Assignment 4
Building Recurrent Neural Network

1 Objective

- Building forward and backward directions of a basic RNN cell.

2 Problem Statement

Recurrent Neural Networks (RNN) are very effective for Natural Language Processing and other sequence tasks because they have "memory". They can read inputs $x^{\langle t \rangle}$ (such as words) one at a time, and remember some information/context through the hidden layer activations that get passed from one time-step to the next. This allows a uni-directional RNN to take information from the past to process later inputs. A bidirection RNN can take context from both the past and the future.
These are the notations that we will use in the assignment:

- Superscript $[l]$ denotes an object associated with the l^{th} layer.
  Example: $a^{[4]}$ is the 4^{th} layer activation. $W^{[5]}$ and $b^{[5]}$ are the 5^{th} layer parameters.

- Superscript (i) denotes an object associated with the i^{th} example.
  Example: $x^{(i)}$ is the i^{th} training example input.

- Superscript $\langle t \rangle$ denotes an object at the t^{th} time-step.
  Example: $x^{\langle t \rangle}$ is the input x at the t^{th} time-step. $x^{(i)\langle t \rangle}$ is the input at the t^{th} timestep of example i.

- Lowerscript i denotes the i^{th} entry of a vector. Example: $a_i^{[l]}$ denotes the i^{th} entry of the activations in layer l.

2.1 Forward propagation for the basic Recurrent Neural Network

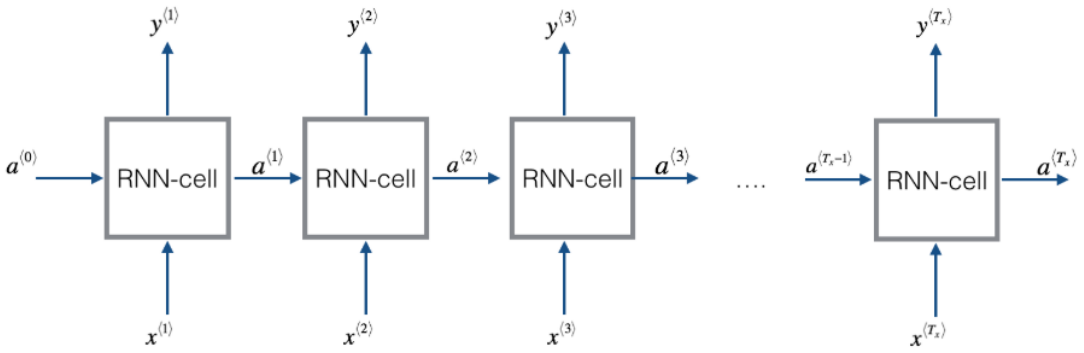


Figure 1

Here's how you can implement an RNN:

1. Implement the calculations needed for one time-step of the RNN.
2. Implement a loop over T_x time-steps in order to process all the inputs, one at a time.

2.1.1 RNN cell

A Recurrent neural network can be seen as the repetition of a single cell. You are first going to implement the computations for a single time-step. The following figure describes the operations for a single time-step of an RNN cell.

