10-418/10-618 Machine Learning for Structured Data

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Neural Potentials + MBR Decoding

Matt Gormley
Lecture 10
Oct. 3, 2022

Reminders

- Homework 2: Learning to Search for RNNs
  - Out: Sun, Sep 18
  - Written (except for Empirical Questions)
    - Due: Thu, Sep 29 at 11:59pm
  - Programming + Empirical Questions
    - Due: Mon, Oct 24 at 9:00am
- Homework 3: General Graph CRF Module
  - Out: Thu, Sep 29
  - Due: Mon, Oct 10 at 11:59pm

Reminders

- Homework 2: Learning to Search for RNNs
  - Out: Sun, Sep 18
  - Written (except for Empirical Questions)
    - Due: Thu, Sep 29 at 11:59pm
  - Programming + Empirical Questions
    - Due: Mon, Oct 24 at 9:00am
  - New autograder disaster…
- Homework 3: General Graph CRF Module
  - Out: Thu, Sep 29
  - Due: Mon, Oct 10 at 11:59pm

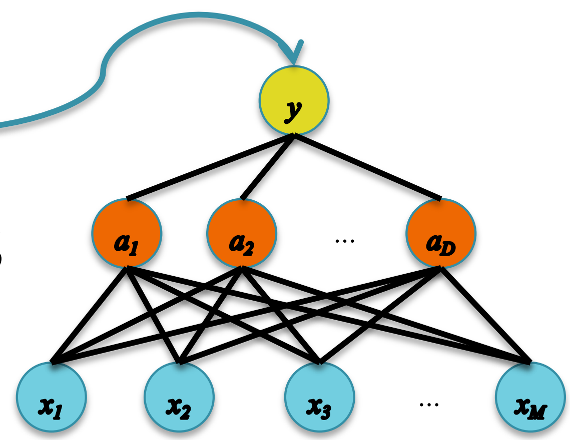
MRF AND CRF LEARNING (LOG-LINEAR PARAMETERIZATION)

Options for MLE of MRFs

- **Setting I:** $\psi_C(\boldsymbol{x}_C) = \theta_{C,\boldsymbol{x}_C}$
  A. MLE by inspection (Decomposable Models)
  B. Iterative Proportional Fitting (IPF)

- **Setting II:** $\psi_C(\boldsymbol{x}_C) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}_C))$
  C. Generalized Iterative Scaling
  D. Gradient-based Methods

- **Setting III:** $\psi_C(\boldsymbol{x}_C) =$
  E. Gradient-based Methods



MRF and CRF Learning

Whiteboard

- log-linear MRF model (i.e. with feature based potentials)
- log-linear MRF derivatives
- log-linear MRF training with SGD
- log-linear CRF model (i.e. with feature based potentials)
- log-linear CRF derivatives
- log-linear CRF training with SGD

Recipe for Gradient-based Learning

1. Write down the objective function
2. Compute the partial derivatives of the objective (i.e. gradient, and maybe Hessian)
3. Feed objective function and derivatives into black box

4. Retrieve optimal parameters from black box

Optimization Algorithms

What is the black box?

- Newton’s method
- Hessian-free / Quasi-Newton methods
  - Conjugate gradient
  - L-BFGS
- Stochastic gradient methods
  - Stochastic gradient descent (SGD)
  - SGD with momentum
  - Adam

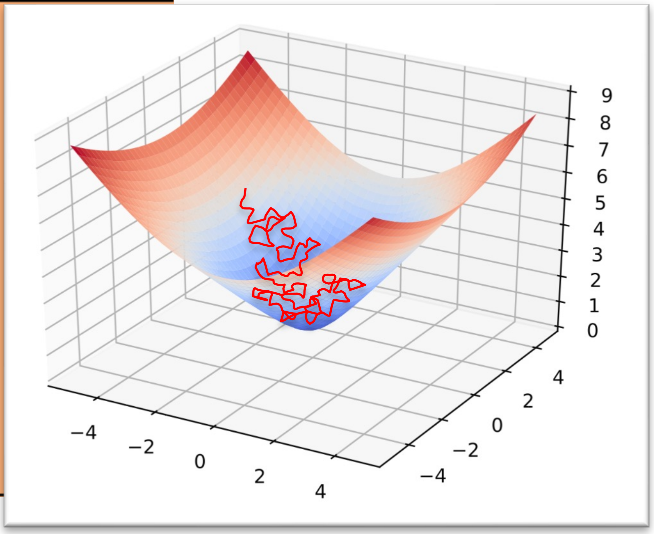
Stochastic Gradient Descent (SGD)

Assume we have an objective that decomposes additively:

$$\text{Let } J(\boldsymbol{\theta}) = \sum_{i=1}^{N} J^{(i)}(\boldsymbol{\theta})$$

Algorithm 2 Stochastic Gradient Descent (SGD)

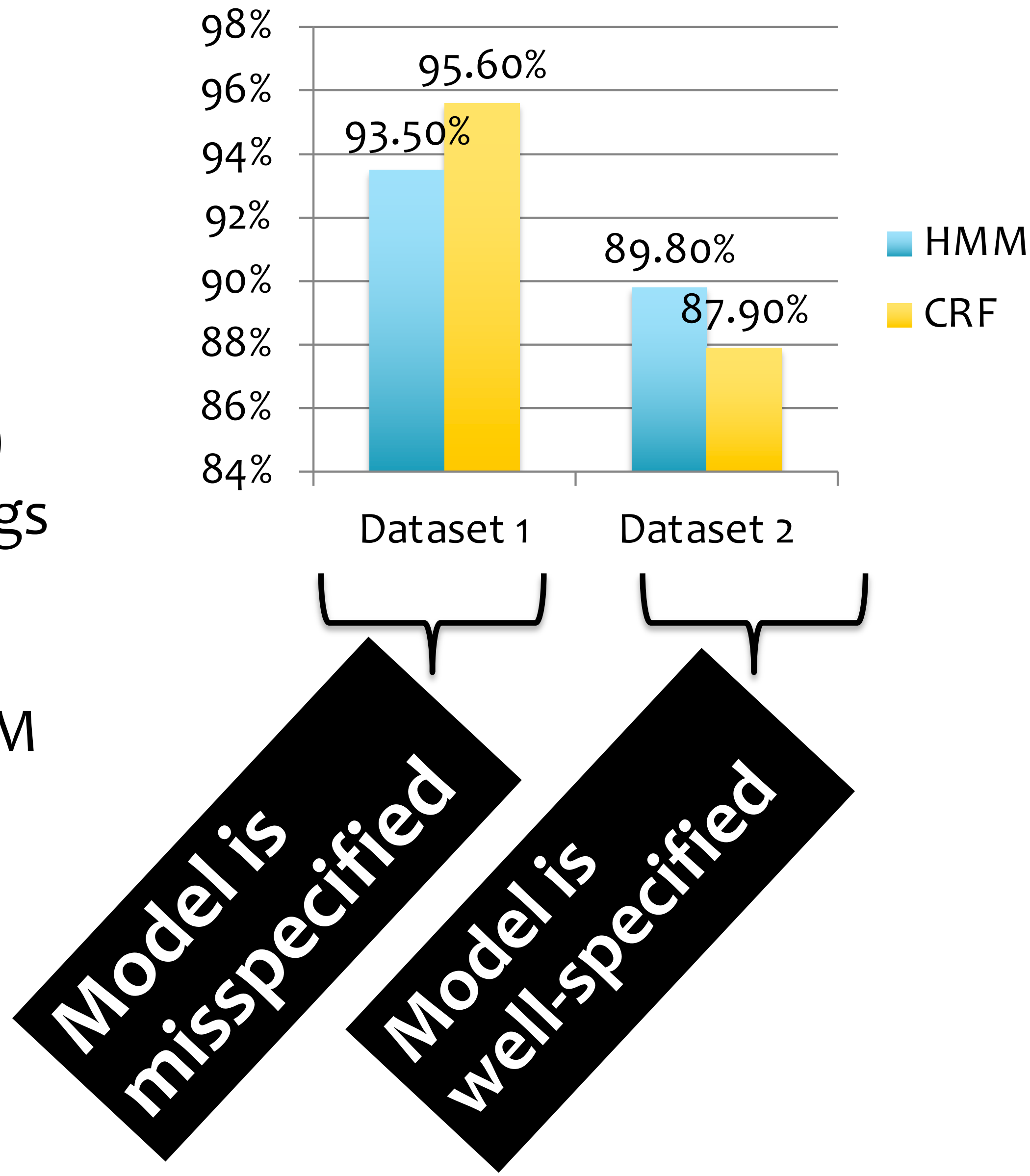
procedure SGD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
    while not converged **do**
        $i \sim$ Uniform($\{1, 2, \ldots, N\}$)
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$
    return $\boldsymbol{\theta}$



Generative vs. Discriminative

Liang & Jordan (ICML 2008) compares **HMM** and **CRF** with **identical features**

- Dataset 1: (Real)
  - WSJ Penn Treebank (38K train, 5.5K test)
  - 45 part-of-speech tags
- Dataset 2: (Artificial)
  - Synthetic data generated from HMM learned on Dataset 1 (1K train, 1K test)
- Evaluation Metric: Accuracy



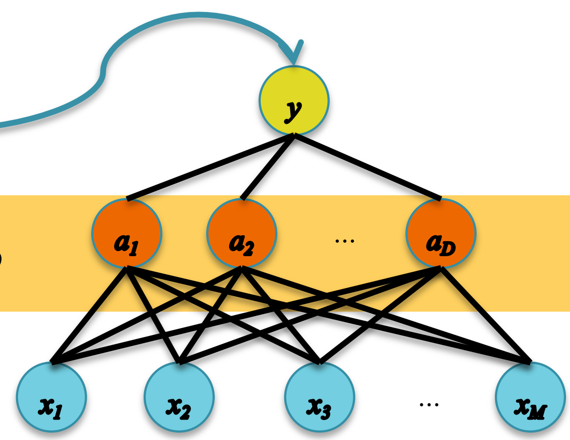
NEURAL PARAMETERIZATION OF CONDITIONAL RANDOM FIELD

Options for MLE of MRFs

- **Setting I:** $\psi_C(\boldsymbol{x}_C) = \theta_{C,\boldsymbol{x}_C}$
  A. MLE by inspection (Decomposable Models)
  B. Iterative Proportional Fitting (IPF)

