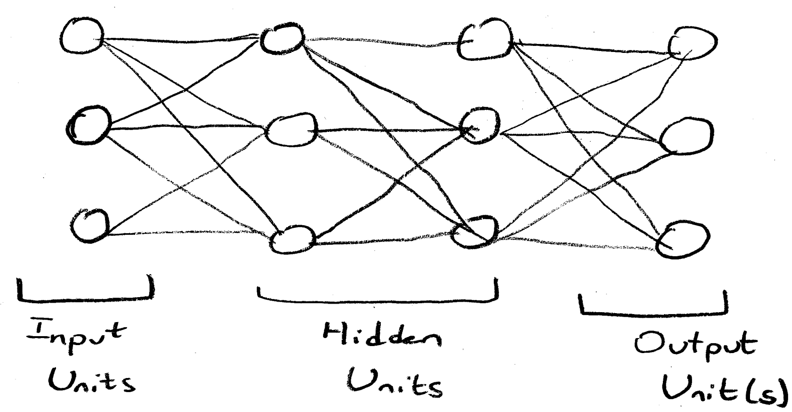
6.036 Recitation Notes - 3/17/2017

Vivek Miglani, Liang Zhou

Agenda

- Brief review of neural networks
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
- Gated RNNs

* Neural Networks



Each node takes sum of inputs times corresponding weights and then applies a non-linearity (e.g. Sigmoid, tanh, ReLU).

Learning parameters = minimize loss using stochastic gradient descent.

For multi-way classification, we can apply softmax on multiple output units to find probability distribution. If outputs are $z_1, z_2, \dots, z_K$

$$P(y=j) = \frac{e^{z_j}}{\sum_{i=1}^{K} e^{z_i}}$$

* Convolutional Neural Network

- Special type of neural network tailored specifically for images
- Has revolutionized image processing, used in facial recognition, image classification, etc.

Intuition - To find a face in an image, you may look for certain smaller features (eyes, nose, etc.) which can be identified by certain shapes or edges. These smaller features can be used to identify the face

Two main components in CNNs

- Convolution Layers
- Pooling Layers

Convolution Layer

- Applies "filter" to each patch in a large image.
- Filter is a linear operation
- Stride controls how much the convolution filter is shifted at each step.

Example

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Apply filter $\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$ to A with stride $= 1$.

Applying to first 2×2 in A. $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$$1(1) + 1(1) + 0(-1) + 0(-1) = 2$$

Final Result

$$\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ -1 & 0 & 0 & 2 \end{bmatrix}$$

Note that this is smaller than the original matrix. In practice, we often pad the input with 0's to control the output size

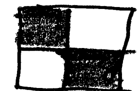
Note: Convolution layer can be interpreted as hidden layer with shared weights between units.

Pooling Layer
- Operation applied to output of convolution layer
- Most common is max pooling
  - Takes max value within $k \times k$ region, where k is the pooling filter size
- Strite controls shift of patch.
- Max pooling contributes to translational invariance
  - Finds feature anywhere within large subregion of image.

Example: Let's apply max pooling to the convolution example with 2×2 filters and stride = 2.

Result

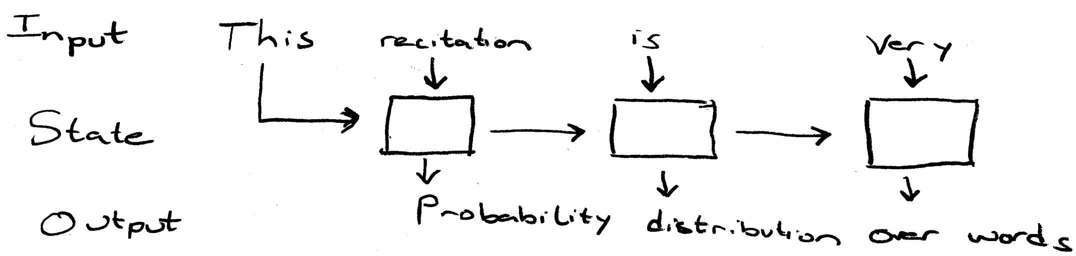
$$\begin{bmatrix} \underline{2} & 0 \\ 1 & \underline{2} \end{bmatrix}$$

Larger values correspond to presence of  image feature in A. (Consider 1 to correspond to darkened squares).

