[Year]

SHRADDHA KUMAR
SDBCT

INDEX

Unit-4

Lecture Notes

Recurrent neural network

Problems where the sequence or order of the events is really important for predicting the next event are usually tackled by Recurrent Neural Networks or RNN.

Speech recognition, language translation, stock market price prediction, music generation etc. all these tasks are performed by RNNs.

The recurrent neural network is a special type of neural network which not just looks at the current input being presented to it but also the previous input. So instead of

Input → Hidden → Output

it becomes

Input + Previous Hidden → Hidden → Output

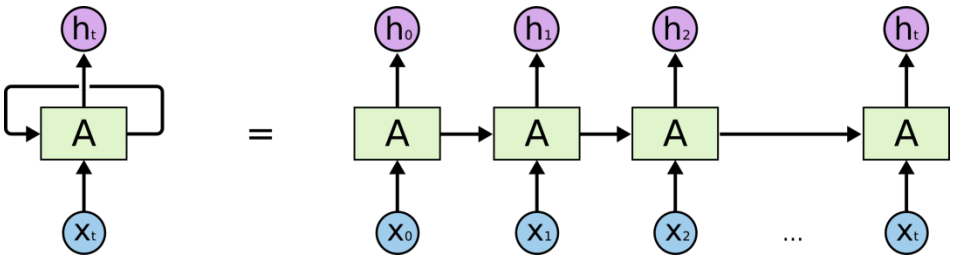


Figure 4.1: Recurrent Neural Network

The right-hand side of the equality sign in the image shows the network for each time step i.e. at t=0 the input X0 goes into the network to produce h_0, the next time step the input is X1 but there is an additional input from the previous time step from the block A.

This way the neural network not only looks at the current input but has the context from the previous inputs as well. As the recurrent units hold the past values, we can refer to this as memory.

RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being dependent on the previous computations.

The current state can be written as $h_t = f(h_{t-1}, x_t)$

Here, ht is the new state, ht-1 is the previous state while xt is the current input. Because the input neuron would have applied the transformations on our previous input. So each successive input is called as a time step.

Let's say that the activation function is tanh, the weight at the recurrent neuron is Whh and the weight at the input neuron is W_{xh}, we can write the equation for the state at time t as –

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

The Recurrent neuron in this case is just taking the immediate previous state into consideration.

For longer sequences the equation can involve multiple such states.

Once the final state is calculated we can go on to produce the output. $y_t = W_{hy}h_t$

Comparison between RNN and a simple feed forward neural network.

A simple feed forward neural network is really good at learning a pattern between a set of inputs and outputs. For example, given an image the network can tell you whether the image contains a dog or not. But if you input another image of, say an elephant followed by an image of a dog it has no context or memory of the dog's image. It will classify the elephant's image independently. This type of mechanism is not suited well to tasks which require previous context for making future predictions.

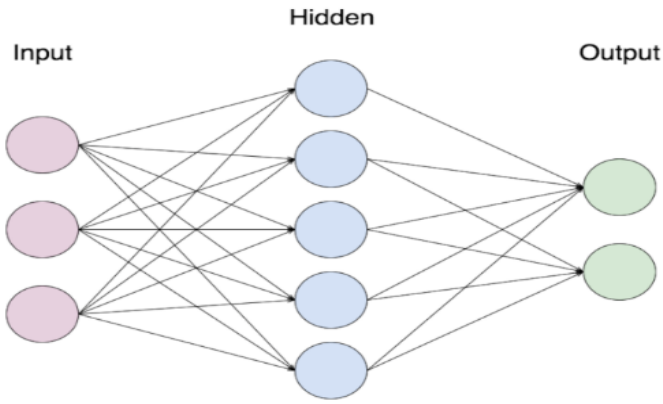


Figure 4.2: Feed Forward Neural Network

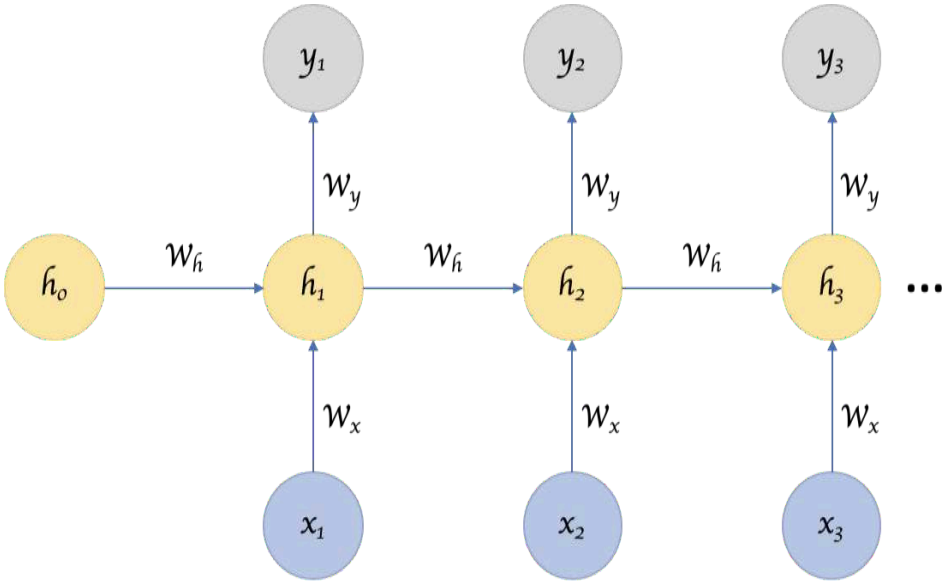


Figure 4.3: Recurrent Neural Network

Training through RNN

1. A single time step of the input is supplied to the network i.e. x_t is supplied to the network
2. We then calculate its current state using a combination of the current input and the previous state i.e. we calculate h_t
3. The current h_t becomes h_{t-1} for the next time step
4. We can go as many time steps as the problem demands and combine 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 y_t
6. The output is then compared to the actual output and the error is generated
7. The error is then backpropagated to the network to update the weights.

Advantages of Recurrent Neural Network

- RNN is an extremely useful type of neural network when it comes to time series prediction, because of the feature of **remembering previous inputs**.
- RNN can **process inputs of any lengths and the model size doesn't increase**. This makes the network **more flexible** than the traditional artificial neural networks.
- In a RNN, weights are shared across time, which results in a **lower computational cost.**

Disadvantages of Recurrent Neural Network

- The occurrence of "**Vanishing Gradient" or "Exploding Gradient".**
- The **computation is a slow process** and the training of RNN can become very difficult.
- It becomes **difficult to access any information** that was given to the network if a long time period has passed.

RNN: Commonly used activation functions

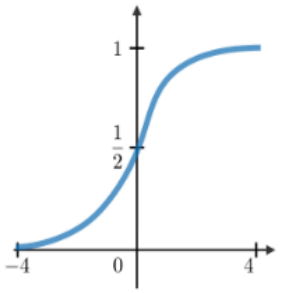
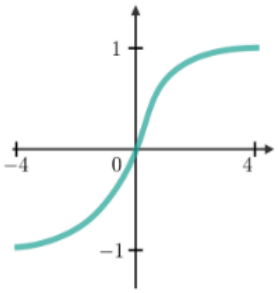
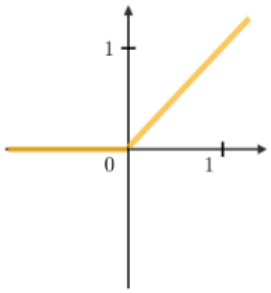
Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$
		

Figure 4.4: Activation functions used in RNN

RNN: Gradient Problem

- **Vanishing/exploding gradient**: The vanishing and exploding gradient phenomena are often encountered in the context of RNNs.
- The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.
- During back propagation, recurrent neural networks suffer from the vanishing gradient problem.
- Gradients are values used to update a neural networks weight.
- The vanishing gradient problem is when the gradient shrinks as it back propagates through time.
- If a gradient value becomes extremely small, it doesn't contribute too much learning

There are two methods to deal with these problems:

- **Gradient clipping:** It is a technique used to cope with the **exploding gradient** problem sometimes encountered when performing backpropagation. By **capping the maximum value** for the gradient, this phenomenon is controlled in practice.
- **Types of Gates** In order to remedy the **vanishing gradient** problem, **specific gates are used** in some types of RNNs and usually have a well-defined purpose.

RNN: Short-Term Memory problem

- RNN's are good for processing sequence data for predictions but suffers from short-term memory.

- If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So, if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.
- So in recurrent neural networks, **layers that get a small gradient update stops learning.** Those are usually the earlier layers. So because these layers don't learn, RNN's can forget what it seen in longer sequences, thus having a short-term memory.

new weight = weight - learning rate*gradient

2.0999 = 2.1 - 0.001

Not much of a difference — update value

How to overcome short-term memory problem

LSTM's (Long Short-Term Memory) and GRU's (Gated Recurrent Unit) were created as a method to mitigate short-term memory, using a mechanism called gates.

> *Both LSTMs and GRUs are designed to address the problem of "vanishing gradients" in RNNs, which occurs when the gradients of the weights in the network become very small and the network has difficulty learning.*

Structure of LSTM:

- LSTM has a chain structure that contains different memory blocks called cells.

