10-708: Probabilistic Graphical Models 10-708, Spring 2016

27: Hybrid Graphical Models and Neural Networks

Lecturer: Matt Gormley *Scribes: Jakob Bauer, Otilia Stretcu, Rohan Varma*

1 Motivation

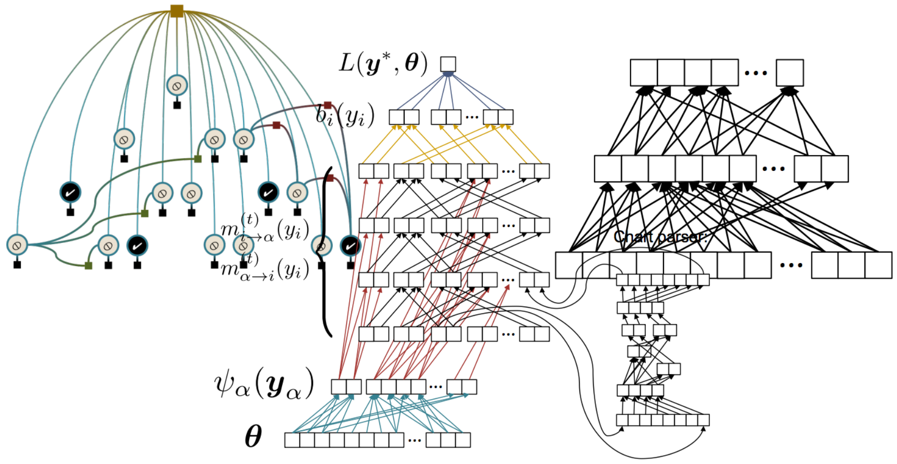
We first look at a high-level comparison between deep learning and standard machine learning techniques (like graphical models). The empirical goal in deep learning is usually that of classification or feature learning, whereas in graphical models we are often interested in transfer learning and latent variable inference. The main learning algorithm in deep learning is back-propagation whereas in machine learning, this is a major focus of open research with many inference algorithms, some of which we have studied in this course. Deep learning is also often tested and evaluated upon massive datasets.

When evaluating performance in the many different techniques, we want to able to conclusively determine the factor that led to the improvement in performance. That is, whether it came from using a better model architecture, or a better algorithm that is more accurate and/or faster, or is it merely because of better training data. In current deep learning research, since we evaluate using a black-box performance score, we often lose the ability to make these distinctions. Additionally, the concept of training error is perfectly valid when we have a classifier with no hidden states and hence inference and inference accuracy is not an issue. However, deep neural networks are not just classifiers and possess many hidden states, so inference quality is something that should be looked at more carefully. This is often not the case in current deep learning research. Hence, in graphical models, lots of efforts are directed towards improving inference accuracy and convergence, whereas in deep learning, most effort is directed towards comparing different architectures and activation functions. A convergence between these two fields is something that we might see in the near future given a lot of the similarities between how they both started.

1.1 Hybrid NN + HMM

We have seen how graphical models lets us encode domain knowledge in an easy way. Also neural networks are good at fitting data discriminatively to make good prediction. We now examine whether we can combine the best of these two paradigms, that is, can we define a neural network that incorporates domain knowledge.

In this hybrid model, we want to use a neural network to learn features for a graphical model, then train the entire model using back-propagation.



$L(\boldsymbol{y}^*, \boldsymbol{\theta})$
$b_i(y_i)$
$m^{(t)}_{i\to\alpha}(y_i)$
$m^{(t)}_{\alpha\to i}(y_i)$
$\psi_\alpha(\boldsymbol{y}_\alpha)$
$\boldsymbol{\theta}$
Chart parser

(a)

Figure 1: Hybrid Models

The general recipe for classification is as follows. Given training data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, we choose a decision function $\hat{\mathbf{y}} = f_\theta(\mathbf{x}_i)$ and a loss function $\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$. Now we can define our objective and can train (minimize the objective) using gradient descent as below.

$$\theta^* = \arg\min_\theta \sum_{i=1}^N \ell(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$$

In our hybrid model, we incorporate our graphical model into the decision function $\hat{\mathbf{y}} = f_\theta(\mathbf{x}_i)$. We know how to compute the marginal probabilities, but we now want to make a prediction $\mathbf{y}$.

