Recurrent Neural Network

TINGWU WANG,

MACHINE LEARNING GROUP,

UNIVERSITY OF TORONTO

FOR CSC 2541, SPORT ANALYTICS

Contents

Part One

Why do we need Recurrent Neural Network?

1. What Problems are Normal CNNs good at?
2. What is Sequence Learning?
3. Ways to Deal with Sequence Labeling.

1. What Problems are CNNs normally good at?

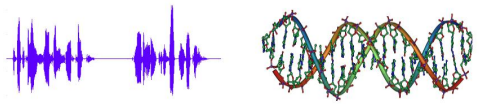
1. Image classification as a naive example
   1. Input: one image.
   2. Output: the probability distribution of classes.
   3. You need to provide one guess (output), and to do that you only need to look at one image (input).

P(Cat|image) = 0.1

P(Panda|image) = 0.9

2. What is Sequence Learning?

1. Sequence learning is the study of machine learning algorithms designed for sequential data [1].



2. Language model is one of the most interesting topics that use sequence labeling.
    1. Language Translation
        1. Understand the meaning of each word, and the relationship between words
        2. Input: one sentence in German

           input = "Ich will **stark** Steuern senken"
        3. Output: one sentence in English

           output = "I want to cut taxes **bigly**" (**big league**?)

2. What is Sequence Learning?

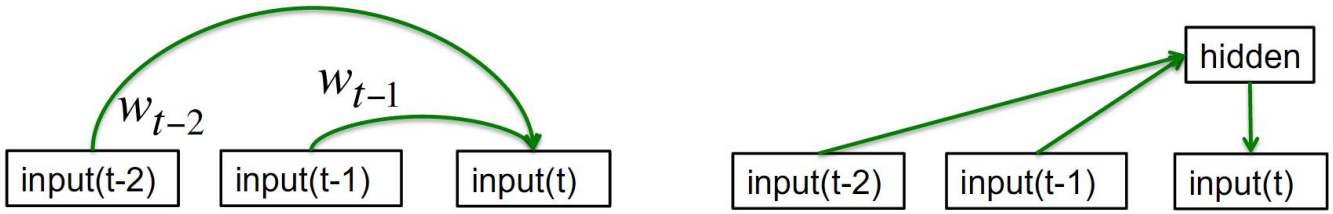
1. To make it easier to understand why we need RNN, let's think about a simple speaking case (let's violate neuroscience a little bit)
    1. We are given a hidden state (free mind?) that encodes all the information in the sentence we want to speak.
    2. We want to generate a list of words (sentence) in an one-by-one fashion.
        1. At each time step, we can only choose a single word.
        2. The hidden state is affected by the words chosen (so we could remember what we just say and complete the sentence).

2. What is Sequence Learning?

1. Plain CNNs are not born good at length-varying input and output.
    1. Difficult to define input and output
        1. Remember that
            1. Input image is a 3D tensor (width, length, color channels)
            2. Output is a distribution on fixed number of classes.
        2. Sequence could be:
            1. "I know that you know that I know that you know that I know that you know that I know that you know that I know that you know that I know that you know that I don't know"
            2. "I don't know"
    2. Input and output are strongly correlated within the sequence.
    3. Still, people figured out ways to use CNN on sequence learning (e.g. [8]).

3. Ways to Deal with Sequence Labeling

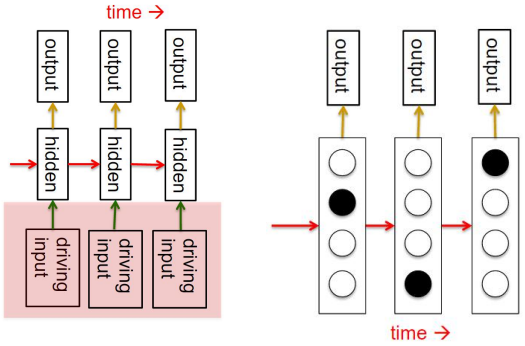
1. Autoregressive models
   1. Predict the next term in a sequence from a fixed number of previous terms using delay taps.
2. Feed-forward neural nets
   1. These generalize autoregressive models by using one or more layers of non-linear hidden units



Memoryless models: limited word-memory window; hidden state cannot be used efficiently.

3. Ways to Deal with Sequence Labeling

1. Linear Dynamical Systems
   1. These are generative models. They have a real-valued hidden state that cannot be observed directly.
2. Hidden Markov Models
   1. Have a discrete one-of-N hidden state. Transitions between states are stochastic and controlled by a transition matrix. The outputs produced by a state are stochastic.

