Applied ML for veterinary epidemiologists

ISVEE pre-conference workshop - Day 4

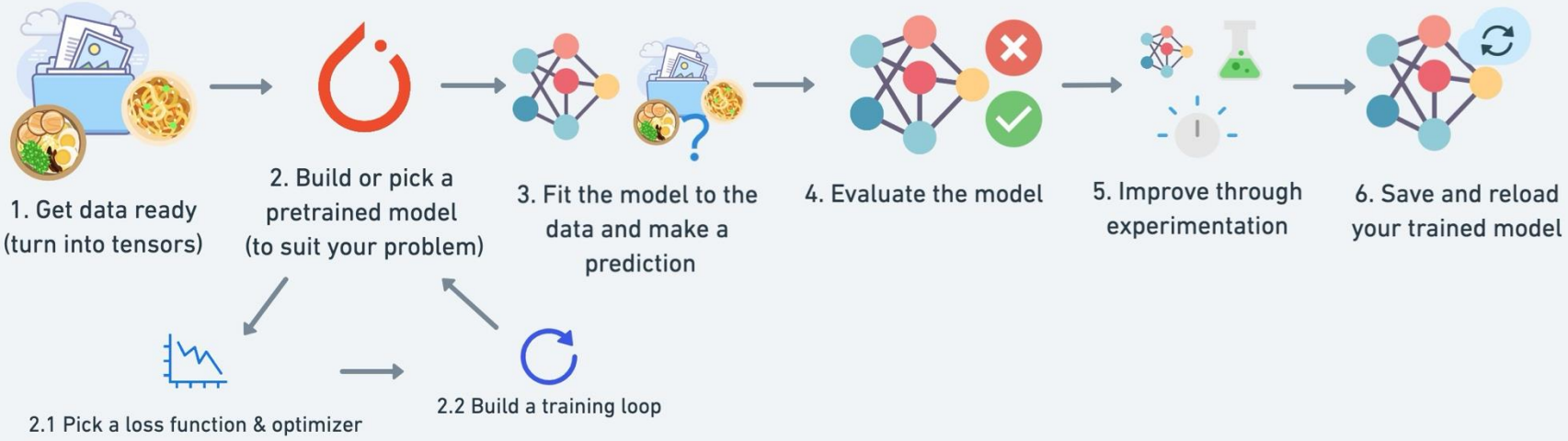
Dr Tom Brownlie

SEE TOMORROW FIRST
NEW ZEALAND TECHNOLOGY

ingenum


PyTorch

PyTorch workflow



Machine Learning: A game of two halves

Inputs

Numerical encoding

[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
...,

Learns representations (patterns, features, weights)

Representation outputs

[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
...,

Outputs

1. Cut veg...

Part 1: Turn data into numbers

Part 2: Build model to learn patterns in numbers

ingenum

Supervised learning

[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
…,

1. Initialise with random weights

2. Show examples

[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
…,

1. Cut veg…

3. Update representation outputs

4. Repeat with more samples

ingenum

PyTorch Training Loop

Pass the data through the model for a number of epochs (e.g. 200 for 200 passes of the data)
Create empty lists for storing useful values (helpful for tracking model progress)

```python
# Setting the Learning Stage: Planting the Seed
torch.manual_seed(42)  # Ensures consistent results, like planting a seed for a predictable harvest

# Training Journey: Embarking on the Epochs
epochs = 200  # Number of times the model will explore the training data

# Progress Tracker: Charting the Course
train_loss_values = []  # Recording the model's progress during training
test_loss_values = []  # Assessing the model's performance on unseen data
epoch_count = []  # Marking milestones along the way

# The Grand Loop: Guiding the Model's Learning
for epoch in range(epochs):
    # Training Phase: Sharpening the Skills
    model_0.train()  # Setting the model to training mode, like entering a practice arena

    # 1. Forward Pass: Making Predictions
    y_pred = model_0(X_train)  # The model takes its first steps, attempting to predict outcomes

    # 2. Loss Calculation: Evaluating Performance
    loss = loss_fn(y_pred, y_train)  # The teacher (loss function) assesses the model's predictions

    # 3. Optimizer Reset: Clearing the Path
    optimizer.zero_grad()  # The coach (optimizer) prepares the model for the next step

    # 4. Backpropagation: Learning from Mistakes
    loss.backward()  # The model reflects on its errors, seeking areas for improvement

    # 5. Parameter Update: Refining Skills
    optimizer.step()  # The coach guides the model, adjusting its parameters for better predictions
```

