Introduction to Neural Networks

ECEN 678

Dr. Mahmoud Nabil Mahmoud
mnmahmoud@ncat.edu

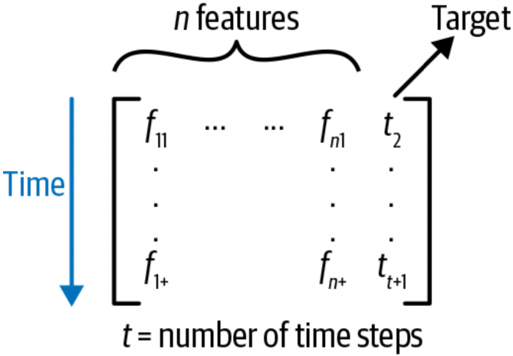
North Carolina A & T State University

March 27, 2023

Outline

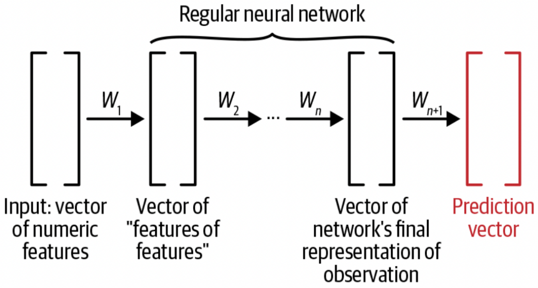
Introduction

- Recurrent neural networks are designed to handle data that appears in sequences
- Instead of each observation being a vector with, say, features, it is now a two-dimensional array of dimension features by time steps.



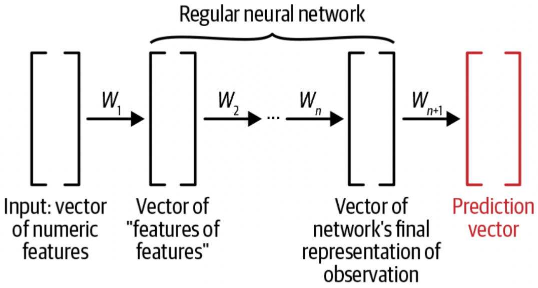
Why this kind of data is different?

- Suppose an application of converting your speech into text (i.e., automatic captioning system)
- The previous matrix can encode the set of signal features at time window t, and the target is the spoken word
- Regular Neural Network shown below will give bad performance if each target is predicted in isolation. **Why?**



Why this kind of data is different?

- Suppose an application of converting your speech into text (i.e., automatic captioning system)
- The previous matrix can encode the set of signal features at time window t, and the target is the spoken word
- Regular Neural Network shown below will give bad performance if each target is predicted in isolation. **Why?**



Because the temporal dependency is ignored.

RNN Solution

- In the first time step, $t = 1$, the target is a function of the features of the first time step

RNN Solution

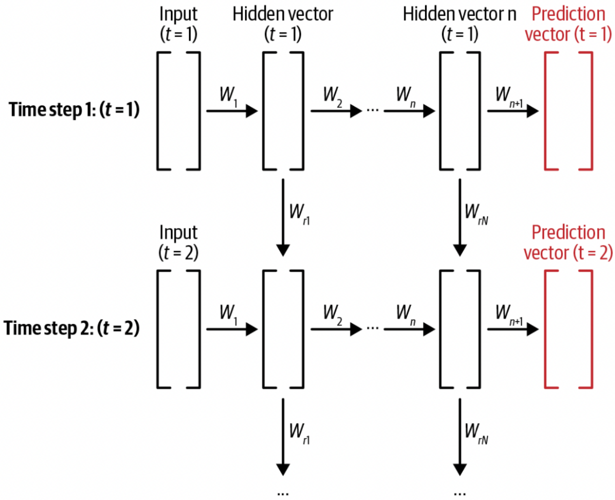
- In the first time step, $t = 1$, the target is a function of the features of the first time step
- In the second time step, $t = 2$, we would pass the features of this time along with a learned representation from the previous time step feature to make to make predictions for $t = 2$.

