Applied ML for veterinary epidemiologists

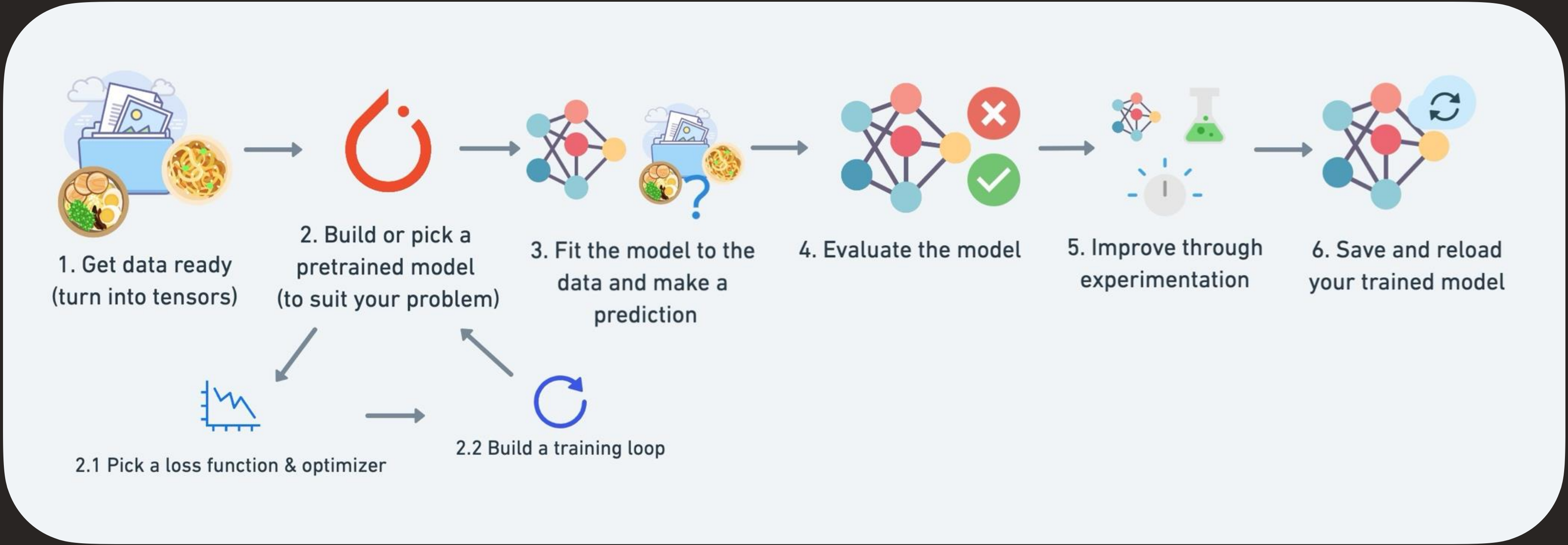
ISVEE pre-conference workshop - Day 4 – Session 3

Dr Tom Brownlie

PyTorch

ingenum

PyTorch workflow



Source https://github.com/mrdbourke/pytorch-deep-learning

ingenum

Machine Learning: A game of two halves

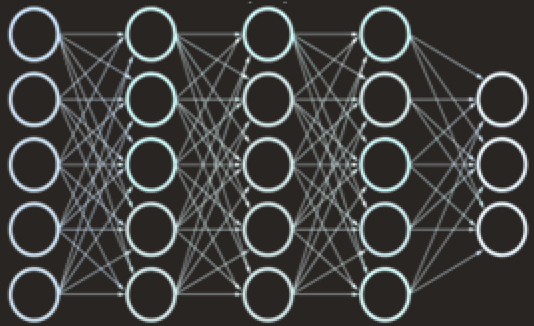
Inputs

Numerical encoding

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

Part 1: Turn data into numbers

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],
...,

Part 2: Build model to learn patterns in numbers

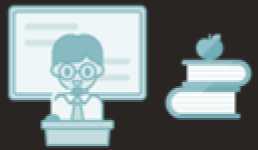
Outputs

1. Cut veg...

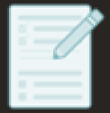
ingenum

1. Set up data

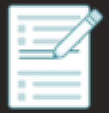
Splitting data is almost always the first step in a



