CO4

DEEP LEARNING

23AD2205A

Topic:

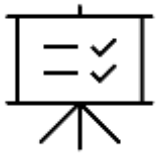
Vanishing and Exploding Gradients Problems in RNN GATED RECURRENT UNITS (GRUS)

Session - 24

AIM OF THE SESSION

To familiarize students with the sequence prediction problems

INSTRUCTIONAL OBJECTIVES



This Session is designed to:

1. Discuss the Contractive Autoencoders and Variational autoencoder
2. Demonstrate the concept of Contractive Autoencoders and Variational autoencoder
   Discussion on Contractive Autoencoders and Variational autoencoder

LEARNING OUTCOMES

At the end of this session, you should be able to: concepts for real time applications

1. To build Contractive Autoencoders and Variational autoencoder
2. To apply different types of Contractive Autoencoders and Variational autoencoder

Vanishing and Exploding Gradients Problems in RNN

The Vanishing and Exploding Gradient Problems in RNNs

- Recurrent neural networks (RNNs) are a powerful type of neural network that can handle sequential data.

- However, RNNs can suffer from two problems during training: vanishing gradients and exploding gradients.

Vanishing Gradients

- The vanishing gradient problem occurs when the gradients propagated backward through time become very small or vanish entirely.
- This can happen in RNNs because the gradients are multiplied by weights at each time step.
- If the weights are less than 1 in magnitude, the gradients will shrink with each multiplication.
- When the gradients become too small, the network becomes difficult to train, especially for long sequences.

Mathematical Illustration of Vanishing Gradients

- Let's consider a simple RNN with a hidden state h(t) at time step t. The hidden state is updated using the following equation:

$$h(t) = f(W\_h * h(t-1) + b\_h)$$

where:

- f is the activation function
- W_h is the weight matrix for the hidden layer
- b_h is the bias vector for the hidden layer
- During backpropagation, the gradients are calculated for each weight and bias in the network. The gradient for W_h can be computed using the chain rule:

$$\nabla W\_h = \partial E(t) / \partial h(t) * \partial h(t) / \partial W\_h$$

- The term $\partial E(t) / \partial h(t)$ represents the gradient of the error at time step t with respect to the hidden state at time step t.
- The term $\partial h(t) / \partial W\_h$ captures how the hidden state at time step t is influenced by the weights in W_h.
- If the weights in W_h are less than 1, then the term $\partial h(t) / \partial W\_h$ will also be less than 1. As the gradients are propagated backward through time, they will be multiplied by this term at each step.
- If this term is less than 1, the gradients will continue to shrink and eventually vanish.

Effects of Vanishing Gradients

- Vanishing gradients can prevent the RNN from learning long-term dependencies in the data.

- The network may only be able to learn from the most recent inputs in the sequence, while earlier inputs have little or no impact on the output.

Exploding Gradients

- The exploding gradient problem is the opposite of the vanishing gradient problem.
- It occurs when the gradients propagated backward through time become very large.
- This can happen if the weights in the RNN are greater than 1 in magnitude.
- With each multiplication during backpropagation, the gradients will explode and become unusable.

Mathematical Illustration of Exploding Gradients

- Similar to vanishing gradients, exploding gradients can be illustrated using the weight update equation for W_h:

$$\nabla W\_h = \partial E(t) / \partial h(t) * \partial h(t) / \partial W\_h$$

- If the weights in W_h are greater than 1, then the term $\partial h(t) / \partial W\_h$ will also be greater than 1.
- As the gradients are propagated backward through time, they will be multiplied by this term at each step.
- If this term is greater than 1, the gradients will continue to grow and eventually explode.

Effects of Exploding Gradients

- Exploding gradients can also make it difficult to train RNNs.
- The large gradients can cause the weights in the network to update erratically, making it impossible for the network to converge on a solution.

Addressing Vanishing and Exploding Gradients

- Several techniques can be used to address vanishing and exploding gradients in RNNs:

o Long Short-Term Memory (LSTM) networks: LSTMs are a type of RNN that uses gating mechanisms to control the flow of information within the network. These gates can help to prevent gradients from vanishing or exploding.

o Gated Recurrent Unit (GRU) networks: GRUs are another type of RNN with gating mechanisms that can alleviate vanishing and exploding gradients.

o Gradient clipping: Gradient clipping is a technique that limits the magnitude of the gradients during backpropagation. This can help to prevent gradients from exploding.

