A Hybrid: Deep Learning and Graphical Models

Kayhan Batmanghelich

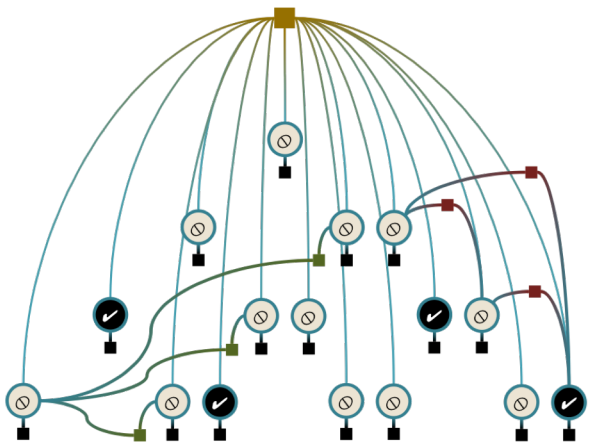
Outlines

- Joint deep feature extraction and learning
- Variational Auto Encoder (VAE)
- Generative Adversarial Networks (GANs)

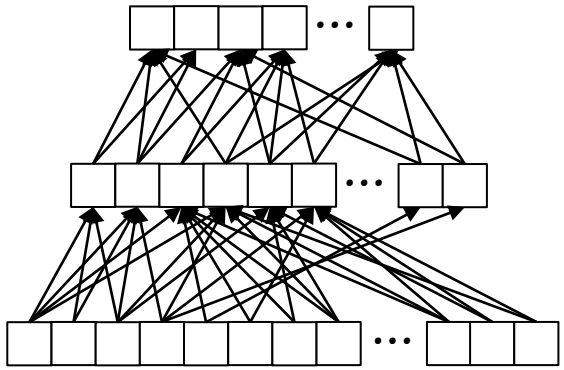
Motivation:

Hybrid Models

Graphical models let you encode domain knowledge



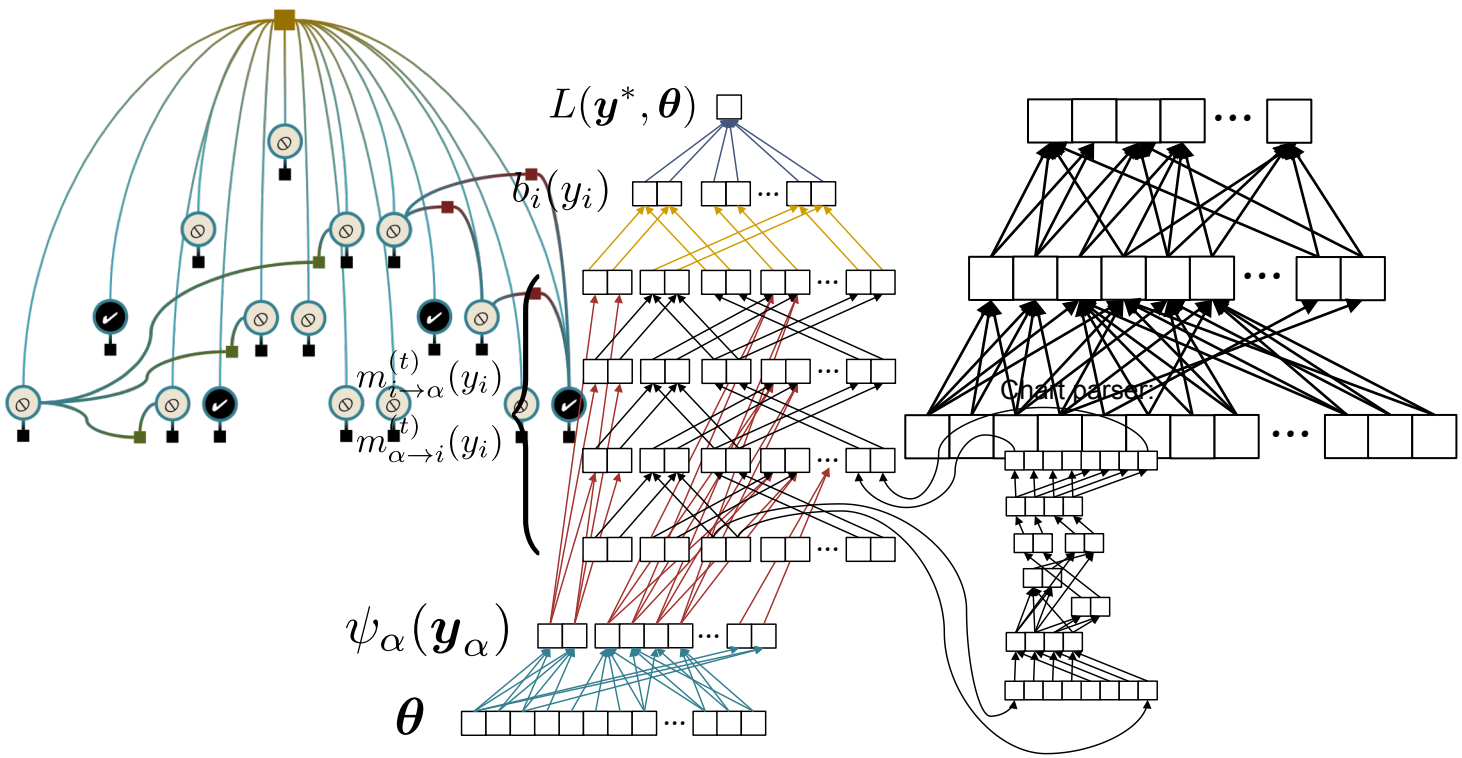
Neural nets are really good at fitting the data discriminatively to make good predictions



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



HYBRID:
NEURAL NETWORK + HMM

Recall…

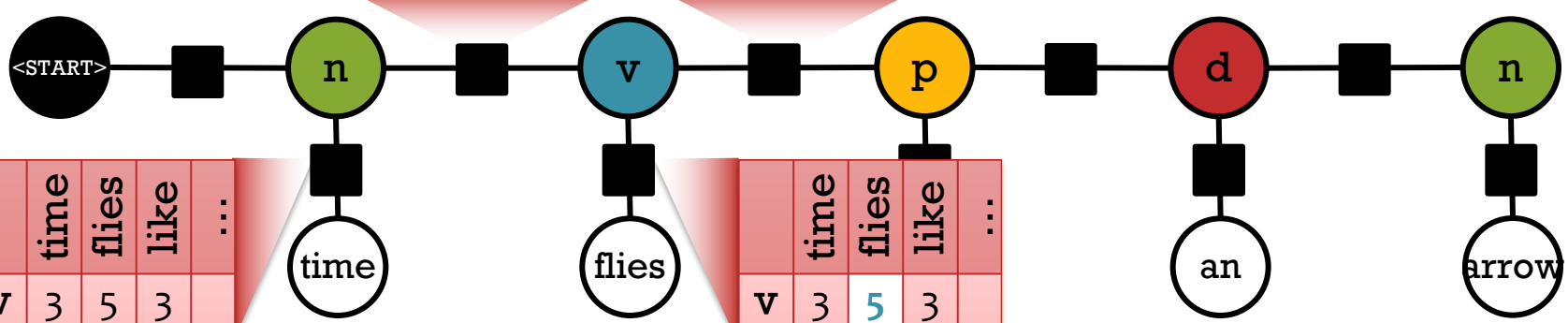
Markov Random Field (MRF)

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…

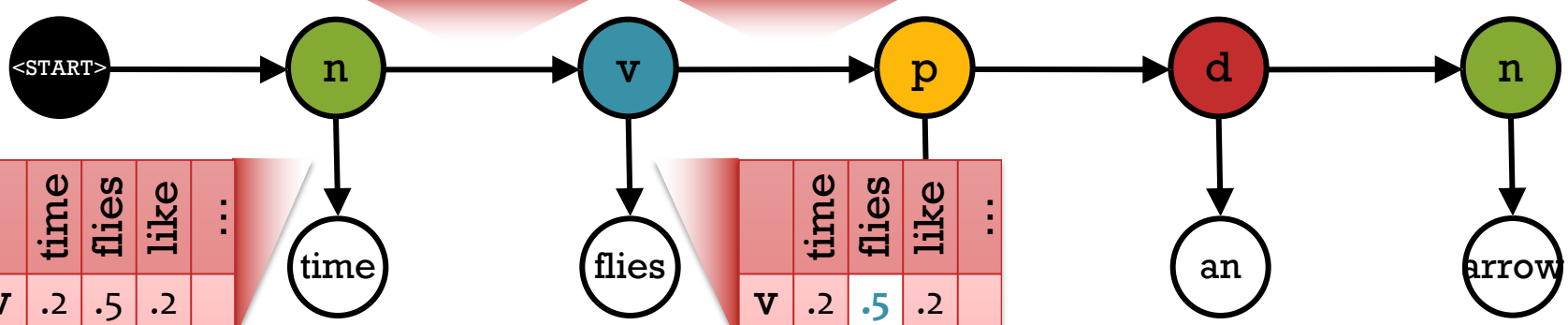
Hidden Markov Model

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

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

	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)