RNN Solution

- In the first time step, $t = 1$, the target is a function of the features of the first time step
- In the second time step, $t = 2$, we would pass the features of this time along with a learned representation from the previous time step feature to make to make predictions for $t = 2$.
- In the third time step, we would pass through the features from $t = 3$ as well as the representations that now incorporate the information from $t = 1$ and , $t = 2$ and use this information to make predictions for $t = 3$.

Outline

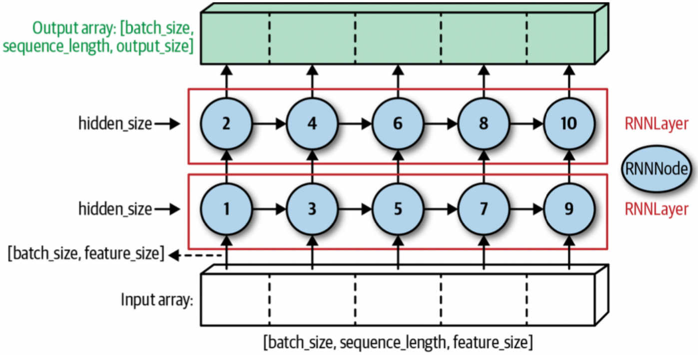
RNN Architecture



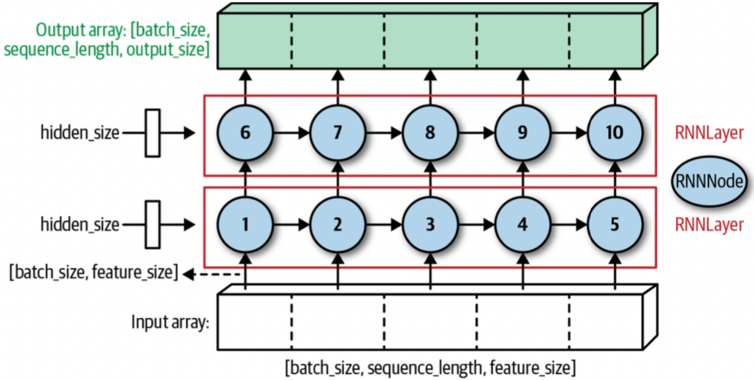
RNN Input Data Shape

- RNNs deal with data in which each input sample is two-dimensional matrix, with shape `(sequence_length, num_features)`
- Since it is always more efficient computationally to pass data forward in batches, the input size will be 3d matrix of shape `(batch_size,sequence_length, num_features)`
- Output is also 3d matrix with shape `(batch_size,sequence_length, output_size)`

RNN Architecture And Order



RNN Architecture And Order

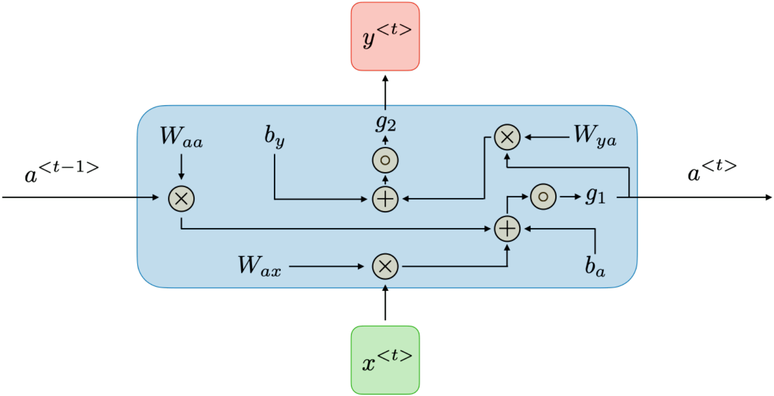


RNN Architecture

Two key components of the RNN architecture:

- **RNN Node:** It is the building block of the RNN layer. It processes the input at each time step with the information learned from the previous time step.
- **RNN Layer:** It is a group of RNN nodes that process the input sequence and can be cascaded for better overall performance.

