10-708 Probabilistic Graphical Models

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Hybrids of NN/PGM
+
MAP Inference with MILP

Matt Gormley
Lecture 8
Feb. 24, 2021

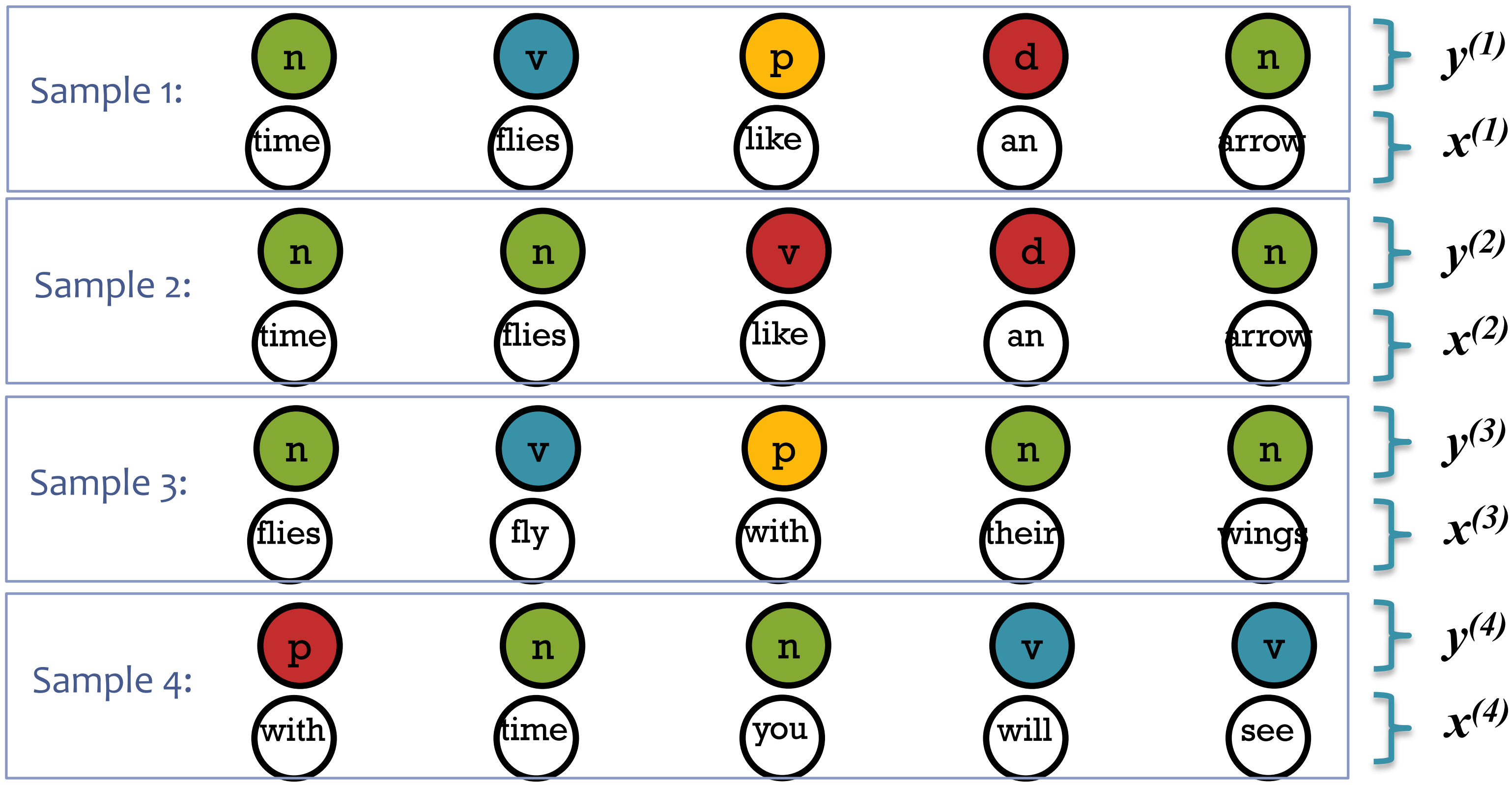
Reminders

- **Homework 2: Exact inference and supervised learning (CRF+RNN)**
  - **Out: Wed, Feb. 24**
  - **Due: Wed, Mar. 10 at 11:59pm**

RECURRENT NEURAL NETWORKS

Dataset for Supervised Part-of-Speech (POS) Tagging

Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$



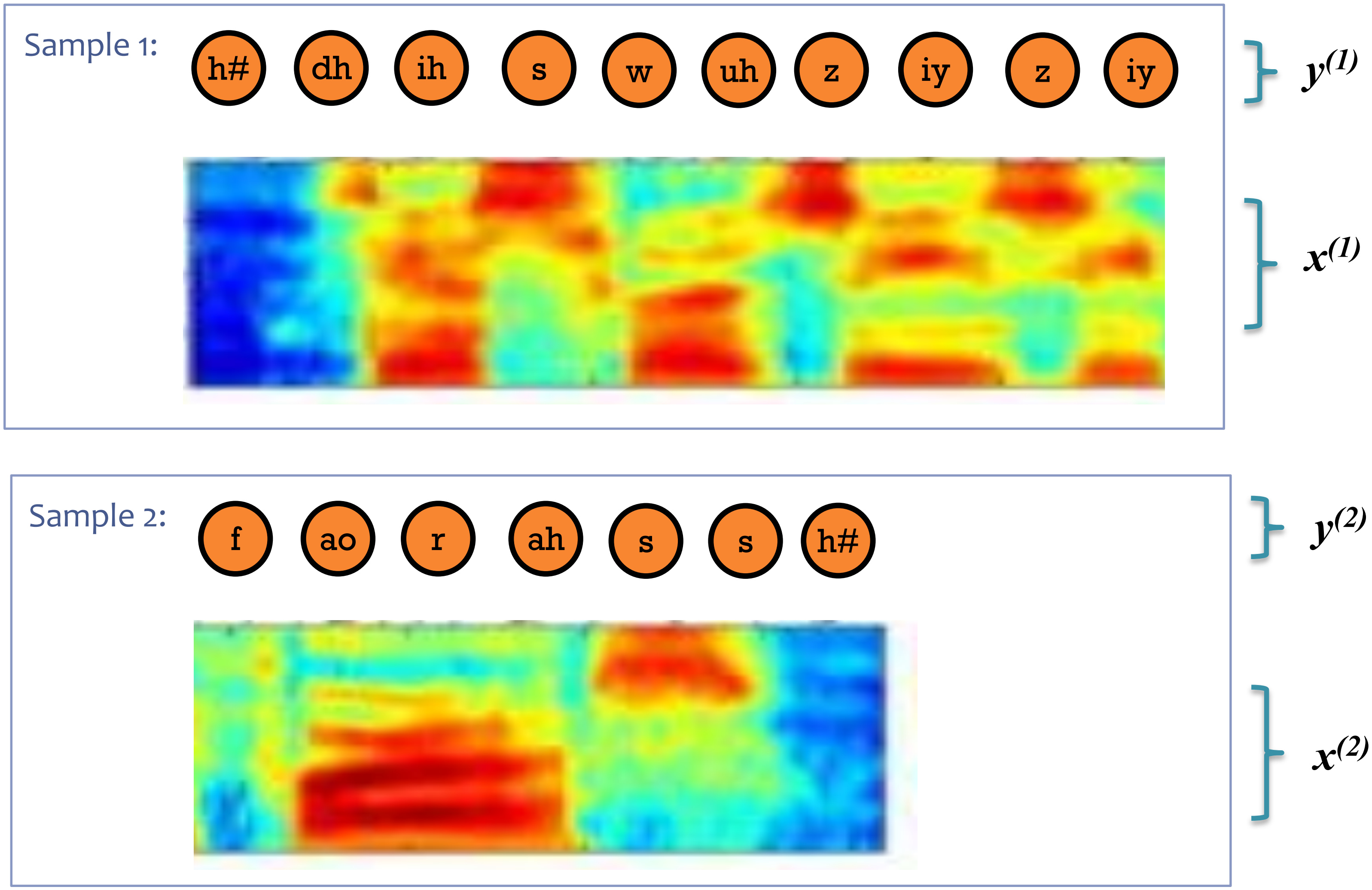
Dataset for Supervised Handwriting Recognition

Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$

Figures from (Chatzis & Demiris, 2013)

Dataset for Supervised Phoneme (Speech) Recognition

Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$



Figures from (Jansen & Niyogi, 2013)

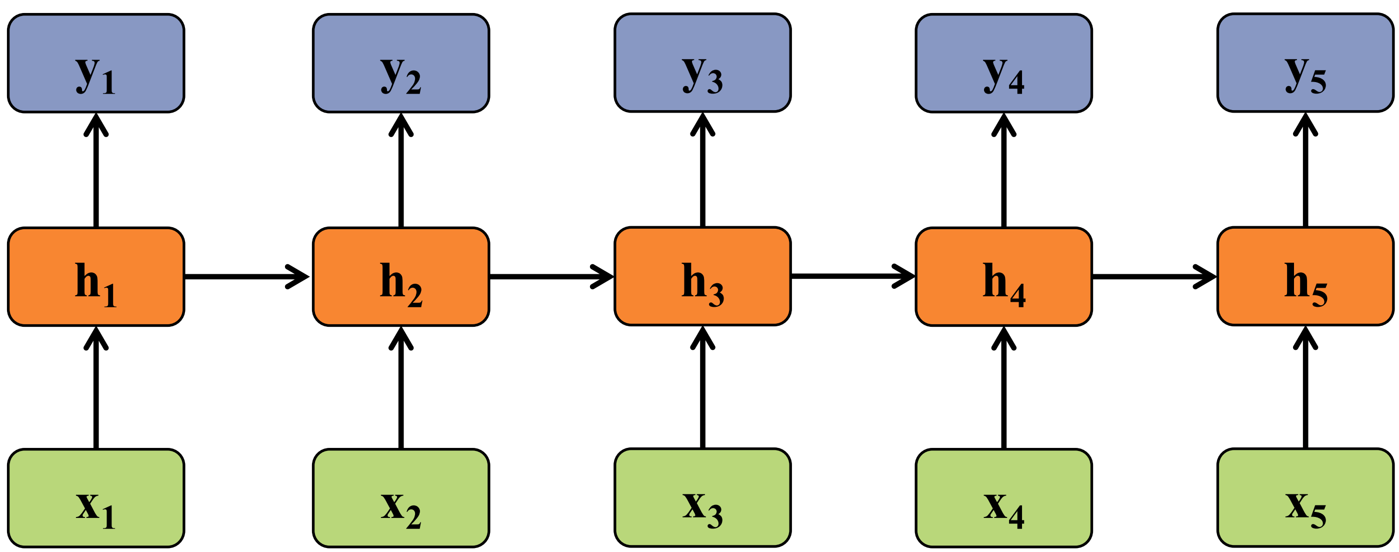
Recurrent Neural Networks (RNNs)

$$\begin{aligned}
\text{inputs:}\ \ &\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\
\text{hidden units:}\ \ &\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\
\text{outputs:}\ \ &\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\
\text{nonlinearity:}\ \ &\mathcal{H}
\end{aligned}$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



Recurrent Neural Networks (RNNs)

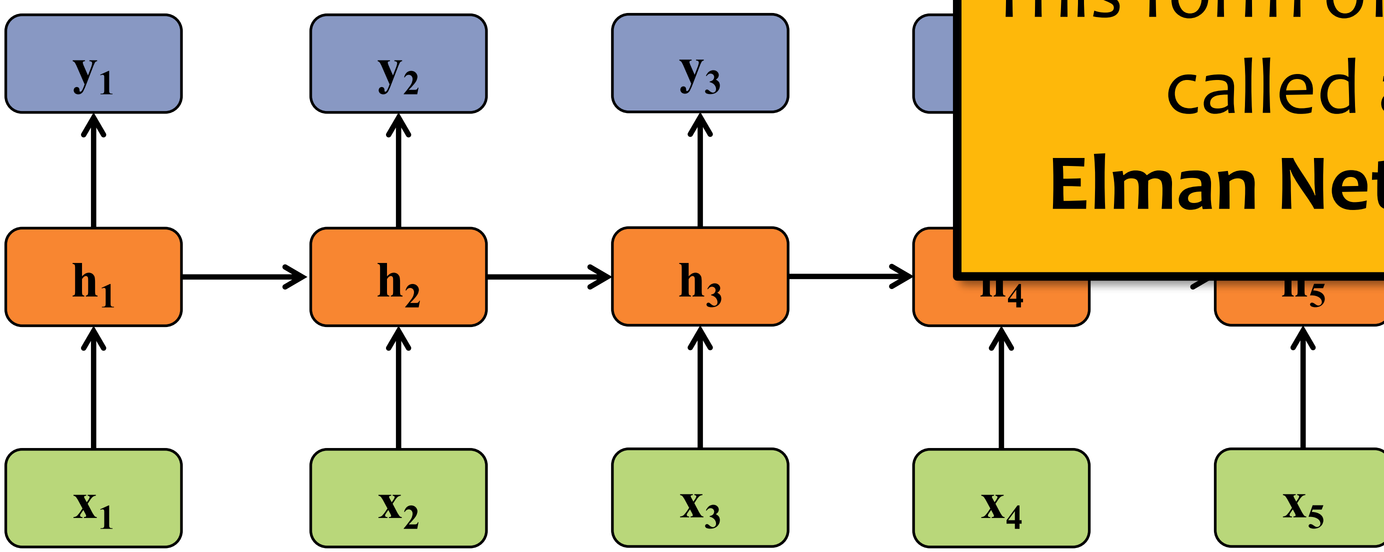
$$\begin{aligned} \text{inputs: } & \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\ \text{hidden units: } & \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\ \text{outputs: } & \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\ \text{nonlinearity: } & \mathcal{H} \end{aligned}$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$

This form of RNN is called an **Elman Network**



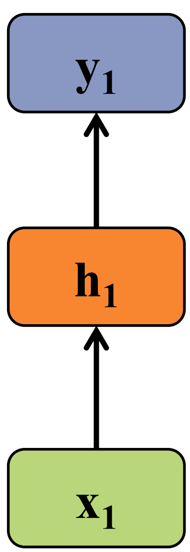
Recurrent Neural Networks (RNNs)

$$
\begin{aligned}
\text{inputs: } & \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\
\text{hidden units: } & \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\
\text{outputs: } & \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\
\text{nonlinearity: } & \mathcal{H}
\end{aligned}
$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



- If $T=1$, then we have a standard feed-forward **neural net with one hidden layer**
- All of the deep nets from last lecture required **fixed size inputs/outputs**

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

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

4. Train with SGD:

(take small steps opposite the gradient)

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

A Recipe for Machine Learning

Background

1.

- Recurrent Neural Networks (RNNs) provide another form of **decision function**
- An RNN is just another differential function

2. Choose each of these:

– Decision function

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

- We'll just need a method of computing the gradient efficiently
- Let's use Backpropagation Through Time...

$$\eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$

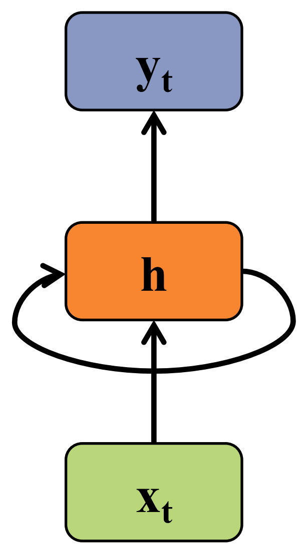
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh} x_t + W_{hh} h_{t-1} + b_h\right)$$

$$y_t = W_{hy} h_t + b_y$$



Recurrent Neural Networks (RNNs)

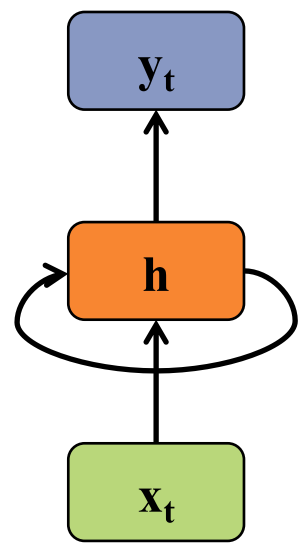
$$
\begin{aligned}
\text{inputs:}\ & \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\
\text{hidden units:}\ & \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\
\text{outputs:}\ & \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\
\text{nonlinearity:}\ & \mathcal{H}
\end{aligned}
$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

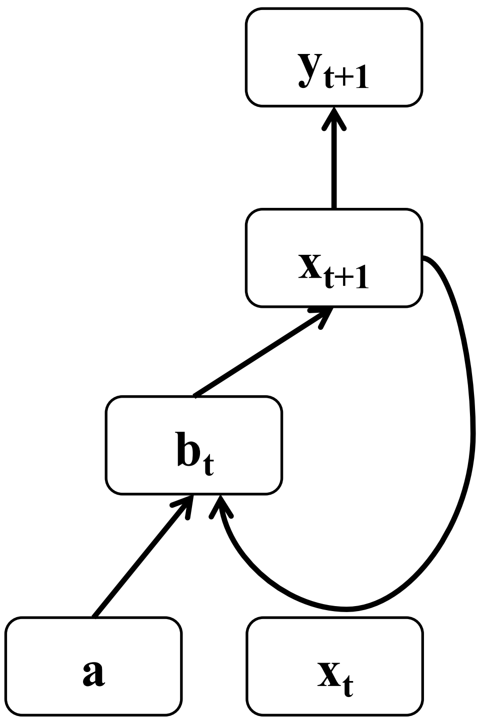
$$y_t = W_{hy}h_t + b_y$$

- By unrolling the RNN through time, we can **share parameters** and accommodate **arbitrary length** input/output pairs
- Applications: **time-series data** such as sentences, speech, stock-market, signal data, etc.