We could use a minimum Bayes risk (MBR) decoded $h(x)$ returns the variable assignment with minimum expected loss under the model's distribution.

$$\begin{aligned} h_\theta(\mathbf{x}) &= \mathbb{E}_{\mathbf{y}\sim p_\theta(\cdot|\mathbf{x})}[\ell(\hat{\mathbf{y}}, \mathbf{y})] \\ &= \arg\min_{\hat{\mathbf{y}}} \sum_{\mathbf{y}} p_\theta(\mathbf{y}|\mathbf{x})\ell(\hat{\mathbf{y}}, \mathbf{y}) \end{aligned}$$

We can use the $0-1$ loss function which returns 1 only if the two assignments are identical and 0 otherwise,

$$\ell(\hat{\mathbf{y}}, \mathbf{y}) = 1 - \mathbb{I}(\hat{\mathbf{y}}, \mathbf{y})$$

in which case the MBR decoder is exactly the MAP inference problem.

$$\begin{aligned} h_\theta(\mathbf{x}) &= \arg\min_{\hat{\mathbf{y}}} \sum_{\mathbf{y}} p_\theta(\mathbf{y}|\mathbf{x})(1 - \mathbb{I}(\hat{\mathbf{y}}, \mathbf{y})) \\ &= \arg\max_{\hat{\mathbf{y}}} p_\theta(\hat{\mathbf{y}}|\mathbf{x}) \end{aligned}$$

We could also use the Hamming loss function which returns the number of incorrect variable assignments

$$\ell(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^{V} (1 - \mathbb{I}(\hat{y}_i, y_i))$$

In this case the MBR decoder is

$$\hat{y}_i = h_\theta(\mathbf{x})_i = \arg\max_{\hat{y}_i} p_\theta(\hat{y}_i|\mathbf{x})$$

and decomposes across variables and requires the variable marginals. Now going back to our recipe, if were to use the MBR decoder as the decision function, we note that we minimized the objective and updated using the gradient descent where we need to compute $\nabla\ell(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$. However the MBR decoder is not differentiable because of the arg max term and we cannot compute the updates.

2 Hybrid NN + HMM

2.1 Model: neural net for emissions

Our model is a hybrid HMM + NN where the HMM has Gaussian emission. Note that this is sometimes called a GMM-HMM model. The hidden states, $S_1, \ldots, S_N$, are phonemes, i.e., $S_t \in \{/p/, /t/, /k/, \ldots, /g/\}$, and the emissions, $Y_1, \ldots, Y_T$, are vectors in k-dimensional space, i.e., $Y_t \in \mathbb{R}^k$. The emission probabilities are given by a mixture of Gaussians:

$$p(Y_t|S_t = i) = \sum_k \frac{Z_k}{\sqrt{(2\pi)^n |\Sigma_k|}} \exp\left\{-\frac{1}{2}(Y_t - \mu_k)\Sigma_k^{-1}(Y_t - \mu_k)^T\right\}. \tag{1}$$

We can combine the HMM with a NN by attaching identical copies of the same NN to each of the emission nodes (see Figure 2a). The NN takes as input a fixed number of frames of the speech signal, $x_1, \ldots, x_M$, and generates learned features of the frames as its ouput. Note that the input frames of neighboring networks are overlapping.

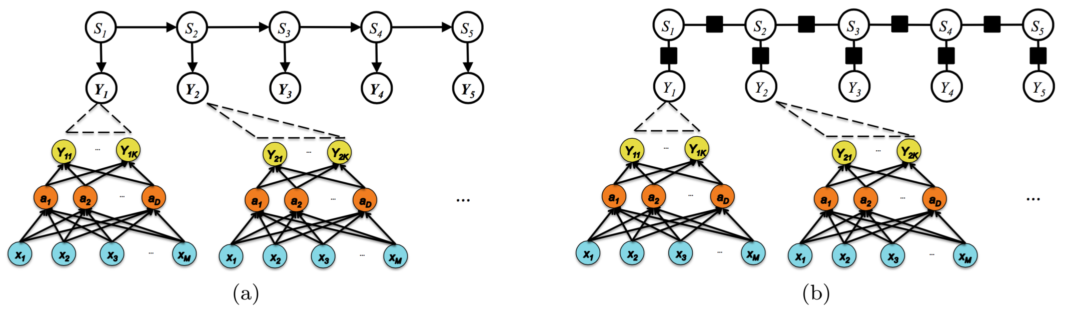


Figure 2: Hybrid HMM + NN model

There are several issues with this model:

- The visual representation of the HMM and NN are similar although they are completely unrelated.
- The emissions are simultaneously generated in a top-down fashion by the HMM and in a bottom-up fashion by the NN.
- The emissions are not actually observed but only appear as a function of the observed speech signal.

