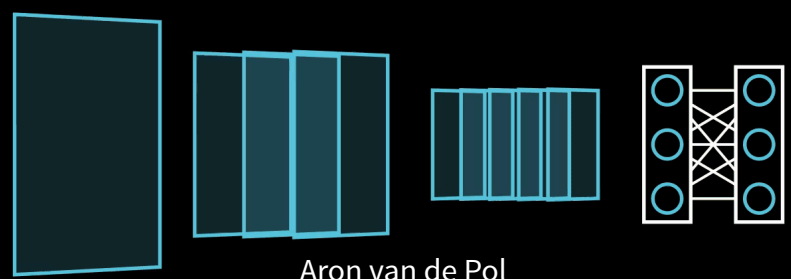
🔥 Deep Learning Workshop



Aron van de Pol

Leiden University Centre For Digital Humanities

2024-02-20

Linear Regression

Today

- Working mainly on simple data
- Learn the basics of model training
- Train your first model!

Linear Regression

📈 Linear Regression

While many will likely remember what linear regression is from high school.

See this scenario:

- Peter is a kid who gets $2 of pocket money a day.
- Peter has been saving for a while and currently has $50 saved.
- How much will Peter have in a week?

Linear Regression

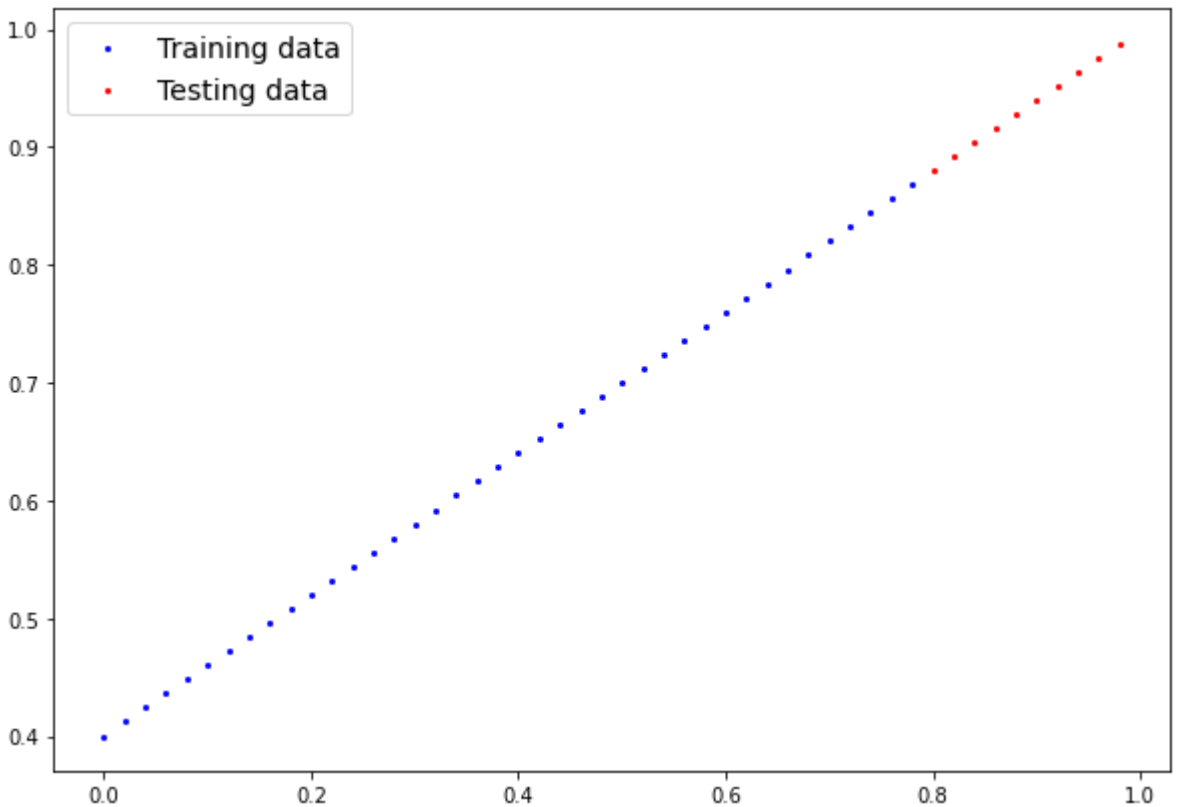
- The answer, of course, should be $64.
- How did you get your answer?

$$y = a * x + b$$

$$y = m * x + b$$

Linear Regression

How then would our data look?