1. Pass the data through the model, this will perform the forward( ) method located within the model object
2. Calculate the loss value (how wrong the model's predictions are)
3. Zero the optimizer gradients (they accumulate every epoch, zero them to start fresh each forward pass)
4. Perform backpropagation on the loss function (compute the gradient of every parameter with requires_grad=True)
5. Step the optimizer to update the model's parameters with respect to the gradients calculated by loss.backward()

ingenum

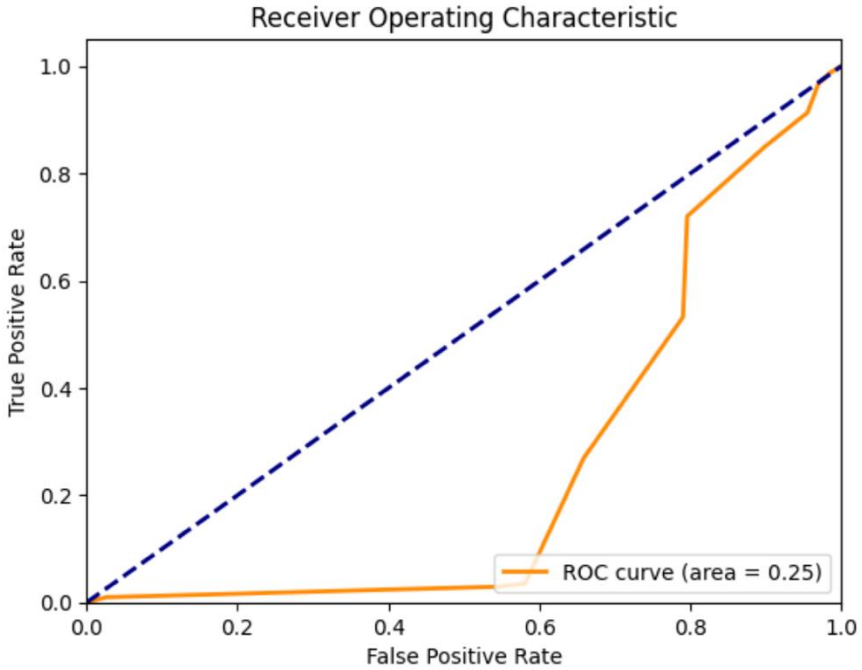
PyTorch Testing Loop

Turn on torch.inference_mode() context manager to disable gradient tracking for inference

```
# Testing Phase: Demonstrating Knowledge
model_0.eval()  # Setting the model to evaluation mode, like entering the performance stage

with torch.inference_mode():  # Ensuring the model's performance is unbiased
    # 1. Forward Pass: Applying Learned Knowledge
    test_pred = model_0(X_test)  # The model tackles unseen challenges, putting its knowledge to the test

    # 2. Loss Calculation: Measuring Performance
    test_loss = loss_fn(test_pred, y_test.type(torch.float))  # The teacher evaluates the model's performance on new data

    # Progress Report: Sharing the Achievements
    if epoch % 10 == 0:  # Reporting at regular intervals, like sharing progress at milestones
        epoch_count.append(epoch)
        train_loss_values.append(loss.detach().numpy())
        test_loss_values.append(test_loss.detach().numpy())
        print(f"Epoch: {epoch} | Training Loss: {loss:.4f} | Testing Loss: {test_loss:.4f}")
```

1. Pass the test data through the model (this will call the model's implemented forward() method)
2. Calculate the test loss value (how wrong the model's predictions are on the test dataset, lower is better)
3. Display information outputs for how the model is doing during training/testing every ~10 epochs (note: what gets printed out here can be adjusted for specific problems)

Don’t expect great things straight away



Pretrained outcomes

- Random weight
- One variable
- One neurone.