RNN Node



RNN Node

- RNN Node receive two inputs:
  - The data inputs to the network, of shape `(batch_size, num_features)`
  - The learned representation "hidden state" up to this RNN node, shape `(batch_size, hidden_size)`
- RNN Node has two outputs:
  - The outputs of the network at that time step, of shape `(batch_size, output_size)`
  - The learned representation "hidden state" up to this RNN node, shape `(batch_size, hidden_size)`

Outline

RNN Layer and Forward Pass

1. Prepare the 2d matrix to be input at the first RNN node from your 3d training batch `(batch_size,sequence_length, num_features)`

RNN Layer and Forward Pass

1. Prepare the 2d matrix to be input at the first RNN node from your 3d training batch `(batch_size,sequence_length, num_features)`
    - `data[:,0,:]`

RNN Layer and Forward Pass

1. Prepare the 2d matrix to be input at the first RNN node from your 3d training batch `(batch_size,sequence_length, num_features)`
   - `data[:,0,:]`
2. For the first RNN node initialize a random matrix hidden state of size `(batch_size, hidden_size)` which will get continually updated with the input sequence.

RNN Layer and Forward Pass

1. Prepare the 2d matrix to be input at the first RNN node from your 3d training batch `(batch_size,sequence_length, num_features)`
   - `data[:,0,:]`
2. For the first RNN node initialize a random matrix hidden state of size `(batch_size, hidden_size)` which will get continually updated with the input sequence.
3. Pass these two arrays through the first RNN Node to get the next hidden state of and the current output

RNN Layer and Forward Pass

1. Prepare the 2d matrix to be input at the first RNN node from your 3d training batch `(batch_size,sequence_length, num_features)`
   - `data[:,0,:]`
2. For the first RNN node initialize a random matrix hidden state of size `(batch_size, hidden_size)` which will get continually updated with the input sequence.
3. Pass these two arrays through the first RNN Node to get the next hidden state of and the current output
   - Output shape `(batch_size, output_size)`

RNN Layer and Forward Pass

1. Prepare the 2d matrix to be input at the first RNN node from your 3d training batch `(batch_size,sequence_length, num_features)`
   - `data[:,0,:]`
2. For the first RNN node initialize a random matrix hidden state of size `(batch_size, hidden_size)` which will get continually updated with the input sequence.
3. Pass these two arrays through the first RNN Node to get the next hidden state of and the current output
   - Output shape `(batch_size, output_size)`
   - Hidden state shape `(batch_size, hidden_size)`

RNN Layer and Forward Pass

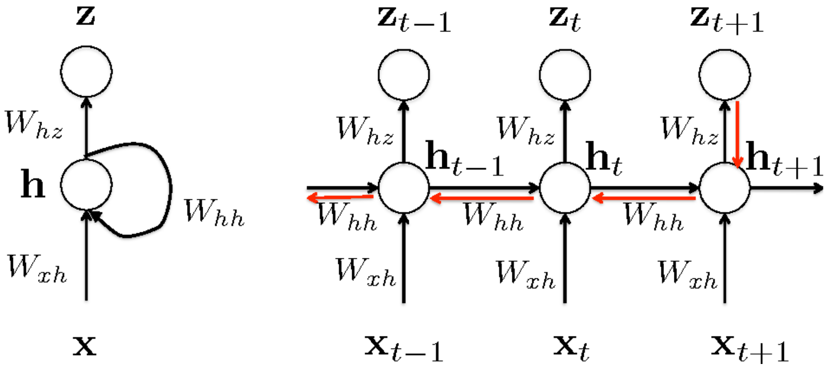
1. Prepare the 2d matrix to be input at the first RNN node from your 3d training batch `(batch_size,sequence_length, num_features)`
   - `data[:,0,:]`
2. For the first RNN node initialize a random matrix hidden state of size `(batch_size, hidden_size)` which will get continually updated with the input sequence.
3. Pass these two arrays through the first RNN Node to get the next hidden state of and the current output
   - Output shape `(batch_size, output_size)`
   - Hidden state shape `(batch_size, hidden_size)`
4. Continue until all `sequence_length` passed through the layer.

