SESSION // 02
ARTIFICIAL NEURAL NETWORKS

FACULTY OF
SCIENCE AND ENGINEERING

+++

Diego Corona Lopez – AI Technical Specialist

AGENDA

Introduction to Artificial Neural Networks

- Understanding neural networks and their components
- Exploring neuron structure and activation functions

Implementing a Neuron in PyTorch

- Hands-on coding: Building a perceptron model
- Exercise: Creating our first neural network component

Case Study: Space Shuttle Challenger

- Historical context of the disaster
- Introduction to the O-ring dataset

Data Preparation and Model Design

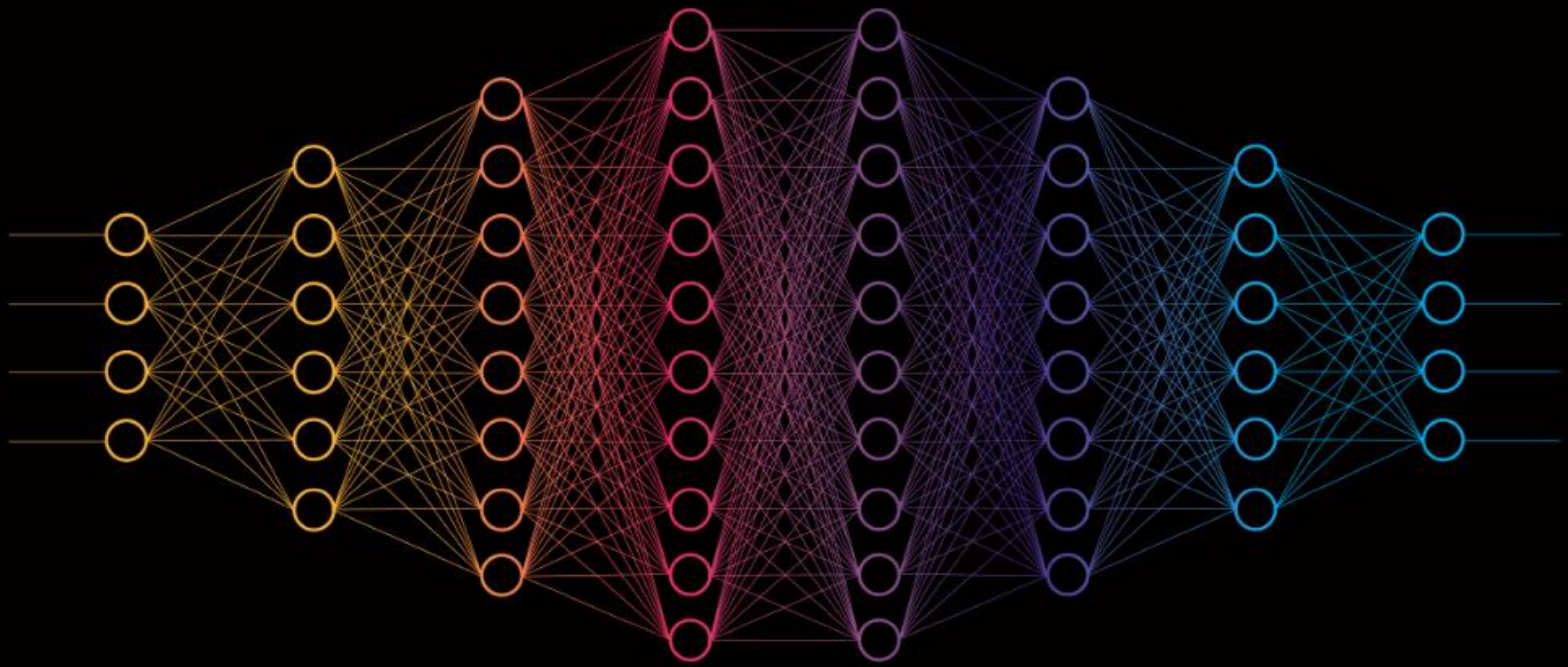
- Exercise: Preprocessing temperature and O-ring data
- Converting, normalizing and structuring data for neural networks

Training and Evaluation

- Exercise: Implementing loss functions and model training
- Visualizing model performance and understanding convergence

Risk Analysis and Ethical Implications

- Exercise: Predicting O-ring failure at Challenger launch temperature
- Discussion: Data science ethics and responsibility



WHAT ARE ARTIFICIAL NEURAL NETWORKS?

ANNs are computational models inspired by the human brain's neural structure. They consist of interconnected neurons organized in layers that transform input data into meaningful outputs. Each neuron performs weighted calculations on its inputs and applies an activation function to introduce non-linearity.