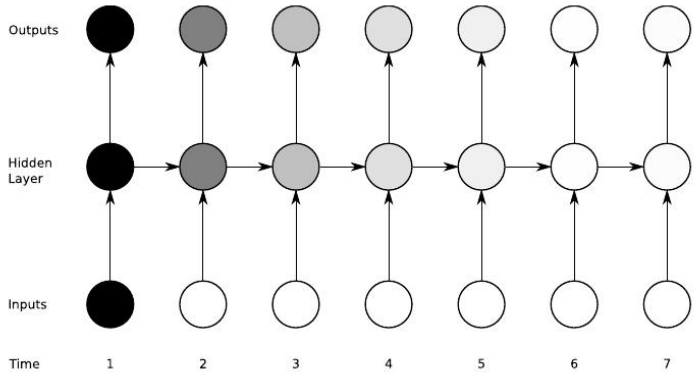


Memoryful models,

time-cost to infer the hidden state distribution.

3. Ways to Deal with Sequence Labeling

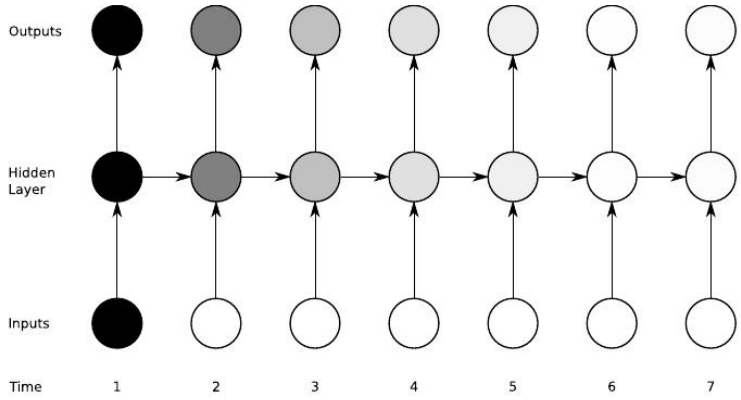
1. Finally, the RNN model!
   1. Update the hidden state in a deterministic nonlinear way.
   2. In the simple speaking case, we send the chosen word back to the network as input.



materials from [4]

3. Ways to Deal with Sequence Labeling

1. RNNs are very powerful, because they:
   1. Distributed hidden state that allows them to store a lot of information about the past efficiently.
   2. Non-linear dynamics that allows them to update their hidden state in complicated ways.
   3. No need to infer hidden state, pure deterministic.
   4. Weight sharing



Part Two

Math in a Vanilla Recurrent Neural Network

1. Vanilla Forward Pass
2. Vanilla Backward Pass
3. Vanilla Bidirectional Pass
4. Training of Vanilla RNN
5. Vanishing and exploding gradient problems

1.Vanilla Forward Pass

1. The forward pass of a vanilla RNN
   1. The same as that of an MLP with a single hidden layer
   2. Except that activations arrive at the hidden layer from both the current external input and the hidden layer activations one step back in time.

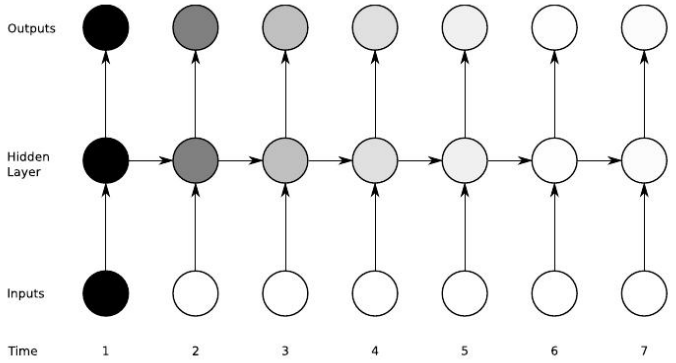
2. For the input to hidden units we have

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$b_h^t = \theta_h(a_h^t)$$

3. For the output unit we have

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$



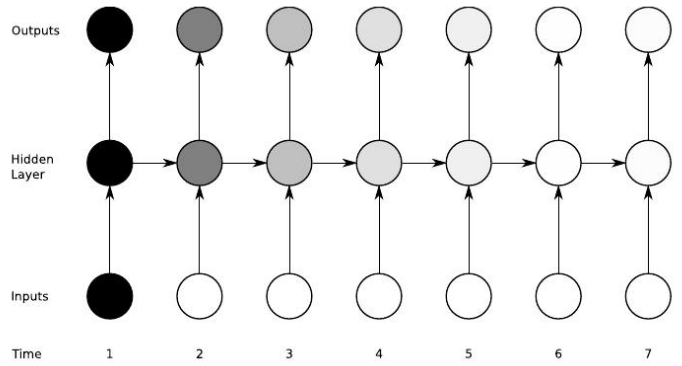
1.Vanilla Forward Pass

1. The complete sequence of hidden activations can be calculated by starting at t = 1 and recursively applying the three equations, incrementing t at each step.