🔪 Data Splitting

- Neural networks learn by example
- Can not feed all data at once.
- Split our data!

🔪 Why split our data?

- The main reason is to fairly evaluate performance
- Counter key conceps:
  - **Overfitting**
  - **Underfitting**

🔪 Dividing the data: Scenario 1

Imagine you want to classify painting in their given art style.

- Data is typically divided into training and test sets.
- **Training Set**: Used to train the model, usually comprising 70-80% of the dataset.
- **Test Set**: Used to test the model, usually comprising 20-30% of the dataset.

🔪 Dividing the data: Scenario 2

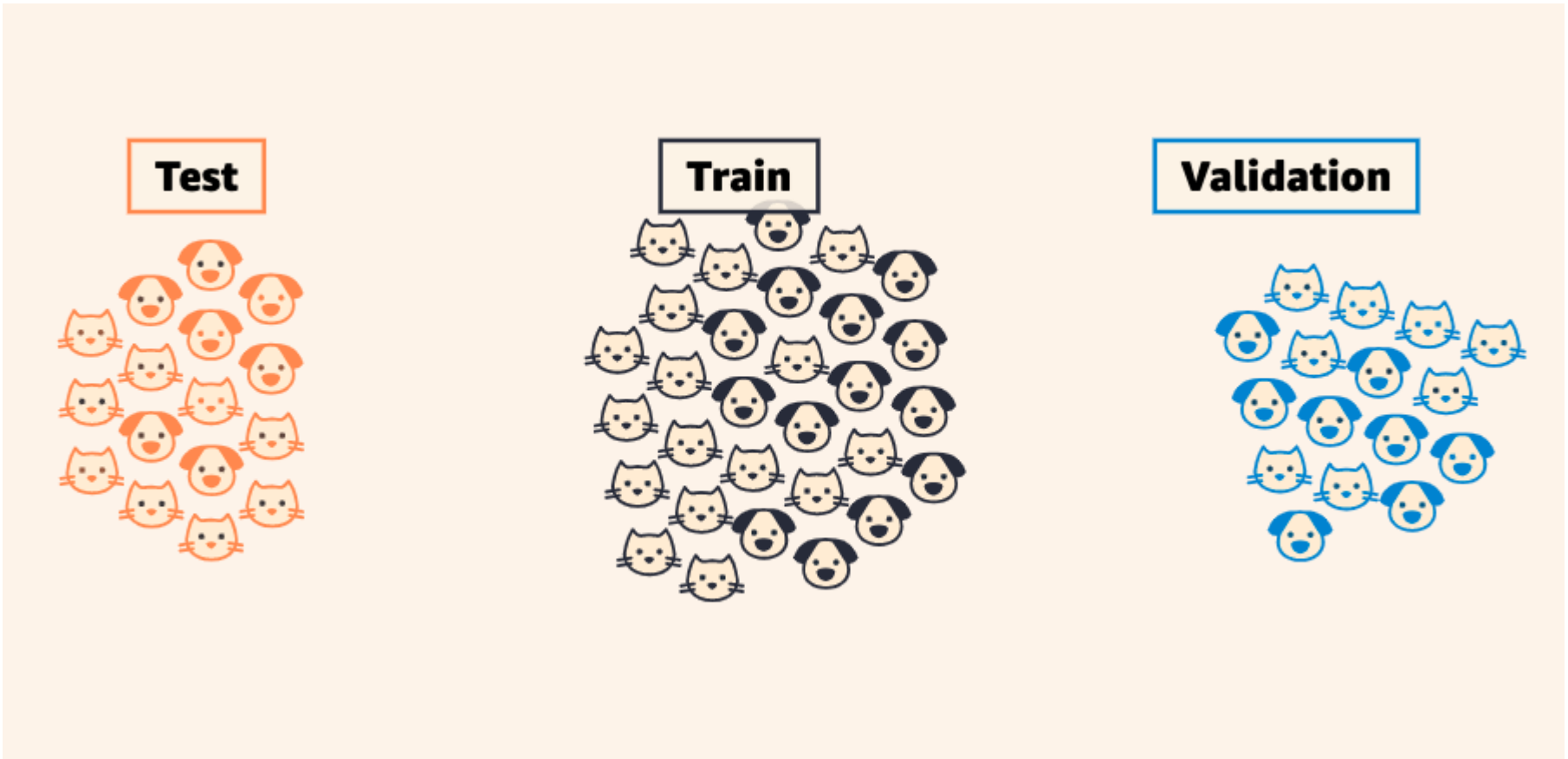
- You are working on classifying the printshops of pages texts from Colonial Korea. You have many labelled examples of such pages.
- Yet you want to also see how the model would perform on ‘unseen’ data.