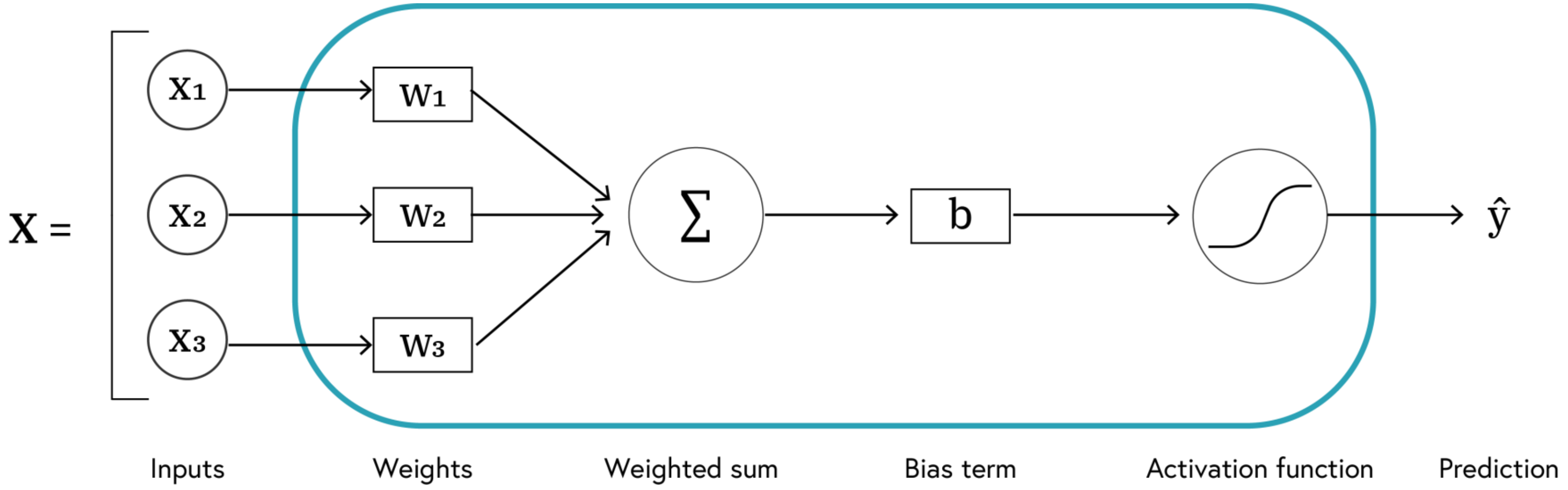
ANATOMY OF A NEURON

Basic computational units of neural networks

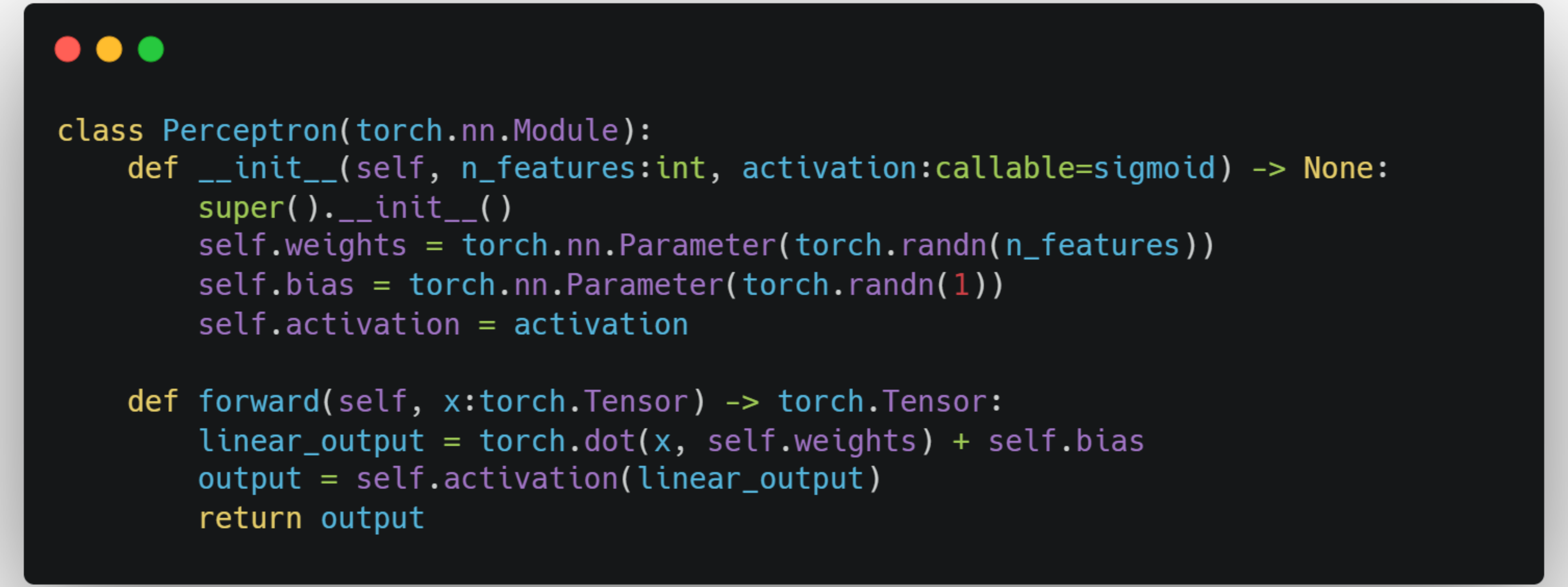
Each neuron:

- Receives inputs
- Applies weights and bias
- Processes through activation function
- Produces output

$$\hat{y} = f(W \cdot X + b)$$

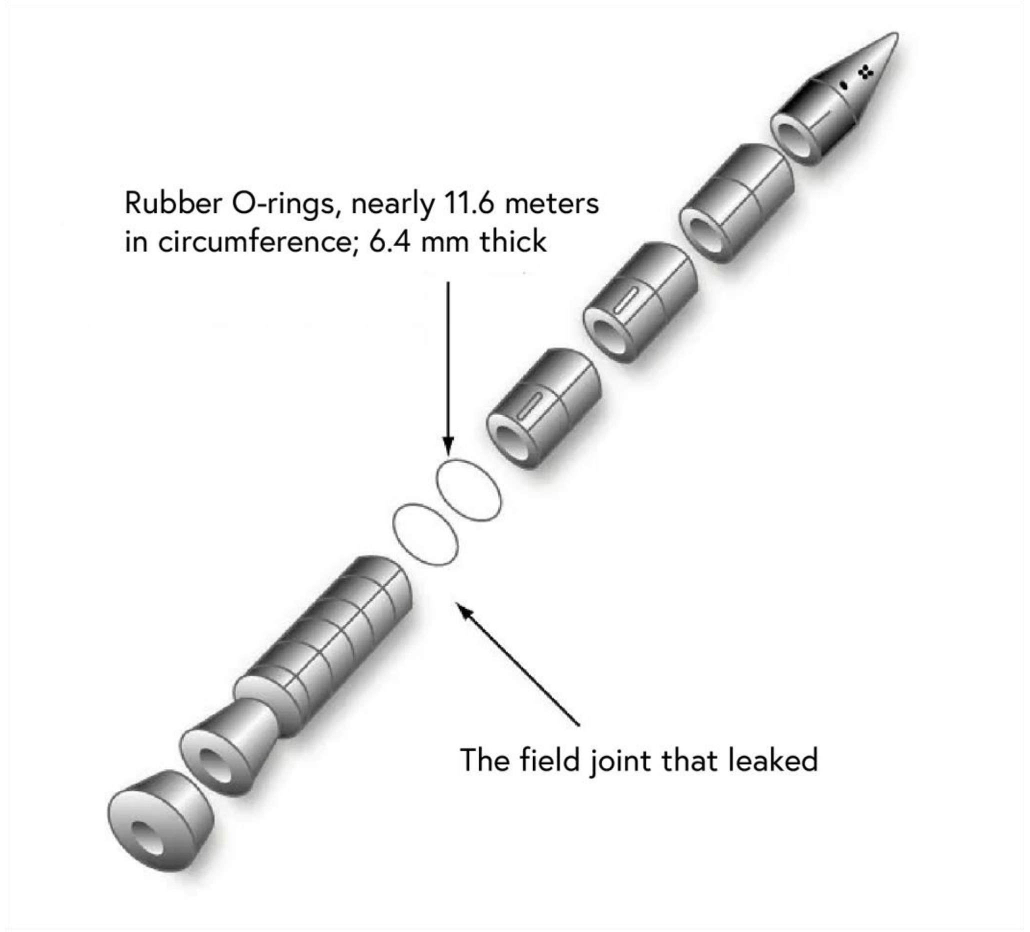


IMPLEMENTING A NEURON



```python
class Perceptron(torch.nn.Module):
    def __init__(self, n_features:int, activation:callable=sigmoid) -> None:
        super().__init__()
        self.weights = torch.nn.Parameter(torch.randn(n_features))
        self.bias = torch.nn.Parameter(torch.randn(1))
        self.activation = activation

    def forward(self, x:torch.Tensor) -> torch.Tensor:
        linear_output = torch.dot(x, self.weights) + self.bias
        output = self.activation(linear_output)
        return output
```

THE SPACE SHUTTLE CHALLENGER DISASTER


Rubber O-rings, nearly 11.6 meters in circumference; 6.4 mm thick
The field joint that leaked

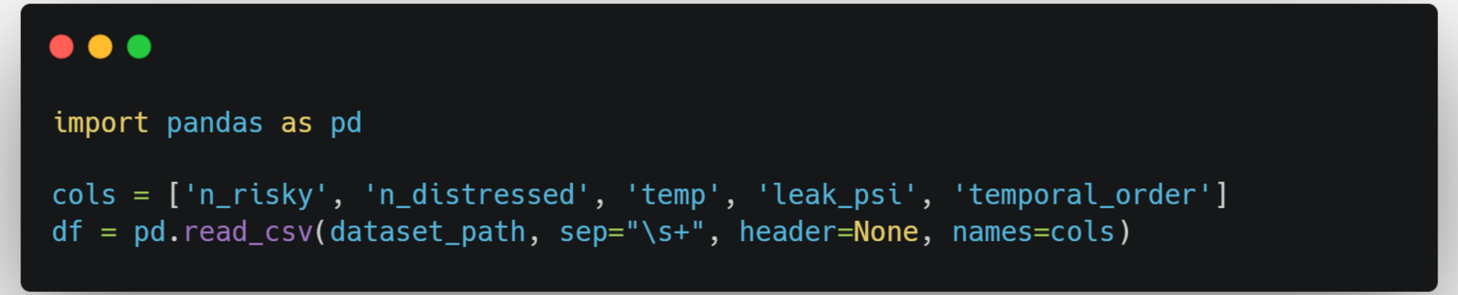
- **January 28, 1986:** Space Shuttle Challenger exploded 73 seconds after launch
- **Cause:** O-ring failure due to low temperatures
- **Question:** Could data analysis have prevented this tragedy?