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$b_h^t = \theta_h(a_h^t)$$

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$



2.Vanilla Backward Pass

1. Given the partial derivatives of the objective function with respect to the network outputs, we now need the derivatives with respect to the weights.
2. We focus on BPTT since it is both conceptually simpler and more efficient in computation time (though not in memory). Like standard back-propagation, BPTT consists of a repeated application of the chain rule.

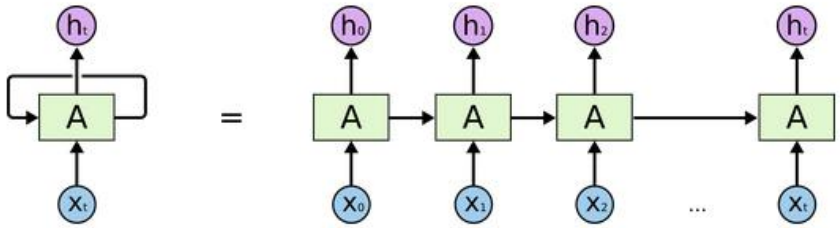
$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$b_h^t = \theta_h(a_h^t)$$

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

2.Vanilla Backward Pass

1. Back-propagation through time
   1. Don't be fooled by the fancy name. It's just the standard back-propagation.



An unrolled recurrent neural network.

$$\delta_h^t = \theta'(a_h^t)\left(\sum_{k=1}^{K}\delta_k^t w_{hk} + \sum_{h'=1}^{H}\delta_{h'}^{t+1} w_{hh'}\right),$$

$$\delta_j^t \stackrel{\text{def}}{=} \frac{\partial O}{\partial a_j^t}$$

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$b_h^t = \theta_h(a_h^t)$$

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

materials from [6]

2.Vanilla Backward Pass

1. Back-propagation through time
   1. The complete sequence of delta terms can be calculated by starting at t = T and recursively applying the below functions, decrementing t at each step.
   2. Note that $\delta_j^{T+1} = 0 \, \forall j$, since no error is received from beyond the end of the sequence.

$$\delta_h^t = \theta'(a_h^t)\left(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'}\right), \qquad a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$b_h^t = \theta_h(a_h^t)$$

$$\delta_j^t \stackrel{\text{def}}{=} \frac{\partial O}{\partial a_j^t} \qquad a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

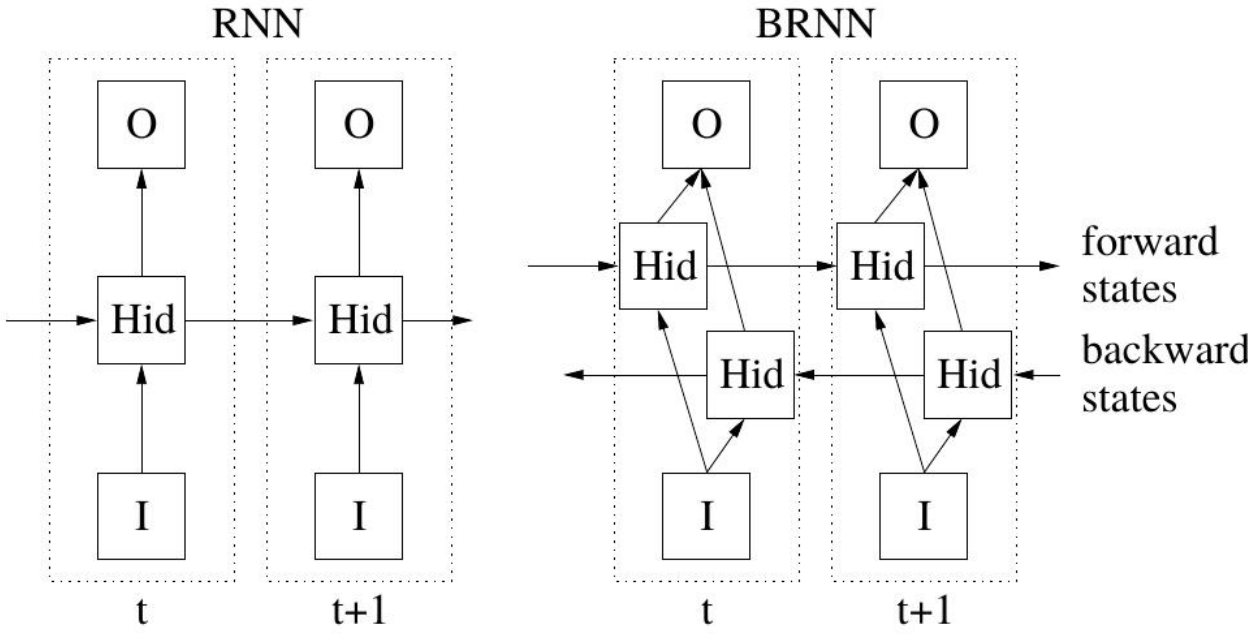
   3. Finally, bearing in mind that the weights to and from each unit in the hidden layer are the same at every time-step, we sum over the whole sequence to get the derivatives with respect to each of the network weights