- By using these techniques, it is possible to train RNNs on longer sequences and improve their ability to learn long-term dependencies.

Problem Statement:

Consider a simple RNN with a single hidden layer and a sigmoid activation function. The hidden state h_t at time step t is calculated as:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t)$$

where σ is the sigmoid function, W_h and W_x are weight matrices, h_{t-1} is the previous hidden state, and x_t is the current input. Given $W_h = 0.5$, $h_0 = 1$, and $x_t = 1$ for all t, calculate the hidden state h_t for $t = 1, 2, 3$. What inference can be drawn about the vanishing gradient problem from the results?

Solution Steps:

1. Initialize the parameters:
   - $W_h = 0.5$
   - $h_0 = 1$
   - $x_t = 1$
2. Calculate h_1:
   $h_1 = \sigma(W_h h_0 + W_x x_1)$
   Since $W_x x_t$ is constant and $W_x x_t = b$, we consider it as a bias term b. Assume $b = 1$:
   $h_1 = \sigma(0.5 \cdot 1 + 1) = \sigma(1.5)$

3. **Apply the sigmoid function:**

$\sigma(x) = \frac{1}{1+e^{-x}}$

$\sigma(1.5) = \frac{1}{1+e^{-1.5}} \approx 0.8176$

So, $h_1 \approx 0.8176$.

4. **Calculate h_2:**

$h_2 = \sigma(0.5 \cdot h_1 + 1)$

$h_2 = \sigma(0.5 \cdot 0.8176 + 1) = \sigma(1.4088)$

$\sigma(1.4088) = \frac{1}{1+e^{-1.4088}} \approx 0.8035$

So, $h_2 \approx 0.8035$.

5. Calculate h_3:

$$h_3 = \sigma(0.5 \cdot h_2 + 1)$$

$$h_3 = \sigma(0.5 \cdot 0.8035 + 1) = \sigma(1.40175)$$

$$\sigma(1.40175) = \frac{1}{1+e^{-1.40175}} \approx 0.8024$$

So, $h_3 \approx 0.8024$.

Inference:

As t increases, the hidden states h_t tend to stabilize, showing how gradients can diminish over time in RNNs. This illustrates the vanishing gradient problem, where the influence of the initial input diminishes, making it difficult for the RNN to learn long-term dependencies.

SELF-ASSESSMENT QUESTIONS

When training an RNN, the issue of _____ gradients occurs when the gradients shrink significantly, causing the model to learn very slowly.

vanishing

The problem of _____ gradients happens when the gradients grow exponentially during backpropagation, leading to instability in the network.

exploding

To address vanishing gradients, advanced RNN architectures like _____ are used to maintain long-term dependencies.

Long Short-Term Memory (LSTM)

Conclusion

- Vanishing and exploding gradients are significant issues in training Recurrent Neural Networks (RNNs). Vanishing gradients cause the model to learn very slowly as the gradients shrink, making it difficult to capture long-term dependencies.

- Exploding gradients lead to instability, with gradients growing exponentially and causing large weight updates.

TERMINAL QUESTIONS

Predict: You're training an RNN to predict weather patterns over several days based on historical data. The RNN uses a sigmoid activation function in its hidden layer. Why might the network struggle to learn long-term dependencies in the data, like predicting weather patterns several days ahead? (Focuses on vanishing gradients and long-term dependencies)

Explain: In simple terms, why can vanishing gradients prevent an RNN from learning effectively from past data points in a sequence? (Focuses on the mechanism of vanishing gradients)

Describe: How does gradient clipping help mitigate the vanishing gradient problem in RNN training? (Focuses on the solution of gradient clipping)

Identify: Besides using a different activation function, what other approach can address vanishing gradients in RNNs? (Focuses on alternative solutions)

Compare: Briefly compare the effects of vanishing gradients and exploding gradients on the training process of RNNs. (Focuses on comparing the impacts of both problems)

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Books:

1 Ian Goodfellow and Yoshua Bengio and Aaron Courville (2016) Deep Learning Book
2.Deep Learning Book. eep Learning with Python, Francois Chollet , Manning publications, 2018

Resources

- https://www.tensorflow.org/tutorials/generative/autoencoder
- https://www.linkedin.com/company/autoencoder?originalSubdomain=in
- https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-are-autoencoders-in-deep-learning
- https://blog.keras.io/building-autoencoders-in-keras.