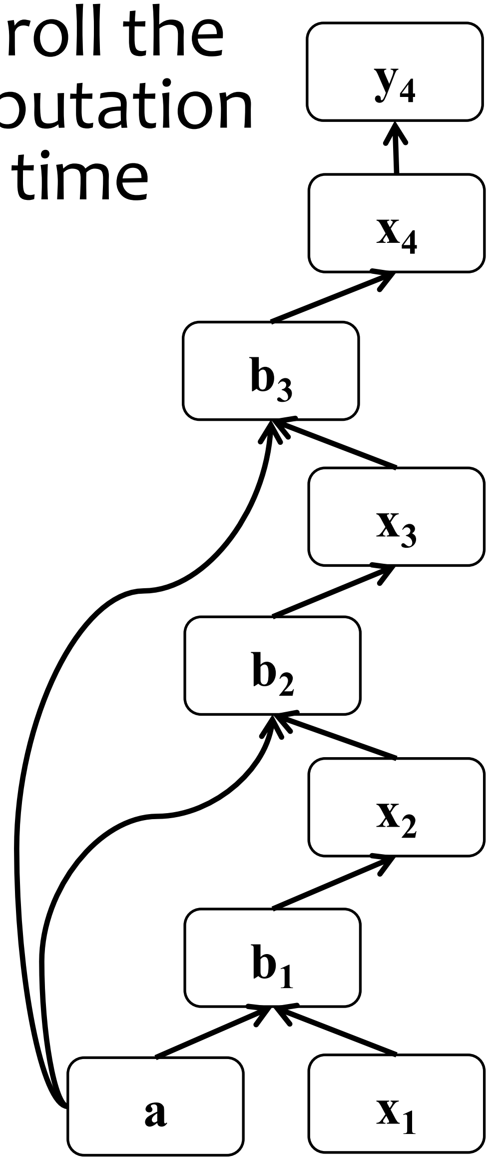
Background: Backprop through time

Recurrent neural network:



BPTT:

1. Unroll the computation over time



2. Run backprop through the resulting feed-forward network

(Robinson & Fallside, 1987)
(Werbos, 1988)
(Mozer, 1995)

Bidirectional RNN

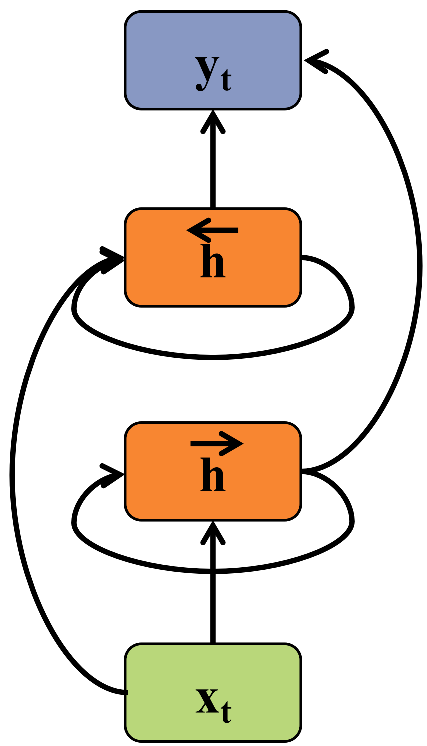
$$\begin{aligned} \text{inputs: } & \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\ \text{hidden units: } & \overrightarrow{\mathbf{h}} \text{ and } \overleftarrow{\mathbf{h}} \\ \text{outputs: } & \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\ \text{nonlinearity: } & \mathcal{H} \end{aligned}$$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Bidirectional RNN

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$

$$\text{hidden units: } \overrightarrow{\mathbf{h}} \text{ and } \overleftarrow{\mathbf{h}}$$

$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$

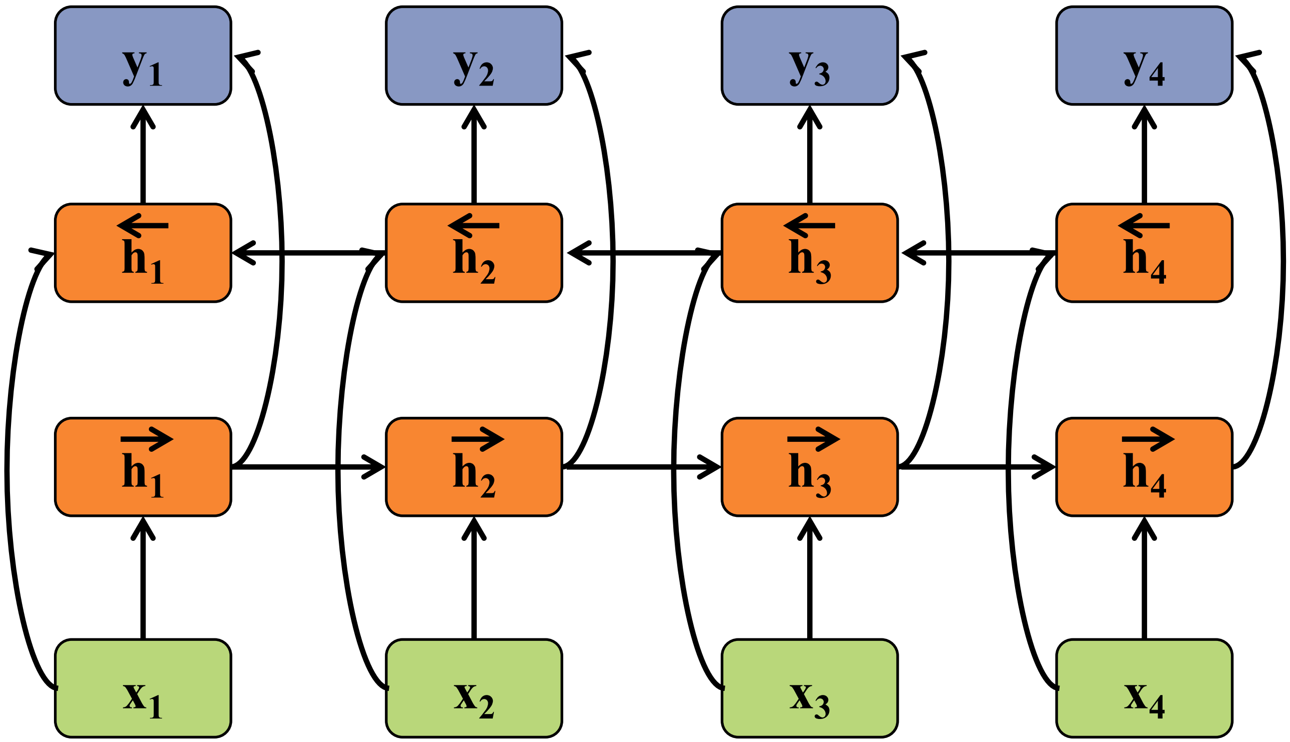
$$\text{nonlinearity: } \mathcal{H}$$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Bidirectional RNN

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

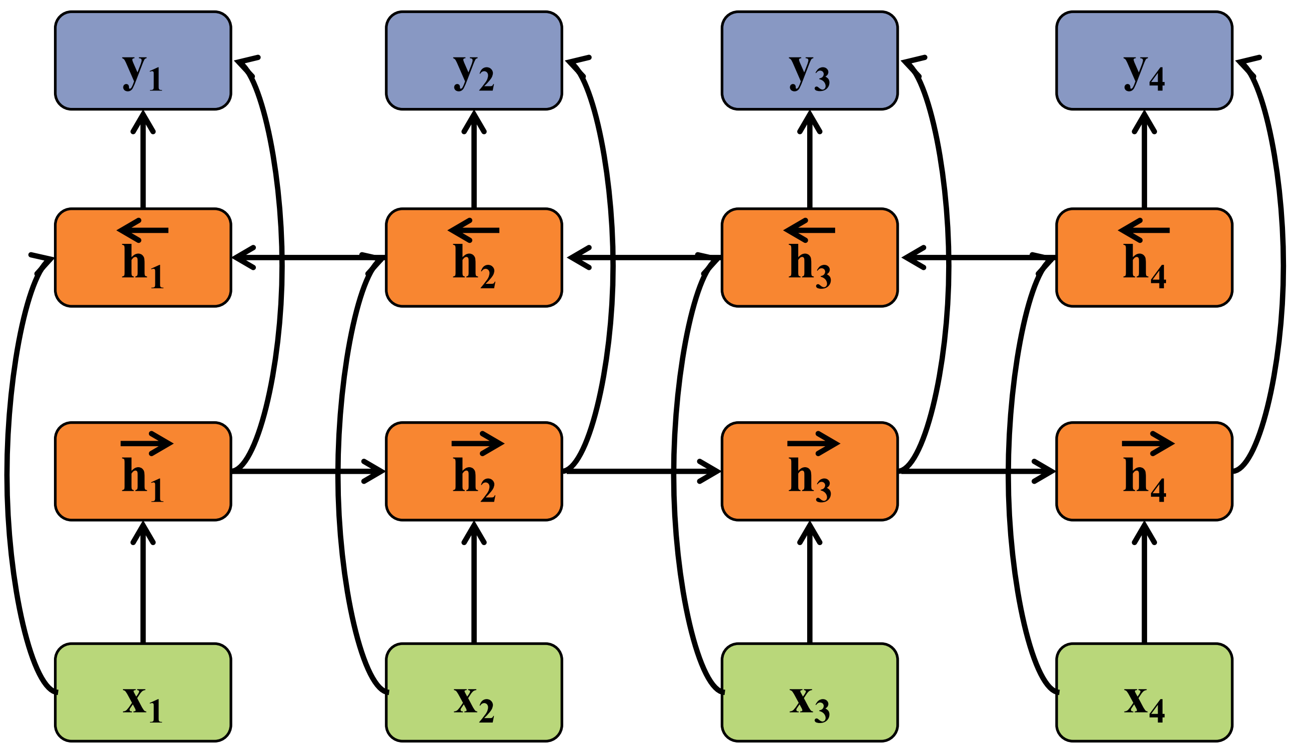
nonlinearity: $\mathcal{H}$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Bidirectional RNN

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

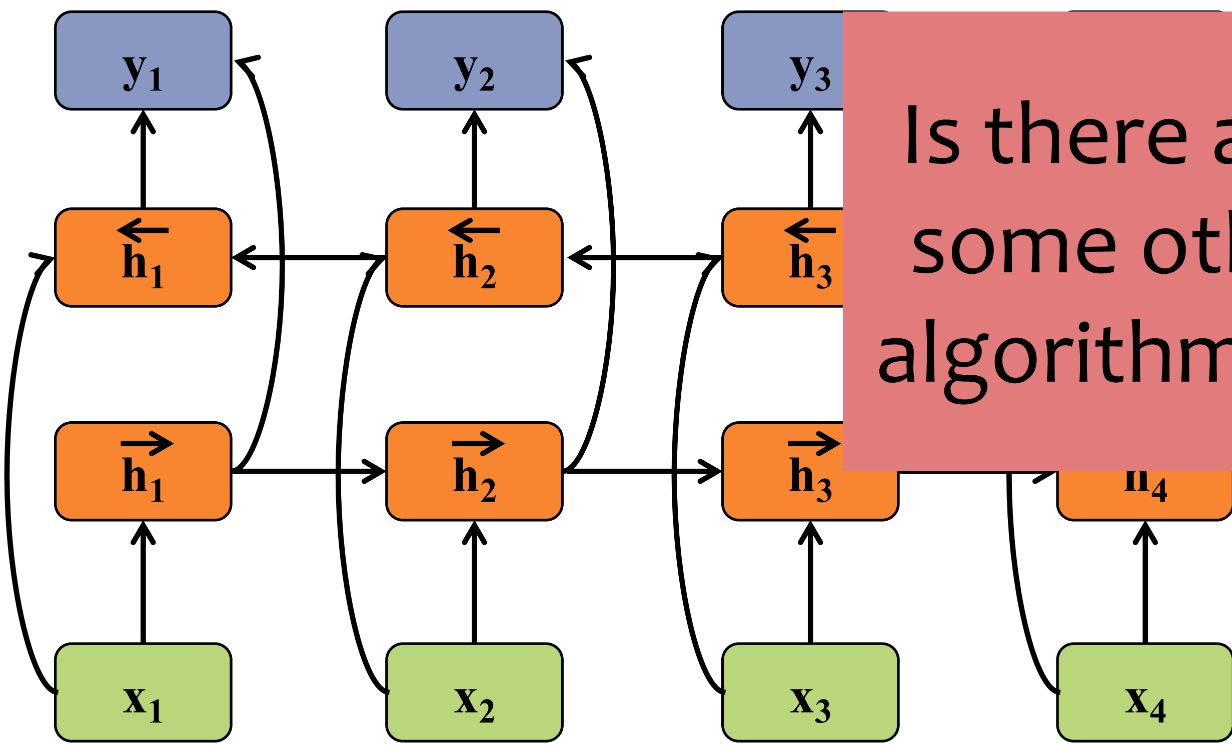
nonlinearity: $\mathcal{H}$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Is there an analogy to some other recursive algorithm(s) we know?

Deep RNNs

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Recursive Definition:

$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n\right)$$

$$y_t = W_{h^N y} h_t^N + b_y$$

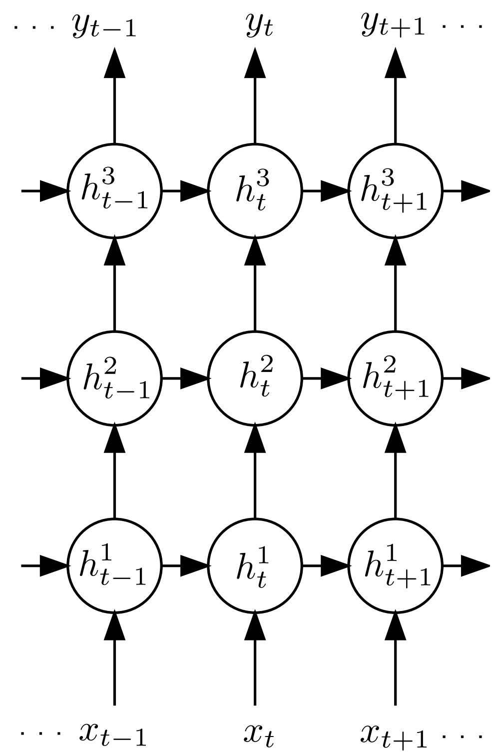


Figure from (Graves et al., 2013)

Deep Bidirectional RNNs

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$
$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$
$$\text{nonlinearity: } \mathcal{H}$$

- Notice that the upper level hidden units have input from **two previous layers** (i.e. wider input)
- Likewise for the output layer
- What analogy can we draw to DNNs, DBNs, DBMs?

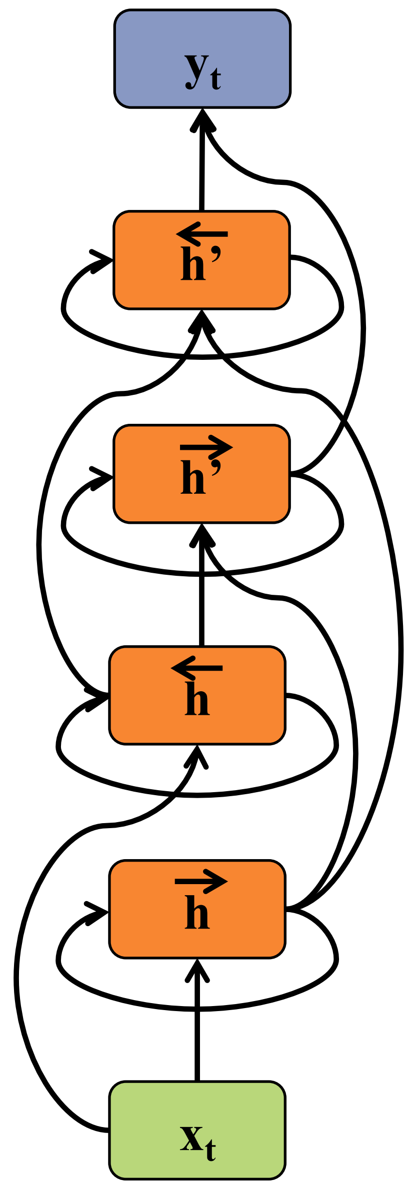
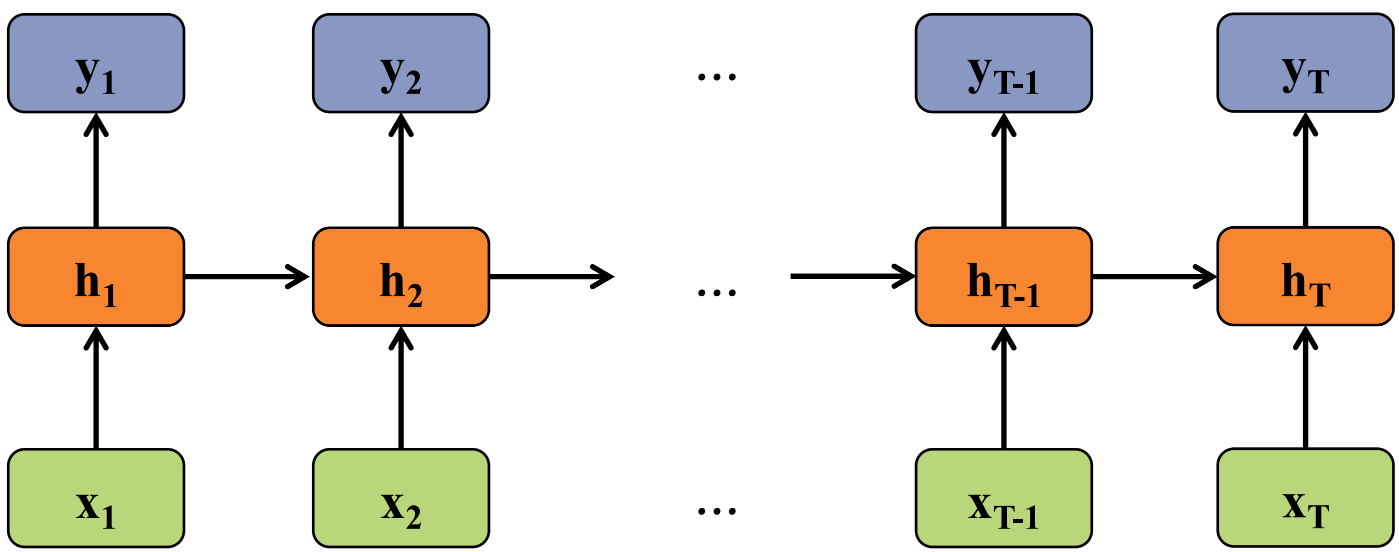


Figure from (Graves et al., 2013)

Long Short-Term Memory (LSTM)

Motivation:

- Standard RNNs have trouble learning long distance dependencies
- LSTMs combat this issue



Long Short-Term Memory (LSTM)

Motivation:
- Vanishing gradient problem for Standard RNNs
- Figure shows sensitivity (darker = more sensitive) to the input at time t=1

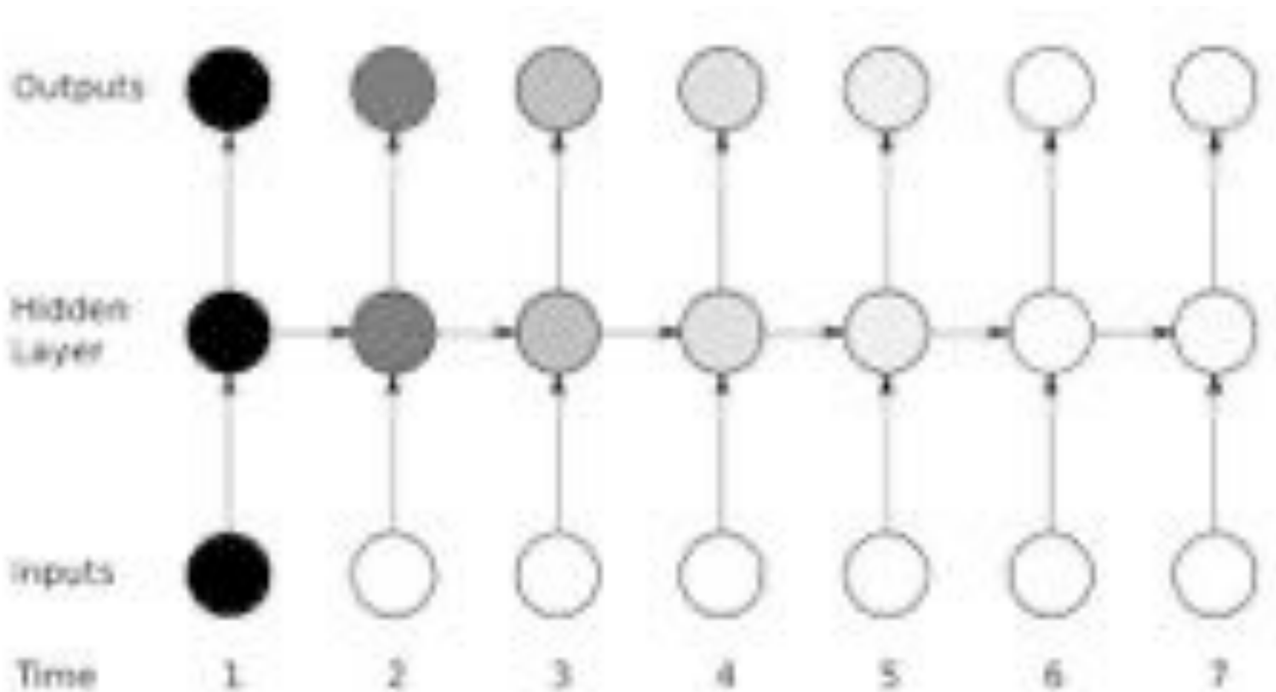


Figure from (Graves, 2012)