🔪 Dividing the data: Scenario 2

- **Training Set**: Used to train the model, usually comprising ~70% of the dataset.
- **Validation Set**: Used to test the model, usually comprising 15% of the dataset.
- **Test Set**: Used to test the model, usually comprising 15% of the dataset.

🔪 Data Splitting

In essence:



🧠 A Linear Neuron

- Imagine as a tiny decision making unit in vast network.
- Each neuron is connected to other neurons
- together they make sense of the data.

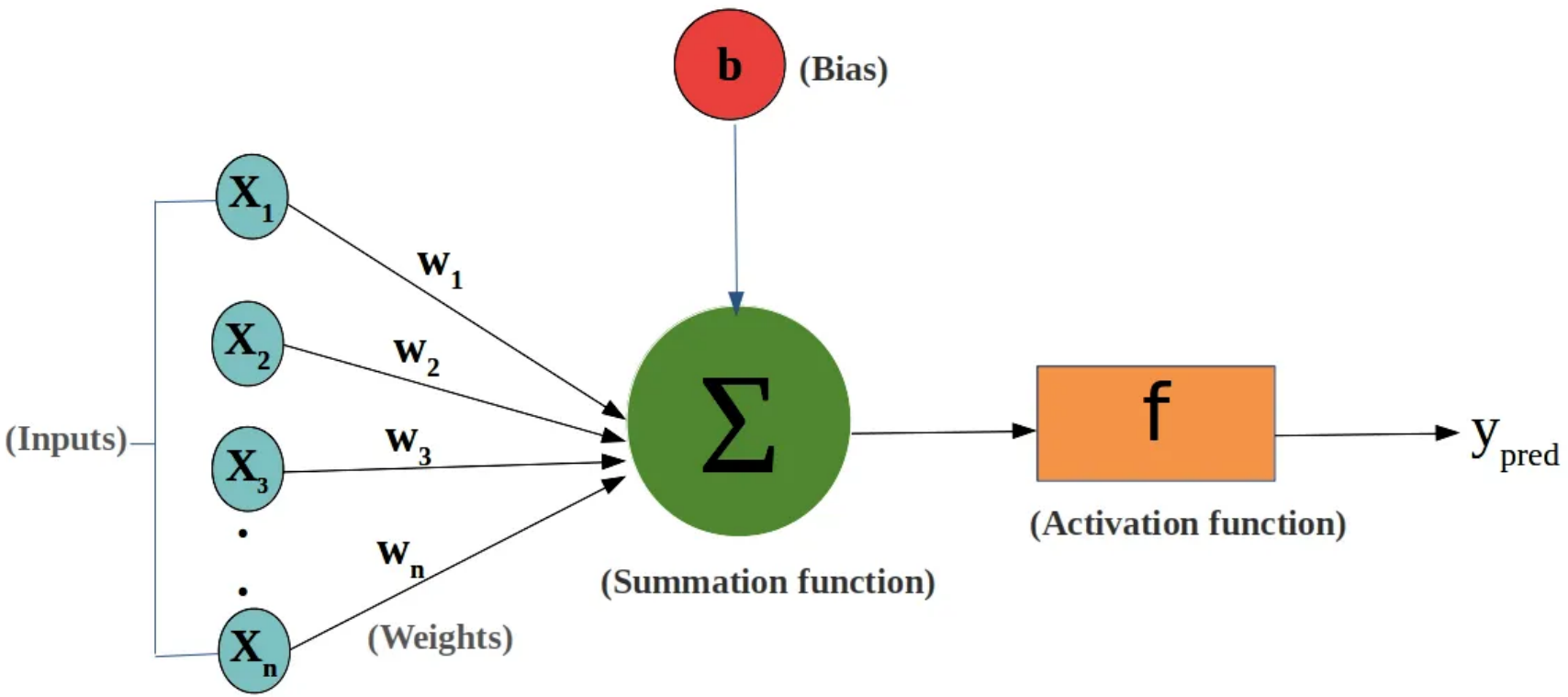
🧠 A Linear Neuron

- Each connection has a **weight**.
- The weight influences of each input on the neuron’s decision
- The neuron also has a **bias**.
- The bias is added to the weighted inputs it receives.

