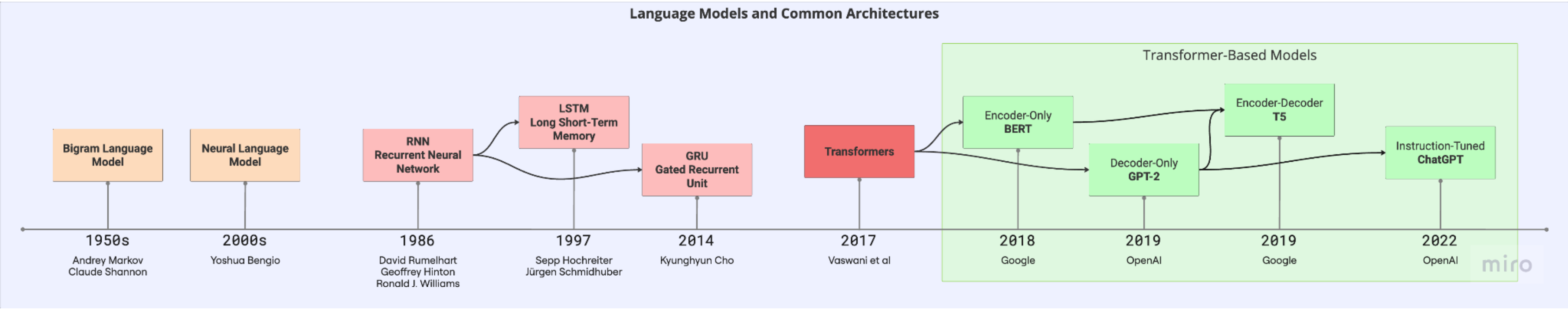
NLP4Web
Practice Session 12

Experiment Reproducibility and NLP Pipeline Debugging

Hovhannes Tamoyan
tamohannes.com

In previous sessions we covered



Experiment Reproducibility in NLP

Seeds

- Randomness in NLP can stem from factors like data shuffling or parameter initialization, and seeding helps mitigate variability.
- Set random seeds for libraries like **numpy**, **torch**, and **random** to ensure that experiments yield consistent results across runs.

```python
import random
import numpy as np
import torch

def set_seed(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)

set_seed(42)  # Set the seed for reproducibility
```

Environment Control

- Specify software and library versions, e.g.
    - **Python**
    - **PyTorch**
    - **Hugging Face**
        - **Transformers**
        - **Datasets**
        - etc
- **Docker** containers or environment management tools, e.g
    - **Conda**
    - **Venv**
- Can help create replicable environments

Version Control for Code and Data

- **Git** is essential for tracking code changes
- Tools like **DVC** help manage data files
- Combined, they allow for exact recreation of any experiment version.

Tracking Tools

- **Tensorboard** and **Aim** allow experiment logging:
  - Metric and hyperparameter tracking
  - Visualizing metrics e.g. loss, accuracy
  - And many more experiment details
- They also provide experiment comparison features, which are useful for understanding parameter change effects.

Best Practices

- Keep all configurations in a central location (e.g., config files).
- Use detailed documentation for experiment setups.
- Log all hyperparameters, model configurations, and other details for every run.

```python
# Log configuration details
config = {
    "seed": 42,
    "model_name": "distilbert-base-uncased",
    "batch_size": 8,
    "learning_rate": 1e-5,
    "num_epochs": 3
}
```

Experiment Tracking Tools

Tensorboard

- Visualizations cover
  - a range of debugging
  - reproducibility needs
    - metrics tracking
    - model graph visualization
    - hyperparameter tuning insights

Aim

- A lightweight, open-source alternative to Tensorboard
- Offers
  - an intuitive interface
  - experiment comparison
  - interactive search
  - customized metrics logging
  - focuses on simplifying reproducibility and debugging.