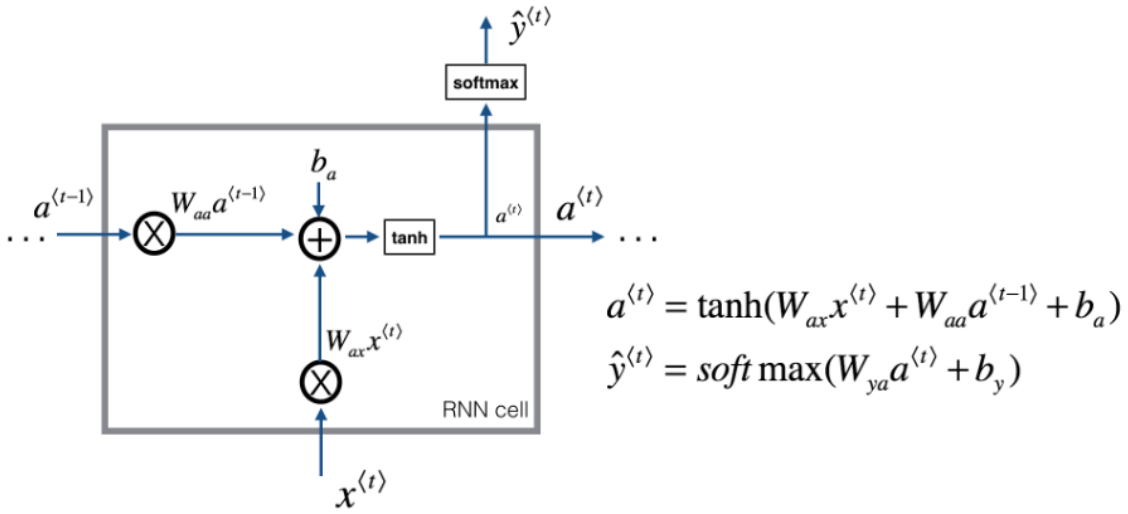


Figure 2

2.1.2 RNN forward pass

You can see an RNN as the repetition of the cell you've just built. If your input sequence of data is carried over 10 time steps, then you will copy the RNN cell 10 times. Each cell takes as input the hidden state from the previous cell ($a^{\langle t-1 \rangle}$) and the current time-step's input data ($x^{\langle t \rangle}$). It outputs a hidden state ($a^{\langle t \rangle}$) and a prediction ($y^{\langle t \rangle}$) for this time-step.

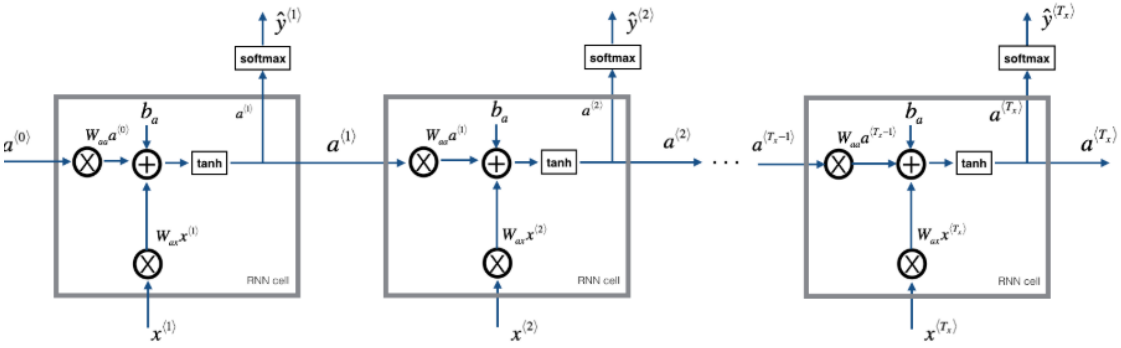


Figure 3

3 Long Short-Term Memory (LSTM) network

This following figure shows the operations of an LSTM-cell

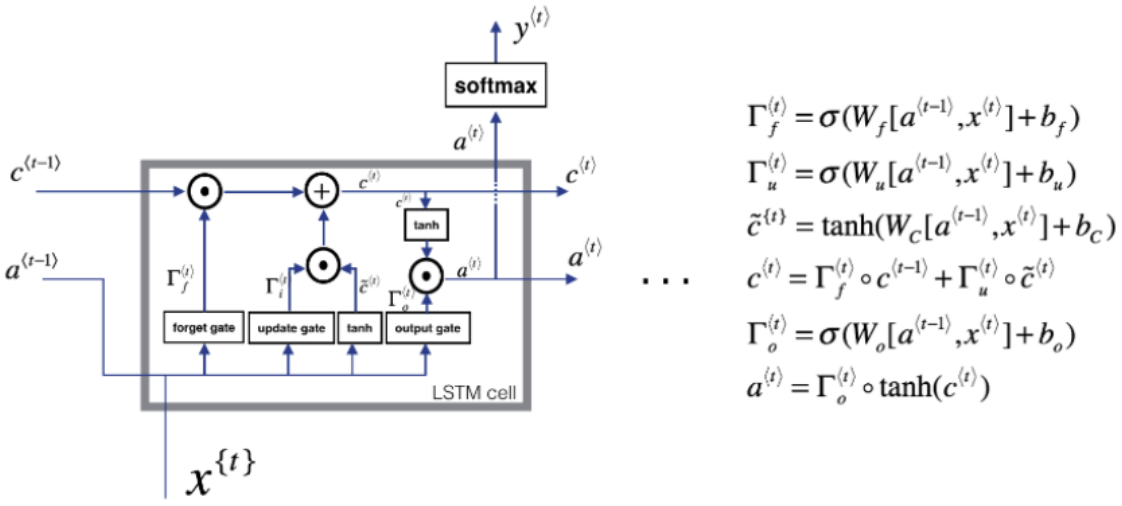


Figure 4

Similar to the RNN example above, you will start by implementing the LSTM cell for a single time-step. Then you can iteratively call it from inside a for-loop to have it process an

input with T_x time-steps.