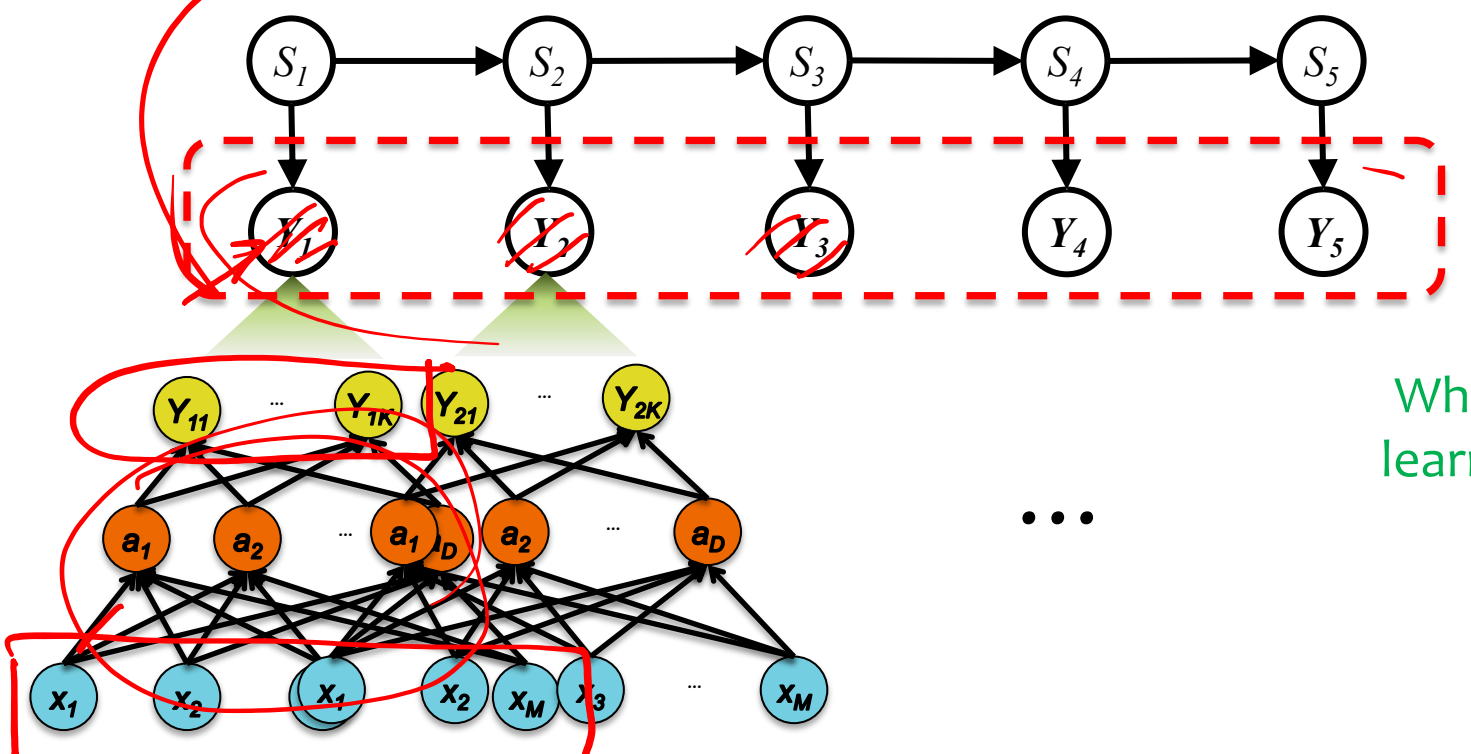
Discrete HMM state: $S_t \in \{/p/, /t/, /k/, /b/, /d/, \ldots, /g/\}$

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:

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



features

...

What if we jointly learn the features?

Hybrid: NN + HMM

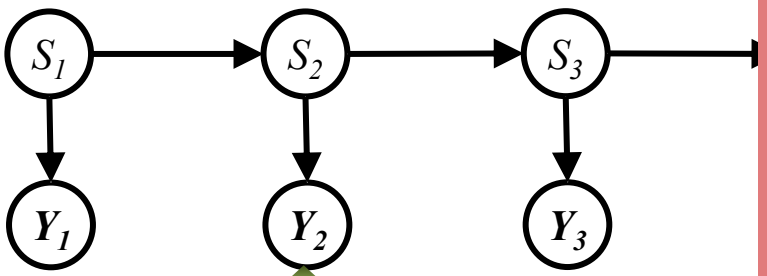
(Bengio et al., 1992)

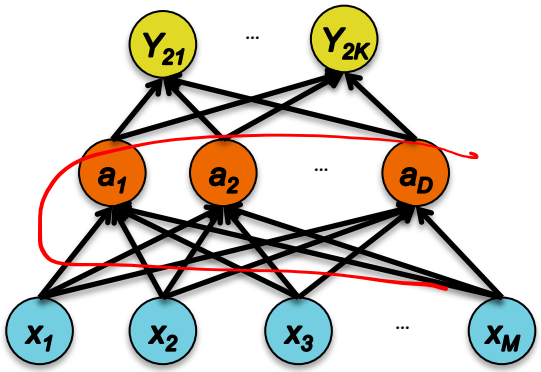
Discrete HMM state: $S_t \in \{/p/, /t/, /k/, /b/, /d/, \ldots, /g/\}$

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

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

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



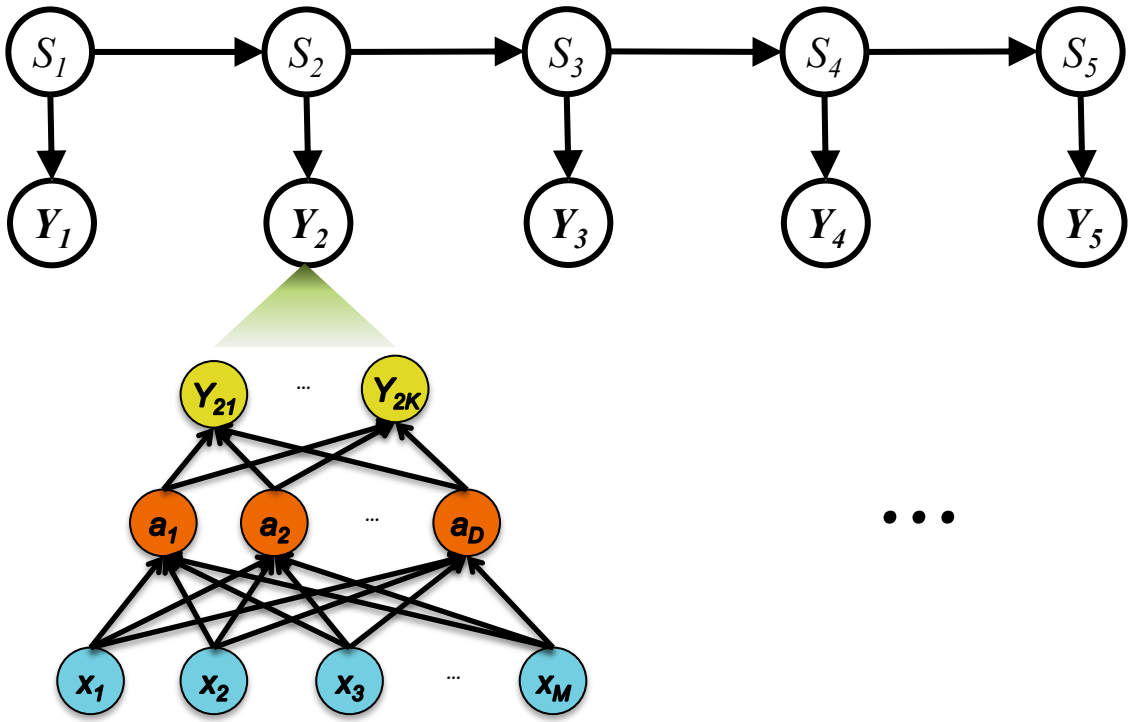


Lots of oddities to this picture:

- **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.