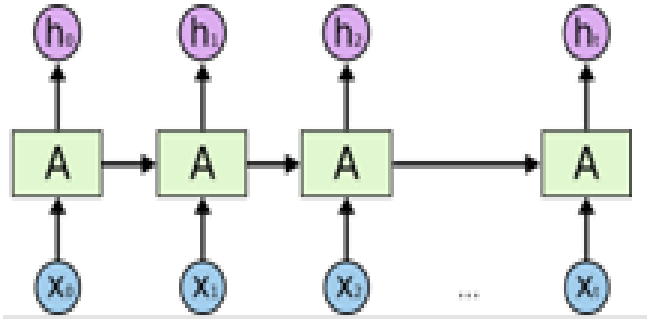


Figure 4.5: Outer Structure of Long Short-Term Memory

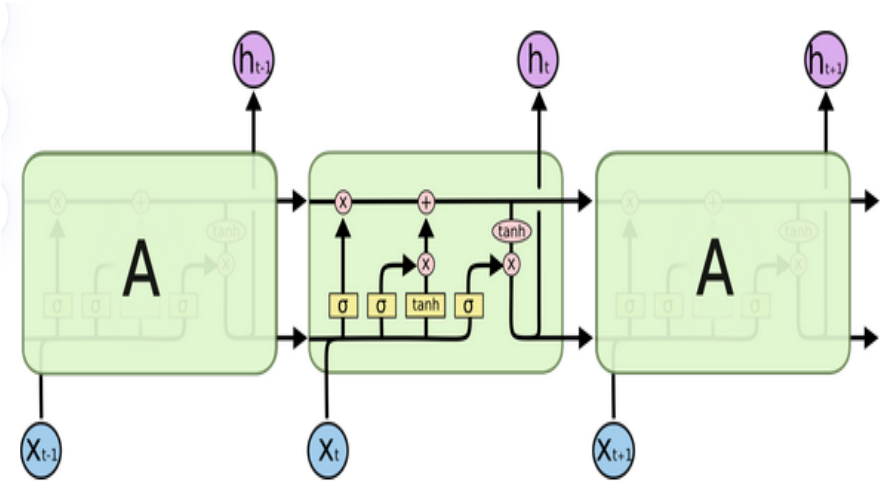


Figure 4.6: Internal Structure of Long Short-Term Memory

- A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate
- The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. *Gates are just neural networks!*
- The gates in an LSTM network are controlled by sigmoid activation functions, which output values between 0 and 1.
- The gates allow the network to selectively store or forget information, depending on the values of the inputs and the previous state of the cell.

Long-short-term memory: Structure of LSTM Cell

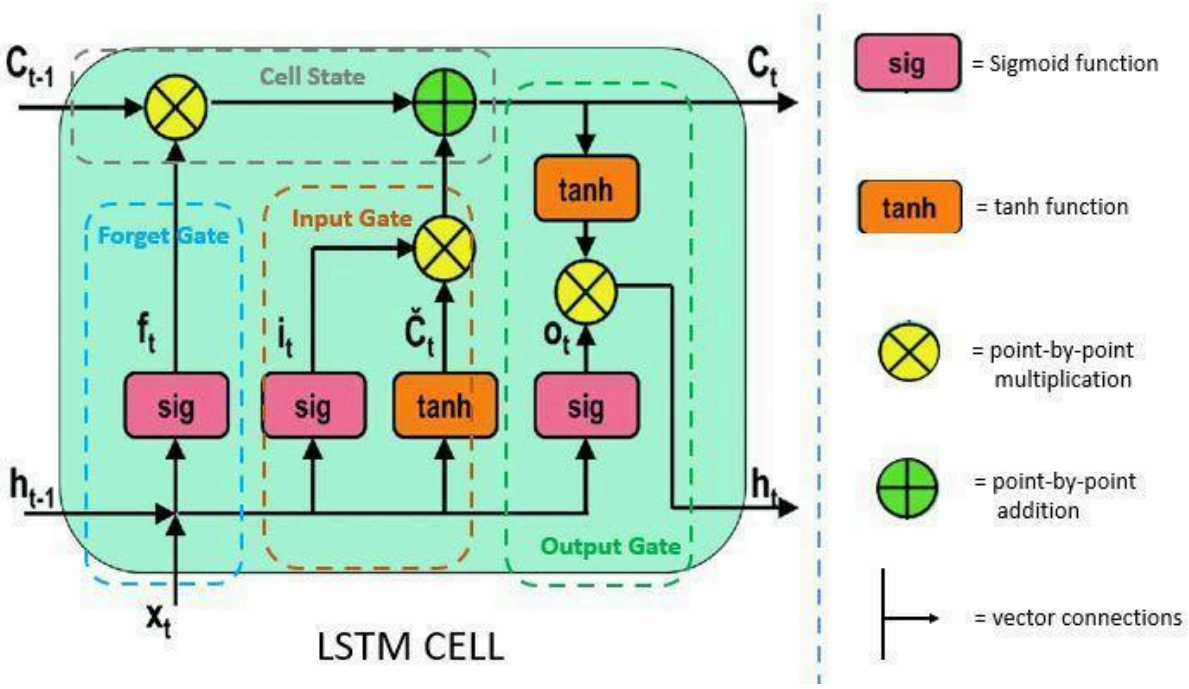


Figure 4.7: Structure of a Cell in Long Short-Term Memory

Forget gates decide what information to discard from a previous state by assigning a previous state, compared to a current input, a value between 0 and 1. A (rounded) value of 1 means to keep the information, and a value of 0 means to discard it.

Input gates decide which pieces of new information to store in the current state, using the same system as forget gates.

Output gates control which pieces of information in the current state to output by assigning a value from 0 to 1 to the information, considering the previous and current states.

> *Selectively outputting relevant information from the current state allows the LSTM network to maintain useful, long-term dependencies to make predictions, both in current and future time-steps*.

Long-short-term memory: Working

1. LSTMs are a type of Recurrent Neural Network (RNN) that can better retain long-term dependencies in the data.

2. They have a more complex structure than regular RNNs, consisting of input, output, and forget gates that can selectively retain or discard information from the hidden state.
3. The input gate determines which information from the current input to store in the hidden state.
4. The forget gate determines which information from the previous hidden state to keep or discard.
5. The output gate determines which information from the hidden state to output as the final prediction.
6. This combination of gates allows LSTMs to retain important information from long sequences and discard irrelevant or outdated information.
7. LSTMs are often used for tasks involving long-term dependencies, such as language translation and language modeling.

Gated Recurrent Unit (GRU): Working

1. GRUs are a type of Recurrent Neural Network (RNN) that uses a simpler structure than LSTMs and is easier to train.
2. They have two gates: an **update gate** and a **reset gate.**
3. The update gate determines which information from the previous hidden state and current input to keep, and the reset gate determines which information to discard.
4. The final hidden state is a combination of the information retained by the update gate and the current input.
5. This combination of gates allows GRUs to retain relevant information from long sequences and discard irrelevant or outdated information.
6. GRUs are often used for tasks involving sequential data, such as language translation and language modeling.

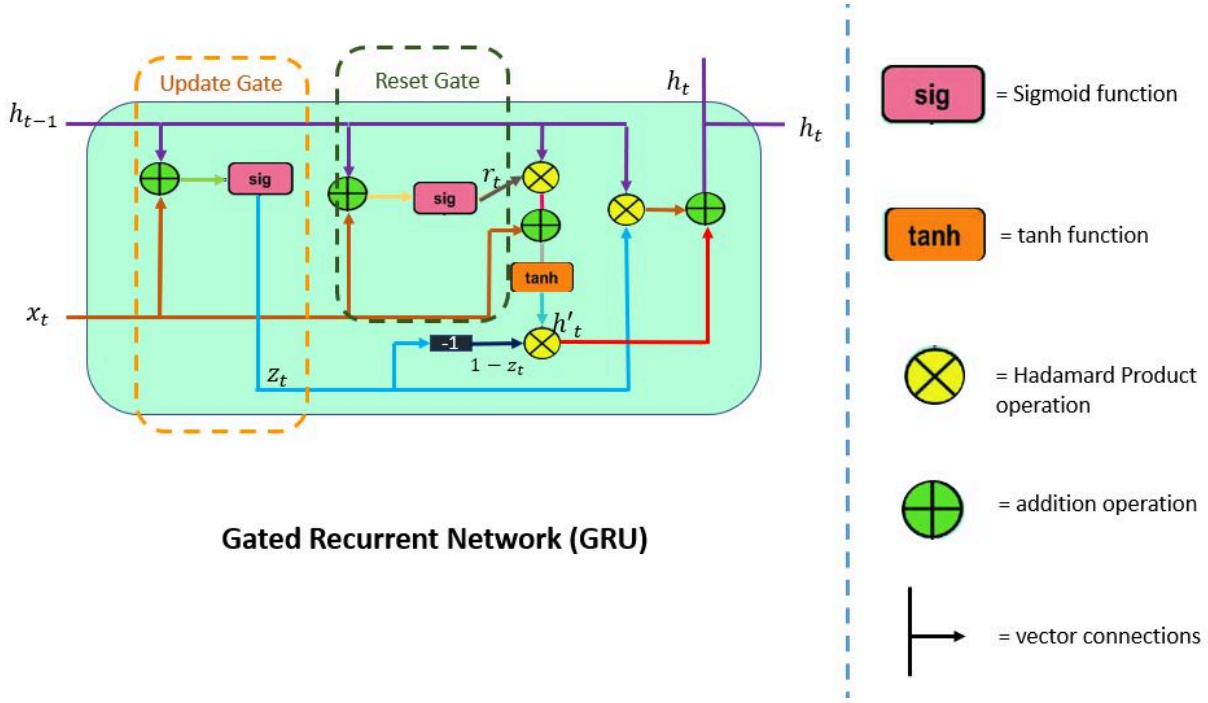


Figure 4.8: Working of GRU

- The **update gate** (z_t) is responsible for determining the amount of previous information (prior time steps) that needs to be passed along the next state.

