Comparative Report: ANN vs RNN vs ONNX with Use Cases

# 1. Introduction

This report provides a comparative overview of Artificial Neural Networks (ANN), Recurrent Neural Networks (RNN), and ONNX (Open Neural Network Exchange). It highlights their key differences, applications, and presents a sample implementation for each.

# 2. Model Overview

|  |  |  |  |
| --- | --- | --- | --- |
| Aspect | ANN (Artificial Neural Network) | RNN (Recurrent Neural Network) | ONNX (Open Neural Network Exchange) |
| Type | Feedforward Neural Network | Sequential Neural Network | Model exchange format |
| Data Type | Static / Tabular / Non-sequential | Sequential (e.g., time-series, text, audio) | N/A – used for deployment of various models |
| Memory | No memory of previous inputs | Remembers past input using hidden state | Not a model itself – stores models from other frameworks |
| Framework Support | Keras, TensorFlow, PyTorch | Keras, TensorFlow, PyTorch | Compatible with many (PyTorch, TF, scikit-learn, etc.) |
| Use Case | Classification, Regression | Text classification, language modeling, speech | Deployment, platform transfer |
| Training | Trained using backpropagation | Trained using backpropagation through time (BPTT) | Not trained itself |

# 3. Use Case Implementation

## 3.1 ANN Use Case: Health Risk Prediction (Tabular Data)

Goal: Predict whether a person is at health risk based on age, weight, and blood pressure.  
- Type: Classification  
- Data: Tabular (Age, Weight, BP)  
- Tools: TensorFlow, Keras

CODE;

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

X = np.array([

    [25, 55, 120],

    [40, 90, 145],

    [30, 70, 130],

    [60, 80, 160],

    [22, 50, 110],

    [55, 95, 170]

])

y = np.array([0, 1, 0, 1, 0, 1])

PREPROCESSING

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.3)

MODEL

model\_ann = Sequential([

    Dense(16, activation='relu', input\_shape=(3,)),

    Dense(8, activation='relu'),

    Dense(1, activation='sigmoid')

])

TRAINING AND COMPILATION

model\_ann.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

model\_ann.fit(X\_train, y\_train, epochs=30, verbose=0)

PREDICT

print("ANN prediction (health risk):", model\_ann.predict(X\_test))

**Result:**

**The model gave the outputs as probabilities as I have used sigmoid activation function at last.**

· **0.59988** → About **60% chance** that the first sample is **"high risk" (label 1)**

· **0.42258** → About **42% chance** that the second sample is high risk → Likely **"low risk" (label 0)**

## 3.2 RNN Use Case: Symptom Text Classification

Goal: Classify free-text symptom descriptions as emergency or non-emergency.  
- Type: Text classification  
- Data: Sequential (sentences)  
- Tools: TensorFlow, Keras, Tokenizer, Embedding, RNN layers

MODEL TRAINED TO CLASSIY SYMPTOMS AS EMERGENCY OR NON-EMERGENCY BASED ON THE SEVERITY OF THE SYMPTOMS

IMPORTING IMPORTANT MODULES

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding,SimpleRNN,Dense

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

INPUT TEXTS AND OUTPUTS LABELS;

texts = [

    "chest pain and shortness of breath",

    "mild headache and fatigue",

    "severe bleeding after injury",

    "dizziness after standing up",

    "high fever and cough"

]

labels = [1, 0, 1, 0, 0]

TOKENIZATION TO CONVERT TEXTS INTO TOKENS AND COMPARE THE SEVERITY BASED ON THE INPUT LABELS .1-EMERGENCY,0-NON-EMERGENCY

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(texts)

seqs = tokenizer.texts\_to\_sequences(texts)

padded = pad\_sequences(seqs)

MODEL TRAINING AND EVALUATION;

vocab\_size = len(tokenizer.word\_index) + 1

model\_rnn = Sequential([

    Embedding(input\_dim=vocab\_size, output\_dim=8, input\_length=padded.shape[1]),

    SimpleRNN(16),

    Dense(1, activation='sigmoid')

])

model\_rnn.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

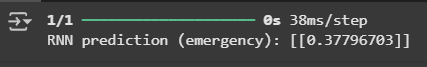
model\_rnn.fit(padded, np.array(labels), epochs=50, verbose=0)

CLASSIFICATION;

sample = ["severe bleeding and chest pain"]

sample\_seq = pad\_sequences(tokenizer.texts\_to\_sequences(sample), maxlen=padded.shape[1])

print("RNN prediction (emergency):", model\_rnn.predict(sample\_seq))



## 3.3 ONNX Use Case: Model Deployment

Goal: Export a trained ANN or RNN model to ONNX format for cross-platform deployment (e.g., mobile or edge device).  
- Type: Model conversion  
- Tools: tf2onnx or PyTorch -> ONNX exporters  
- Output: `.onnx` file for inference in different environments

#  Install tf2onnx if not already installed

!pip install tf2onnx

#  Import required libraries

import numpy as np

import tensorflow as tf

import tf2onnx

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

# Prepare the dummy data

# Features: [Age, Weight, Systolic BP]

X = np.array([

    [25, 55, 120],

    [40, 90, 145],

    [30, 70, 130],

    [60, 80, 160],

    [22, 50, 110],

    [55, 95, 170]

])

y = np.array([0, 1, 0, 1, 0, 1])

#  Preprocess the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.3)

# Define and train the ANN model

model\_ann = Sequential([

    Dense(16, activation='relu', input\_shape=(3,)),

    Dense(8, activation='relu'),

    Dense(1, activation='sigmoid')

])

model\_ann.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

model\_ann.fit(X\_train, y\_train, epochs=30, verbose=0)

model\_ann.save("health\_model.h5")

# Convert to ONNX directly from saved Keras model

!python -m tf2onnx.convert --keras health\_model.h5 --output health\_risk\_ann\_model.onnx --opset 13

print(" ANN model converted and saved as health\_risk\_ann\_model.onnx")

# 4. Summary of Strengths

|  |  |
| --- | --- |
| Model | Strengths |
| ANN | Simple architecture, fast training, great for tabular data |
| RNN | Best for sequence/time-dependent data, can model context over time |
| ONNX | Ideal for deployment, cross-framework compatibility, real-time inference |

# 5. Conclusion

Each architecture serves a different purpose:  
- ANN is best suited for simple input-output mappings in structured data.  
- RNN is powerful for sequential data analysis and modeling temporal relationships.  
- ONNX enhances flexibility in deploying ML models across platforms and environments.  
  
By understanding these differences and strengths, developers can choose the appropriate model and deployment path based on their specific application.