The picture improves somewhat if we use factor graph notation instead (see Figure 2b). The graph still represents an HMM but we have gotten rid of the arrows.

At test time, we first feed the speech signal as input into the NN and then use the resulting features for each timestep as observations in the HMM. In a third step, we run the forward-backward algorithm on the HMM to compute the alpha-beta probabilities and the gamma (marginal) probabilities according to the following formulae:

$$\alpha_{i,t} \triangleq P\big(Y_1^t, S_t = i \big| \text{model}\big) = b_{i,t} \sum_j a_{j,i} \alpha_{j,t-1}, \tag{2}$$

$$\beta_{i,t} \triangleq P\Big(Y_{t+1}^T \Big| S_t = i, \text{model}\Big) = \sum_j a_{i,j} b_{j,t+1} \beta_{j,t+1}, \tag{3}$$

$$\gamma_{i,t} \triangleq P\big(S_t = 1 \big| Y_1^t, \text{model}\big) = \alpha_{i,t}\beta_{i,t}, \tag{4}$$

where the transition and emission probabilities are defined as

$$a_{i,j} \triangleq P(S_t = 1 | S_{t-1} = j)\,, \tag{5}$$

$$b_{i,t} \triangleq P(Y_t | S_t = i)\,. \tag{6}$$

Finally, we compute a loss in the form of the log-likelihood:

$$\log p(\boldsymbol{S}, \boldsymbol{Y}) = \alpha_{\text{end},T}. \tag{7}$$

Note that to get a loss of this form, all we do have to do is to use use the alpha-beta probabilities as the outputs of the decision functions.

2.2 Learning: backprop for end-to-end training

Given the decision function and the the objective function, the only remaining question is how to compute the gradient. Fortunately, all the feed-forward computations are now differentiable since there is no longer an argmax operation but only summations and products to compute the α's and β's.

Recall that in backpropagation, for a given $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$ the gradient can be computed by applying the chain rule:

$$\frac{\mathrm{d}y_i}{\mathrm{d}x_k} = \sum_{j=1}^{J} \frac{\mathrm{d}y_i}{\mathrm{d}u_j} \frac{\mathrm{d}u_j}{\mathrm{d}x_k}, \quad \forall i, k. \tag{8}$$

In our case, $g(\cdot)$ is the graphical model and $h(\cdot)$ the neural network. Hence, all we need to be able to compute the gradient are partial derivatives for the graphical model and partial derivatives for the neural network. In particular, $\frac{\mathrm{d}y_i}{\mathrm{d}u_j}$ is the partial derivative of the log-likelihood with respect to the emissions and $\frac{\mathrm{d}u_j}{\mathrm{d}x_k}$ is the partial derivative of the emission with respect to the NN parameters.

Thus, the forward pass is given by the following equations:

$$c_j = \sum_i \lambda_{ij} x_i, \tag{9}$$

$$z_j = \sigma(c_j)\,, \tag{10}$$

$$d = \sum_j \rho_j z_j, \tag{11}$$

$$Y_{t,k} = \sigma(d)\,. \tag{12}$$

$$\alpha_{i,t} = b_{i,t} \sum_j a_{j,i} \alpha_{j,t-1}, \tag{13}$$

$$\beta_{i,t} = \sum_j a_{i,j} b_{j,t+1} \beta_{j,t+1}, \tag{14}$$

$$\gamma_{i,t} = \alpha_{i,t} \beta_{i,t}, \tag{15}$$

$$J = \log p(\boldsymbol{S}, \boldsymbol{Y}) = \alpha_{\text{end},t}. \tag{16}$$

The first box corresponds to the NN part and the second box to the HMM part. The third box shows how to compute the loss.