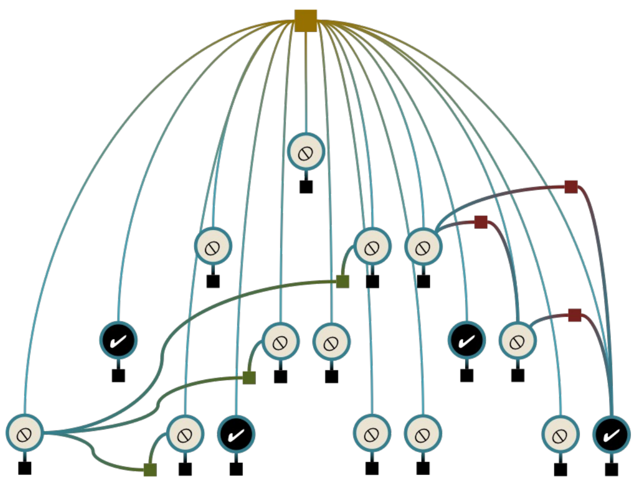
- **Setting II:** $\psi_C(\boldsymbol{x}_C) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}_C))$
  C. Generalized Iterative Scaling
  D. Gradient-based Methods

- **Setting III:** $\psi_C(\boldsymbol{x}_C) =$
  E. Gradient-based Methods

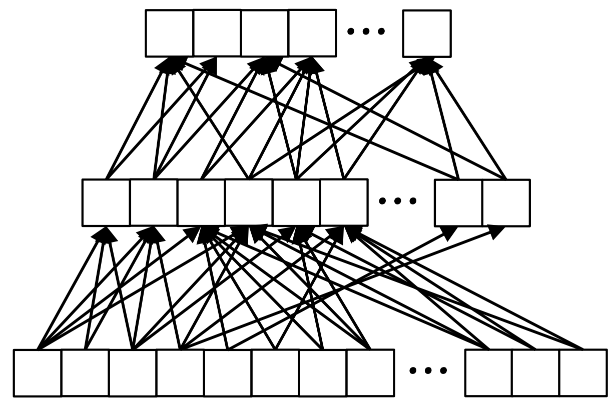


Motivation:
Hybrid Models

Graphical models let you encode domain knowledge



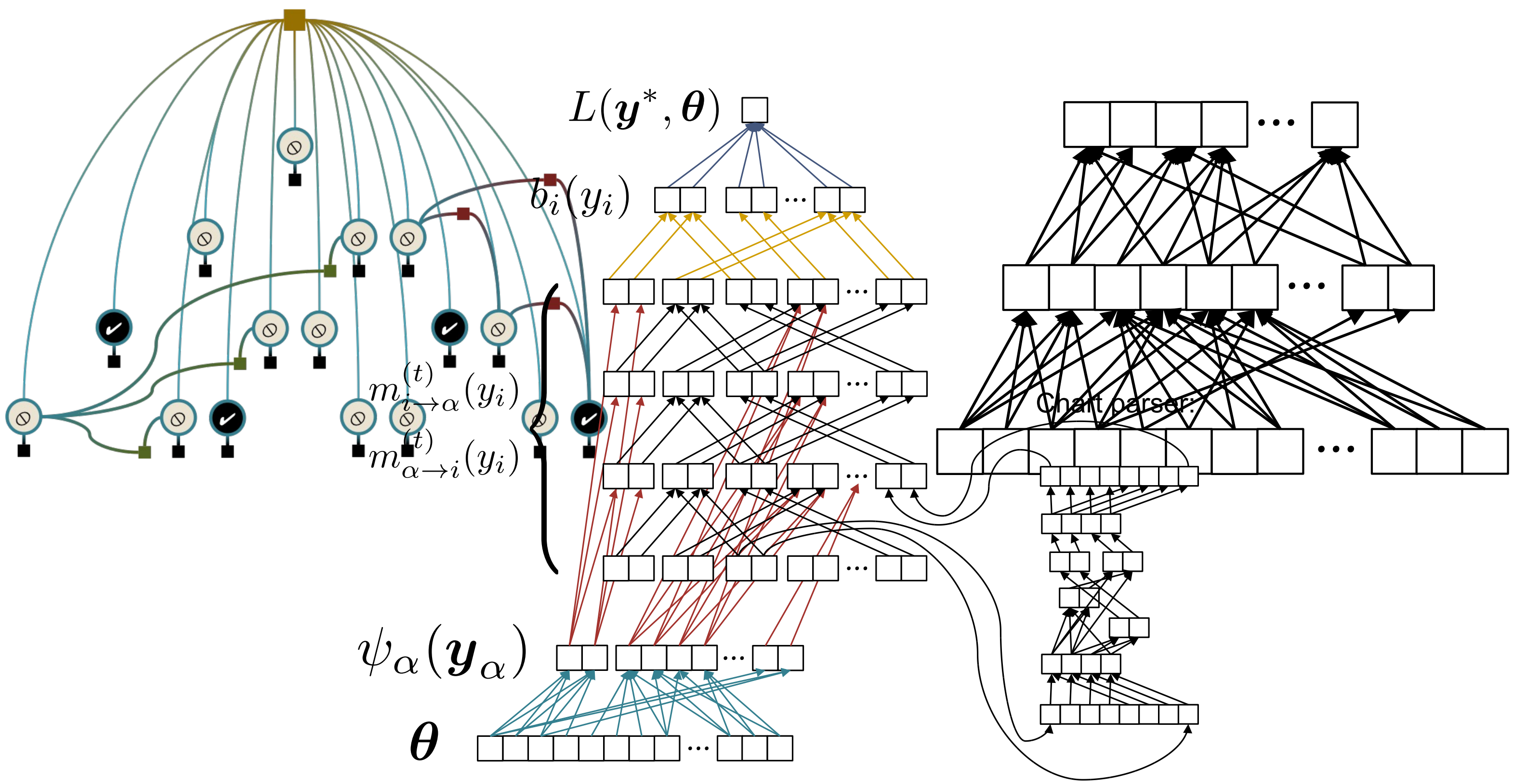
Neural nets are really good at fitting the data discriminatively to make good predictions



Could we define a neural net that incorporates domain knowledge?

Motivation:
Hybrid Models

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



A Recipe for Neural Networks

1. Given training data:

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

Face *Face* *Not a face*

2. Choose each of these:
   - Decision function

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

Examples: Linear regression, Logistic regression, Neural Network

   - Loss function

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

Examples: Mean-squared error, Cross Entropy

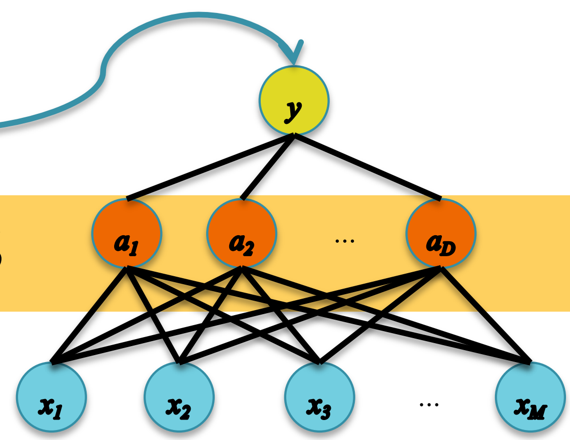
MRF AND CRF LEARNING (NEURAL PARAMETERIZATION)

Options for MLE of MRFs

- **Setting I:** $\psi_C(\boldsymbol{x}_C) = \theta_{C,\boldsymbol{x}_C}$
  A. MLE by inspection (Decomposable Models)
  B. Iterative Proportional Fitting (IPF)

- **Setting II:** $\psi_C(\boldsymbol{x}_C) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}_C))$
  C. Generalized Iterative Scaling
  D. Gradient-based Methods

- **Setting III:** $\psi_C(\boldsymbol{x}_C) =$
  E. Gradient-based Methods



Whiteboard:

- CRF w/LSTM potentials
- Gradient of MRF/CRF log-likelihood with respect to log potentials
- Gradient of MRF/CRF log-likelihood with respect to potentials
- Backprop with MRF/CRF log-likelihood as a loss function

Factor Derivatives

Log-probability:

$$\log p(\mathbf{y}) = \left[\sum_{\alpha} \log \psi_{\alpha}(\mathbf{y}_{\alpha})\right] - \log \sum_{\mathbf{y}' \in \mathcal{Y}} \prod_{\alpha} \psi_{\alpha}(\mathbf{y}'_{\alpha}) \tag{1}$$

Derivatives:

$$\frac{\partial \log p(\mathbf{y})}{\partial \log \psi_{\alpha}(\mathbf{y}'_{\alpha})} = \mathbb{1}(\mathbf{y}_{\alpha} = \mathbf{y}'_{\alpha}) - p(\mathbf{y}'_{\alpha}) \tag{2}$$

$$\frac{\partial \log p(\mathbf{y})}{\partial \psi_{\alpha}(\mathbf{y}'_{\alpha})} = \frac{\mathbb{1}(\mathbf{y}_{\alpha} = \mathbf{y}'_{\alpha}) - p(\mathbf{y}'_{\alpha})}{\psi_{\alpha}(\mathbf{y}'_{\alpha})} \tag{3}$$

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…

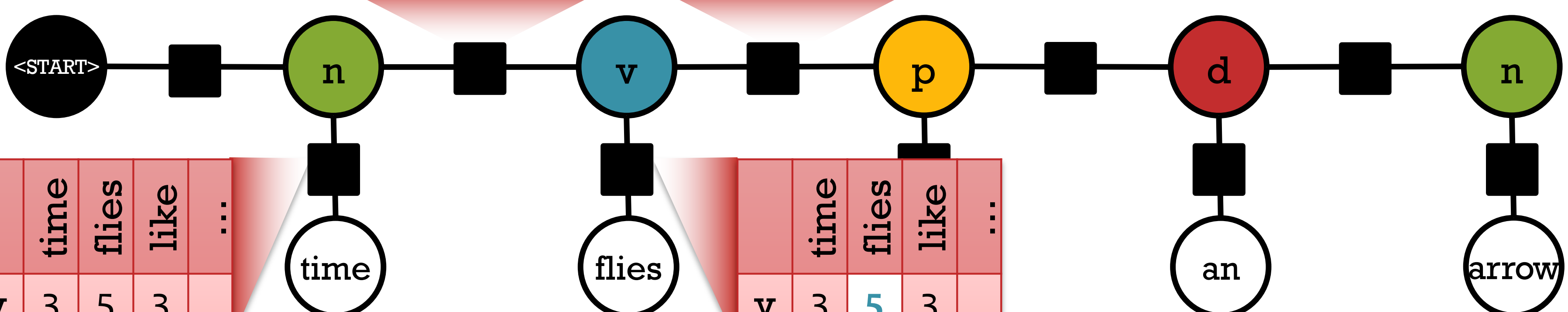
Markov Random Field (MRF)