$$\frac{\partial O}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial O}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t b_i^t$$

materials from [4]

3.Vanilla Bidirectional Pass

1. For many sequence labeling tasks, we would like to have access to future.



3.Vanilla Bidirectional Pass

1. Algorithm looks like this

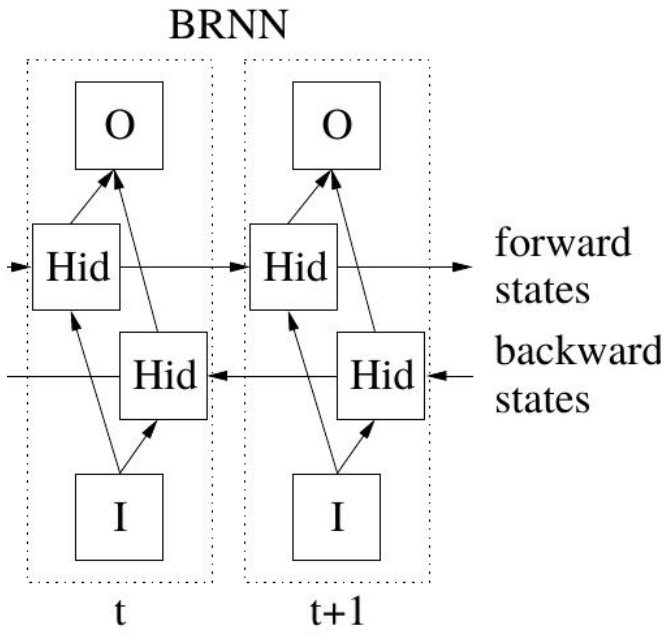
for $t = 1$ to T **do**
Do forward pass for the forward hidden layer, storing activations at each timestep
for $t = T$ to 1 **do**
Do forward pass for the backward hidden layer, storing activations at each timestep
for $t = 1$ to T **do**
Do forward pass for the output layer, using the stored activations from both hidden layers

Algorithm 3.1: BRNN Forward Pass

Similarly, the backward pass proceeds as for a standard RNN trained with BPTT, except that all the output layer δ terms are calculated first, then fed back to the two hidden layers in opposite directions:

for $t = T$ to 1 **do**
Do BPTT backward pass for the output layer only, storing δ terms at each timestep
for $t = T$ to 1 **do**
Do BPTT backward pass for the forward hidden layer, using the stored δ terms from the output layer
for $t = 1$ to T **do**
Do BPTT backward pass for the backward hidden layer, using the stored δ terms from the output layer

Algorithm 3.2: BRNN Backward Pass



4.Training of Vanilla RNN

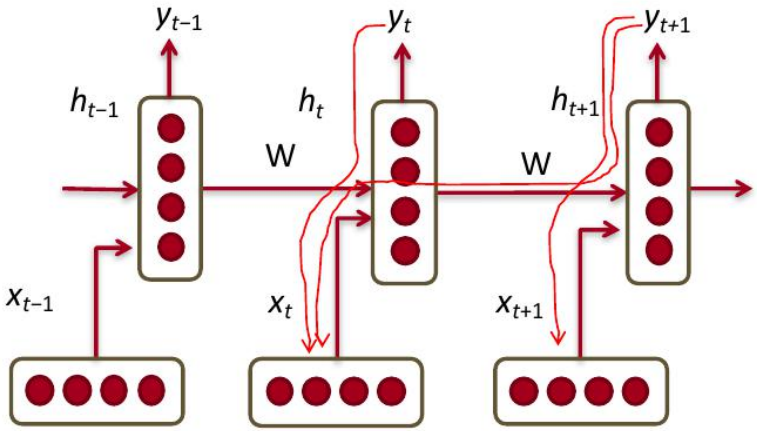
1. So far we have discussed how RNN can be differentiated with respect to suitable objective functions, and thereby they could be trained with any gradient-descent based algorithm
   1. just treat them as a normal CNN

$$\Delta \mathbf{w}(n) = -\alpha \frac{\partial O}{\partial \mathbf{w}(n)}$$

2. One of the great things about RNN: lots of engineering choices
   1. Preprocessing and postprocessing

5.Vanishing and exploding gradient problems

1. Multiply the same matrix at each time step during back-prop



5.Vanishing and exploding gradient problems

1. Toy example how gradient vanishes
   1. Similar but simpler RNN formulation:

$$h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \text{diag}[f'(h_{j-1})]$$