Update gate

$$Z_t = \sigma\,(W_Z \,.\, [h_{t-1}, x_t] + b_Z)$$

t = *timestep*

Z_t = *Update Gate at t*

x_t = *input vector*

h_t = *previous hidden state*

W_Z = *Update gate weight matrix*

b_Z = *connection bias at t*

- The **reset gate** (r_t) is used from the model to decide how much of the past information is needed to neglect. The formula is the same as the update gate. There is a difference in their weights and gate usage.

  Reset gate

  $r_t = \sigma (W_r . [h_{t-1}, x_t] + b_r)$

  t = *timestep*

  Z_t = *Reset Gate at t*

  x_t = *input vector*

  h_t = *previous hidden state*

  W_Z = *Reset gate weight matrix*

  b_Z = *connection bias at t*

Comparison of RNN, LSTM, GRU

Table 4.1: Comparison of RNN, LSTM, GRU

	RNN	LSTM	GRU
Structure	Simple	More complex	Simpler than LSTM
Training	Can be difficult	Can be more difficult	Easier than LSTM
Performance	Good for simple tasks	Good for complex tasks	Can be intermediate between simple and complex tasks
Hidden state	Single	Multiple (memory cell)	Single
Gates	None	Input, output, forget	Update, reset
Ability to retain long-term dependencies	Limited	Strong	Intermediate between RNNs and LSTMs

Machine translation

A machine translation model is similar to a language model except it has an encoder network placed before. For this reason, it is sometimes referred as a conditional language model.

Statistical Machine Translation- is the use of statistical models that learn to translate text from a source language to a target language.

Given a sentence T in the target language, we seek the sentence S from which the translator produced T. We know that our chance of error is minimized by choosing that sentence S that is most probable given T. Thus, we wish to choose S so as to maximize Pr(S|T).

The approach is data-driven, requiring only a corpus of examples with both source and target language text. This means linguists are no longer required to specify the rules of translation.

Neural Machine Translation- is the use of neural network models to learn a statistical model for machine translation.

The key benefit to the approach is that a single system can be trained directly on source and target text, no longer requiring the pipeline of specialized systems used in statistical machine learning.

Unlike the traditional phrase-based translation system which consists of many small sub-components that are tuned separately, neural machine translation attempts to build and train a single, large neural network that reads a sentence and outputs a correct translation

Encoder-Decoder Model- Multilayer Perceptron neural network models can be used for machine translation, although the models are limited by a fixed-length input sequence where the output must be the same length.

These early models have been greatly improved upon recently through the use of recurrent neural networks organized into an encoder-decoder architecture that allow for variable length input and output sequences.

An encoder neural network reads and encodes a source sentence into a fixed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder–decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence

Beam search- Beam width

In machine translation, Given an input sequence x<1>, x<2>, x<3>,…., x<Tx> length Tx are in language-1 and the output generated by the decoder network y<1>, y<2>,…., y<Ty> be of length Ty in language-2. The outputs are given by the probability P(y<1>,y<2>,….., y<Ty>|a<Tx>). To pick up the most likely sentence this probability expression needs to be maximized .The goal is to find a sentence y such that:

$$Y = arg\ max\ P(y^{<1>}, \ldots, y^{<Ty>} \mid x$$

Beam search decoding iteratively creates text candidates (beams) and scores them.

It is a heuristic search algorithm used in machine translation and speech recognition to find the likeliest sentence y given an input x.

- Step 1: Find top B likely words $y^{<1>}$
- Step 2: Compute conditional probabilities $y^{<k>} |x, y^{<1>}, \ldots.., y^{<k-1>}$
- Step 3: keep top b combinations $x, y^{<1>}, \ldots.., y^{<k>}$

Beam width: The beam width B is a parameter for beam search.

- Large values of B yield to better result but with slower performance and increased memory.
- Small values of B lead to worse results but is less computationally intensive.
- A standard value for B is around 10.
    - *if the beam width is set to 1, then this is equivalent to a naive greedy search*

Bleu score

Length normalization: In order to improve numerical stability, beam search is usually applied on the following normalized objective, often called the normalized log-likelihood objective, defined as:

$$\text{Objective} = \frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log \left[p(y^{<t>} | x, y^{<1>}, \ldots, y^{<t-1>}) \right]$$

Bleu score The **Bi**lingual **e**valuation **u**nderstudy (bleu) score quantifies how good a machine translation is by computing a similarity score based on n-gram precision. It is defined as follows:

$$\text{bleu score} = \exp\left(\frac{1}{n} \sum_{k=1}^{n} p_k \right)$$

where p_n is the bleu score on n-gram only defined as follows:

$$p_n = \frac{\sum_{\text{n-gram} \in \hat{y}} \text{count}_{\text{clip}}(\text{n-gram})}{\sum_{\text{n-gram} \in \hat{y}} \text{count}(\text{n-gram})}$$

Attention Model

Attention is the human ability to concentrate on a few specific things while ignoring others. Just like the human brain, the attention mechanism attempts to help neural networks selectively concentrate on a few relevant tasks while ignoring others.

In ANN, **attention** is a technique that is meant to mimic cognitive attention. The effect enhances some parts of the input data while diminishing other parts — the motivation being that the network should devote more focus to the small, but important, parts of the data. Learning which part of the data is more important than another depends on the context, and this is trained by gradient descent.

Example of attention in action, let us say you look at a class group photo. These photos have rows of children sitting or standing along with the class teacher. If someone asks, 'How many children are in the photo?' your brain automatically begins counting the heads of children while ignoring other details in the photo like the colour of the children's uniform. If someone

asks, 'Who is the teacher?' your brain looks for individuals with adult features while ignoring the children. This is an example of the attention mechanism automatically done by our brain.

Attention models involve focusing on the most important components while perceiving some of the additional information. This is similar to the visual attention mechanism that the human brain uses. For example, the human brain may initially focus on a particular aspect image with a higher resolution focus and view the surrounding areas with a lower resolution. However, as the brain begins to understand the image, it adjusts the focal point to understand all aspects thoroughly.

Attention models evaluate inputs to identify the most critical components and assign each of them with a weight.

For example, if using an attention model to translate a sentence from one language to another, the model would select the most important words and assign them a higher weight. Similarly, it assigns the less significant words a lower value. This helps achieve a more accurate output prediction.

Attention Model allows an RNN to pay attention to specific parts of the input that is considered as being important, which improves the performance of the resulting model in practice. Also, attention scores are commonly used in image captioning and machine translation.

$$C^{<t>} = \sum_{t'} \alpha^{<t,t'>} a^{<t'>} \text{ with } \sum_{t'} \alpha^{<t,t'>} = 1$$

$\alpha^{<t,t'>}$ is the amount of attention that the output $y^{<t>}$ should pay to the activation $a^{<t>}$ and $c^{<t>}$

Benefits of Attention Models

- Attention models reduces larger, more complicated tasks into smaller, more manageable areas of attention to understand and process sequentially.
- The models work within neural networks.
- Using attention models enables the network to **focus on a few particular aspects** at a time and ignoring the rest.

- This allows for efficient and sequential data processing, especially when the network needs to **categorize entire datasets.**
- Attention models also called attention mechanisms are deep learning techniques used to provide an additional focus on a specific component.
- The model typically focuses on one component within the network's architecture that's responsible for managing and quantifying the interdependent relationships within input elements, called **self-attention,** or between input and output elements, called **general attention**.

How do attention models work?

The attention framework makes it possible for the neural network to replicate the visual attention mechanism of the human brain. Just like humans do not focus on each word while reading a text, the attention framework allows the neural network to focus on keywords and other important information with intense, high-resolution focus and the other words in low-resolution. The neural network adjusts its focal point as it begins to understand the scene further.

- Typically, programmers code the attention framework as a function.
- The function maps the input query and "s set" of value pairs to an output.
- The input values, keys, query and outcomes are all vectors. The model then calculates the output as a weighted sum of the values.
- A function that is compatible with the initial query and the corresponding key value is the weight of each value.

Comparing The Working Of Seq2seq And Attention Frameworks

- To compare the working of these two models, let us consider the following sentences:
- Input sentence in English: Shalini is a good girl
- Output sentence in Hindi: शालिनी एक अच्छी लड़की है

- The traditional seq2seq model discards the intermediate states of the encoder and uses only the final vector states to feed the decoder. Here is the illustration of this process:

  Shalini -----> Seq 1
  is -----> Seq 2
  a -----> Seq 3
  good -----> Seq 4
  girl -----> Seq 5:

- The seq2seq model discards the encoder outputs from Seq 1 to 5, and the system feeds only the last state of the encoder as input to the decoder. While this technique works for smaller sentences, it becomes a problem as the length of the input sequence increases. It leads to a bottleneck, as it becomes difficult to summarize long sequences into a fixed-length vector.
- The attention framework does not throw away these intermediate encoder states. Instead, it uses all intermediate states, Seq 1 to 5, to construct the context vectors that the decoder uses to generate the output sequence.

When to use attention models

- The initial purpose of attention models was to help improve **computer vision** and the **encoder-decoder-based neural machine translation system**. This system uses natural language processing (NLP) and relies on huge data libraries that have complex functions.
- Using attention models helps **create maps to fixed-length vectors** to generate translations and understanding. While these may not be wholly accurate, it provides a result that represents the general sentiment and intention of the initial input.

