

# Exercise 04 - Project Setup

Lightning, HuggingFace and Weights and Biases

[Chair XII for Natural Language Processing](#)

Prof. Dr. Goran Glavaš  
Benedikt Ebing

# Lightning

- Flexibility and convenience for prototyping
- Reorganization of our training/evaluation loop for convenience and flexibility:
  - LightningModule (Model + Optimization + Train/Dev/Test step)
    - [https://lightning.ai/docs/pytorch/stable/common/lightning\\_module.html#lightningmodule](https://lightning.ai/docs/pytorch/stable/common/lightning_module.html#lightningmodule)
  - LightningDataModule (Data)
    - <https://lightning.ai/docs/pytorch/stable/data/datamodule.html#lightningdatamodule>
  - Trainer (Training Loop)
    - <https://lightning.ai/docs/pytorch/stable/common/trainer.html>

## LightningModule

- Initialization: `__init__()` and `setup()`
- Train Loop: `training_step()`
- Validation Loop: `validation_step()`
- Test Loop: `test_step()`
- Optimizers and LR Schedulers: `configure_optimizers()`



# Training Loop

```
# put model in train mode and enable gradient calculation
model.train()
torch.set_grad_enabled(True)

for batch_idx, batch in enumerate(train_dataloader):
    loss = training_step(batch, batch_idx)

    # clear gradients
    optimizer.zero_grad()

    # backward
    loss.backward()

    # update parameters
    optimizer.step()
```



```
class LitClassifier(pl.LightningModule):
    def __init__(self, model):
        super().__init__()
        self.model = model

    def training_step(self, batch, batch_idx):
        x, y = batch
        logits = self.model(x)
        loss = F.cross_entropy(logits, y)
        return loss
```



## Validation Loop

```
# ...
for batch_idx, batch in enumerate(train_dataloader):
    loss = model.training_step(batch, batch_idx)
    loss.backward()
# ...

if validate_at_some_point:
    # disable grads + batchnorm + dropout
    torch.set_grad_enabled(False)
    model.eval()

    # ----- VAL LOOP -----
    for val_batch_idx, val_batch in enumerate(val_dataloader):
        val_out = model.validation_step(val_batch, val_batch_idx)
    # ----- VAL LOOP -----

    # enable grads + batchnorm + dropout
    torch.set_grad_enabled(True)
    model.train()
```



```
class LitModel(pl.LightningModule):
    def validation_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self.model(x)
        loss = F.cross_entropy(y_hat, y)
        self.log("val_loss", loss)
```

## LightningDataModule

- *prepare\_data(): how to download, tokenize, etc...*
- *setup(): how to split, define dataset, etc...*
- *train\_dataloader(): return dataloader for training*
- *val\_dataloader(): return dataloader for validation*
- *test\_dataloader(): return dataloader for testing*

# HuggingFace

- Install
  - pip install transformers
  - pip install datasets
- Loading models
  - [https://huggingface.co/docs/transformers/main/en/autoclass\\_tutorial](https://huggingface.co/docs/transformers/main/en/autoclass_tutorial)
  - Only allowed to use AutoModel (not AutoModelFor...)
  - Write your classification head from scratch
- Loading datasets
  - <https://huggingface.co/docs/datasets/loading>
  - Datasets:
    - Train, Validation: <https://huggingface.co/datasets/conll2003>
    - Test: <https://huggingface.co/datasets/masakhaner>
- Preprocessing datasets
  - <https://huggingface.co/docs/transformers/training#prepare-a-dataset>
  - <https://huggingface.co/docs/datasets/process>
  - [https://huggingface.co/docs/transformers/main\\_classes/data\\_collator](https://huggingface.co/docs/transformers/main_classes/data_collator)



## Weights and Biases

- <https://docs.wandb.ai/quickstart>
- Create an account (to get an API key)
- Install wandb
  - `pip install wandb`
- Log in: `wandb login`
- Track your experiments





## Resources

- Lightning: <https://lightning.ai/docs/pytorch/stable/>
- Huggingface: <https://huggingface.co/>
- Weights & Biases: <https://docs.wandb.ai/>