About the gates

-Forget gate
For the sake of this illustration, lets assume we are reading words in a piece of text, and want use an LSTM to keep track of grammatical structures, such as whether the subject is singular or plural. If the subject changes from a singular word to a plural word, we need to find a way to get rid of our previously stored memory value of the singular/plural state. In an LSTM, the forget gate lets us do this:

$$\Gamma_f^{\langle t \rangle} = \sigma(W_f[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f)$$

Here, W_f are weights that govern the forget gate's behavior. We concatenate $[a^{\langle t-1 \rangle}, x^{\langle t \rangle}]$ and multiply by W_f. The equation above results in a vector $\Gamma_f^{\langle t \rangle}$ with values between 0 and 1. This forget gate vector will be multiplied element-wise by the previous cell state $c^{\langle t-1 \rangle}$. So if one of the values of $\Gamma_f^{\langle t \rangle}$ is 0 (or close to 0) then it means that the LSTM should remove that piece of information (e.g. the singular subject) in the corresponding component of $c^{\langle t-1 \rangle}$. If one of the values is 1, then it will keep the information.

- Update gate
Once we forget that the subject being discussed is singular, we need to find a way to update it to reflect that the new subject is now plural. Here is the formulat for the update gate:

$$\Gamma_u^{\langle t \rangle} = \sigma(W_u[a^{\langle t-1 \rangle}, x^{\{t\}}] + b_u)$$

Similar to the forget gate, here $\Gamma_u^{\langle t \rangle}$ is again a vector of values between 0 and 1. This will be multiplied element-wise with $\tilde{c}^{\langle t \rangle}$, in order to compute $c^{\langle t \rangle}$.

- Updating the cell To update the new subject we need to create a new vector of numbers that we can add to our previous cell state. The equation we use is:

$$\tilde{c}^{\langle t \rangle} = \tanh(W_c[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c)$$

Finally, the new cell state is:
0

$$c^{\langle t \rangle} = \Gamma_f^{\langle t \rangle} * c^{\langle t-1 \rangle} + \Gamma_u^{\langle t \rangle} * \tilde{c}^{\langle t \rangle}$$

- Output gate To decide which outputs we will use, we will use the following two formulas:

$$\Gamma_o^{\langle t \rangle} = \sigma(W_o[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o)$$

$$a^{\langle t \rangle} = \Gamma_o^{\langle t \rangle} * \tanh(c^{\langle t \rangle})$$

Where in equation 5 you decide what to output using a sigmoid function and in equation 6 you multiply that by the tanh of the previous state.

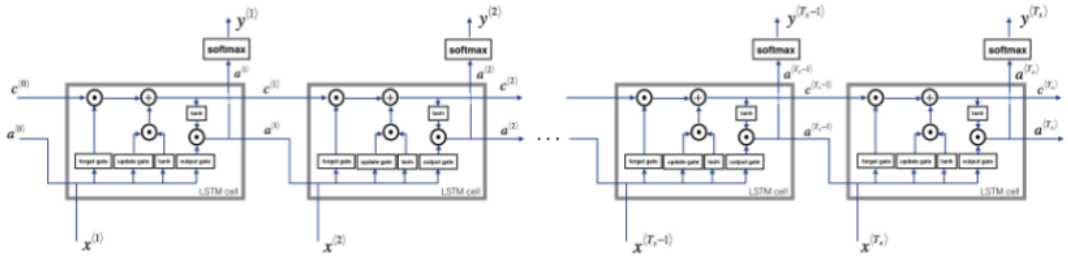
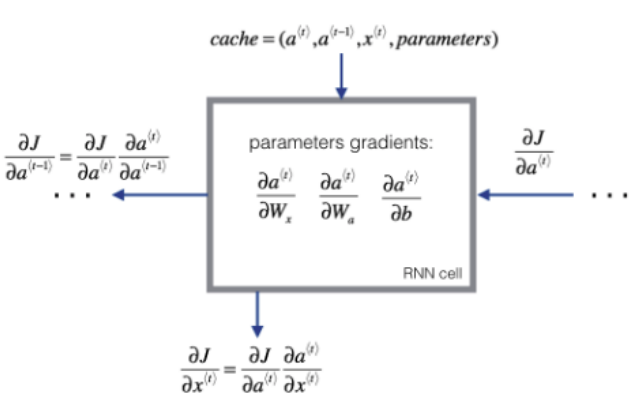


Figure 5

4 Backpropagation in recurrent neural networks

4.1 Basic RNN backward pass

Here we will compute the backward pass for the basic RNN-cell.



