Review Recurrent Neural Networks

Deep averaging models are the simplest form of sequence-to-sequence model. However, one weakness of deep averaging networks is that they ignore the ordering of the input text. For instance, the decoder of a deep averaging network will generate the same code vector for the sentence "John enjoys eating the shark" and the sentence "The shark enjoys eating John." Both sentences are very different (the second one probably has a more negative sentiment than the first), and we need more complex neural networks to encode this sequential nature of text.

☆ **Key Points**

A recurrent neural network (RNN) is a neural network that is applied at each element of the sequence but which keeps track of a hidden state vector and uses it to encode and decode sequential information.

Use stochastic gradient descent (SGD) to train recurrent neural networks.

This is where recurrent neural networks (RNNs) come into play. In essence, a recurrent neural network keeps track of a hidden state variable and uses this hidden state variable to encode sequential information.

Concretely, the simplest recurrent neural network consists of two learned parameters, $\mathbf{U}$ and $\mathbf{W}$. Suppose you have a sequence of inputs $\boldsymbol{s_1, s_2, s_3, \ldots, s_n}$ (this can be a sentence where each $\boldsymbol{s_i}$ is the embedding of a word in the sentence) and an initial hidden state $\mathbf{h_0}$.

Computing Output of a Recurrent Neural Network

A recurrent neural network essentially does the following for each input $\boldsymbol{s_i}$ in the sequence:

1. We process the previous hidden state $\mathbf{h_{i-1}}$. This is done with matrix multiplication to yield $\mathbf{W h_{i-1}}$. This represents the "history" component of the sentence before the word $\boldsymbol{s_i}$.
2. Process the current input $\boldsymbol{s_i}$ by using matrix multiplication to yield $\mathbf{U} \boldsymbol{s_i}$. This represents the information from the current input $\boldsymbol{s_i}$.
3. Add up the the output of the previous two steps to yield $\mathbf{W h_{i-1}} + \mathbf{U} \boldsymbol{s_i}$.
4. Apply a nonlinear transition function $\boldsymbol{\sigma}$ (such as ReLU) to the output of (3) to yield the hidden state $\mathbf{h_i} = \boldsymbol{\sigma}\left(\mathbf{W h_{i-1}} + \mathbf{U} \boldsymbol{s_i}\right)$.
5. (Optional) Depending on the application, we can further process the hidden state variable $\boldsymbol{o_i}$.

The final hidden state can be viewed as the vector representation of the whole sequence.

Intuitively, think of the hidden state as a temporary representation of all the previous words and this hidden state will determine how to process the current input. Having this hidden state allows the neural network to "remember" the sequence and encode the sequential information in the sequence.

Training Recurrent Neural Networks

Training of recurrent neural networks can be done using stochastic gradient descent (SGD). However, the output of the RNN can be used in a variety of ways, which impacts the way it should be trained.

For instance, in machine translation, we want to have a model that can take in an English sentence and output a German sentence. We can use one RNN as our encoder to encode the English sentence into a vector representation $\mathbf{h}$, then have another RNN as our decoder that decodes $\mathbf{h}$ into a German sentence. The encoder is straightforward: we pass in a sequence of embedded English words and follow the above steps.

However, for the decoder, we pass in a vector (the encoded English sentence) instead of a sequence of words. How can we use an RNN if the input is a vector and not a sequence of word embeddings? The key is to use $\mathbf{h}$ as the initial hidden state and input a starting word for the sentence. This can be as simple as a non-word, or a "token" that indicates the start of a sentence. We then pass the hidden state variable through some function that picks the best German word. For example, given the hidden state $\mathbf{h}$, you predict word i with word vector $\mathbf{w}_i$ with probability $P(i|\mathbf{h}) = \frac{\exp(\mathbf{w}_i^\top \mathbf{h})}{\sum_k \exp(\mathbf{w}_k^\top \mathbf{h})}$. Sometimes people pick the most likely word; sometimes they sample one randomly from this word distribution.

The word vector of the generated German word is the next input passed into the RNN. The whole process can be repeated until some stopping criterion. We can then compute the loss between the ground truth translation and the prediction made by the decoder. With this loss, we can do back-propagation and SGD to update the parameters of the encoder and decoder.