RNN Layer and Forward Pass

1. Prepare the 2d matrix to be input at the first RNN node from your 3d training batch `(batch_size,sequence_length, num_features)`
   - `data[:,0,:]`
2. For the first RNN node initialize a random matrix hidden state of size `(batch_size, hidden_size)` which will get continually updated with the input sequence.
3. Pass these two arrays through the first RNN Node to get the next hidden state of and the current output
   - Output shape `(batch_size, output_size)`
   - Hidden state shape `(batch_size, hidden_size)`
4. Continue until all `sequence_length` passed through the layer.
5. Concatenate all the results together to get an output from that layer of shape `(batch_size,sequence_length, output_size)`

Outline

Another Perspective of RNN



- It should be noted that the weights and biases are Shared through the time

Back Propagation Through Time

- Back propagation is done at each point in time.
- At timestep T, the derivative of the loss $\mathcal{L}$ with respect to each weight matrix $W_{(.)}$ is calculated, and the same $W_{(.)}$ are updated.
- This cause a problem widely known as vanishing/exploding gradient problem

Vanishing/Exploding Gradient Problem - Vanilla RNN

Assume we have a hidden state a_t at time step t. If we make things simple and remove biases and inputs, we have

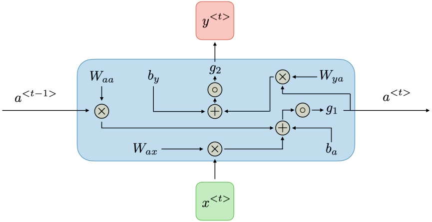
$$a_t = \sigma(w.a_{t-1})$$

we can show that

$$\frac{\partial a_{t_n}}{\partial a_t} = \prod_{i=1}^{i=t_n-t} w.\sigma'(a_{t_n-i})$$

$$= w^{t_n-t} \prod_{i=1}^{i=t_n-t} .\sigma'(a_{t_n-i})$$

The factored w^{t_n-t} is a crucial term.



Vanishing/Exploding Gradient Problem

- If the weight $w^{t_n - t}$ is less than 1, it will make the gradient decay to zero exponentially fast when backpropagating $t_n - t$ time steps

Vanishing/Exploding Gradient Problem

- If the weight $w^{t_n - t}$ is less than 1, it will make the gradient decay to zero exponentially fast when backpropagating $t_n - t$ time steps
- If the weight $w^{t_n - t}$ is greater than 1, it will make the gradient grows exponentially fast when backpropagating $t_n - t$ time steps

Vanishing/Exploding Gradient Problem

- If the weight $w^{t_n - t}$ is less than 1, it will make the gradient decay to zero exponentially fast when backpropagating $t_n - t$ time steps
- If the weight $w^{t_n - t}$ is greater than 1, it will make the gradient grows exponentially fast when backpropagating $t_n - t$ time steps
- A nature question is do not we also have the product-sums of a sigmoid term which can also decay very fast

Vanishing/Exploding Gradient Problem

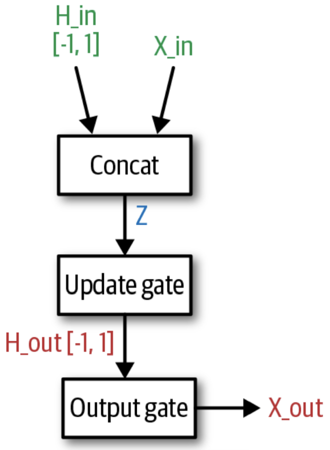
- If the weight $w^{t_n - t}$ is less than 1, it will make the gradient decay to zero exponentially fast when backpropagating $t_n - t$ time steps
- If the weight $w^{t_n - t}$ is greater than 1, it will make the gradient grows exponentially fast when backpropagating $t_n - t$ time steps
- A nature question is do not we also have the product-sums of a sigmoid term which can also decay very fast
- The answer is yes, but if we are able to get rid of this term $w^{t_n - t}$, the decay/growth rate will be lessened.