$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b)$$
$$\frac{\partial \tanh(x)}{\partial x} = 1 - \tanh(x)^2$$
$$\frac{\partial a^{\langle t \rangle}}{\partial W_{ax}} = (1 - \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b)^2).x^{\langle t \rangle T}$$
$$\frac{\partial a^{\langle t \rangle}}{\partial W_{aa}} = (1 - \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b)^2)a^{\langle t-1 \rangle T}$$
$$\frac{\partial a^{\langle t \rangle}}{\partial b} = \sum_{batch} (1 - \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b)^2)$$
$$\frac{\partial a^{\langle t \rangle}}{\partial x^{\langle t \rangle}} = W_{ax}{}^T.(1 - \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b)^2)$$
$$\frac{\partial a^{\langle t \rangle}}{\partial a^{\langle t-1 \rangle}} = W_{aa}{}^T.(1 - \tanh(W_{ax}x^{\langle t-1 \rangle} + W_{aa}a^{\langle t-1 \rangle} + b)^2)$$

Figure 6

Deriving the one step backward functions: To compute the rnn_cell_backward you need to compute the following equations.

The derivative of tanh is $1 - \tanh(x)^2$. Note that: $\sec(x)^2 = 1 - \tanh(x)^2$

Similarly for $\frac{\partial a^{\langle t \rangle}}{\partial W_{ax}}$, $\frac{\partial a^{\langle t \rangle}}{\partial W_{aa}}$, $\frac{\partial a^{\langle t \rangle}}{\partial b}$, the derivative of $\tanh(u)$ is $(1 - \tanh(u)^2)du$.

The final two equations also follow the same rule and are derived using the tanh derivative. Note that the arrangement is done in a way to get the same dimensions to match.

5 Requirements

- Implement the RNN-cell described in Figure 2.
  - Compute the hidden state with tanh activation: $a^{\langle t \rangle} = \tanh(W_{aa}a^{\langle t-1 \rangle} + W_{ax}x^{\langle t \rangle} + b_a)$.
  - Using your new hidden state $a^{\langle t \rangle}$, compute the prediction $\hat{y}^{\langle t \rangle} = softmax(W_{ya}a^{\langle t \rangle} + b_y)$. We provided you a function: softmax.
  - Store $(a^{\langle t \rangle}, a^{\langle t-1 \rangle}, x^{\langle t \rangle}, parameters)$ in cache
  - Return $a^{\langle t \rangle}$, $y^{\langle t \rangle}$ and cache

  We will vectorize over m examples. Thus, $x^{\langle t \rangle}$ will have dimension (n_x, m), and $a^{\langle t \rangle}$ will have dimension (n_a, m).
  Expected Output:

a_next[4]:	[0.59584544 0.18141802 0.61311866 0.99808218 0.85016201 0.99980978 -0.18887155 0.99815551 0.6531151 0.82872037]
a_next.shape:	(5, 10)
yt[1]:	[0.9888161 0.01682021 0.21140899 0.36817467 0.98988387 0.88945212 0.36920224 0.9966312 0.9982559 0.17746526]
yt.shape:	(2, 10)

- Code the forward propagation of the RNN described in Figure 3.
  - Create a vector of zeros (a) that will store all the hidden states computed by the RNN.
  - Initialize the "next" hidden state as a_0 (initial hidden state).
  - Start looping over each time step, your incremental index is t :
    - Update the "next" hidden state and the cache by running rnn_cell_forward
    - Store the "next" hidden state in a (t^{th} position)
    - Store the prediction in y
    - Add the cache to the list of caches
  - Return a, y and caches.