CO4

DEEP LEARNING

22AD3105R

Topic:

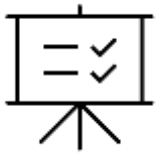
GATED RECURRENT UNITS (GRUS)

Session - 15

AIM OF THE SESSION

To familiarize students with the sequence prediction problems

INSTRUCTIONAL OBJECTIVES



This Session is designed to:

1 Discuss Gated Recurrent Units (GRUs)
2 Demonstrate the Gated Recurrent Units (GRUs)
3 Discussion on Gated Recurrent Units (GRUs)

LEARNING OUTCOMES

At the end of this session, you should be able to: concepts for real time applications

1. To demonstrate Gated Recurrent Units (GRUs)
2. To apply **GATED RECURRENT UNITS (GRUS)**

Gated Recurrent Units (GRUs)

Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) that was introduced by Cho et al. in 2014 as a simpler alternative to Long Short-Term Memory (LSTM) networks. Like LSTM, GRU can process sequential data such as text, speech, and time-series data.

The basic idea behind GRU is to use gating mechanisms to selectively update the hidden state of the network at each time step. The gating mechanisms are used to control the flow of information in and out of the network. The GRU has two gating mechanisms, called the reset gate and the update gate.

The reset gate determines how much of the previous hidden state should be forgotten, while the update gate determines how much of the new input should be used to update the hidden state. The output of the GRU is calculated based on the updated hidden state.

The equations used to calculate the reset gate, update gate, and hidden state of a GRU are as follows:

Reset gate: **r_t = sigmoid(W_r * [h_{t-1}, x_t])**
Update gate: **z_t = sigmoid(W_z * [h_{t-1}, x_t])**
Candidate hidden state: **h_t' = tanh(W_h * [r_t * h_{t-1}, x_t])**
Hidden state: **h_t = (1 − z_t) * h_{t-1} + z_t * h_t'**
where W_r, W_z, and W_h are learnable weight matrices, x_t is the input at time step t, h_{t-1} is the previous hidden state, and h_t is the current hidden state.

GRU networks are a type of RNN that use gating mechanisms to selectively update the hidden state at each time step, allowing them to effectively model sequential data.

They have been shown to be effective in various natural language processing tasks, such as language modeling, machine translation, and speech recognition

Prerequisites: Recurrent Neural Networks, Long Short Term Memory Networks

To solve the Vanishing-Exploding gradients problem often encountered during the operation of a basic Recurrent Neural Network, many variations were developed. One of the most famous variations is the Long Short Term Memory Network(LSTM). One of the lesser-known but equally effective variations is the Gated Recurrent Unit Network(GRU).

Unlike LSTM, it consists of only three gates and does not maintain an Internal Cell State. The information which is stored in the Internal Cell State in an LSTM recurrent unit is incorporated into the hidden state of the Gated Recurrent Unit. This collective information is passed onto the next Gated Recurrent Unit.