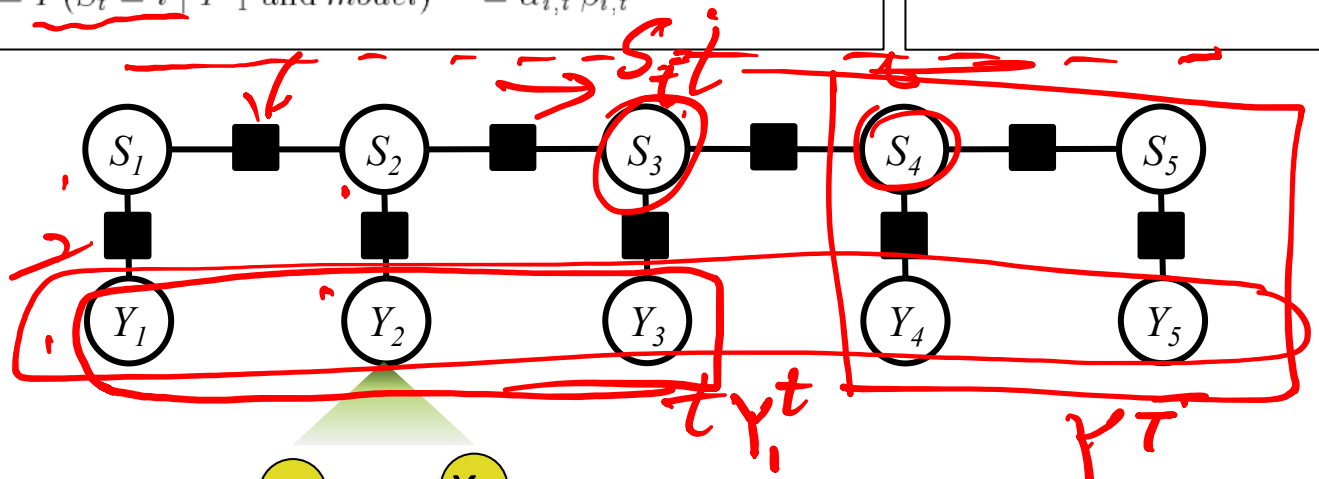
$$\alpha_{i,t} = P(Y_1^t \text{ and } S_t = i \mid model) = b_{i,t} \sum_j a_{ji} \alpha_{j,t-1}$$

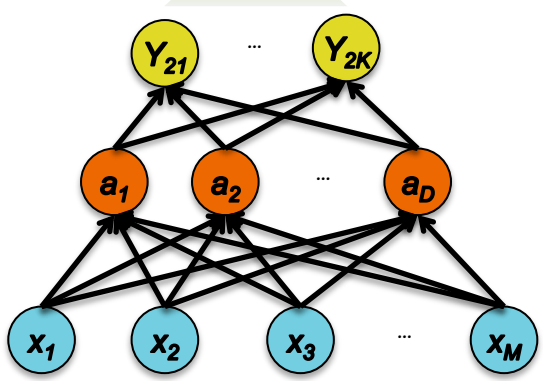
$$\beta_{i,t} = P(Y_{t+1}^T \mid S_t = i \text{ and } model) = \sum_j a_{ij} b_{j,t+1} \beta_{j,t+1}$$

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

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.

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

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

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?

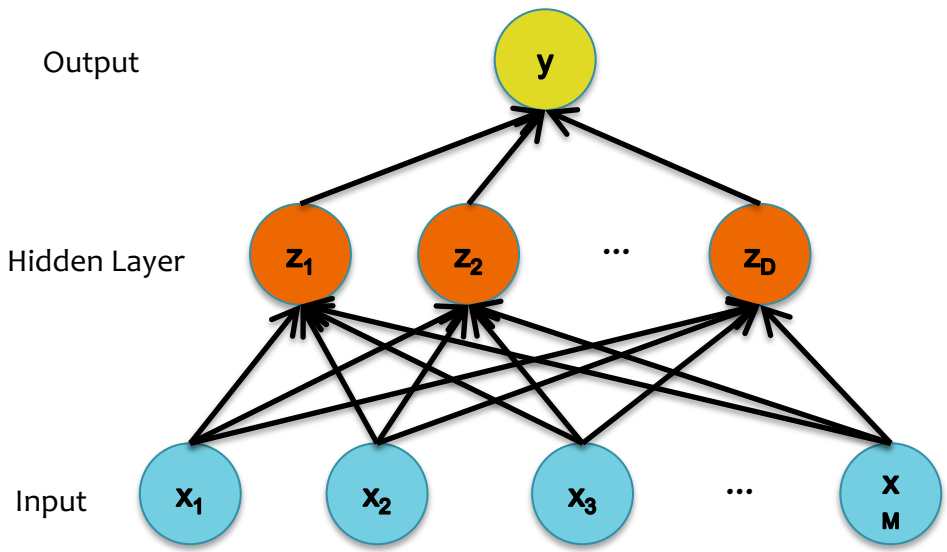
Chain Rule:

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

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 $x_i, \ \forall i$

Recall...

Training Backpropagation

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

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

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

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

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

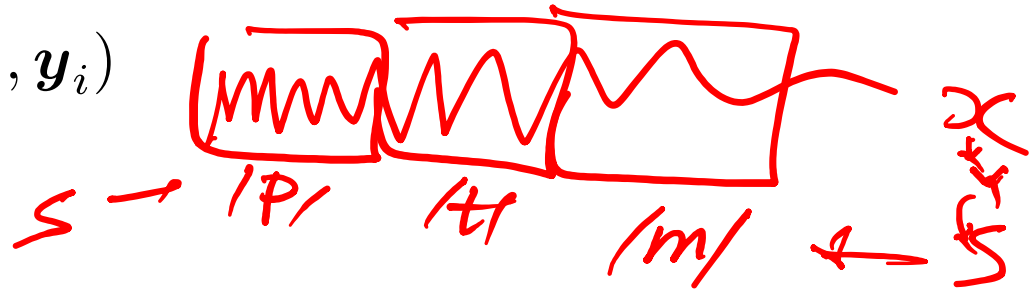
$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(a_j)}{(\exp(a_j) + 1)^2}$$

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

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

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)

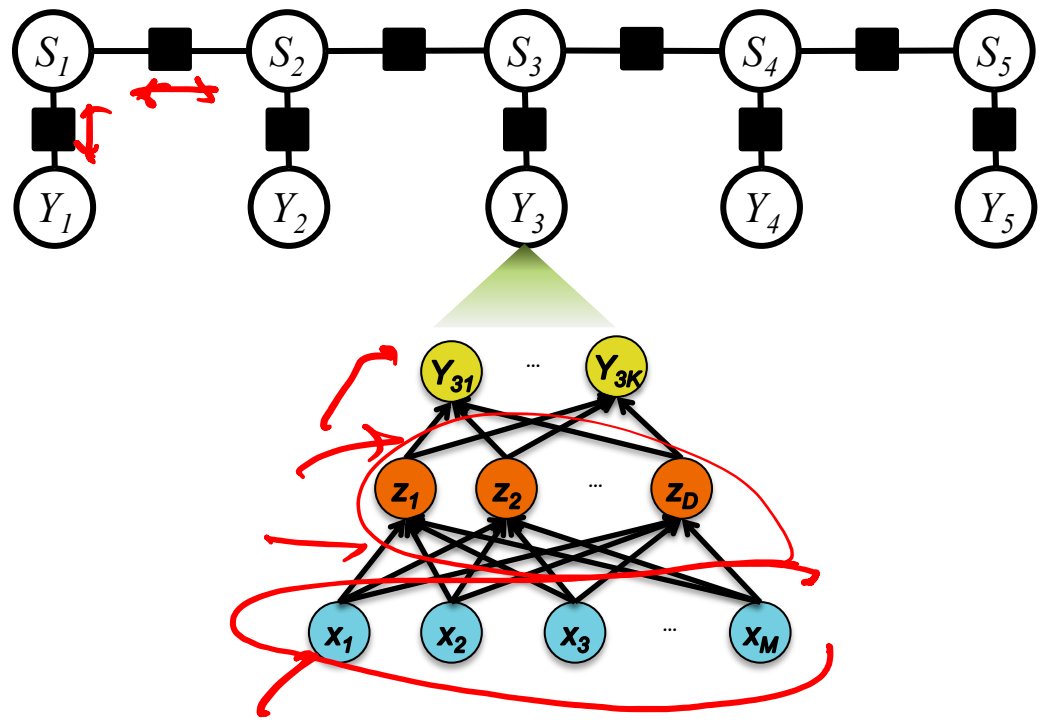
$b_{i,t} = \ldots$ (emissions)

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

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

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

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



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)

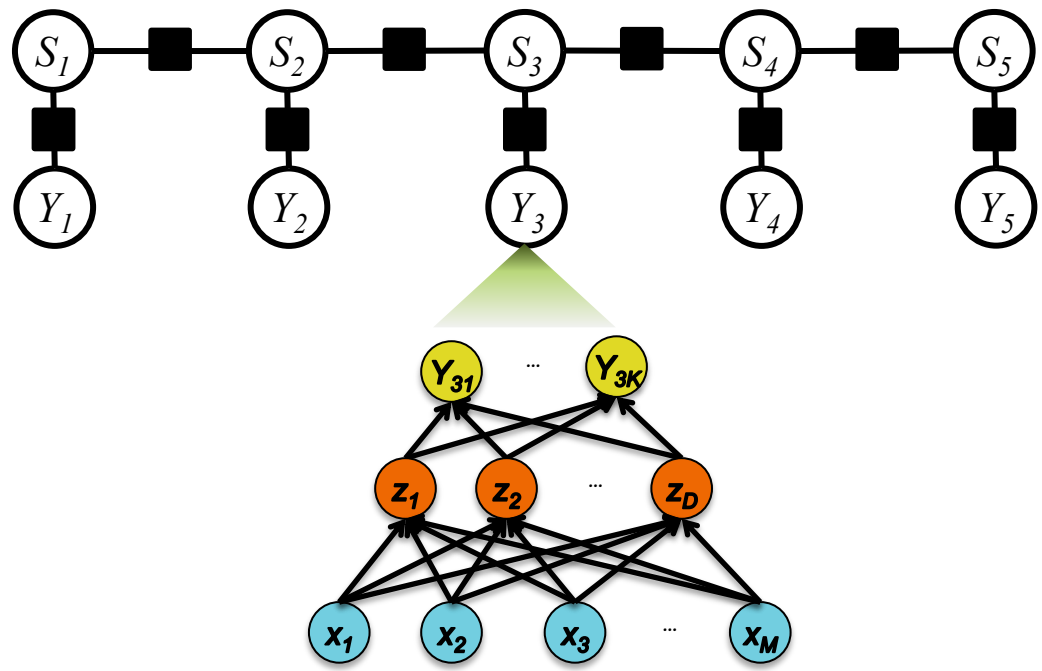
$b_{i,t} = \ldots$ (emissions)

