TP3: Artificial Neural Networks (ANN) with TensorFlow

Objectives

- Understand the principles and architecture of Artificial Neural Networks (ANN).
- Implement and train both *simple* and *deep* neural networks using *TensorFlow*.
- Apply ANN models to a real cybersecurity dataset (NSL-KDD).
- Compare and interpret the *performance differences* between simple and deep architectures.
- Discuss the impact of ANN models in network security systems.

Background

Artificial Neural Networks (ANNs) are computational models inspired by the human brain's structure, capable of learning complex patterns from data.

In network security, ANNs are used to detect anomalies or malicious behaviors in traffic data, enabling proactive identification of cyberattacks such as:

- **Denial of Service (DoS)** - overwhelming a system with traffic
- **Probing/Scanning attacks** - reconnaissance for vulnerabilities
- **Remote-to-Local (R2L)** - unauthorized access from remote machines
- **User-to-Root (U2R)** - privilege escalation attacks

The NSL-KDD dataset is a widely used benchmark for Intrusion Detection Systems (IDS).
It includes network connection records labeled as *Normal* or *Attack*.

Dataset Overview

Feature Type	Examples	Description
Basic Features	Duration, protocol type, service, flag	Connection-level info
Content Features	Failed logins, root shell, guest login	Content-based info
Traffic Features	Count, serror rate, rerror rate	Time-based traffic statistics
Target Label	Normal (0) / Attack (1)	Classification target

Part 1: Environment Setup

Python and TensorFlow Installation

1. Download Python 3.10 from https://www.python.org/downloads/
2. Check **"Add Python to PATH"** during installation

Create a Virtual Environment

```
python3.10 -m venv tf_env
tf_env\Scripts\activate
```

If execution error occurs:

```
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
```

Install Packages

```
pip install --upgrade pip
pip install tensorflow pandas scikit-learn matplotlib
```

Verification

```
python -c "import tensorflow as tf; print(tf.__version__)"
```

Recommended option: Jupyter Notebook

```
pip install notebook ipykernel
python -m ipykernel install --user --name=tf_env --display-name "Python (tf_env)"
jupyter notebook
```

Part 2: Data Loading and Exploration

1: Why do we set random seeds? What is reproducibility in machine learning?

Tasks to complete:

— Load the NSL-KDD dataset from the provided URL into a Pandas DataFrame.

— Display the first 5 rows and verify the dataset shape.

— Assign the following 43 column names:

```
# Load dataset
url = https://raw.githubusercontent.com/defcom17/NSL_KDD/master/KDDTrain+.txt

columns = [ "duration","protocol_type","service","flag","src_bytes","dst_bytes",
            "land","wrong_fragment","urgent","hot","num_failed_logins",
            "logged_in","num_compromised","root_shell","su_attempted",
            "num_root","num_file_creations","num_shells","num_access_files",
            "num_outbound_cmds","is_host_login","is_guest_login","count",
            "srv_count","serror_rate","srv_serror_rate","rerror_rate",
            "srv_rerror_rate","same_srv_rate","diff_srv_rate","srv_diff_host_rate",
            "dst_host_count","dst_host_srv_count","dst_host_same_srv_rate",
            "dst_host_diff_srv_rate","dst_host_same_src_port_rate",
            "dst_host_srv_diff_host_rate","dst_host_serror_rate",
            "dst_host_srv_serror_rate","dst_host_rerror_rate",
            "dst_host_srv_rerror_rate","label"]
```

— Display dataset information using `df.info()`.

— Analyze the label distribution (normal vs attack) using `value_counts()`.

2: How many samples are in the dataset?

3: How many features are present (excluding label and difficulty)?

4: What is the distribution between normal samples and attacks?

5: Why is this distribution important for model training?

Part 3: Data Preprocessing

Tasks to complete:

— Remove the difficulty column as it will not be used for training.

— Separate features (X) and labels (y).

— Encode categorical features (protocol_type, service, flag) using pd.get_dummies().

— Encode labels for binary classification:

- Normal traffic → 0
- Any attack → 1

— Apply `StandardScaler` to normalize all numeric features.