O-RING FAILURE DATASET

Historical data from previous shuttle launches

Key variables:

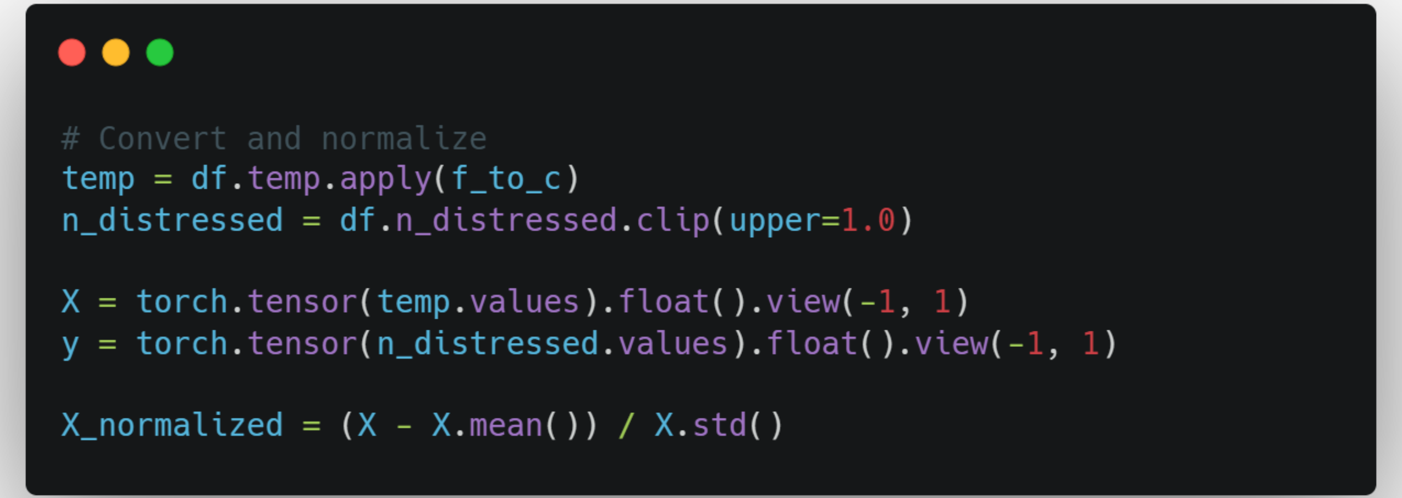
- Temperature at launch (°F)
- Number of O-rings at risk
- Number of O-rings experiencing thermal distress



```python
import pandas as pd

cols = ['n_risky', 'n_distressed', 'temp', 'leak_psi', 'temporal_order']
df = pd.read_csv(dataset_path, sep="\s+", header=None, names=cols)
```

DATA PREPARATION

- Convert temperature from Fahrenheit to Celsius
- Convert distress to binary outcome (0 or 1)
- Normalize temperature data
- Split into training and test sets

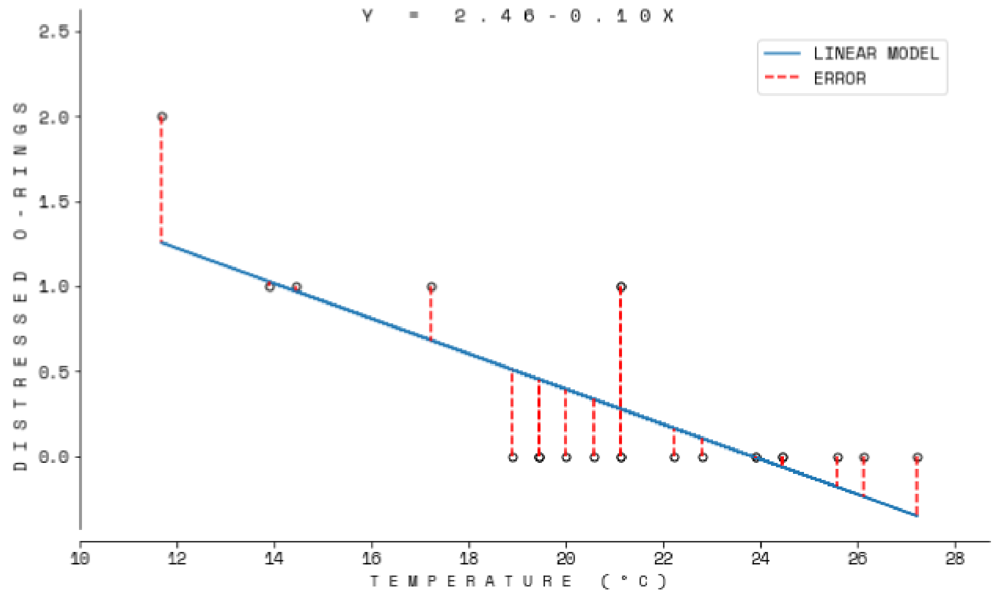


```python
# Convert and normalize
temp = df.temp.apply(f_to_c)
n_distressed = df.n_distressed.clip(upper=1.0)

X = torch.tensor(temp.values).float().view(-1, 1)
y = torch.tensor(n_distressed.values).float().view(-1, 1)

X_normalized = (X - X.mean()) / X.std()
```