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

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

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

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



Hybrid: NN + HMM

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

Forward computation

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

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

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

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

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

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

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

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

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

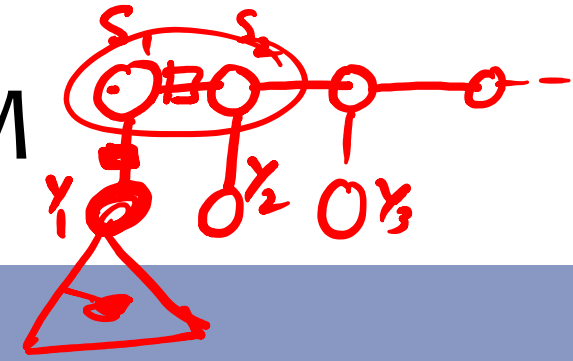
$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

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

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

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

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

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

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

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

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

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

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

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

Backward computation

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

$$b_{i,t}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

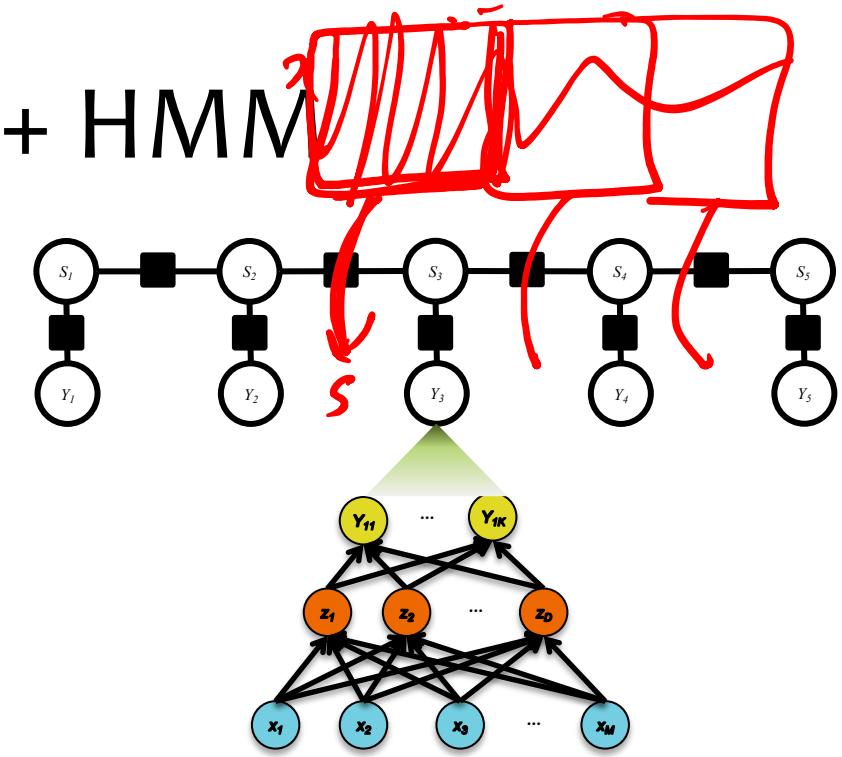
$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

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

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

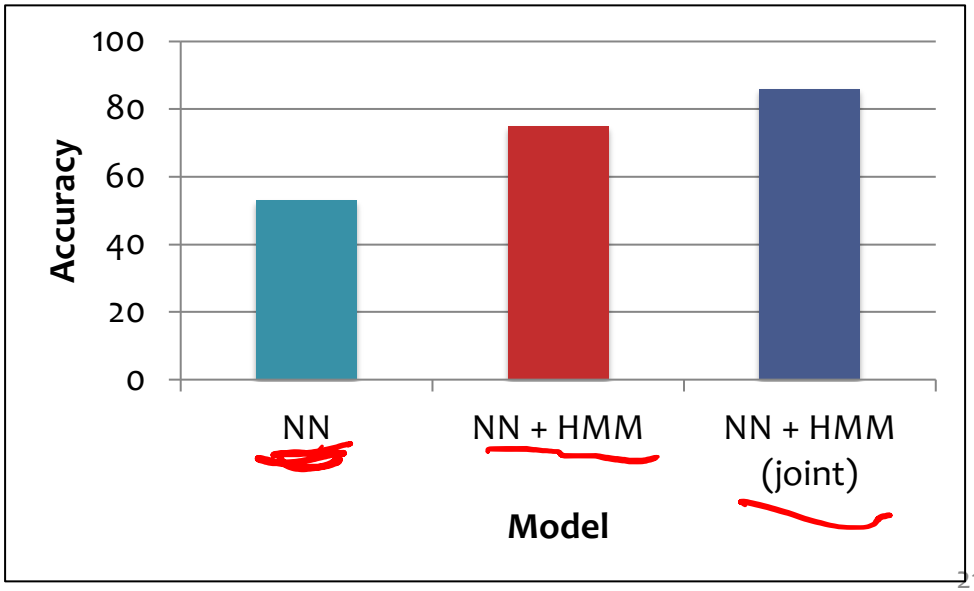
(Bengio et al., 1992)

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:
CNN + CRF

Recall…

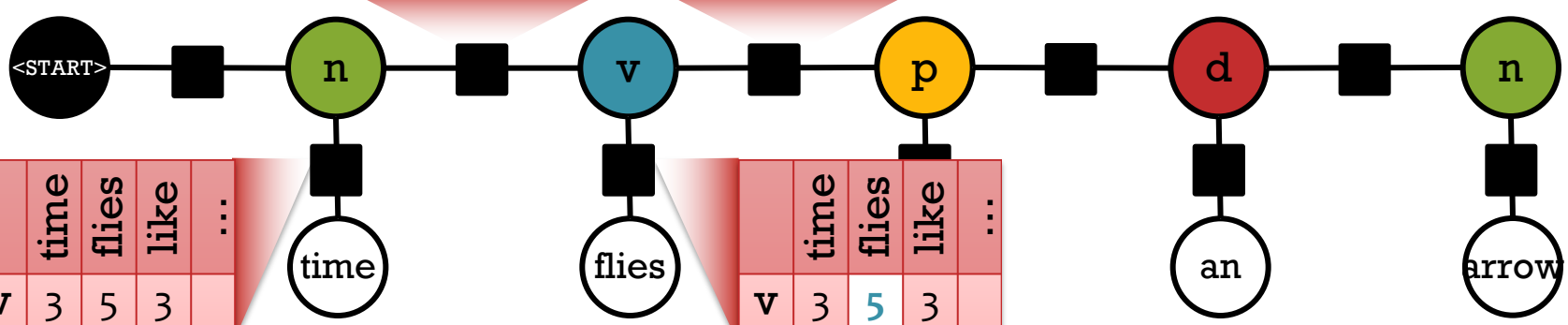
Markov Random Field (MRF)

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

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

	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)