Long Short-Term Memory (LSTM)

Motivation:

- LSTM units have a rich internal structure
- The various "gates" determine the propagation of information and can choose to "remember" or "forget" information

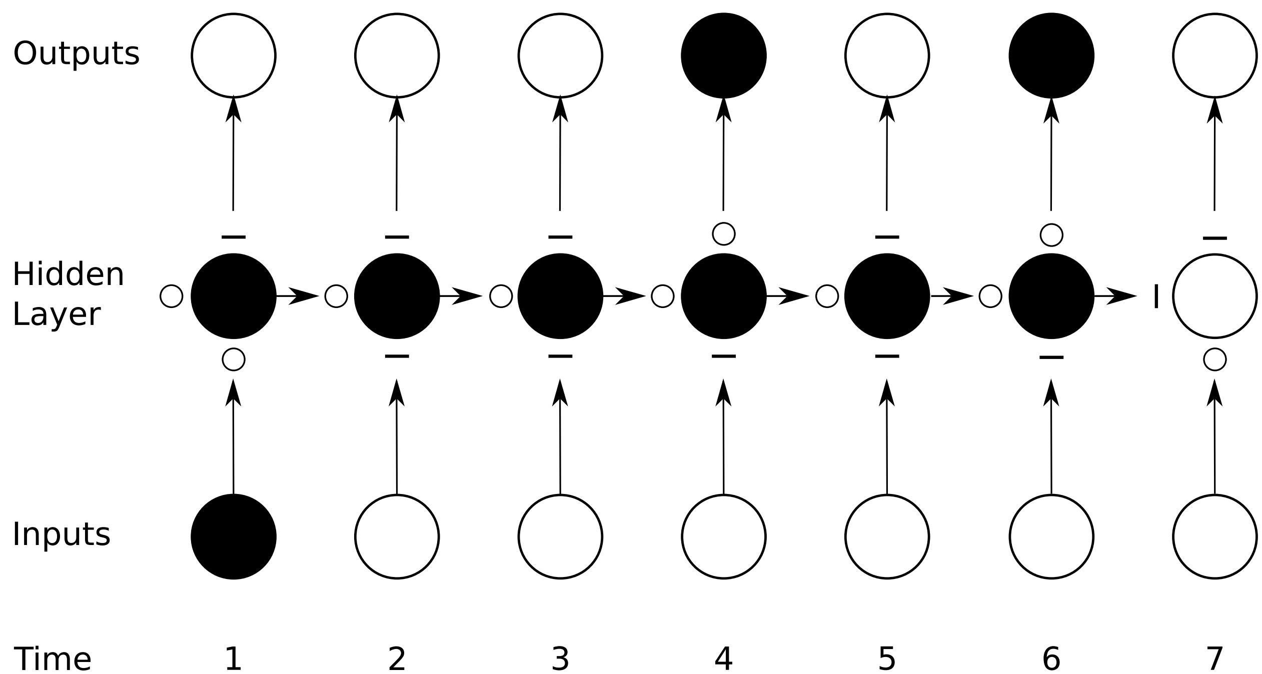
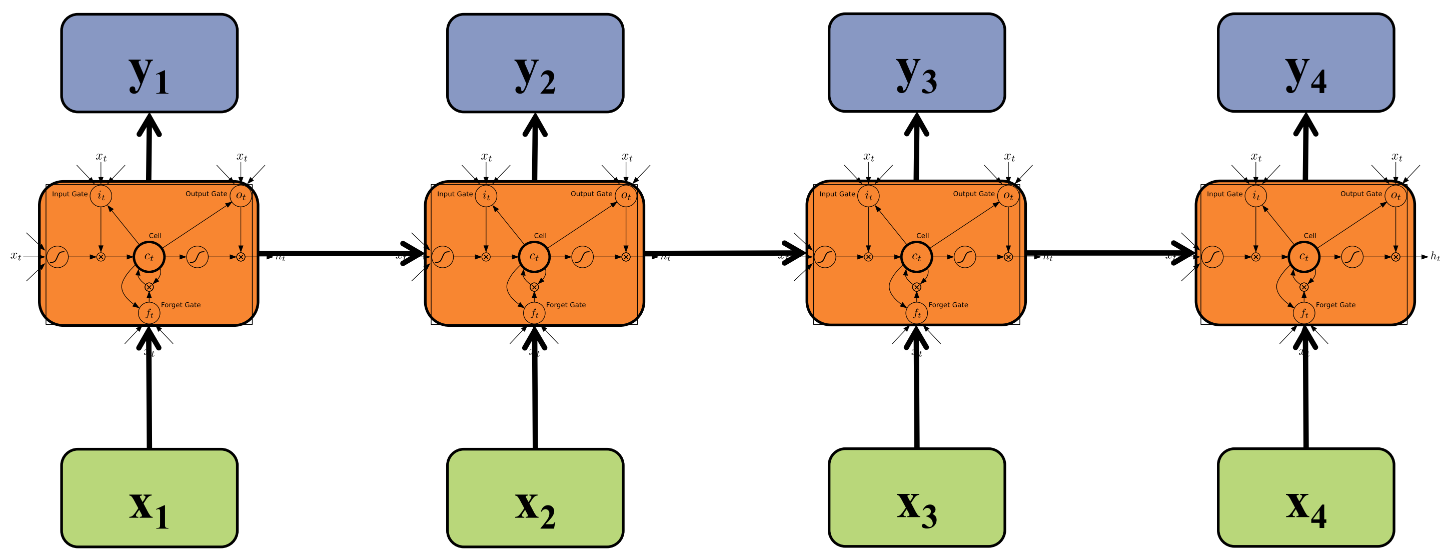


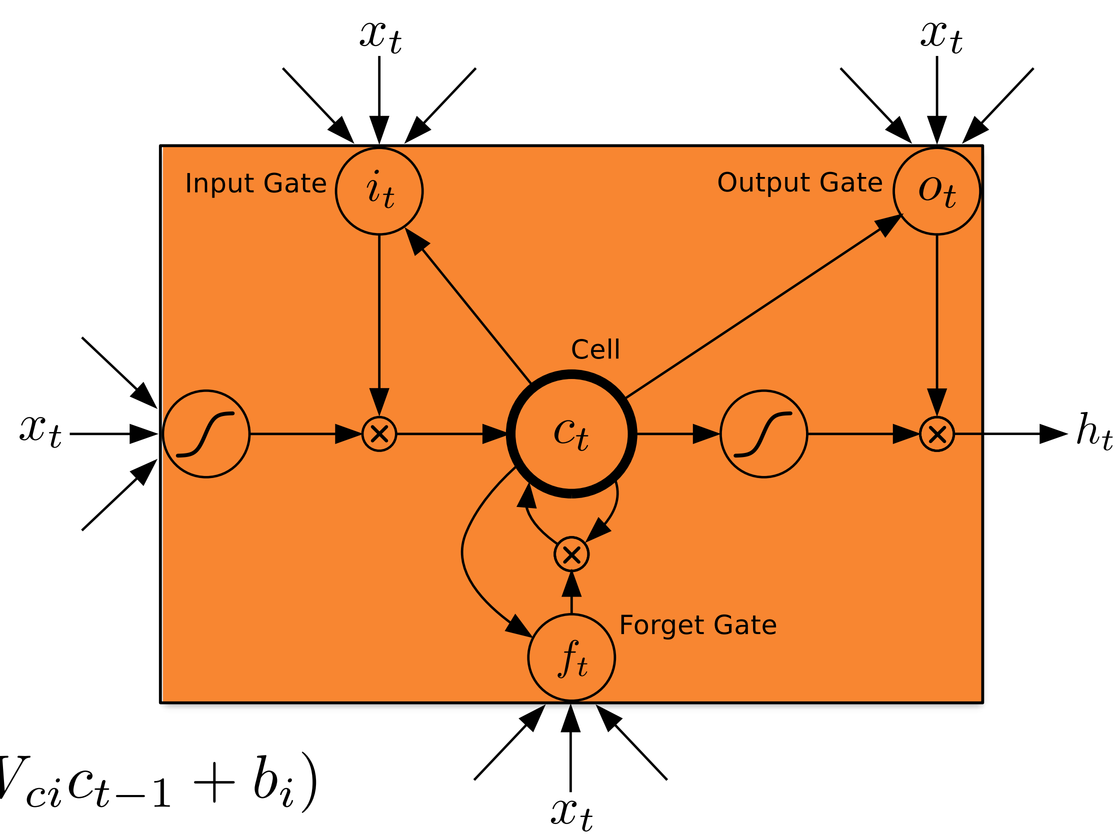
Figure from (Graves, 2012)

Long Short-Term Memory (LSTM)



Long Short-Term Memory (LSTM)

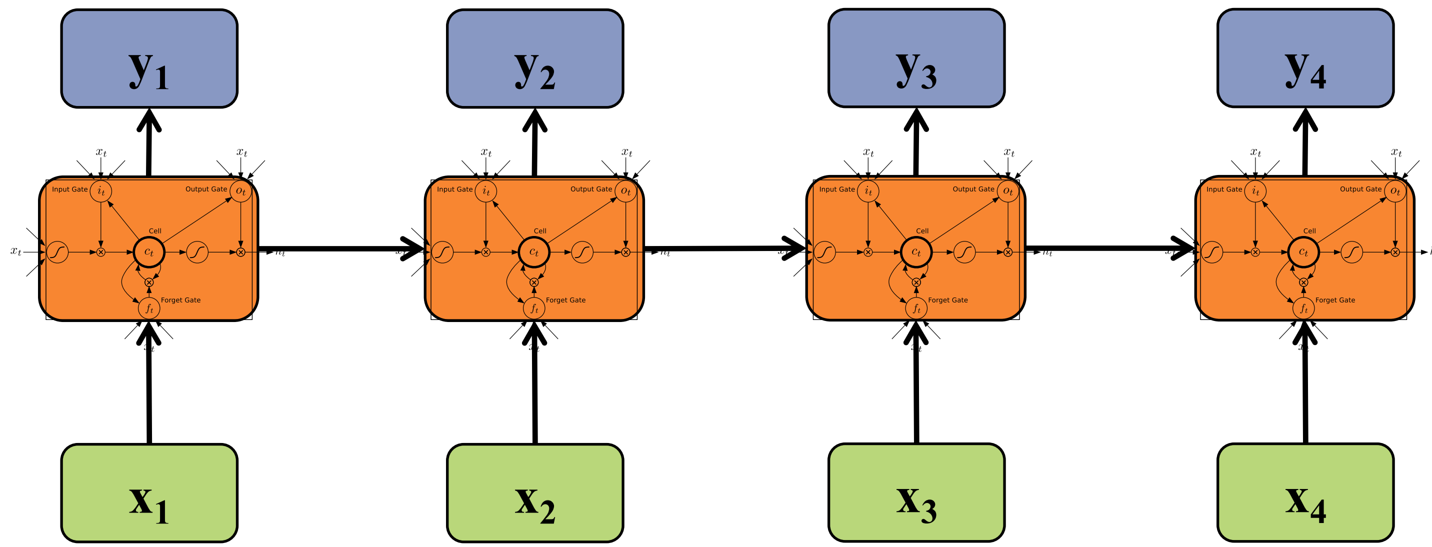
- **Input gate:** masks out the standard RNN inputs
- **Forget gate:** masks out the previous cell
- **Cell:** stores the input/forget mixture
- **Output gate:** masks out the values of the next hidden



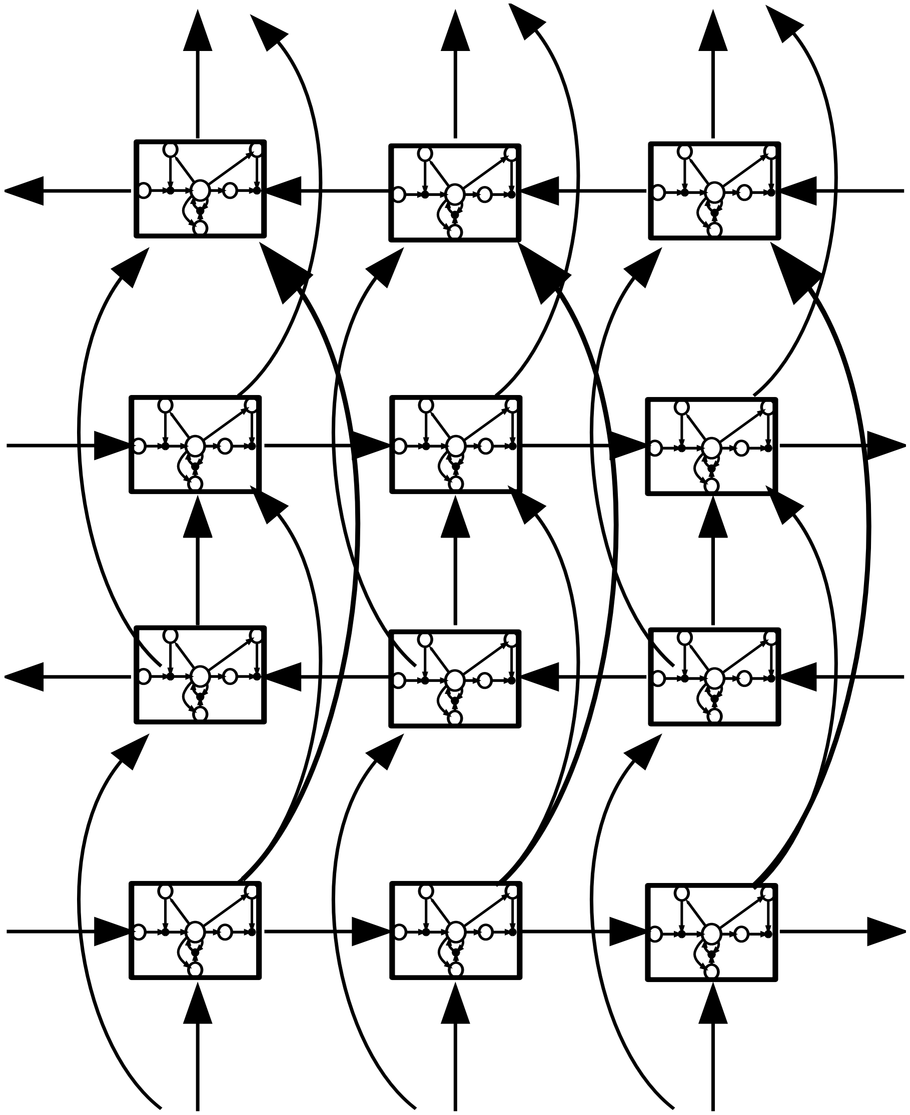
$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$
$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)$$
$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)$$
$$h_t = o_t \tanh(c_t)$$

Figure from (Graves et al., 2013)

Long Short-Term Memory (LSTM)



Deep Bidirectional LSTM (DBLSTM)



- Figure: input/output layers not shown
- **Same general topology** as a Deep Bidirectional RNN, but with **LSTM units** in the hidden layers
- No additional **representational power** over DBRNN, but **easier to learn** in practice

Figure from (Graves et al., 2013)

Deep Bidirectional LSTM (DBLSTM)

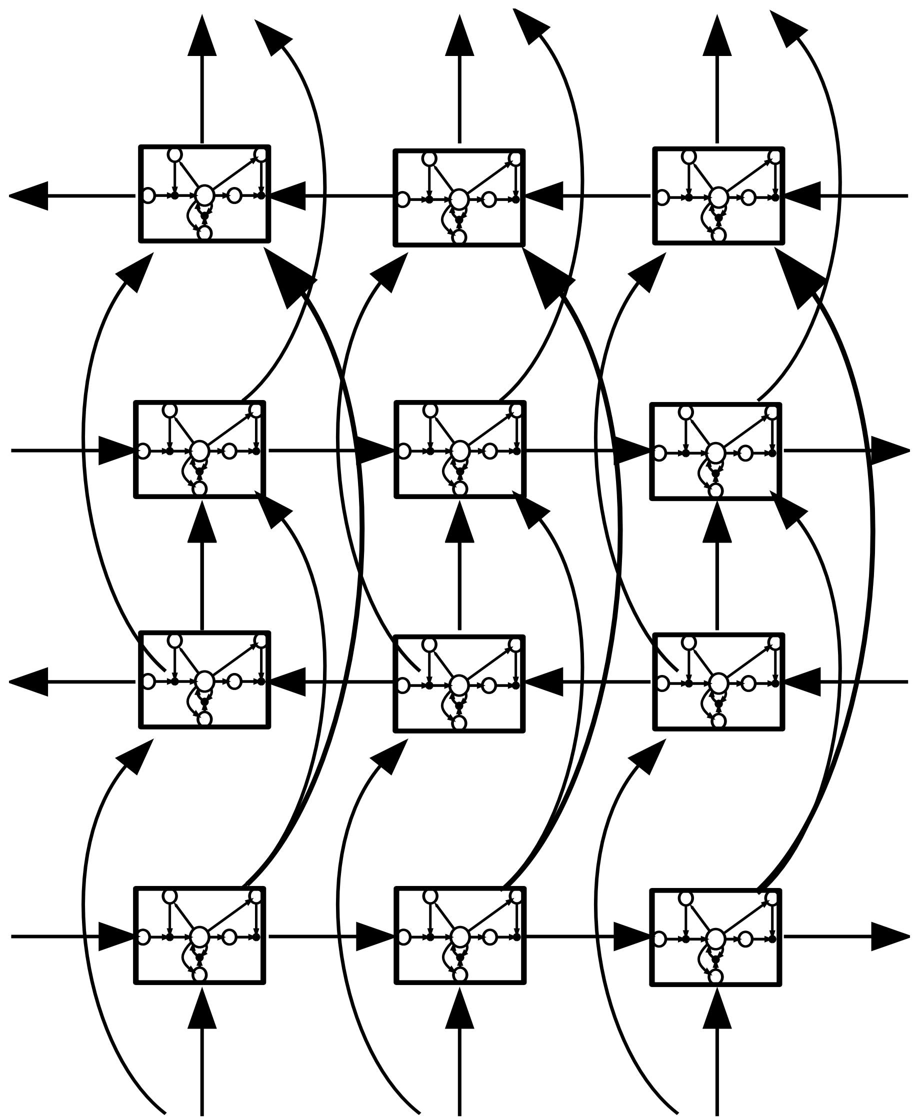


Figure from (Graves et al., 2013)

How important is this particular architecture?