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…

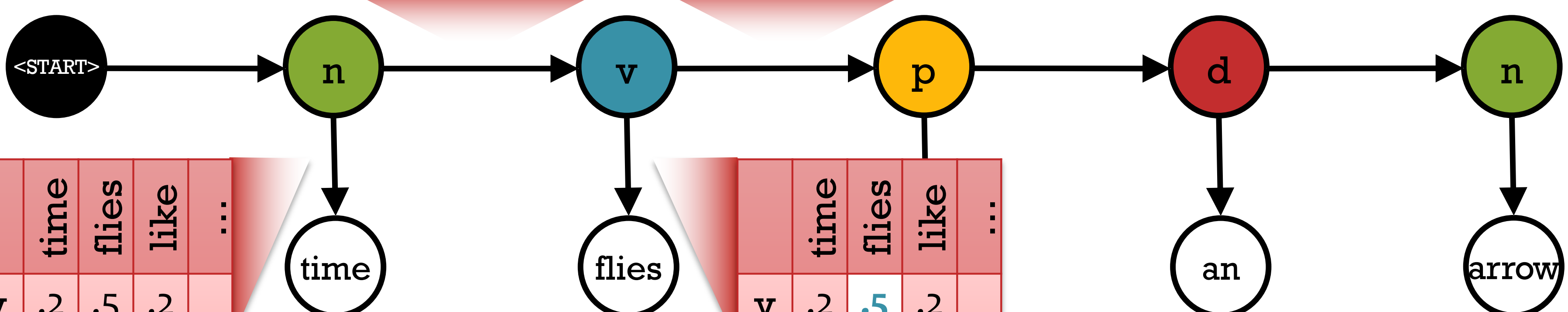
Hidden Markov Model

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)

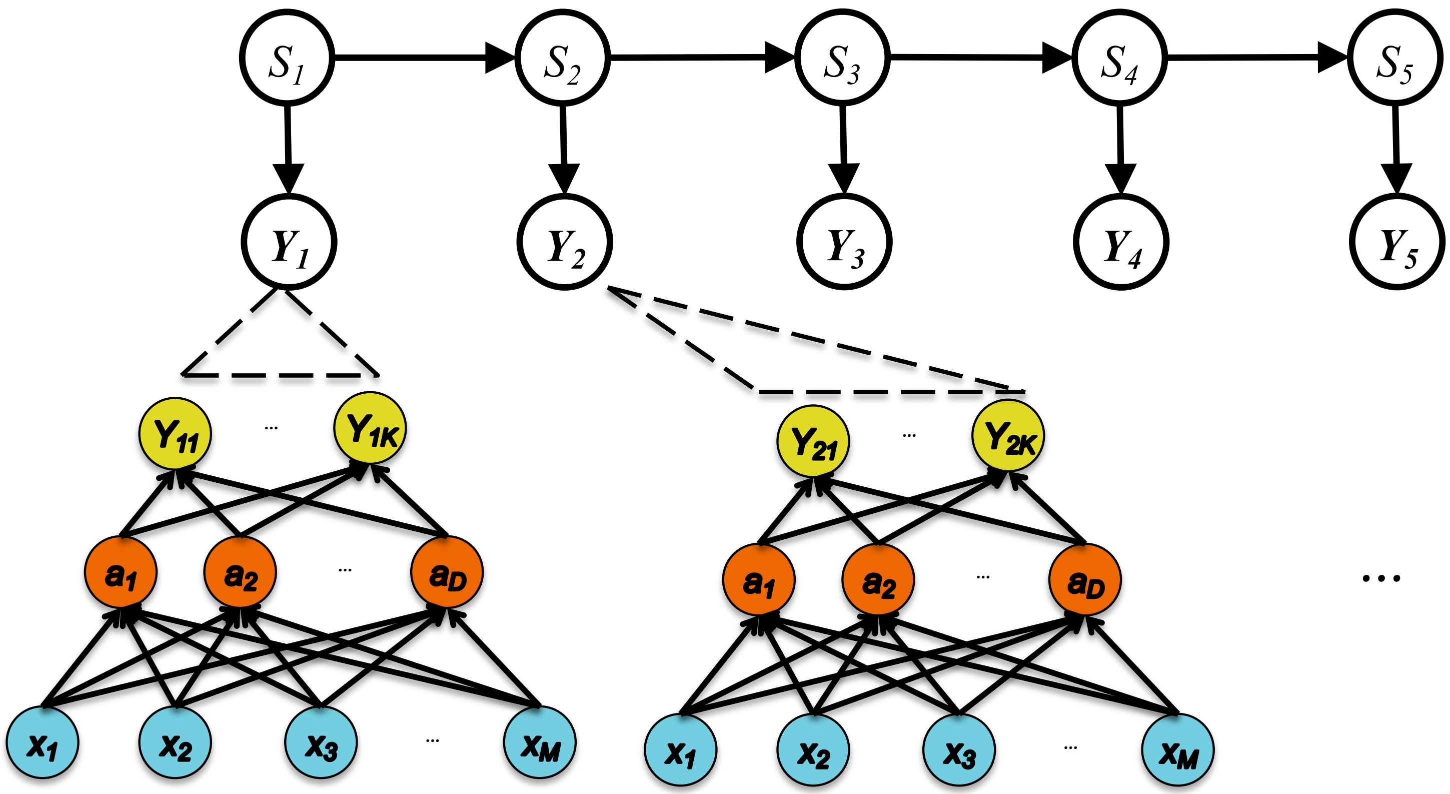
Discrete HMM state: $S_t \in \{/p/, /t/, /k/, /b/, /d/, \ldots, /g/\}$

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

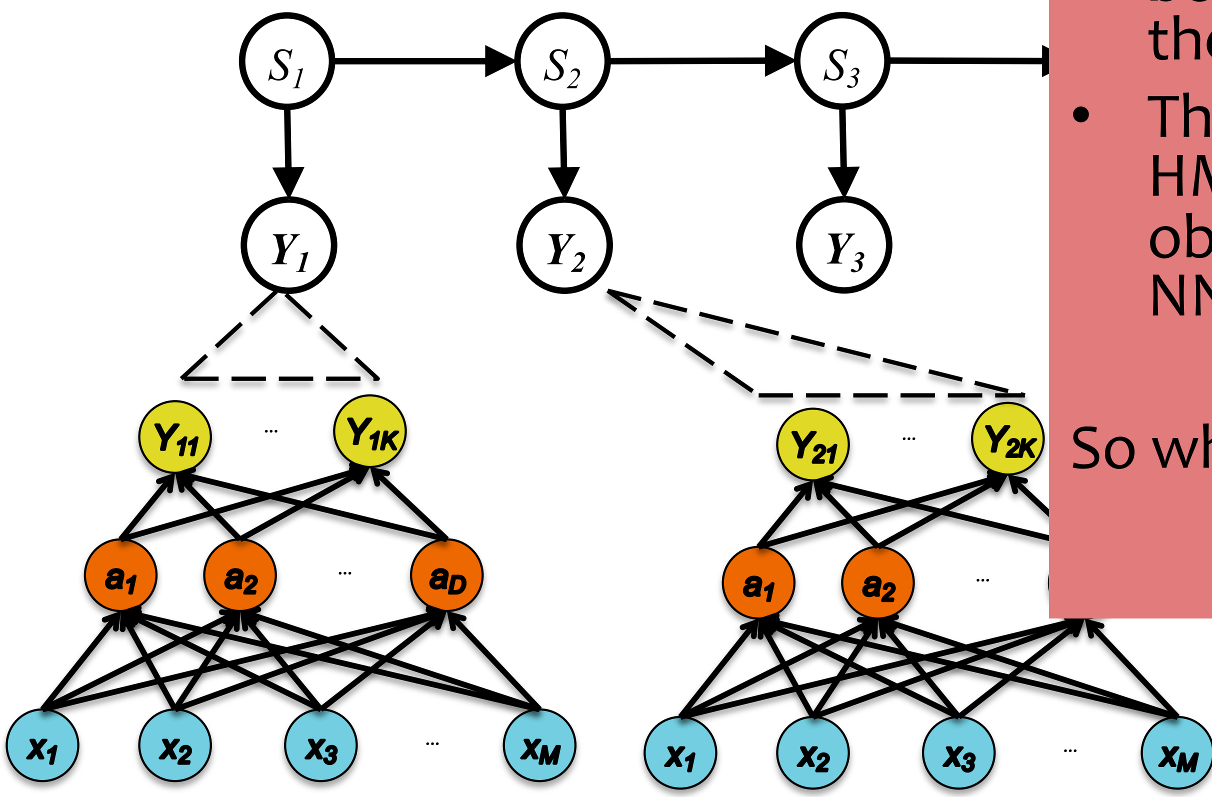
(Bengio et al., 1992)

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

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

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

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

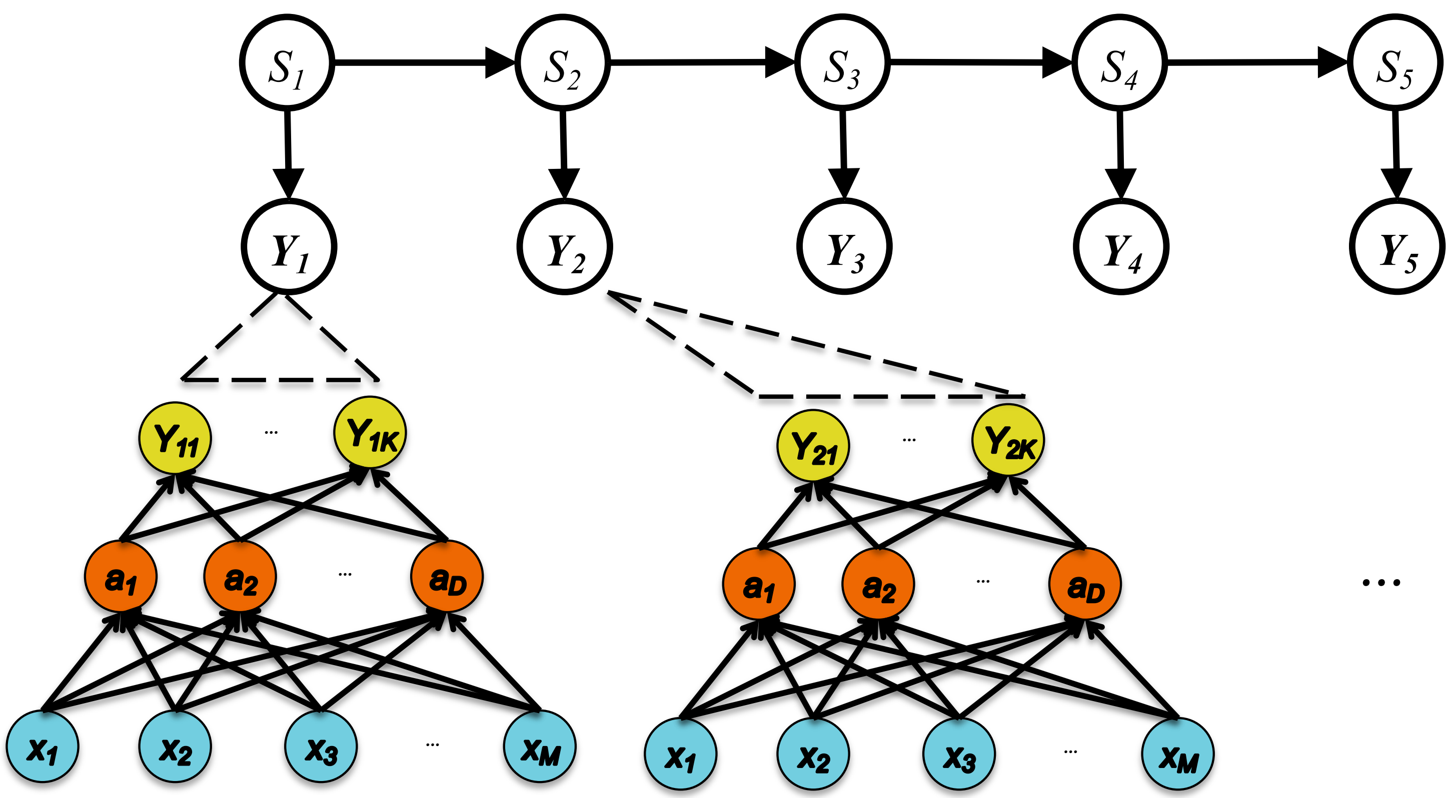


Lots of oddities to this picture:

- **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.