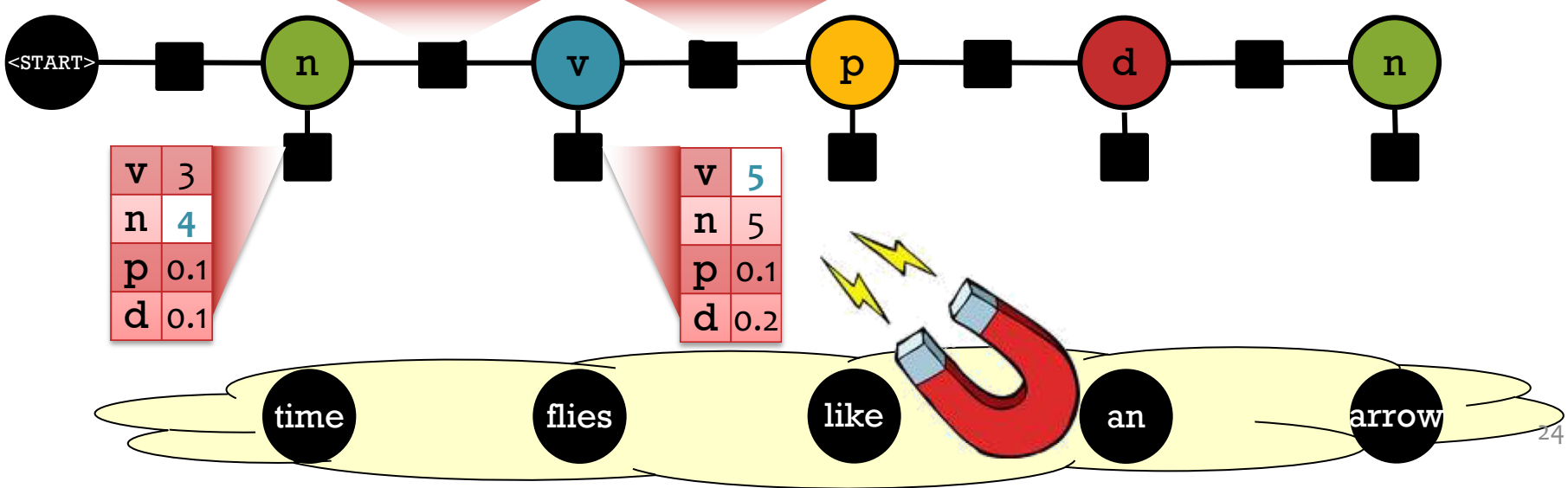
Recall…

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

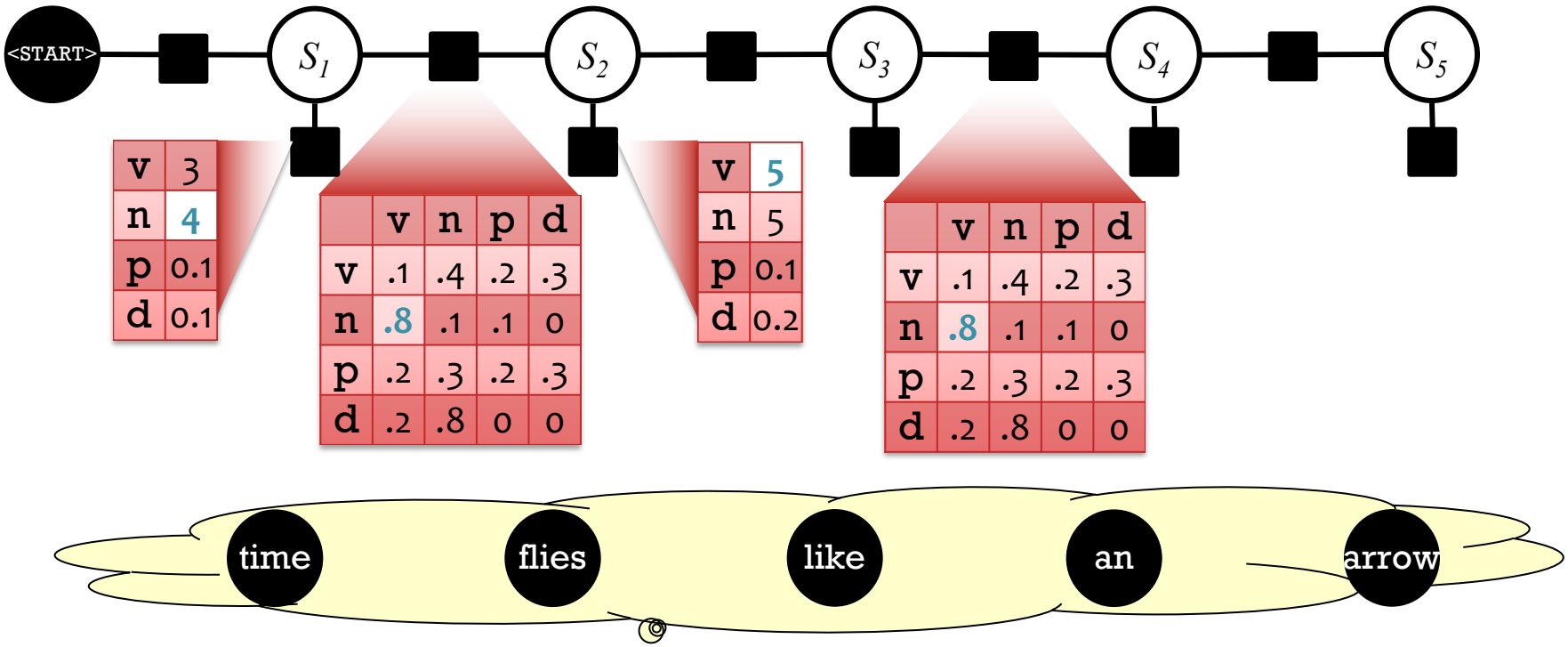
	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0



v	3
n	4
p	0.1
d	0.1

v	5
n	5
p	0.1
d	0.2

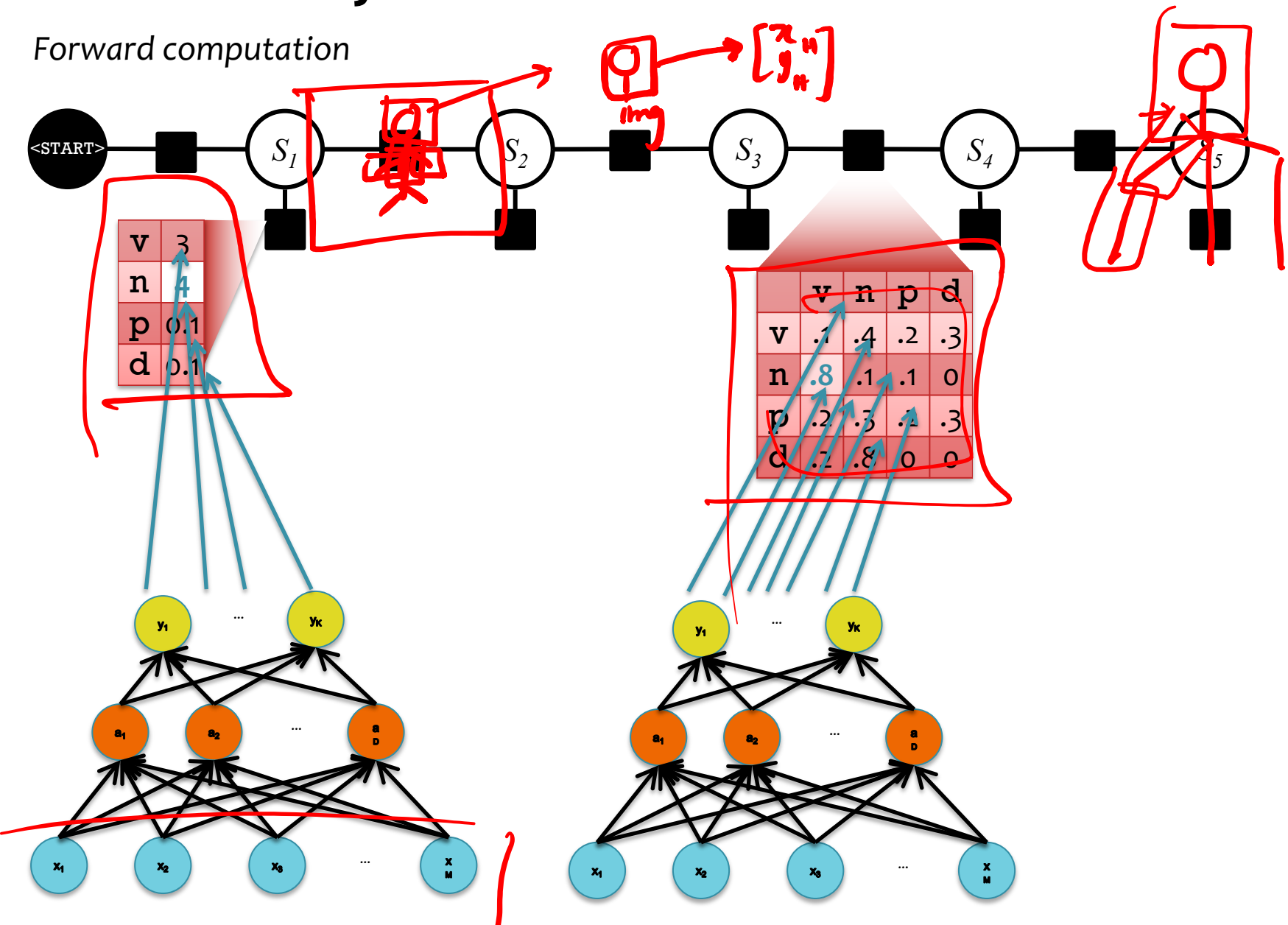
Hybrid: Neural Net + CRF



- 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

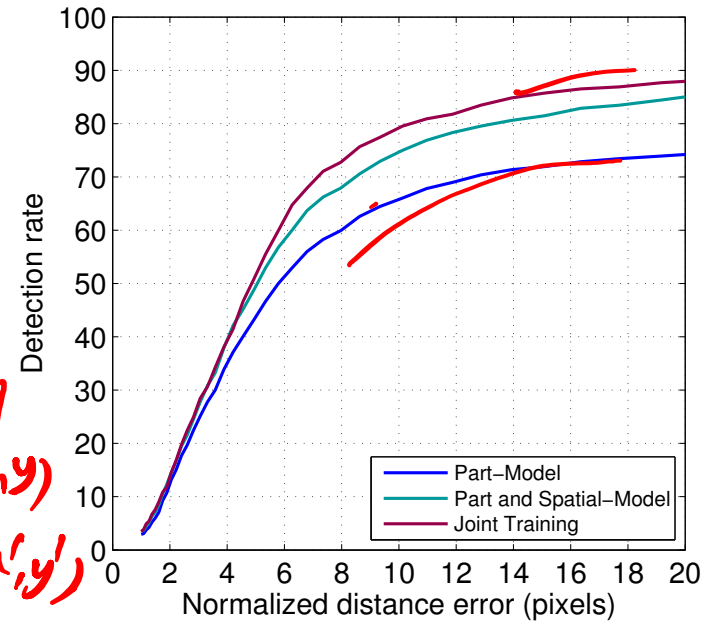
Forward computation



Hybrid: CNN + MRF

Experimental Setup:

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



Thompson et al., "Joint Training of a Convolutional Network and a Graphical Model for Human Pose Estimation" 2014

VAE: VARIATIONAL AUTO-ENCODER

Recap: Interpreting the Lower Bound (ELBO)

$$\mathcal{F}(y, q) = \boxed{\mathbb{E}_{q(z)}\left[\log p(y|z)\right]} - \boxed{KL\left[q(z)||p(z)\right]}$$

Data

Approximate Posterior

Reconstruction

Penalty

P(D)

KL(.,.)