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|\text{diag}[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_W \beta_h)^{t-k}$$

2. Solutions?
   1. For vanishing gradients: Initialization + ReLus
   2. Trick for exploding gradient: clipping trick

Part Three

From Vanilla to LSTM

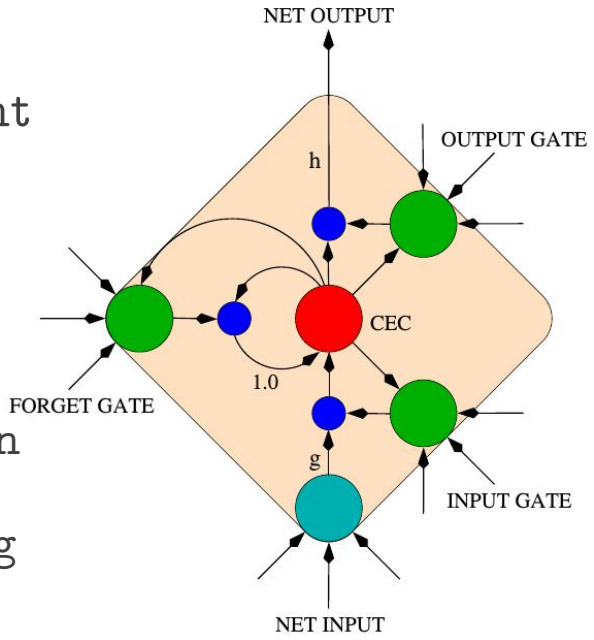
1. Definition
2. Forward Pass
3. Backward Pass

1. Definition

1. As discussed earlier, for standard RNN architectures, the range of context that can be accessed is limited.
    1. The problem is that the influence of a given input on the hidden layer, and therefore on the network output, either decays or blows up exponentially as it cycles around the network's recurrent connections.
2. The most effective solution so far is the Long Short Term Memory (LSTM) architecture (Hochreiter and Schmidhuber, 1997).
3. The LSTM architecture consists of a set of recurrently connected subnets, known as memory blocks. These blocks can be thought of as a differentiable version of the memory chips in a digital computer. Each block contains one or more self-connected memory cells and three multiplicative units that provide continuous analogues of write, read and reset operations for the cells
    1. The input, output and forget gates.

1. Definition

1. The multiplicative gates allow LSTM memory cells to store and access information over long periods of time, thereby avoiding the vanishing gradient problem
    1. For example, as long as the input gate remains closed (i.e. has an activation close to 0), the activation of the cell will not be overwritten by the new inputs arriving in the network, and can therefore be made available to the net much later in the sequence, by opening the output gate.



1. Definition

1. Comparison

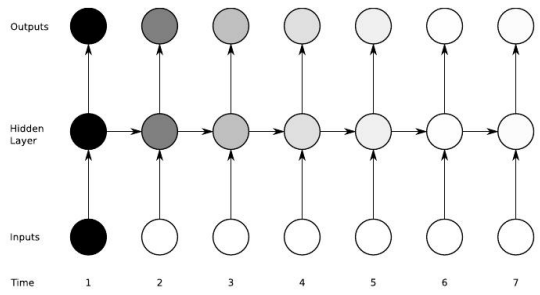


Figure 4.1: **Vanishing gradient problem for RNNs.** The shading of the nodes indicates the sensitivity over time of the network nodes to the input at time one (the darker the shade, the greater the sensitivity). The sensitivity decays exponentially over time as new inputs overwrite the activation of hidden unit and the network 'forgets' the first input.