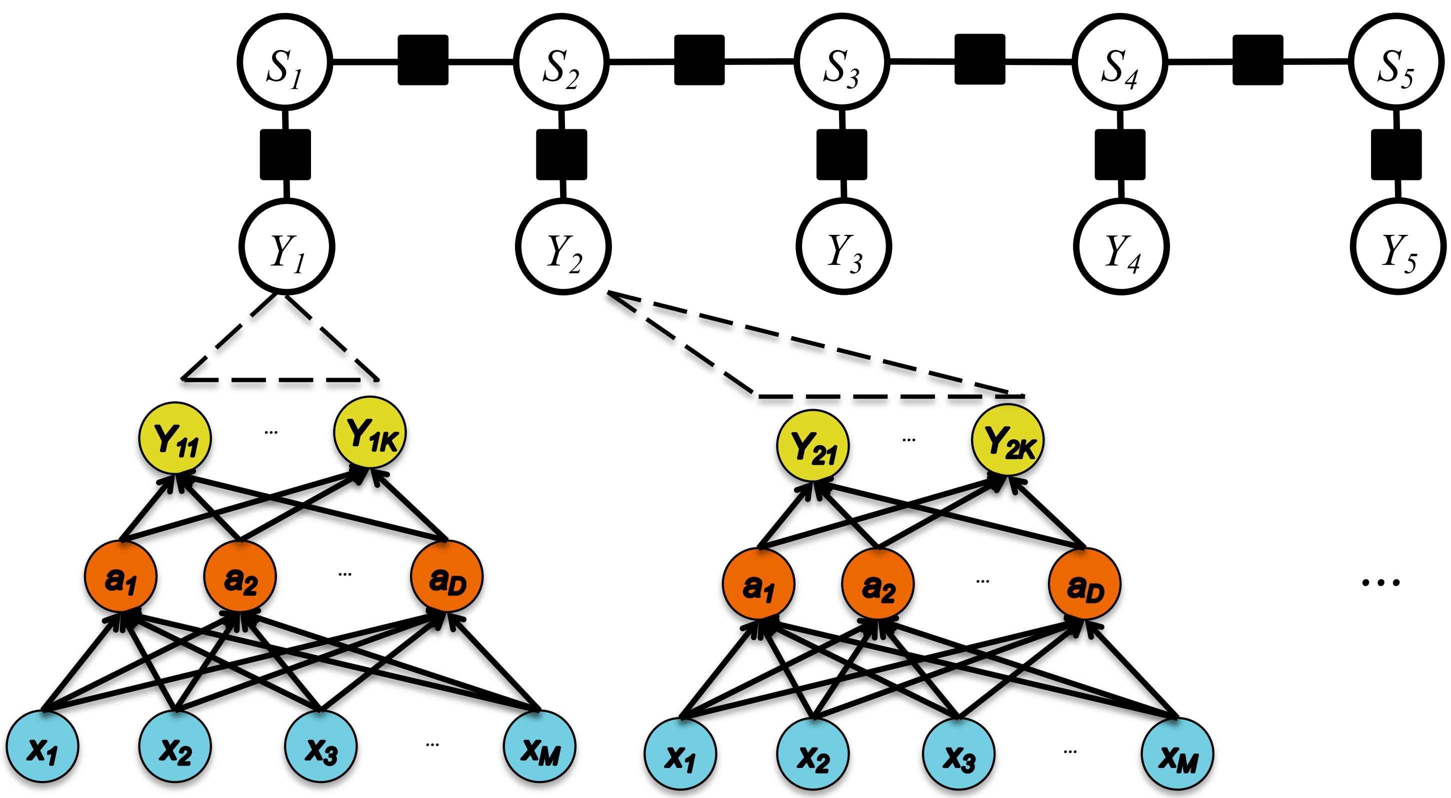
$$\alpha_{i,t} = P(Y_1^t \text{ and } S_t = i \mid model) = b_{i,t} \sum_j a_{ji} \alpha_{j,t-1}$$

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

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

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.

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

4. Train with SGD:

nt)

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

How do we compute the gradient?

Recall…

Training

Backpropagation

Graphical Model and Log-likelihood

Neural Network

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

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

How to compute these partial derivatives?

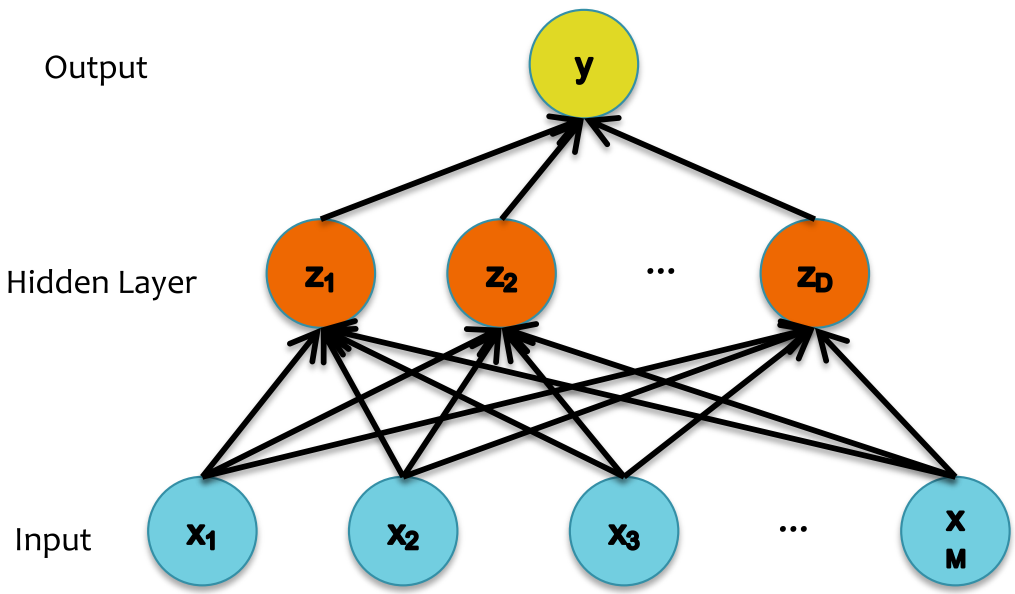
Chain Rule:

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

Training Backpropagation

Recall...

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

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

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

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)

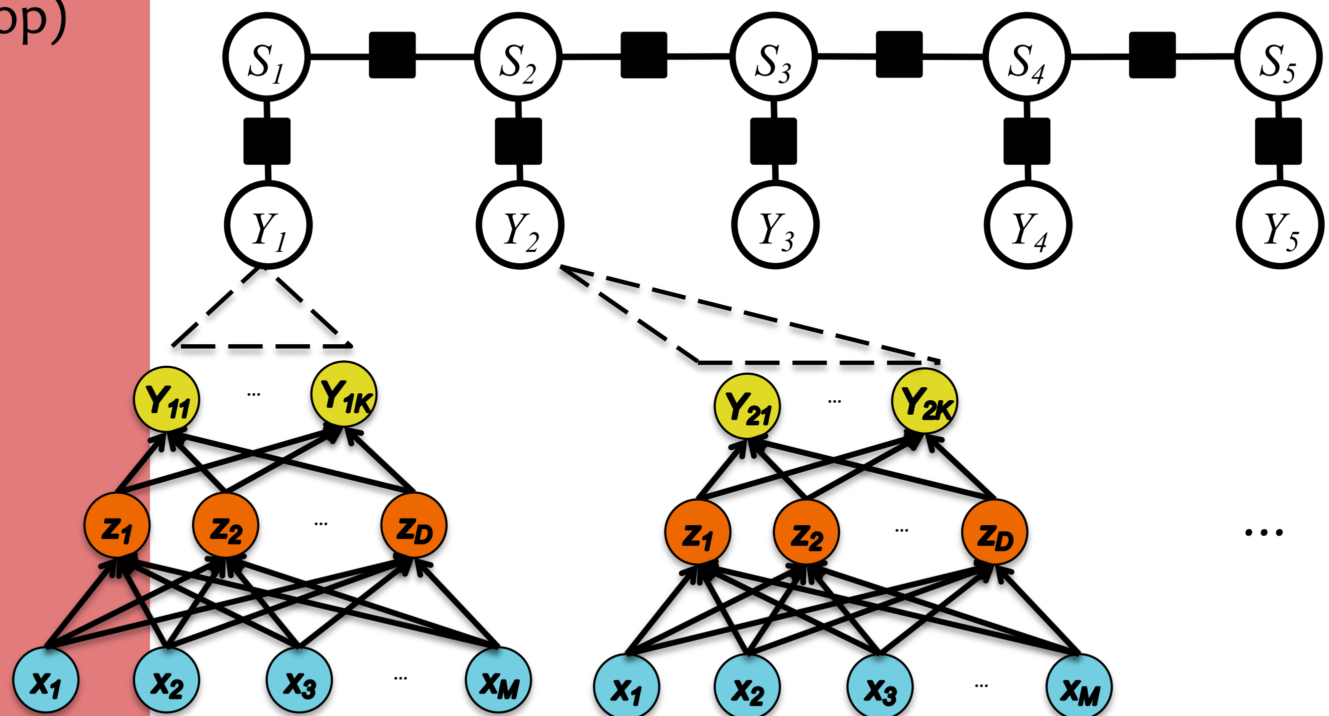
$b_{i,t} = \ldots$ (emissions)

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

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

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

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



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)