ELBO

Approximate posterior distribution *q(z)*: Best match to true posterior *p*(*z*|*y*), one of the unknown inferential quantities of interest to us.

Reconstruction Cost: The expected log-likelihood measure how well samples from *q*(*z*) are able to explain the data *y*.

Penalty: Ensures the the explanation of the data *q*(*z*) doesn't deviate too far from your beliefs *p*(*z*). A mechanism for realising Okham's razor.

$x \xrightarrow[q(z|x)]{\text{encoder}} z \xrightarrow[p(x|z)]{\text{decoder}} x$

Recap: Decoder Encoder View

$$\max_{\phi,\theta} \mathcal{F}(x, q_\phi) = \mathbb{E}_{q_\phi(z)}\left[\log p_\theta(x|z)\right] - KL\left[q_\phi(z)||p(z)\right]$$

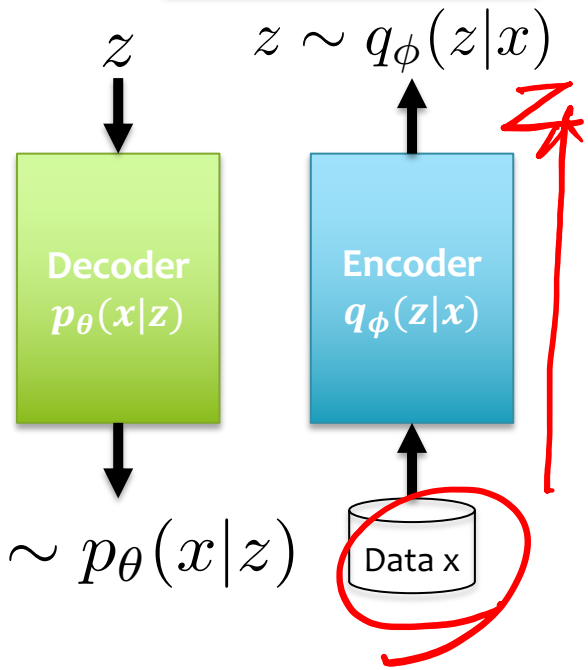
Stochastic encoder

Data code-length

Hypothesis code

Encoder: variational distribution $q_\phi(z|y)$

Decoder: likelihood $p_\theta(y|z)$



The goal: $q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$

$p_\theta(z, x)$

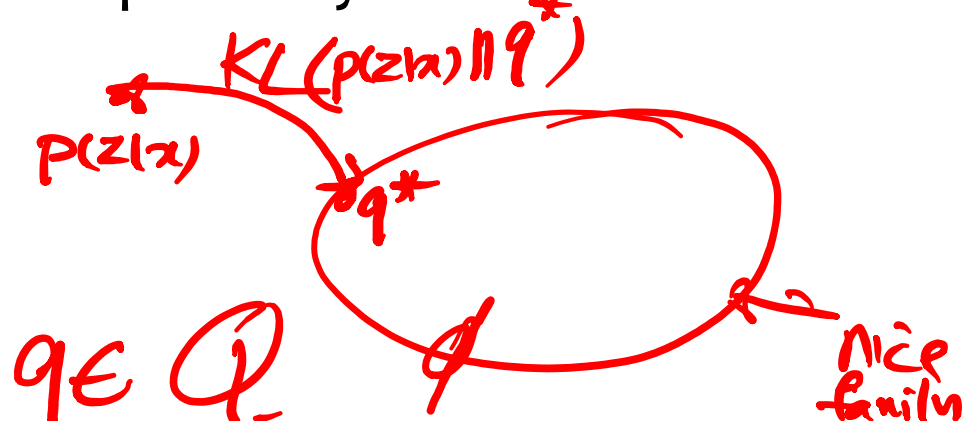
$p(x)$

$$\max_{\phi,\theta} \mathcal{F}(x, q_\phi) = \mathbb{E}_{q_\phi(z)}\left[\log p_\theta(x|z)\right] - KL\left[q_\phi(z)||p(z)\right]$$

Approximate Posterior

How to implement it?

What is q exactly?



Variational Auto Encoder (VAE)

$$\max_{\phi,\theta} \mathcal{F}(x, q_\phi) = \mathbb{E}_{q_\phi(z)}[\log p_\theta(x|z)] - KL[q_\phi(z)||p(z)]$$

Stochastic encoder

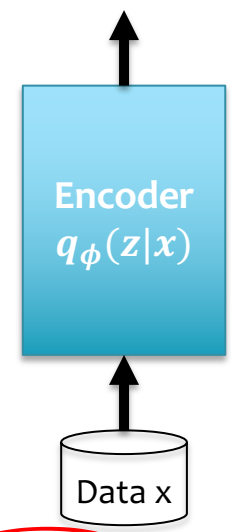
Data code-length

Hypothesis code

$$z \sim q_\phi(z|x)$$

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = q_{\boldsymbol{\phi}}(\mathbf{z}_1, ..., \mathbf{z}_M|\mathbf{x}) = \prod_{j=1}^{M} q_{\boldsymbol{\phi}}(\mathbf{z}_j|Pa(\mathbf{z}_j), \mathbf{x})$$

Amortization
(sharing parameters)



Example: $\mu(x)$ $\log\sigma(x)$

$$(\boldsymbol{\mu}, \log\boldsymbol{\sigma}) = \text{EncoderNeuralNet}_{\boldsymbol{\phi}}(\mathbf{x})$$

$$q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}))$$

$q_\phi(z)$

Mapping the Distributions

$q(z|x) = N(\mu(x), \sigma(x))$

- VAE learns a mapping that transforms a simple distribution, q(z), to a complicate distribution, q(z|x) so that the empirical distribution (in the x-space) are matched.