For the backward pass, we have:

$$\frac{\mathrm{d}J}{\mathrm{d}b_{i,t}} = \frac{\gamma_{i,t}}{b_{i,t}}, \tag{17}$$

$$\frac{\mathrm{d}J}{\mathrm{d}Y_{t,k}} = \sum_{b_{i,t}} \frac{\mathrm{d}J}{\mathrm{d}b_{i,t}} \frac{\mathrm{d}b_{i,t}}{\mathrm{d}Y_{i,k}}, \tag{18}$$

$$\frac{\mathrm{d}b_{i,t}}{\mathrm{d}Y_{jt}} = \sum_k \frac{Z_k}{\sqrt{(2\pi)^n |\Sigma_k|}} \left(\sum_l w_{kl} \left(\mu_{kl} - Y_{lt} \right) \right) \exp\left\{ -\frac{1}{2} \left(Y_t - \mu_k \right) \Sigma_k^{-1} \left(Y_t - \mu_k \right)^T \right\} \tag{19}$$

$$\frac{\mathrm{d}J}{\mathrm{d}d} = \frac{\mathrm{d}J}{\mathrm{d}y} \frac{\mathrm{d}y}{\mathrm{d}d}, \quad \frac{\mathrm{d}y}{\mathrm{d}d} = \sigma(d) \left(1 - \sigma(d)\right), \tag{20}$$

$$\frac{\mathrm{d}J}{\mathrm{d}\rho_j} = \frac{\mathrm{d}J}{\mathrm{d}d} \frac{\mathrm{d}d}{\mathrm{d}\rho_j}, \quad \frac{\mathrm{d}d}{\mathrm{d}\rho_j} = z_j, \tag{21}$$

$$\frac{\mathrm{d}J}{\mathrm{d}z_j} = \frac{\mathrm{d}J}{\mathrm{d}d} \frac{\mathrm{d}d}{\mathrm{d}z_j}, \quad \frac{\mathrm{d}d}{\mathrm{d}z_j} = \rho_j, \tag{22}$$

$$\frac{\mathrm{d}J}{\mathrm{d}c_j} = \frac{\mathrm{d}J}{\mathrm{d}z_j} \frac{\mathrm{d}z_j}{\mathrm{d}c_j}, \quad \frac{\mathrm{d}z_j}{\mathrm{d}c_j} = \sigma(c_j) \left(1 - \sigma(c_j)\right), \tag{23}$$

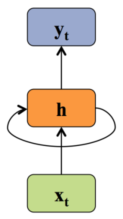
$$\frac{\mathrm{d}J}{\mathrm{d}\lambda_{ij}} = \frac{\mathrm{d}J}{\mathrm{d}c_j} \frac{\mathrm{d}c_j}{\mathrm{d}\lambda_{ij}}, \quad \frac{\mathrm{d}c_j}{\mathrm{d}\lambda_{ij}} = x_i. \tag{24}$$

3 Recurrent Neural Networks (RNNs)

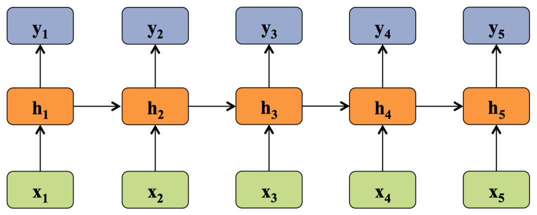
3.1 RNN definition

Deep Neural Networks, such DNN, DBN, DBM, have several limitations. For instance, they only accept inputs of fixed size, and produce outputs of fixed size. This makes it difficult to work with sequences of data, such as time series. In addition, these models have a fixed number of computation steps, given by the number of layers in the model. Recurrent Neural Networks (RNNs) are able to deal with these problems and model sequences of data, such as sentences, speech, stock market, and signal data. RNNs are neural networks with loops, which allows information to persist. Figure 3 (a) shows a typical RNN with one hidden layer. The loop allows information to be passed from one step of the network to the next. We can define a RNN as follows:

$$\begin{aligned}\text{inputs} &: x = (x_1, x_2, ..., x_T), x_i \in \mathbb{R}^I \\ \text{hidden units} &: h = (h_1, h_2, ..., h_T), h_i \in \mathbb{R}^J \\ \text{outputs} &: y = (y_1, y_2, ..., y_T), y_i \in \mathbb{R}^K \\ \text{nonlinearity} &: \mathcal{H}\end{aligned} \quad \text{where} \quad \begin{aligned} h_t &= \mathcal{H}(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \\ y_t &= W_{hy}h_t + b_y\end{aligned}$$