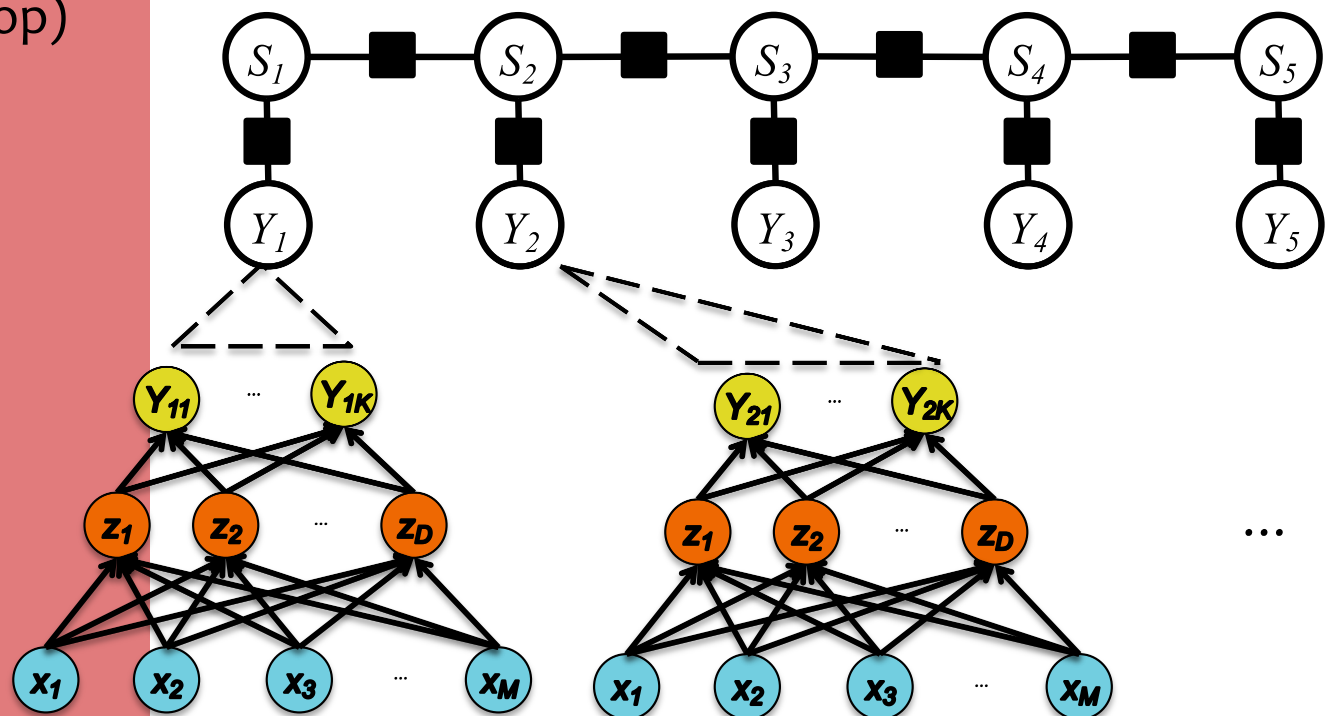
$b_{i,t} = \ldots$ (emissions)

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

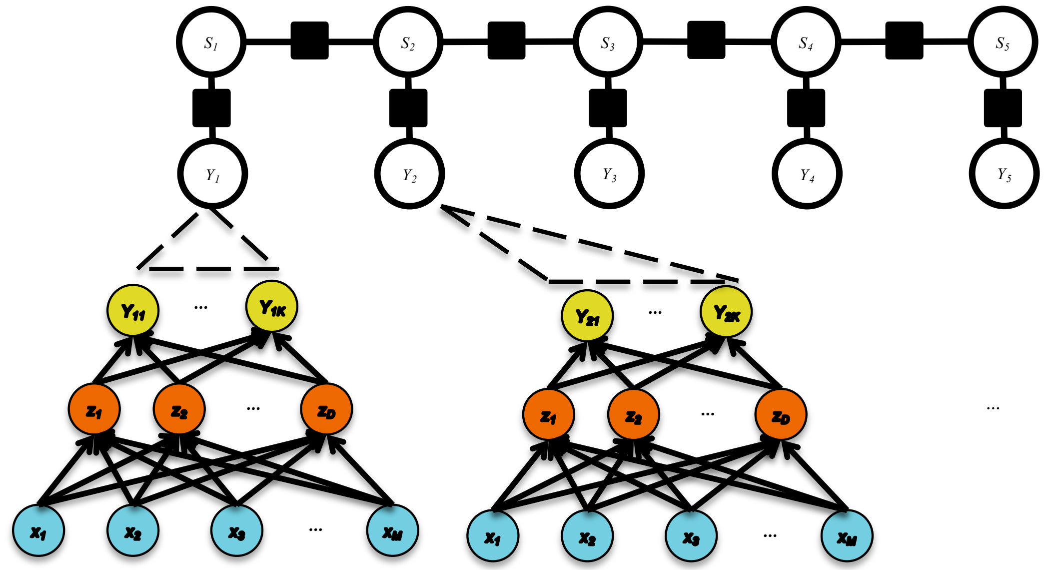
$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

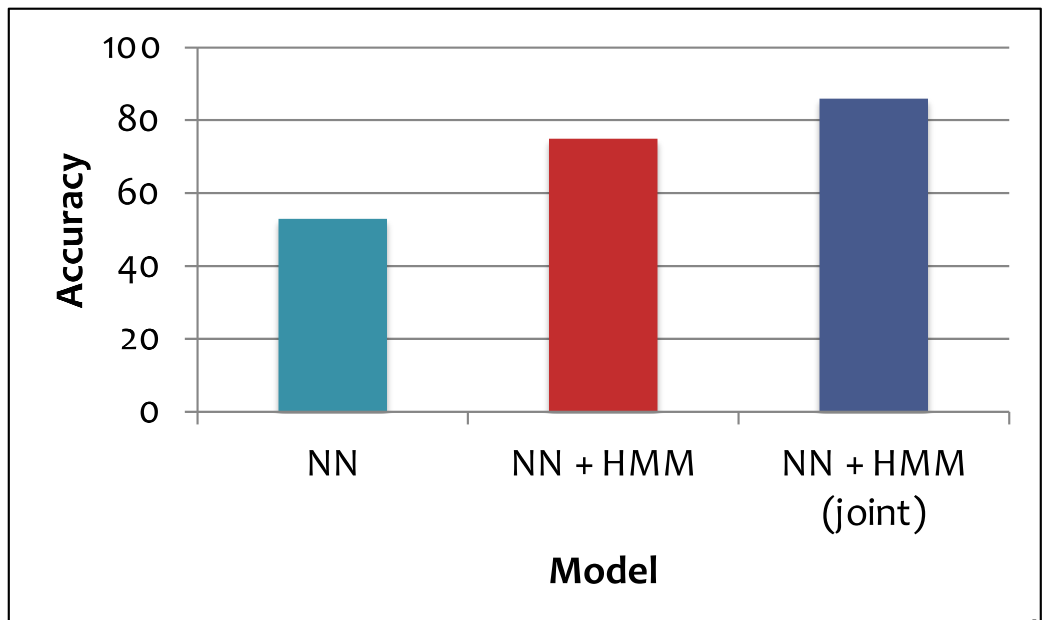
(Bengio et al., 1992)

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)