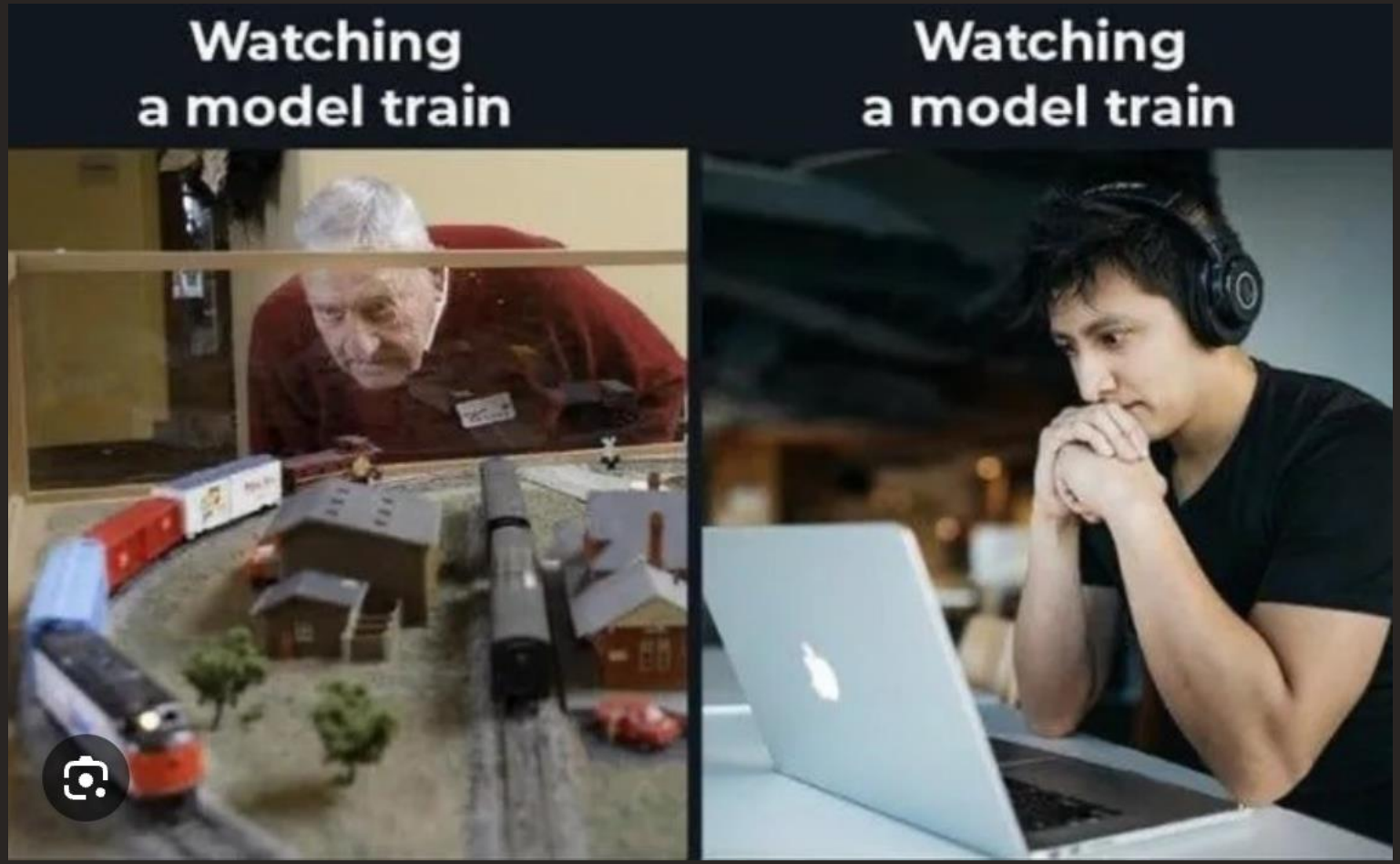
ingenum

Resources

Course tutors

Google’s in-built native LLM

https://pytorch.org/



```
def lets_code(language):
    print(f"Let's start coding in {language}!")
```

https://github.com/ingenum-ai/ISVEE_deepLearning_2024/

Open Notebook 4…

ingenum