Types Of Attention Models

- Global attention model
- Local attention model
- Self-attention model

Global attention model

Global attention model is also known as the soft attention framework. Collects inputs from all encoder and decoder states before evaluating the current state to generate the output. It uses each encoder step and decoder preview step to align or calculate attention weights. It multiplies the results from each encoder step with **globally aligned weights** to find the content value to feed to the recurrent neural network (RNN). This model uses values from the RNN to find the decoder's outputs.

Local attention model

Local attention model is also known as the hard attention model. Has a similar working structure to the global model. The significant difference is that the local attention mechanism uses only a few encoder positions to calculate the align weights. This model determines the context vector and aligns weights by selecting words from the encoder's source and the first-aligned position.

It allows for predictive and monotonic alignment. The predictive alignment enables the model to predict the final alignment position and the monotonic alignment considers only select information. It also combines specific aspects of hard and soft attention.

Self-attention model

The self-attention mechanism focuses on various positions from a single input sequence. We can combine the global and local attention frameworks to create this model. The difference is that it considers the same input sequence instead of focusing on the target output sequence

Reinforcement Learning

Reinforcement learning is a branch of machine learning that is concerned to take a sequence of actions in order to maximize some reward. RL does not know anything about the environment, it learns what to do by exploring the environment. It uses actions, and receive states and rewards. The agent can only change the environment through actions.

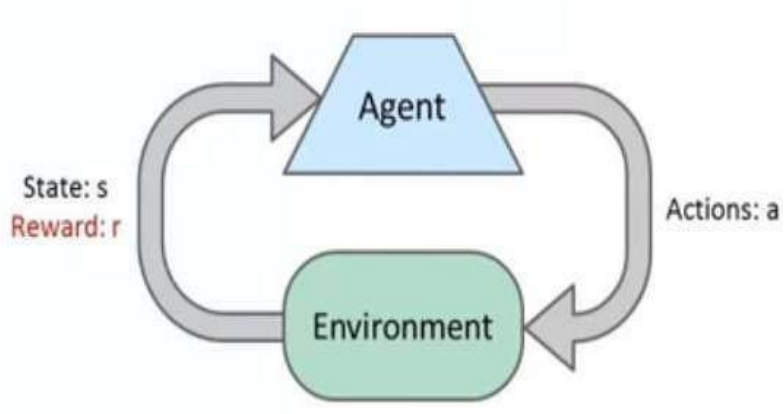


Figure 4.9: Reinforcement Learning Model

Reinforcement learning (RL) is a general framework where agents learn to perform actions in an environment so as to maximize a reward. The two main components are the **environment**, which represents the problem to be solved, and the **agent**, which represents the learning algorithm.

- The agent and environment continuously interact with each other.
- At each time step, the agent takes an action on the environment based on its policy , where S_t is the current observation from the environment, and receives a reward r_{t+1} and the next observation S_{t+1} from the environment.
- The goal is to improve the policy so as to maximize the sum of rewards (return).
- *Note: It is important to distinguish between the state of the environment and the observation, which is the part of the environment state that the agent can see, e.g. in a poker game, the environment state consists of the cards belonging to all the players and the community cards, but the agent can observe only its own cards and a few community cards. In most literature, these terms are used interchangeably and observation is also denoted as S.*

One of the big difficulties of RL is that some actions take time to create a reward. Also, the reward received by the environment is not related to the last action, but some action on the past. We are not told what actions lead to higher rewards. Instead, we learn it from experience. We repeat try-and-error attempts and observe which action gives us a higher reward and which

gives a lower reward. Moreover, we are not even told when rewards are given in the beginning. They might be given immediately or might be given after a few time steps after we take action.

Therefore, we need a dynamical framework that captures those two features, "try-and-error search" and "delayed rewards.".

Markov Decision Process (MDP)

MDP is a framework that can solve most Reinforcement Learning problems with discrete actions. With the Markov Decision Process, an agent can arrive at an **optimal policy** for maximum rewards over time. The Markov decision process (MDP), is an approach in reinforcement learning to take decisions in a grid world environment.

A grid world environment consists of states in the form of grids. The MDP tries to capture a world in the form of a grid by dividing it into states, actions, models/transition models, and rewards. The solution to an MDP is called a policy and the objective is to find the optimal policy for that MDP task.

An MDP consists of two elements; the agent and the environment.

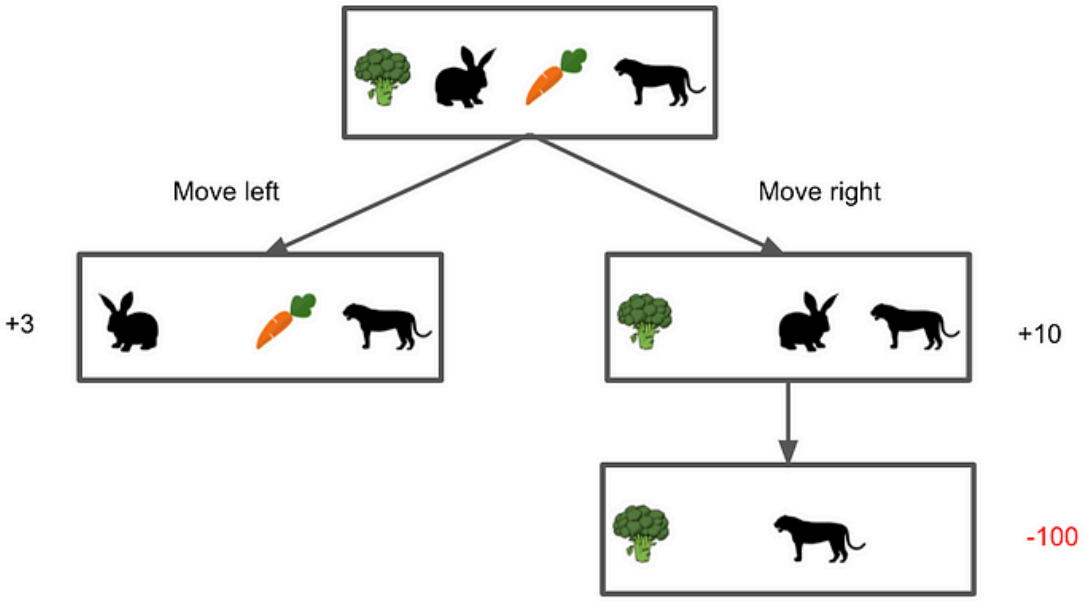


Figure 4.9: Example of Markov Decision Process

The agent is a learner or decision-maker. In the above example, the agent is the rabbit. The environment is everything surrounding the agent. In the example, the environment includes everything in the field where the rabbit is with food and the tiger. After the agent taking an action, we face a different situation. We call these different situations states. For example, the situation where the rabbit has not moved yet is considered as a state, and the situation where the rabbit is between the broccoli and the tiger after she eats the carrot is considered as another state.

Markov Decision Process (MDP) Framework

The following diagram is a formalization of MDP. At time t, the agent at state S_t chooses an action A_t from the action space and the environment returns a new state S_{t+1} from the state space. Then, the agent receives the reward R_{t+1} depending on the starting state, the taken action, and the subsequent state.

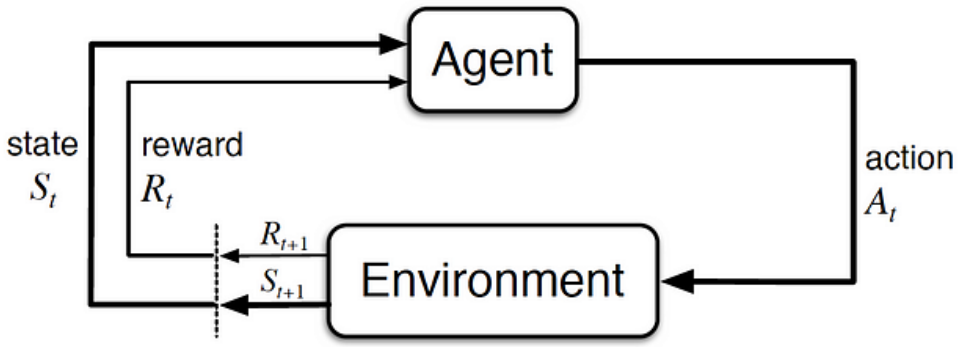


Figure 4.10: Reinforcement Learning used in MDP Framework

In the rabbit example, the **action space** consists of moving right and left. The **state space** includes all the four possible situations, 1) the rabbit is at the initial position, 2) the rabbit is between the broccoli and tiger, 3) the rabbit is the leftmost after eating the broccoli, and 4) The rabbit being eaten by the tiger. The possible rewards are +3(Situation 2), +10(Situation 3), and -100(Situation 4). The details are described in the following diagram.

- The MDP framework (of reinforcement learning) is important because it accounts for a change in situations that might lead to change in optimal action, which is not considered in the k-armed bandit framework.

- An MDP consists of its agent and environment. The agent interacts with the environment observing transitions of states and receiving rewards to find an optimal action (the maximum cumulative rewards) at each state.

Bellman Equation

The agent tries to get the most expected sum of rewards from every state it lands in. In order to achieve that we must try to get the optimal value function, i.e. the maximum sum of cumulative rewards. Bellman equation will help us to do so.

