Deep Learning

BCSE-332L

Module 4:

Recurrent Neural Network

Dr . Saurabh Agrawal

Faculty Id: 20165

School of Computer Science and Engineering

VIT, Vellore-632014

Tamil Nadu, India

Outline

- Recurrent Neural Networks
- Bidirectional RNNs
- Auto Encoders
- Encoder
- Decoder
- Sequence-to-Sequence Architectures
- Deep Recurrent Networks
- Bidirectional Encoder Representations from Transformers (BERT)

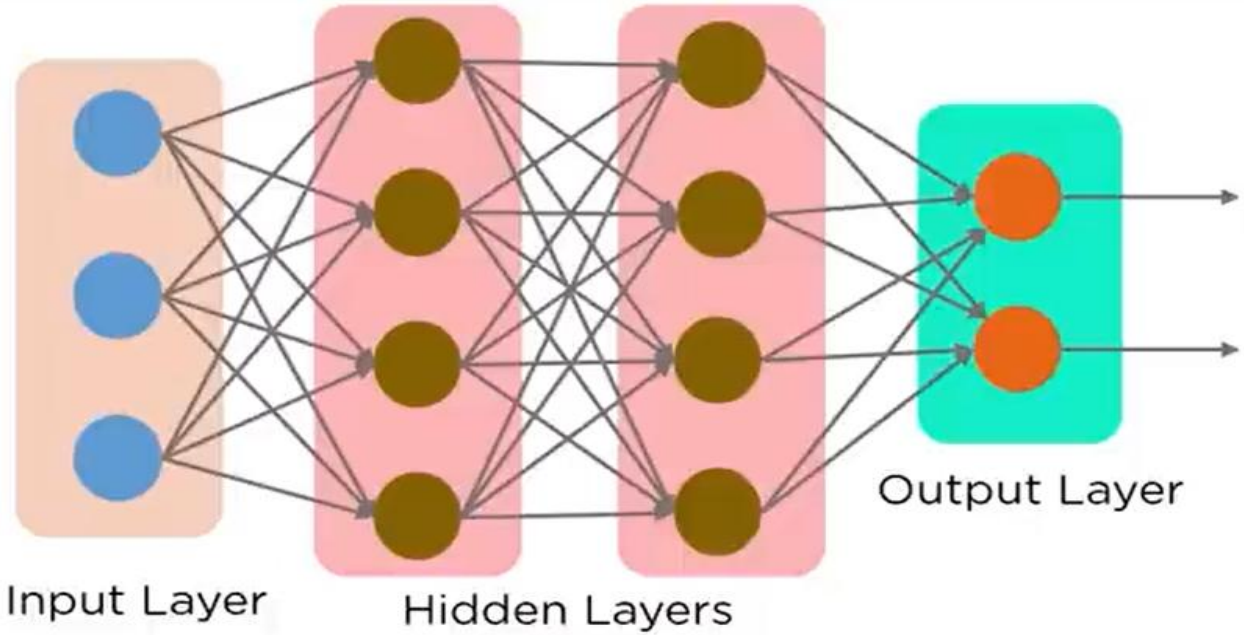
Recurrent Neural Networks

- **Why RNN??**
- CNNs are great at recognizing **objects**, **animals** and **people**, **but what if we want to understand**
- **what is happening in a picture?**
- Consider a picture of a ball in the air.
- Determining whether the **ball** is rising or falling would require more context than a single Picture.
- **For example**, a video whose sequence could clarify whether the ball is **going up** or **down**.
- This, in turn, would require the **neural network** to "**remember**" previously encountered **information and factor** that into **future calculations**.
- And the problem of remembering goes beyond videos:
- **For example**: **Many Natural Language** understanding algorithms typically deal only with **text**, **but need to recall information** such as the **topic of a discussion** or **previous words in a sentence**.

Recurrent Neural Networks

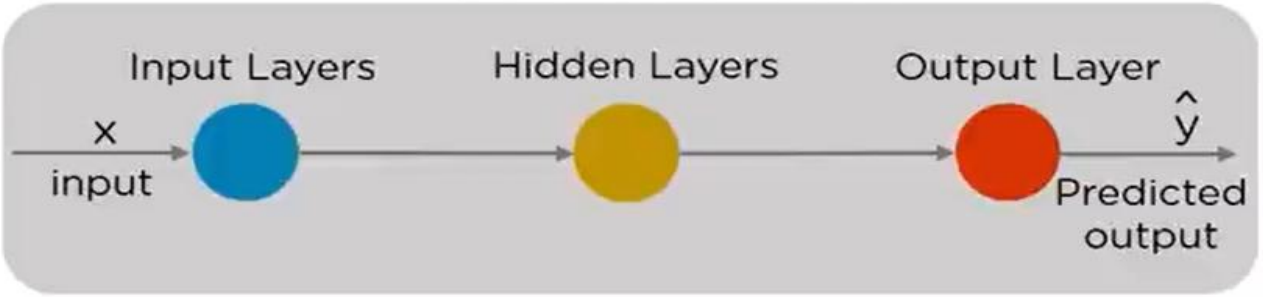
❑Issues in FEED Forward Neural Network?

In a Feed-Forward Network , information flows only in forward direction, from the input nodes, through the hidden layers (if any) and to the output nodes. There are no cycles or loops in the network.



Decisions are based on current input
No memory about the past
No future scope

Simplified presentation



Recurrent Neural Networks

❑Issues in FEED Forward Neural Network?

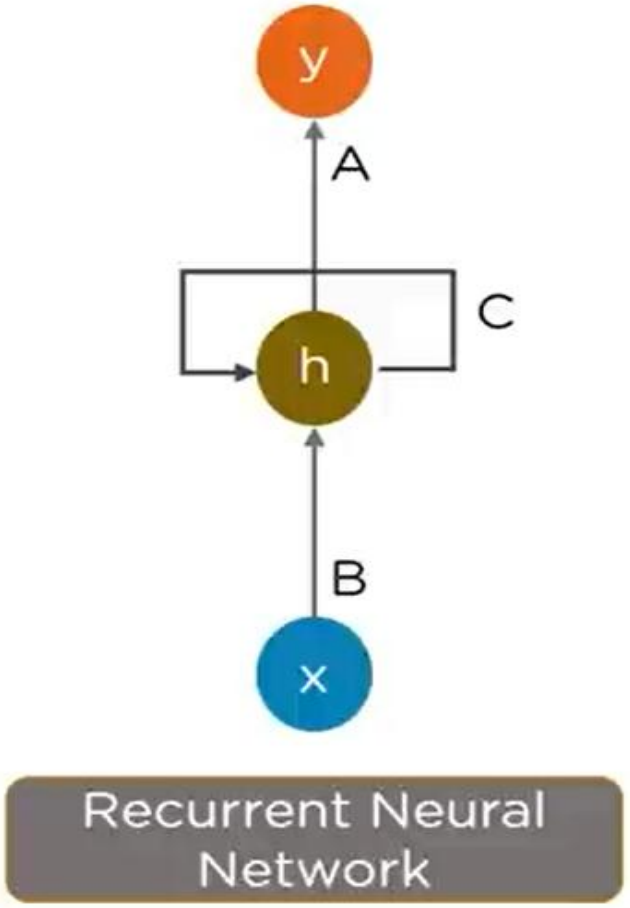
01 cannot handle sequential data

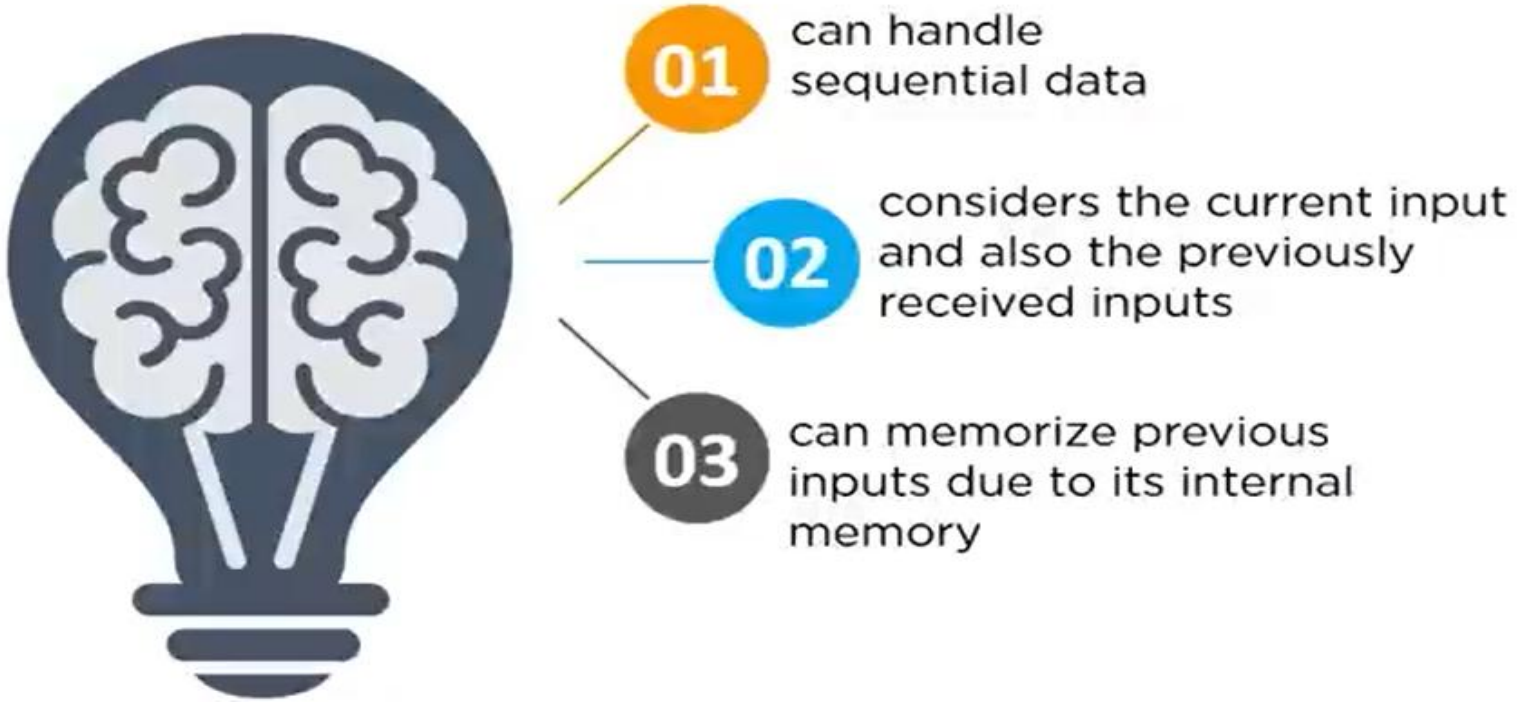
02 considers only the current input

03 cannot memorize previous inputs

Recurrent Neural Networks

❑Reason to use RNN:





Recurrent Neural Networks

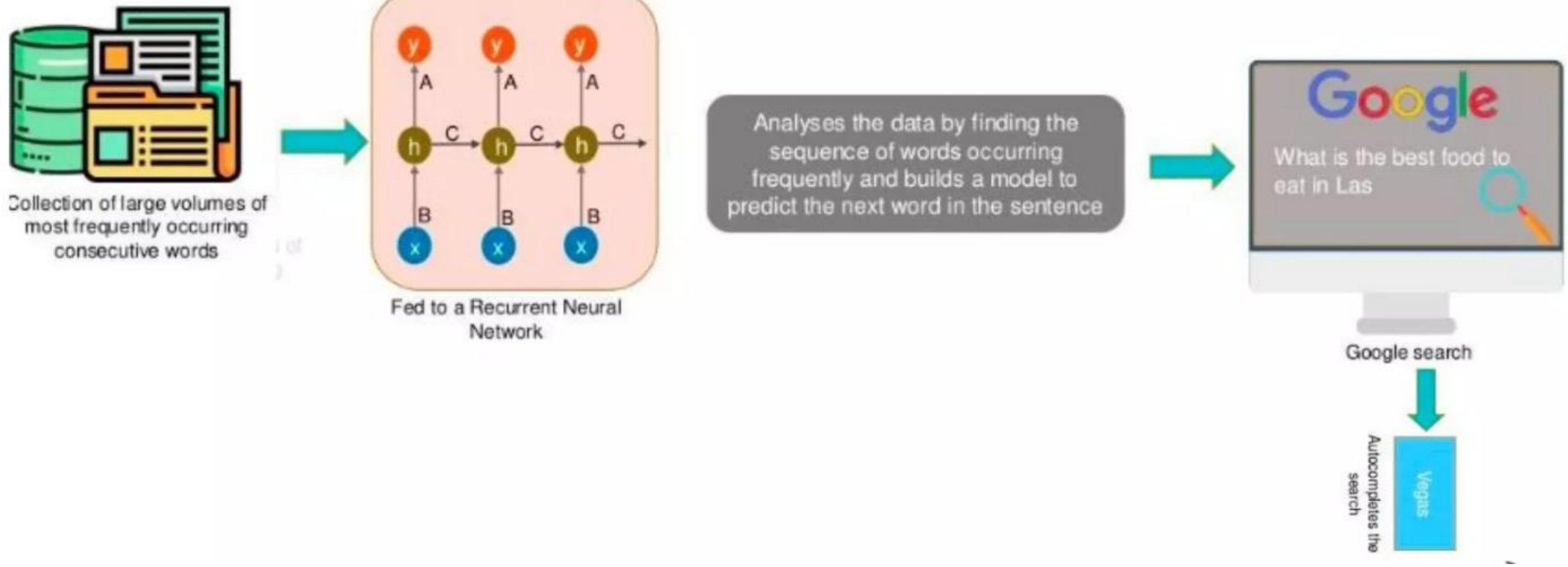
❑RNN in Google:

Do you know how Google's autocomplete feature predicts the rest of the words a user is typing?

Recurrent Neural Networks

❑RNN in Google:

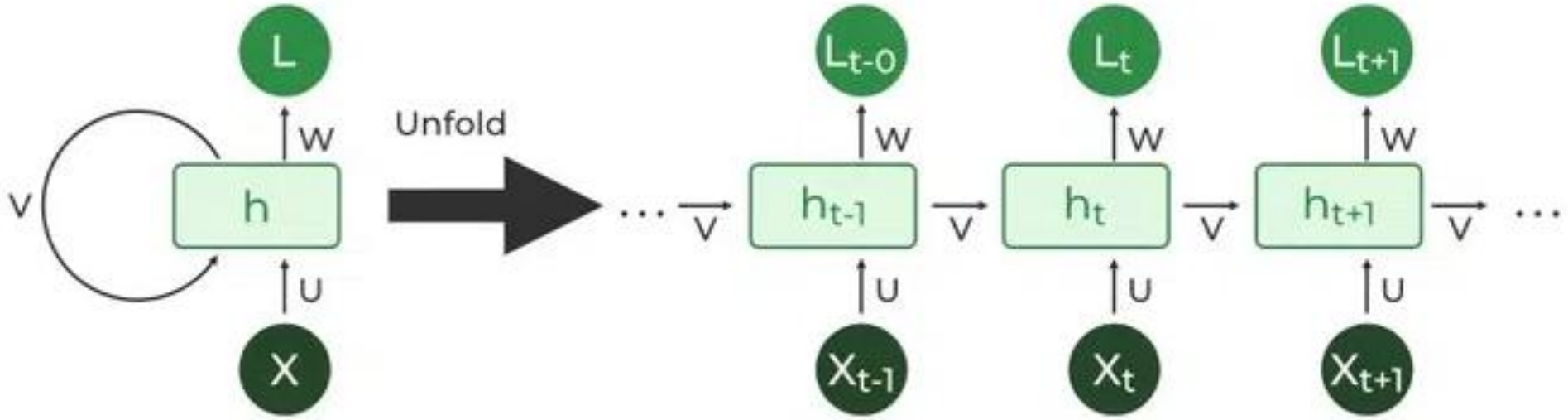
Do you know how Google's autocomplete feature predicts the rest of the words a user is typing?



Recurrent Neural Networks

- Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step.
- In traditional neural networks, all the inputs and outputs are independent of each other.
- Still, in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words.
- Thus RNN came into existence, which solved this issue with the help of a Hidden Layer.
- The main and most important feature of RNN is its **Hidden state**, which remembers some information about a sequence.
- The state is also referred to as Memory State since it remembers the previous input to the network.
- It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output.
- This reduces the complexity of parameters, unlike other neural networks.

Recurrent Neural Networks



Recurrent Neural Networks

❑How RNN differs from Feedforward Neural Network?

❑Artificial neural networks that do not have looping nodes are called feed forward neural networks.

❑Because all information is only passed forward, this kind of neural network is also referred to as a multi-layer neural network.