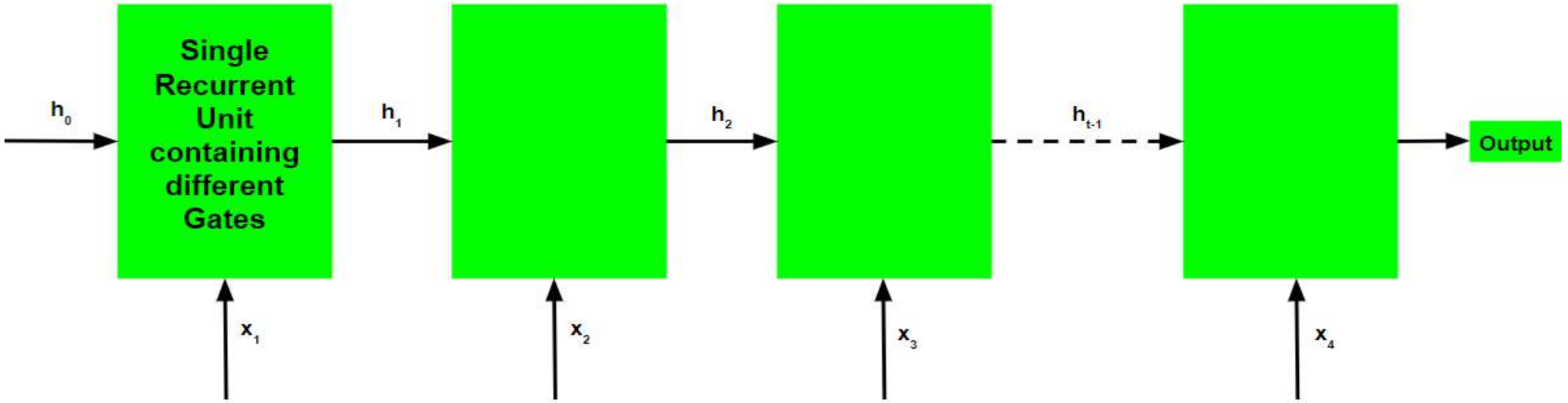
The different gates of a GRU are as described below:-

Update Gate(z): It determines how much of the past knowledge needs to be passed along into the future. It is analogous to the Output Gate in an LSTM recurrent unit.

Reset Gate(r): It determines how much of the past knowledge to forget. It is analogous to the combination of the Input Gate and the Forget Gate in an LSTM recurrent unit.

Current Memory Gate $\overline{h_t}$ It is often overlooked during a typical discussion on Gated Recurrent Unit Network. It is incorporated into the Reset Gate just like the Input Modulation Gate is a sub-part of the Input Gate and is used to introduce some non-linearity into the input and to also make the input Zero-mean. Another reason to make it a sub-part of the Reset gate is to reduce the effect that previous information has on the current information that is being passed into the future

The basic work-flow of a Gated Recurrent Unit Network is similar to that of a basic Recurrent Neural Network when illustrated, the main difference between the two is in the internal working within each recurrent unit as Gated Recurrent Unit networks consist of gates which modulate the current input and the previous hidden state.



Working of a Gated Recurrent Unit:

Take input the current input and the previous hidden state as vectors.

Calculate the values of the three different gates by following the steps given below:-
For each gate, calculate the parameterized current input and previously hidden state vectors by performing element-wise multiplication (Hadamard Product) between the concerned vector and the respective weights for each gate.

Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.