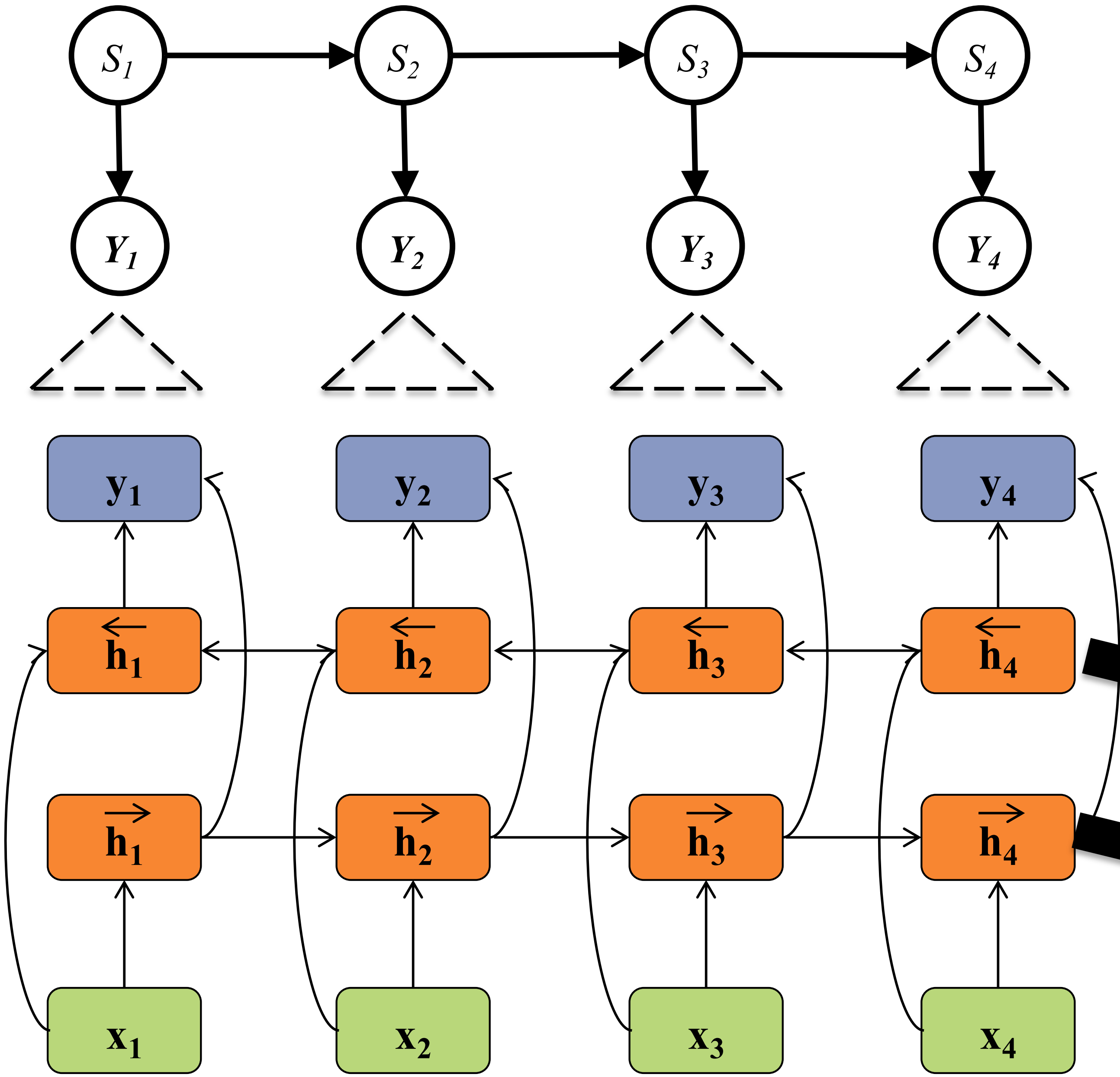


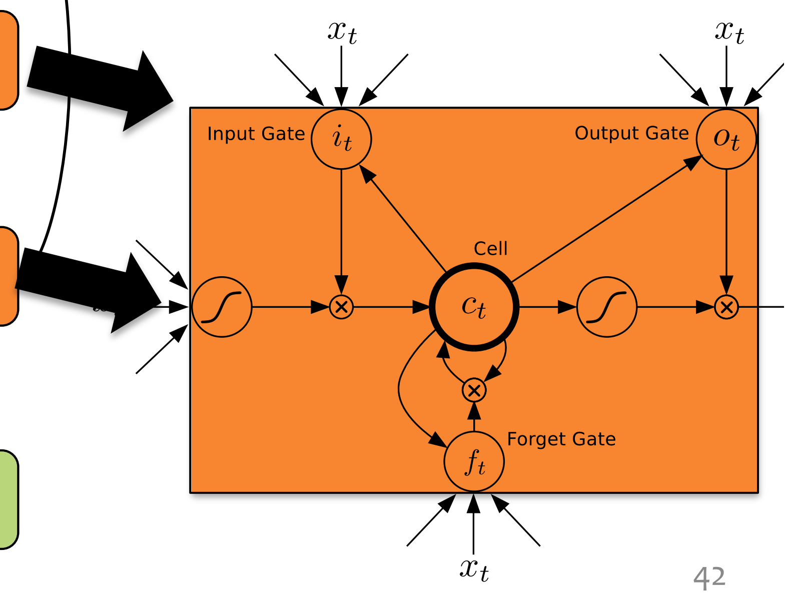
HYBRID:
RNN + HMM

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



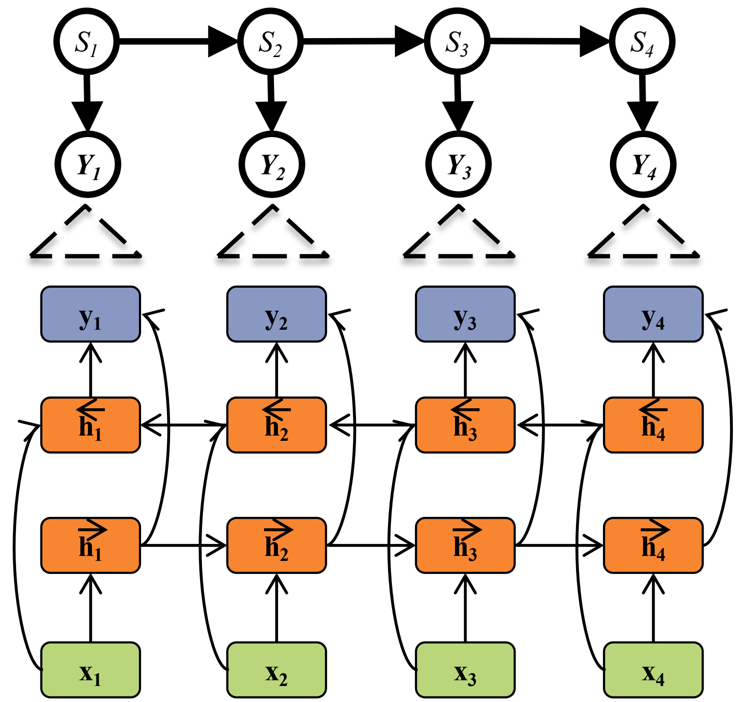


Hybrid: RNN + HMM

(Graves et al., 2013)

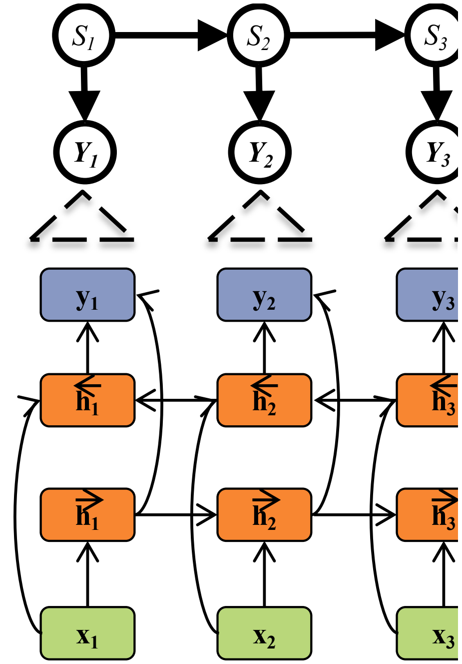
The model, inference, and learning can be **analogous** to our NN + HMM hybrid

- **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

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 $\mathbf{17.99 \pm 0.13}$

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

1. Neural Net only

2. NN + HMM hybrids

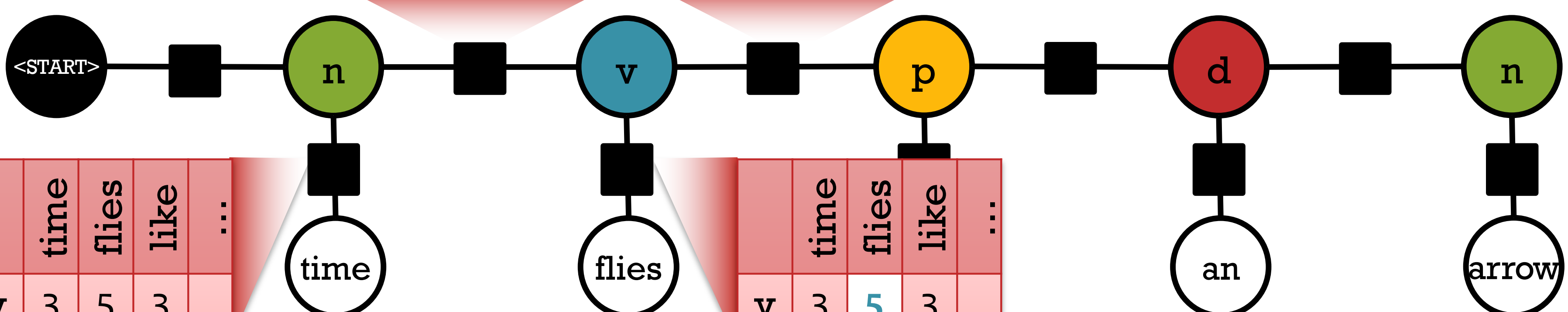
HYBRID:
CNN + CRF

Recall...

Markov Random Field (MRF)

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

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



	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	

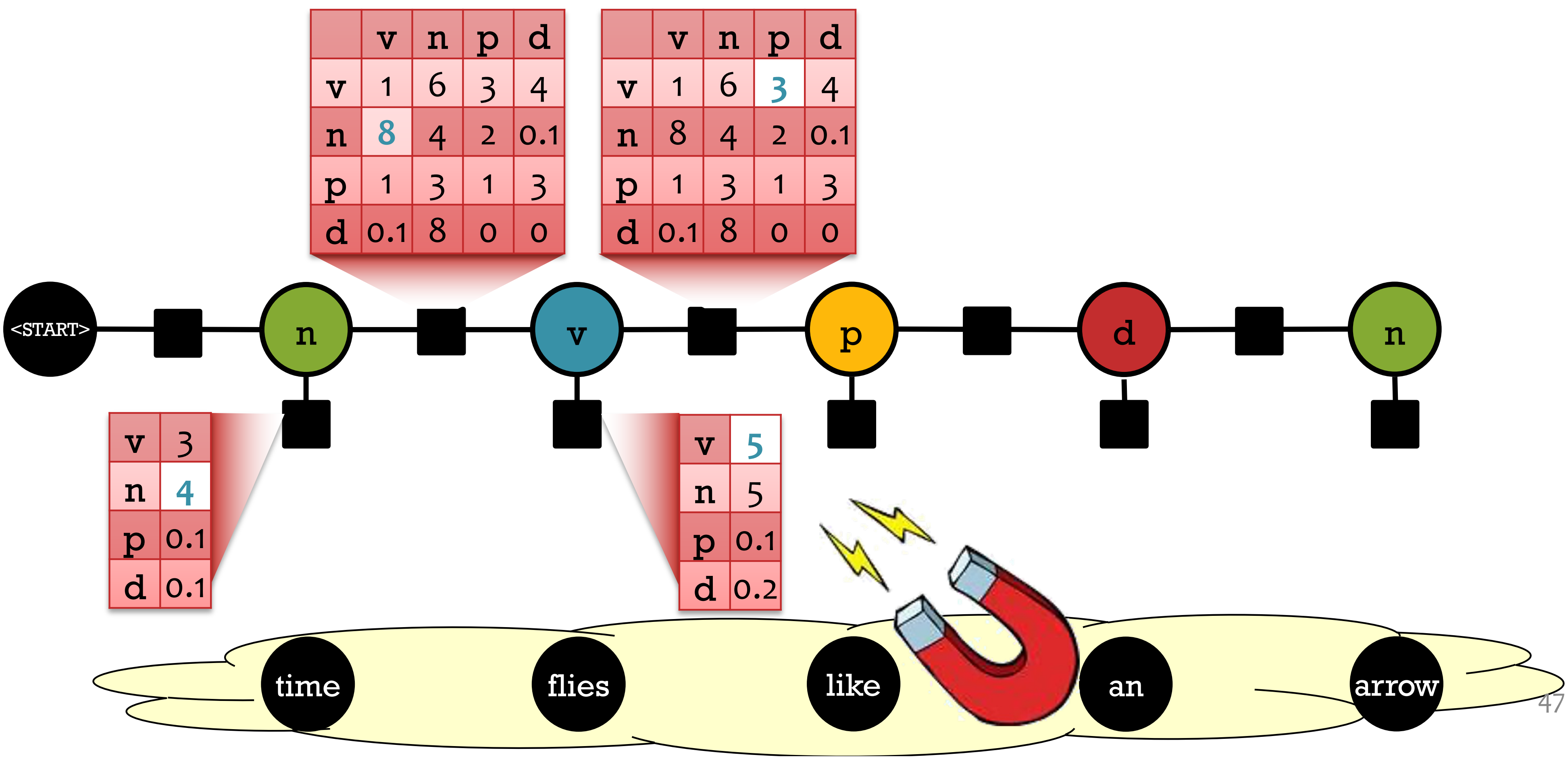
	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

Conditional Random Field (CRF)

Recall...