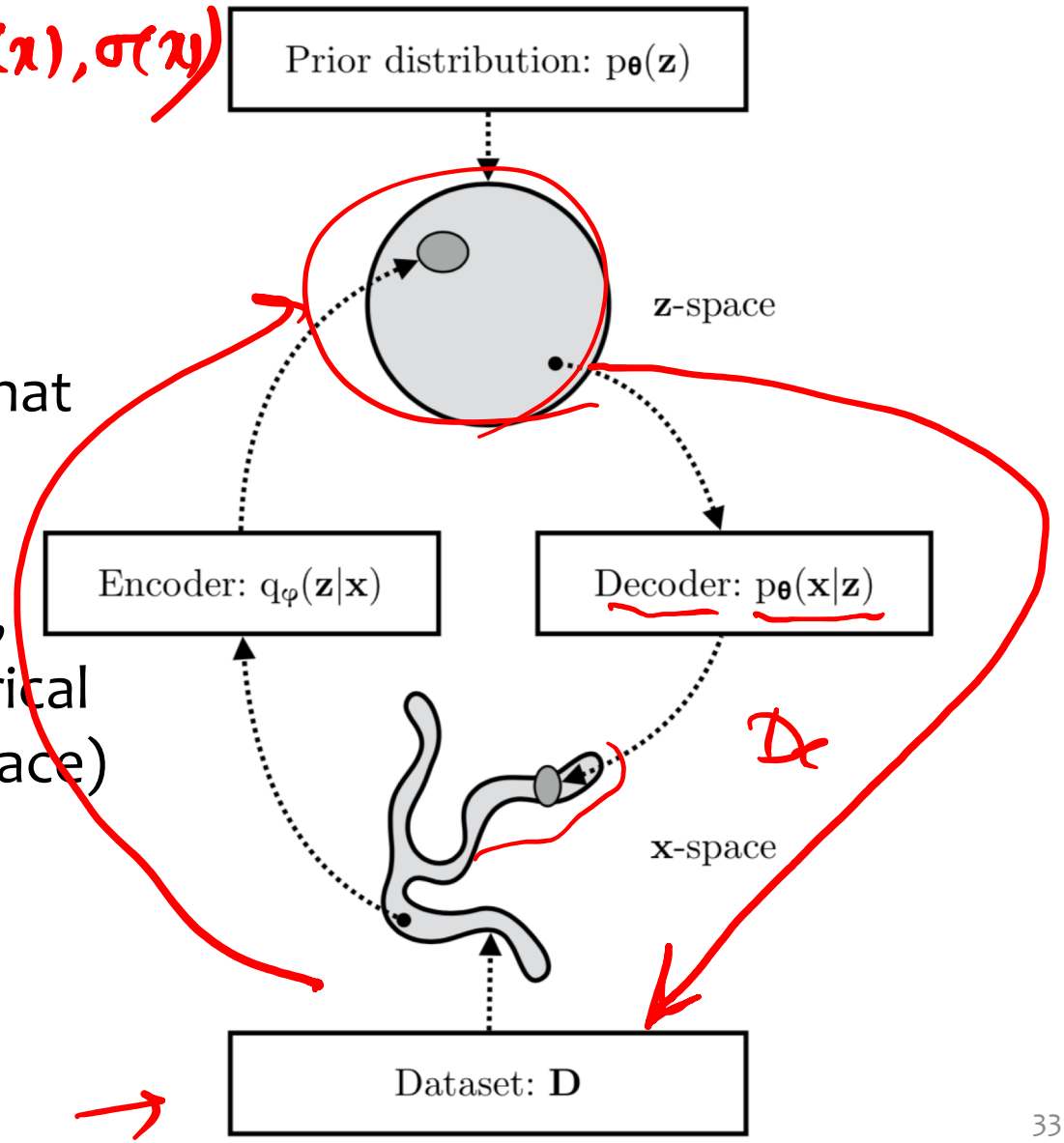


Fig Credit: Kingma's thesis

$$\text{ELBO} = \mathbb{E}_{z \sim q(\cdot|x)} \left[\log \frac{p_\theta(x,z)}{q(z|x)} \right]$$

Learning

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}) \right]$$

Alternative optimization for the variational parameters and then model parameters (VEM).

$$\nabla_\theta \mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\nabla_\theta (\log p_\theta(\mathbf{x},\mathbf{z}) \right]$$

$$\nabla_\phi \mathcal{L}_{\theta,\phi}(\mathbf{x}) = \nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}) \right]$$

$q(z|x) = N(\mu(x), \sigma(x))$

Reparametrization Trick

Changes of variables:

$$\mathbf{z} = \mathbf{g}(\epsilon, \boldsymbol{\phi}, \mathbf{x}) \qquad \epsilon \sim p(\epsilon)$$

Examples (white board):

$z \sim N(\mu, \sigma)$ $\qquad$ $\varepsilon \sim N(0,1)$

$$z = \mu + \sigma\varepsilon$$

$z \sim N(\mu, \Sigma)$ $\qquad$ $\varepsilon \sim N(0, I)$

$\Sigma = LL^T$

$$z = \mu + \Sigma^{\frac{1}{2}}\varepsilon$$

Reparametrization Trick

Changes of variables:

$$\mathbf{z} = \mathbf{g}(\epsilon, \boldsymbol{\phi}, \mathbf{x}) \qquad \epsilon \sim p(\epsilon)$$

What is it good for?

z is deterministic

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[f(\mathbf{z})] = \nabla_{\boldsymbol{\phi}} \mathbb{E}_{p(\epsilon)}[f(\mathbf{z})]$$

ϵ is not a function of ϕ

$$= \mathbb{E}_{p(\epsilon)}[\nabla_{\boldsymbol{\phi}} f(\mathbf{z})]$$

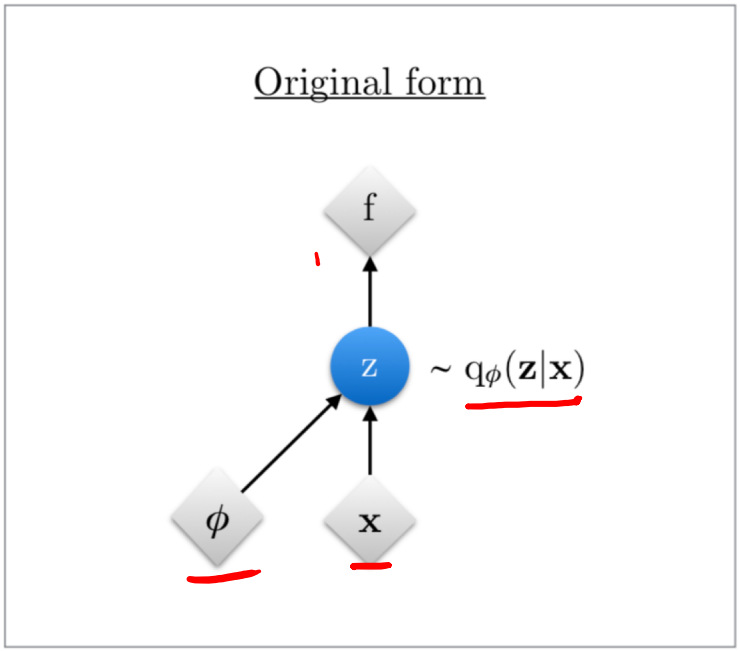
MC estimation with a single sample

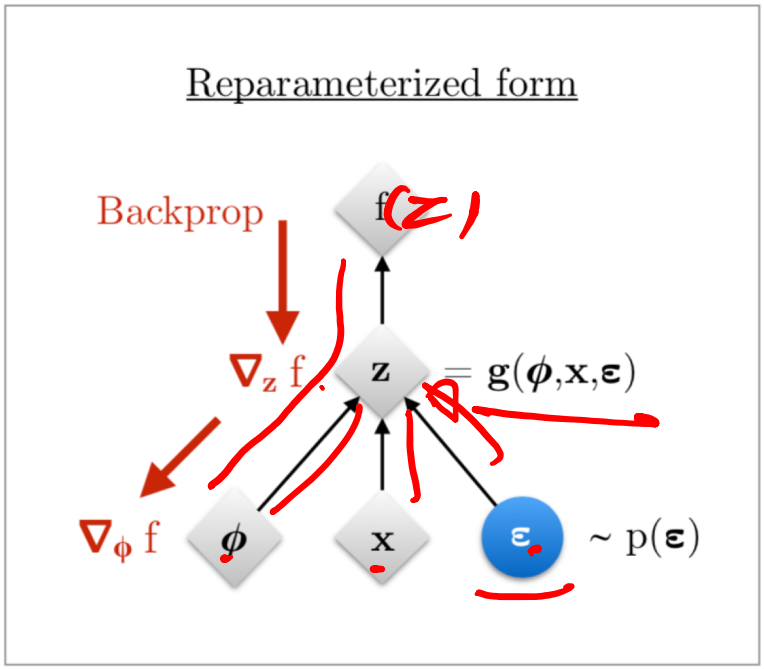
$$\simeq \nabla_{\boldsymbol{\phi}} f(\mathbf{z})$$

Reparametrization Trick

Changes of variables:

$$\mathbf{z} = \mathbf{g}(\epsilon, \boldsymbol{\phi}, \mathbf{x}) \qquad \epsilon \sim p(\epsilon)$$





$p(z) = p(g^{-1}(\varepsilon))\left(\frac{dz}{d\varepsilon}\right)^{-1}$

Back to the Learning…

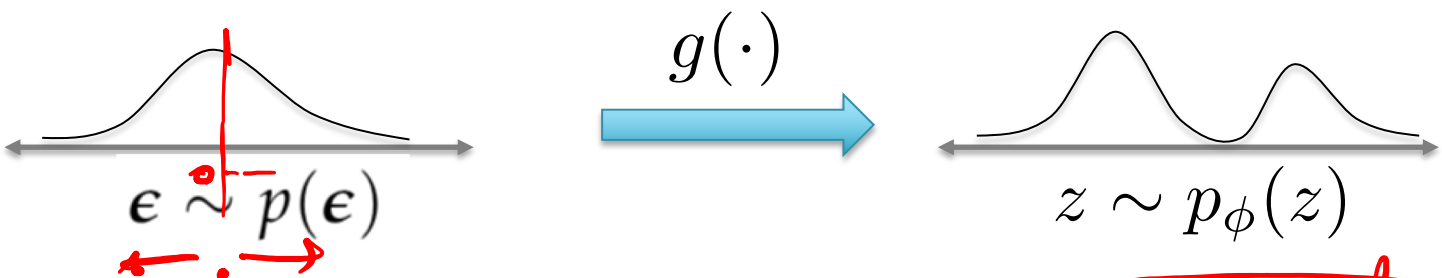
$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right]$$

$$= \mathbb{E}_{p(\epsilon)}\left[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})\right]$$

$\varepsilon \xrightarrow{g} p(z)$

Can we improve the estimation of gradient?
(reduce the variance)

Let's revisit the change of variables:



$$\cancel{\log q_\phi(\mathbf{z}|\mathbf{x})} = \log p(\epsilon) - \log\left|\det\left(\frac{\partial \mathbf{z}}{\partial \epsilon}\right)\right|$$