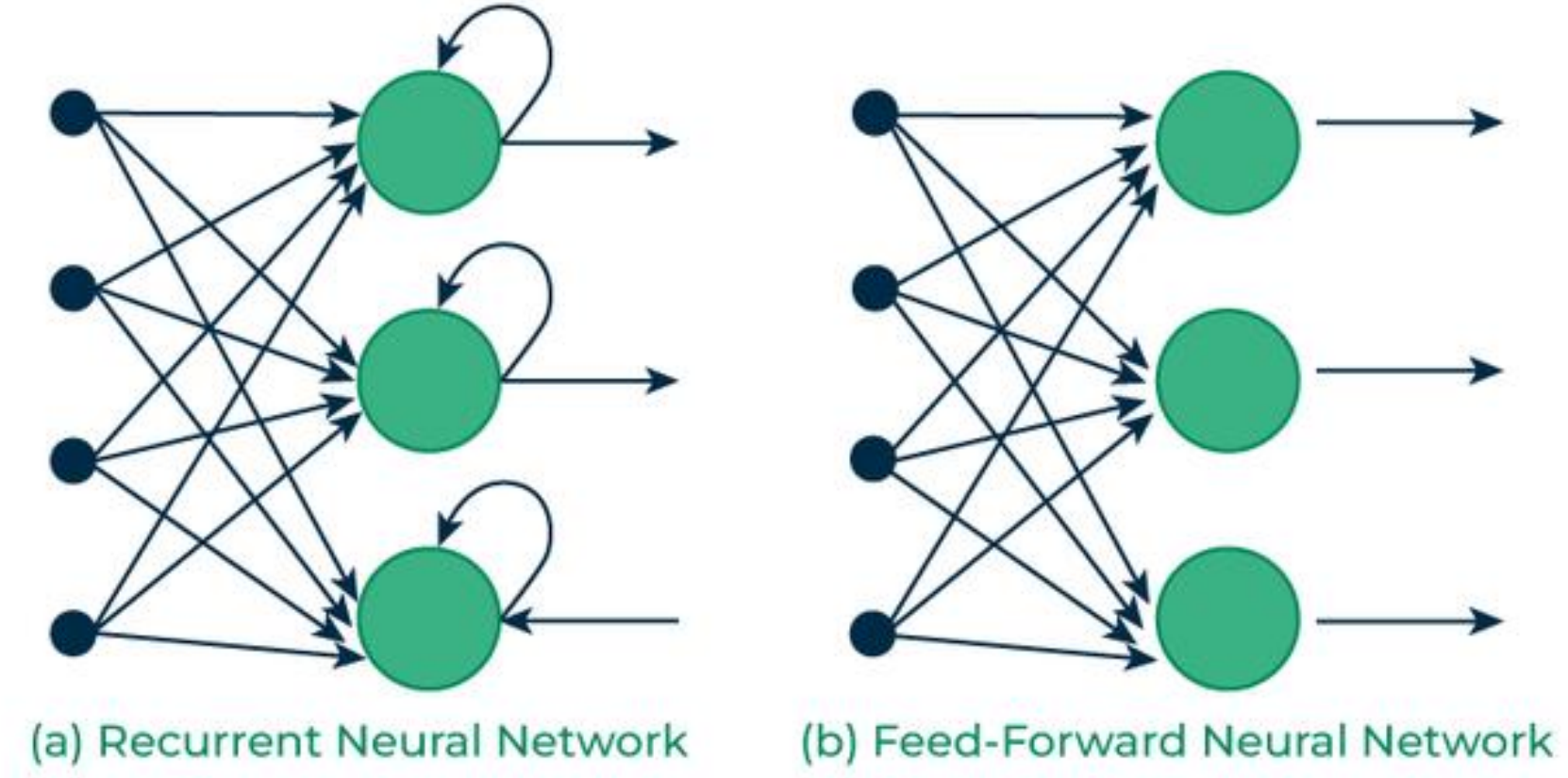
❑Information moves from the input layer to the output layer: if any hidden layers are present: unidirectionally in a feedforward neural network.

❑These networks are appropriate for image classification tasks, for example, where input and output are independent.

❑Nevertheless, their inability to retain previous inputs automatically renders them less useful for sequential data analysis.

Recurrent Neural Networks

❑How RNN differs from Feedforward Neural Network?



Recurrent Neural Networks

❑Recurrent Neuron and RNN Unfolding

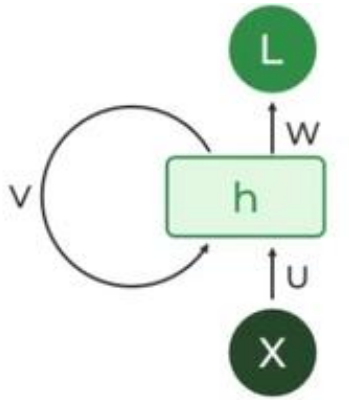
❑The fundamental processing unit in a Recurrent Neural Network (RNN) is a Recurrent Unit, which is not explicitly called a “Recurrent Neuron.”

❑This unit has the unique ability to maintain a hidden state, allowing the network to capture sequential dependencies by remembering previous inputs while processing.

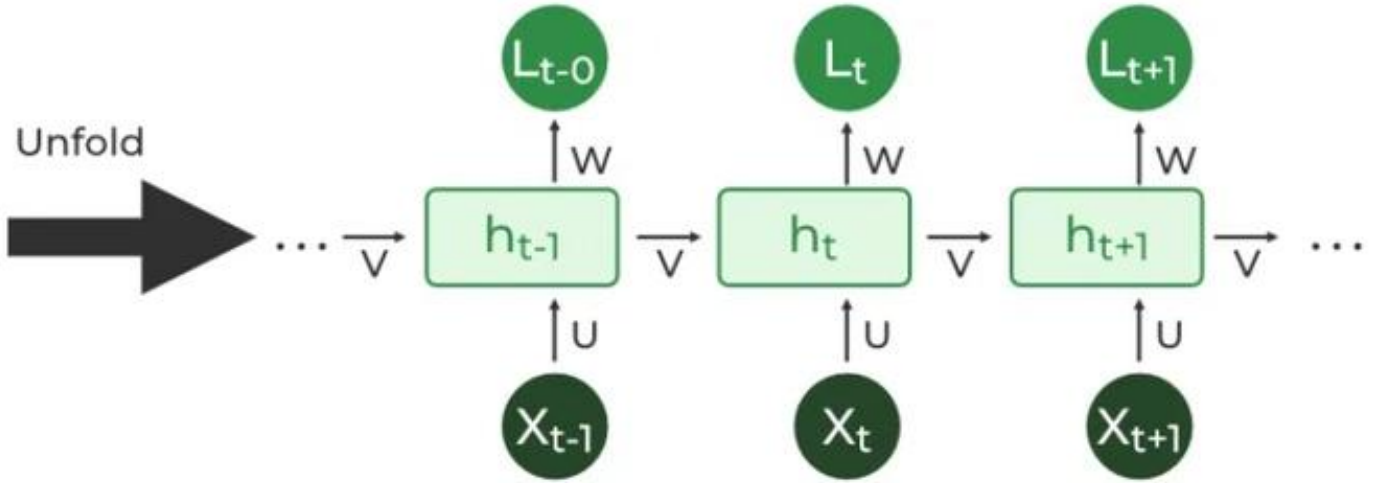
❑Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) versions improve the RNN’s ability to handle long-term dependencies.

Recurrent Neural Networks

❑Recurrent Neuron and RNN Unfolding



Recurrent Neuron



Recurrent Neural Networks

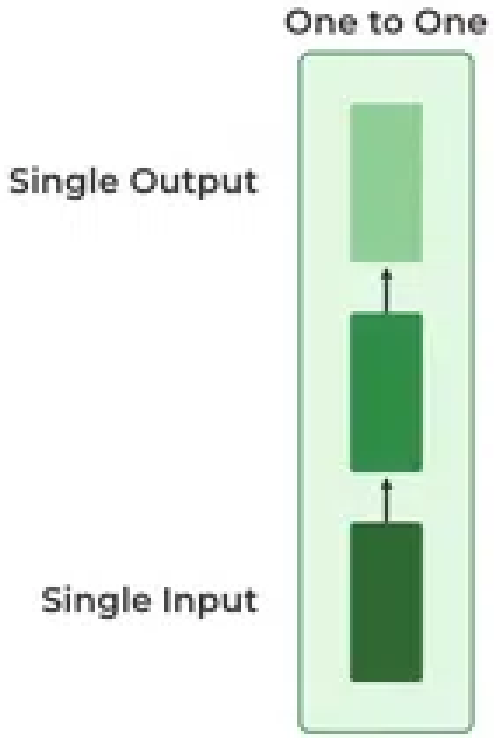
❑**Types Of RNN:** There are four types of RNNs based on the number of inputs and outputs in the network.

1. **One to One**
2. **One to Many**
3. **Many to One**
4. **Many to Many**

❑Types Of RNN: One to One

❑This type of RNN behaves the same as any simple Neural network it is also known as Vanilla Neural Network.

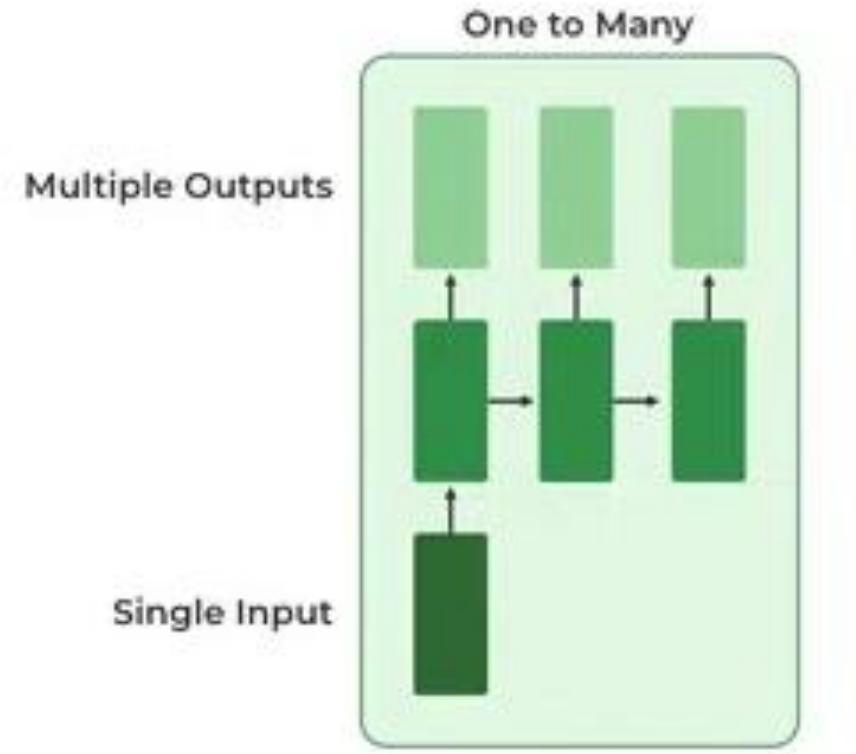
❑In this Neural network, there is only one input and one output.



❑Types Of RNN: One To Many

❑In this type of RNN, there is one input and many outputs associated with it.

❑One of the most used examples of this network is Image captioning where given an image we predict a sentence having Multiple words.

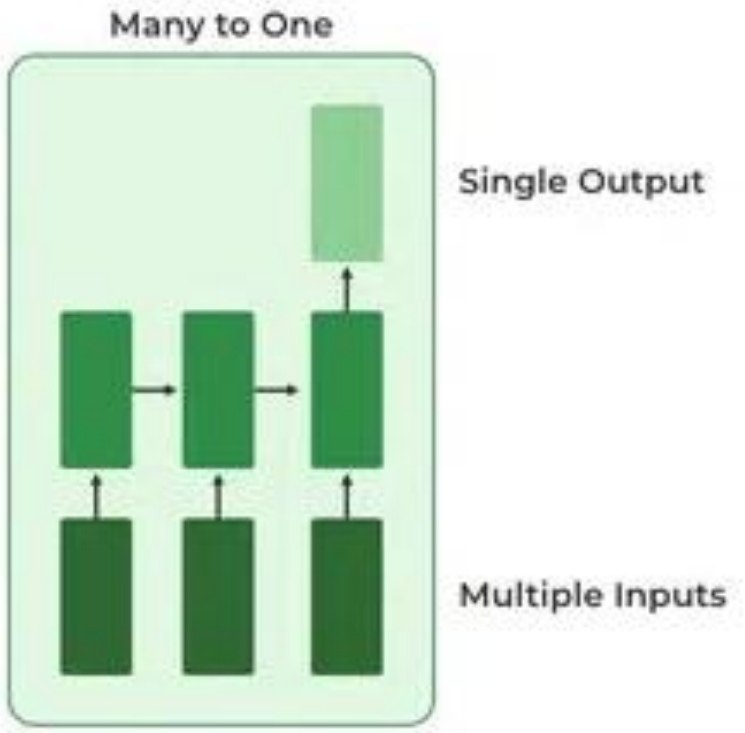


❑Types Of RNN: Many to One

❑In this type of network, Many inputs are fed to the network at several states of the network generating only one output.

❑This type of network is used in the problems like sentimental analysis.

❑Where we give multiple words as input and predict only the sentiment of the sentence as output.



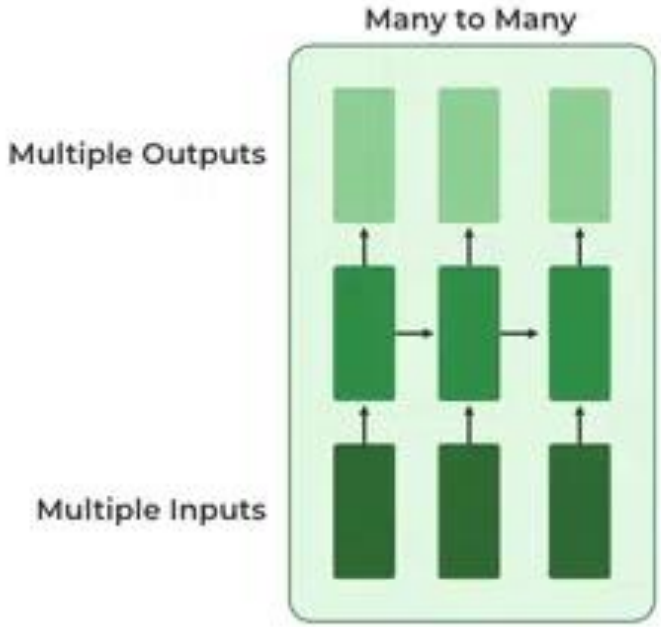
Recurrent Neural Networks

❑**Types Of RNN: Many to Many**

❑In this type of neural network, there are multiple inputs and multiple outputs corresponding to a problem.

❑One Example of this Problem will be language translation.

❑In language translation, we provide multiple words from one language as input and predict multiple words from the second language as output.



Recurrent Neural Networks

❑Recurrent Neural Network Architecture:

❑RNNs have the same input and output architecture as any other deep neural architecture.

❑However, differences arise in the way information flows from input to output.

❑Unlike Deep neural networks where we have different weight matrices for each Dense network in RNN, the weight across the network remains the same.

❑It calculates state hidden state H_i for every input $\mathbf{X_i}$.