Update Gate : Sigmoid Function
Reset Gate : Sigmoid Function

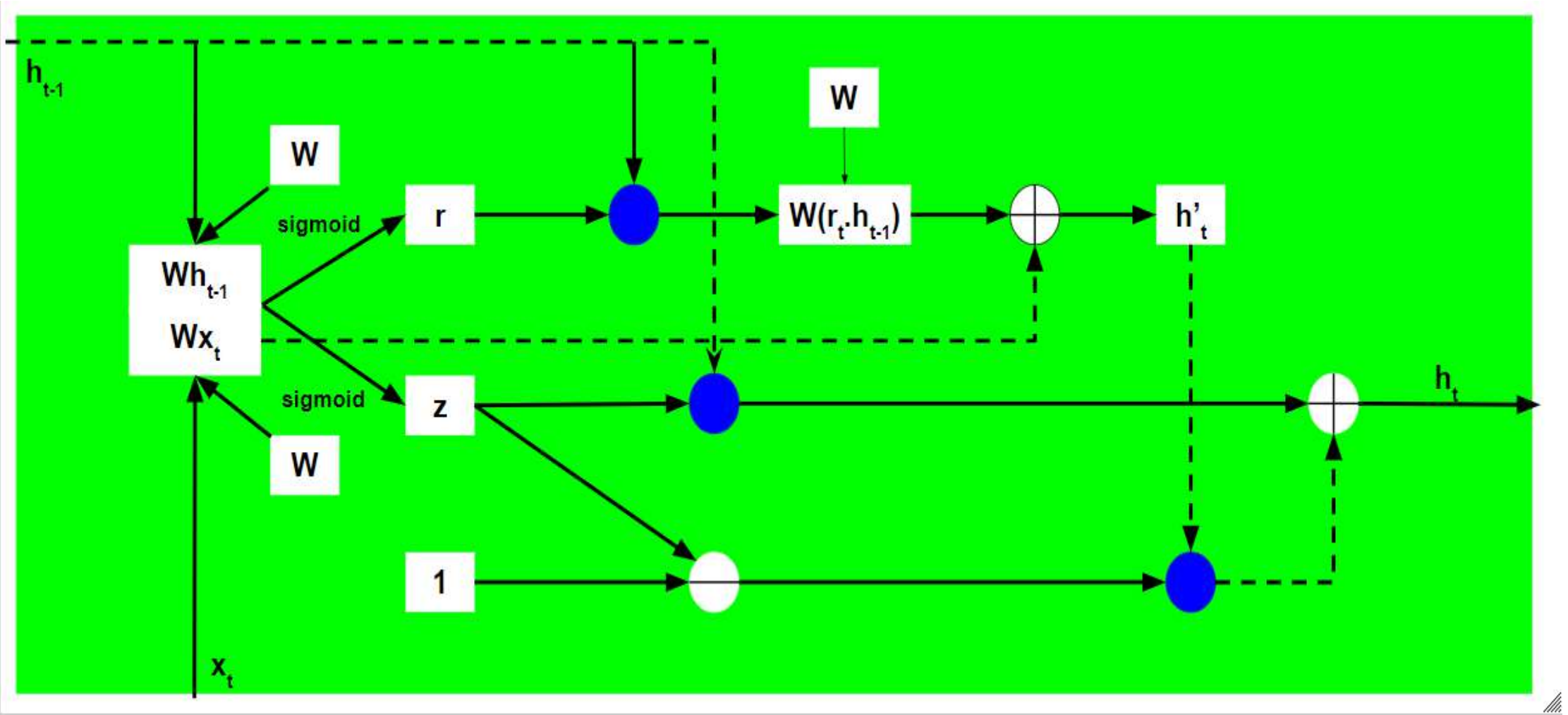
The process of calculating the Current Memory Gate is a little different. First, the Hadamard product of the Reset Gate and the previously hidden state vector is calculated. Then this vector is parameterized and then added to the parameterized current input vector.

$$\overline{h}_t = tanh(W \odot x_t + W \odot (r_t \odot h_{t-1}))$$

To calculate the current hidden state, first, a vector of ones and the same dimensions as that of the input is defined. This vector will be called ones and mathematically be denoted by 1. First, calculate the Hadamard Product of the update gate and the previously hidden state vector. Then generate a new vector by subtracting the update gate from ones and then calculate the Hadamard Product of the newly generated vector with the current memory gate. Finally, add the two vectors to get the currently hidden state vector.

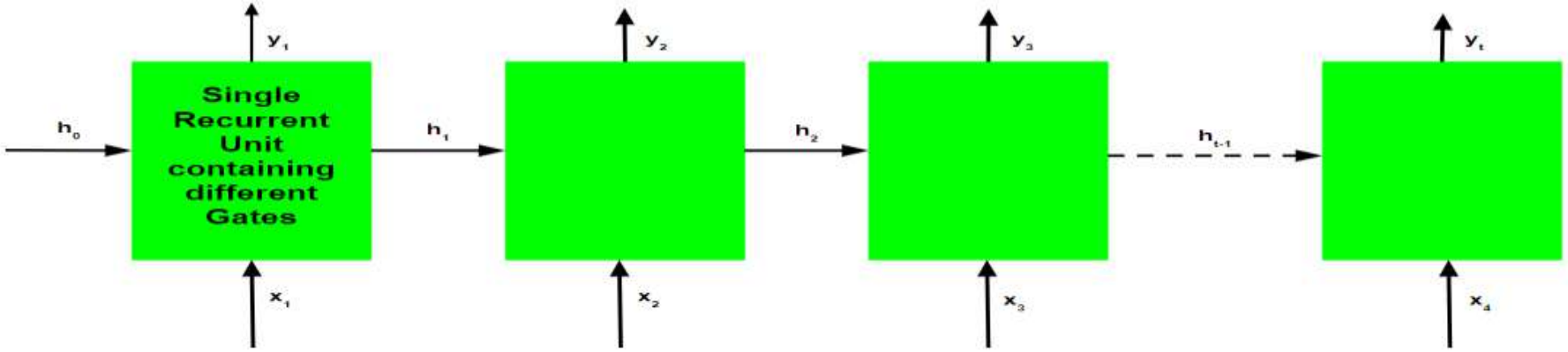
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \overline{h}_t$$