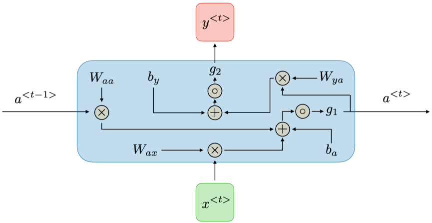
Vanishing/Exploding Gradient Problem

- If the weight $w^{t_n - t}$ is less than 1, it will make the gradient decay to zero exponentially fast when backpropagating $t_n - t$ time steps
- If the weight $w^{t_n - t}$ is greater than 1, it will make the gradient grows exponentially fast when backpropagating $t_n - t$ time steps
- A nature question is do not we also have the product-sums of a sigmoid term which can also decay very fast
- The answer is yes, but if we are able to get rid of this term $w^{t_n - t}$, the decay/growth rate will be lessened.
- Note, the original proof is very mathematically rigor

Outline

Vanilla RNN Two Perspectives





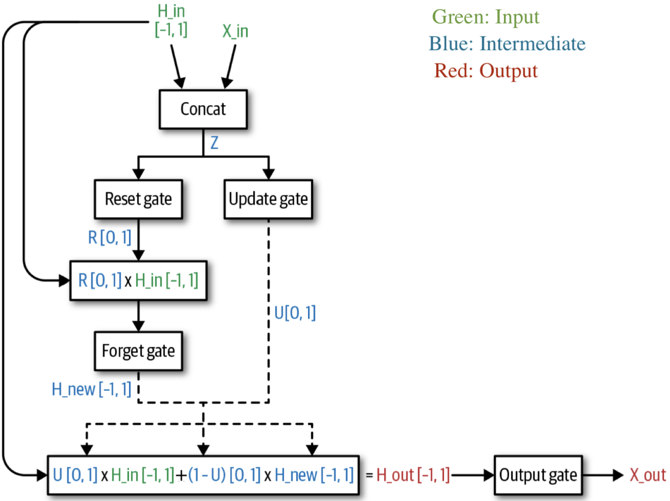
Gate is simply weight multiplication, bias addition, and activation function (sigmoid[0,1]/tanh[-1,1]).

Modifications

We will update the previous architecture by introducing some gates :

- **Update Gate [0,1]:** How much past should matter now?
- **Reset Gate [0,1]:** Reset previous information?
- **Forget Gate:** Erase a cell or not? (Important in LSTM)
- **Output Gate:** How much to reveal to the output?

Gated Recurrent Unit (GRU)



Green: Input
Blue: Intermediate
Red: Output

Back propagation with GRU

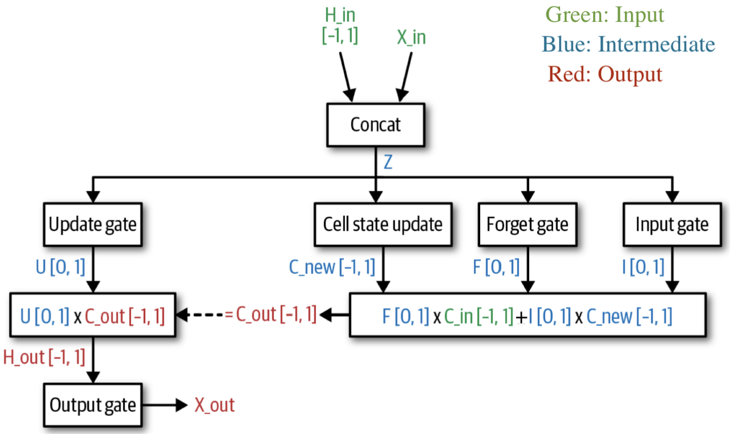
We can show that the gradient received by the hidden state is in the form

$$\frac{\partial s_{t_n}}{\partial s_t} = \prod_{i=1}^{i=t_n-t} \sigma'(.)$$

Long Short Term Memory (LSTM)

- LSTM is the first solution to the vanishing gradient problem
- It is much more complicated than the GRU
- Each LSTM node has two inputs as usual and three outputs
  - Hidden State
  - Cell State
  - Output
- The cell state is meant to encode a kind of aggregation of data from all previous time-steps that have been processed, while the hidden state is meant to encode a kind of characterization of the previous time-step's data.