❑**By using the following formulas:**

$$h = \sigma(UX + Wh_{-1} + B)$$

$$Y = O(Vh + C)$$

Hence

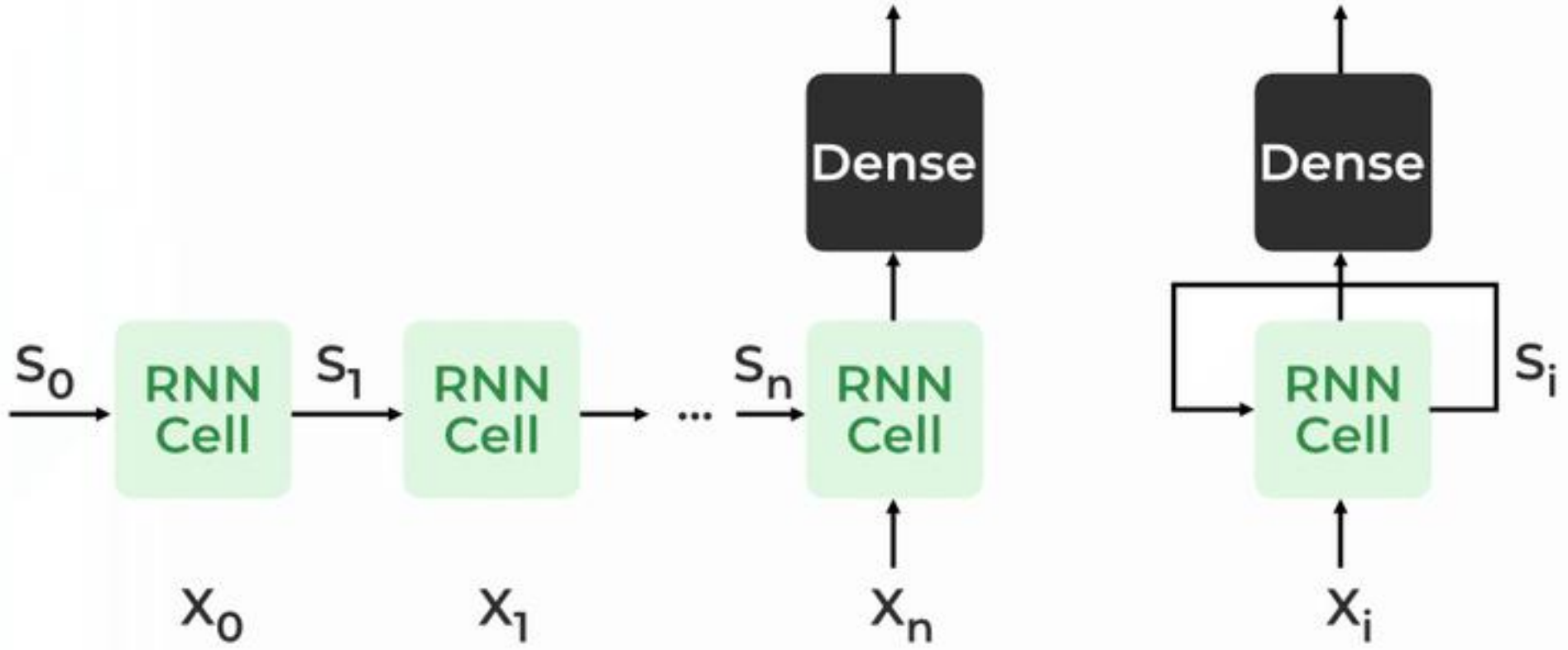
$$Y = f(X, h, W, U, V, B, C)$$

Here S is the State matrix which has element si as the state of the network at timestep i

The parameters in the network are W, U, V, c, b which are shared across timestep

Recurrent Neural Networks

❑Recurrent Neural Network Architecture:



Recurrent Neural Networks

❑How does RNN work?

❑The Recurrent Neural Network consists of multiple fixed activation function units, one for each time step.

❑Each unit has an internal state which is called the hidden state of the unit.

❑This hidden state signifies the past knowledge that the network currently holds at a given time step.

❑This hidden state is updated at every time step to signify the change in the knowledge of the network about the past.

❑The hidden state is updated using the following recurrence relation:

❑**The formula for calculating the current state**: $h_t = f(h_{t-1}, x_t)$

where,

- h_t -> current state
- h_{t-1} -> previous state
- x_t -> input state

Recurrent Neural Networks

- How does RNN work?
- Formula for applying Activation function(tanh)

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

where,

- w_{hh} -> weight at recurrent neuron
- w_{xh} -> weight at input neuron

- The formula for calculating output:

$$y_t = W_{hy}h_t$$

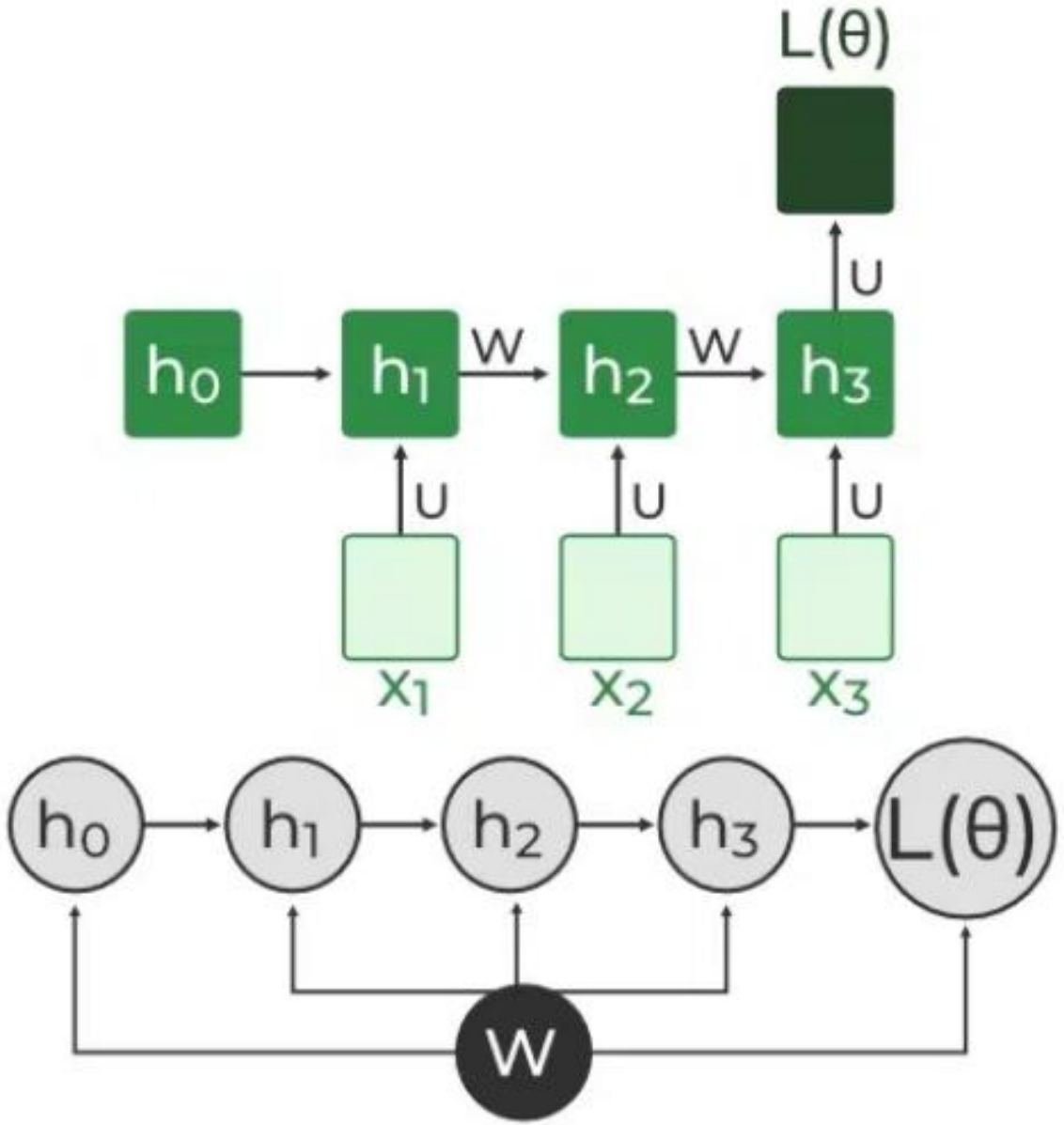
- Y_t -> output
- W_{hy} -> weight at output layer

- These parameters are updated using Backpropagation.
- However, since RNN works on sequential data here we use an updated backpropagation which is known as Backpropagation through time.

❑Backpropagation Through Time (BPTT)

❑In RNN the neural network is in an ordered fashion and since in the ordered network each variable is computed one at a time in a specified order like first h1 then h2 then h3 so on.

❑Hence we will apply backpropagation throughout all these hidden time states sequentially.



❑Backpropagation Through Time (BPTT)

- L(θ)(loss function) depends on h3
- h3 in turn depends on h2 and W
- h2 in turn depends on h1 and W
- h1 in turn depends on h0 and W
- where h0 is a constant starting state.

$$\frac{\partial \mathbf{L}(\theta)}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathbf{L}(\theta)}{\partial W}$$

For simplicity of this equation, we will apply backpropagation on only one row

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \frac{\partial h_3}{\partial W}$$

We already know how to compute this one as it is the same as any simple deep neural network backpropagation.

$$\frac{\partial L(\theta)}{\partial h_3}$$

❑Backpropagation Through Time (BPTT)

.However, we will see how to apply backpropagation to this term $\frac{\partial h_3}{\partial W}$

```
As we know h3 = σ(Wh2 + b)
```

And In such an ordered network, we can't compute $\frac{\partial h_3}{\partial W}$ by simply treating h3 as a constant because as it also depends on W. the total derivative $\frac{\partial h_3}{\partial W}$ has two parts:

1. **Explicit:** $\frac{\partial h_3^+}{\partial W}$ treating all other inputs as constant
2. **Implicit:** Summing over all indirect paths from h_3 to W

Let us see how to do this

$$\begin{aligned}\frac{\partial h_3}{\partial W} &= \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} \\ &= \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2^+}{\partial W} + \frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W} \\ &= \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1^+}{\partial W}\end{aligned}$$

❑Backpropagation Through Time (BPTT)

For simplicity, we will short-circuit some of the paths

$$\frac{\partial h_3}{\partial W} = \frac{\partial h_3^+}{\partial W} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2^+}{\partial W} + \frac{\partial h_3}{\partial h_1}\frac{\partial h_1^+}{\partial W}$$

Finally, we have

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \cdot \frac{\partial h_3}{\partial W}$$

Where

$$\frac{\partial h_3}{\partial W} = \sum_{k=1}^{3} \frac{\partial h_3}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$

Hence,

$$\frac{\partial L(\theta)}{\partial W} = \frac{\partial L(\theta)}{\partial h_3} \sum_{k=1}^{3} \frac{\partial h_3}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$

This algorithm is called backpropagation through time (BPTT) as we backpropagate over all previous time steps

Recurrent Neural Networks

❑Issues of Standard RNNs

❑**Vanishing Gradient:** Text generation, machine translation, and stock market prediction are just a few examples of the time-dependent and sequential data problems that can be modelled with recurrent neural networks. You will discover, though, that the gradient problem makes training RNN difficult.

❑**Exploding Gradient:** An Exploding Gradient occurs when a neural network is being trained and the slope tends to grow exponentially rather than decay. Large error gradients that build up during training lead to very large updates to the neural network model weights, which is the source of this issue.

Recurrent Neural Networks

❑Training through RNN

1. A single-time step of the input is provided to the network.
2. Then calculate its current state using a set of current input and the previous state.
3. The current ht becomes ht-1 for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.
5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained using Backpropagation through time.

❑Advantages and Disadvantages of Recurrent Neural Network

❑Advantages

1. An RNN remembers each and every piece of information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.
2. Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighborhood.

❑Disadvantages

1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or relu as an activation function.

Recurrent Neural Networks

❑Applications of Recurrent Neural Network

1. Language Modelling and Generating Text
2. Speech Recognition
3. Machine Translation
4. Image Recognition
5. Face detection
6. Time series Forecasting

Recurrent Neural Networks

❑**Variation Of Recurrent Neural Network (RNN):** To overcome the problems like vanishing gradient and exploding gradient descent several new advanced versions of RNNs are formed some of these are as;

❑**Bidirectional Neural Network (BiNN):** A BiNN is a variation of a Recurrent Neural Network in which the input information flows in both direction and then the output of both direction are combined to produce the input. BiNN is useful in situations when the context of the input is more important such as NLP tasks and Time-series analysis problems.

❑**Long Short-Term Memory (LSTM):** Long Short-Term Memory works on the read-write-and-forget principle where given the input information network reads and writes the most useful information from the data and it forgets about the information which is not important in predicting the output. For doing this three new gates are introduced in the RNN. In this way, only the selected information is passed through the network.

Bidirectional Recurrent Neural Networks

- Recurrent Neural Networks (RNNs) are a particular class of neural networks that was created with the express purpose of processing sequential input, including speech, text, and time series data.
- RNNs process data as a sequence of vectors rather than feedforward neural networks, which process data as a fixed-length vector.
- Each vector is processed depending on the hidden state from the previous phase.
- The network can store data from earlier steps in the sequence in a type of memory by computing the hidden state by taking into account both the current input and the hidden state from the previous phase.
- RNNs are thus well suited for jobs that call for knowledge of the context and connections among sequence elements.

Bidirectional Recurrent Neural Networks

- Even though conventional RNNs can handle variable-length sequences, they sometimes have trouble with the vanishing gradient problem.
- Gradients during backpropagation become extremely small at this point, making it challenging for the network to learn from the data.
- Many RNN versions, such LSTMs, and GRUs, which use gating methods to regulate the flow of information and enhance learning, have been created to address this problem.

Bidirectional Recurrent Neural Networks

❑Bi-directional Recurrent Neural Network

❑An architecture of a neural network called a bidirectional recurrent neural network (BRNN) is made to process sequential data.

❑In order for the network to use information from both the past and future context in its predictions, BRNNs process input sequences in both the forward and backward directions.

❑This is the main distinction between BRNNs and conventional recurrent neural networks.

❑A BRNN has two distinct recurrent hidden layers, one of which processes the input sequence forward and the other of which processes it backward.

❑After that, the results from these hidden layers are collected and input into a prediction-making final layer.

❑Any recurrent neural network cell, such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit, can be used to create the recurrent hidden layers.

Bidirectional Recurrent Neural Networks

❑Bi-directional Recurrent Neural Network

❑The BRNN functions similarly to conventional recurrent neural networks in the forward direction, updating the hidden state depending on the current input and the prior hidden state at each time step.

❑The backward hidden layer, on the other hand, analyses the input sequence in the opposite manner, updating the hidden state based on the current input and the hidden state of the next time step.

❑Compared to conventional unidirectional recurrent neural networks, the accuracy of the BRNN is improved since it can process information in both directions and account for both past and future contexts.

❑Because the two hidden layers can complement one another and give the final prediction layer more data, using two distinct hidden layers also offers a type of model regularization.

Bidirectional Recurrent Neural Networks

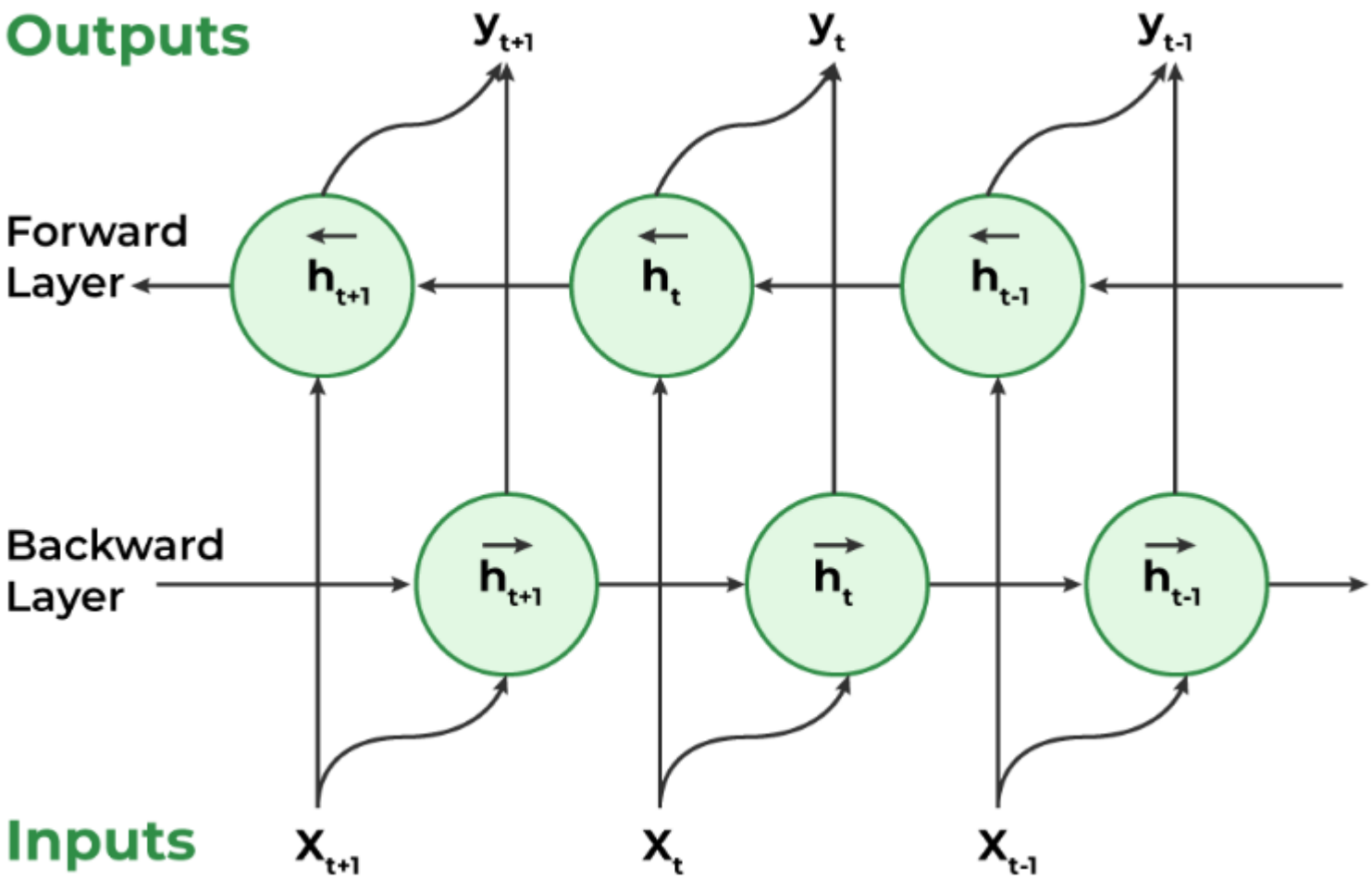
❑Bi-directional Recurrent Neural Network

❑In order to update the model parameters, the gradients are computed for both the forward and backward passes of the backpropagation through the time technique that is typically used to train BRNNs.

