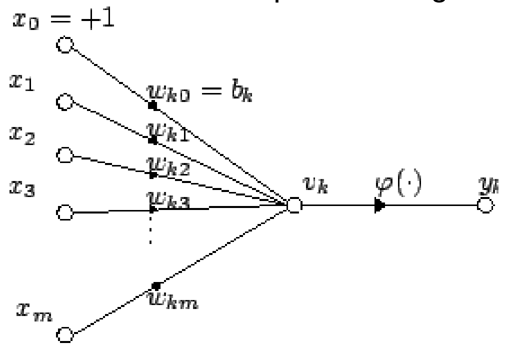
1. Introduction to Deep Learning And Neural Networks

Deep learning is an area of machine learning and pattern recognition. Deep learning is based on the technique of multiple levels of representation. Each layer makes use of the output provided by the previous layer as input. This means that deep learning is based on unsupervised learning of multiple layers. There are various architectures for deep learning, some of them include: Deep Belief Networks(DBN), Convolutional, Recurrent Neural Networks(RNN), etc.

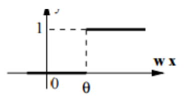
So now, what is a neural network? Neural networks are a model which are used in machine learning and are based on the collection of artificial neurons, which are similar to biological neurons. Their main goal is to solve the problem in a same way that a human brain would. We can define the basic structure of an artificial neuron as follows:

Here there are m+1 inputs with signals, x_0 through x_m and weights are assigned to this inputs as w_0 through w_m. For convenience, we add the biases/ threshold. If first input is assigned the value +1, so that the weight for first input becomes equal to the bias of first input. The output of neuron is given as, $y = \varphi(v)$ where v is the inner product of weight and input vectors, i.e. v = w.x



Here, phi is an activation/transfer function. For example, for a step function with bias, phi can be represented as,

$$y = \varphi(v) = \begin{cases} 1 & \text{if } \mathbf{w}\cdot\mathbf{x} \geq \theta \\ 0 & \text{if } \mathbf{w}\cdot\mathbf{x} < \theta \end{cases}$$



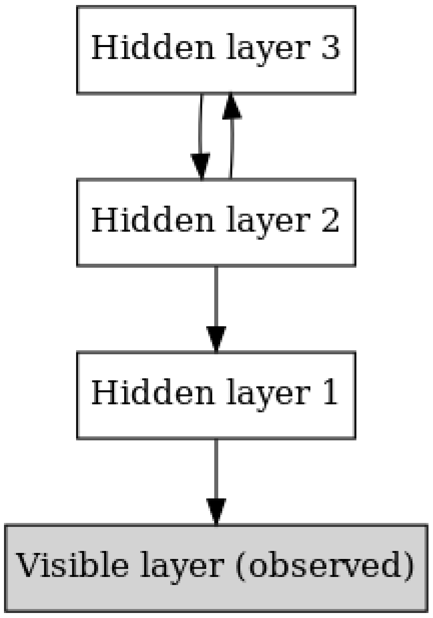
(The McCulloch-Pitts perceptron — 1943)

1.1 Deep Learning Architectures:

Now, since we are clear with the understanding of what exactly is neural networks and deep learning, let's have a look at what are the different architectures of deep learning.

1.1.a Deep Belief Networks:

A simple neural network, with multiple layers, leads to overfitting, because conventional neural networks, cannot take advantage of unlabeled data, which is easiest to collect in big data. To overcome this disadvantage, Deep Belief Networks (DBN), uses a layer-by-layer, unsupervised procedure. Thus, DBN is a feed forward neural network with a deep architecture, i.e. with many hidden layers. It uses unsupervised learning for data distributions, without label information and supervised stages to perform local search for tuning.



The two layers of DBN's are visible layer and hidden layer. The parameter's for DBN are same as conventional neural network, that is they have weights W and biases b. DBN's are constructed using Restricted Boltzmann Machine(RBM), which works on the technique of unsupervised learning. The output of hidden layer of one RBM, can be used as the input to visible layer of another RBM. Thus, DBN's architecture are stacked by n numbers of RBM's.

For both the layers, sampling probabilities can be given by,

$$P(h_j = 1 \mid v; W) = f(b_i + W_i v)$$

$$P(v_i = 1 \mid h; W) = f(a_i + W_i h)$$

Here, W is the weight matrix b and a are biases for hidden and visible layers, respectively. The function f is a logistic function where $f(x) = 1/(1+e^{-a})$, which gives the output in range of (0,1).