Jozefowicz et al. (2015) **evaluated 10,000 different LSTM-like architectures** and found several variants that worked just as well on several tasks.

RNN Training Tricks

- Deep Learning models tend to consist largely of **matrix multiplications**
- Training tricks:
  - **mini-batching with masking**

	Metric	DyC++	DyPy	Chainer	DyC++ Seq	Theano	TF
RNNLM (MB=1)	words/sec	190	190	114	494	189	298
RNNLM (MB=4)	words/sec	830	825	295	1510	567	473
RNNLM (MB=16)	words/sec	1820	1880	794	2400	1100	606
RNNLM (MB=64)	words/sec	2440	2470	1340	2820	1260	636

  - **sorting into buckets of similar-length sequences,** so that mini-batches have same length sentences
  - **truncated BPTT,** when sequences are too long, divide sequences into chunks and use the final vector of the previous chunk as the initial vector for the next chunk (but don't backprop from next chunk to previous chunk)

Table from Neubig et al. (2017)

RNN Summary

- **RNNs**
  - Applicable to tasks such as **sequence labeling**, speech recognition, machine translation, etc.
  - Able to **learn context features** for time series data
  - Vanishing gradients are still a problem – but **LSTM units** can help
- **Other Resources**
  - Christopher Olah's blog post on LSTMs http://colah.github.io/posts/2015-08-Understanding-LSTMs/

HYBRIDS OF NEURAL NETWORKS WITH GRAPHICAL MODELS

Outline of Examples

- **Hybrid NN + HMM**
  - Model: neural net for emissions
  - Learning: backprop for end-to-end training
  - Experiments: phoneme recognition (Bengio et al., 1992)
- **Hybrid RNN + HMM**
  - Model: neural net for emissions
  - Experiments: phoneme recognition (Graves et al., 2013)
- **Hybrid CNN + CRF**
  - Model: neural net for factors
  - Experiments: natural language tasks (Collobert & Weston, 2011)
  - Experiments: pose estimation
- **Tricks of the Trade**

HYBRID:
NEURAL NETWORK + HMM

Recall...

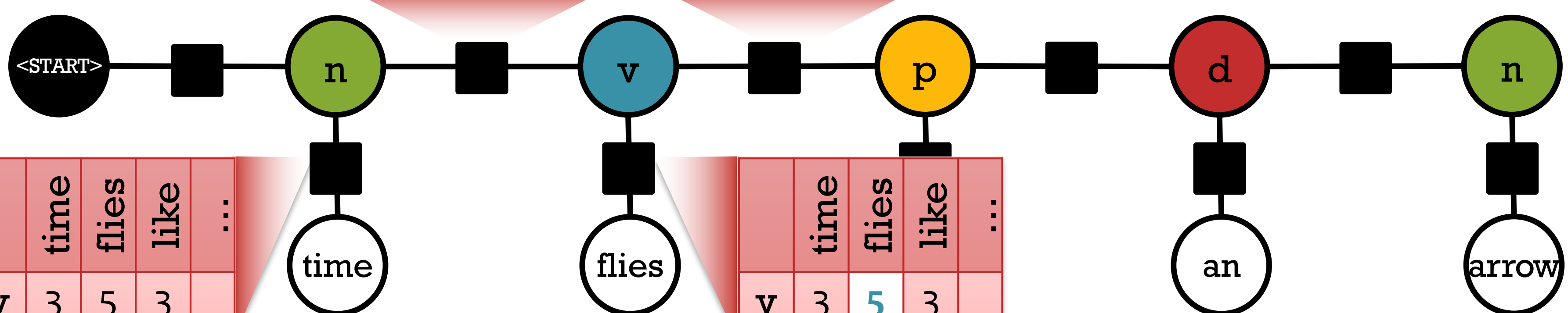
Markov Random Field (MRF)

Joint distribution over tags Y_i and words X_i
The individual factors aren't *necessarily* probabilities.

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0



	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

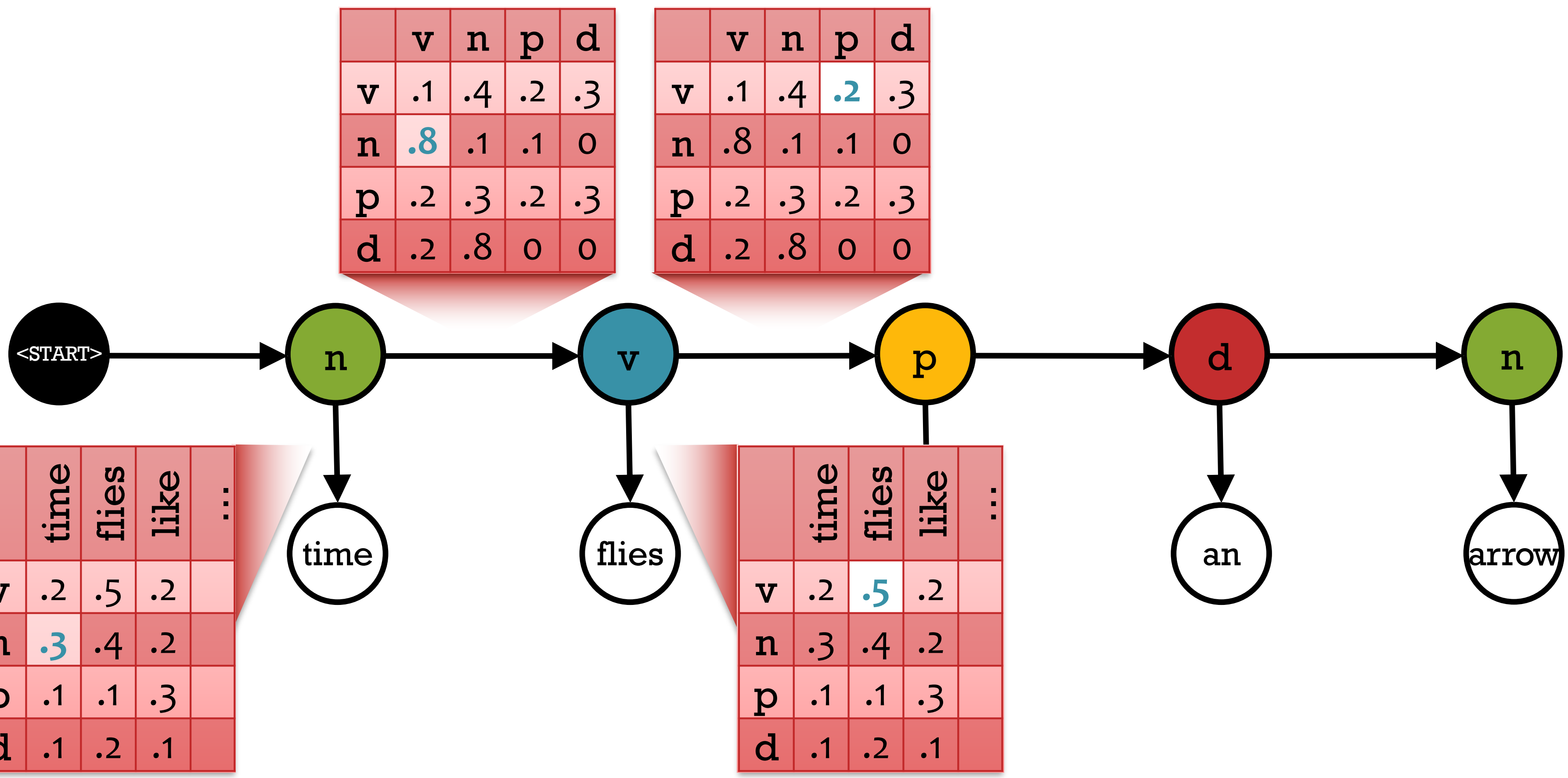
	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

Recall...

Hidden Markov Model

But sometimes we *choose* to make them probabilities.
Constrain each row of a factor to sum to one. Now $Z = 1$.

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = \cancel{\frac{1}{Z}}(.3 * .8 * .2 * .5 * \ldots)$$



	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

Hybrid: NN + HMM

(Bengio et al., 1992)

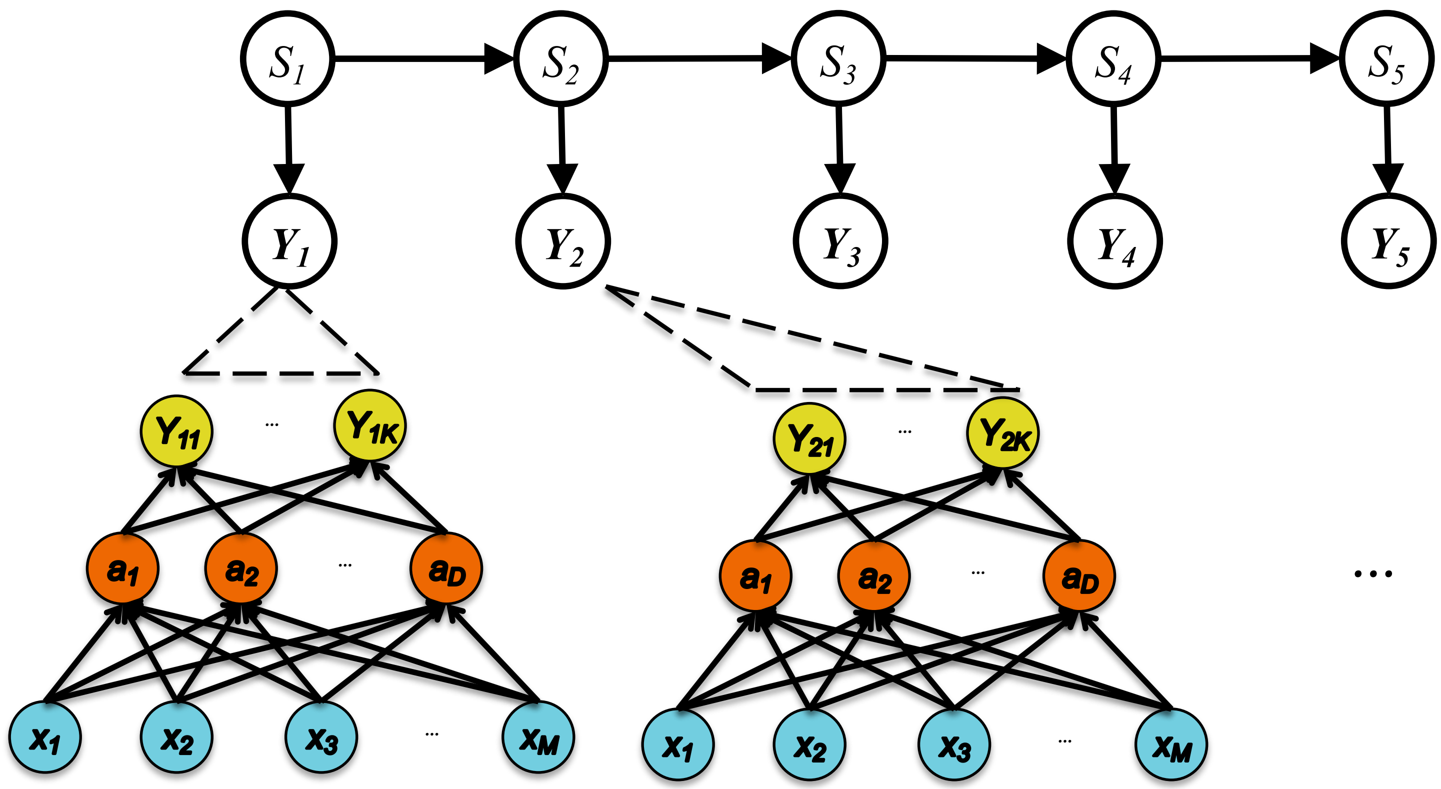
Discrete HMM state: $S_t \in \{/p/, /t/, /k/, /b/, /d/, \ldots, /g/\}$

Continuous HMM emission: $Y_t \in \mathcal{R}^K$

HMM: $p(\mathbf{Y}, \mathbf{S}) = \prod_{t=1}^{T} p(Y_t|S_t) p(S_t|S_{t-1})$

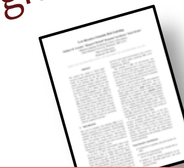
Gaussian emission:

$$p(Y_t|S_t = i) = b_{i,t} = \sum_k \frac{Z_k}{((2\pi)^n \mid \Sigma_k \mid)^{1/2}} \exp(-\frac{1}{2}(Y_t - \mu_k)\Sigma_k^{-1}(Y_t - \mu_k)^T)$$



Hybrid: NN + HMM

(Bengio et al., 1992)

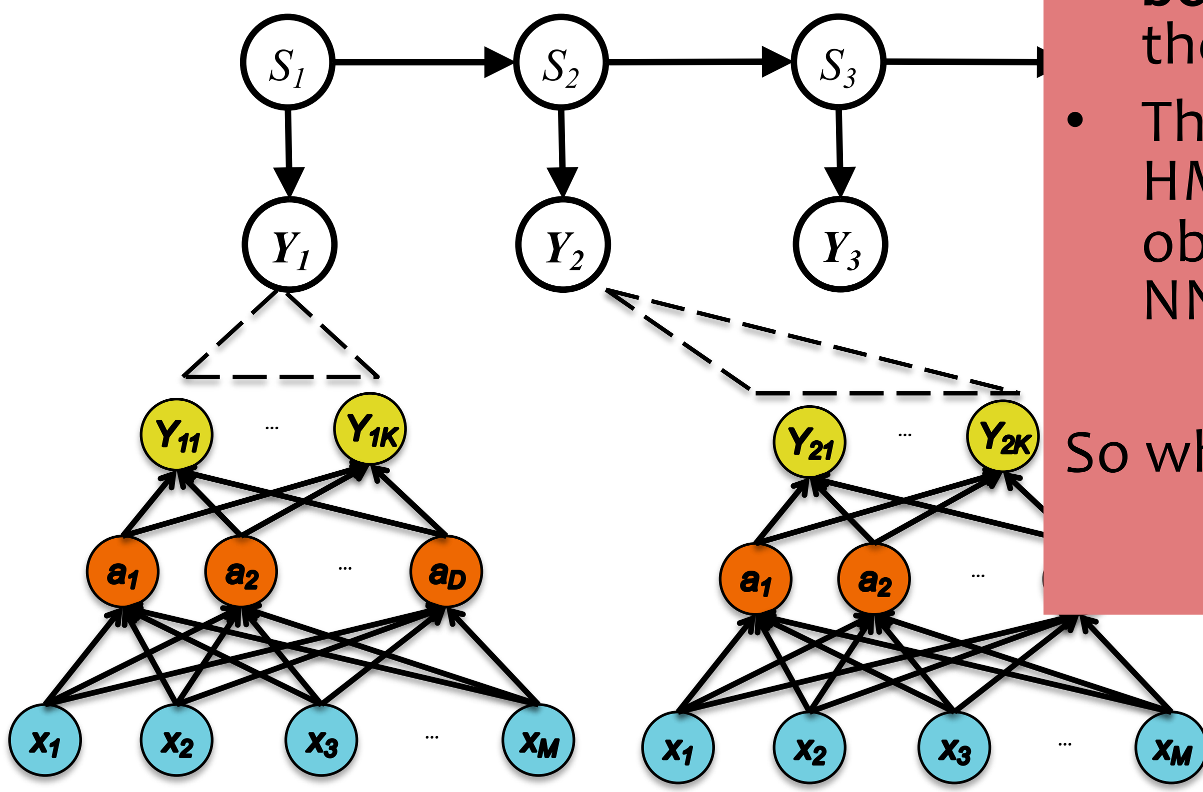


Discrete HMM state: $S_t \in \{/p/, /t/, /k/, /b/, /d/, \ldots, /g/\}$

Continuous HMM emission: $Y_t \in \mathcal{R}^K$

HMM: $p(\mathbf{Y}, \mathbf{S}) = \prod_{t=1}^{T} p(Y_t|S_t) p(S_t|S_{t-1})$

$$p(Y_t|S_t = i) = b_{i,t} = \sum_k \frac{Z_k}{((2\pi)^n \mid \Sigma_k \mid)^{1/2}} e$$

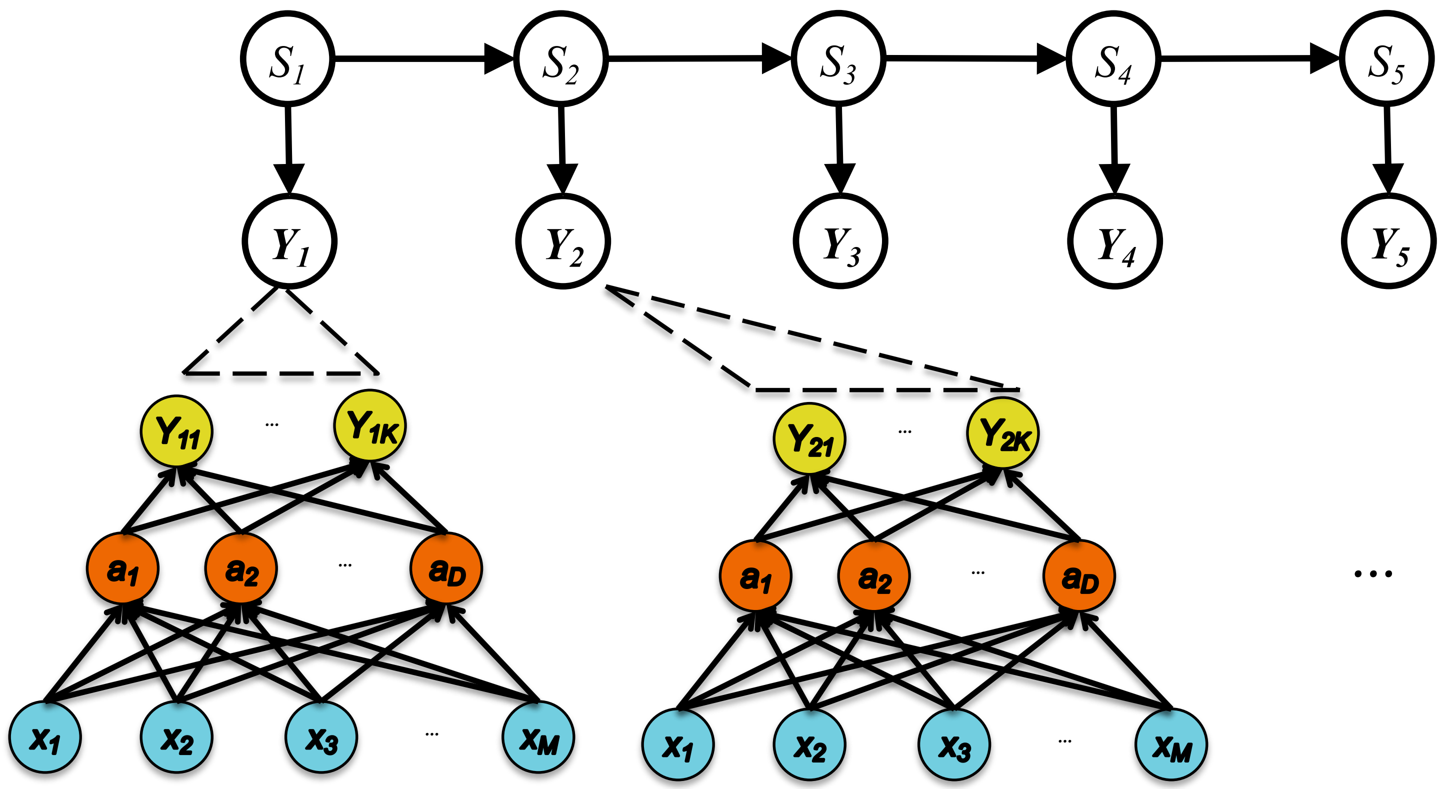


Lots of oddities to this picture:

- **Clashing visual notations** (graphical model vs. neural net)
- HMM generates data **top-down,** NN generates **bottom-up** and they meet in the middle.
- The "observations" of the HMM are not actually observed (i.e. x's appear in NN only)

So what are we missing?