❑The input sequence is processed by the BRNN in a single forward pass at inference time, and predictions are made based on the combined outputs of the two hidden layers.

Bidirectional Recurrent Neural Networks

❑Bi-directional Recurrent Neural Network



Bidirectional Recurrent Neural Networks

❑Working of Bidirectional Recurrent Neural Network

1. **Inputting a sequence:** A sequence of data points, each represented as a vector with the same dimensionality, are fed into a BRNN. The sequence might have different lengths.
2. **Dual Processing:** Both the forward and backward directions are used to process the data. On the basis of the input at that step and the hidden state at step t-1, the hidden state at time step t is determined in the forward direction. The input at step t and the hidden state at step t+1 are used to calculate the hidden state at step t in a reverse way.
3. **Computing the hidden state:** A non-linear activation function on the weighted sum of the input and previous hidden state is used to calculate the hidden state at each step. This creates a memory mechanism that enables the network to remember data from earlier steps in the process.

Bidirectional Recurrent Neural Networks

❑Working of Bidirectional Recurrent Neural Network

4. **Determining the output:** A non-linear activation function is used to determine the output at each step from the weighted sum of the hidden state and a number of output weights. This output has two options: it can be the final output or input for another layer in the network.
5. **Training:** The network is trained through a supervised learning approach where the goal is to minimize the discrepancy between the predicted output and the actual output. The network adjusts its weights in the input-to-hidden and hidden-to-output connections during training through backpropagation.

Bidirectional Recurrent Neural Networks

❑**To calculate the output from an RNN unit, we use the following formula:**

H_t (Forward) = A(X_t * W_{XH} (forward) + H_{t-1} (Forward) * W_{HH} (Forward) + b_H (Forward)

H_t (Backward) = A(X_t * W_{XH} (Backward) + H_{t+1} (Backward) * W_{HH} (Backward) + b_H (Backward)

where,

A = activation function,

W = weight matrix

b = bias

❑The hidden state at time t is given by a combination of H_t (Forward) and H_t (Backward). The output at any given hidden state is :

$Y_t = H_t * W_{AY} + b_y$

Bidirectional Recurrent Neural Networks

❑**To calculate the output from an RNN unit, we use the following formula:**

❑The training of a BRNN is similar to backpropagation through a time algorithm. BPTT algorithm works as follows:

1. **Roll out the network and calculate errors at each iteration.**
2. **Update weights and roll up the network.**

❑However, because forward and backward passes in a BRNN occur simultaneously, updating the weights for the two processes may occur at the same time.

❑This produces inaccurate outcomes. Thus, the following approach is used to train a BRNN to accommodate forward and backward passes individually.

Bidirectional Recurrent Neural Networks

❑**Applications of Bidirectional Recurrent Neural Network:** Bi-RNNs have been applied to various natural language processing (NLP) tasks, including:

1. **Sentiment Analysis:** By taking into account both the prior and subsequent context, BRNNs can be utilized to categorize the sentiment of a particular sentence.
2. **Named Entity Recognition:** By considering the context both before and after the stated thing, BRNNs can be utilized to identify those entities in a sentence.
3. **Part-of-Speech Tagging:** The classification of words in a phrase into their corresponding parts of speech, such as nouns, verbs, adjectives, etc., can be done using BRNNs.
4. **Machine Translation:** BRNNs can be used in encoder-decoder models for machine translation, where the decoder creates the target sentence and the encoder analyses the source sentence in both directions to capture its context.
5. **Speech Recognition:** When the input voice signal is processed in both directions to capture the contextual information, BRNNs can be used in automatic speech recognition systems.

Bidirectional Recurrent Neural Networks

❑Advantages of Bidirectional RNN

1. **Context from both past and future:** With the ability to process sequential input both forward and backward, BRNNs provide a thorough grasp of the full context of a sequence. Because of this, BRNNs are effective at tasks like sentiment analysis and speech recognition.
2. **Enhanced accuracy:** BRNNs frequently yield more precise answers since they take both historical and upcoming data into account.
3. **Efficient handling of variable-length sequences:** When compared to conventional RNNs, which require padding to have a constant length, BRNNs are better equipped to handle variable-length sequences.
4. **Resilience to noise and irrelevant information:** BRNNs may be resistant to noise and irrelevant data that are present in the data. This is so because both the forward and backward paths offer useful information that supports the predictions made by the network.

Bidirectional Recurrent Neural Networks

❑Disadvantages of Bidirectional RNN

1. **Computational complexity:** Given that they analyze data both forward and backward, BRNNs can be computationally expensive due to the increased amount of calculations needed.
2. **Long training time**: BRNNs can also take a while to train because there are many parameters to optimize, especially when using huge datasets.
3. **Difficulty in parallelization:** Due to the requirement for sequential processing in both the forward and backward directions, BRNNs can be challenging to parallelize.
4. **Overfitting:** BRNNs are prone to overfitting since they include many parameters that might result in too complicated models, especially when trained on short datasets.
5. **Interpretability:** Due to the processing of data in both forward and backward directions, BRNNs can be tricky to interpret since it can be difficult to comprehend what the model is doing and how it is producing predictions.

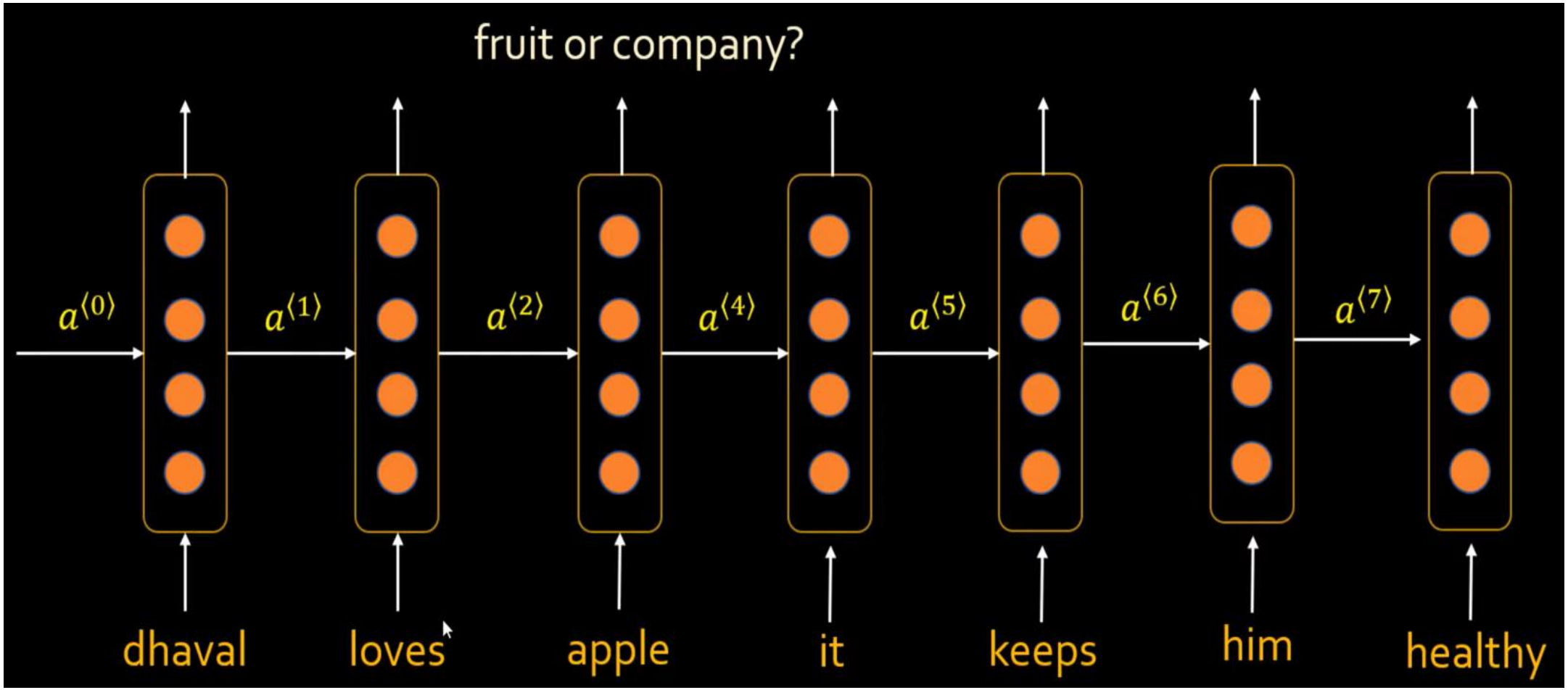
Bidirectional Recurrent Neural Networks

- **Bidirectional RNN Example:**
- **Sentence: "Dhaval loves Apple"**
- **What's Prediction?**

Bidirectional Recurrent Neural Networks

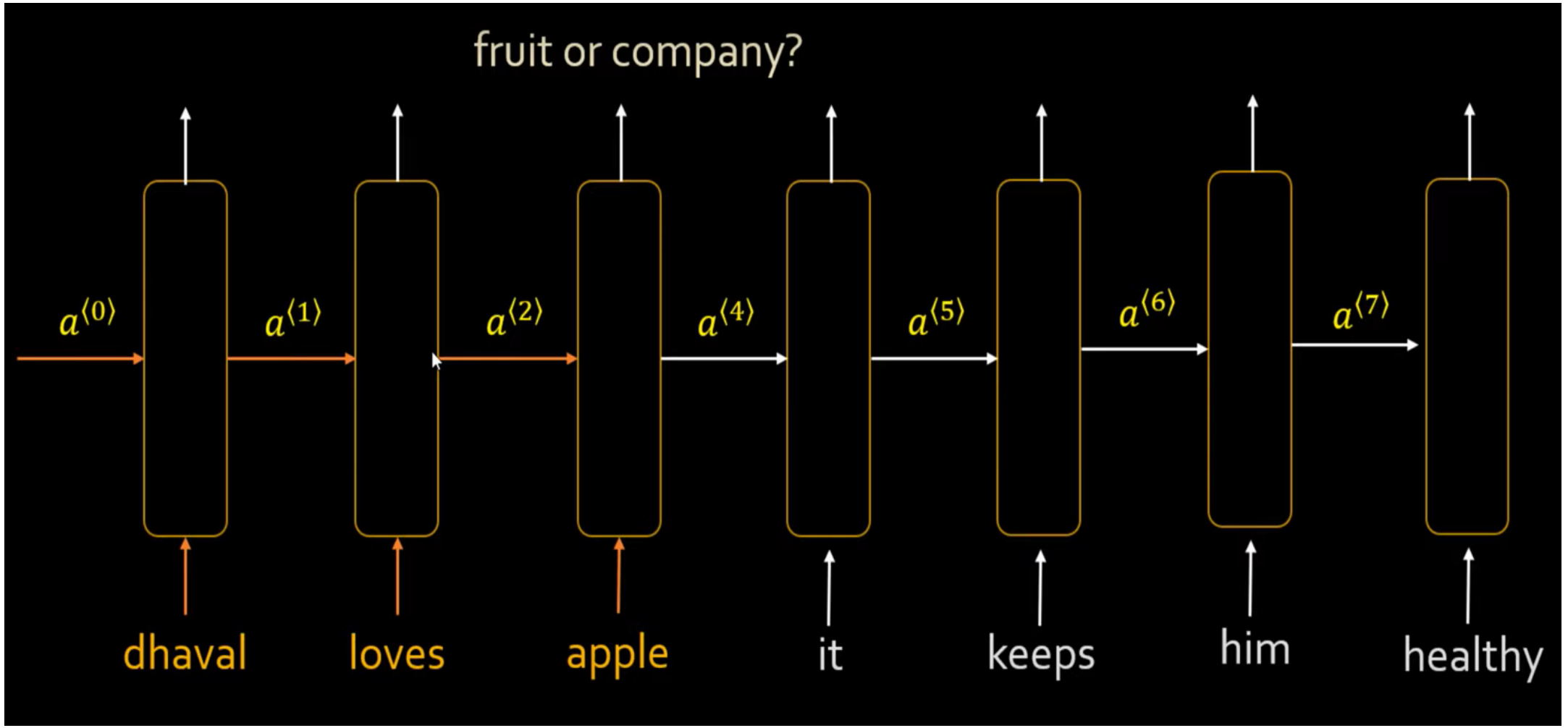
❑ Bidirectional RNN Example:

❑Unidirectional:



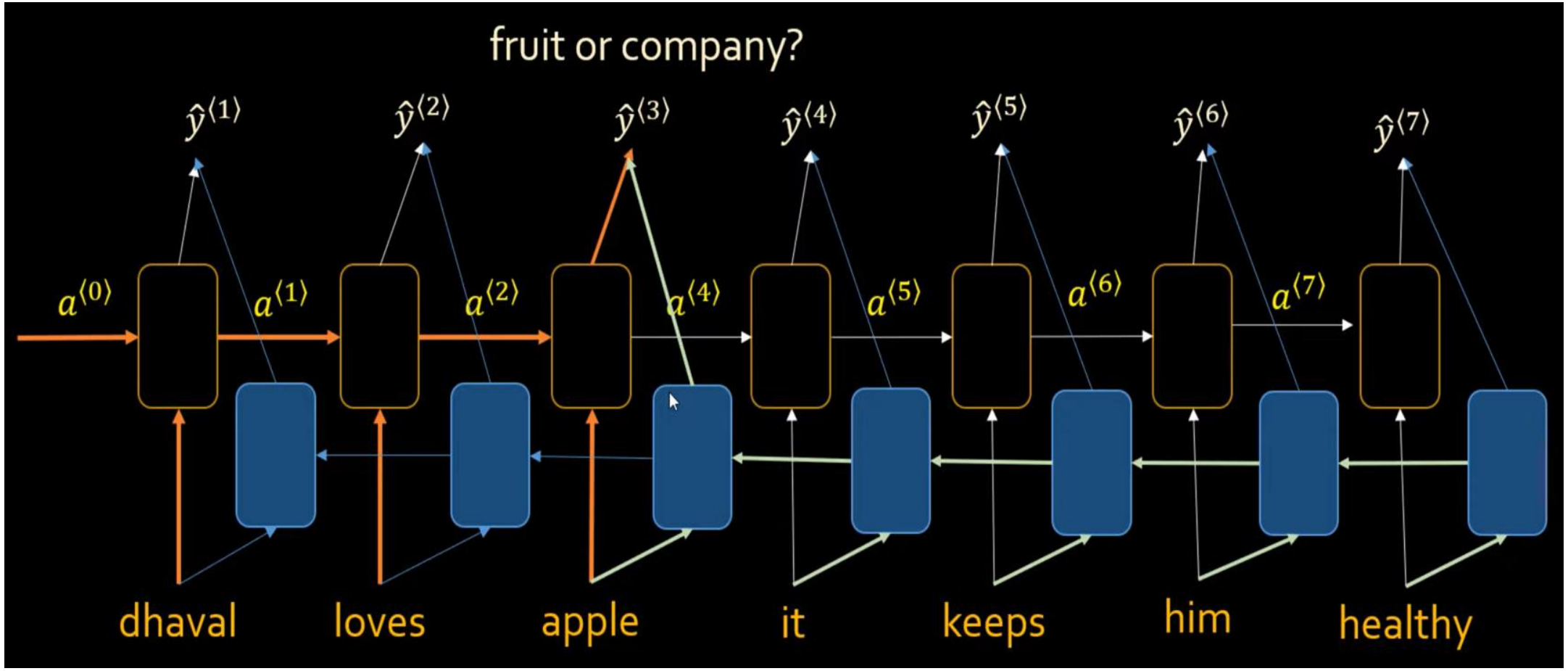
Bidirectional Recurrent Neural Networks

- **Bidirectional RNN Example:**
- **Bidirectional:**



Bidirectional Recurrent Neural Networks

- **Bidirectional RNN Example:**
- **Bidirectional:**



AutoEncoders

- **Autoencoders** emerge as a fascinating subset of neural networks, offering a unique approach to unsupervised learning.
- Autoencoders are an adaptable and strong class of architectures for the dynamic field of deep learning, where neural networks develop constantly to identify complicated patterns and representations.
- With their ability to learn effective representations of data, these unsupervised learning models have received considerable attention and are useful in a wide variety of areas, from image processing to anomaly detection.