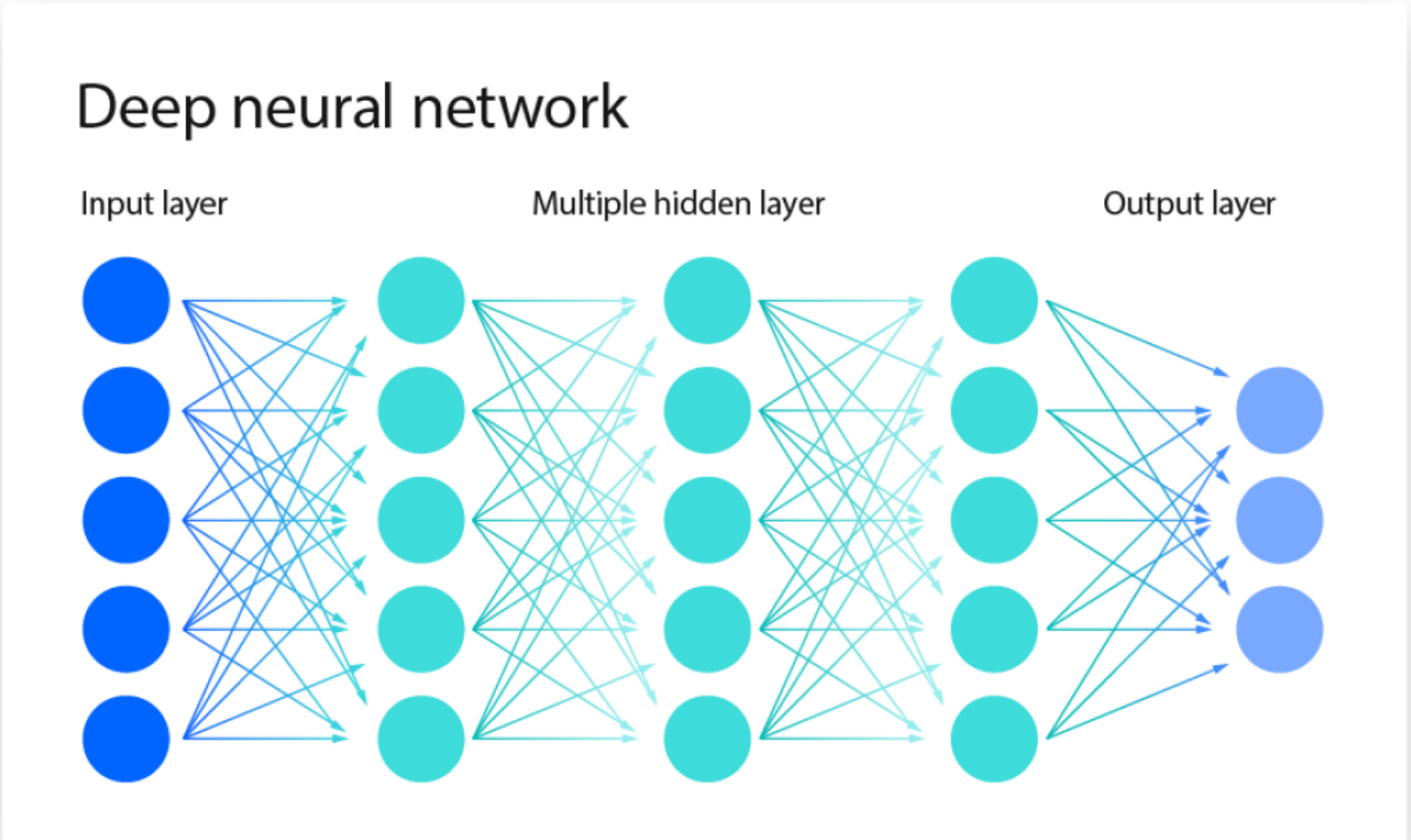
🧠 A Linear Neuron

$$\text{Output} = (\text{Input}_1 \times \text{Weight}_1) + (\text{Input}_2 \times \text{Weight}_2) + \ldots + \text{Bia}$$