- Bellman equation **decomposes the value function** into two parts, the immediate reward plus the discounted future values.
- This equation simplifies the computation of the value function, such that rather than summing over multiple time steps, we can find the optimal solution of a complex problem by breaking it down into simpler, recursive sub-problems and finding their optimal solutions.

Bellman equation for the State-value function

- This equation tells us how to find the value of a state 's' following a policy π. It recursively breaks down the value computation into an immediate expected reward from the next state, r(s,a), plus the value of a successor state, Vπ(s'), with a discount factor γ.

$$V_\pi(s) = \sum_a \pi(a|s) \cdot \sum_{s'} P^a_{ss'}(r(s,a) + \gamma V_\pi(s'))$$

Bellman equation for the Action-value function

- The state-value function, this equation tells us how to find recursively the value of a state-action pair following a policy π.

$$Q_\pi(s,a) = \sum_{s'} P^a_{ss'}(r(s,a) + \gamma \cdot \sum_{a'} \pi(a'|s') \cdot Q_\pi(s',a'))$$

- The state-value function V(s') is equivalent to the sum of the action-value functions Q(s',a') of all outgoing actions a', multiplied by the policy probability of selecting each action, π(a'|s'), the previous formula can be expressed as follows:

$$Q_\pi(s,a) = \sum_{s'} P^a_{ss'}(r(s,a) + \gamma V_\pi(s'))$$

γ, denotes the discount factor, is a factor multiplying the future expected reward, and varies on the range of [0,1]. It controls the importance of the future rewards versus the immediate ones. The lower the discount factor is, the less important future rewards are, and the Agent will tend to focus on actions which will yield immediate rewards only

Unfortunately, in most scenarios, we do not know the probability P and the reward r, so we cannot solve MDPs by directly applying the Bellman equation. But there are some alternatives to find them from experience.

Actor-Critic model

Actor-Critics aim to take advantage of all the good stuff from both value-based and policy-based while eliminating all their drawbacks. And how do they do this? The principal idea is to **split the model in two**:

- one for computing an action based on a state and
- another one to produce the Q values of the action.

- The "**Actor" updates the policy distribution** in the direction suggested by the Critic (such as with policy gradients).
- The "**Critic" estimates the value function**. This could be the action-value (the Q value) or state-value (the V value).
- And both the Critic and Actor functions are parameterized with neural networks.
- Actor-Critic algorithms combine the two methods in order to create a more robust method.

How Actor Critic works

Imagine you play a video game with a friend that provides you some feedback. You're the Actor and your friend is the Critic.

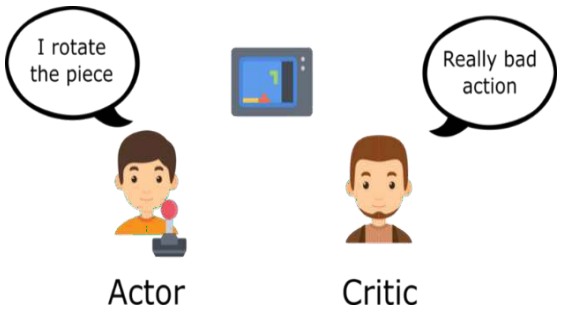


Figure 4.11: Example of Actor Critic Model

- At the beginning, you don't know how to play, so you try some action randomly. The Critic observes your action and provides feedback.
- Learning from this feedback, you'll update your policy and be better at playing that game.
- On the other hand, your friend (Critic) will also update their own way to provide feedback so it can be better next time.

The idea of Actor Critic is to have two neural networks. We estimate both:

ACTOR: A policy function, controls how our agent acts.	CRITIC: A value function, measures how good these actions are.

Both run in parallel. Because we have two models (Actor and Critic) that must be trained, it means that we have two set of weights that must be optimized separately.

Q Value Function

Q Value (Q Function): Usually denoted as Q(s,a) (sometimes with a π subscript, and sometimes as Q(s,a; θ) in Deep RL), Q Value is a measure of the overall expected reward assuming the Agent is in state s and performs action a, and then continues playing until the end

of the episode following some policy π. Its name is an abbreviation of the word "Quality", and it is defined mathematically as:

$$Q(s,a) = \mathbf{E}\left[\sum_{n=0}^{N} \gamma^n r_n\right]$$

where N is the number of states from state '*s*' till the terminal state, *'γ'* is the discount factor and r^0 is the immediate reward received after performing action a'' in state *'s'*.

Q Learning

Q-learning is a **model-free reinforcement** learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards without requiring adaptations. Approximate the state-action pairs Q-function from the samples of Q(s, a) that we observe during interaction with the environment. This approach is known as **Time-Difference Learning**.

Q-learning is an **off-policy reinforcement learning** algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward.

Figure 4.12: Q Learning Model

The *'q'* in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward. In its most simplified form, it uses a table to store all Q-Values of all possible state-action pairs possible. It updates this table using the Bellman equation, while action selection is usually made with an ε-greedy policy (no uncertainties in state-transitions and expected rewards), the update rule of Q-Learning is:

$$Q(s, a) = r(s,a) + \gamma \, max_a \, Q(s', a)$$

where N is the number of states from state *'s'* till the terminal state, *'γ'* is the discount factor and *'r^0'* is the immediate reward received after performing action *'a'* in state *'s'*.

State–action–reward–state–action (SARSA)

- **The Markov Property**

The Markov Property state that: **"**Future is Independent of the past given the present"

- **Transition:** Moving from one state to another is called Transition**.**
- **Transition Probability:** The probability that the agent will move from one state to another is called transition probability.

Mathematically we can express this statement as: S[t] denotes the current state of the agent and s[t+1] denotes the next state. What this equation means is that the transition from state S[t] to S[t+1] is entirely independent of the past.

State–action–reward–state–action (SARSA) is an algorithm for learning a Markov decision process policy, used in the reinforcement learning area of machine learning. The main function for updating the Q-value depends on the current state of the agent "**S1"**, the action the agent chooses "**A1"**, the reward "**R"** the agent gets for choosing this action, the state "**S2"** that the agent enters after taking that action, and finally the next action "A2" the agent chooses in its new state. The acronym for the quintuple (s_t, a_t, r_t, s_{t+1}, a_{t+1}) or (s_t, a_t, r_{t+1} s_{t+1}, a_{t+1},) is SARSA.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

A SARSA agent interacts with the environment and updates the policy based on actions taken, hence this is known as an **on-policy learning** algorithm. The **Q value** for a state-action is **updated by an error**, **adjusted by the learning rate alpha**. Q values represent the possible reward received in the next time step for taking action a in state *'s'*, plus the discounted future reward received from the next state-action observation.

Difference between Sarsa & Q-Learning

The Sarsa algorithm is pretty much the Q-Learning algorithm with a slight modification in order to make it an **on-policy algorithm.** The Q-Learning update rule is based on the Bellman equation for the optimal Q-Value, and so in the case on no uncertainties in state-transitions and expected rewards, the Q-Learning update rule is:

$$Q(s, a) = r(s,a) + \gamma \max_a Q(s', a)$$

In order to transform this into an on-policy algorithm, the last term is modified:

$$Q(s, a) = r(s, a) + \gamma Q(s', a')$$

when here, both actions a and a' are chosen by the same policy. The name of the algorithm is derived from its update rule, which is based on (*s,a,r,s',a'*), all coming from the same policy.

The difference between these two algorithms is that SARSA chooses an action following the same current policy and updates its Q-values whereas Q-learning chooses the greedy action, that is, the action that gives the maximum Q-value for the state, that is, it follows an optimal policy.

S. No.	Topic	Reference
1	Recurrent neural network	https://www.ibm.com/topics/recurrent-neural-networks
2	Long short-term memory	https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/lstm
3	Gated Recurrent Unit	https://www.analyticsvidhya.com/blog/2021/03/introduction-to-gated-recurrent-unit-gru/
4	Translation	https://leonardoaraujosantos.gitbook.io/artificial-inteligence/machine_learning/supervised_learning/recurrent_neural_networks/machine-translation-using-rnn
5	Beam search and width	https://www.javatpoint.com/define-beam-search
6	Bleu score	https://towardsdatascience.com/foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b
7	Attention model	https://in.indeed.com/career-advice/career-development/attention-model#:~:text=An%20attention%20model%2C%20also%20known,attention%20and%20processing%20them%20sequentially.
8	Reinforcement Learning	https://www.techtarget.com/searchenterpriseai/definition/reinforcement-learning
9	RL-framework	https://www.turing.com/kb/best-tools-for-reinforcement-learning
10	MDP	https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-markov-decision-process/#:~:text=A%20Markov%20decision%20process%20(MDP,makes%20sequential%20decisions%20over%20time.
11	Actor-critic model	https://medium.com/intro-to-artificial-intelligence/the-actor-critic-reinforcement-learning-algorithm-c8095a655c14

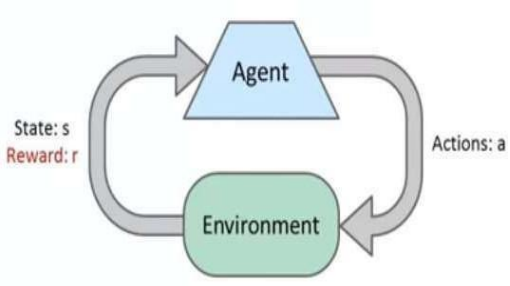
12	Q-learning	https://www.techtarget.com/searchenterpriseai/definition/Q-learning#:~:text=Q%2Dlearning%20is%20a%20machine,way%20animals%20or%20children%20learn.
13	SARSA	https://builtin.com/machine-learning/sarsa

Unit-4

Questions Bank (Descriptive) with Answers

Q1. What is Reinforcement learning? Explain the detailed concepts. (May 2022) (May 2023)

Ans. Reinforcement learning is a branch of machine learning that is concerned to take a sequence of actions in order to maximize some reward. RL does not know anything about the environment, it learns what to do by exploring the environment. It uses actions, and receive states and rewards. The agent can only change the environment through actions.