Now, for training of this model, we cannot train the entire model at a time without any strategy, because its n layer architecture will lead to low efficiency of learning. Hilton, used the following steps for training of these models,

A. Layer-Wise Unsupervised Learning: Here it trains the RBM, with original data as input. Next, the output of these RBM, acts as an input to another RBM and so on. Finally, we achieve N layers in DBM's, which are suitable of extracting features from this data.
B. Fine-Turning: It adds one more classifier to the end the DBN obtained from step 1. Here the parameters of RBM are slightly changed since the error is propagated in backwards manner.

Example of DBN include Spam filtering.

1.1.b Recurrent Neural Networks:

The Deep Belief Network, which we learned in above section, is a feedforward network, in which activation is "piped" through the network from input to output units. On contrary, RNN's (Recurrent Neural Networks) are not feed forward ones, in fact the neurons in these networks sends feedback to one another, forming directed cycles providing implicit internal memory. The difference between these networks can be shown as below:

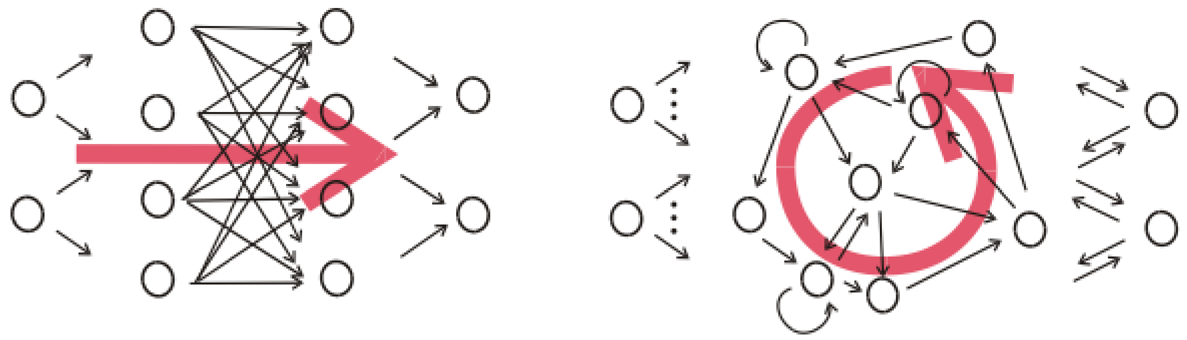


Figure 1.1: *Typical structure of a feedforward network (left) and a recurrent network (right).*

We can process the sequence of given input vectors by applying recurrence formula at every step:

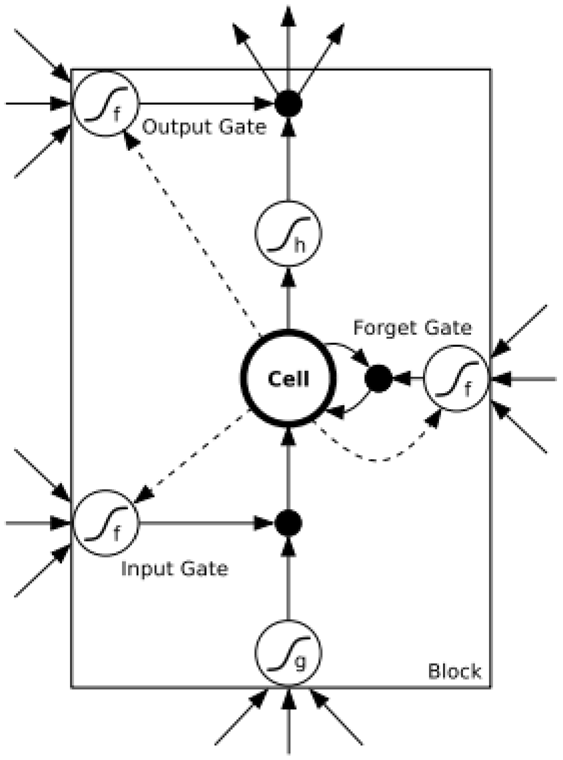
$h_t = f(h_{t-1}, x_t)$ .. (Here h_t is the new state, h_{t-1} is the old state and x_t is input vector).