Course materials
training set



Practice exam
(validation set)



Final exam
Test set

```python
# Split raw dataset

dataset_split = train_test_split(*[feature_set_encoded, target],
                                 test_size=0.2,
                                 stratify=target,
                                 random_state=42)
```

2. Convert data to tensors

Using the tensors command from nn package

```
# Convert to tensors

import torch
from torch import nn                          # nn contains all of PyTorch's building blocks for neural networks

tensors = {k:torch.tensor(v, dtype=torch.long) for k,v in data.items()}

X_train = tensors['X_train'].type(torch.float32)
X_test = tensors['X_test'].type(torch.float32)
y_train = tensors['y_train'].type(torch.float32)
y_test = tensors['y_test'].type(torch.float32)
```

PyTorch essential neural network building modules

PyTorch module	What does it do?
torch.nn	Contains all of the building blocks for computational graphs (essentially a series of computations executed in a particular way).
torch.nn.Module	The base class for all neural network modules, all the building blocks for neural networks are subclasses. If you're building a neural network in PyTorch, your models should subclass nn.Module. Requires a forward()method be implemented.
torch.optim	Contains various optimization algorithms (these tell the model parameters stored in nn.Parameter how to best change to improve gradient descent and in turn reduce the loss).
torch.utils.data.Dataset	Represents a map between key (label) and sample (features) pairs of your data. Such as images and their associated labels.
torch.utils.data.DataLoader	Creates a Python iterable over a torch Dataset (allows you to iterate over your data).

ingenum

3. Select model

Linear Layers

`nn.Identity`	A placeholder identity operator that is argument-insensitive.
`nn.Linear`	Applies an affine linear transformation to the incoming data: $y = xA^T + b$.
`nn.Bilinear`	Applies a bilinear transformation to the incoming data: $y = x_1^T A x_2 + b$.
`nn.LazyLinear`	A `torch.nn.Linear` module where *in_features* is inferred.

https://pytorch.org/docs/stable/torch.html

3. Select model

```
[ ]  # Create a logisitic regression model class using the training data

     class LogisticRegressionModel(nn.Module):
         def __init__(self, input_size):
             super().__init__()

             self.linear = nn.Linear(input_size, 1)

     # Forward defines the computation in the model
         def forward(self, x):
             return torch.sigmoid(self.linear(x))
```

1. Subclass nn.Module (this contains all the building blocks for neural networks)

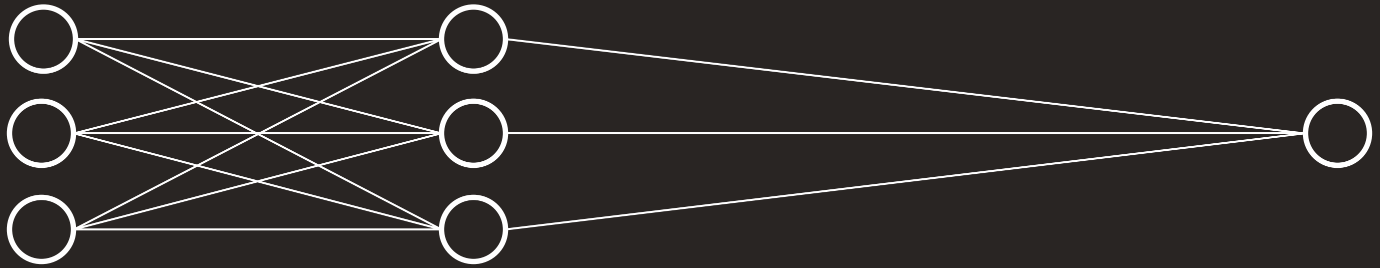
2. Initialise model parameters to be used in various computations (these could be different layers from torch.nn, single parameters, hard-coded values or functions)

3. Any subclass of nn.Module needs to override forward(). This defines the forward computation of the model

4. We wish to return a sigmoid function (think of it as a logit link) for our data

ingenum

A logistic regression model, while often considered a 'simple' neural network, has structure that you will recognise from graphical representations of neural networks



Input Layer (11)

Represents features (independent variables) fed into the model.