Hybrid: NN + HMM



$$a_{i,j} = p(S_t = i | S_{t-1} = j)$$
$$b_{i,t} = p(Y_t | S_t = i)$$

Hybrid: NN + HMM

Forward-backward algorithm: a "feed-forward" algorithm for computing alpha-beta probabilities.

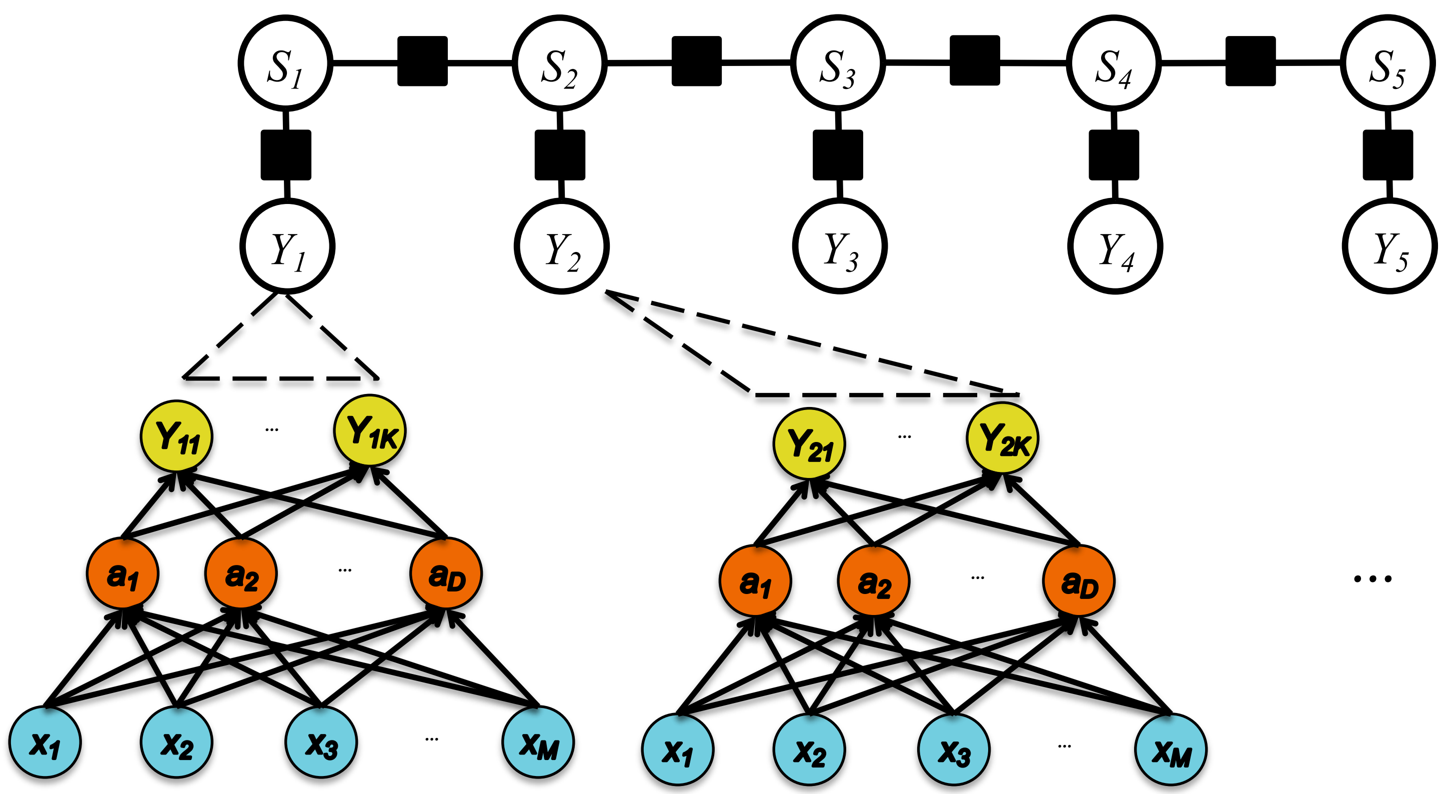
$$\alpha_{i,t} = P(Y_1^t \text{ and } S_t = i \mid model) = b_{i,t} \sum_j a_{ji} \alpha_{j,t-1}$$

$$\beta_{i,t} = P(Y_{t+1}^T \mid S_t = i \text{ and } model) = \sum_j a_{ij} b_{j,t+1} \beta_{j,t+1}$$

$$\gamma_{i,t} = P(S_t = i \mid Y_1^t \text{ and } model) = \alpha_{i,t}\, \beta_{i,t}$$

Log-likelihood: a "feed-forward" objective function.

$$\log p(\mathbf{S}, \mathbf{Y}) = \alpha_{\text{END},T}$$



A Recipe for Graphical Models

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:
   - Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

Decision / Loss Function for Hybrid NN + HMM

Forward-backward algorithm: a "feed-forward" algorithm for computing alpha-beta probabilities.

$$\alpha_{i,t} = P(Y_1^t \text{ and } S_t = i \mid model) = b_{i,t} \sum_j a_{ji} \alpha_{j,t-1}$$

$$\beta_{i,t} = P(Y_{t+1}^T \mid S_t = i \text{ and } model) = \sum_j a_{ij} b_{j,t+1} \beta_{j,t+1}$$

$$\gamma_{i,t} = P(S_t = i \mid Y_1^t \text{ and } model) = \alpha_{i,t}\, \beta_{i,t}$$

Log-likelihood: a "feed-forward" objective function.

$$\log p(\mathbf{S}, \mathbf{Y}) = \alpha_{\text{END},T}$$

nt)

$- \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$

How do we compute the gradient?

Backpropagation

Recall...

Backpropagation is just repeated application of the **chain rule** from Calculus 101.

Graphical Model and Log-likelihood

Neural Network

$$\boldsymbol{y} = g(\boldsymbol{u}) \text{ and } \boldsymbol{u} = h(\boldsymbol{x}).$$

How to compute these partial derivatives?

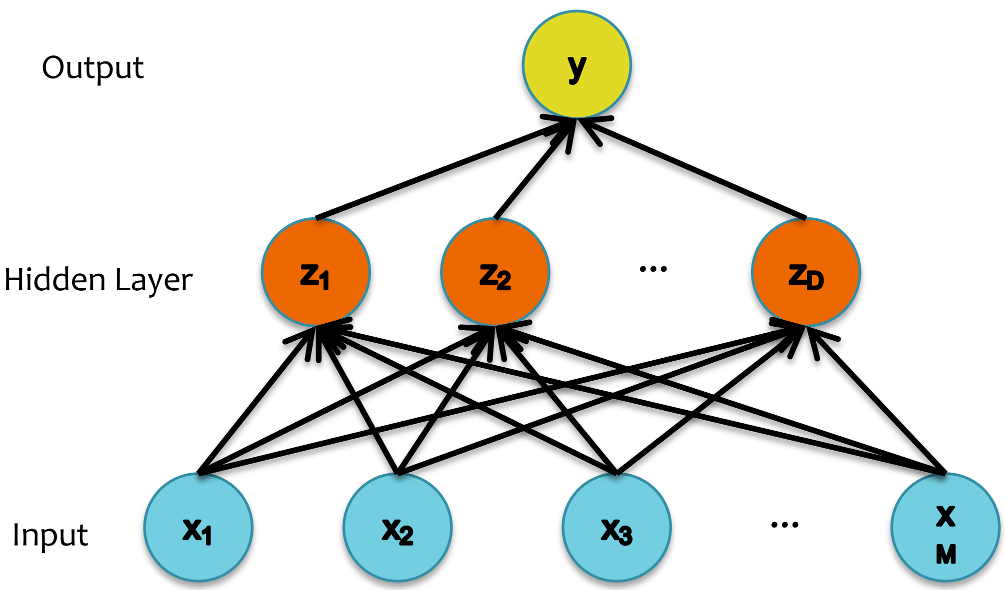
Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j}\frac{du_j}{dx_k}, \quad \forall i, k$$

Recall…

Training Backpropagation

What does this picture actually mean?



(F) Loss
$J = \frac{1}{2}(y - y^{(d)})^2$

(E) Output (sigmoid)
$y = \frac{1}{1+\exp(b)}$

(D) Output (linear)
$b = \sum_{j=0}^{D} \beta_j z_j$

(C) Hidden (sigmoid)
$z_j = \frac{1}{1+\exp(a_j)}, \ \forall j$

(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) Input
Given $x_i, \ \forall i$

Training

Backpropagation

Recall...

Case 2: Neural Network

Forward

$$J = y^* \log q + (1 - y^*) \log(1 - q)$$

$$q = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dq} = \frac{y^*}{q} + \frac{(1 - y^*)}{q - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(b)}{(\exp(b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(a_j)}{(\exp(a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$$

Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$

Forward computation

$$\log p(\mathbf{S}, \mathbf{Y}) = \alpha_{\text{END},T}$$

$\alpha_{i,t} = \ldots$ (forward prob)

$\beta_{i,t} = \ldots$ (backward prop)

$\gamma_{i,t} = \ldots$ (marginals)

$a_{i,j} = \ldots$ (transitions)

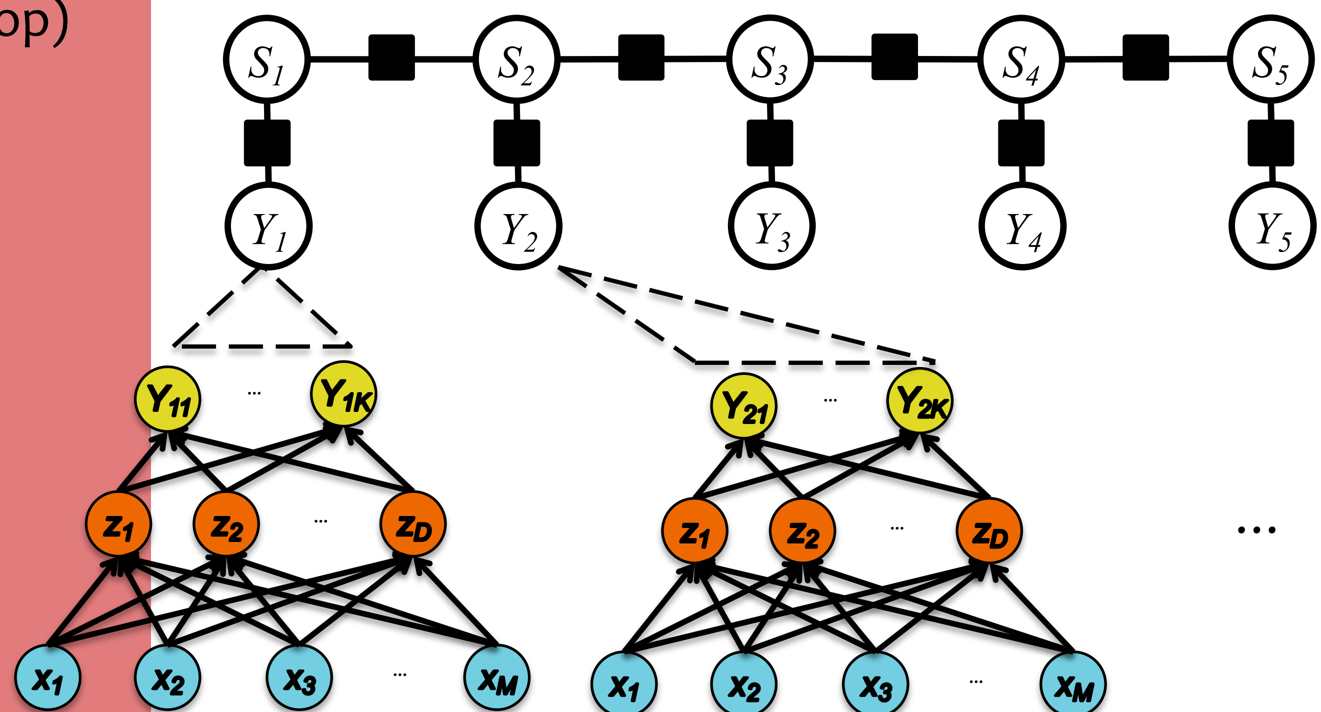
$b_{i,t} = \ldots$ (emissions)

$$y_{tk} = \frac{1}{1+\exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1+\exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$



Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$

Forward computation

$$J = \log p(\mathbf{S}, \mathbf{Y}) = \alpha_{\text{END},T}$$

$\alpha_{i,t} = \ldots$ (forward prob)

$\beta_{i,t} = \ldots$ (backward prop)

$\gamma_{i,t} = \ldots$ (marginals)

$a_{i,j} = \ldots$ (transitions)

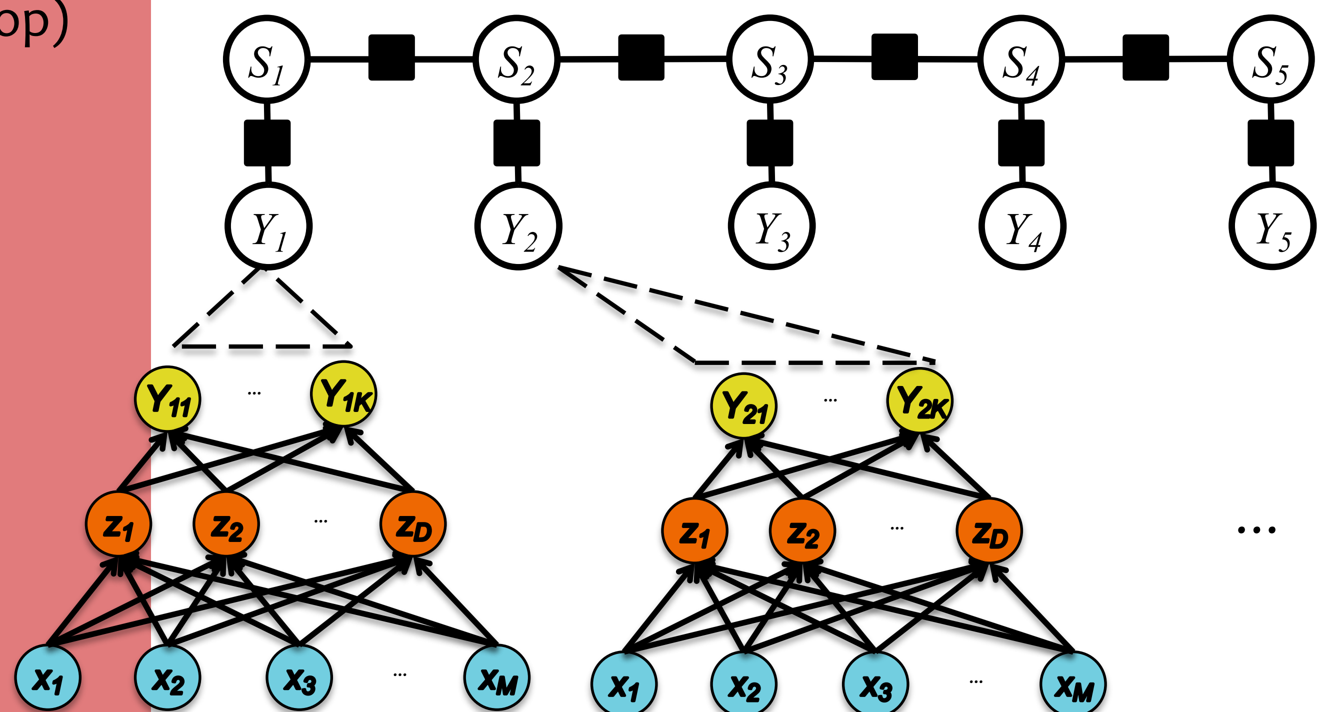
$b_{i,t} = \ldots$ (emissions)

$$y_{tk} = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$



Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$

Forward computation

$$J = \log p(\mathbf{S}, \mathbf{Y}) = \alpha_{\text{END},T}$$

$$\alpha_{i,t} = \ldots \text{(forward prob)}$$

$$\beta_{i,t} = \ldots \text{(backward prop)}$$

$$\gamma_{i,t} = \ldots \text{(marginals)}$$

$$a_{i,j} = \ldots \text{(transitions)}$$

$$b_{i,t} = \ldots \text{(emissions)}$$

$$y_{tk} = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward computation

$$\frac{dJ}{db_{i,t}} = \frac{\partial \alpha_{F_{model},T}}{\partial \alpha_{i,t}} \frac{\partial \alpha_{i,t}}{\partial b_{i,t}} = \left(\sum_j \frac{\partial \alpha_{j,t+1}}{\partial \alpha_{i,t}} \frac{\partial L_{model}}{\partial \alpha_{j,t+1}}\right)\left(\sum_j a_{ji} \alpha_{j,t-1}\right)$$

$$= \left(\sum_j b_{j,t+1} a_{ji} \frac{\partial \alpha_{F_{model},T}}{\partial \alpha_{j,t+1}}\right)\left(\sum_j a_{ji} \alpha_{j,t-1}\right) = \beta_{i,t} \frac{\alpha_{i,t}}{b_{i,t}} = \frac{\gamma_{i,t}}{b_{i,t}}$$

Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$

Forward computation

$$J = \log p(\mathbf{S}, \mathbf{Y}) = \alpha_{\text{END},T}$$

$$\alpha_{i,t} = \ldots \text{(forward prob)}$$

$$\beta_{i,t} = \ldots \text{(backward prop)}$$

$$\gamma_{i,t} = \ldots \text{(marginals)}$$

$$a_{i,j} = \ldots \text{(transitions)}$$

$$b_{i,t} = \ldots \text{(emissions)}$$

$$y_{tk} = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward computation

$$\frac{dJ}{db_{i,t}} = \frac{\gamma_{i,t}}{b_{i,t}}$$

$$\frac{dJ}{dy_{t,k}} = \sum_{b_{i,t}} \frac{dJ}{db_{i,t}} \frac{db_{i,t}}{dy_{t,k}}$$

$$\frac{\partial b_{i,t}}{\partial Y_{jt}} = \sum_k \frac{Z_k}{((2\pi)^n \mid \Sigma_k \mid)^{1/2}} \left(\sum_l d_{k,lj}(\mu_{kl} - Y_{lt})\right) \exp\left(-\frac{1}{2}(Y_t - \mu_k)\Sigma_k^{-1}(Y_t - \mu_k)^T\right)$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(b)}{(\exp(b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(a_j)}{(\exp(a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$

Forward computation

$$J = \log p(\mathbf{S}, \mathbf{Y}) = \alpha_{\text{END},T}$$

$\alpha_{i,t} = \ldots$ (forward prob)

$\beta_{i,t} = \ldots$ (backward prop)

$\gamma_{i,t} = \ldots$ (marginals)

The derivative of the log-likelihood with respect to the neural network parameters!