Long Short Term Memory (LSTM)

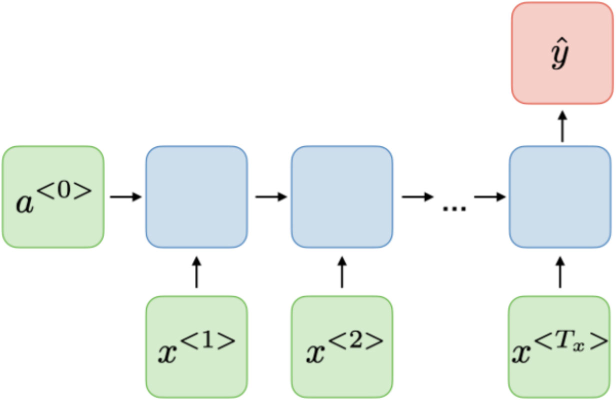


Outline

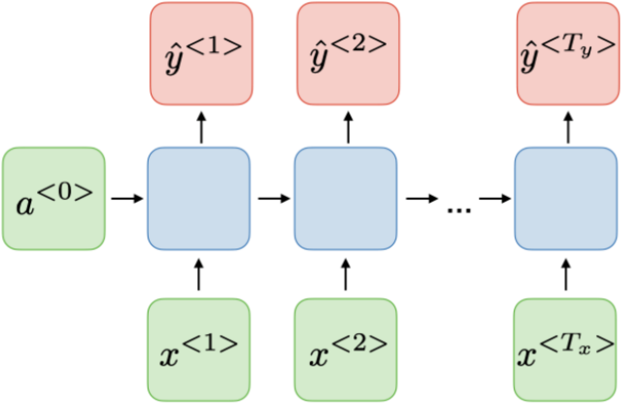
Language Modeling

- Language modeling is one of the most common tasks RNNs are used for.
- A neural network that learns to write text in the style of Shakespeare !!
- The hardest part is building the training data !!
- We can use one hot encoding to represent each word as big vector of zeros and a single one at the word position in the vocabulary dictionary
- Then determining the sequence length, and start building the training data

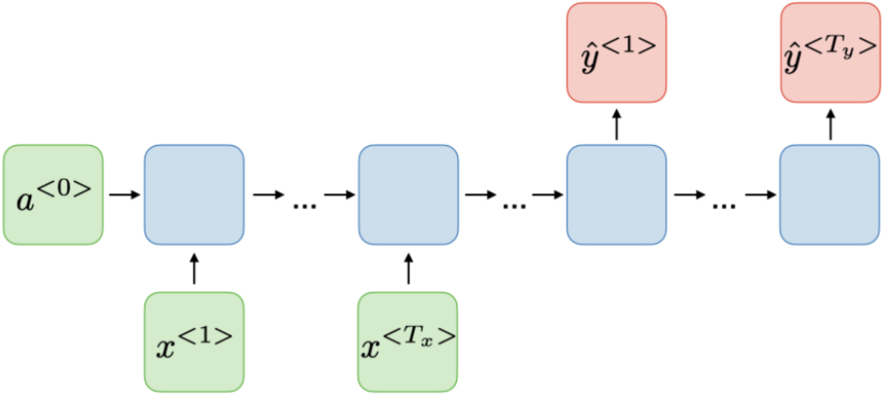
Sentiment Classification



Named Entity Recognition



Machine Translation



Questions