(a) RNN



(b) Unrolled RNN

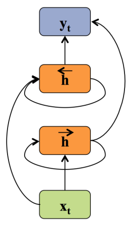
Figure 3: Recurrent neural network (RNN)

We can *unroll* the RNN through time as shown in Figure 3 (b). Notice that when $T = 1$, we have a standard feed-forward neural network with one hidden layer. When $T > 1$, we can share parameters and allow input/output pairs of arbitrary length. To train the RNN, we unroll it through time and apply backpropagation, as in a typical feed-forward neural net.

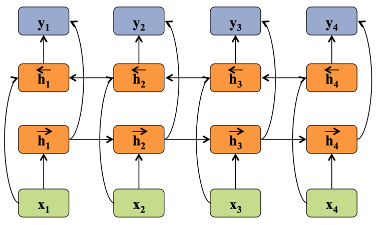
3.2 Bidirectional RNNs

The loop in the RNN allows it to incorporate past information into the current prediction. However, in certain applications, such as speech processing, we want to model both the left (backward in time) and right (forward in time) context in order to make a prediction. This can be modeled using Bidirectional RNNs.

Bidirectional RNNs contain two types of hidden layers, one which has forward loop, and the other has a backward loop (see Figure 4).



(a) Bidirectional RNN



(b) Bidirectional RNN unrolled

Figure 4: Bidirectional recurrent neural network

We can define a Bidirectional RNN as follows:

$$\begin{aligned} \text{inputs} &: x = (x_1, x_2, ..., x_T), x_i \in \mathbb{R}^I \\ \text{hidden units} &: \overrightarrow{h} \text{ and } \overleftarrow{h} \\ \text{outputs} &: y = (y_1, y_2, ..., y_T), y_i \in \mathbb{R}^K \\ \text{nonlinearity} &: \mathcal{H} \end{aligned} \qquad \text{where} \qquad \begin{aligned} \overrightarrow{h}_t &= \mathcal{H}(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}) \\ \overleftarrow{h}_t &= \mathcal{H}(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t-1} + b_{\overleftarrow{h}}) \\ y_t &= W_{\overleftarrow{h}y} \overleftarrow{h}_t + W_{\overrightarrow{h}y} \overrightarrow{h}_t + b_y \end{aligned}$$

3.3 Deep RNNs

In the examples above, we defined, for simplicity, a RNN with only one hidden layer, and a bidirectional RNN with only one hidden layer in each direction. However, we can also define RNNs and bidirectional RNNs with several hidden layers. For example, Figure 5 shows a deep bidirectional RNN. The upper level hidden units, as well as the output layer, contain inputs from the previous two layers (i.e. wider input). This model allows us to capture information from both left and right context, as well as learn interesting features in the higher levels of the RNN.

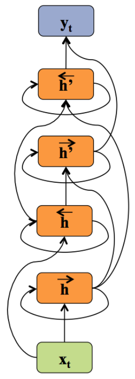
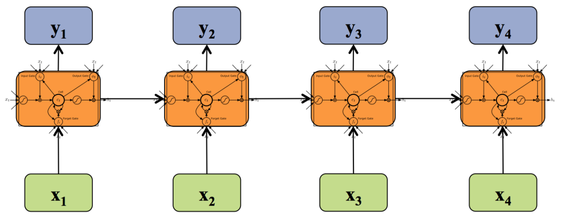


Figure 5: Deep bidirectional RNN

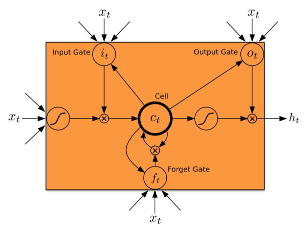
3.4 LSTM

RNNs are very useful due to their ability to incorporate past information to the present task. For instance, in sentence processing, it can use the previous words in order to make predictions for the next word. In theory, the structure of the RNN should allow it to handle *long-term dependencies*, meaning it should be able to incorporate information from inputs as far back in time as necessary. In practice, this is most often not possible, due to what is known as *the vanishing gradient* problem. This refers to the fact that, in standard RNNs, the signal from much earlier inputs is lost through time, because we multiply many very low valued gradients.