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward computation

$$\frac{dJ}{db_{i,t}} = \frac{\gamma_{i,t}}{b_{i,t}}$$

$$\frac{dJ}{dy_{t,k}} = \sum_{b_{i,t}} \frac{dJ}{db_{i,t}} \frac{db_{i,t}}{dy_{t,k}}$$

$$\frac{\partial b_{i,t}}{\partial Y_{jt}} = \sum_k \frac{Z_k}{((2\pi)^n \mid \Sigma_k \mid)^{1/2}} (\sum_l d_{k,lj}(\mu_{kl} - Y_{lt})) \exp(-\frac{1}{2}(Y_t - \mu_k)\Sigma_k^{-1}(Y_t - \mu_k)^T)$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(b)}{(\exp(b)+1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(a_j)}{(\exp(a_j)+1)^2}$$

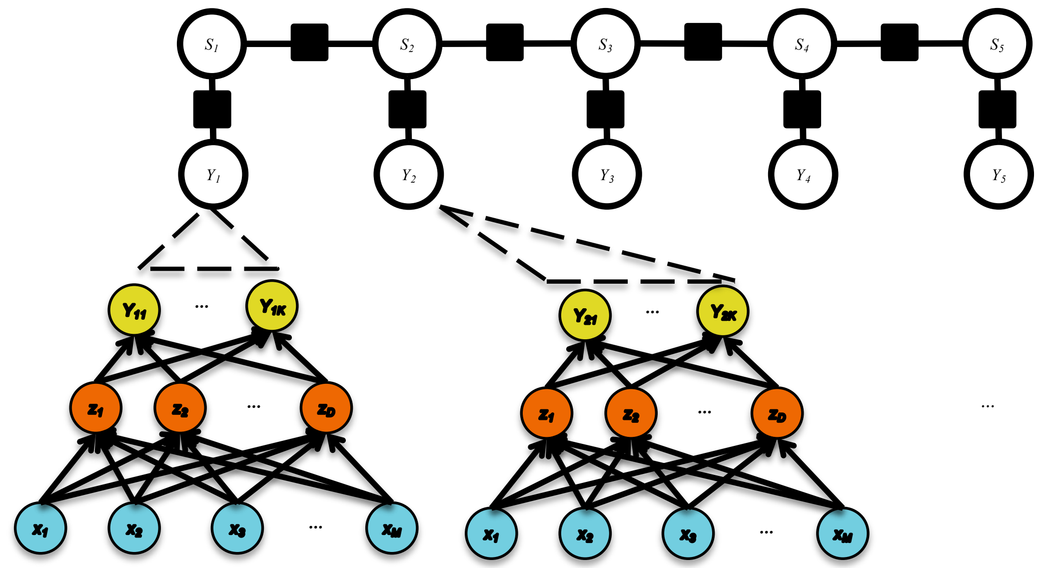
$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

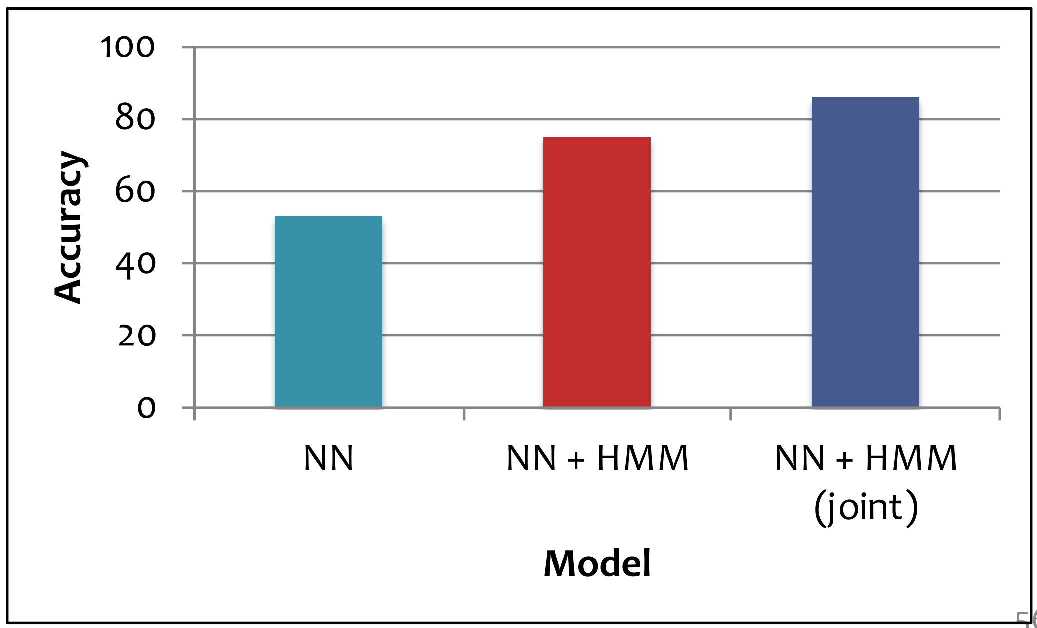
(Bengio et al., 1992)

Hybrid: NN + HMM

Experimental Setup:

- **Task:** Phoneme Recognition (aka. speaker independent recognition of plosive sounds)
- **Eight output labels:**
  - /p/, /t/, /k/, /b/, /d/, /g/, /dx/, /all other phonemes/
  - These are the HMM hidden states
- **Metric:** Accuracy
- **3 Models:**
  1. NN only
  2. NN + HMM (trained independently)
  3. NN + HMM (jointly trained)



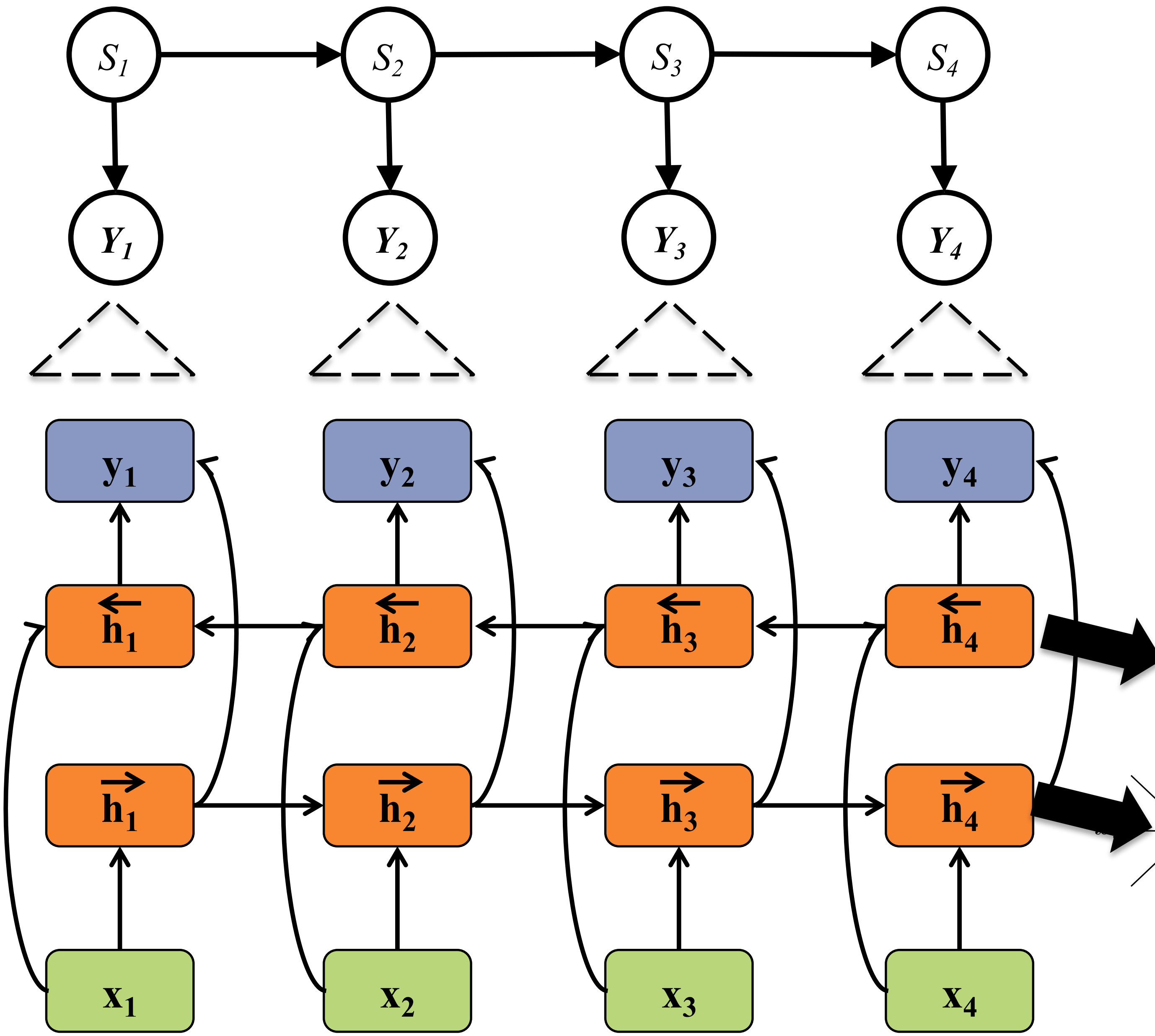


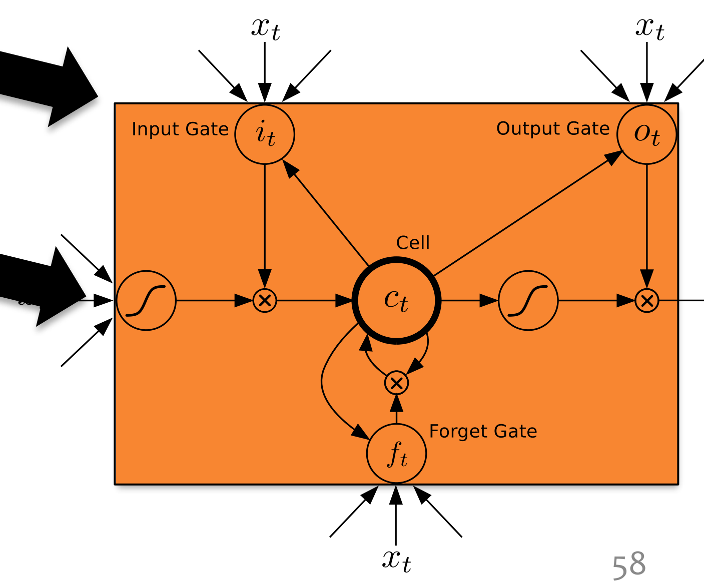
HYBRID:
RNN + HMM

Hybrid: RNN + HMM

(Graves et al., 2013)

- Graves et al. (2013) uses a Deep Bidirectional LSTM
- Each hidden unit is an LSTM
- Deep → More than two layers



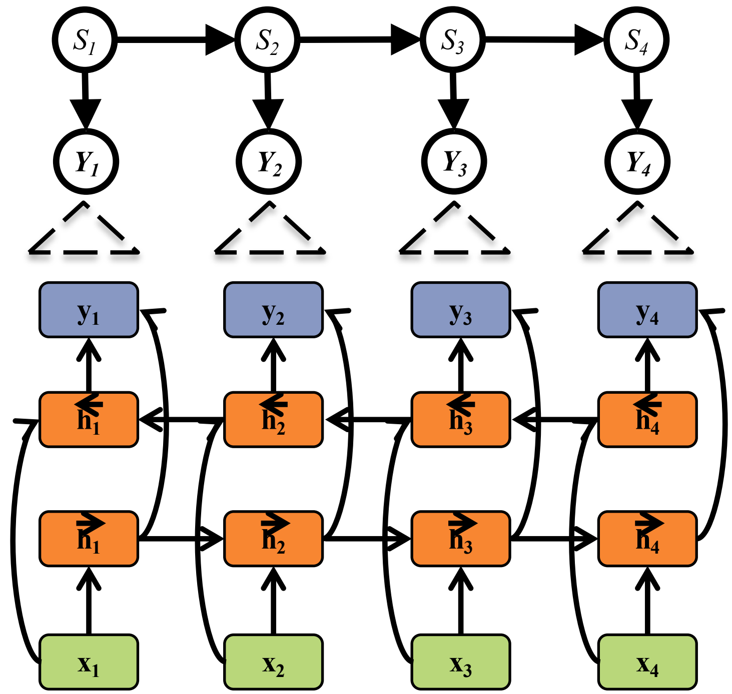


Hybrid: RNN + HMM

(Graves et al., 2013)

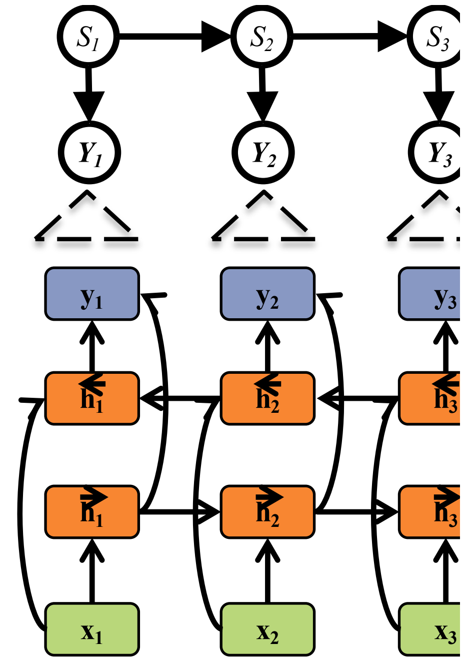
The model, inference, and learning can be **analogous** to our NN + HMM hybrid

- **Objective:** log-likelihood
- **Model:** HMM/Gaussian emissions
- **Inference:** forward-backward algorithm
- **Learning:** SGD with gradient by backpropagation



(Graves et al., 2013)

Hybrid: RNN + HMM



Experimental Setup:

- **Task:** Phoneme Recognition
- **Dataset:** TIMIT
- **Metric:** Phoneme Error Rate
- **Two classes of models:**
  1. Neural Net only
  2. NN + HMM hybrids

Training Method	Test PER
CTC	21.57 ± 0.25
CTC (noise)	18.63 ± 0.16
Transducer	$\mathbf{18.07} \pm \mathbf{0.24}$

1. Neural Net only

Network	Dev PER Test PER
DBRNN	19.91 ± 0.22 21.92 ± 0.35
DBLSTM	17.44 ± 0.156 19.34 ± 0.15
DBLSTM (noise)	16.11 ± 0.15 $\mathbf{17.99} \pm \mathbf{0.13}$

2. NN + HMM hybrids

HYBRID:
CNN + CRF

Recall...

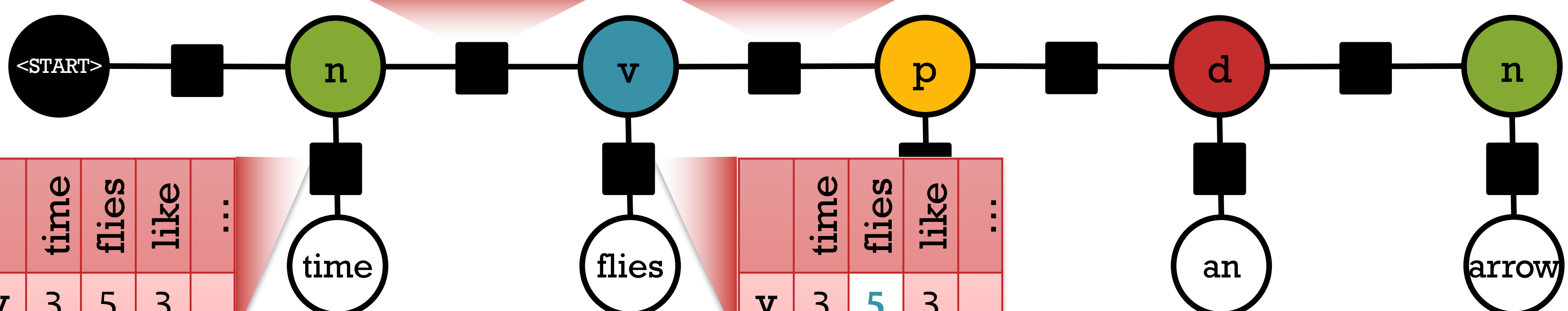
Markov Random Field (MRF)

Joint distribution over tags Y_i <u>and</u> words X_i

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) \quad = \quad \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0



	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

Recall...

Conditional Random Field (CRF)

Conditional distribution over tags Y_i <u>given</u> words x_i.
The factors and Z are now specific to the sentence $\boldsymbol{x}$.