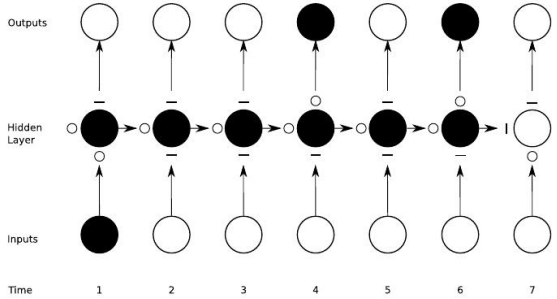


Figure 4.3: **Preservation of gradient information by LSTM.** As in Figure 4.1 the shading of the nodes indicates their sensitivity to the input unit at time one. The state of the input, forget, and output gate states are displayed below, to the left and above the hidden layer node, which corresponds to a single memory cell. For simplicity, the gates are either entirely open ('O') or closed ('—'). The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed, and the sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

2. Forward Pass

1. Basically very similar to the vanilla RNN forward pass
   1. But it's a lot more complicated
   2. Can you do the backward pass by yourself?
      1. **Quick quiz**, get a white sheet of paper. Write your student number and name.
      2. We are going to give you 10 minutes
      3. No discussion
      4. 10% of your final grades

Input Gates

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1}$$
$$b_\iota^t = f(a_\iota^t)$$

Forget Gates

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1}$$
$$b_\phi^t = f(a_\phi^t)$$

Cells

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1}$$
$$s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t)$$

Output Gates

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t$$
$$b_\omega^t = f(a_\omega^t)$$

Cell Outputs

$$b_c^t = b_\omega^t h(s_c^t)$$

3. Backward Pass

1. Just kidding! The math to get the backward pass should be very similar to the one used in vanilla RNN backward pass
    1. But it's a lot more complicated, too...
    2. We are not going to derive that in the class

$$\epsilon_c^t \stackrel{\text{def}}{=} \frac{\partial O}{\partial b_c^t} \qquad \epsilon_s^t \stackrel{\text{def}}{=} \frac{\partial O}{\partial s_c^t}$$

Cell Outputs

$$\epsilon_c^t = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{h=1}^{H} w_{ch}\delta_h^{t+1}$$

Output Gates

$$\delta_\omega^t = f'(a_\omega^t)\sum_{c=1}^{C} h(s_c^t)\epsilon_c^t$$

States

$$\epsilon_s^t = b_\omega^t h'(s_c^t)\epsilon_c^t + b_\phi^{t+1}\epsilon_s^{t+1} + w_{c\iota}\delta_\iota^{t+1} + w_{c\phi}\delta_\phi^{t+1} + w_{c\omega}\delta_\omega^t$$

Cells

$$\delta_c^t = b_\iota^t g'(a_c^t)\epsilon_s^t$$

Forget Gates

$$\delta_\phi^t = f'(a_\phi^t)\sum_{c=1}^{C} s_c^{t-1}\epsilon_s^t$$

Input Gates

$$\delta_\iota^t = f'(a_\iota^t)\sum_{c=1}^{C} g(a_c^t)\epsilon_s^t$$

Part Four

Miscellaneous

1. More than Language Model
2. GRU

1. More than Language Model

1. Like I said, RNN could do a lot more than modeling language
    1. Drawing pictures:

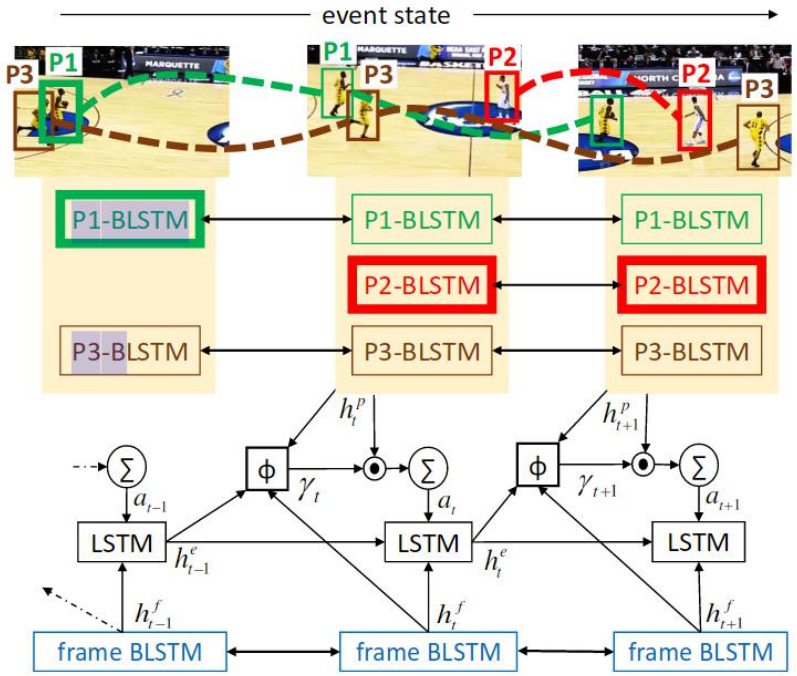
       [9] DRAW: A Recurrent Neural Network For Image Generation
    2. Computer-composed music