Reinforcement learning (RL) is a general framework where agents learn to perform actions in an environment so as to maximize a reward. The two main components are the **environment**, which represents the problem to be solved, and the **agent**, which represents the learning algorithm

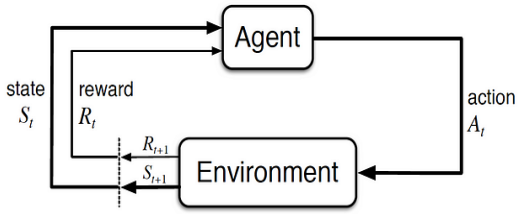
- The agent and environment continuously interact with each other.
- At each time step, the agent takes an action on the environment based on its policy , where S_t is the current observation from the environment, and receives a reward r_{t+1} and the next observation S_{t+1} from the environment.

- The goal is to improve the policy so as to maximize the sum of rewards (return).

One of the big difficulties of RL is that some actions take time to create a reward. Also, the reward received by the environment is not related to the last action, but some action on the past. We are not told what actions lead to higher rewards. Instead, we learn it from experience. We repeat try-and-error attempts and observe which action gives us a higher reward and which gives a lower reward. Moreover, we are not even told when rewards are given in the beginning. They might be given immediately or might be given after a few time steps after we take action.

Q2. Describe the concept of MDP. (May 2022)

Ans. MDP (Markov Decision Process) is a framework that can solve most Reinforcement Learning problems with discrete actions. With the Markov Decision Process, an agent can arrive at an **optimal policy** for maximum rewards over time. The Markov decision process (MDP), is an approach in reinforcement learning to take decisions in a grid world environment.
A grid world environment consists of states in the form of grids. The MDP tries to capture a world in the form of a grid by dividing it into states, actions, models/transition models, and rewards. The solution to an MDP is called a policy and the objective is to find the optimal policy for that MDP task.



An MDP consists of two elements; the agent and the environment. The agent is a learner or decision-maker. In the above example, the agent is the rabbit. The environment is everything surrounding the agent. In the example, the environment includes everything in the field where the rabbit is with food and the tiger. After the agent taking an action, we face a different situation. We call these different situations states. For example, the situation where the rabbit has not moved yet is considered as a state, and the situation where the rabbit is between the broccoli and the tiger after she eats the carrot is considered as another state.

Q3. Explain Q learning algorithm assuming deterministic rewards and actions. (May 2022)

Ans. Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence "model-free"), and it can

handle problems with stochastic transitions and rewards without requiring adaptations. Approximate the state-action pairs Q-function from the samples of *Q(s, a)* that we observe during interaction with the environment. This approach is known as Time-Difference Learning.

Q-learning is an off-policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed.

Specifically for deterministic rewards and actions, q-learning seeks to learn a policy that maximizes the total reward. The 'q' in **q-learning stands for quality.** Quality in this case represents how useful a given action is in gaining some future reward. In its most simplified form, it uses a **table to store all Q-Values** of all possible state-action pairs possible. It updates this table using the Bellman equation, while action selection is usually made with an ε-greedy policy. (no uncertainties in state-transitions and expected rewards), the update rule of Q-Learning is:

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

where N is the number of states from state *'s'* till the terminal state, γ is the discount factor and r^0 is the immediate reward received after performing action a in state s.

Q4. Explain the following term: Attention Model. (May 2022)

Ans. In ANN, attention is a technique that is meant to mimic cognitive attention. The effect enhances some parts of the input data while diminishing other parts — the motivation being that the network should devote more focus to the small, but important, parts of the data.Learning which part

of the data is more important than another depends on the context, and this is trained by gradient descent. Attention models involve focusing on the most important components while perceiving some of the additional information. This is similar to the visual attention mechanism that the human brain uses.

For example, the human brain may initially focus on a particular aspect image with a higher resolution focus and view the surrounding areas with a lower resolution. However, as the brain begins to understand the image, it adjusts the focal point to understand all aspects thoroughly.

- The aim of attention models is to reduce larger, more complicated tasks into smaller, more manageable areas of attention to understand and process sequentially.
- The models work within neural networks.
- Using attention models enables the network to focus on a few particular aspects at a time and ignoring the rest.
- This allows for efficient and sequential data processing, especially when the network needs to categorize entire datasets.

Q5. What are the structural and operational differences between a feed-forward network and a recurrent neural network? Identify the difference between LSTM, GRU and Vanilla-RNN. (May 2023)

Comparison Attribute	**Feed-forward Neural Networks**	**Recurrent Neural Networks**
Signal flow direction	Forward only	Bidirectional
Delay introduced	No	Yes
Complexity	Low	High
Neuron independence in the same layer	Yes	No
Speed	High	slow
Commonly used for	Pattern recognition, speech recognition, and character recognition	Language translation, speech-to-text conversion, and robotic control.

Vanilla RNN, also known as simple RNN, processes inputs sequentially, maintaining the hidden state that encodes information about the inputs it has processed. At each step, the RNN does a series of calculations before producing an output. For classification tasks, a single output is needed. For text generation based on the previous word, an output is required at every time step.This iterative generation process allows the model to progressively generate a coherent sequence of words, building upon the context provided by the preceding words.

	RNN	LSTM	GRU
Structure	Simple	More complex	Simpler than LSTM
Training	Can be difficult	Can be more difficult	Easier than LSTM
Performance	Good for simple tasks	Good for complex tasks	Can be intermediate between simple and complex tasks
Hidden state	Single	Multiple (memory cell)	Single
Gates	None	Input, output, forget	Update, reset
Ability to retain long-term dependencies	Limited	Strong	Intermediate between RNNs and LSTMs

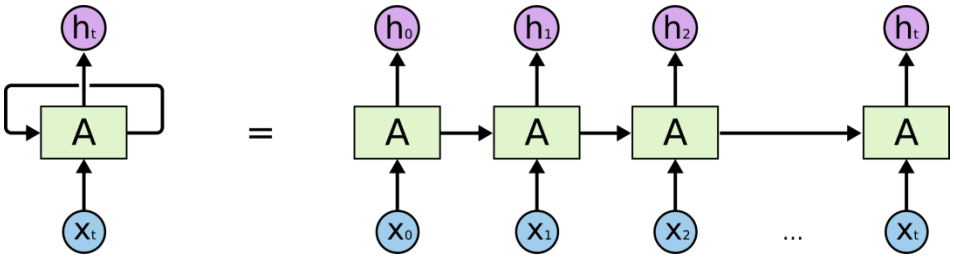
Q6. What do you mean by Recurrent Neural Network? Explain with the help of diagram. In which cases this model is suitable. (Dec 2020)

Ans. The recurrent neural network is a special type of neural network which not just looks at the current input being presented to it but also the previous input. So instead of

Input → Hidden → Output

it becomes

Input + Previous Hidden → Hidden → Output



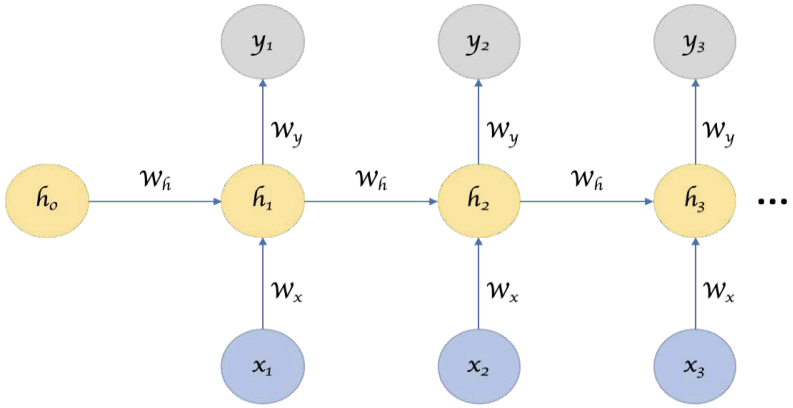
- RNNs can take one or more input vectors and produce one or more output vectors and the output(s) are influenced not just by weights applied on inputs like a regular NN, but also by a "hidden" state vector representing the context based on prior input(s)/output(s).
- So, the **same input could produce a different output depending on previous inputs in the series**

- The current state can be written as $h_t = f(h_{t-1}, x_t)$

- Here, ht is the new state, ht-1 is the previous state while xt is the current input. Because the input neuron would have applied the transformations on our previous input. So each successive input is called as a time step.

- Let's say that the activation function is tanh, the weight at the recurrent neuron is Whh and the weight at the input neuron is Wxh, we can write the equation for the state at time t as –

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



The model is suitable for wide range of applications such as:

1. Natural Language Processing (NLP): (i) Language Modeling: RNNs can predict the next word in a sentence, useful for tasks like text generation and autocomplete. (ii) Machine Translation: RNNs can be used for translation tasks, with one RNN encoding the input sentence and another decoding it in the target language. (iii) Sentiment Analysis: RNNs can analyze text sentiment by capturing contextual information from words in a sequence.

2. Speech Recognition and Synthesis: (i) Speech-to-Text: RNNs are employed to convert spoken language into written text, making them the backbone of speech recognition systems. (ii) Text-to-Speech: RNNs can generate human-like speech from text input, improving voice assistants and accessibility tools.