There are many architectures for RNNs like, Bi-directional RNN, Second Order RNN, Neural Turing Machines, Continuous Time RNN, etc. One of the architectures/type of RNN also include Long Short Term Memory Network (LSTM), which are capable of learning long term dependencies. We will understand the concepts of LSTM, in detail in below sections.

2. Introduction to Long Short Term Memory(LSTM) Networks

As defined in previous section, LSTM is an recurrent neural network, which when given enough number of network units or memory cell, can compute anything a computer program can compute.

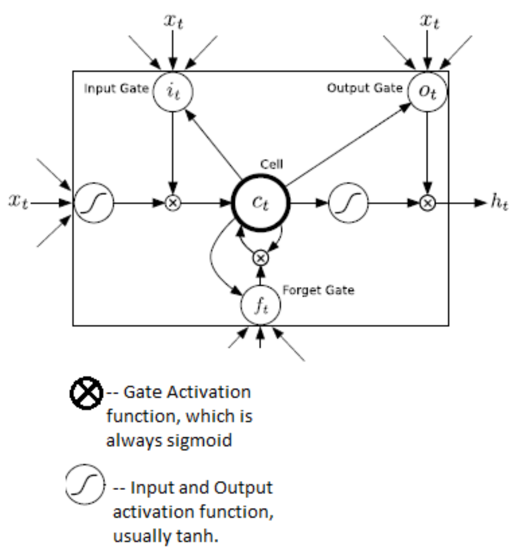
A memory cell in LSTM mainly consists of 4 elements: an input gate, a neuron which connects to itself, a forget gate which allows a LSTM memory cell to reset itself, and, an output gate. The weights (U and W) in a LSTM memory cell are to direct the operations of gates. The weights are the entities used between the input nodes and each of the gates. Now we evaluate below expressions, which will describe how each layer of memory cell is updated at every timestamp t. But before that, let's have a look at LSTM block with one memory cell.



The three gates described above are the units which sums up the activations from both inside and outside of the block and controls the activation of cell (small dark circles). The forget gate takes into consideration the cell's previous state. Here, the function f is the logistic sigmoid function , so that its activation is between value 0 and 1. The g and h functions, which is responsible for activation of input and output cell, are usually tanh or logistic sigmoid or even an identity function. As mentioned above, the connections from input/output gates have weights and all the other connections are not weighted.

2.1. Detailed Description of the LSTM Memory Cell:

Now, in the diagram, we can see that the inputs are not only fed to the input gate, but to all the gates. The gate activation function decides whether to let input in or erase the current state and/or let that state impact the output at the present time step. C_t is the current state of memory cell.



x_t
x_t
Input Gate
i_t
Output Gate
o_t
Cell
x_t
c_t
h_t
Forget Gate
f_t
-- Gate Activation function, which is always sigmoid
-- Input and Output activation function, usually tanh.

Let, $x = x_1 \ldots x_T$ be the input vector.

$h = h_1 \ldots h_T$ be the hidden vector sequence.

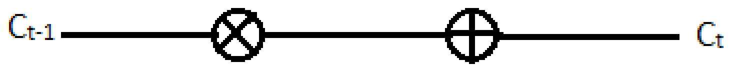
$y = y_1 \ldots y_T$ be the output sequence.

At each time input to the LSTM memory cell is x_t (input for the current time stamp), h_{t-1} (output from the hidden layer of the previous LSTM unit) and C_{t-1} ("memory" or the state of cell from previous LSTM unit).

Now, lets start step by step.

Step1:

Suppose no gates exists in the memory cell, then the cell would look like this:



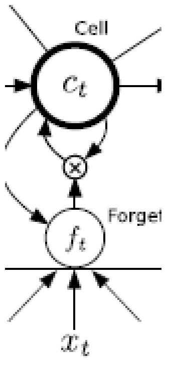
⊗ This is element wise multiplication, i.e. if we multiply the old memory with a vector that is close to 0, it means that we want to forget most of the memory.

⊕ This operator means piece wise summation i.e we merge old memory and new memory with this operator.

Now why are gates used? Because they are the ones to decide what information to let through. Gates are usually composed of sigmoid function, which outputs a number between 0 and 1. 0 means that "let nothing through" and 1 means that "let everything through".

Step 2:

Now, lets first consider the forget gate. As mentioned above, each gate is associated with a sigmoid function and a multiplication operator.



