COMP-432: Machine Learning

Fall 2025

Week 2 LAB

Contents of this slide

- Probality review
- Linear Algebra Review
- Relating Probability and ML
- Relating Linear Algebra and ML

Math Review

Probability Concepts

Linear Algebra

Probability Review

- Probability Space
- Random Variables
- Probability Distributions
- Conditional Distributions
- Bayes and Chain Rule

Linear Algebra Review

- Basic Matrix Operations
- Matrix Multiplications
- Transpose of a Matrix
- Different Kind Of Matrices
- Determinant of a Matrix

Implementation Pytorch with different activation functions

```
class ExtendedLayer(nn.Module):
  def __init__(self, input_size=2, hidden_size=4, output_size=1,
activation='linear'):
    super(ExtendedLayer, self).__init__()
    self.fc1 = nn.Linear(input_size, hidden_size)
    self.fc2 = nn.Linear(hidden_size, output_size)
    self.activation = activation.lower()


  def forward(self, x):
    x = self.fc1(x)
    if self.activation == 'relu':
      x = nn.ReLU()(x)
    elif self.activation == 'silu':
      x = nn.SiLU()(x)
    elif self.activation == 'sigmoid':
      x = nn.Sigmoid()(x)
    elif self.activation == 'tanh':
      x = nn.Tanh()(x)
    else:
      pass  # linear
    x = self.fc2(x)
    return x
```

Implementation wrt Probability

```
# Expectation E(X) for features (discrete approx)
exp_hum = torch.mean(X[:, 0]).item()
exp_temp = torch.mean(X[:, 1]).item()
print(f"E(Humidity): {exp_hum}, E(Temperature): {exp_temp}")


# Variance Var(X)
var_hum = torch.var(X[:, 0]).item()
var_temp = torch.var(X[:, 1]).item()
print(f"Var(Humidity): {var_hum}, Var(Temperature): {var_temp}")


# Covariance Cov(Humidity, Temperature)
cov_matrix = torch.cov(X.T)
print("Covariance matrix:\n", cov_matrix.tolist())
```

Implementation wrt Probability

```
# Conditional expectation E(Rain|Humidity > mean)
high_hum_mask = X[:, 0] > exp_hum
cond_exp_rain_high_hum = torch.mean(y[high_hum_mask]).item()
print(f"E(Rain|Humidity > mean): {cond_exp_rain_high_hum}")


# Check approximate independence (cov ~0?)
cov_hum_rain = torch.cov(torch.cat((X[:, 0].unsqueeze(0), y.T),
dim=0))[0, 1].item()
print(f"Cov(Humidity, Rain) ~0 for independence: {cov_hum_rain}")
```

Implementation wrt Linear Algebra

```
# Cov matrix from prob section
print(f"Trace of cov: {torch.trace(cov_matrix).item()}")


# Determinant (for PSD check, |cov| >0)
det_cov = torch.det(cov_matrix).item()
print(f"Det of cov: {det_cov}")


# Inverse (if invertible)
if det_cov != 0:
    inv_cov = torch.inverse(cov_matrix)
    print("Inverse cov:\n", inv_cov.tolist())
```

Implementation wrt Linear Algebra

```
# Eigen decomp of cov (symmetric, relates to PSD)
eigvals, eigvecs = torch.linalg.eigh(cov_matrix)
print("Eigenvalues:\n", eigvals.tolist())
print("Eigenvectors:\n", eigvecs.tolist())
```

```
# Quadratic form x^T A x (A=cov, x=first sample)
quad_form = (X[0].T @ cov_matrix @ X[0]).item()
print(f"Quadratic form: {quad_form}")
```

Thank You