Aim - Core Components

- Aim SDK:
  - Python interface to define and track any object
  - Query tracked metadata with fully supported pythonic expressions
  - Integrations with essential ML tools and frameworks
- Aim Storage:
  - Modular (runs isolation - easily copy, move, delete runs)
  - Extendable (easily store any python object)
- Aim UI:
  - Metadata management and visualization
  - Deep comparison and exploration of multi-dimensional metadata

Aim - What to cover

- **Setup**: Local and remote tracking, the Run class
- **Tracking**: Tracking objects such as Metric, Text, Audio, and Image
- **Adapters**: Integrating Aim into an existing project
- **Migrate**: Importing runs from other trackers into Aim
- **UI**
  - **Runs Management**: Run explorer, bookmarks and tags
  - **Explorers**: Metrics, Parameters, and Text explorers

More in depth content for Experiment Tracking tools

https://tamohannes.com/docs/ExperimentTracking.pdf

NLP Pipeline Debugging Techniques

Pipeline Monitoring

- Continuously monitor intermediate blocks outputs, such as:
- Data preprocessor ins and outs
- Observe lengths/shapes and print random samples

```python
# Inspect a single tokenization example
sample_text = "This is a test sentence for tokenization."
tokenized_output = tokenizer(sample_text, padding="max_length", truncation=True, max_length=128)
print("Tokenized Output:", tokenized_output)
```

Architecture Monitoring

- Monitor intermediate outputs e.g. tokenization, embeddings, attention weights
- Ensure each stage works correctly by passing test tensors and comparing the output with the expected output

```
# Inspect a single tokenization example
sample_text = "This is a test sentence for tokenization."
tokenized_output = tokenizer(sample_text, padding="max_length", truncation=True, max_length=128)
print("Tokenized Output:", tokenized_output)
```

Error Analysis

- Identify common types of errors by using evaluation metrics and visualization tools
- Misclassifications, for instance, can reveal where a model's understanding might diverge from human intuition.

Running a Sanity Check on a Small Subset of Data

- If the model can overfit on a small subset of data, it's more likely that everything is working as expected.

```
# Using a small subset of data to check for overfitting
small_dataloader = DataLoader(tokenized_datasets["train"].select(range(10)), batch_size=2)

# Run a quick overfitting loop
for epoch in range(2):
    model.train()
    total_loss = 0

    for batch in small_dataloader:
        inputs = {k: v.to(device) for k, v in batch.items() if k != "label"}
        labels = batch["label"].to(device)

        outputs = model(**inputs, labels=labels)
        loss = outputs.loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / len(small_dataloader)
    print(f"Sanity Check Epoch {epoch+1}, Loss: {avg_loss:.4f}")
```

Gradient and Loss Inspection

- Use Experiment Tracking tool to monitor the following over time:
  - Gradients
  - Losses
  - Model parameter updates
- Sudden spikes or drops might indicate issues like vanishing or exploding gradients.

```python
# Training loop with gradient inspection
for epoch in range(num_epochs):
    model.train()
    total_loss = 0

    for batch in train_dataloader:
        inputs = {k: v.to(device) for k, v in batch.items() if k != "label"}
        labels = batch["label"].to(device)

        outputs = model(**inputs, labels=labels)
        loss = outputs.loss

        optimizer.zero_grad()
        loss.backward()

        # Gradient check
        for name, param in model.named_parameters():
            if param.grad is not None:
                avg_grad = param.grad.abs().mean().item()
                writer.add_scalar(f"Gradient/{name}", avg_grad, epoch)

        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / len(train_dataloader)
    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {avg_loss:.4f}")
    writer.add_scalar("Loss/train", avg_loss, epoch)
    run.track(avg_loss, name="train_loss", epoch=epoch)
```

Interpretability Tools

- For deeper debugging, interpretability libraries like **LIME** and **SHAP** allow analysis of model behavior, highlighting which input features most influence predictions.
- This can uncover biases or unexpected dependencies in the model.