A Linear Neuron

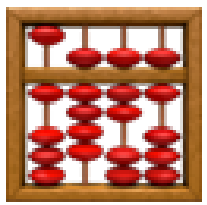


A Linear Neuron



🔥 A Network in PyTorch

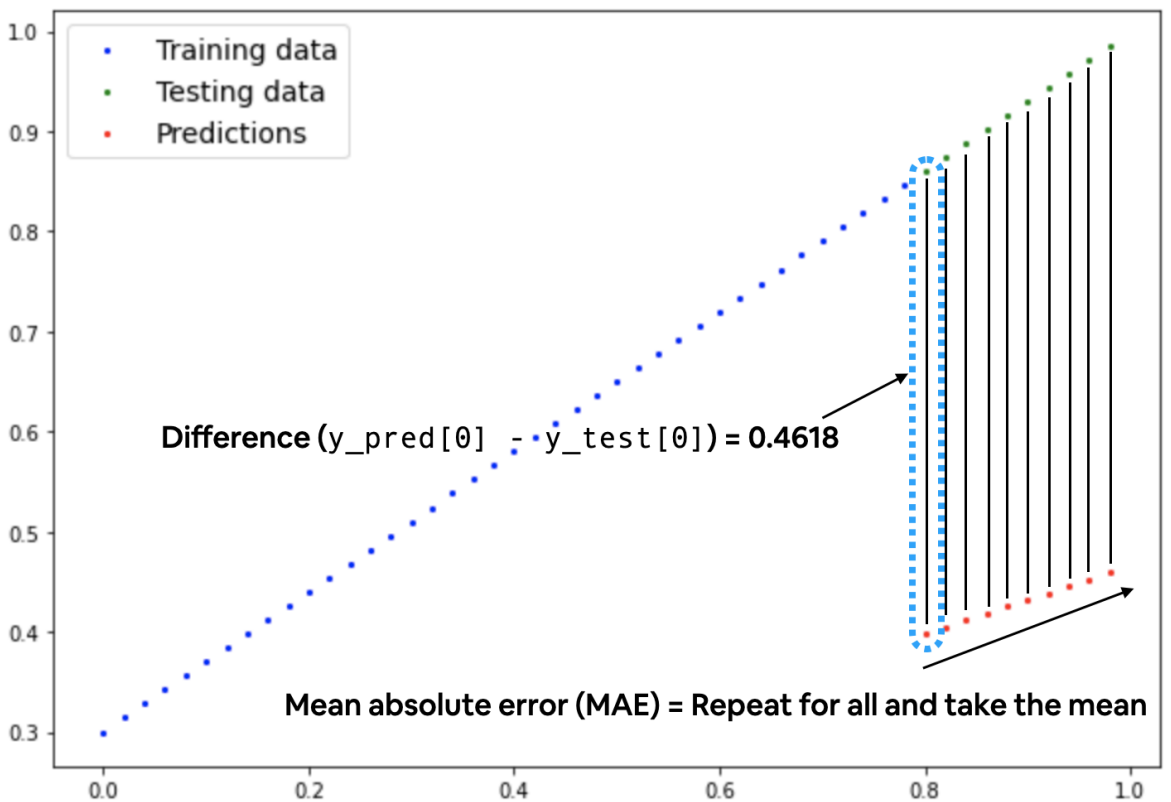
```
class LinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linearlayer = nn.Sequential(
          nn.Linear(in_features=1,
                    out_features=1)
        )

    def forward(self, x):
        x = self.linearlayer(x)
        return x
```



Functions for Training

The Loss (Cost) Function

- Measures how ‘wrong’ the network is.



