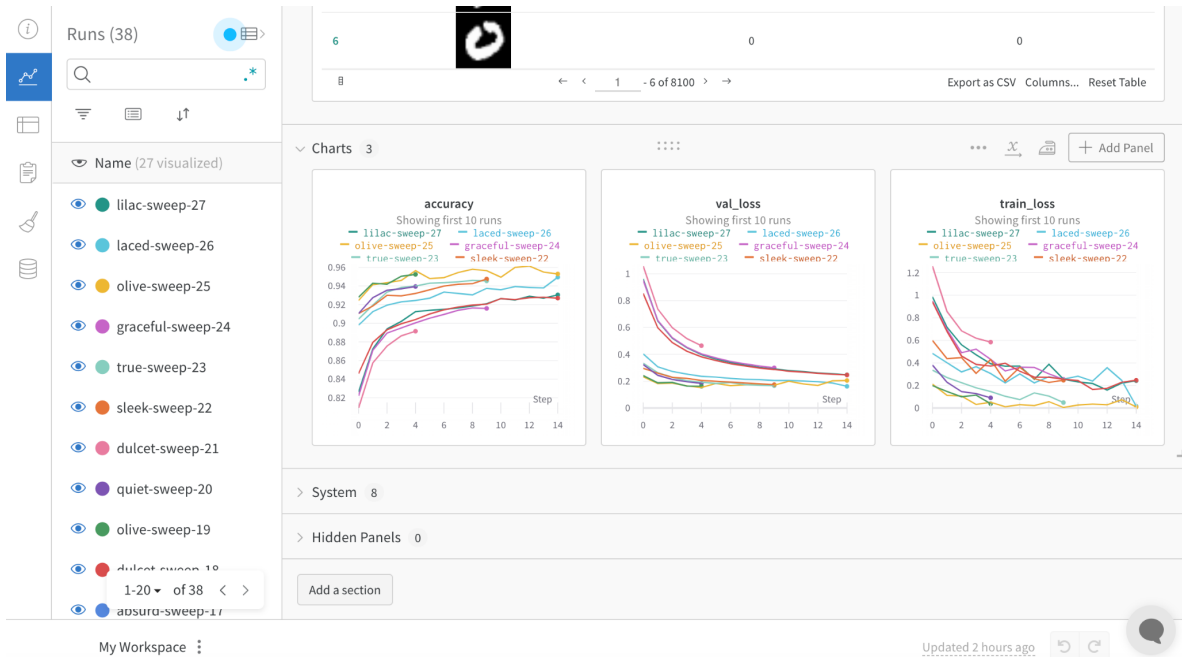
Using sweep config values

```
config_dict = {
    "epochs": wandb.config.epochs,
    "batch_size": wandb.config.batch_size,
    "lr": wandb.config.lr,
    "dropout": random.uniform(0.01, 0.80),
}
```

Initializing and running the sweep

```
sweep_id = wandb.sweep(sweep_configuration, project="cs197-pset4")
wandb.agent(sweep_id, function=train)
```

Sweeps visualized in wandb



Exercise 5

config.yaml

```
_target_: torch.nn.Sequential
_args_:
- _target_: torch.nn.Linear
  in_features: 9216
  out_features: 100

- _target_: torch.nn.Linear
  in_features: ${..[0].out_features}
  out_features: 10
```

train.py

```
from omegaconf import DictConfig
import hydra
from hydra.utils import instantiate

@hydra.main(version_base=None, config_path="./configs", config_name="config")
def run(cfg: DictConfig):
    opt = instantiate(cfg)
    print(opt)

if __name__ == "__main__":
    run()
```

Verified that running python train.py outputted in the right result

```
● (lec8) alyssahuang@dhcp-10-250-31-248 CS197 Pset4 % python train.py
Sequential(
  (0): Linear(in_features=9216, out_features=100, bias=True)
  (1): Linear(in_features=100, out_features=10, bias=True)
)
```


Exercise 6

config.yaml

```
defaults:  
  - classifier: small
```

small.yaml

```
_target_: torch.nn.Sequential  
_args_:  
  - _target_: torch.nn.Linear  
    in_features: 9216  
    out_features: 100  
  
  - _target_: torch.nn.Linear  
    in_features: ${..[0].out_features}  
    out_features: 10
```

large.yaml

```
_target_: torch.nn.Sequential  
_args_:  
  - _target_: torch.nn.Linear  
    in_features: 9216  
    out_features: 2040  
  
  - _target_: torch.nn.Linear  
    in_features: ${..[0].out_features}  
    out_features: 300  
  - _target_: torch.nn.Linear  
    in_features: ${..[1].out_features}  
    out_features: 10
```

Folder structure:

```
+--configs/  
| +--config.yaml  
| +--classifiers/  
|   +--small.yaml  
|   +--large.yaml
```

Verified that running python train.py outputted in the right result

```
• (lec8) alyssahuang@dhcp-10-250-31-248 CS197 Pset4 % python train.py classifier=small
{'classifier': Sequential(
  (0): Linear(in_features=9216, out_features=100, bias=True)
  (1): Linear(in_features=100, out_features=10, bias=True)
)}
• (lec8) alyssahuang@dhcp-10-250-31-248 CS197 Pset4 % python train.py
{'classifier': Sequential(
  (0): Linear(in_features=9216, out_features=100, bias=True)
  (1): Linear(in_features=100, out_features=10, bias=True)
)}
• (lec8) alyssahuang@dhcp-10-250-31-248 CS197 Pset4 % python train.py classifier=large
{'classifier': Sequential(
  (0): Linear(in_features=9216, out_features=2040, bias=True)
  (1): Linear(in_features=2040, out_features=300, bias=True)
  (2): Linear(in_features=300, out_features=10, bias=True)
)}
```

Extra Credit

<https://twitter.com/AlyssaH39378240/status/1582936244941516805>

← Thread

Alyssa Huang
@AlyssaH39378240

After going through the PyTorch Blitz tutorial, I came up with a question to help better understand the `requires_grad` flag in tensors. #harvardcs197

In the code snippet below, what does line 6 output? What does line 7 output? What does line 10 output?

```
1 import torch
2
3 x = torch.rand(2, 3)
4 y = torch.rand(2, 3, requires_grad=True)
5
6 print(x.requires_grad)
7 print(y.requires_grad)
8
9 z = x + y
10 print(z.requires_grad)
```

11:26 PM · Oct 19, 2022 · Twitter Web App

Alyssa Huang @AlyssaH39378240 · 1m
Replying to @AlyssaH39378240

Line 6 outputs False since `requires_grad` is False by default.

Line 7 outputs True since we explicitly set `requires_grad` to True.

Line 10 also outputs True since `z` was created based on a tensor with `requires_grad` set to True (`y`)

🗨️ ↺️ ❤️ ⬆️ 📄