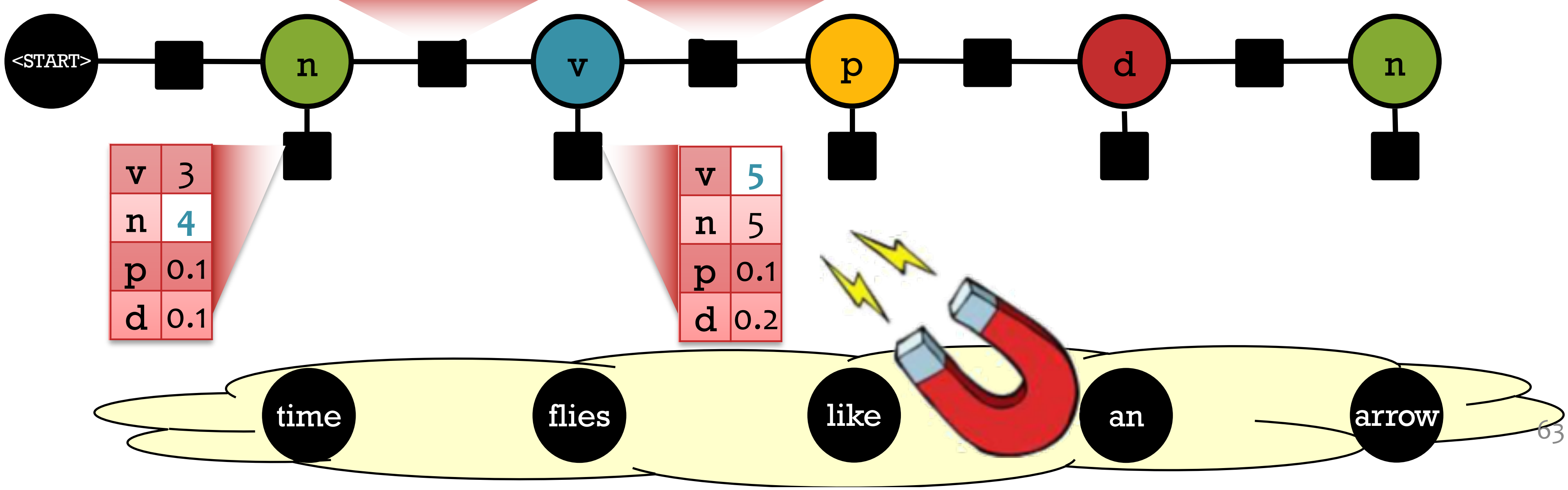
$$p(\text{n, v, p, d, n} \mid \text{time, flies, like, an, arrow}) \quad = \frac{1}{Z}\,(4 * 8 * 5 * 3 * \ldots)$$

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

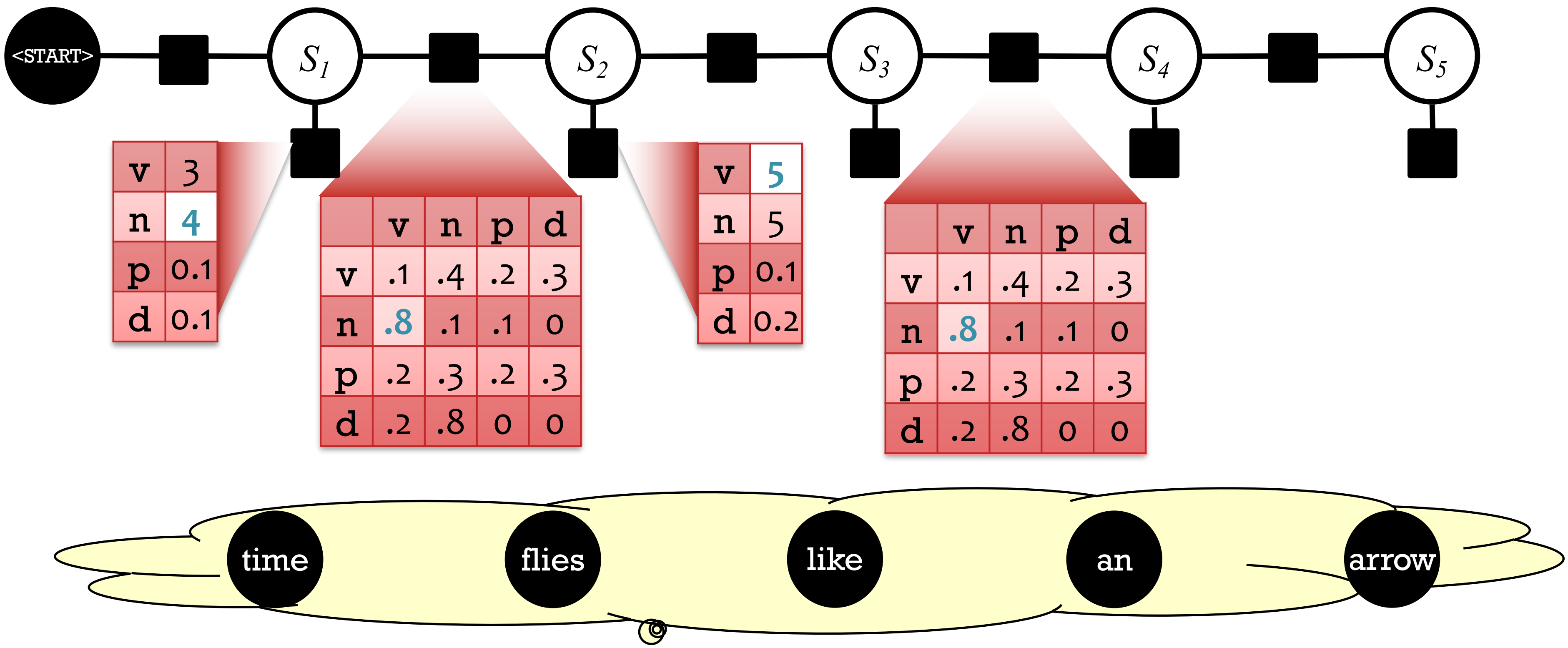
	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

v	3
n	4
p	0.1
d	0.1

v	5
n	5
p	0.1
d	0.2



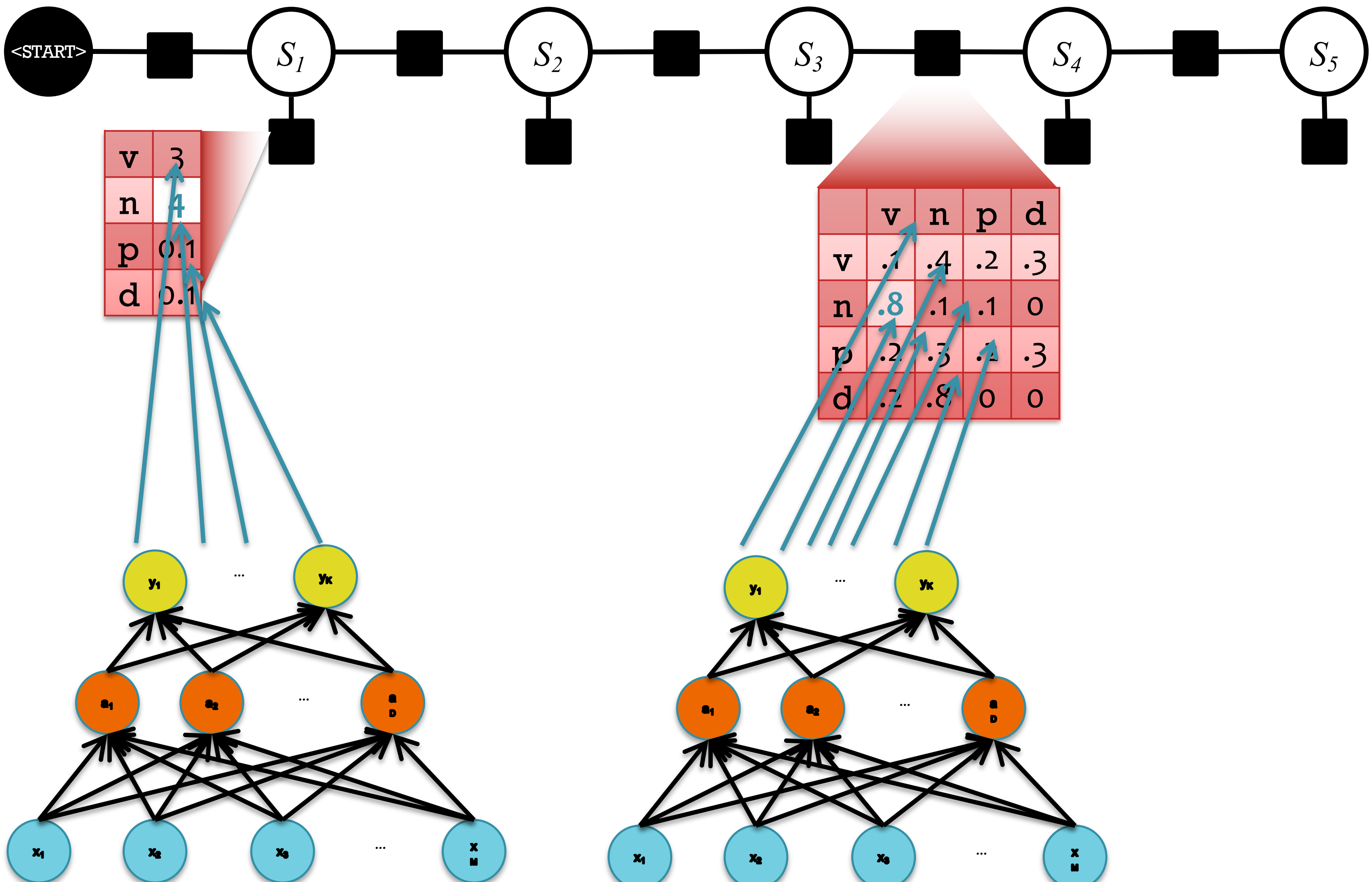
Hybrid: Neural Net + CRF



- In a standard CRF, each of the factor cells is a parameter (e.g. transition or emission)
- In the hybrid model, these values are computed by a neural network with its own parameters

Hybrid: Neural Net + CRF

Forward computation



(Collobert & Weston, 2011)

Hybrid: CNN + CRF

- For **computer vision,** Convolutional Neural Networks are in **2-dimensions**
- For **natural language,** the CNN is **1-dimensional**

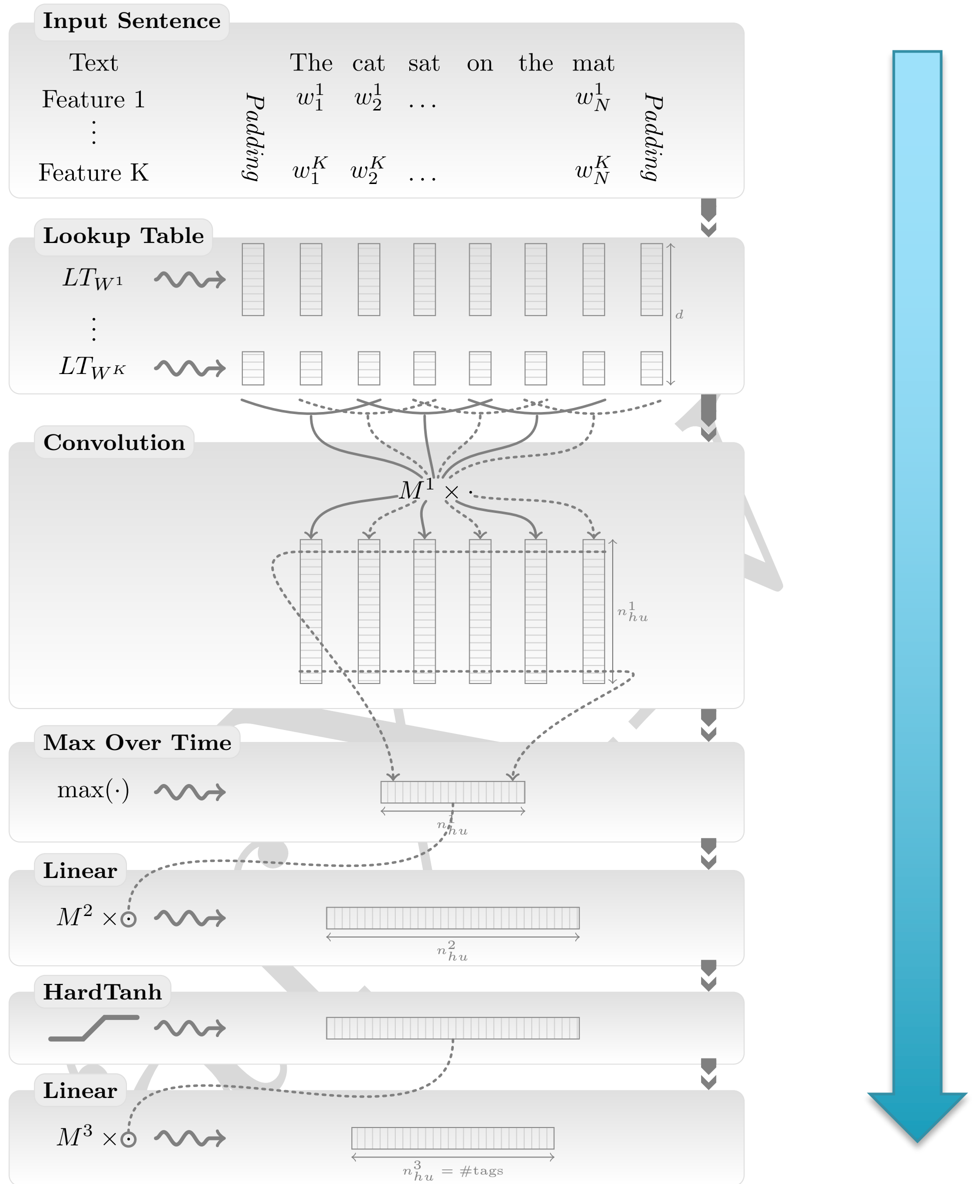


Figure from (Collobert & Weston, 2011)

Hybrid: CNN + CRF

"NN + SLL"

- Model: Convolutional Neural Network (CNN) with **linear-chain CRF**
- Training objective: maximize **sentence-level likelihood** (SLL)

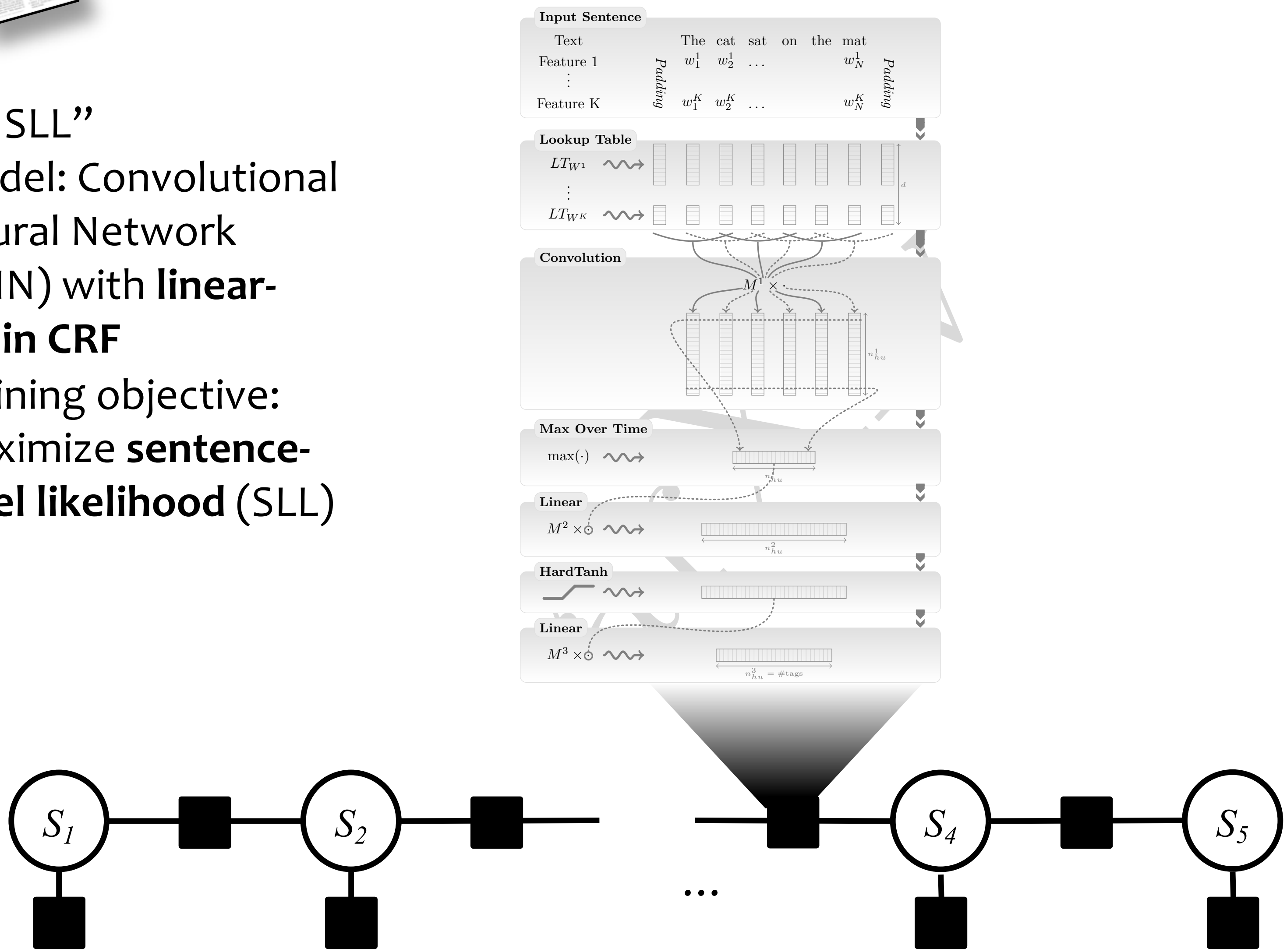
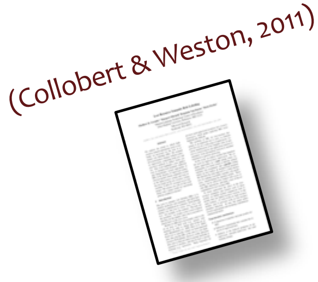


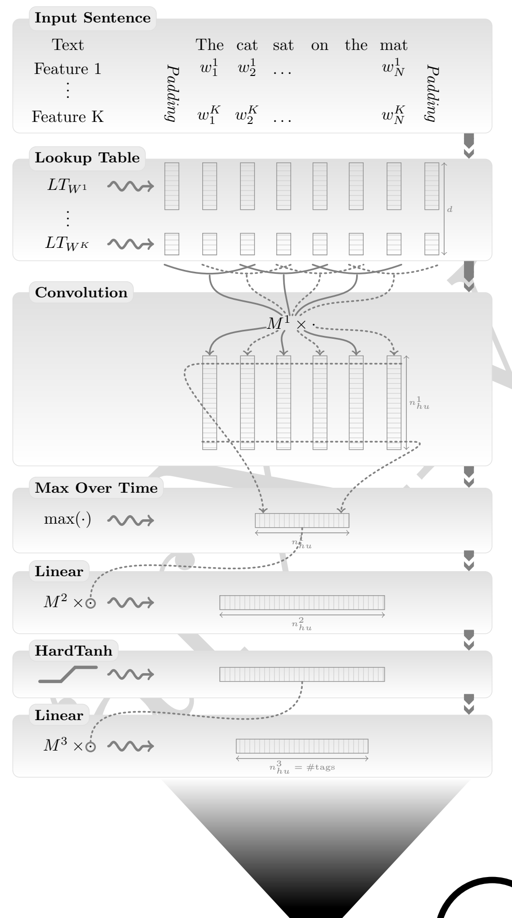
Figure from (Collobert & Weston, 2011)



Hybrid: CNN + CRF

"NN + WLL"

- Model: Convolutional Neural Network (CNN) with **logistic regression**
- Training objective: maximize **word-level likelihood** (WLL)



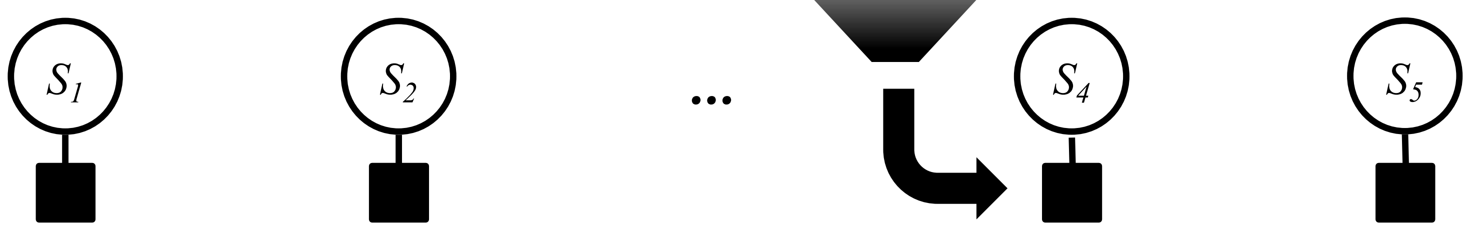


Figure from (Collobert & Weston, 2011)

(Collobert & Weston, 2011)