— Split the dataset: 80% training, 20% testing (random_state=42).

6: What encoding technique do you use for categorical variables? Why?

7: How many features do you have after encoding? Why did the number increase?

8: What type of classification problem is this (binary, multi-class, multi-label)?

9: Why is feature scaling important for neural networks?

10: How many samples are in the training and testing sets?

Part 4: Model Architecture Design

Critical Observation:

After preprocessing, you have **122 input features**.

11: If you create a shallow network with only 4 neurons in a single hidden layer, what problem might occur?

Tasks to complete:

— Create a function `build_model(n_hidden_layers, n_neurons, learning_rate, dropout_rate)` that:

1. Creates a Sequential model
2. Adds the specified number of hidden layers with ReLU activation
3. Includes dropout for regularization (only for deep networks)

4. Adds a sigmoid output layer for binary classification
5. Compiles with Adam optimizer and binary crossentropy loss

— Design two contrasting architectures:

Model 1: Shallow Network

Parameter	Value
Hidden layers	1
Neurons per layer	4
Learning rate	0.05 (high)
Batch size	512 (large)
Dropout	0.0
Epochs	15

Model 2: Deep Network

Parameter	Value
Hidden layers	3
Neurons per layer	32
Learning rate	0.001 (low)
Batch size	64 (medium)
Dropout	0.2
Epochs	15

12: Why use sigmoid activation for the output layer?

13: What is the purpose of dropout layers?

14: Compare the two architectures. Calculate:

- How many trainable parameters does each model have?
- What is the compression ratio for the shallow network? (122 inputs → 4 neurons)

15: Predict which model will perform better and explain why.

Part 5: Model Training

Tasks to complete:

For each model:

— Build the model using your function.

— Display the model architecture with model.summary().

— Train using model.fit() with validation_split=0.2.

— Evaluate on the test set.

— Store the training history and test accuracy.

16: What does validation_split=0.2 mean?

17: Do you observe any differences in training stability between the two models? Why?

Part 6: Results Analysis

Tasks to complete:

— Calculate and display:

- Shallow network test accuracy
- Deep network test accuracy
- Accuracy difference

18: Which model performs better? By how much?

19: Is this result expected based on your prediction in Question 15?

20: Explain why the shallow network underperforms. Consider:

- The bottleneck effect (122 features → 4 neurons)
- Learning capacity limitations
- Hyperparameter choices (high learning rate, large batch size)

21: Explain why the deep network performs better. Consider:

- Better capacity per layer (32 neurons)
- Multiple layers for hierarchical feature learning
- Better hyperparameters (lower learning rate, smaller batch size)
- Regularization (dropout)

22: Both models trained for the same number of epochs (15). Why does the deep network still perform better?

Part 7: Visualization

Tasks to complete:

— Create two subplots:

1. **Training Accuracy** over epochs for both models
2. **Validation Accuracy** over epochs for both models

— Create a bar chart comparing the test accuracies of both models. Include:

- Accuracy values on top of bars
- Visual annotation showing the improvement

23: What patterns do you observe in the training curves?
24: Is there evidence of overfitting in either model?
25: How would you present these results to non-technical stakeholders?

Part 8: Experimentation

Experiment 1: Degrade the shallow network

— Modify the shallow network to make it even worse:

- Change neurons from 4 to 2

26: What happens to the accuracy? Why?

Experiment 2: Improve the shallow network

— Try to improve the shallow network (keeping 1 layer):

- Increase neurons to **64**
- Decrease learning rate to **0.001**
- Decrease batch size to **64**
- Train for the same **15 epochs**

27: Does it perform better now? How does it compare to the deep network?
28: What does this tell you about the importance of:

- Network capacity (number of neurons)?
- Hyperparameter tuning?

Part 9: Discussion Questions

Q1: In what scenarios might a shallow network be preferable despite lower accuracy?
Q2: What are the trade-offs between shallow and deep networks in terms of:

- Training time
- Inference speed
- Memory requirements
- Interpretability

Q3: The deep network uses dropout. What would happen if we removed dropout? Would accuracy increase or decrease?