Not explicitly defined as a layer in the code

Hidden Layer

(Single Layer; 5 – 10)

In basic logistic regression, effectively a single hidden layer performs a linear transformation on the input features using weights and biases.

Not explicitly defined in the code but represented by weight matrix and bias vector within the model.

Activation Function

Sigmoid (or logistic) activation function applied to output of the hidden layer.

Introduces non-linearity, allowing the model to learn relationships between features and target variable.

Imposed within the model's forward method.

Output Laye r (1)

Model's prediction, representing the probability of the target variable belonging to a specific class (e.g., 0 or 1).

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()

PyTorch Testing Loop

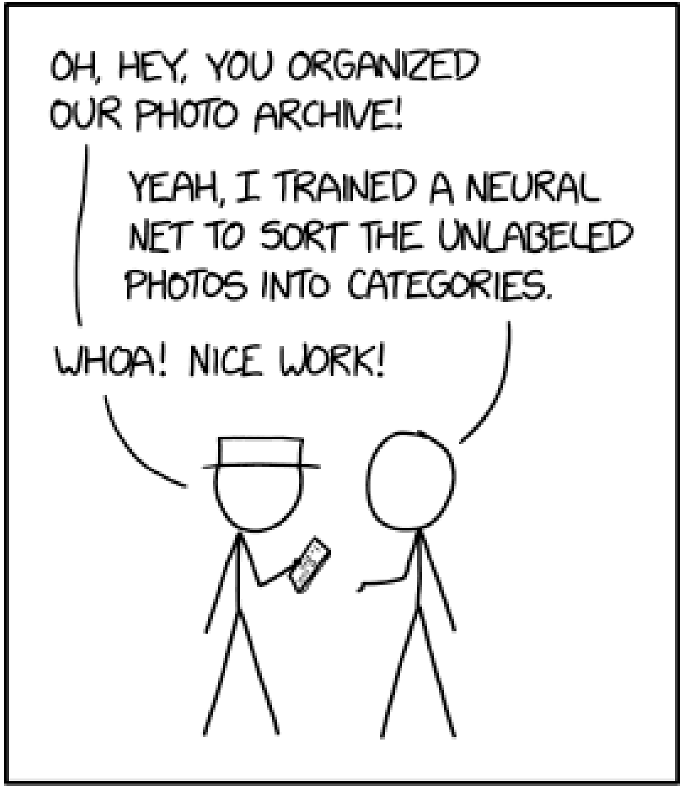
Turn on torch.inference_mode() context manager to disable gradient tracking for inference

```python
# 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)

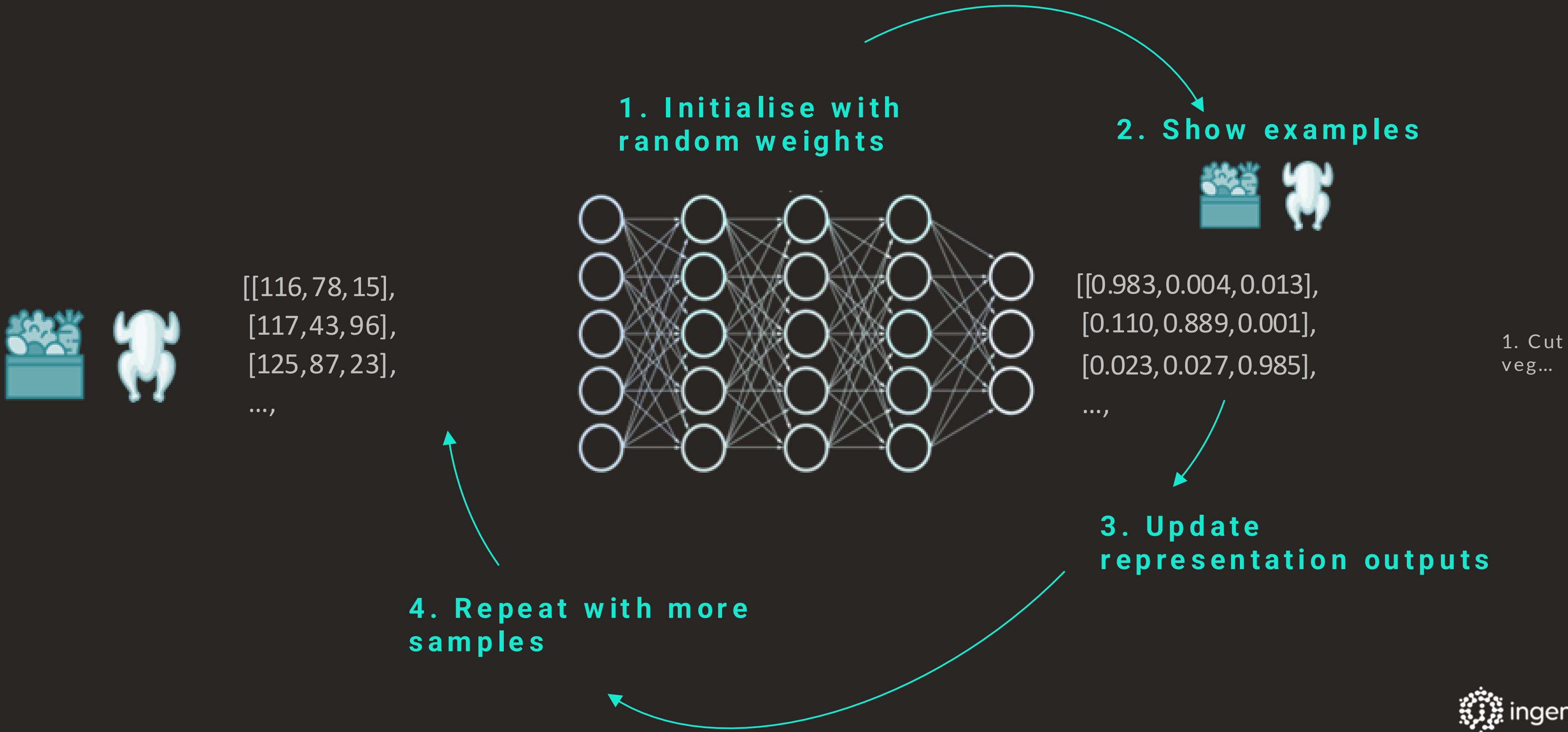
2. Calculate the test loss value (how wrong the model's predictions are on the test dataset, lower is better)