BASICS OF TRAINING

1. **Forward Pass:** Feed data through network to get predictions
2. **Loss Calculation**: Measure error between predictions and targets
3. **Backward Pass:** Compute gradients to adjust parameters
4. **Parameter Update:** Improve model using gradient descent

$$\hat{x} = \underset{x}{\arg\min} \, ||Ax - b||_2$$

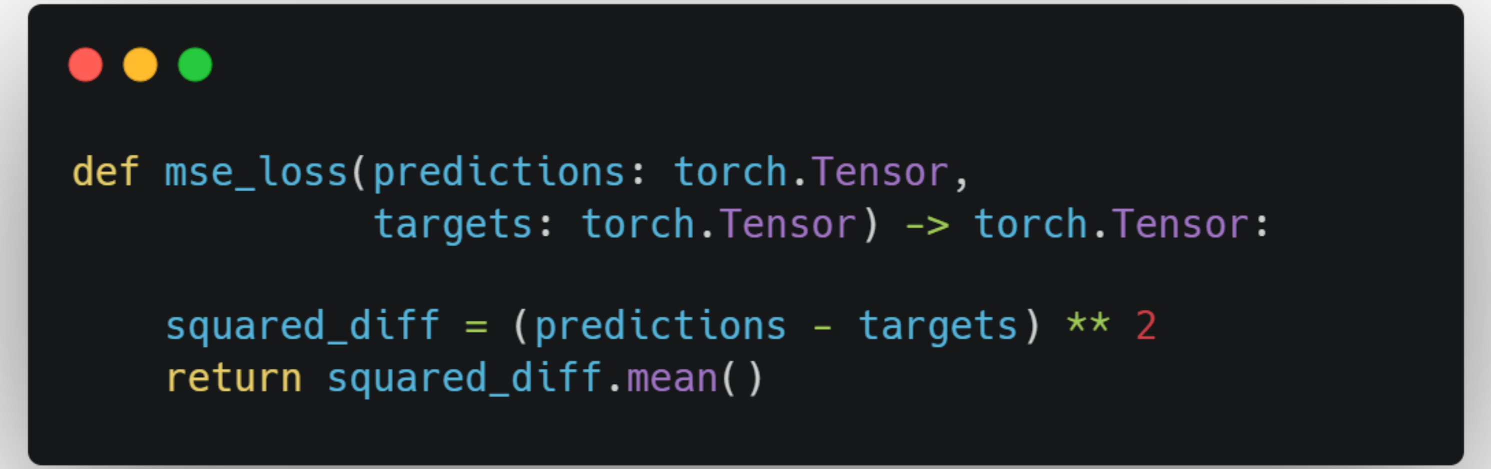


LOSS FUNCTION

- Loss functions quantify prediction errors
- We use Mean Squared Error (MSE):

$$MSE = 1/n\,\Sigma(y - \hat{y})^2$$

- Provides direction for optimization



```
def mse_loss(predictions: torch.Tensor,
             targets: torch.Tensor) -> torch.Tensor:

    squared_diff = (predictions - targets) ** 2
    return squared_diff.mean()
```

BACKWARD PROPAGATION

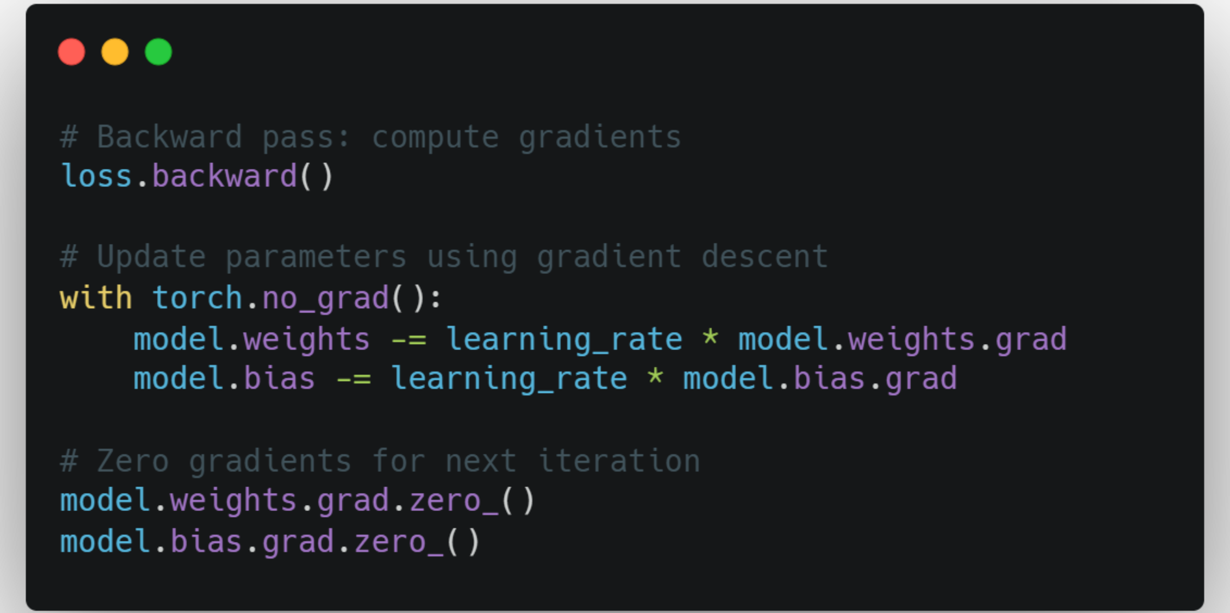
Neural networks learn by adjusting their parameters based on prediction errors. The backward pass, or backpropagation, is the algorithm that enables this learning process:

