Traditional Neural Networks (TNNs)

What are Traditional Neural Networks?

Traditional Neural Networks (TNNs), also known as feedforward neural networks, are models made up of layers of connected nodes or "neurons." These networks process inputs by passing them through layers in one direction—from the input layer, through one or more hidden layers, to the output layer.

Why Use TNNs?

TNNs are suitable for tasks with fixed-size input and output. Common applications include:

- Predicting house prices based on fixed features (e.g., size, location, number of rooms).
- Classifying fixed-size images (e.g., as "cat" or "dog").
- Recognizing handwritten digits (e.g., using the MNIST dataset).

How Do TNNs Work?

1. **Input Layer**: Receives raw data (e.g., pixel values, house features).
2. **Hidden Layers**: Each neuron computes a weighted sum of its inputs, applies an activation function, and passes the output forward.
3. **Output Layer**: Produces the final result (e.g., classification label or numeric prediction).

Key Characteristics

- No memory: Each input is processed independently without reference to past data.
- Fixed input/output sizes.
- Straightforward flow from input to output.

Analogy

A TNN can be imagined as a production line:

- Raw materials (input) enter one end.
- They are processed at each station (layer).
- The final product (output) comes out the other end.

Summary

- TNNs are feedforward models.
- They work with fixed-size data.
- Each input is processed independently.
- Best for simple classification or regression tasks.

Key Components of TNNs

1. **Neurons (Nodes)**

- Basic units of the network.
- Receive inputs, process them, and pass output to the next layer.

2. **Weights**

- Control the strength of connections between neurons.
- Learned during training to improve performance.

3. **Bias**

- A constant added to the input sum, allowing flexibility in fitting data.

4. **Weighted Sum**

- Each neuron computes: z = w1*x1* + *w2*x2 + … + wn*xn + b

5. **Activation Function**

- Introduces non-linearity to allow learning of complex patterns.
- Common functions:
    - ReLU: max(0, x)
    - Sigmoid: compresses input to 0–1
    - Tanh: compresses input to -1 to 1

6. **Layers**

- Input Layer: Receives data.
- Hidden Layers: Process data.
- Output Layer: Produces prediction.

7. **Forward Propagation**

- Data moves through the network layer by layer to compute an output.

8. **Loss Function**

- Measures prediction error.
- Examples:
    - Mean Squared Error (regression)
    - Cross-Entropy Loss (classification)

9. **Training and Backpropagation**

- Adjusts weights to minimize loss.
- Uses gradient descent based on backpropagation.
- Repeats over many epochs to learn patterns.

Recurrent Neural Networks (RNNs) — Explained

What are RNNs?

RNNs are designed to process sequences by maintaining a memory of previous inputs. Unlike TNNs, RNNs can handle temporal data where order matters.

Why Use RNNs?

RNNs are useful for sequence-based tasks:

- Text or speech (sequences of words or sounds)
- Time series data
- Video (sequences of frames)

TNNs lack memory, making them unsuitable for such problems. RNNs retain context from earlier steps.

How Do RNNs Work?

At each time step:

1. Takes the current input (xt).
2. Combines it with the previous hidden state (ht-1).
3. Produces a new hidden state (ht), which is passed forward.

Mathematically:
ht = activation(Wx *xt* + *Wh* ht-1 + b)

Types of RNNs Based on Input/Output

- **Vector-to-Sequence**: Fixed input, variable-length output.
    - Example: Image captioning.
- **Sequence-to-Vector**: Sequence input, single output.
    - Example: Sentiment analysis.
- **Sequence-to-Sequence**: Sequence input, sequence output.
    - Example: Language translation.

Summary of RNNs

- Handle sequential data with memory.
- Process inputs one step at a time.
- Suitable for tasks like language modeling, speech recognition, and time series prediction.

RNNs vs TNNs

Feature	TNN	RNN
Memory	No	Yes
Input Type	Fixed	Sequential
Use Case	Classification, Regression	Language, Time Series

Limitations of RNNs

- Slow due to step-by-step processing.
- Struggle with long-term dependencies (forgetting earlier inputs).
- Can suffer from vanishing/exploding gradients.

Advancements

To address these issues, enhanced models were developed:

- **LSTM (Long Short-Term Memory)**
- **GRU (Gated Recurrent Unit)**

More recently, **Transformers** have largely replaced RNNs for many tasks due to their ability to handle long sequences efficiently.

LSTM (Long Short-Term Memory) — Full Process Recap

How LSTM Works: Word by Word

Given a sentence like "I love deep learning models," the LSTM processes each word one at a time. At every time step (word), the LSTM performs the following:

1. Forget Gate

- **Question**: "Should I forget anything from my memory?"
- Inputs: Current word (x_t) and previous hidden state (h_{t-1})
- Role: Decides what part of the past (cell state C_{t-1}) should be erased.