3. Time-Series Analysis and Forecasting: (i) Financial Forecasting: RNNs can predict stock prices, currency exchange rates, and other financial variables. (ii) Weather Prediction: RNNs can analyze historical weather data to forecast future weather conditions.

4. Music Generation: RNNs can generate music sequences, learning patterns from existing music compositions and producing new compositions.

5. Video Analysis and Action Recognition: (i) Video Understanding: RNNs can process frames in a video sequence to understand actions, objects, and activities.

Gesture Recognition: RNNs can recognize hand gestures and movements in videos for applications like sign language translation.

6. Robotics and Autonomous Systems: (i) Path Prediction: RNNs can help robots predict the paths of moving objects and plan their actions accordingly. (ii) Gesture Control: RNNs enable natural interaction with robots through gesture recognition.

7. Language Generation and Dialogue Systems: (i) Chatbots: RNNs power chatbots and virtual assistants by generating coherent responses in conversations. (ii) Storytelling: RNNs can create stories or narratives based on input prompts.

Q7. Explain Actor-Critic model. List down what are its advantages in reinforcement learning. (Dec 2020)

Ans. Actor-Critics aim to take advantage of all the good stuff from both value-based and policy-based while eliminating all their drawbacks. And how do they do this?

The principal idea is to split the model in two: (i) one for computing an action based on a state and

(ii) another one to produce the Q values of the action.

The "Actor" updates the policy distribution in the direction suggested by the Critic (such as with policy gradients). The "Critic" estimates the value function. This could be the action-value (the Q value) or state-value (the V value). And both the Critic and Actor functions are parameterized with neural networks.

Actor-Critic algorithms combine the two methods in order to create a more robust method.

- The **Actor** takes as **input the state** and **outputs** the best **action.**

- It essentially **controls how the agent behaves** by learning the optimal policy (policy-based).
- The **Critic,** on the other hand, **evaluates the action** by computing the value function (value based).
    - They both get better in their own role as the time passes.
    - The overall architecture will work more efficiently than the two methods separately.

Q8. Describe Q-Learning in brief. What is SARSA algorithm? Explain this. (Dec 2020)

Ans. **Q-learning** is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards without requiring adaptations. Approximate the state-action pairs Q-function from the samples of *Q(s, a)* that we observe during interaction with the environment. This approach is known as Time-Difference Learning.

Q-learning is an off-policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed.

State–action–reward–state–action (SARSA) is an algorithm for learning a Markov decision process policy, used in the reinforcement learning area of machine learning.

The main function for updating the Q-value depends on the current state of the agent "S1", the action the agent chooses "A1", the reward "R" the agent gets for choosing this action, the state "S2" that the agent enters after taking that action, and finally the next action "A2" the agent chooses in its new state.

The acronym for the quintuple *(st, at, rt, st+1, at+1) or (st, at, rt+1, st+1, at+1)* is SARSA.

A SARSA agent interacts with the environment and updates the policy based on actions taken, hence this is known as an on-policy learning algorithm. The Q value for a state-action is updated by an error, adjusted by the learning rate alpha. Q values represent the possible reward received in the

next time step for taking action a in state *'s'*, plus the discounted future reward received from the next state-action observation.

Q9. Explain the difference between Value iteration and Policy iteration. What is Markov Decision Process (MDP)? (Dec 2020)

Ans. Policy iteration and value iteration are both dynamic programming algorithms that find an optimal policy π_* in a reinforcement learning environment. They both employ variations of Bellman updates and exploit one-step look-ahead:

- In policy iteration, we start with a fixed policy. Conversely, in value iteration, we begin by selecting the value function. Then, in both algorithms, we iteratively improve until we reach convergence.
- The policy iteration algorithm updates the policy. The value iteration algorithm iterates over the value function instead. Still, both algorithms implicitly update the policy and state value function in each iteration.
- In each iteration, the policy iteration function goes through two phases. One phase evaluates the policy, and the other one improves it. The value iteration function covers these two phases by taking a maximum over the utility function for all possible actions.

The value iteration algorithm is straightforward. It combines two phases of the policy iteration into a single update operation. However, the value iteration function runs through all possible actions at once to find the maximum action value. Subsequently, the value iteration algorithm is computationally heavier.

Both algorithms are guaranteed to converge to an optimal policy in the end. Yet, the policy iteration algorithm converges within fewer iterations. As a result, the policy iteration is reported to conclude faster than the value iteration algorithm.

Policy Iteration	Value Iteration
Starts with a random policy	Starts with a random value function
Algorithm is more complex	Algorithm is simpler
Guaranteed to converge	Guaranteed to converge
Cheaper to compute	More expensive to compute
Requires few iterations to converge	Requires more iterations to converge
Faster	Slower

Markov Decision Process (MDP): MDP is a framework that can solve most Reinforcement Learning problems with discrete actions. With the Markov Decision Process, an agent can arrive at an **optimal policy** for maximum rewards over time. The Markov decision process (MDP), is an approach in reinforcement learning to take decisions in a grid world environment. A grid world environment consists of states in the form of grids.

The MDP tries to capture a world in the form of a grid by dividing it into states, actions, models/transition models, and rewards. The solution to an MDP is called a policy and the objective is to find the optimal policy for that MDP task.

Unit-4

MCQ/ Quiz Practice Set

1. What is the primary purpose of a Recurrent Neural Network (RNN)?

a) Image classification

b) Text generation

c) Reinforcement learning

d) Object detection

Answer: b) Text generation

2. Which layer type is typically used to capture sequential dependencies in an RNN?

a) Input layer

b) Hidden layer

c) Output layer

d) Activation layer

Answer: b) Hidden layer

3. What is the advantage of using recurrent layers in an RNN?

a) They can capture temporal dependencies in the input data

b) They can handle variable-length inputs

c) They can generate synthetic data

d) They can handle non-linear transformations

Answer: a) They can capture temporal dependencies in the input data

4. What is the purpose of the hidden state in an RNN?

a) To store the information from the previous time step

b) To adjust the learning rate during training

c) To compute the gradients for backpropagation

d) None of the above

Answer: a) To store the information from the previous time step

5. Which activation function is commonly used in the recurrent layers of an RNN?

a) ReLU (Rectified Linear Unit)

b) Sigmoid

c) Tanh (Hyperbolic Tangent)

d) Softmax

Answer: c) Tanh (Hyperbolic Tangent)

6. What is the purpose of the time step parameter in an RNN?

a) To determine the number of recurrent layers in the network

b) To adjust the learning rate during training

c) To specify the length of the input sequence

d) None of the above

Answer: c) To specify the length of the input sequence

7. Which layer type is commonly used to initialize the hidden state in an RNN?

a) Input layer

b) Hidden layer

c) Output layer

d) Activation layer

Answer: b) Hidden layer

8. What is the purpose of the bidirectional RNN architecture?

a) To handle sequential data in both forward and backward directions

b) To reduce the computational complexity of the network

c) To adjust the learning rate during training

d) None of the above

Answer: a) To handle sequential data in both forward and backward directions

9. Which layer type is responsible for making final predictions in an RNN?

a) Input layer

b) Hidden layer

c) Output layer

d) Activation layer

Answer: c) Output layer

10. What is the purpose of the recurrent connection in an RNN?

a) To propagate the hidden state across different time steps

b) To adjust the weights and biases of the network

c) To reduce the dimensionality of the input data

d) None of the above

Answer: a) To propagate the hidden state across different time steps

11. Which layer type is commonly used in RNNs for sequence-to-sequence tasks?

a) Input layer

b) Hidden layer

c) Output layer

d) Attention layer

Answer: d) Attention layer

12. What is the purpose of the backpropagation through time (BPTT) algorithm in RNN training?

a) To compute the gradients and update the network's parameters

b) To adjust the learning rate during training

c) To prevent overfitting by regularizing the model

d) None of the above

Answer: a) To compute the gradients and update the network's parameters

13. Which layer type is commonly used in RNNs to handle variable-length inputs?

a) Input layer

b) Hidden layer

c) Output layer

d) None of the above

Answer: a) Input layer

14. What is the purpose of the initial hidden state in an RNN?

a) To provide the starting point for the recurrent computation

b) To adjust the learning rate during training

c) To compute the gradients for backpropagation

d) None of the above

Answer: a) To provide the starting point for the recurrent computation

15. Which layer type is responsible for handling the output at each time step in an RNN?

a) Input layer

b) Hidden layer

c) Output layer

d) Activation layer

Answer: c) Output layer

16. What is the purpose of the teacher forcing technique in RNN training?

a) To adjust the learning rate during training

b) To propagate the gradients through time

c) To reduce the computational complexity of the network

d) None of the above

Answer: b) To propagate the gradients through time

17. Which layer type is commonly used in RNNs for language modeling tasks?

a) Input layer

b) Hidden layer

c) Output layer

d) None of the above

Answer: c) Output layer

18. What is the purpose of the sequence-to-vector architecture in an RNN?

a) To process an input sequence and produce a fixed-length representation

b) To adjust the weights and biases of the network

c) To reduce the dimensionality of the input data

d) None of the above

Answer: a) To process an input sequence and produce a fixed-length representation