1. After computing the loss (error), gradients are calculated showing how each parameter affects the error
2. These gradients flow backward through the network using the chain rule of calculus
3. Parameters are updated using gradient descent: weights and biases are adjusted in the direction that reduces error
4. The update formula is:

$$w_{\text{new}} = w_{\text{old}} - \alpha \frac{\partial L}{\partial w}$$

$$b_{\text{new}} = b_{\text{old}} - \alpha \frac{\partial L}{\partial b}$$

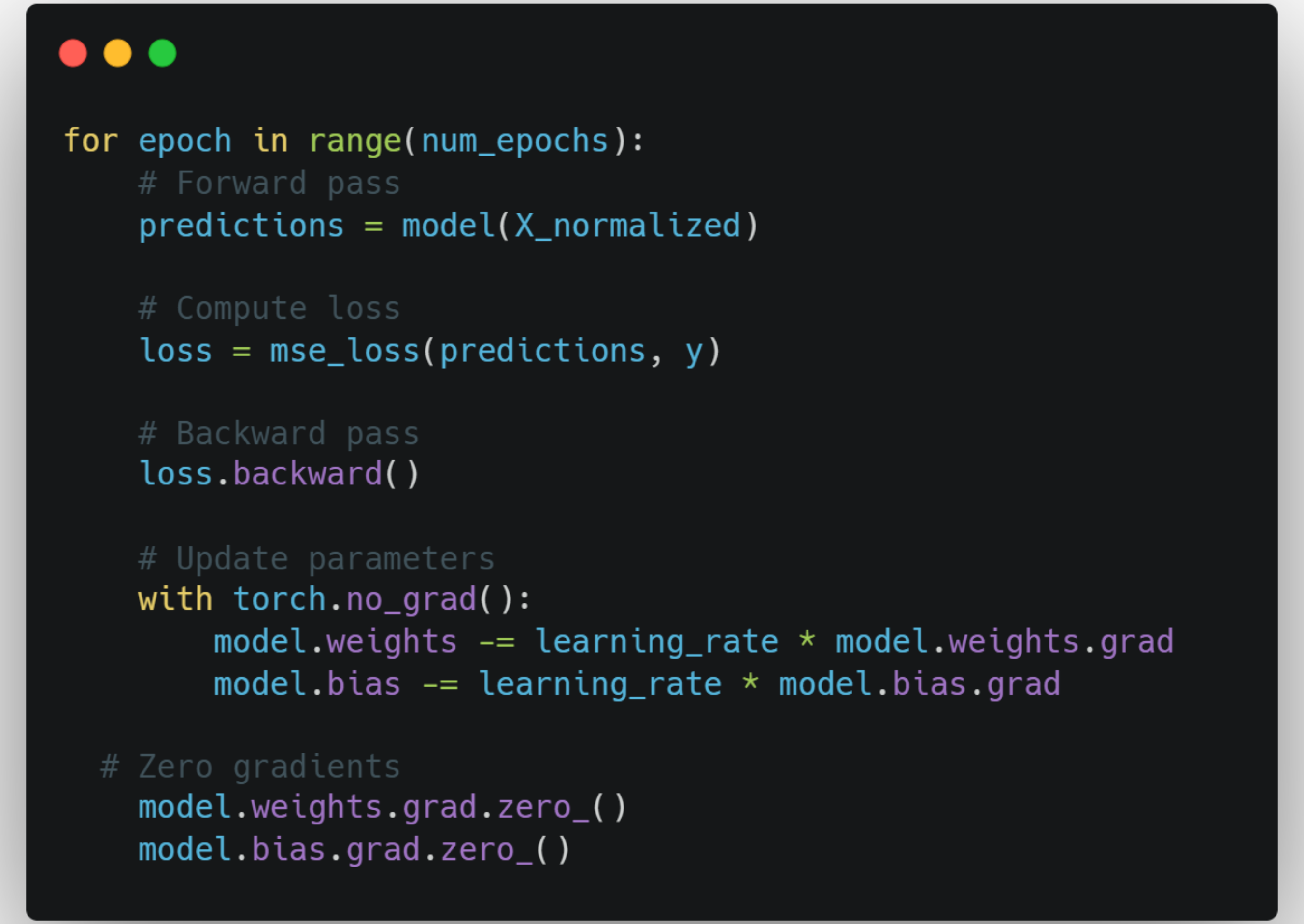
Where α is the learning rate $\alpha \frac{\partial L}{\partial w}$ is the gradient of loss with respect to weight



```python
# Backward pass: compute gradients
loss.backward()

# Update parameters using gradient descent
with torch.no_grad():
    model.weights -= learning_rate * model.weights.grad
    model.bias -= learning_rate * model.bias.grad

# Zero gradients for next iteration
model.weights.grad.zero_()
model.bias.grad.zero_()
```