Recurrent Neural Networks (RNNs)

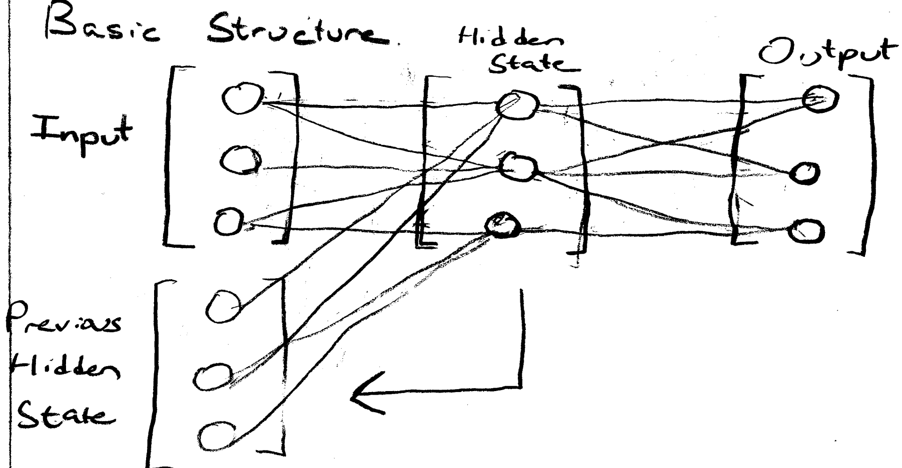
- Variant of basic neural networks when inputs/outputs are sequences.
- Often used in natural language processing.

Example - Sentence completion



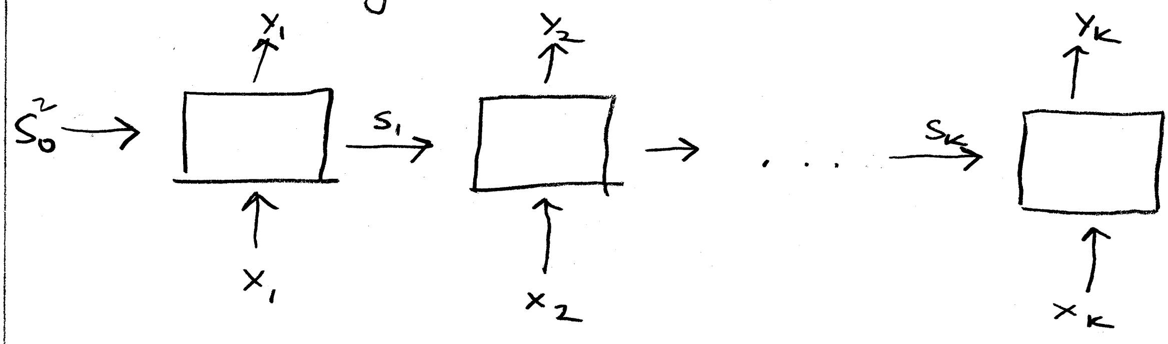
Next word is dependent on all previous words, not just predecessor

Basic Structure.

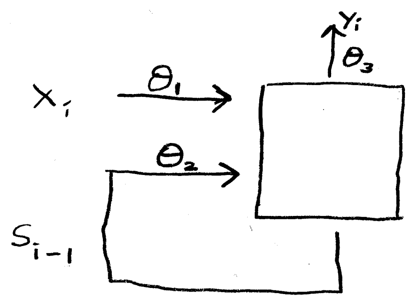


Training an RNN – Backpropogation

- Unravel the recurrent neural network, and then calculate gradient like basic NN.



Let's look at a very simple example RNN

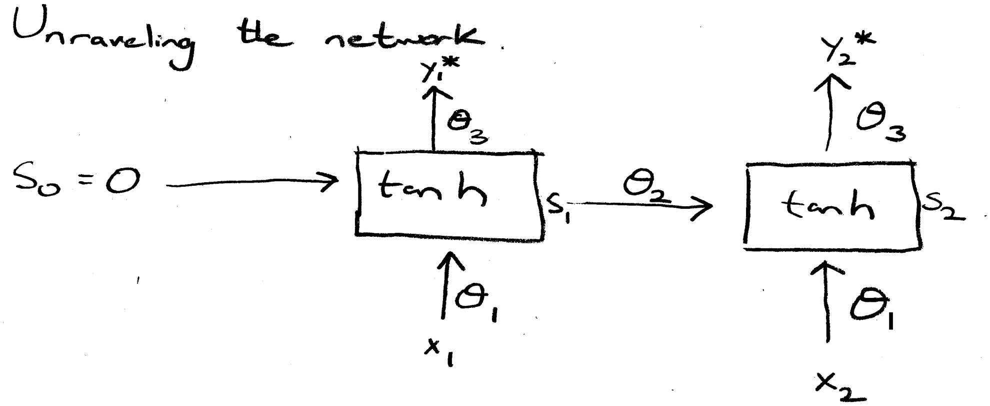


Assume that x_i, y_i and s_i are all scalars

Parameters $\theta_1, \theta_2, \theta_3$ are also scalars.