AutoEncoders

❑What are Autoencoders?

❑Autoencoders are a specialized class of algorithms that can learn efficient representations of input data with no need for labels.

❑It is a class of artificial neural networks designed for unsupervised learning.

❑Learning to compress and effectively represent input data without specific labels is the essential principle of an automatic decoder.

❑This is accomplished using a two-fold structure that consists of an encoder and a decoder.

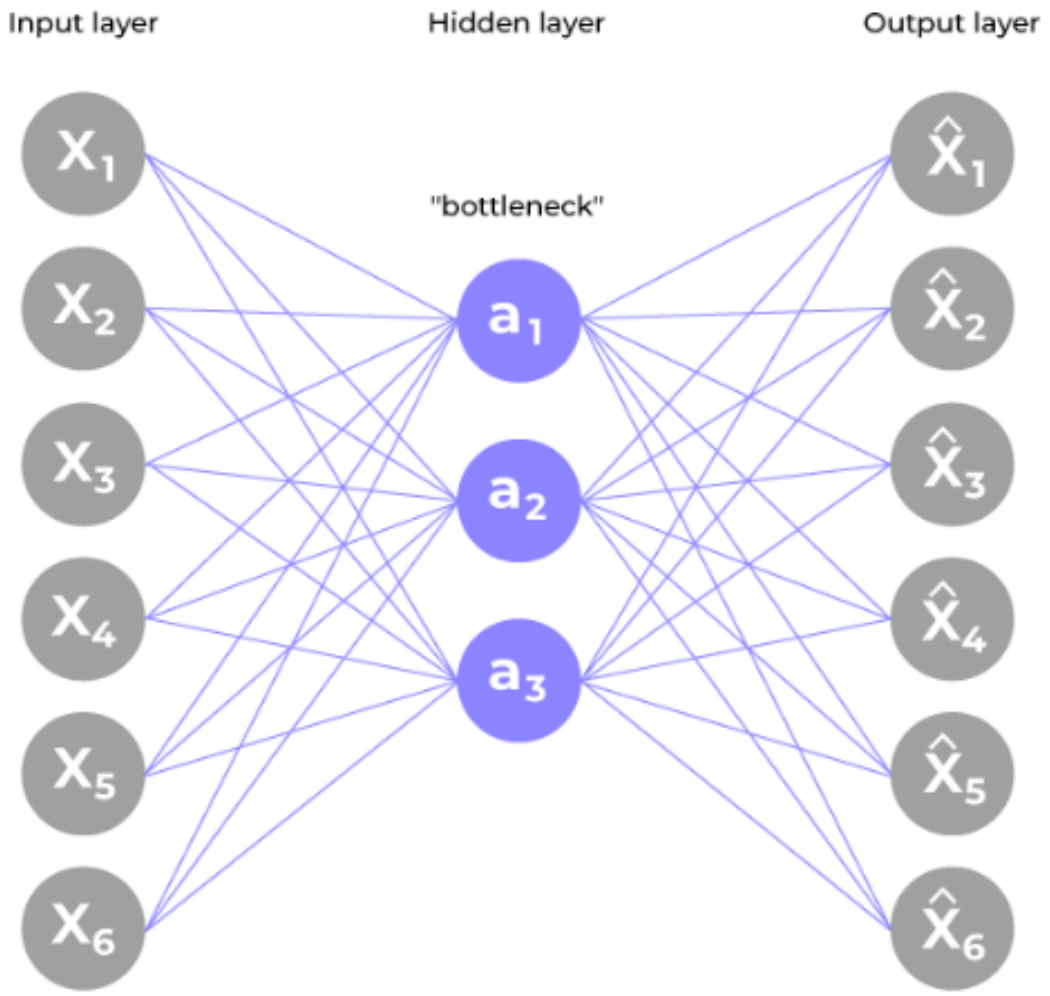
❑The encoder transforms the input data into a reduced-dimensional representation, which is often referred to as “latent space” or “encoding”.

❑From that representation, a decoder rebuilds the initial input.

❑For the network to gain meaningful patterns in data, a process of encoding and decoding facilitates the definition of essential features.

❑Architecture of Autoencoder in Deep Learning

❑The general architecture of an autoencoder includes an encoder, decoder, and bottleneck layer.



❑Architecture of Autoencoder in Deep Learning

1. Encoder

- Input layer take raw input data
- The hidden layers progressively reduce the dimensionality of the input, capturing important features and patterns. These layer compose the encoder.
- The bottleneck layer (latent space) is the final hidden layer, where the dimensionality is significantly reduced. This layer represents the compressed encoding of the input data.

2. Decoder

- The bottleneck layer takes the encoded representation and expands it back to the dimensionality of the original input.
- The hidden layers progressively increase the dimensionality and aim to reconstruct the original input.
- The output layer produces the reconstructed output, which ideally should be as close as possible to the input data.

❑Architecture of Autoencoder in Deep Learning

3. The loss function used during training is typically a reconstruction loss, measuring the difference between the input and the reconstructed output. Common choices include mean squared error (MSE) for continuous data or binary cross-entropy for binary data.
4. During training, the autoencoder learns to minimize the reconstruction loss, forcing the network to capture the most important features of the input data in the bottleneck layer.

AutoEncoders

❑Architecture of Autoencoder in Deep Learning

❑After the training process, only the encoder part of the autoencoder is retained to encode a similar type of data used in the training process.

❑The different ways to constrain the network are: –

1. **Keep small Hidden Layers:** If the size of each hidden layer is kept as small as possible, then the network will be forced to pick up only the representative features of the data thus encoding the data.
2. **Regularization:** In this method, a loss term is added to the cost function which encourages the network to train in ways other than copying the input.
3. **Denoising:** Another way of constraining the network is to add noise to the input and teach the network how to remove the noise from the data.
4. **Tuning the Activation Functions:** This method involves changing the activation functions of various nodes so that a majority of the nodes are dormant thus, effectively reducing the size of the hidden layers.

AutoEncoders

❑Types of Autoencoders

❑There are diverse types of autoencoders and analyze the advantages and disadvantages associated with different variation:

❑**Denoising Autoencoder:** Denoising autoencoder works on a partially corrupted input and trains to recover the original undistorted image.

❑As mentioned above, this method is an effective way to constrain the network from simply copying the input and thus learn the underlying structure and important features of the data.

❑**Advantages**

1. This type of autoencoder can extract important features and reduce the noise or the useless features.
2. Denoising autoencoders can be used as a form of data augmentation, the restored images can be used as augmented data thus generating additional training samples.

❑**Disadvantages**

1. Selecting the right type and level of noise to introduce can be challenging and may require domain knowledge.
2. Denoising process can result into loss of some information that is needed from the original input. This loss can impact accuracy of the output.

AutoEncoders

❑Types of Autoencoders

❑Sparse Autoencoder

❑This type of autoencoder typically contains more hidden units than the input but only a few are allowed to be active at once.

❑This property is called the sparsity of the network. The sparsity of the network can be controlled by either manually zeroing the required hidden units, tuning the activation functions or by adding a loss term to the cost function.

❑Advantages

1. The sparsity constraint in sparse autoencoders helps in filtering out noise and irrelevant features during the encoding process.
2. These autoencoders often learn important and meaningful features due to their emphasis on sparse activations.

❑Disadvantages

1. The choice of hyperparameters play a significant role in the performance of this autoencoder. Different inputs should result in the activation of different nodes of the network.
2. The application of sparsity constraint increases computational complexity.

AutoEncoders

❑Types of Autoencoders

❑Variational Autoencoder

❑Variational autoencoder makes strong assumptions about the distribution of latent variables and uses the **Stochastic Gradient Variational Bayes** estimator in the training process. It assumes that the data is generated by a **Directed Graphical Model** and tries to learn an approximation to to the conditional property where and are the parameters of the encoder and the decoder respectively.

❑Advantages

1. Variational Autoencoders are used to generate new data points that resemble the original training data. These samples are learned from the latent space.
2. Variational Autoencoder is probabilistic framework that is used to learn a compressed representation of the data that captures its underlying structure and variations, so it is useful in detecting anomalies and data exploration.

❑Disadvantages

1. Variational Autoencoder use approximations to estimate the true distribution of the latent variables. This approximation introduces some level of error, which can affect the quality of generated samples.
2. The generated samples may only cover a limited subset of the true data distribution. This can result in a lack of diversity in generated samples.

AutoEncoders

❑Types of Autoencoders

❑Convolutional Autoencoder

❑Convolutional autoencoders are a type of autoencoder that use convolutional neural networks (CNNs) as their building blocks.

❑The encoder consists of multiple layers that take a image or a grid as input and pass it through different convolution layers thus forming a compressed representation of the input.

❑The decoder is the mirror image of the encoder it deconvolves the compressed representation and tries to reconstruct the original image.

❑Advantages

1. Convolutional autoencoder can compress high-dimensional image data into a lower-dimensional data. This improves storage efficiency and transmission of image data.
2. Convolutional autoencoder can reconstruct missing parts of an image. It can also handle images with slight variations in object position or orientation.

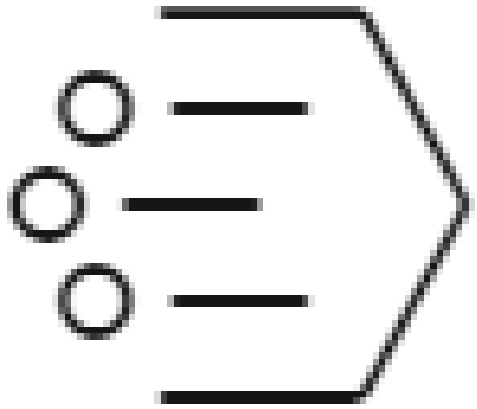
❑Disadvantages

1. These autoencoder are prone to overfitting. Proper regularization techniques should be used to tackle this issue.
2. Compression of data can cause data loss which can result in reconstruction of a lower quality image.

Encoders – Decoder

❑**Encoder:** The **encoder** comprises layers that encode a compressed representation of the input data through dimensionality reduction.

❑In a typical autoencoder, the hidden layers of the neural network contain a progressively smaller number of nodes than the input layer: as data traverses the encoder layers, it is compressed by the process of “squeezing” itself into fewer dimensions.



Encoders – Decoder

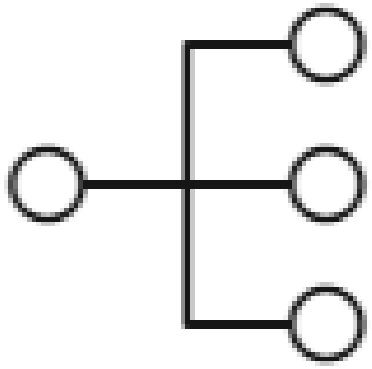
- **Bottleneck:** The bottleneck (or “code”) contains the most compressed representation of the input: it is both the output layer of the encoder network and the input layer of the decoder network.
- A fundamental goal of the design and training of an autoencoder is discovering the minimum number of important features (or dimensions) needed for effective reconstruction of the input data.
- The latent space representation–that is, the code–emerging from this layer is then fed into the decoder.

Encoders – Decoder

❑**Decoder:** The **decoder** comprises hidden layers with a progressively larger number of nodes that decompress (or decode) the encoded representation of data, ultimately reconstructing the data back to its original, pre-encoding form.

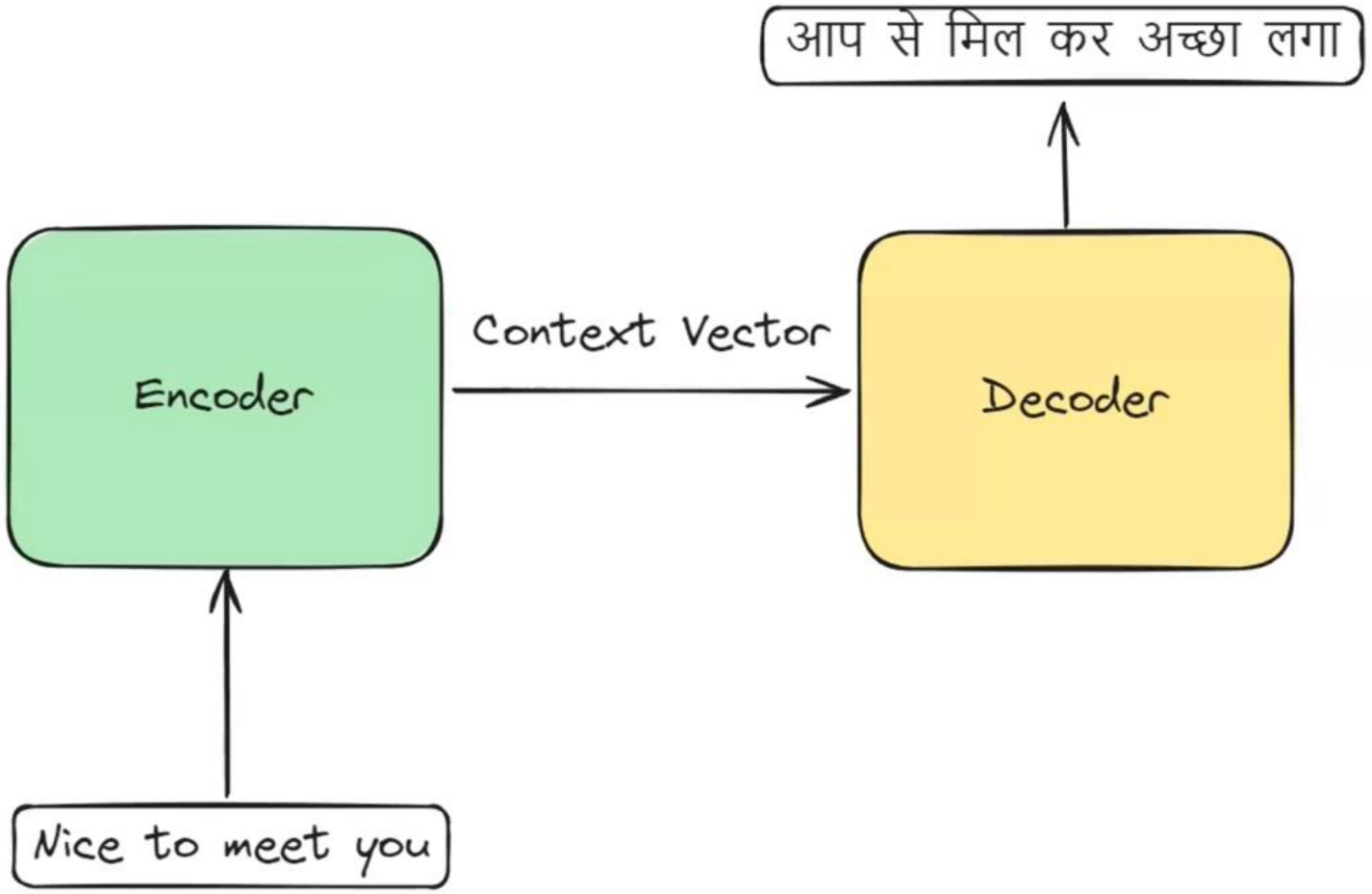
❑This reconstructed output is then compared to the “ground truth”–which in most cases is simply the original input—to gauge the efficacy of the autoencoder.

❑The difference between the output and ground truth is called the reconstruction error.