Expected Output:

a[4][1]:	[-0.99999375 0.77911235 -0.99861469 -0.99833267]
a.shape:	(5, 10, 4)
y[1][3]:	[0.79560373 0.86224861 0.11118257 0.81515947]
y.shape:	(2, 10, 4)
cache[1][1][3]:	[-1.1425182 -0.34934272 -0.20889423 0.58662319]
len(cache):	2

- Implement the LSTM cell described in the Figure 4.

  1. Concatenate $a^{\langle t-1 \rangle}$ and $x^{\langle t \rangle}$ in a single matrix: $concat = \begin{bmatrix} a^{\langle t-1 \rangle} \\ x^{\langle t \rangle} \end{bmatrix}$
  2. Compute all the formulas 1-6. You can use sigmoid() (provided) and np.tanh().
  3. Compute the prediction $y^{\langle t \rangle}$. You can use softmax() (provided).
     Expected Output:

a_next[4]:	[-0.66408471 0.0036921 0.02088357 0.22834167 -0.85575339 0.00138482 0.76566531 0.34631421 -0.00215674 0.43827275]
a_next.shape:	(5, 10)
c_next[2]:	[0.63267805 1.00570849 0.35504474 0.20690913 -1.64566718 0.11832942 0.76449811 -0.0981561 -0.74348425 -0.26810932]
c_next.shape:	(5, 10)
yt[1]:	[0.79913913 0.15986619 0.22412122 0.15606108 0.97057211 0.31146381 0.00943007 0.12666353 0.39380172 0.07828381]
yt.shape:	(2, 10)
cache[1][3]:	[-0.16263996 1.03729328 0.72938082 -0.54101719 0.02752074 -0.30821874 0.07651101 -1.03752894 1.41219977 -0.37647422]
len(cache):	10

- Implement lstm_forward() to run an LSTM over T_x time-steps, Note that $c^{\langle 0 \rangle}$ is initialized with zeros.
  Expected Output:

a[4][3][6] =	0.172117767533
a.shape =	(5, 10, 7)
y[1][4][3] =	0.95087346185
y.shape =	(2, 10, 7)
caches[1][1][1] =	[0.82797464 0.23009474 0.76201118 -0.22232814 -0.20075807 0.18656139 0.41005165]
c[1][2][1] =	-0.855544916718
len(caches) =	2

- Implement rnn_cell_backward() to run an LSTM over T_x time-steps, Note that $c^{\langle 0 \rangle}$ is initialized with zeros.
  Expected Output:

gradients["dxt"][1][2] =	-0.460564103059
gradients["dxt"].shape =	(3, 10)
gradients["da_prev"][2][3] =	0.0842968653807
gradients["da_prev"].shape =	(5, 10)
gradients["dWax"][3][1] =	0.393081873922
gradients["dWax"].shape =	(5, 3)
gradients["dWaa"][1][2] =	-0.28483955787
gradients["dWaa"].shape =	(5, 5)
gradients["dba"][4] =	[0.80517166]
gradients["dba"].shape =	(5, 1)

- Implement the rnn_backward function. Initialize the return variables with zeros first and then loop through all the time steps while calling the rnn_cell_backward at each time timestep, update the other variables accordingly.

Expected Output:

gradients["dx"][1][2] =	[-2.07101689 -0.59255627 0.02466855 0.01483317]
gradients["dx"].shape =	(3, 10, 4)
gradients["da0"][2][3] =	-0.314942375127
gradients["da0"].shape =	(5, 10)
gradients["dWax"][3][1] =	11.2641044965
gradients["dWax"].shape =	(5, 3)
gradients["dWaa"][1][2] =	2.30333312658
gradients["dWaa"].shape =	(5, 5)
gradients["dba"][4] =	[-0.74747722]
gradients["dba"].shape =	(5, 1)

6 BONUS: Sentiment Analysis

Sentiment analysis is one of the most common applications in Natural Language processing. So here we are, we will train a classifier for movie reviews in IMDB data set, using TF.Keras Recurrent Neural Networks.

1. Load the IMDB data from Keras
2. Build the model.
3. Decide the percentage of the training data that you will use as validation
4. Train the model, then evaluate it.
5. Report the best accuracy you got in each of the 3 considered alternatives.

7 Notes

- Parts of this assignment are based on Andrew Ng Stanford deep learning course.
- The dataset files and the starter code for the problems can be found in the resources section in Piazza.

- Cheating will be severely penalized (for both parties). So, it is better to deliver nothing than deliver a copy. Any online resources used must be clearly identified.