The Loss (Cost) Function

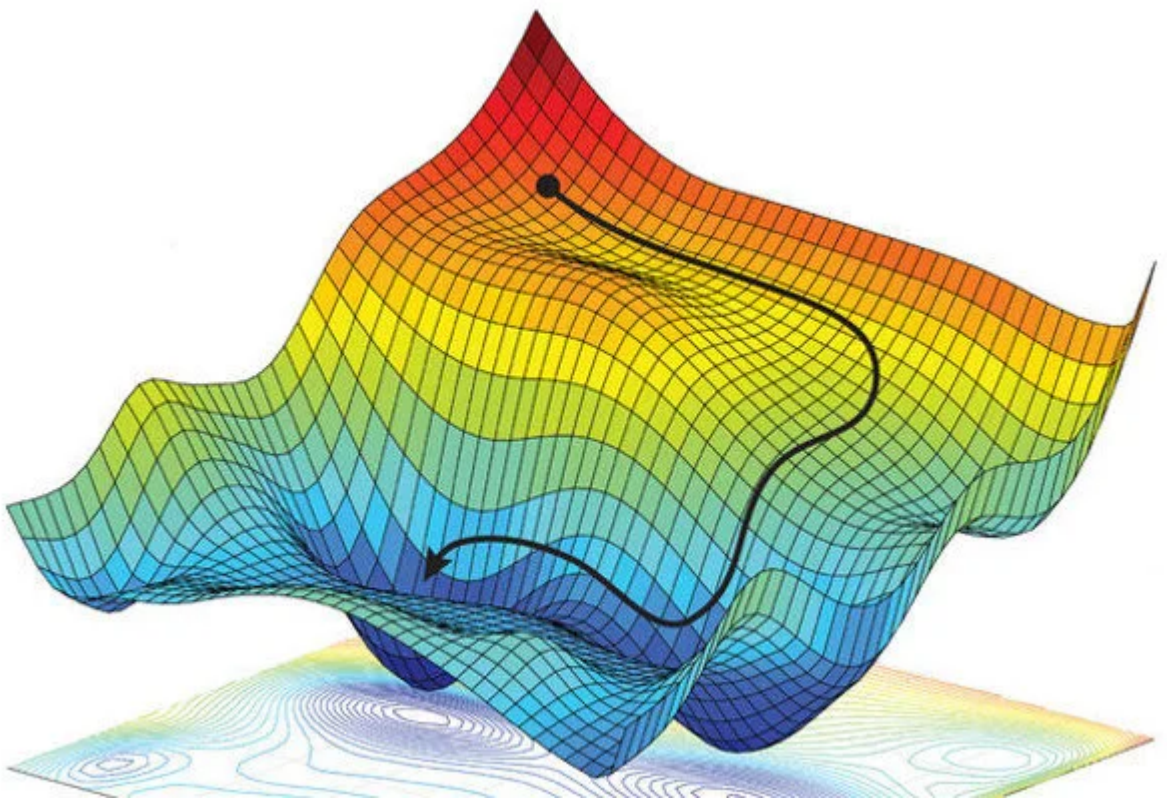
```
# Create loss function
loss_fn = nn.L1Loss()
```

🔧 The Optimizer

- Optimizers adjust the model's parameters to reduce losses.
- SGD (Stochastic Gradient Descent) is a common choice.

🔧 How does it optimize?

- Backpropagation.



Source: Amini et al.[1]

Training

Training Loop

1. **Forward pass - Train data through model**
2. **Calculate Loss**
3. **Optimizer zero grad (reset optimizer)**
4. **Loss backward (Backpropagation)**
5. **Optimizer step (adjust parameters)**
6. **Repeat…**

💪 Test Loop

1. **Forward pass - Test data through model**
2. **Calculate Loss & Accuracy metrics**

🔥 Training in PyTorch

```
# Set the number of epochs
epochs = 1000

# send model to the correct device
model.to(device)

# Put data on the available device
X_train = X_train.to(device)
X_test = X_test.to(device)
y_train = y_train.to(device)
y_test = y_test.to(device)
```