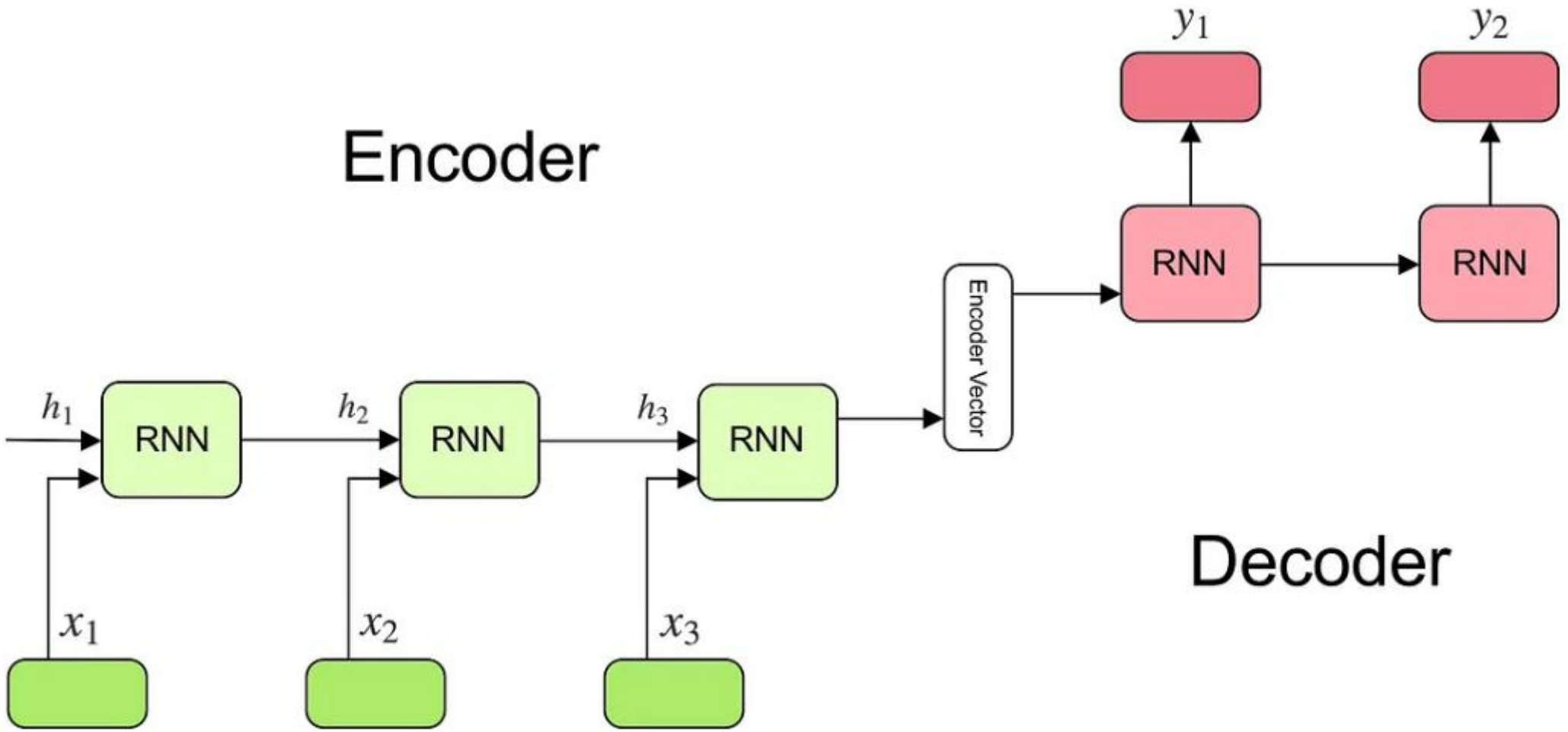
Encoders – Decoder

❑Encoder-Decoder:



Encoders – Decoder

❑Encoder-Decoder:



Encoder-decoder sequence to sequence model

Sequence-to-Sequence (Seq2Seq)Architectures

- Seq2Seq model or Sequence-to-Sequence model, is a machine learning architecture designed for tasks involving sequential data.
- It takes an input sequence, processes it, and generates an output sequence.
- The architecture consists of two fundamental components: an encoder and a decoder.
- Seq2Seq models have significantly improved the quality of machine translation systems making them an important technology.

Sequence-to-Sequence (Seq2Seq)Architectures

❑What is Seq2Seq model?

❑The seq2Seq model is a kind of machine learning model that takes sequential data as input and generates also sequential data as output.

❑Before the arrival of Seq2Seq models, the machine translation systems relied on statistical methods and phrase-based approaches.

❑The most popular approach was the use of phrase-based statistical machine translation (SMT) systems.

❑That was not able to handle long-distance dependencies and capture global context.

Sequence-to-Sequence (Seq2Seq)Architectures

❑What is Seq2Seq model?

❑Seq2Seq models addressed the issues by leveraging the power of neural networks, especially recurrent neural networks (RNN).

❑The concept of seq2seq model was introduced in the paper titled “Sequence to Sequence Learning with Neural Networks” by Google.

❑The architecture discussed in this research paper is fundamental framework for natural language processing tasks.

❑The seq2seq models are encoder-decoder models. The encoder processes the input sequence and transforms it into a fixed-size hidden representation.

❑The decoder uses the hidden representation to generate output sequence.

❑The encoder-decoder structure allows them to handle input and output sequences of different lengths, making them capable to handle sequential data. Seq2Seq models are trained using a dataset of input-output pairs, where the input is a sequence of tokens, and the output is also a sequence of tokens.

❑The model is trained to maximize the likelihood of the correct output sequence given the input sequence.

Sequence-to-Sequence (Seq2Seq)Architectures

❑What is Seq2Seq model?

❑The advancement in neural networks architectures led to the development of more capable seq2seq model named transformers.

❑“Attention is all you need! ” was a research paper that first introduced the transformer model in the era of Deep Learning after which language-related models have taken a huge leap.

❑The main idea behind the transformers model was that of attention layers and different encoder and decoder stacks which were highly efficient to perform language-related tasks.

❑Seq2Seq models have been widely used in NLP tasks due to their ability to handle variable-length input and output sequences.

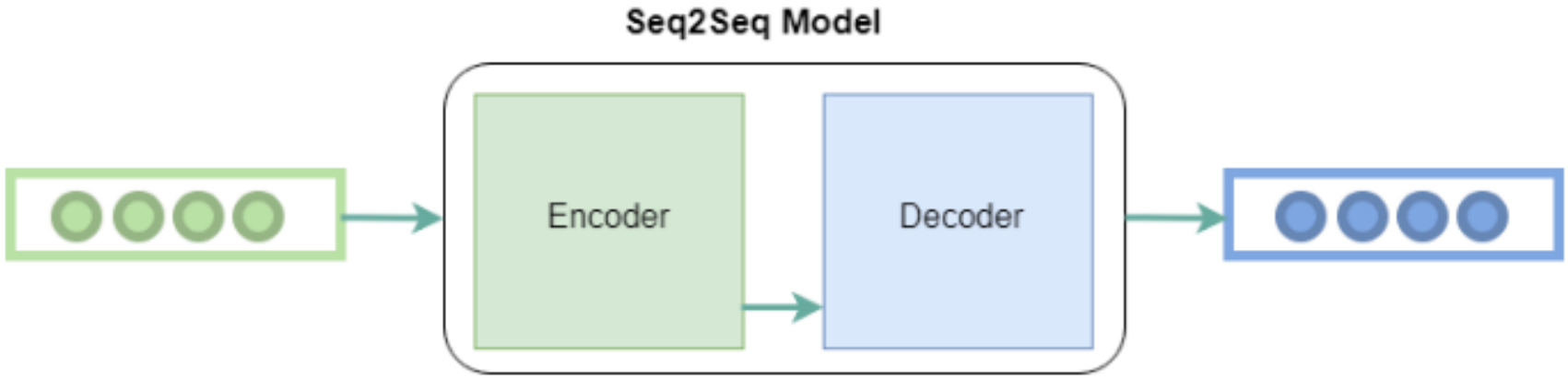
❑Additionally, the attention mechanism is often used in Seq2Seq models to improve performance and it allows the decoder to focus on specific parts of the input sequence when generating the output.

Sequence-to-Sequence (Seq2Seq)Architectures

❑What is Encoder and Decoder in Seq2Seq model?

❑In the seq2seq model, the Encoder and the Decoder architecture plays a vital role in converting input sequences into output sequences.

❑ Let's explore about each block:



Encoder and Decoder Stack in seq2seq model

Sequence-to-Sequence (Seq2Seq)Architectures

❑What is Encoder and Decoder in Seq2Seq model?

❑**Encoder Block:** The main purpose of the encoder block is to process the input sequence and capture information in a fixed-size context vector.

❑**Architecture:**

- The input sequence is put into the encoder.
- The encoder processes each element of the input sequence using neural networks (or transformer architecture).
- Throughout this process, the encoder keeps an internal state, and the ultimate hidden state functions as the context vector that encapsulates a compressed representation of the entire input sequence. This context vector captures the semantic meaning and important information of the input sequence.

❑**The final hidden state of the encoder is then passed as the context vector to the decoder.**

Sequence-to-Sequence (Seq2Seq)Architectures

❑What is Encoder and Decoder in Seq2Seq model?

❑**Decoder Block:** The decoder block is similar to encoder block. The decoder processes the context vector from encoder to generate output sequence incrementally.

❑**Architecture:**

- In the training phase, the decoder receives both the context vector and the desired target output sequence (ground truth).
- During inference, the decoder relies on its own previously generated outputs as inputs for subsequent steps.

Sequence-to-Sequence (Seq2Seq)Architectures

❑What is Encoder and Decoder in Seq2Seq model?

❑The decoder uses the context vector to comprehend the input sequence and create the corresponding output sequence.

❑It engages in autoregressive generation, producing individual elements sequentially.

❑At each time step, the decoder uses the current hidden state, the context vector, and the previous output token to generate a probability distribution over the possible next tokens.

❑The token with the highest probability is then chosen as the output, and the process continues until the end of the output sequence is reached.

Sequence-to-Sequence (Seq2Seq)Architectures

❑RNN based Seq2Seq Model:

The decoder and encoder architecture utilizes RNNs to generate desired outputs. Let's look at the simplest seq2seq model.

For a given sequence of inputs $(x_1, x_2, ..., x_T)$, a RNN generates a sequence of outputs $(y_1, y_2, ..., y_T)$ through iterative computation based on the following equation:

$$h_t = \sigma(W^{hx}x_t + W^{hh}h_{t-1})$$

$$y_t = W^{yh}h_t$$

Here,

- h_t represents hidden state at time step t
- x_t represents input at time step t
- W^{hx} represents the weight matrix for the input
- h_{t-1} represents hidden state from the previous time step (t-1)
- σ represents the sigmoid activation function.
- y_t represents output at time step t
- W^{yh} represents the wight matrix for the output
- T is the length of the sequence.

Sequence-to-Sequence (Seq2Seq)Architectures

❑RNN based Seq2Seq Model:

❑Recurrent Neural Networks can easily map sequences to sequences when the alignment between the inputs and the outputs are known in advance.

❑Although the vanilla version of RNN is rarely used, its more advanced version i.e. LSTM or GRU is used.

❑This is because RNN suffers from the problem of vanishing gradient. LSTM develops the context of the word by taking 2 inputs at each point in time.

❑One from the user and the other from its previous output, hence the name recurrent (output goes as input).

Sequence-to-Sequence (Seq2Seq)Architectures

❑Advantages of seq2seq Models

1. **Flexibility:** Seq2Seq models can handle a wide range of tasks such as machine translation, text summarization, and image captioning, as well as variable-length input and output sequences.
2. **Handling Sequential Data:** Seq2Seq models are well-suited for tasks that involve sequential data such as natural language, speech, and time series data.
3. **Handling Context:** The encoder-decoder architecture of Seq2Seq models allows the model to capture the context of the input sequence and use it to generate the output sequence.
4. **Attention Mechanism:** Using attention mechanisms allows the model to focus on specific parts of the input sequence when generating the output, which can improve performance for long input sequences.

Sequence-to-Sequence (Seq2Seq)Architectures

❑Disadvantages of seq2seq Models

1. **Computationally Expensive:** Seq2Seq models require significant computational resources to train and can be difficult to optimize.
2. **Limited Interpretability:** The internal workings of Seq2Seq models can be difficult to interpret, which can make it challenging to understand why the model is making certain decisions.
3. **Overfitting:** Seq2Seq models can overfit the training data if they are not properly regularized, which can lead to poor performance on new data.
4. **Handling Rare Words:** Seq2Seq models can have difficulty handling rare words that are not present in the training data.
5. **Handling Long input Sequences:** Seq2Seq models can have difficulty handling input sequences that are very long, as the context vector may not be able to capture all the information in the input sequence.

Sequence-to-Sequence (Seq2Seq)Architectures

❑Applications of Seq2Seq model

1. **Text Summarization:** The seq2seq model effectively understands the input text which makes it suitable for news and document summarization.
2. **Speech Recognition:** Seq2Seq model, especially those with attention mechanisms, excel in processing audio waveform for ASR. They are able to capture spoken language patterns effectively.
3. **Image Captioning:** The seq2seq model integrate image features from CNNs with textual generation capabilities for image captioning. They are capable to describe images in a human readable format.

Deep Recurrent Networks

- Speech Recognition is the identification of the text in speech by computers.
- Speech, as we perceive it, is sequential in nature.
- If you are to model a speech recognition problem in deep learning, which model do you think suits the task best?
- Yet, unanticipatedly, when a speech recognition RNN model was fit onto the data, the results were not as promising.
- Deep feedforward neural networks generated better accuracy in comparison to a typical RNN.
- After analyzing why the RNNs failed, researchers proposed a possible solution to attain greater accuracy: by introducing depth into the network, similar to how a deep feed-forward neural network is composed.

Deep Recurrent Networks

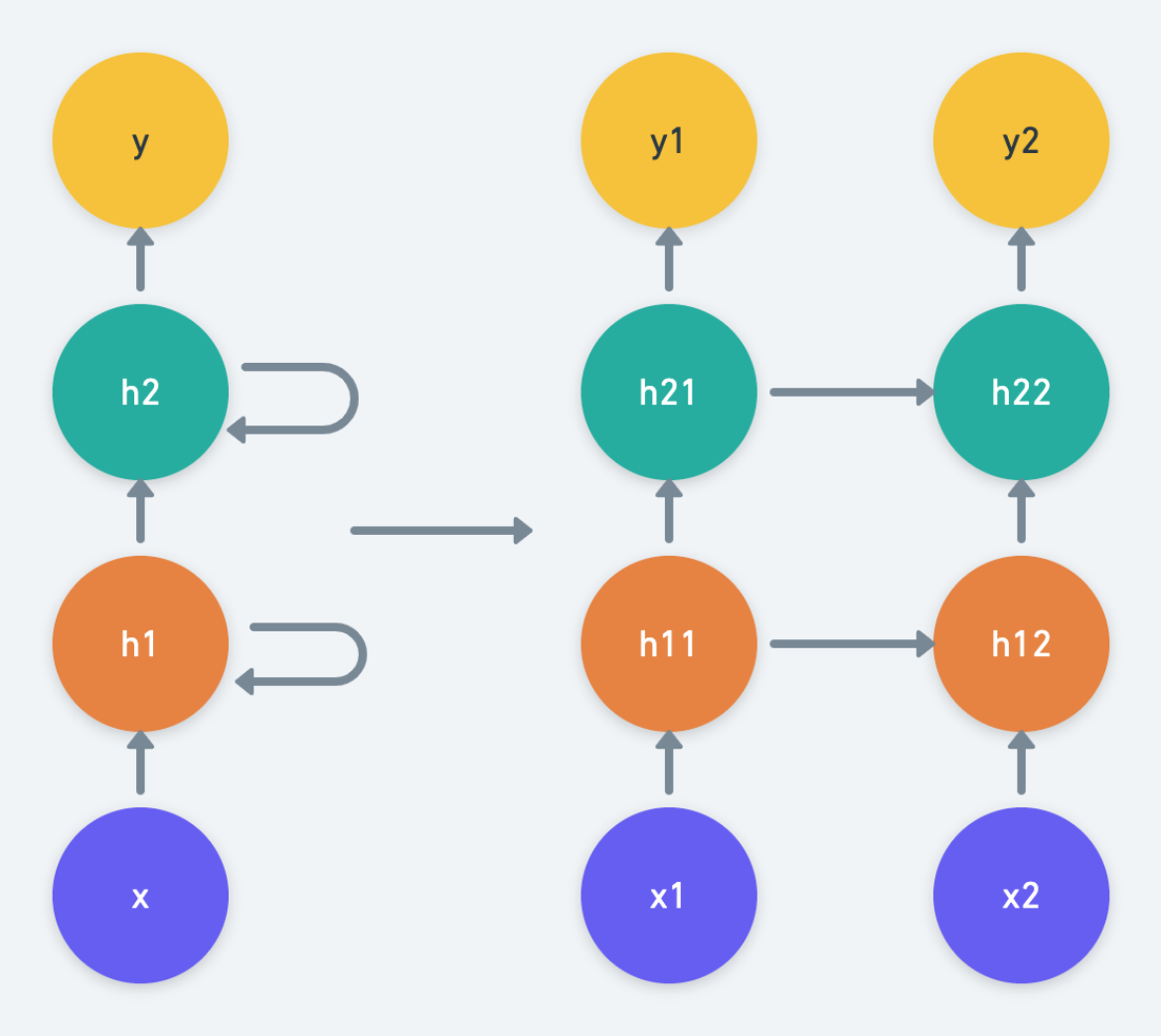
❑Introducing Depth into the Network: Deep RNNs

- ❑An RNN is deep with respect to time.
- ❑But what if it's deep with respect to space as well, as in a feed forward network?
- ❑This is the fundamental notion that has inspired researchers to explore Deep Recurrent Neural Networks, or Deep RNNs.
- ❑In a typical deep RNN, the looping operation is expanded to ***multiple hidden units***.