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

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



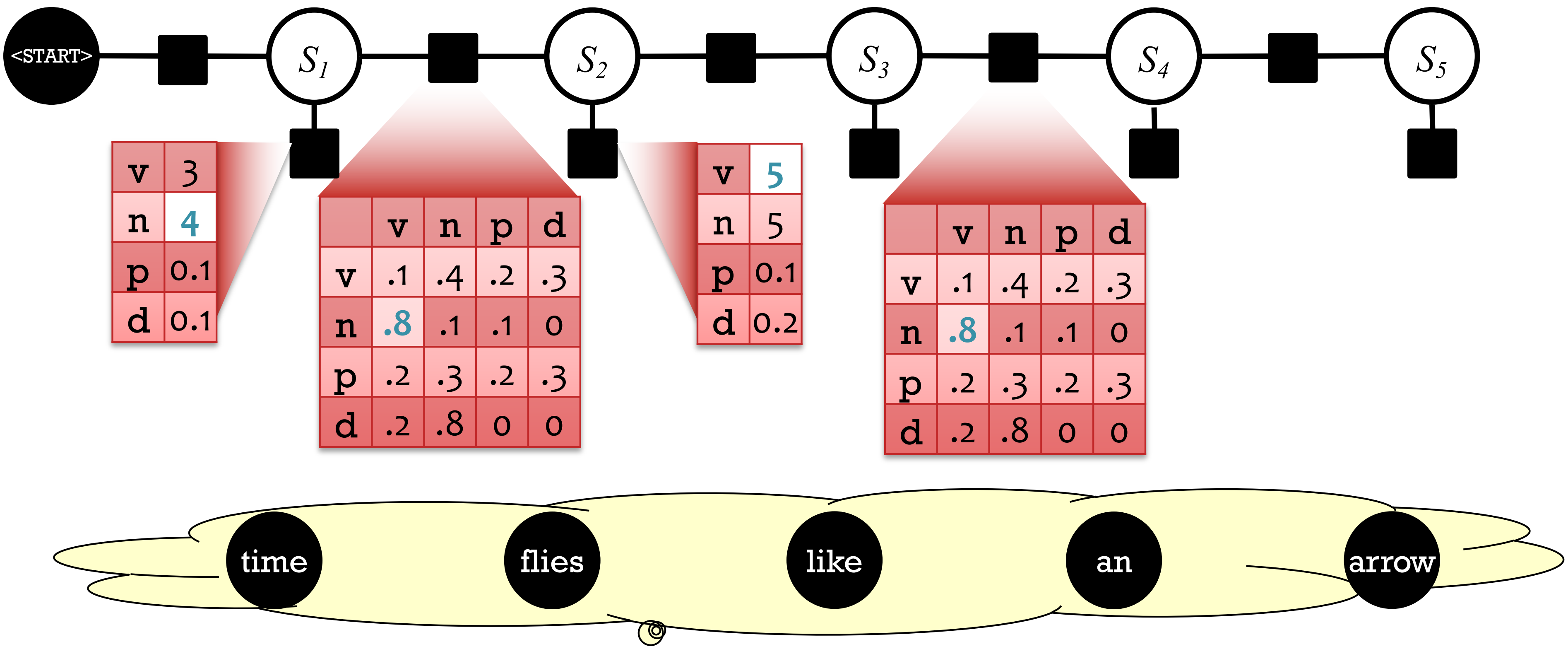
	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	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

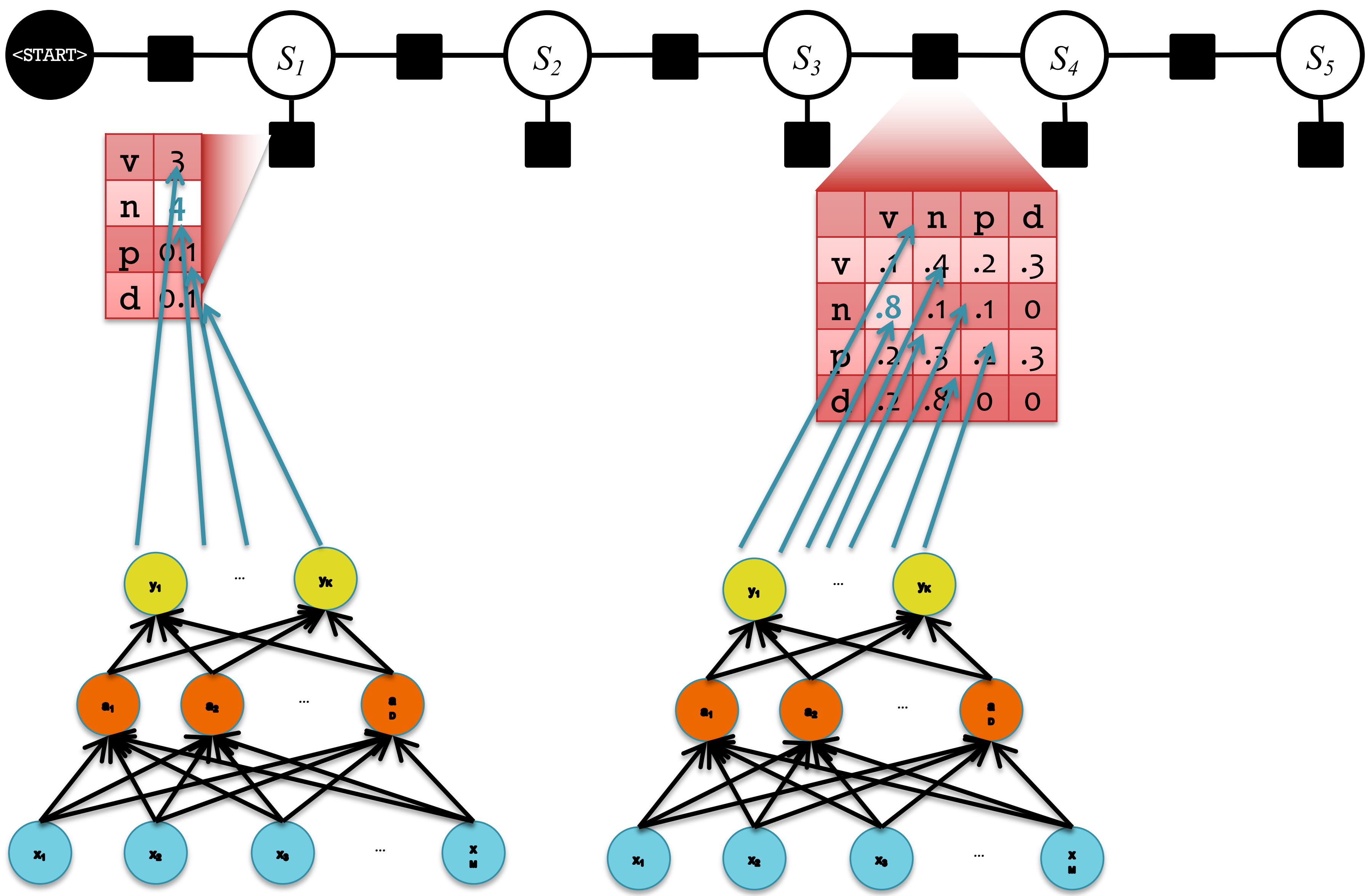
Hybrid: Neural Net + CRF



- 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**

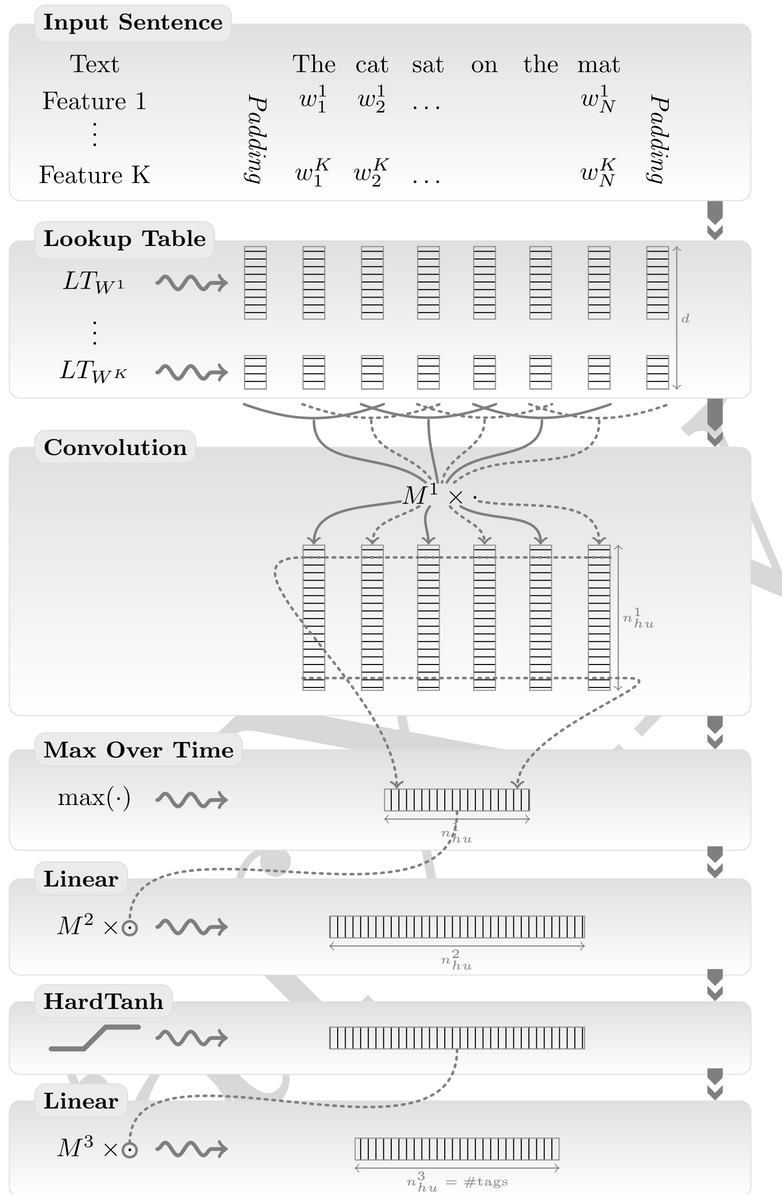


Figure from (Collobert & Weston, 2011)

(Collobert & Weston, 2011)

Hybrid: CNN + CRF

"NN + SLL"