Long Short-Term Memorm (LSTM) neural networks are a type RNN that deals with this problem, and are able to learn long-term dependencies. In LSTMs, the hidden units contain special *gates*, which determine how the information is propagated through time. These gates allow the network to choose to *remember* or *forget* information. Figure 6 (a) shows an unrolled LSTM, and Figure 6 (b) shows a LSTM hidden unit, which controls the propagation of information.



(a) LSTM



(b) LSTM unit

Figure 6: Long - short term memory network (LSTM)

The LSTM hidden unit in Figure 6 (b) consists of: (i) an **input gate**, i_t, which masks out the standard RNN inputs, (ii) a **forget gate**, f_t, which masks out the previous cell, (iii) a **cell**, c_t, which combines the input with the forget mask in order to learn keep a current state of the LSTM, (iv) an **output gate**, o_t, which masks out the values of the next hidden input. Note that there are several LSTM gate architectures that have been proposed in the literature. Here we show a simple version. However, [Jozefowicz et al., 2015] have explored 10,000 different LSTM architectures, and found several variants that worked similarly well on several tasks.

We can also have Deep Bidirectional LSTMs (DBLSTM), which has the same general topology as a Deep Bidirectional RNN, but the hidden units consist of LSTM units, instead of single neurons. Such networks present no additional representation power over Deep Bidirectional RNNs. However, they are easier to learn, and therefore are preferred in practice.

4 Hybrid RNN + HMM

Hybrid RNN and HMM are similar in principle with the hybrid NN+HMM described in section 2. The difference is that the we replace the NN with a RNN. We can also make the RNN bidirectional, and replace the hidden units of the RNN with LSTM units. We show this architecture in Figure 7.

Inference and training in this model is similar to that of the hybrid NN+HMM. The objective is to maximize the log-likelihood, and inference can be done with the forward-backward algorithm. To learn the model, we can use Stochastic Gradient Descent (SGD) by backpropagation.

[Graves et al., 2013] applied this model on the task of phoneme recognition. Their results show that the hybrid NN+HMM models perform better than neural net only models, and among the hybrid architectures, the HMM + Deep Bidirectional LSTM perform best.

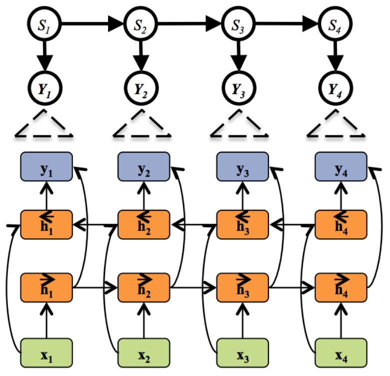


Figure 7: Hybrid RNN + HMM

5 Hybrid CNN + CRF

We can also create hybrid graphical models and neural networks using Conditional Random Fields (CRFs) and Convolutional Neural Networks (CNNs). Such a model is shown in Figure 8. This model allows us to compute the values of the factors using a CNN. We allow each factor to have its own parameters (i.e. its own CNN). This hybrid model is particularly suited for Natural Language Processing applications such as part-of-speech tagging, noun-phrase and verb-phrase chunking, named-entity recognition, and semantic role labeling. In such NLP tasks, the CNN is 1-dimensional. The output of the CNN is fed into the input of the CRF. Experiments on these tasks show that the CNN+HMM model gives results close to the state of the art on benchmark systems. In addition, the advantage of this model is that it does not require hand-engineered features.

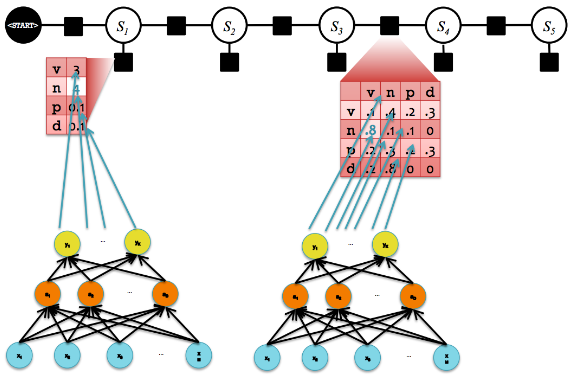


Figure 8: Hybrid CNN + CRF

References

[Graves et al., 2013] Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE.

[Jozefowicz et al., 2015] Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350.