Hybrid: CNN + CRF

Experimental Setup:

- **Tasks:**
  - Part-of-speech tagging (POS),
  - Noun-phrase and Verb-phrase Chunking,
  - Named-entity recognition (NER)
  - Semantic Role Labeling (SRL)
- **Datasets / Metrics:** Standard setups from NLP literature (higher PWA/F1 is better)
- **Models:**
  - Benchmark systems are typical – non-neural network systems
  - NN+WLL: hybrid CNN with logistic regression
  - NN+SLL: hybrid CNN with linear-chain CRF

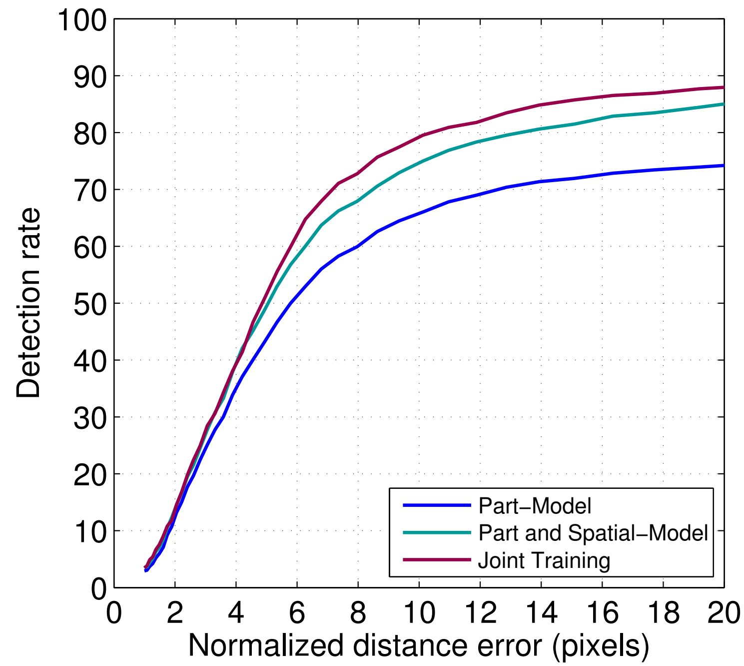
Approach	POS (PWA)	Chunking (F1)	NER (F1)	SRL (F1)
Benchmark Systems	97.24	94.29	89.31	77.92
NN+WLL	96.31	89.13	79.53	55.40
NN+SLL	96.37	90.33	81.47	70.99

(Thompson et al., 2014)

Hybrid: CNN + MRF

Experimental Setup:

- **Task:** pose estimation
- **Model:** Deep CNN + MRF



TRICKS OF THE TRADE

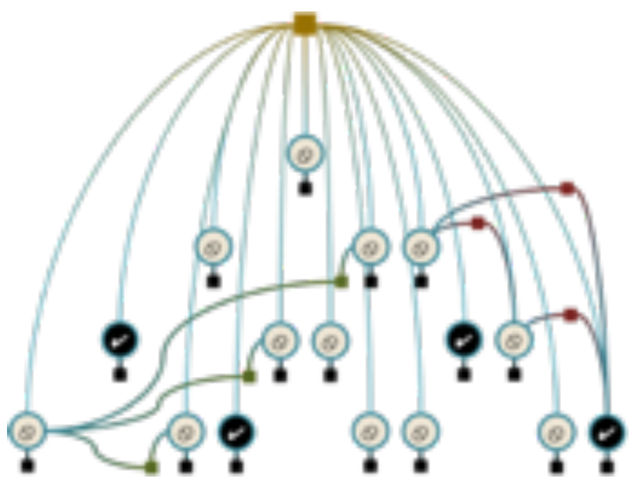
Tricks of the Trade

- **Lots of them:**
  - Pre-training helps (but isn't always necessary)
  - Train with adaptive gradient variants of SGD (e.g. Adam)
  - Use max-margin loss function (i.e. hinge loss) – though only sub-differentiable it often gives better results
  - …
- A few years back, they were considered "**poorly documented**" and "requiring great expertise"
- Now there are lots of **good tutorials** that describe (very important) specific implementation details
- Many of them **also apply to training graphical models**!

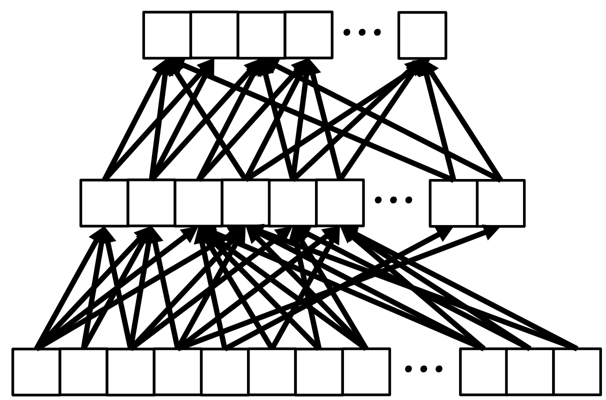
SUMMARY

Summary:
Hybrid Models

Graphical models let you encode domain knowledge



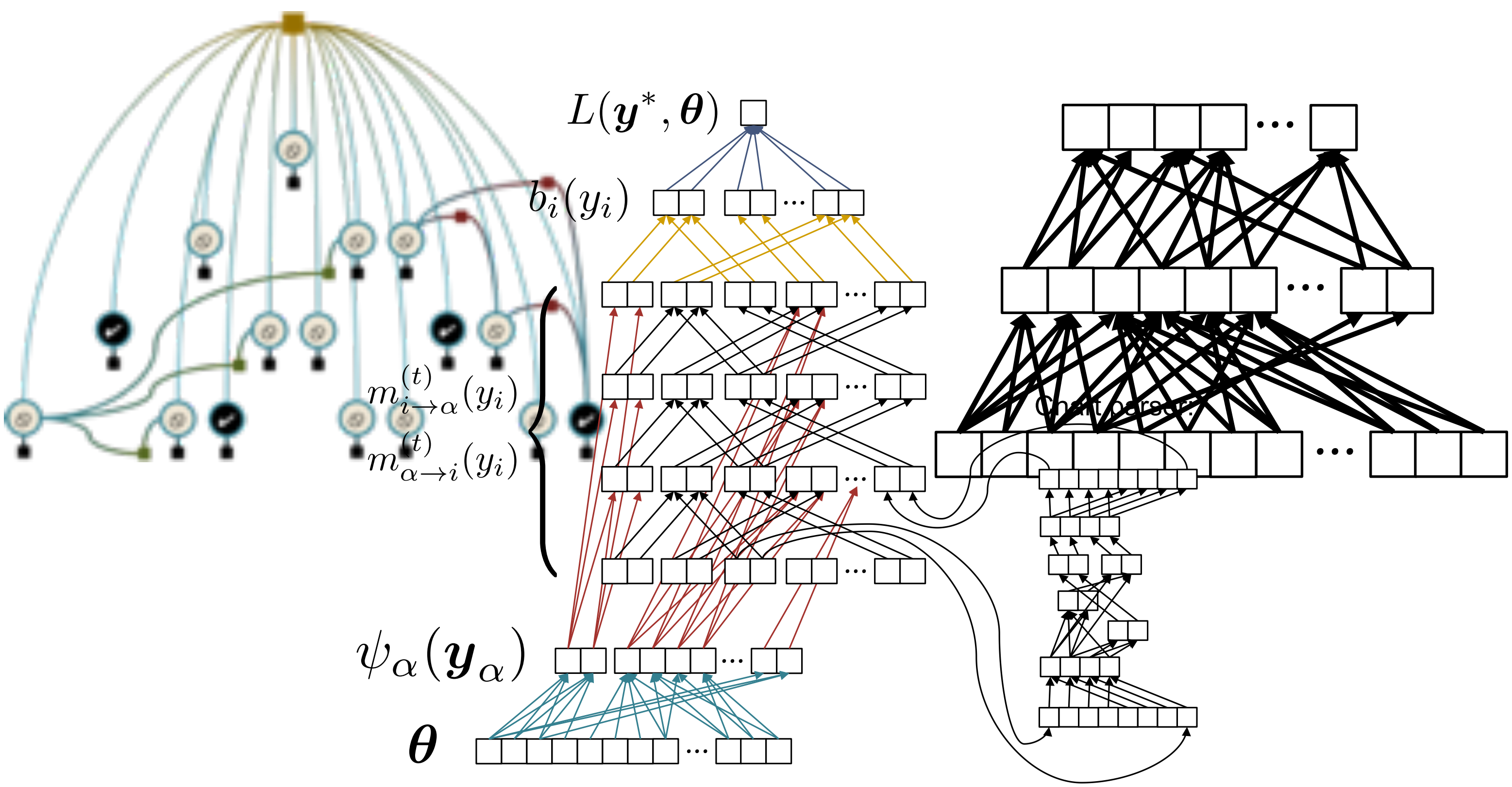
Neural nets are really good at fitting the data discriminatively to make good predictions



Could we define a neural net that incorporates domain knowledge?

Summary:
Hybrid Models

Key idea: Use a NN to learn features for a GM, then train the entire model by backprop



MBR DECODING

Minimum Bayes Risk Decoding

- Suppose we given a loss function $l(\mathbf{y}', \mathbf{y})$ and are asked for a single tagging
- How should we choose just one from our probability distribution $p(\mathbf{y}|\mathbf{x})$?
- A minimum Bayes risk (MBR) decoder $h(\mathbf{x})$ returns the variable assignment with minimum **expected** loss under the model's distribution

$$
\begin{aligned}
h_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \underset{\hat{\boldsymbol{y}}}{\operatorname{argmin}} \; \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})] \\
&= \underset{\hat{\boldsymbol{y}}}{\operatorname{argmin}} \; \sum_{\boldsymbol{y}} p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \ell(\hat{\boldsymbol{y}}, \boldsymbol{y})
\end{aligned}
$$

Minimum Bayes Risk Decoding

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \underset{\hat{\boldsymbol{y}}}{\text{argmin}} \ \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})]$$

Consider some example loss functions:

The **Hamming loss** corresponds to accuracy and returns the number of incorrect variable assignments:

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

The MBR decoder is:

$$\hat{y}_i = h_{\boldsymbol{\theta}}(\boldsymbol{x})_i = \underset{\hat{y}_i}{\text{argmax}} \ p_{\boldsymbol{\theta}}(\hat{y}_i \mid \boldsymbol{x})$$

This decomposes across variables and requires the variable marginals.

Minimum Bayes Risk Decoding

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \underset{\hat{\boldsymbol{y}}}{\text{argmin}} \; \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})]$$

Consider some example loss functions:

The ***0-1* loss function** returns *1* only if the two assignments are identical and *0* otherwise:

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

The MBR decoder is:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \underset{\hat{\boldsymbol{y}}}{\text{argmin}} \; \sum_{\boldsymbol{y}} p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})(1 - \mathbb{I}(\hat{\boldsymbol{y}}, \boldsymbol{y}))$$

$$= \underset{\hat{\boldsymbol{y}}}{\text{argmax}} \; p_{\boldsymbol{\theta}}(\hat{\boldsymbol{y}} \mid \boldsymbol{x})$$

which is exactly the MAP inference problem!

Minimum Bayes Risk Decoding

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \underset{\hat{\boldsymbol{y}}}{\text{argmin}} \ \mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})}[\ell(\hat{\boldsymbol{y}}, \boldsymbol{y})]$$

Consider some example loss functions:

The ***0-1* loss function** returns *1* only if the two assignments are identical and *0* otherwise:

$$h_\theta(x) = \underset{\hat{y}}{\text{argmin}} \left[\sum_y p(y|x) \left(1 - \mathbb{1}(\hat{y} = y)\right) \right]$$

$$= \underset{\hat{y}}{\text{argmin}} \underbrace{\left[\sum_y p(y|x) \right]}_{\text{constant wrt } \hat{y}} - \underbrace{\left[\sum_y p(y|x) \mathbb{1}(\hat{y} = y) \right]}_{= p(\hat{y}|x)}$$

$$= \underset{\hat{y}}{\text{argmin}} - p(\hat{y}|x)$$

$$= \underset{\hat{y}}{\text{argmax}} \ p(\hat{y}|x)$$

MBR Decoders

Q: If loss(y, y*) decomposes in the same way as p(y|x), can we efficiently compute the MBR decoder h(x) for that loss/model pair?

A: Yes.

How to do so is left as an exercise…

LINEAR PROGRAMMING & INTEGER LINEAR PROGRAMMING

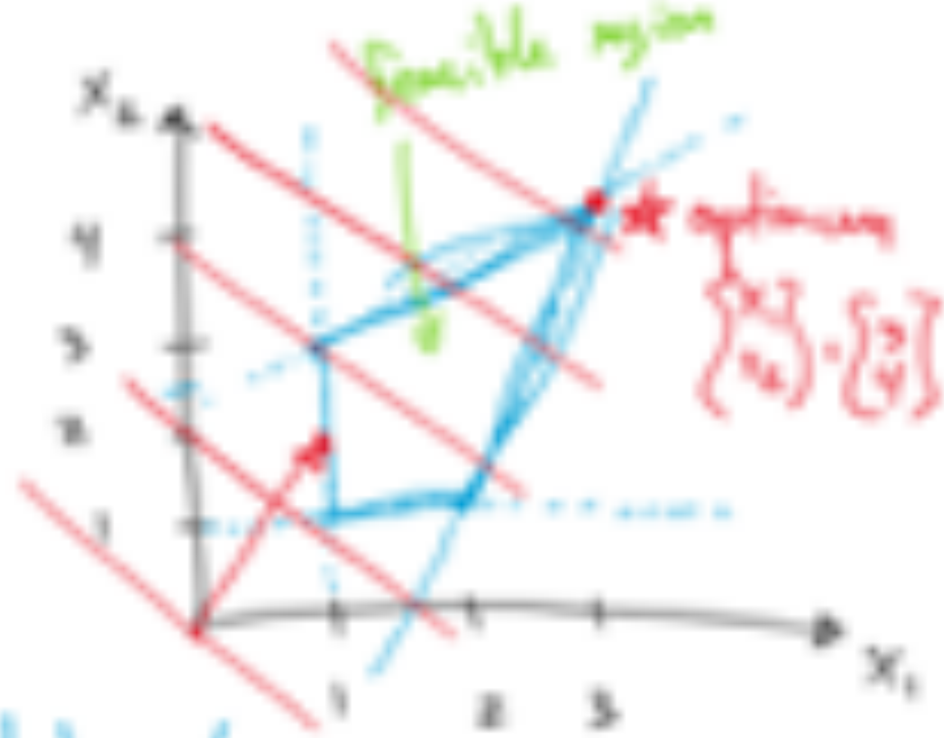
Example of Linear Program in 2D

Ex: 2D

$\max \; x_1 + 2x_2$ $\leftarrow \begin{bmatrix}1\\2\end{bmatrix}^T \begin{bmatrix}x_1\\x_2\end{bmatrix}$] objective

s.t.] constraints

$$x_1 \geq 1$$
$$x_2 \geq 1$$
$$x_2 - 3x_1 \geq -5$$
$$x_2 - \tfrac{1}{2}x_1 \leq \tfrac{5}{2}$$



Def: feasible region is a polyhedron (and possibly unbounded)

Def: problem is infeasible if feasible region is empty
$\Rightarrow$ no solution

LP Standard Form

LP Standard Form

$$\max \ c^T x \qquad \text{[max obj.]}$$

$$\text{s.t.} \ Ax \leq b \qquad \text{[inequalities in } \leq \text{ form]}$$

$$0 \leq x_i \qquad \text{[nonnegative variables]}$$

Notes:

- c, x, b are vectors
- A is a matrix
- x are variables
- c, b, A are constants

Conversion to Standard Form

Every LP can be written in standard form

1. min → max by negating obj

   $\min c^T x \rightarrow \max -c^T x$

2. geq → leq by negating constraint.

   $a_1 x_1 + a_2 x_2 \geq b \rightarrow -a_1 x_1 - a_2 x_2 \leq -b$

3. eq → geq + leq

   $a_1 x_1 + a_2 x_2 = b \rightarrow a_1 x_1 + a_2 x_2 \leq b$

   $a_1 x_2 + a_2 x_2 \geq b$

4. var w/u.b.

   $x_i \leq u \rightarrow$ new variable $w_i = u - x_i$

   with $0 \leq w_i$

5. var w/interval

   Exercise

6. free variable

   new variables x' and x'' s.t. $x = x' - x''$

   $0 \leq x'$ and $0 \leq x''$

Linear Programming

Whiteboard

- In pictures…
  - Simplex algorithm (tableau method)
  - Interior points algorithm(s)

Integer Linear Programming

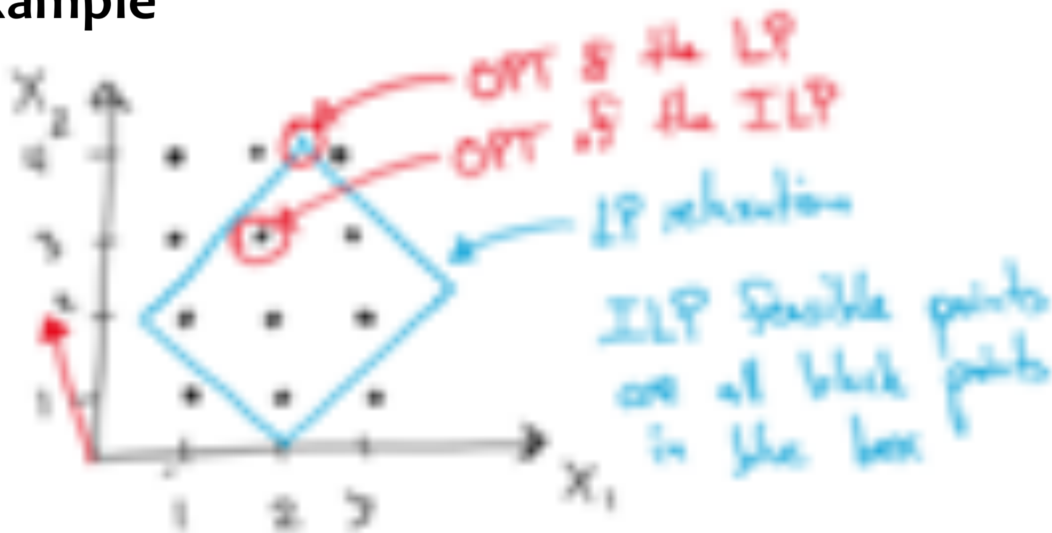
ILP in Standard Form

$$
\begin{aligned}
\max \quad & c^T x \\
\text{s.t.} \quad & Ax \le b \\
& 0 \le x_i \quad \forall i \\
& x_i \in \mathbb{Z} \quad \forall i
\end{aligned}
$$

ILP

LP relaxation of the ILP = everything except integer constraints

Example



Mixed-Integer Linear Programming

MILP in Standard Form

$$\begin{aligned} \max\ & c^T x \\ \text{s.t. } & Ax \leq b \\ & 0 \leq x_i \quad \forall i \\ & x_i \in \mathbb{Z} \quad \forall i \in S \subset \{1, \ldots, N\} \end{aligned}$$

only some vars are integers

Example

if $x_2 \in \mathbb{Z}$ but $x_1 \in \mathbb{R}$