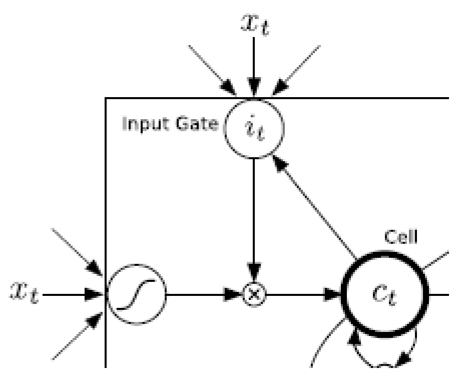
Now the input here is x_t from the input vector at time t, h_{t-1} which is the output from the hidden layer of the previous LSTM unit and a bias b. The input passes through the sigmoid function as activation, the output of which is the forget value, which is applied to the previous unit memory cell C_{t-1}. Thus the equation can be given by,

$$f_t = \text{sigmoid}(W_{xf}.x_t + W_{hf}.h_{t-1} + W_{cf}.C_{t-1} + b_f)$$

where, W_{xf} : Weight on input to forget layer, x_t : Input vector, W_{hf} : weight on hidden/forget layer, h_{t-1} : previous LSTM output, W_{cf} : weight on previous state input and forget layer, C_{t-1} : memory of previous LSTM unit, b_f : bias for forget gate.

Step 3:

Now, after the activation of forget gate, the next step is to decide what new information will be stored in the cell's present state. This step requires 2 executions. Similar to forget gate, input gate also takes 3 inputs. As we can see that along with sigmoid function, the input also passes through other activation function, which is usually tanh. The equation for input gates can be give as:



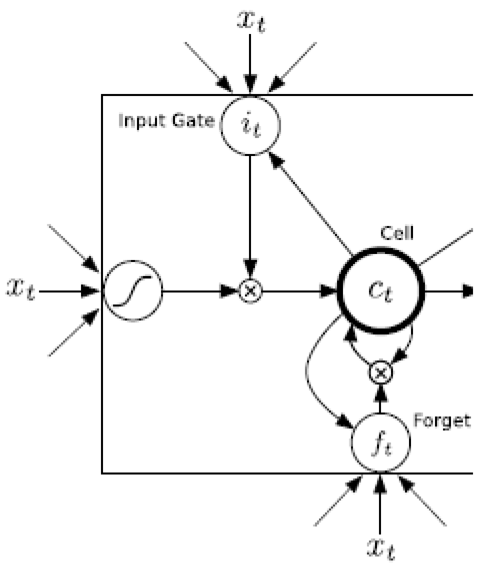
$i_t = \text{sigmoid}(W_{xi}.x_t + W_{hi}.h_{t-1} + W_{ci}.C_{t-1} + b_i)$

The gate activation tanh function can be given by

$C'_t = \tanh(W_{xc}.x_t + W_{hc}.h_{t-1}+ b_i)$

Step 4:

Now combining the above states, the new cell state C_t can be given as,

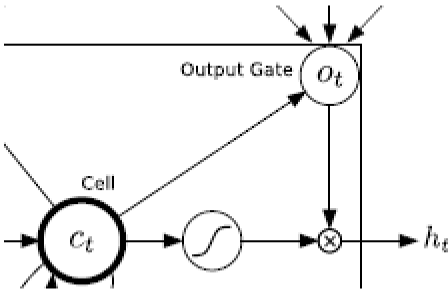


$C_t = f_t.C_{t-1} + i_t.C'_t$.

This equation implies that, we multiply the old state by f_t deciding how much information needs to be remembered. And this value is added to the new memory, which is formed by adding the required amount of old memory to the new inputs.

Step 5:

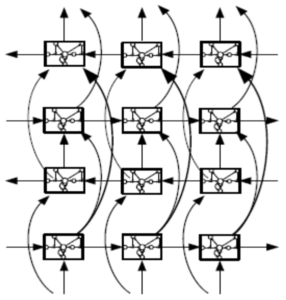
Now, finally calculating the output value. Similar to input and forget gates, the equation for output gate can be given as,



$o_t = \text{sigmoid}(W_{xo}.x_t + W_{ho}.h_{t-1} + W_{co}.C_{t-1} + b_o)$.

Thus, the final output can be given by, $h_t = o_t.\tanh(C_t)$. Here h_t is filtered by running through tanh to push values to be in between -1 and +1. As there can be multiple LSTM memory cell, the general equation for the output can be given as, $y_t = W_{hy}.h_t + b_y$.