Assume that we have sequences x_1, x_2 map to y_1, y_2

Unraveling the network.



We can write the following equations based on this network:

$$y_1^* = \theta_3 s_1$$
$$s_1 = \tanh(\theta_1 x_1)$$
$$s_2 = \tanh(\theta_2 s_1 + \theta_1 x_2)$$
$$y_2^* = \theta_3 s_2$$

2

$$Loss = (y_1^* - y_1)^2 + (y_2^* - y_2)^2$$

Let's apply back propagation to find $\frac{\partial Loss}{\partial \theta_1}$

$$\frac{\partial Loss}{\partial \theta_1} = \frac{\partial Loss}{\partial y_1^*} \frac{\partial y_1^*}{\partial \theta_1} + \frac{\partial Loss}{\partial y_2^*} \frac{\partial y_2^*}{\partial \theta_1}$$

$$\frac{\partial y_1^*}{\partial \theta_1} = \frac{\partial y_1^*}{\partial s_1} \frac{\partial s_1}{\partial \theta_1} = \theta_3 (1 - \tanh^2(\theta_1 x_1))(x_1)$$

$$\frac{\partial y_2^*}{\partial \theta_1} = \frac{\partial y_2^*}{\partial s_2} \frac{\partial s_2}{\partial \theta_1} = \theta_3 (1 - \tanh^2(\theta_2 s_1 + \theta_1 x_2)) \frac{\partial(\theta_2 s_1 + \theta_1 x_2)}{\partial \theta_1}$$

$$= \theta_3 (1 - \tanh^2(\theta_2 s_1 + \theta_1 x_2))(x_2 + \theta_2 \frac{\partial s_1}{\partial \theta_1})$$

$$= \theta_3 (1 - \tanh^2(\theta_2 s_1 + \theta_1 x_2))(x_2 + \theta_2 (1 - \tanh^2(\theta_1 x_1)) x_1)$$

Plug these in to,

$$\frac{\partial Loss}{\partial \theta_1} = 2(y_1^* - y_1) \frac{\partial y_1^*}{\partial \theta_1} + 2(y_2^* - y_2) \frac{\partial y_2^*}{\partial \theta_1}$$

We can now update θ_1!

$$\theta_1^{new} = \theta_1 - \eta \frac{\partial Loss}{\partial \theta_1}$$

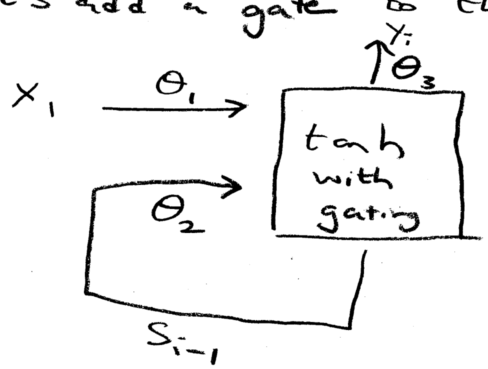
One issue here - note that the tanh derivative terms multiply in $\frac{\partial y_2^*}{\partial \theta_1}$

In a longer recurrent sequence, this would cause vanishing gradients or exploding gradients!

Gated Recurrent Neural Networks

- Gates cause the system to "forget" some part of the hidden state each time.
- This helps mitigate the vanishing/exploding gradient problem.

Let's add a gate to the simple RNN example



Equations:

$$s_i = g_i \tanh(\theta_1 x_i + \theta_2 s_{i-1}) + (1-g_i) s_{i-1}$$

$$g_i = \text{sigmoid}(\theta_5 x_i + \theta_6 s_{i-1}).$$

Exercise Question: If g_i were constant instead, what value of g_i corresponds to simple RNN without gating?

Answer: 1

Some common architectures for gated RNNs

- LSTM - Long Short Term Memory
- GRU - Gated Recurrent Unit