🔥 Training in PyTorch

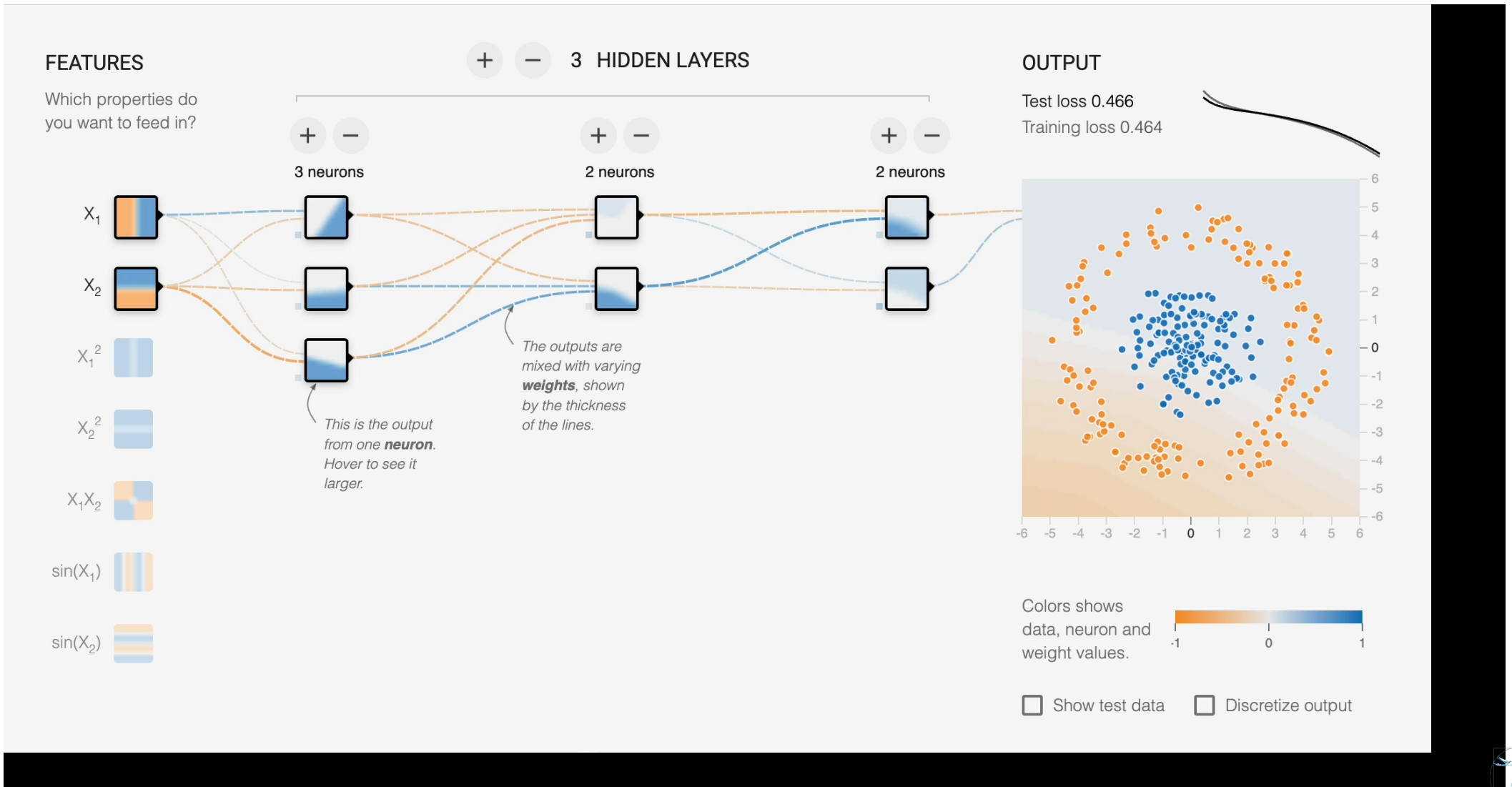
```
for epoch in range(epochs):
    ### Training
    model.train() # train mode is on by default after construction

    # 1. Forward pass
    y_pred = model(X_train)

    # 2. Calculate loss
    loss = loss_fn(y_pred, y_train)

    # 3. Zero grad optimizer
    optimizer.zero_grad()

    # 4. Loss backward
    loss.backward()

    # 5. Step the optimizer
    optimizer.step()
```

🔥 Testing in PyTorch

```
### Testing
model.eval()
# put the model in evaluation mode for testing (inference)
# 1. Forward pass
with torch.inference_mode():
    test_pred = model(X_test)

    # 2. Calculate the loss
    test_loss = loss_fn(test_pred, y_test)

# use % operator to see if fully divisable.
if epoch % 100 == 0:
    # reports per 100 epochs
    print(f"Epoch: {epoch:.2f} | Train loss: {loss:.2f} | Test
```

Next Session

Fitting to Non Linear data!



Works Cited

Amini, Alexander, Ava Soleimany, Sertac Karaman, and Daniela Rus. “Spatial Uncertainty Sampling for End-to-End Control,” 2019. https://arxiv.org/abs/1805.04829.