2. Input Gate

- **Question**: "Should I learn anything new from this word?"
- Builds candidate information from the current input.
- Filters it to determine what should be added to the memory.

3. Memory Update (Cell State C_t)

- Combines retained past memory with new filtered input.
- Updates the long-term memory.

4. Output Gate

- **Question**: "What should I pass forward or output?"
- Uses updated memory to generate hidden state (h_t).
- h_t is passed to the next time step and/or used for prediction.

Memory Types in LSTM

Memory Type	Role
Cell State (C_t)	Long-term memory (main memory bank)
Hidden State (h_t)	Short-term memory (used in output)

Strengths of LSTM

- Remembers important data over long sequences.
- Learns to forget irrelevant information.
- Balances memory retention, new learning, and forgetting.

Use Cases

- **Sentiment Analysis**: Detecting polarity in phrases like "I don't like this movie."

- **Machine Translation**: Capturing sentence-level meaning for accurate translation.
- **Time Series Forecasting**: Remembering previous trends to predict future values.

What Happens After Sequence Ends?

1. **Final Output**: Last hidden state (h_t) holds summary of entire input.
2. **Task Application**:
    - Classification: Final h_t is passed to a dense layer for class prediction.
    - Sequence Generation: Use all hidden states in a decoder model.
3. **Loss Calculation**: Prediction is compared with the correct answer.
4. **Backpropagation Through Time (BPTT)**: Model learns by adjusting weights across time steps.
5. **Repeat**: Training continues over many epochs to improve accuracy.

GRU (Gated Recurrent Unit) — Step-by-Step Explanation

What is a GRU?

A GRU is a streamlined version of LSTM with fewer gates and computations. It processes sequences efficiently while preserving essential memory.

Working of GRU

Step 1: Inputs

- Current input (x_t)
- Previous hidden state (h_{t-1})

Step 2: Update Gate (z)

- Controls how much of the past hidden state to keep.
- z = 1 → keep all past info, z = 0 → forget all.

Step 3: Reset Gate (r)

- Decides how much past info to ignore for the current step.
- Helps focus on new input when past isn't useful.

Step 4: Candidate Hidden State ($\hat{h}$)

- Combines current input and reset-modified past info to generate a candidate for the new hidden state.

Step 5: Final Hidden State (h_t)

- Blends previous hidden state and candidate state based on update gate.
- h_t is passed forward for the next step.

Why GRU?

- Fewer gates (no separate memory like C_t).
- Trains faster and uses fewer resources.
- Suitable for sequence tasks requiring speed and efficiency.

Simple Example: Sentiment Analysis

Sentence: "The movie was amazing"

- "The": GRU starts with no memory.
- "movie": Updates with topic info.
- "was": Retains relevant past context.
- "amazing": Heavily influences final memory.
- Final hidden state is passed to a classifier → Output: "Positive."

CNN (Convolutional Neural Network) Overview

What is a CNN?

CNNs are deep learning models optimized for capturing spatial features. Initially built for images, they can also be adapted for text and sequence data.

How CNNs Work (for Images)

1. **Input**: Image (e.g., 64x64 RGB) as a 3D array.
2. **Convolution Layer**:
    - Applies filters (e.g., 3x3 matrix) across the image.
    - Each filter detects specific features like edges or patterns.
    - Produces a "feature map."
3. **Activation (ReLU)**:
    - Converts negative values to zero.

- Adds non-linearity to the model.

4. **Pooling (Max Pooling)**:
    - Reduces feature map size.
    - Keeps only the most important values.
    - Adds translation invariance.
5. **Stacking Layers**:
    - Deeper layers extract more abstract patterns (e.g., object shapes).
6. **Flattening**:
    - Converts 3D feature maps into a 1D vector.
7. **Fully Connected Layers**:
    - Use features to predict final class.
8. **Output Layer**:
    - Softmax function outputs class probabilities.
    - Prediction = class with highest probability.

CNN Workflow

Image → Convolution → ReLU → Pooling → (repeat) → Flatten → Dense Layers → Softmax → Prediction

CNN in NLP

- Input: Sentence converted to word embeddings.
- CNN filters slide over n-grams (2–3 words at a time).
- Detects local patterns like phrases or expressions.
- Used in:
    - Sentence classification
    - Text categorization
    - Named entity recognition

Comparison Table

Model	Strengths	Weaknesses
CNN	Fast, detects local patterns	Poor at long-range dependencies

RNN	Good at sequential data	Slower, hard to parallelize
LSTM/GRU	Handles memory over sequences	More compute than CNNs
Transformer	Best at long-range context, parallel	High compute cost

CNN Summary

- CNNs extract features efficiently.
- Best for tasks focusing on local patterns.
- Useful in both image and text domains.
- Not ideal for capturing long-term dependencies.