Deep Recurrent Networks

❑Introducing Depth into the Network: Deep RNNs

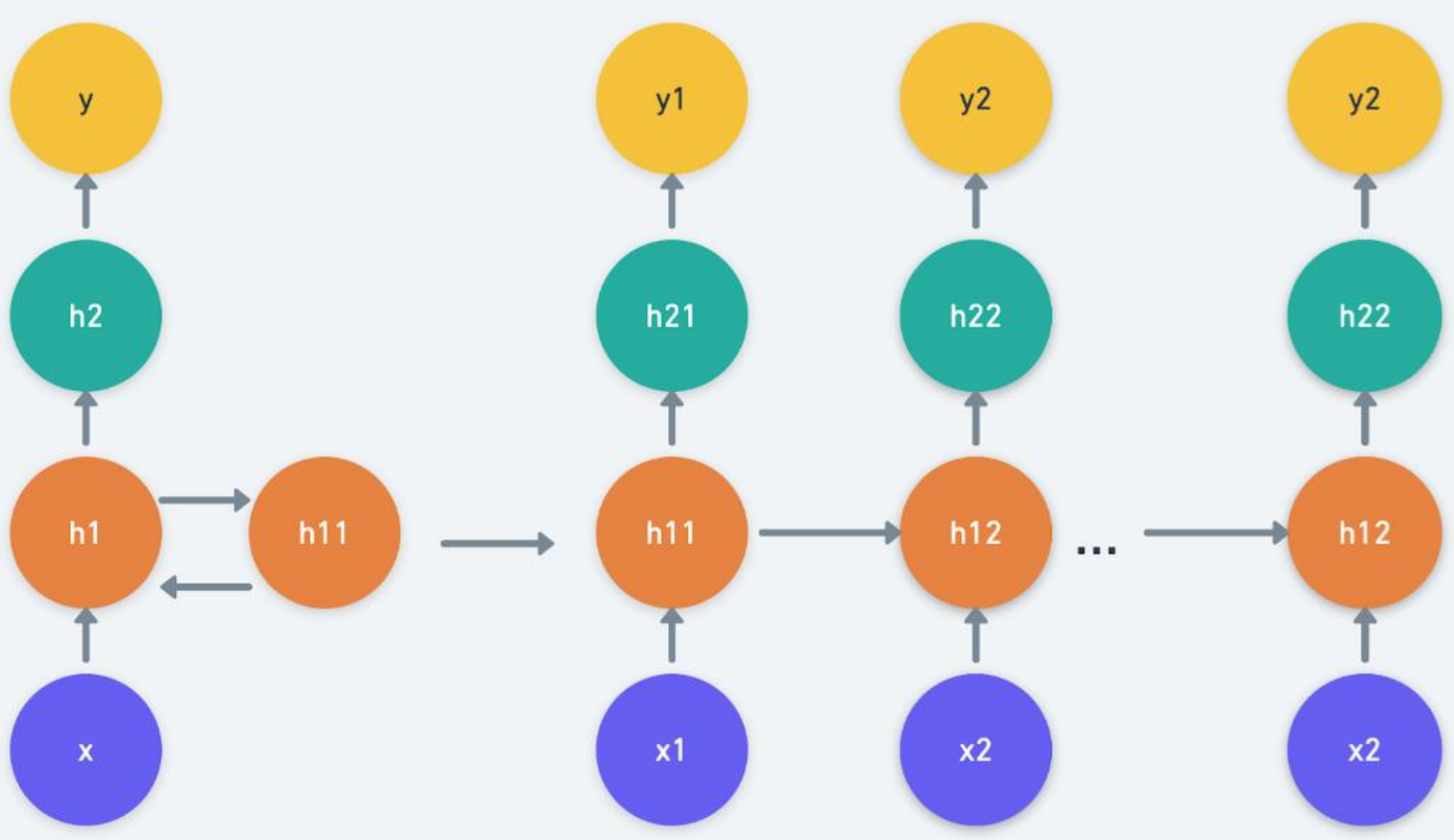
❑2- Layer Deep RNN



Deep Recurrent Networks

❑Introducing Depth into the Network: Deep RNNs

❑Multi-Layer Deep RNN - A Varied Representation



Deep Recurrent Networks

❑Introducing Depth into the Network: Deep RNNs

❑Multi-Layer Deep RNN - A Varied Representation:

❑This model increases the distance traversed by a variable from time t to time t+1.

❑It can have simple RNNs, GRUs, or LSTMs as its hidden units.

❑It helps in modeling varying data representations by capturing the data from all ends, and enables passing multiple/single hidden state(s) to the multiple hidden states of the subsequent layers.

Deep Recurrent Networks

❑Introducing Depth into the Network: Deep RNNs

❑Multi-Layer Deep RNN - A Varied Representation: Mathematical Notation

The hidden state in a deep RNN can be given by the following equation:

$$H_t^l = \phi^l(H_{t-1}^l * W_{hh}^l + H_t^{l-1} * W_{xh}^l + b_h^l)$$

Where ϕ is the activation function, W is the weight matrix, and b is the bias.

The output at a hidden state H_t is given by:

$$O_t = H_t^l * W_{hy} + b_y$$

Deep Recurrent Networks

❑Deep RNNs

❑The training of a deep RNN is similar to the Backpropagation Through Time (BPTT) algorithm, as in an RNN but with additional hidden units.

❑It capture complex relationships between the different pieces of information and make better predictions about what might come next.

❑Deep RNNs are used in many real-life applications, such as speech recognition systems like Siri or Alexa, language translation software, and even self-driving cars.

❑They’re particularly useful in situations where there’s a lot of sequential data to process, like when you’re trying to teach a computer to understand human language.

❑Deep RNNs, with their ability to handle sequential data and capture complex relationships between input and output sequences, have become a powerful tool in various real-life applications, ranging from speech recognition and natural language processing to music generation and autonomous driving

Deep Recurrent Networks

❑Deep RNNs Applications:

1. Speech Recognition: Deep RNNs have been used to build speech recognition systems, such as Google's Speech API, Amazon's Alexa, and Apple's Siri. These systems use deep RNNs to convert speech signals into text.
2. Natural Language Processing (NLP): Deep RNNs are used in various NLP applications, such as language translation, sentiment analysis, and text classification. For example, Google Translate uses a deep RNN to translate text from one language to another.
3. Music Generation: Deep RNNs have been used to generate music, such as Magenta's MusicVAE, which uses a deep RNN to generate melodies and harmonies.
4. Image Captioning: Deep RNNs are used in image captioning systems, such as Google's Show and Tell, which uses a deep RNN to generate captions for images.
5. Autonomous Driving: Deep RNNs have been used in autonomous driving systems to predict the behaviour of other vehicles on the road, such as the work done by Waymo.

Deep Recurrent Networks

❑Steps for Deep RNNs with Sentimental Analysis Example:

1. **Data preparation:** The first step is to gather and preprocess the data. In this case, we'll need a dataset of text reviews labelled with positive or negative sentiment. The text data needs to be cleaned, tokenized, and converted to the numerical format. This can be done using libraries like NLTK or spaCy in Python.
2. **Model architecture design:** The next step is to design the deep RNN architecture. We'll need to decide on the number of layers, number of hidden units, and type of recurrent unit (e.g. LSTM or GRU). We'll also need to decide how to handle the input and output sequences, such as using padding or truncation.
3. **Training the model:** Once the architecture is designed, we'll need to train the model using the preprocessed data. We'll split the data into training and validation sets and train the model using an optimization algorithm like stochastic gradient descent. We'll also need to set yperparameters like learning rate and batch size.

Deep Recurrent Networks

❑Steps for Deep RNNs with Sentimental Analysis Example:

4. **Evaluating the model:** After training, we'll evaluate the model's performance on a separate test set. We'll use metrics like accuracy, precision, recall, and F1 score to assess the model's performance.
5. **Deploying the model:** Finally, we'll deploy the trained model to a production environment, where it can be used to classify sentiment in real-time. This could involve integrating the model into a web application or API. Overall, developing an end-to-end deep RNN application requires a combination of technical skills in programming, data preprocessing, and machine learning, as well as an understanding of the specific application domain.

Bidirectional Encoder Representation from Transformers (BERT)

- **BERT:** is a Natural Language Processing Model developed by researchers in Google AI.
- When it was proposed it achieved start-of-the-art accuracy on 11 NLP and NLU tasks including the very competitive Stanford Question Answering Dataset (SQuAD v1.1), GLUE (General Language Understanding Evaluation), SWAG (Situation With Adversarial Generations)…..
- The BERT model was pre-trained using text from Wikipedia(that's 2,500 million words!) and Book Corpus (800 million words) and can be fine-tuned with the question and answer datasets.

Bidirectional Encoder Representation from Transformers (BERT)

❑Why we Moving on to BERT:

❑One of the biggest challenges in the Language Model is the lack of training data.

❑Because NLP is a diversified field with many distinct tasks, most task-specific datasets contain only a few thousand or a few hundred thousand human-labeled training examples.

❑To overcome the above issue, BERT trains a language model on a large unlabelled text corpus (unsupervised or semi-supervised).

❑And we can also be Fine-tuning this large model to specific NLP tasks to utilize the large repository of knowledge this model has gained (supervised)

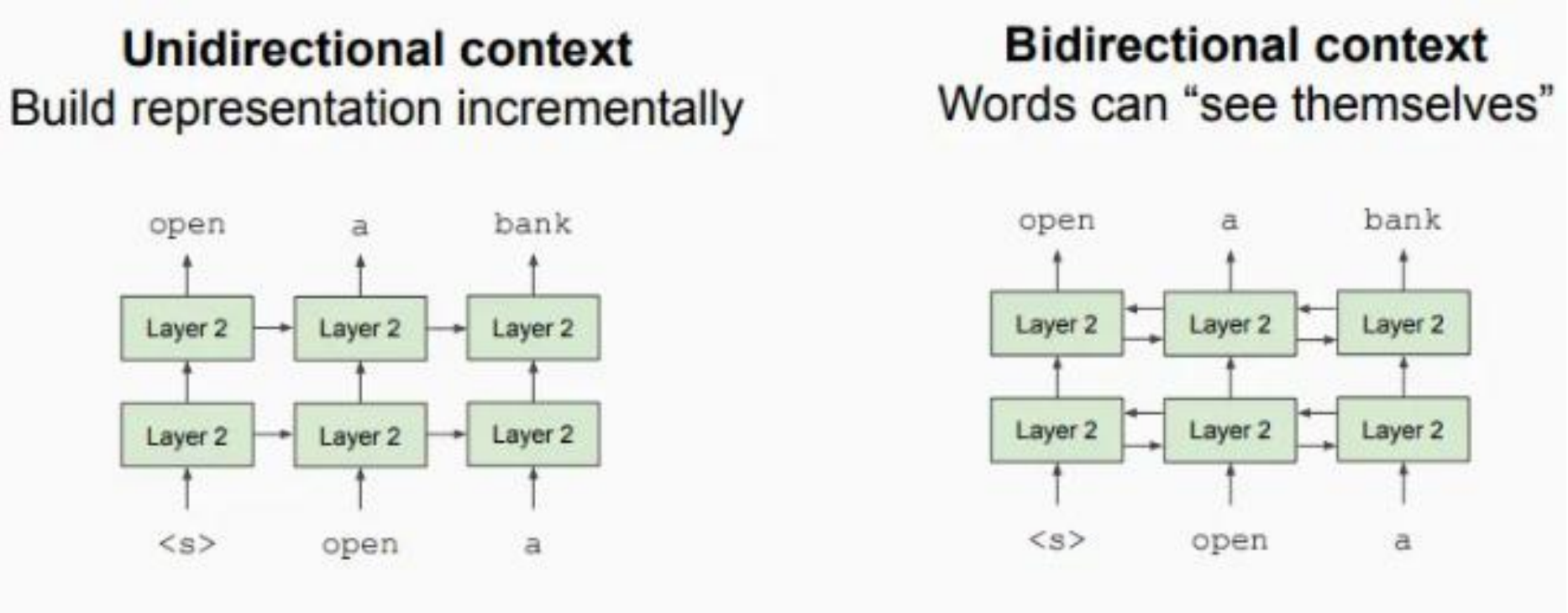
❑Language models could only read text input sequentially — either left-to-right or right-to-left — but couldn't do both at the same time.

❑BERT is different because it is designed to read in both directions at once.

❑This capability, enabled by the introduction of Transformers, is known as bi-directionality.

Bidirectional Encoder Representation from Transformers (BERT)

❑Why we Moving on to BERT:

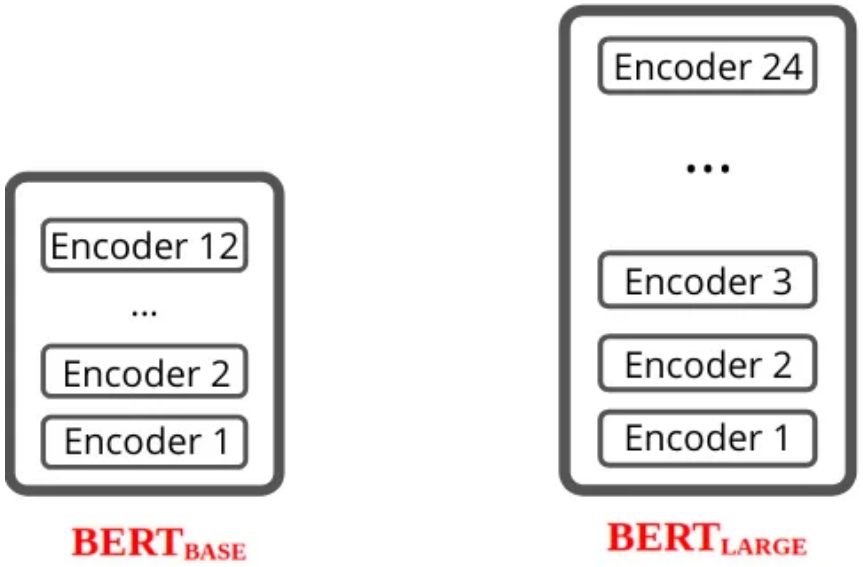


Bidirectional Encoder Representation from Transformers (BERT)

❑**BERT Model Architecture:** BERT is released in two versions of the pre-trained model, which is trained on a huge dataset.

❑BERT also used Many previous NLP algorithms and architectures such that semi-supervised training, OpenAI Transformers, ELMo Embeddings, ULMFit, Transformer.

❑BERT is basically an Encoder stack of transformer architecture.