TRAINING

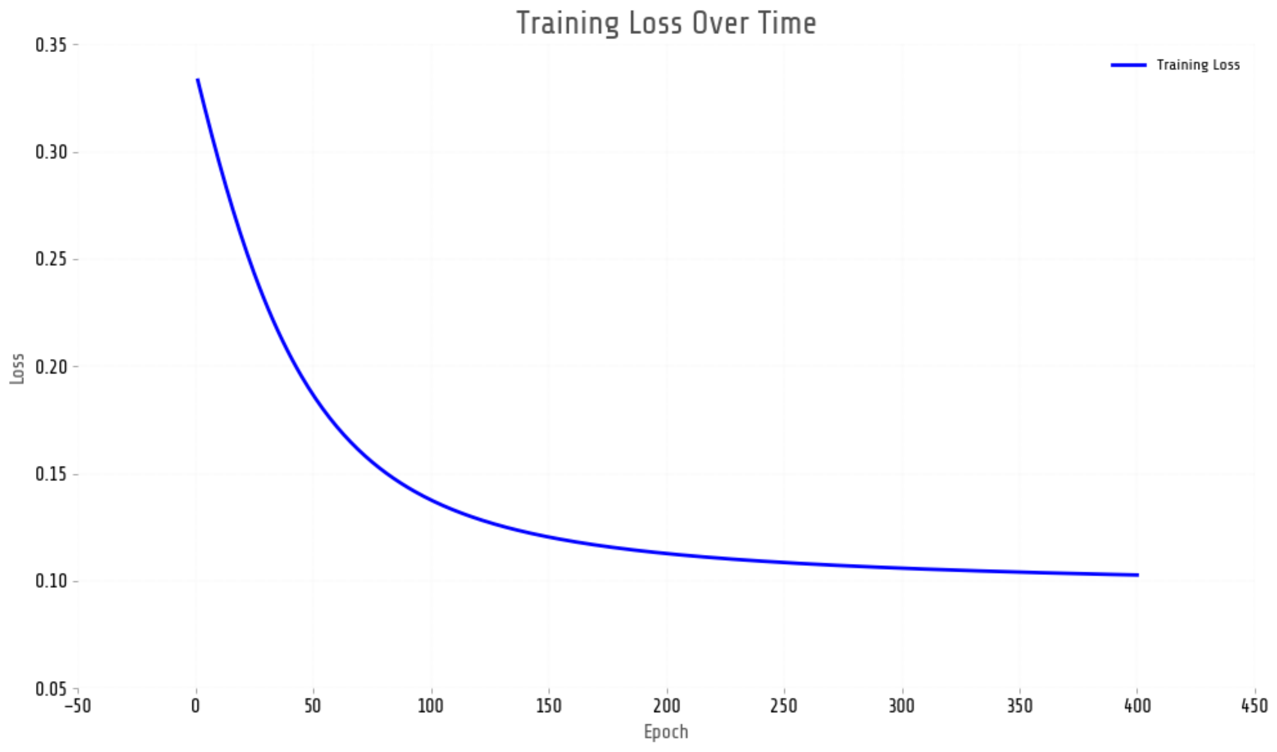
- Set learning rate and number of epochs
- Perform forward pass, loss calculation, backward pass, parameter update
- Monitor training progress