The above-stated working is stated as below:-



Note that the blue circles denote element-wise multiplication. The positive sign in the circle denotes vector addition while the negative sign denotes vector subtraction(vector addition with negative value). The weight matrix W contains different weights for the current input vector and the previous hidden state for each gate.

Just like Recurrent Neural Networks, a GRU network also generates an output at each time step and this output is used to train the network using gradient descent.



Note that just like the workflow, the training process for a GRU network is also diagrammatically similar to that of a basic Recurrent Neural Network and differs only in the internal working of each recurrent unit.

The Back-Propagation Through Time Algorithm for a Gated Recurrent Unit Network is similar to that of a Long Short Term Memory Network and differs only in the differential chain formation.

Let $\overline{y}_t$ be the predicted output at each time step and y_t be the actual output at each time step. Then the error at each time step is given by:-

$$E_t = -y_t log(\overline{y}_t)$$

The total error is thus given by the summation of errors at all time steps.

$$E = \sum_t E_t$$
$$\Rightarrow E = \sum_t -y_t log(\overline{y}_t)$$

Similarly, the value $\frac{\partial E}{\partial W}$ can be calculated as the summation of the gradients at each time step.

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

Using the chain rule and using the fact that $\overline{y}_t$ is a function of h_t and which indeed is a function of $\overline{h}_t$, the following expression arises:-

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial \overline{y}_t} \frac{\partial \overline{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots\cdots \frac{\partial h_0}{\partial W}$$

Thus the total error gradient is given by the following:-

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial \overline{y}_t} \frac{\partial \overline{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots\cdots \frac{\partial h_0}{\partial W}$$

Note that the gradient equation involves a chain of ∂h_t which looks similar to that of a basic Recurrent Neural Network but this equation works differently because of the internal workings of the derivatives of h_t.

SELF-ASSESSMENT QUESTIONS

GRUs are a type of _____ (type of neural network) introduced in 2014.

recurrent neural network)

Compared to Long Short-Term Memory (LSTM) networks, GRUs are a _____ (simplicity comparison) alternative.

simpler)

Like LSTMs, GRUs can process _____ (type of data) data such as text, speech, and time-series data

sequential)

Conclusion

In conclusion, GRU networks are a type of RNN that effectively model sequential data by using gating mechanisms to selectively update the hidden state at each time step. They have been shown to be successful in various natural language processing tasks. GRUs are similar to LSTMs, but they have a simpler architecture and are computationally less expensive.

TERMINAL QUESTIONS

•**Compre** How do Gated Recurrent Unit Networks (GRUs) improve upon regular RNNs in remembering information over long sequences?

•**Choose** You're building a system to write stories. Which is better, a basic RNN or a GRU? Explain your answer.

•**Explain** In a chat system that remembers past conversations, how do GRUs help keep things flowing smoothly?

•**Describe** Briefly explain the steps to train a system that translates sentences between languages using GRUs. What might be some difficulties, and how could you overcome them?

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Books:

1 Ian Goodfellow and Yoshua Bengio and Aaron Courville (2016) Deep Learning Book
2.Deep Learning Book. eep Learning with Python, Francois Chollet , Manning publications, 2018

Resources

- https://www.tensorflow.org/tutorials/generative/autoencoder
- https://www.linkedin.com/company/autoencoder?originalSubdomain=in
- https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-are-autoencoders-in-deep-learning
- https://blog.keras.io/building-autoencoders-in-keras.