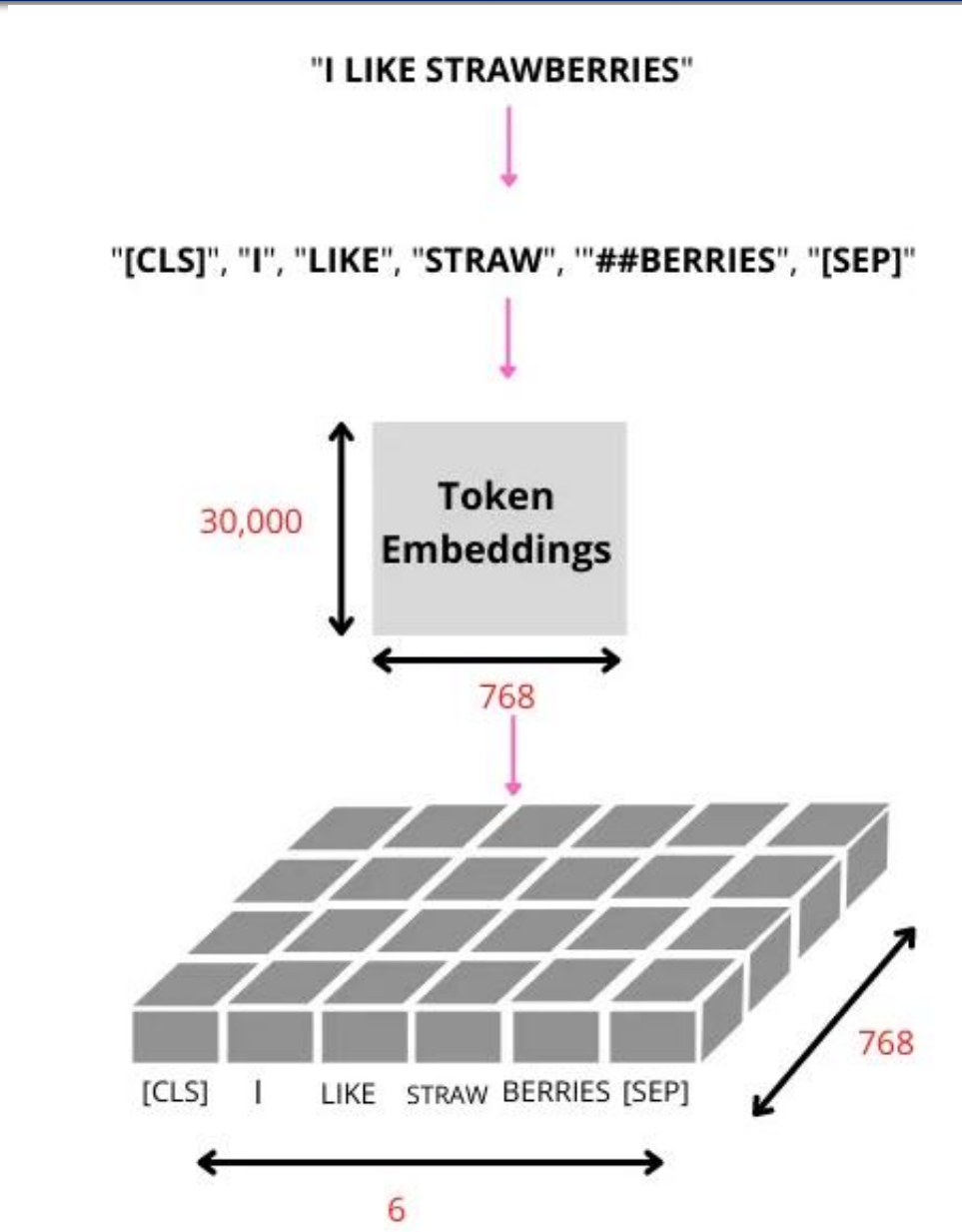
Bidirectional Encoder Representation from Transformers (BERT)

- A transformer architecture is an encoder-decoder network that uses self-attention on the encoder side and attention on the decoder side. An encoder that reads the text input and a decoder that produces a prediction for the task.
- **BERT BASE Model:** It has 12 layers in the Encoder stack, 12 Attention Heads, 110 million parameters, 768 Hidden Sizes
- **BERT LARGE Model:** It has 24 layers in the Encoder stack, 16 Attention Heads, 340 million parameters, 1024 Hidden Sizes

Bidirectional Encoder Representation from Transformers (BERT)

❑Model Input/Output Representation:

The input of the model is unambiguously represented in a single sentence and a pair of sentences (eg: Question and Answer)in one token sequence.

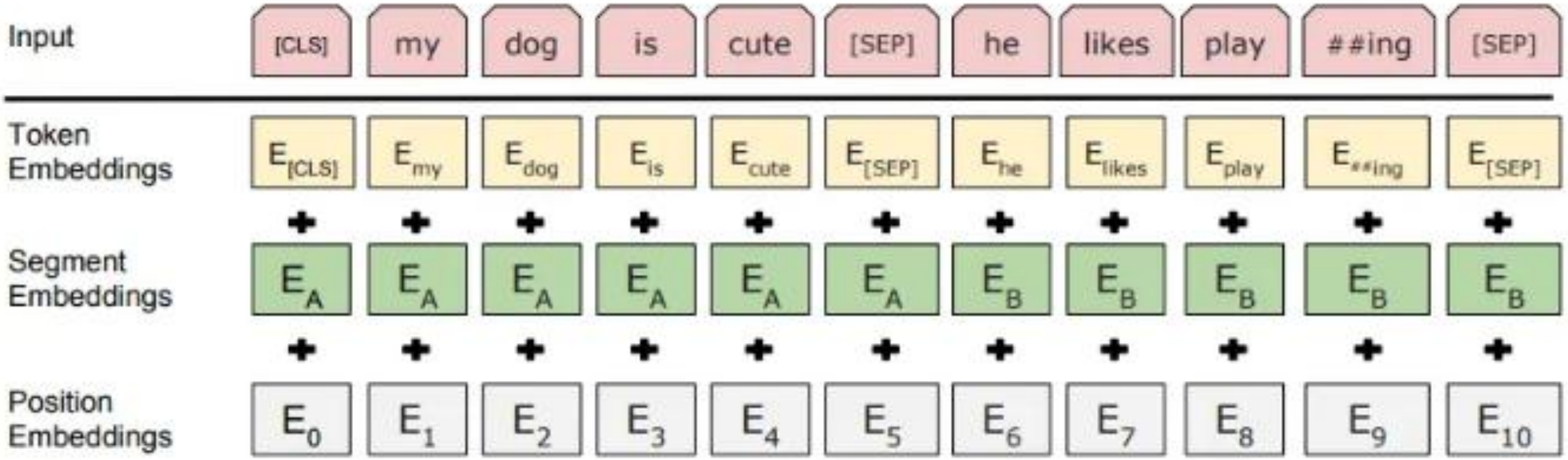


Bidirectional Encoder Representation from Transformers (BERT)

❑BERT uses WordPiece embeddings with a 30,000 token vocabulary.

1. The first token of every sequence of input is always a[CLS], which means the classification token.
2. The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks.
3. If the Sentence pairs are packed together into a single sequence, they will differentiate the sentence in two ways. First, using [SEP] token. Second, add a learned embedding to every token indicating whether it belongs to sentence A or sentence B.

Bidirectional Encoder Representation from Transformers (BERT)



Bidirectional Encoder Representation from Transformers (BERT)

❑How does BERT work?

❑The new model features an important improvement when it comes to **context understanding**.

❑Especially with certain texts which are “context-heavy,” for which context understanding is so important for analytics purposes, BERT is a great solution.

❑Say we have two sentences like:

1. You were **right**.
2. Make a **right** turn at the light.

❑In the first sentence, the word “**right**” refers to a decision; in the second sentence, “**right**” refers to direction. We know that because of the context.

Bidirectional Encoder Representation from Transformers (BERT)

❑**Example 2:**

1. My favorite flower is a **rose**.
2. He quickly **rose** from his seat.

❑In the first sentence, the word "**rose**" refers to a flower; in the second one, it's referred to the past tense of rise.

❑Again, we know that because of the context.

❑Because BERT practices to predict missing words in the text, and because it analyzes every sentence with no specific direction, it does a better job at understanding the meaning of homonyms than previous NLP methodologies, such as embedding methods.

❑To overcome this challenge, BERT is pre-trained on two different NLP tasks:

1. MLM (Masked Language Modeling)
2. Next Sentence Prediction (NSP)

Bidirectional Encoder Representation from Transformers (BERT)

❑MLM (Masked Language Modeling)

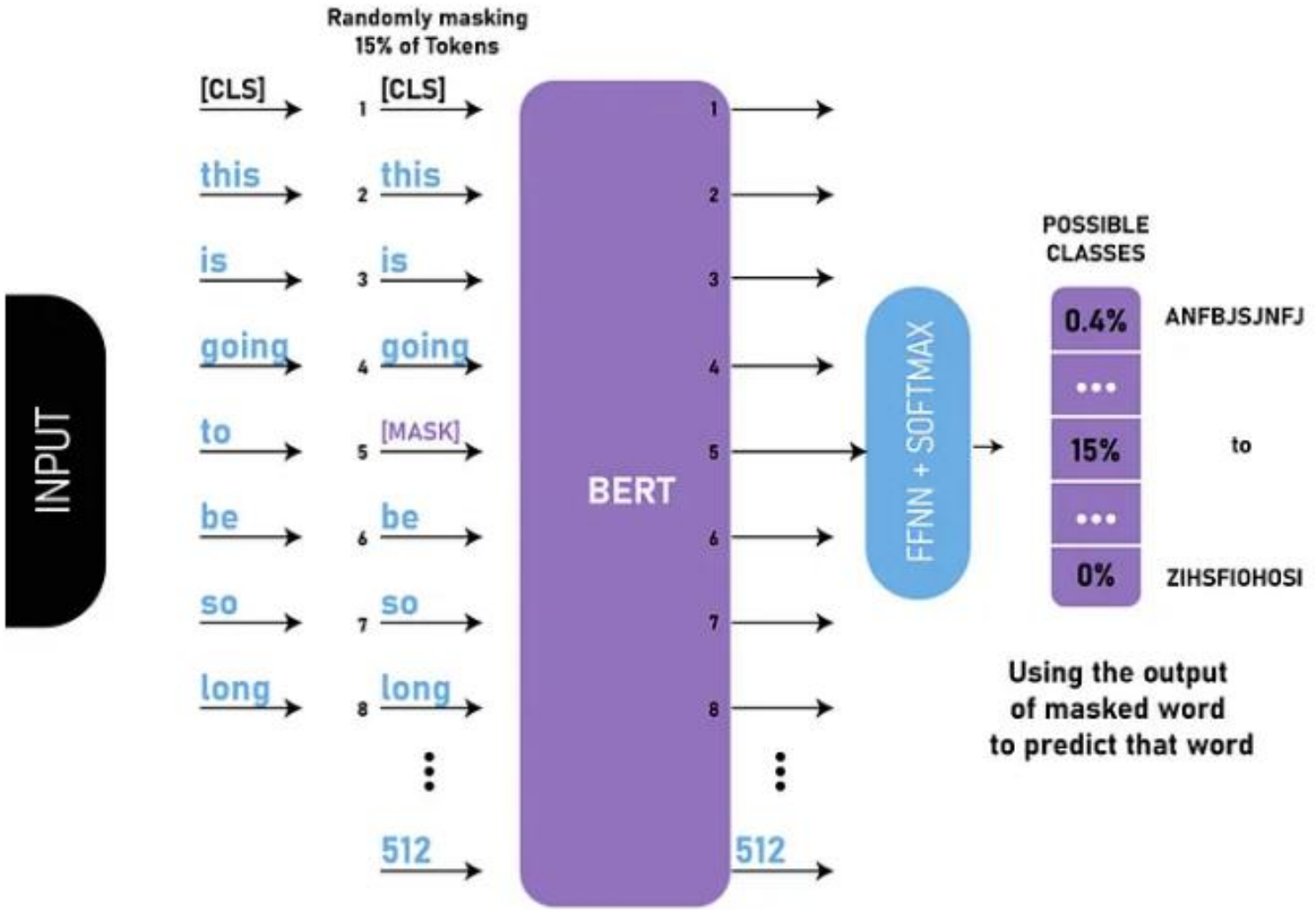
❑Usually, Language Model can only be trained in one specific direction. BERT can handle this issue with **MLM.**

❑Masked Language Modeling(MLM) training is to hide a word in a sentence and have the program predict what word has hidden(masked) based on the context of the hidden words.

❑BERT does this by masking 15% of all WordPiece tokens in each sequence at random.

Bidirectional Encoder Representation from Transformers (BERT)

❑MLM (Masked Language Modeling)



Bidirectional Encoder Representation from Transformers (BERT)

- **MLM (Masked Language Modeling)**
- For every mask input sequence,Randomly select 15% of tokens (not all masked in the same way) and don't replace them with [MASK] 100% of the time
- **Example Sentence: This is going to be so long:** For 80% of the time: Replace the word with the [MASK] token.
- **This is going to be so long → This is going [MASK] be so long:** For 10% of the time: Replace the word with a random word
- **This is going to be so long → This is going the be so long:** For 10% of the time: Keep the word unchanged
- **This is going to be so long → This is going to be so long**

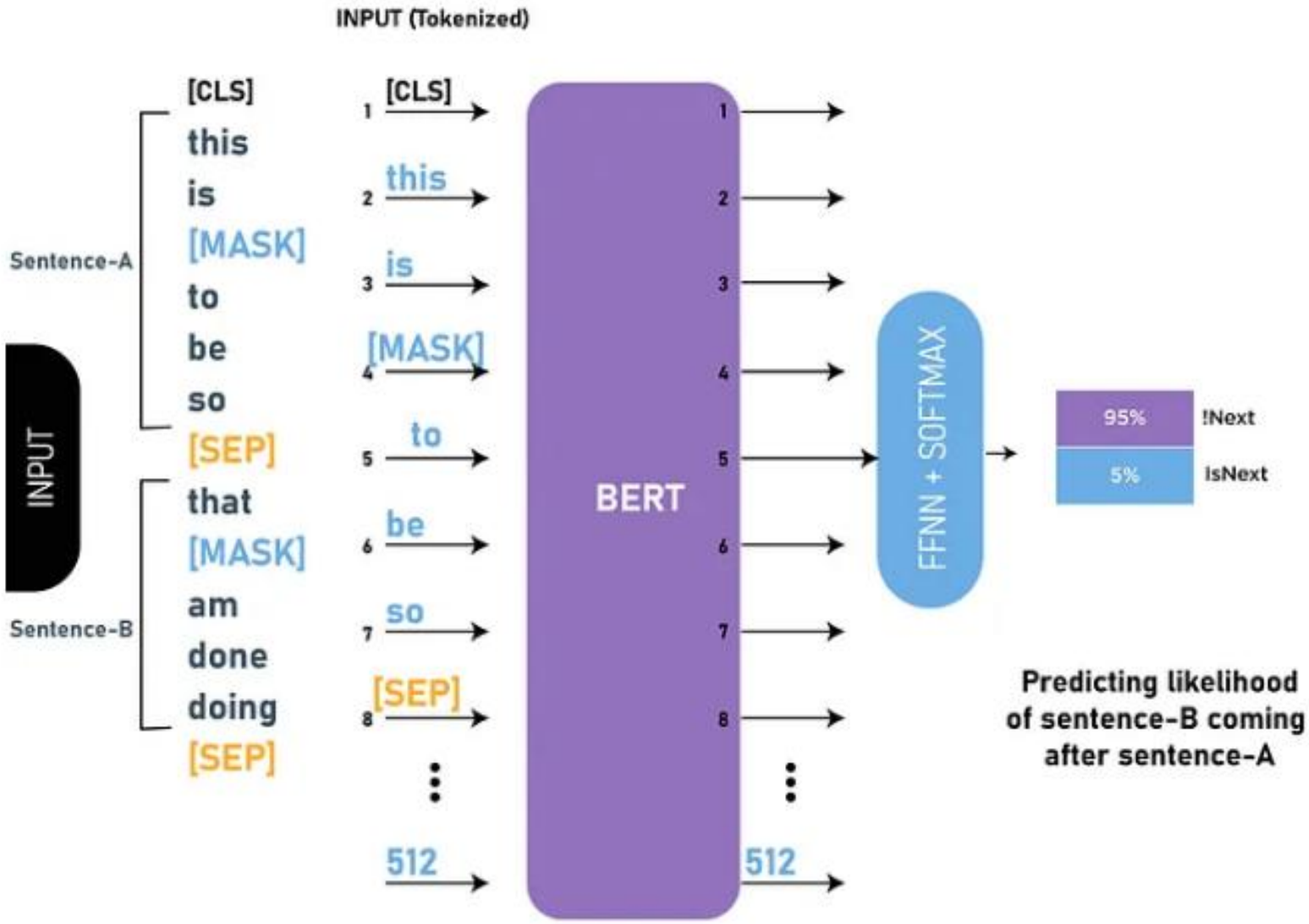
Bidirectional Encoder Representation from Transformers (BERT)

❑Next Sentence Prediction (NSP)

- ❑Most downstream tasks such as Question and Answer(QA) and Natural Language Inference (NLI) are based on understanding the relationship between two sentences.
- ❑It is not directly captured by language modeling.
- ❑In the BERT training process, the model receives a pair of sentences as an input and it will be trained to predict whether the second sentence is subsequent to the first sentence.
- ❑It is Binary Classification, having two labels **IsNext, NotNext.**
- ❑When choosing the sentences A and B for each pre-training example, 50% of the time Sentence B is an actual sentence that follows Sentence A and it is labeled as **IsNext.**
- ❑Next 50% of the time Sentence B will be random sentences from the corpus and it is labeled as **NotNext.**

Bidirectional Encoder Representation from Transformers (BERT)

❑Next Sentence Prediction (NSP)



Bidirectional Encoder Representation from Transformers (BERT)

- **Next Sentence Prediction (NSP)**
- **Example:**
- **Input**: [CLS] the man went to [MASk] store [SEP] he bought a gallon [MASK] milk [SEP]
- **Output**: **IsNext**
- **Another Example:**
- **Input**: [CLS] the man [MASk] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]
- **Output**: **NotNext**

Note for Students

- This power point presentation is for lecture, therefore it is suggested that also utilize the text books and lecture notes.