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

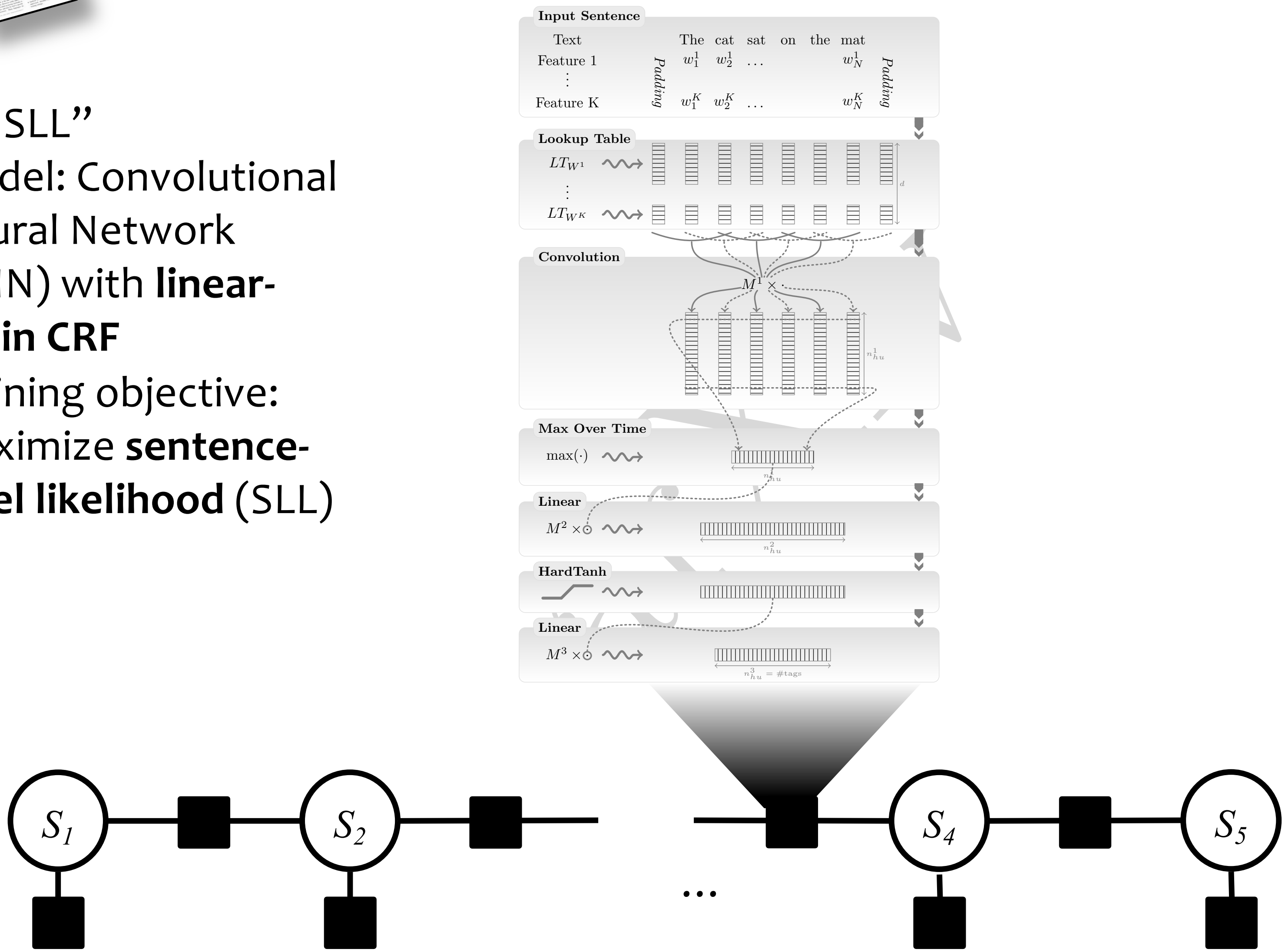


Figure from (Collobert & Weston, 2011)

(Collobert & Weston, 2011)

Hybrid: CNN + CRF

"NN + WLL"

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

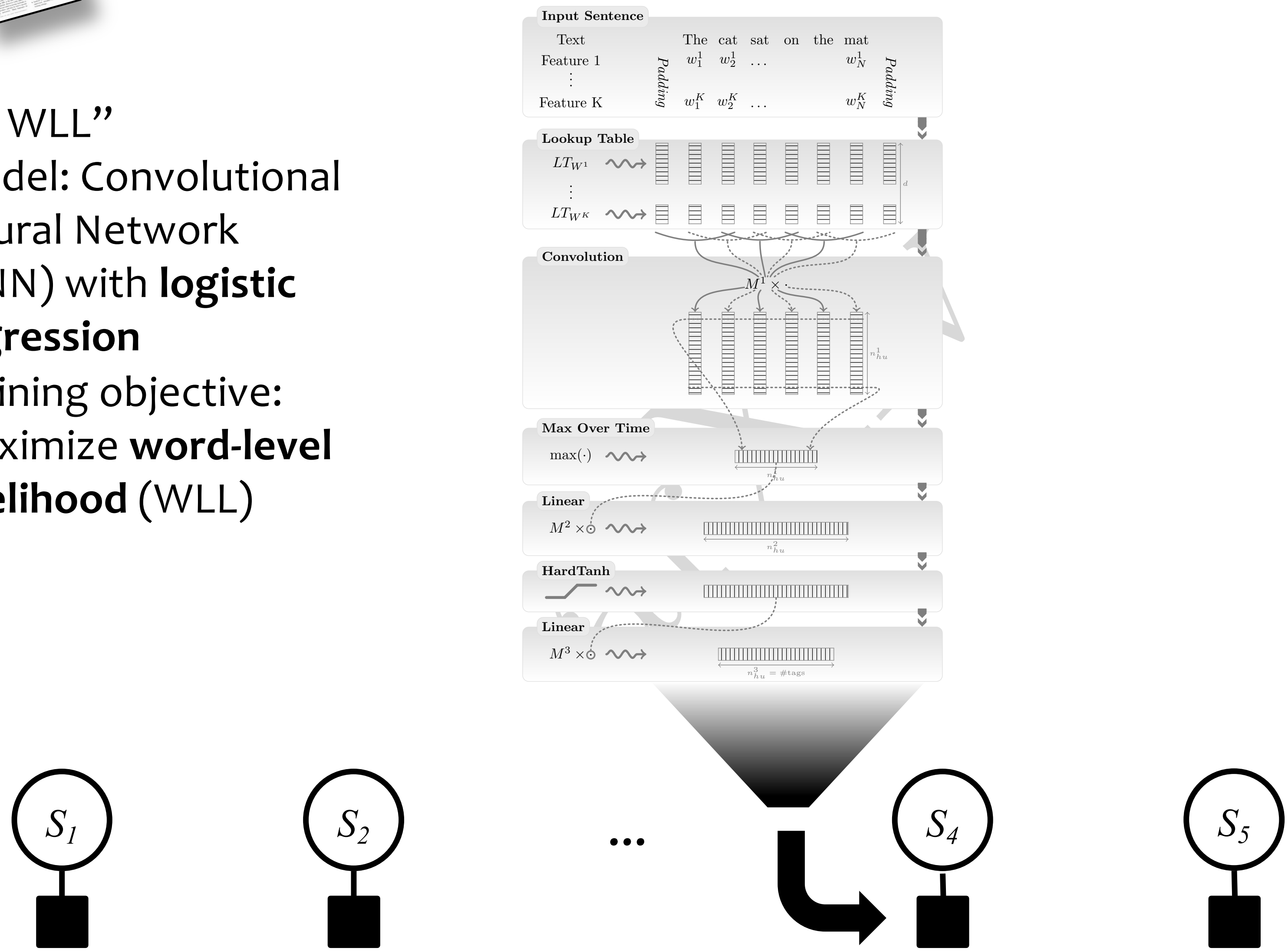


Figure from (Collobert & Weston, 2011)

(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

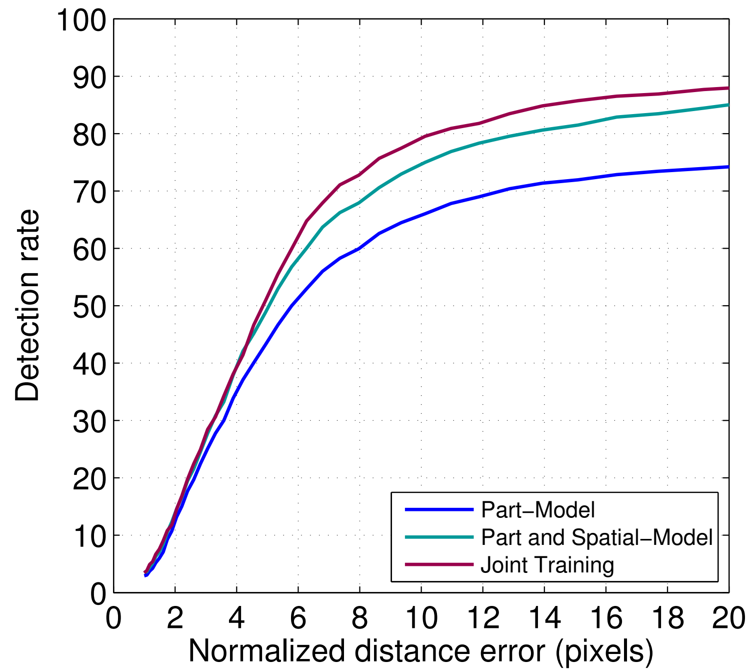
Approach	**POS** (PWA)	**Chunking** (F1)	**NER** (F1)	**SRL** (F1)
Benchmark Systems	97.24	94.29	89.31	77.92
NN+WLL	96.31	89.13	79.53	55.40
NN+SLL	96.37	90.33	81.47	70.99

(Thompson et al., 2014)

Hybrid: CNN + MRF

Experimental Setup:

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



Tricks of the Trade

- **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**!

MBR DECODING

Minimum Bayes Risk Decoding

- Suppose we given a loss function $l(\boldsymbol{y}', \boldsymbol{y})$ and are asked for a single tagging
- How should we choose just one from our probability distribution $p(\boldsymbol{y}|\boldsymbol{x})$?
- A minimum Bayes risk (MBR) decoder $h(\boldsymbol{x})$ 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

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

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:

$$\begin{aligned} h_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \underset{\hat{\boldsymbol{y}}}{\operatorname{argmin}} \sum_{\boldsymbol{y}} p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})(1 - \mathbb{I}(\hat{\boldsymbol{y}}, \boldsymbol{y})) \\ &= \underset{\hat{\boldsymbol{y}}}{\operatorname{argmax}}\ p_{\boldsymbol{\theta}}(\hat{\boldsymbol{y}} \mid \boldsymbol{x}) \end{aligned}$$

which is exactly the MAP inference problem!

Minimum Bayes Risk Decoding

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

Consider some example loss functions:

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

$$h_\theta(x) = \operatorname*{argmin}_{\hat{y}} \left[\sum_y p(y|x) \left(1 - \mathbb{I}(\hat{y} = y)\right) \right]$$

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

$$= \operatorname*{argmin}_{\hat{y}} - p(\hat{y}|x)$$

$$= \operatorname*{argmax}_{\hat{y}} \; p(\hat{y}|x)$$

MBR Decoders

Q: If loss(y, y*) additively decomposes in the same way as log 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…