19. Which layer type is responsible for introducing non-linearity in an RNN?

a) Input layer

b) Hidden layer

c) Output layer

d) Activation layer

Answer: d) Activation layer

20. What is the purpose of the forget gate in a Gated Recurrent Unit (GRU)?

a) To control the flow of information from the previous hidden state

b) To adjust the learning rate during training

c) To compute the gradients for backpropagation

d) None of the above

Answer: a) To control the flow of information from the previous hidden state

21. Which layer type is commonly used in RNNs for machine translation tasks?

a) Input layer

b) Hidden layer

c) Output layer

d) Attention layer

Answer: d) Attention layer

22. What is the purpose of the peephole connections in a Long Short-Term Memory (LSTM) network?

a) To allow the cell state to influence the gating mechanisms

b) To adjust the learning rate during training

c) To introduce non-linearity to the network

d) None of the above

Answer: a) To allow the cell state to influence the gating mechanisms

23. Which layer type is responsible for handling variable-length outputs in an RNN?

a) Input layer

b) Hidden layer

c) Output layer

d) None of the above

Answer: c) Output layer

24. What is the purpose of the cell state in an LSTM network?

a) To store long-term dependencies in the input sequence

b) To adjust the learning rate during training

c) To compute the gradients for backpropagation

d) None of the above

Answer: a) To store long-term dependencies in the input sequence

25. Which layer type is commonly used in RNNs for speech recognition tasks?

a) Input layer

b) Hidden layer

c) Output layer

d) None of the above

Answer: c) Output layer

26. What is the purpose of the input gate in an LSTM network?

a) To control the flow of information from the current input

b) To adjust the learning rate during training

c) To introduce non-linearity to the network

d) None of the above

Answer: a) To control the flow of information from the current input

27. Which layer type is responsible for handling variable-length inputs and outputs in an RNN?

a) Input layer

b) Hidden layer

c) Output layer

d) None of the above

Answer: d) None of the above

28. What is the purpose of the output gate in an LSTM network?

a) To control the flow of information to the current output

b) To adjust the learning rate during training

c) To introduce non-linearity to the network

d) None of the above

Answer: a) To control the flow of information to the current output

29. Which layer type is commonly used in RNNs for time series prediction tasks?

a) Input layer

b) Hidden layer

c) Output layer

d) None of the above

Answer: c) Output layer

30. What is the purpose of the reset gate in a Gated Recurrent Unit (GRU)?

a) To reset the hidden state based on the current input

b) To adjust the learning rate during training

c) To introduce non-linearity to the network

d) None of the above

Answer: a) To reset the hidden state based on the current input

31. What sets Reinforcement Learning apart from other machine learning paradigms?

a) Pre-trained models

b) Supervised labeling

c) Interaction with an environment

d) Batch processing

Answer: c) Interaction with an environment

32. What term describes the strategy of choosing actions to maximize cumulative rewards over time?

a) Hyperparameter tuning

b) Feature extraction

c) Reinforcement learning

d) Policy optimization

Answer: d) Policy optimization

33. In Reinforcement Learning, what does the term “agent” refer to?

a) A person supervising the learning process

b) A software program making decisions

c) A labeled data point

d) A neural network architecture

Answer: b) A software program making decisions

34. What are the numerical values used to evaluate the outcomes of actions taken by an agent?

a) Observations

b) Rewards

c) Policies

d) Loss functions

Answer: b) Rewards

35. The “reward function” in RL is used for:

a) Defining the neural network architecture

b) Calculating the probability of actions

c) Evaluating the quality of an agent’s actions

d) Filtering noisy observations

Answer: c) Evaluating the quality of an agent’s actions

36. Balancing between trying new actions and exploiting known actions is known as:

a) Exploration vs. exploitation

b) Model validation

c) Feature extraction

d) Dimensionality reduction

Answer: a) Exploration vs. exploitation

37. Which algorithm is particularly well-suited for environments with continuous action spaces?

a) Q-Learning

b) Deep Q-Network (DQN)

c) Policy Gradient

d) Monte Carlo Tree Search (MCTS)

Answer: c) Policy Gradient

38. How do Double Q-Learning and target network updates address training instabilities in deep RL?

a) By reducing the size of the neural network

b) By updating the target network more frequently

c) By using dropout regularization

d) By mitigating overestimation of action values

Answer: d) By mitigating overestimation of action values

39. What challenge does sparse rewards pose in Reinforcement Learning?

a) Agents become too greedy

b) Agents stop exploring new actions

c) Agents focus only on exploitation

d) Agents struggle to learn effective strategies

Answer: b) Agents stop exploring new actions

40. Which real-world applications benefit from Reinforcement Learning?

a) Image classification

b) Text generation

c) Autonomous driving

d) Data clustering

Answer: c) Autonomous driving

Unit-4

Unit wise list of previous year questions

Q1. What is Reinforcement learning? Explain the detailed concepts. (May 2022) (May 2023)

Q2. Describe the concept of MDP. (May 2022)

Q3. Explain Q learning algorithm assuming deterministic rewards and actions. (May 2022)

Q4. Explain the following term: Attention Model. (May 2022)

Q5. What are the structural and operational differences between a feed-forward network and a recurrent neural network? Identify the difference between LSTM, GRU and Vanilla-RNN. (May 2023)

Q6. What do you mean by Recurrent Neural Network? Explain with the help of diagram. In which cases this model is suitable. (Dec 2020)

Q7. Explain Actor-Critic model. List down what are its advantages in reinforcement learning. (Dec 2020)

Q8. Describe Q-Learning in brief. What is SARSA algorithm? Explain this. (Dec 2020)

Q9. Explain the difference between Value iteration and Policy iteration. What is Markov Decision Process (MDP)? (Dec 2020)

Unit-4

Assignment-IV

Q. No	Question	CO	Learning Level
Q-1	What is Recurrent Neural Network? Explain with the help of a diagram? Also write applications of RNN.	CO4	L3
Q-2	Explain the working of LSTM and GRU.	CO4	L2
Q-3	Discuss Actor critic model. List down its advantages in reinforcement learning?	CO4	L4
Q-4	Describe Q-learning in brief. What is SARSA Algorithm? Explain	CO4	L2
Q-5	Explain the difference between Value iteration and Policy iteration. What is Markov Decision Process (MDP)?	CO4	L2

Unit-4

Placement Specific Questions & Answers

1. What is the primary purpose of a Recurrent Neural Network (RNN)?
   Ans. Text generation

2. Which layer type is typically used to capture sequential dependencies in an RNN?
   Ans. Hidden layer

3. What is the advantage of using recurrent layers in an RNN?
   Ans. They can capture temporal dependencies in the input data

4. What is the purpose of the hidden state in an RNN?
   Ans. To store the information from the previous time step

5. Which activation function is commonly used in the recurrent layers of an RNN?
   Ans. Tanh (Hyperbolic Tangent)

6. What is the purpose of the time step parameter in an RNN?
   Ans. To specify the length of the input sequence

7. Which layer type is commonly used to initialize the hidden state in an RNN?
   Ans. Hidden layer

8. What is the purpose of the bidirectional RNN architecture?
   Ans. To handle sequential data in both forward and backward directions

9. Which layer type is responsible for making final predictions in an RNN?
   Ans. Output layer

10. What is the purpose of the recurrent connection in an RNN?
    Ans. To propagate the hidden state across different time steps

11. What is the purpose of the backpropagation through time (BPTT) algorithm in RNN training?
    Ans. To compute the gradients and update the network's parameters

12. What is the purpose of the initial hidden state in an RNN?
    Ans. To provide the starting point for the recurrent computation

13. What is the purpose of the teacher forcing technique in RNN training?
    Ans. To propagate the gradients through time

14. What is the purpose of the sequence-to-vector architecture in an RNN?
    Ans. To process an input sequence and produce a fixed-length representation

15. What is the purpose of the forget gate in a Gated Recurrent Unit (GRU)?
    Ans. To control the flow of information from the previous hidden state

16. Which layer type is responsible for handling variable-length outputs in an RNN?
    Ans. Output layer

17. What is the purpose of the cell state in an LSTM network?
    Ans. To store long-term dependencies in the input sequence

18. Which layer type is commonly used in RNNs for speech recognition tasks?
    Ans. Output layer

19. What is the purpose of the cell state in an LSTM network?
    Ans. To store long-term dependencies in the input sequence

20. Which layer type is commonly used in RNNs for machine translation tasks?
    Ans. Attention layer

21. What is the purpose of the input gate in an LSTM network?
    Ans. To control the flow of information from the current input

Reference Books

1. Deep Learning (Adaptive Computation and Machine Learning series) 18 November 2016, by Aaron Courville, Ian Goodfellow
2. Introduction to Machine Learning with Python: A Guide for Data Scientists (Greyscale Indian Edition) Paperback – 1 January 2016, by Andreas Mulle
3. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, Third Edition (Full Colour Print), by Aurélien Géron O'Reilly-Media-2019

Content Beyond Syllabus

Workshop on Artificial Intelligence & Machine Learning

Workshop on AIML

Glad to share that a Workshop on AIML has been organized on 21 February 2024 for CSE/AIML/DS/IT/EC/ME Students.

The workshop was conducted by experienced professionals of the field Mr. Faizal Ali, Mr. Rajitram singh and Mr. Bhimesh Verma from Mind Coders, Indore.

During the workshop students gained insights of Artificial Intelligence, Machine Learning, Deep Learning, Big Data.

Students got to know development of live projects and working with clients.

Highlight of the workshop was hands-on project and assignments related to latest technology.

Special Thanks to Respected Director Dr. P. S. Chauhan sir, for his constant Support.

Coordinators

Dr. Praveen Patidar

Ms. Manisha Kadam