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)



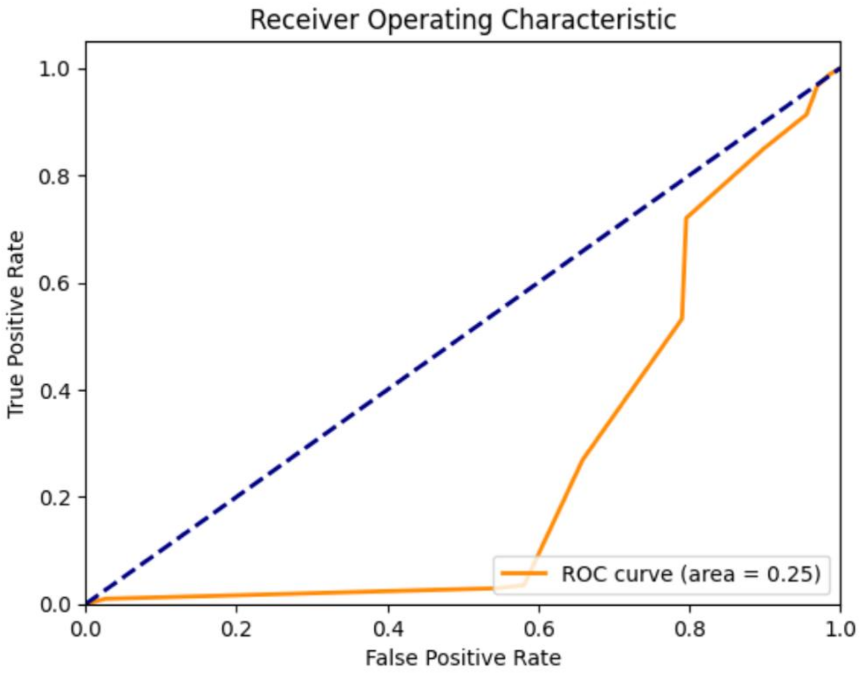
ENGINEERING TIP:
WHEN YOU DO A TASK BY HAND, YOU CAN TECHNICALLY SAY YOU TRAINED A NEURAL NET TO DO IT.

Supervised learning



ingenum

Don’t expect great things straight away



Pretrained outcomes

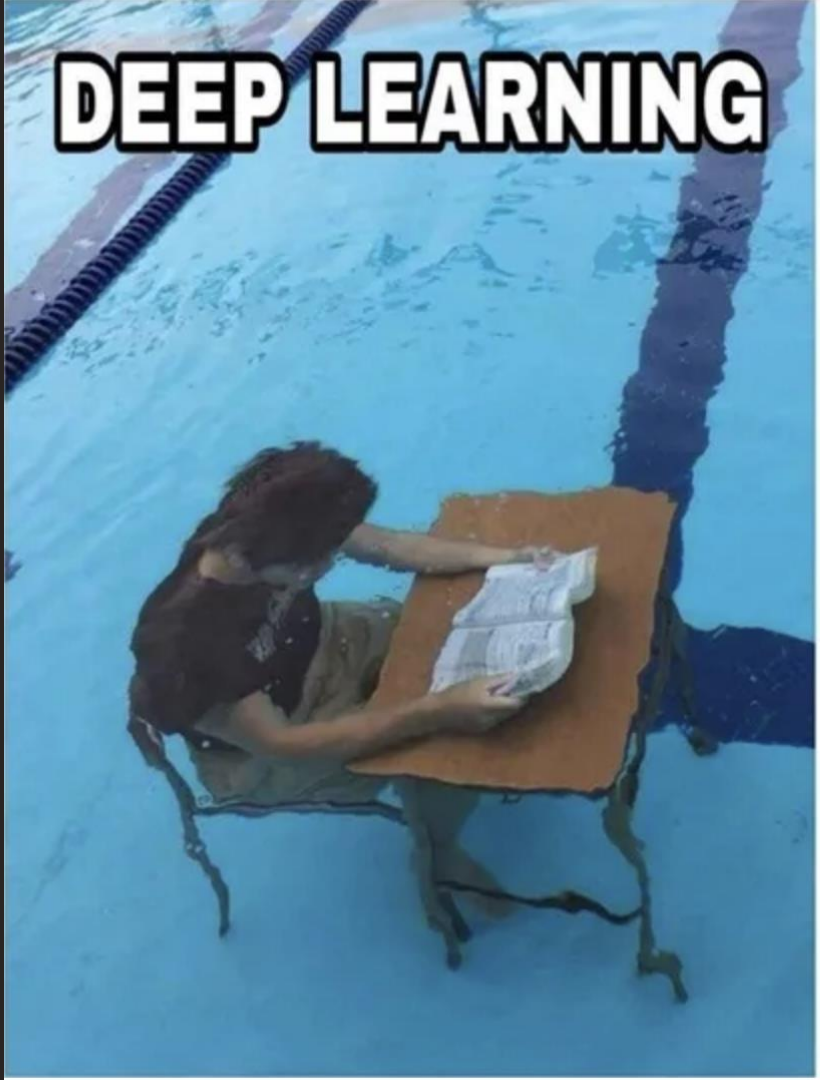
- Random weight
- One variable
- One neurone.

Resources

Course tutors

Google’s in-built native LLM

https://pytorch.org/


DEEP LEARNING

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

https://github.com/ingenum-ai/isvee_deepLearning_2024/

Open Notebook 3…