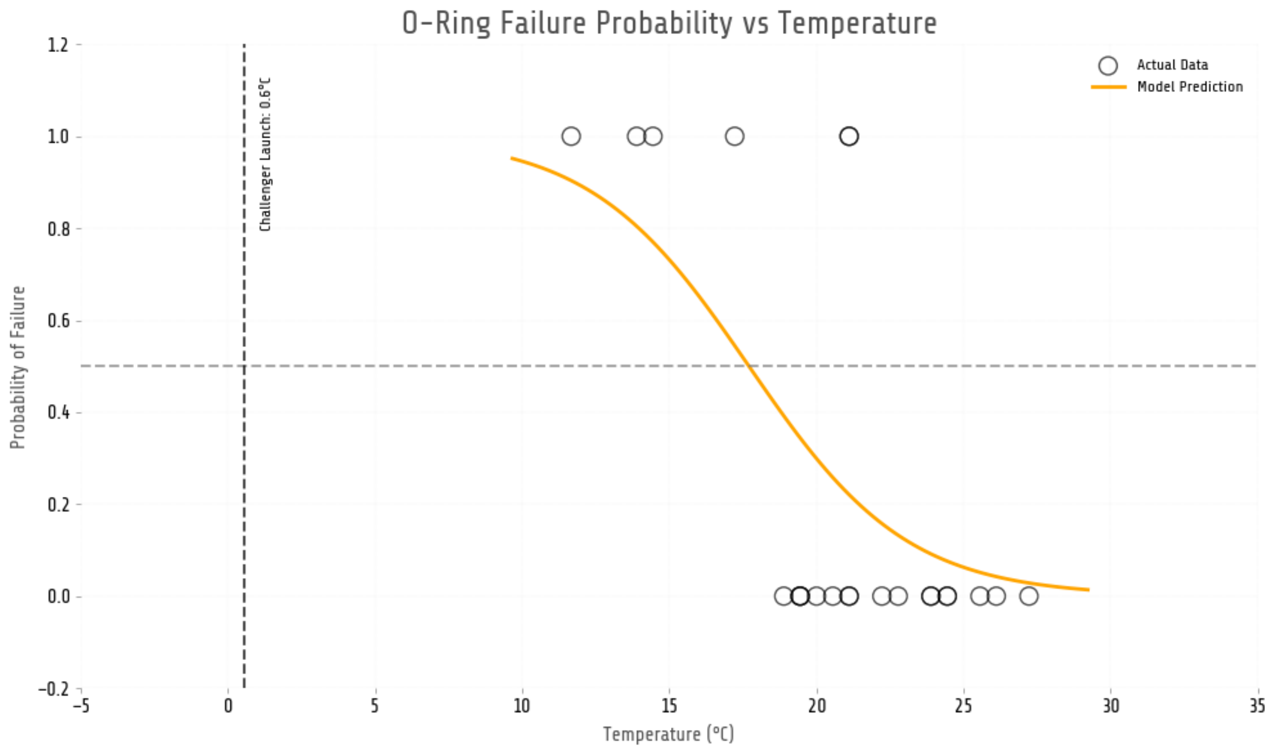


```
for epoch in range(num_epochs):
    # Forward pass
    predictions = model(X_normalized)

    # Compute loss
    loss = mse_loss(predictions, y)

    # Backward pass
    loss.backward()

    # Update parameters
    with torch.no_grad():
        model.weights -= learning_rate * model.weights.grad
        model.bias -= learning_rate * model.bias.grad

  # Zero gradients
    model.weights.grad.zero_()
    model.bias.grad.zero_()
```

TRACKING RESULTS

- Loss decreases over time (model improves)
- Model captures relationship between temperature and O-ring failure

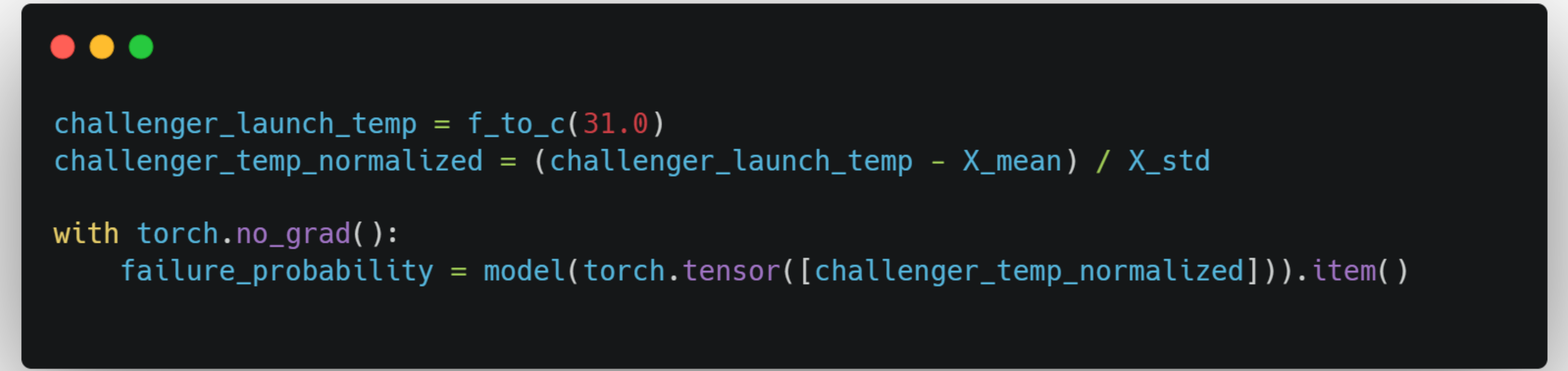




PREDICTING LAUNCH RISK

Challenger launch temperature: 31°F (-0.56°C)

- Model prediction: **99.9%** probability of O-ring failure
- Comparison: Risk at launch temperature is **3.3X** times higher than at normal temperature



```python
challenger_launch_temp = f_to_c(31.0)
challenger_temp_normalized = (challenger_launch_temp - X_mean) / X_std

with torch.no_grad():
    failure_probability = model(torch.tensor([challenger_temp_normalized])).item()
```

ETHICAL IMPLICATIONS

- Lives depend on accurate data analysis and interpretation
- Responsibility to thoroughly analyse data and clearly communicate risks
- Technical findings must be presented in ways decision-makers can understand
- Question: How can we ensure data insights are properly considered in critical decisions?