       [10] Song From PI: A Musically Plausible Network for Pop Music Generation
    3. Semantic segmentation

       [11] Conditional random fields as recurrent neural networks

1. More than Language Model

1. RNN in sports
   1. Sport is a sequence of event (sequence of images voices)
   2. Detecting events and key actors in multi-person videos [12]
      1. "In particular, we track people in videos and use a recurrent neural network (RNN) to represent the track features. We learn time-varying attention weights to combine these features at each time-instar The attended features are then processed using another RNN for event detection/classification"



1. More than Language Model

1. RNN in sports
    1. Applying Deep Learning to Basketball Trajectories
        1. This paper applies recurrent neural networks in the form of sequence modeling to predict whether a three-point shot is successful [13]

    2. Action Classification in Soccer Videos with Long Short-Term Memory Recurrent Neural Networks [14]

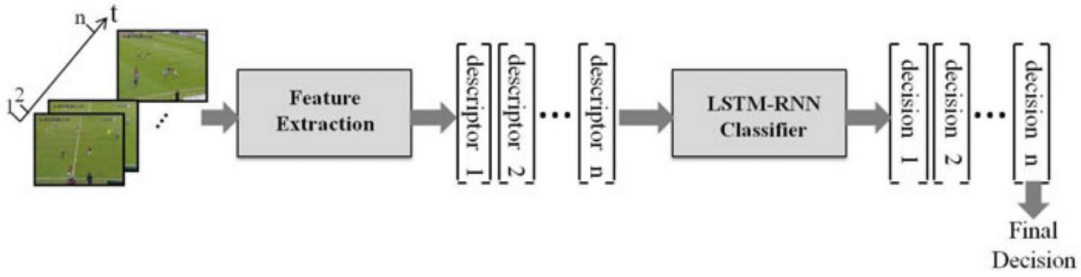
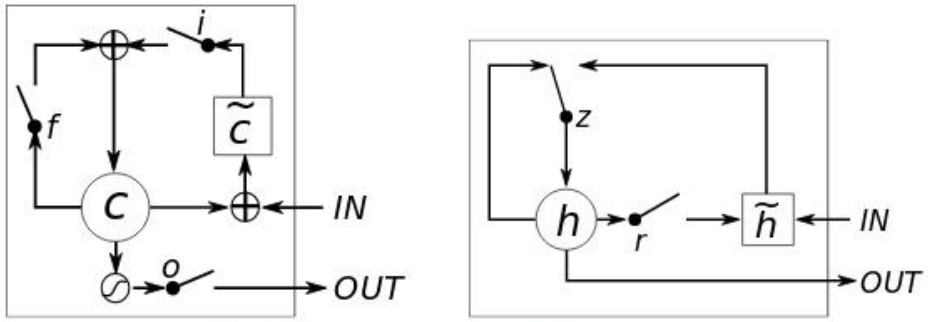


Fig. 1. Proposed classification scheme

2. GRU

1. A new type of RNN cell (Gated Feedback Recurrent Neural Networks)
    1. Very similar to LSTM
    2. It merges the cell state and hidden state.
    3. It combines the forget and input gates into a single "update gate".
    4. Computationally more efficient.
        1. less parameters, less complex structure.
2. Gaining popularity nowadays [15,16]



(a) Long Short-Term Memory (b) Gated Recurrent Unit

Part Five

Implementing RNN in Tensorflow

1. Implementing RNN in Tensorflow

1. The best way should be reading the docs on Tensorflow website [17].
2. Let's assume you already manage how to use CNN in Tensorflow (toy sequence decoder model)

```
lstm = rnn_cell.BasicLSTMCell(lstm_size)
# Initial state of the LSTM memory.
state = tf.zeros([batch_size, lstm.state_size])
probabilities = []
loss = 0.0
for current_batch_of_words in words_in_dataset:
    # The value of state is updated after processing each batch of words.
    output, state = lstm(current_batch_of_words, state)

    # The LSTM output can be used to make next word predictions
    logits = tf.matmul(output, softmax_w) + softmax_b
    probabilities.append(tf.nn.softmax(logits))
    loss += loss_function(probabilities, target_words)
```

1. Implementing RNN in Tensorflow

1. Standard feed dictionary just like other CNN models in Tensorflow

```
# A numpy array holding the state of LSTM after each batch of words.
numpy_state = initial_state.eval()
total_loss = 0.0
for current_batch_of_words in words_in_dataset:
    numpy_state, current_loss = session.run([final_state, loss],
        # Initialize the LSTM state from the previous iteration.
        feed_dict={initial_state: numpy_state, words: current_batch_of_words})
    total_loss += current_loss
```

2. Word-embedding (make the words in the sentence understandable by the program)

```
# embedding_matrix is a tensor of shape [vocabulary_size, embedding size]
word_embeddings = tf.nn.embedding_lookup(embedding_matrix, word_ids)
```

1. Implementing RNN in Tensorflow

1. Simple example using Tensorflow.
    1. The task: let the robot learn the atom behavior it should do, by following human instructions
    2. The result we could get by using RNN.