2.2. RNN's in Speech Recognition:



Simple RNN's are not used in speech recognition, because in speech recognition, the whole utterance is transcribed at once and the RNN does not makes use of the previous context. BRNN (Bidirectional RNN) process the data in both directions and has a 2 separate hidden layer. Combining these BRNN with LSTM gives bidirectional LSTM. The figure on the right represents bidirectional LSTM.

2.3. Numerical Example for working of LSTM memory cell:

Let, $x_0 = [1\ 2]$ with label 0.5 ; $x_1 = [0.5\ 3]$ with label 1.25.

Let, $W_{xi} = [0.95\ 0.8]$, $W_{hi} = [0.8]$, $b_i = [0.65]$; $W_{xo} = [0.6\ 0.4]$, $W_{ho} = [0.25]$, $b_o = [0.1]$; $W_{xf} = [0.7\ 0.45]$, $W_{hf} = [0.1]$, $b_f = [0.15]$; $W_{xc} = [0.45\ 0.25]$, $W_{hc} = [0.15]$, $b_c = [0.2]$.

Plugging this values, in the equations above we get, at state 0, i.e. t=0,

$i_t = \text{sigmoid}([0.95\ 0.8]\ [1\ 2] + [0.8]*0 + [0.65]) = \text{sigmoid}(0.95+1.6+0.65) = \text{sigmoid}(3.2) = 0.9606$.

$C'_t = \tanh(\ [0.45\ 0.25]\ [1\ 2] + [0.15]*0 + [0.2]) = \tanh(0.45+0.5+0.2) = \tanh(1.15) = 0.8178$.

$f_t = \text{sigmoid}(\ [0.7\ 0.45]\ [1\ 2] + 0 + [0.15]) = \text{sigmoid}(0.7+ 0.9 + 0.15) = \text{sigmoid}(1.75) = 0.851956$

Now, $C_t = 0.851956*0+0.9606*0.8178 = 0.78558$.

$o_t = \text{sigmoid}(\ [0.6\ 0.4]\ [1\ 2] + 0 + [0.1]) = \text{sigmoid}(0.6+0.8+0.1) = \text{sigmoid}(1.5) = 0.81757$

Final output, $h_t = 0.81757 * \tanh(0.78558) = 0.81757 * 0.65589 = 0.5362$.

Thus, this is assigned label 0.5

Now, at state 1, t=1,

i_t = sigmoid([0.95 0.8] [0.5 3] + [0.8]*0.5362 + [0.65]) = sigmoid(2.875+0.42896+0.65) = sigmoid(3.95396) = 0.98118.

C'_t = tanh([0.45 0.25] [0.5 3] + [0.15]*0.5362 + [0.2]) = tanh(0.975+0.08043+0.2) = tanh(1.25043) = 0.8484.

f_t = sigmoid([0.7 0.45] [0.5 3] + 0.1*0.5362 + [0.15]) = sigmoid(1.725+0.05362+0.15) =sigmoid(1.92862) = 0.8731.

Now, C_t = 0.8731*0.78558+0.98118*0.848 = 0.68589 + 0.8324 = 1.518

o_t = sigmoid([0.6 0.4] [0.5 3] + 0.25*0.5632 + [0.1]) = sigmoid(1.5+0.13405+0.1) = sigmoid(1.73605) = 0.8501

Final output, h_t = 0.8501 * tanh(1.518) = 0.8501 * 0.90835 = 0.77218.

Thus, this is assigned label 1.25

2.4. What is Deep RNN:

Deep RNN are the ones which are created by stacking multiple RNN hidden layers on top of each other. Thus the output of one layer serves as the input to other. Using simple LSTM RNN, there exits no problem of vanishing gradient. Using gradient methods or backpropagation methods to train artificial neural network, it may happen that with more number of hidden layers, the neurons in the starting layers learn much more slowly than the neuron in the later layer. This is known as vanishing gradient problem, where the gradient is reduced or vanished when it is backpropagated from the output layer.

Gradient Based Learning Methods:

The main objective of the gradient descent algorithms is to minimize the functions. With given initial values, gradient descent methods starts with those values and iteratively moves towards a set of values that minimizes the function. For artificial neural network, this method is used for minimizing the loss function.