2. Task:
    1. Input: "Sit down on the couch and watch T.V. When you are done watching television turn it off. Put the pen on the table. Toast some bread in the toaster and get a knife to put butter on the bread while you sit down at the table."

1. Implementing RNN in Tensorflow

1. Simple demo result.

http://www.cs.toronto.edu/~tingwuwang/outputscript_synthetic_data_clean_is_rnn_encoder_True_decoder_dim_150_model.ckpt.html

GT

1. Walk , TELEVISION , None
2. SwitchOn , TELEVISION , None
3. Walk , SOFA , None
4. Sit , SOFA , None
5. Watch , TELEVISION , None
6. StandUp , None , None
7. Walk , TELEVISION , None
8. SwitchOff , TELEVISION , None
9. Walk , PEN , None
10. Grab , PEN , None
11. Walk , TABLE , None
12. PutBack , PEN , TABLE
13. Walk , KNIFE , None
14. Grab , KNIFE , None
15. Walk , TOASTER , None
16. SwitchOn , TOASTER , None
17. SwitchOff , TOASTER , None
18. Walk , TABLE , None
19. PutBack , KNIFE , TABLE

PRED

1. Walk , TELEVISION , None
2. SwitchOn , TELEVISION , None
3. Walk , SOFA , None
4. Sit , SOFA , None
5. Watch , TELEVISION , None
6. StandUp , None , None
7. Walk , TELEVISION , None
8. SwitchOff , TELEVISION , None
9. Walk , PEN , None
10. Grab , PEN , None
11. Walk , TABLE , None
12. PutBack , PEN , TABLE
13. Walk , TOASTER , None
14. SwitchOn , TOASTER , None
15. SwitchOff , TOASTER , None
16. Walk , KNIFE , None
17. Grab , KNIFE , None
18. Walk , TABLE , None
19. PutBack , KNIFE , TABLE

References

Most of the materials in the slides come from the following tutorials / lecture slides:

[1] Machine Learning I Week 14: Sequence Learning Introduction, Alex Graves, Technische Universitaet Muenchen.

[2] CSC2535 2013: Advanced Machine Learning, Lecture 10: Recurrent neural networks, Geoffrey Hinton, University of Toronto.

[3] CS224d Deep NLP, Lecture 8: Recurrent Neural Networks, Richard Socher, Stanford University.

[4] Supervised Sequence Labelling with Recurrent Neural Networks, Alex Graves, Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigten Dissertation.

[5] The Unreasonable Effectiveness of Recurrent Neural Networks, Andrej Karpathy, blog About Hacker's guide to Neural Networks.

[6] Understanding LSTM Networks, Christopher Olah, github blog.

Other references

[7] Kiros, Ryan, et al. "Skip-thought vectors." Advances in neural information processing systems. 2015.

[8] Dauphin, Yann N., et al. "Language Modeling with Gated Convolutional Networks." arXiv preprint arXiv:1612.08083 (2016).

[9] Gregor, Karol, et al. "DRAW: A recurrent neural network for image generation." arXiv preprint arXiv:1502.04623 (2015).

References

[10] Chu, Hang, Raquel Urtasun, and Sanja Fidler. "Song From PI: A Musically Plausible Network for Pop Music Generation." arXiv preprint arXiv:1611.03477 (2016).

[11] Zheng, Shuai, et al. "Conditional random fields as recurrent neural networks." Proceedings of the IEEE International Conference on Computer Vision. 2015.

[12] Ramanathan, Vignesh, et al. "Detecting events and key actors in multi-person videos." arXiv preprint arXiv:1511.02917 (2015).

[13] Shah, Rajiv, and Rob Romijnders. "Applying Deep Learning to Basketball Trajectories." arXiv preprint arXiv:1608.03793 (2016).

[14] Baccouche, Moez, et al. "Action classification in soccer videos with long short-term memory recurrent neural networks." International Conference on Artificial Neural Networks. Springer Berlin Heidelberg, 2010.

[15] Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." arXiv preprint arXiv:1412.3555 (2014).

[16] Chung, Junyoung, et al. "Gated feedback recurrent neural networks." CoRR, abs/1502.02367 (2015).

[17] Tutorials on Tensorflow. https://www.tensorflow.org/tutorials/

Q&A

Thank you for listening ;P