Backpropagation Method:

This is another method for training artificial neural networks. This method works layer by layer. It starts with the input layer and is propagated through network, layer by layer, until it reaches the output layer. The output is compared to the desired output and an error value is calculated. The error value is thus propagated backwards, starting from output. These error values are used to calculate the gradient loss function. This gradient is then fed to optimization method which in turn uses it to update weights for minimizing the loss function.

2.5. Network training:

Before going forward with network training of LSTM model, lets have a look at what is GMM HMM system model. GMM is a gaussian mixture model which is a probabilistic model that assumes that all the data points are generated from finite number of gaussian distributions. HMM is a hidden markov model, which can be determined as the set of observations with a set of hidden state. For example, in speech recognition, observation is sound signal and hidden states are part of speech, words.

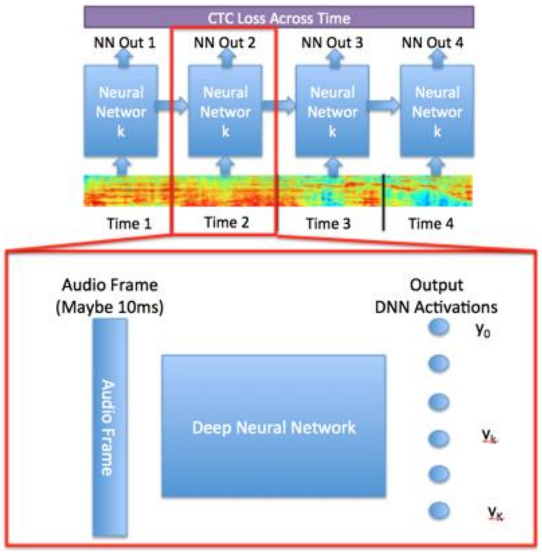
Now, let us understand the most basic concepts which can be used in speech recognition: phone and frame. Phone is a single unit if speech. Example, in ‘ah’, there is a single phone ‘a’. A word may consists of many different phones. Frames, on the other hand, is a short segment of audio. It is short enough that only one phone is uttered.

Again, in order to recognize speech, we must be able to identify which phone was uttered in every frame. Most of the HMM architecture for speech recognition, have below states : beginning, middle and end. In each state, a phone may sound a bit different. Each state is modeled by GMM, to determine the likelihood of observation.

2.6. What is TIMIT dataset?

TIMIT is a speech dataset which was developed by **T**exas **I**nstruments and **MIT.** It has recording of almost 630 speakers each reading 10 phonetically rich sentences. More info for same can be found on online documentation of TIMIT dataset: https://catalog.ldc.upenn.edu/docs/LDC93S1/.

2.7. Example of speech recognition using CTC



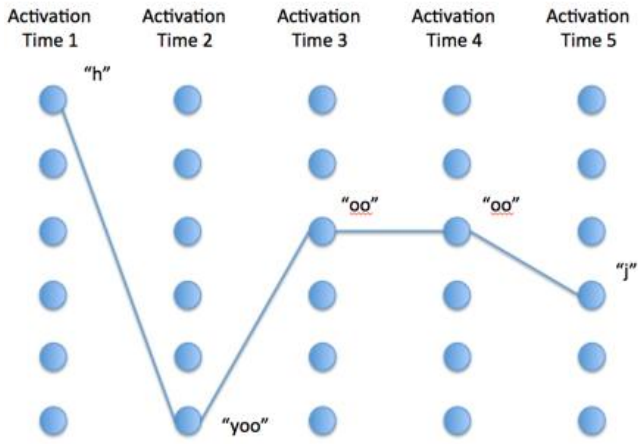
CTC is connectionist temporal classification. Now, let's suppose RNN outputs neurons c, distribution over symbol, c – {A, B, C,….,Z, blank, space}.

Output neurons, defines distribution over whole character sequences, assuming, P(C|x) is the probability of the individual character given x.

Lets assume that the input to RNN is the output of audio waveform, which further produces the output of: HHH_E__LL__LO___.

Now this can be predicted as,

P(C= HHH_E__LL__LO___|x) = P(c1 = H |x) P(c2 = H |x) … P(c15 = blank |x)



P(C|x) = { 0.1 HHH_E_LL_LO___ "Hello"

0.02 HH__E__LL_LO___ "Hello"

0.01 HHH_E__L_L_OH__ "Hell Oh"

0.01 HHH_EE_LL_L_O__ "Hello"

… YY__E__LL_LO_W "YELLOW" }

Thus, P("Hello") = 0.1+0.02+0.01+….