$x + d$

An Example of the Encoder

$$\epsilon \sim \mathcal{N}(0, \mathbf{I})$$

$$(\mu, \log \sigma) = \text{EncoderNeuralNet}_{\phi}(\mathbf{x})$$

$$\mathbf{z} = \overset{\mu(x)}{\mu} + \overset{\sigma(x)}{\sigma} \odot \epsilon$$

Remember the change of the variable:

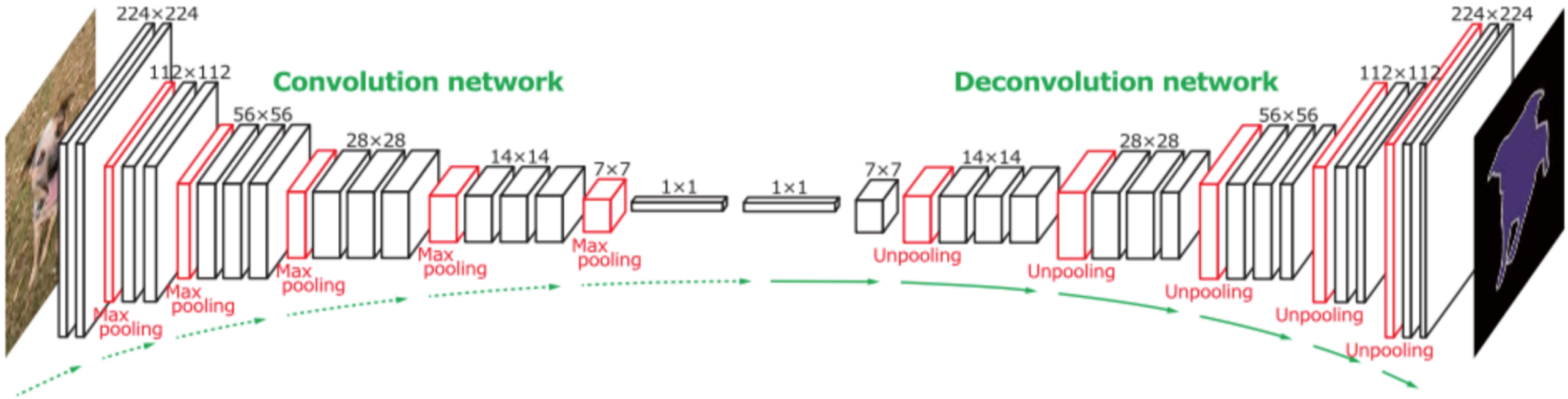
$$\log q_{\phi}(\mathbf{z}|\mathbf{x}) = \underbrace{\log p(\epsilon)}_{\text{Constant}} - \log \left| \det \left(\frac{\partial \mathbf{z}}{\partial \epsilon} \right) \right|$$

$$\sum_i \log \sigma_i$$

Some Results

http://www.dpkingma.com/sgvb_mnist_demo/demo.html

Variations



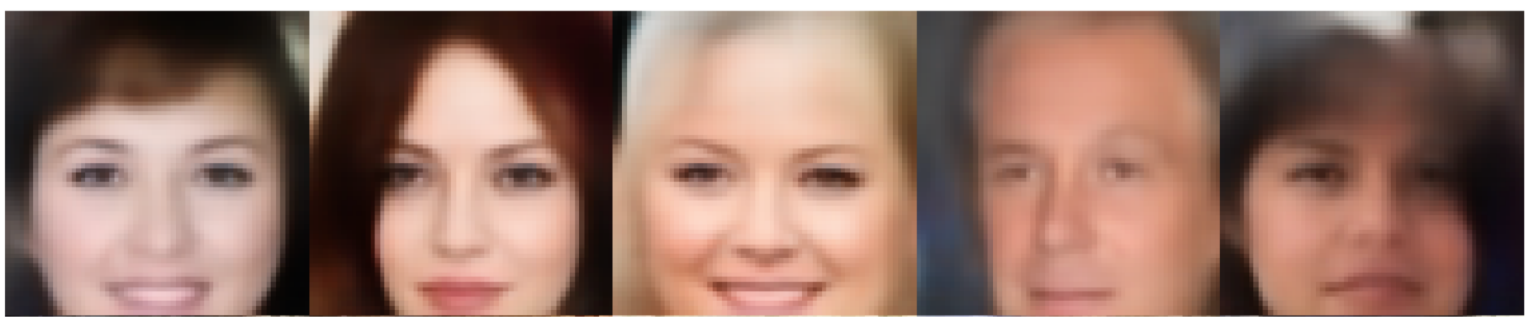
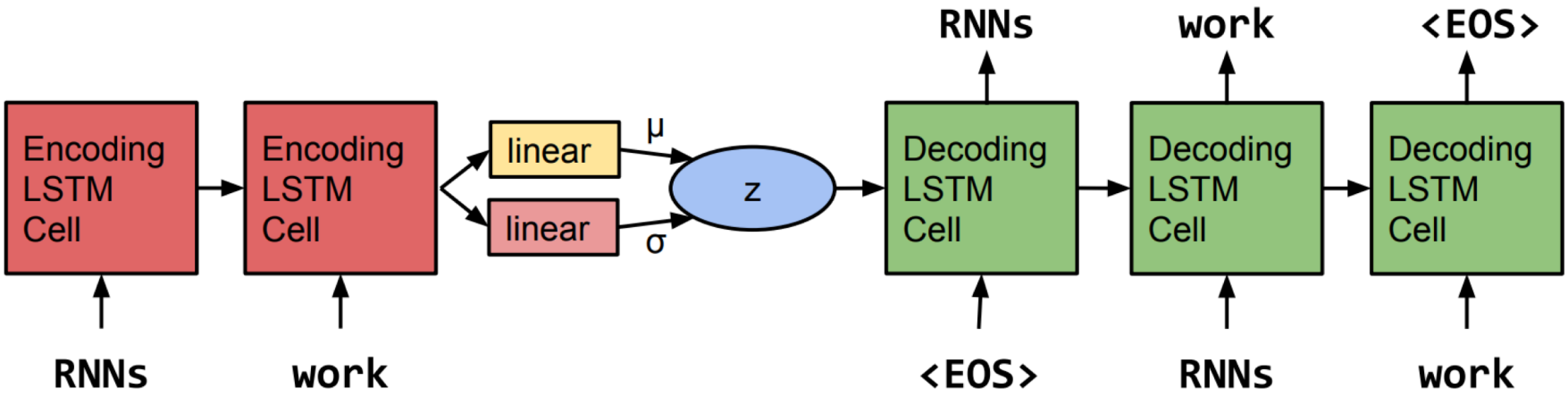


Fig Credit: https://arxiv.org/pdf/1607.07539.pdf
Fig Credit: Learning Deconvolution Network for Semantic Segmentation http://arxiv.org/abs/1505.04366.

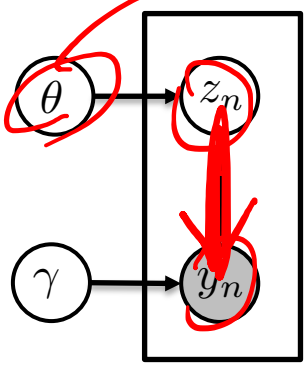
Variations



Generating Sentences from a Continuous Space. S. R. Bowman, et al.c

How to use this technique in a Graphical Model?

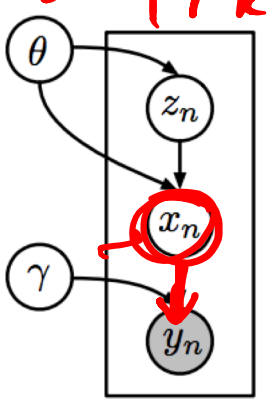
Example: Mixture Model



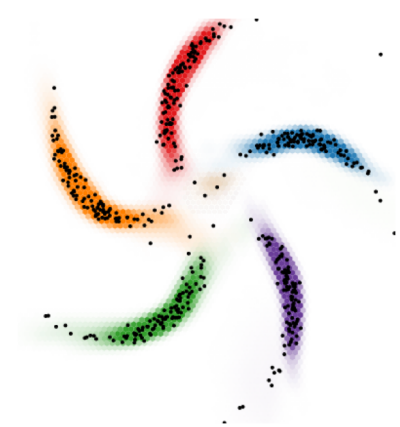
$$\pi \sim \mathrm{Dir}(\alpha),$$
$$(\mu_k, \Sigma_k) \overset{\text{iid}}{\sim} \mathrm{NIW}(\lambda),$$
$$z_n \mid \pi \overset{\text{iid}}{\sim} \pi$$
$$y_n \mid z_n, \{(\mu_k, \Sigma_k)\}_{k=1}^{K} \overset{\text{iid}}{\sim} \mathcal{N}(\mu_{z_n}, \Sigma_{z_n}).$$

(b) GMM

$\gamma = \{\mu_k, \Sigma_k\}$



$$\pi \sim \mathrm{Dir}(\alpha),$$
$$(\mu_k, \Sigma_k) \overset{\text{iid}}{\sim} \mathrm{NIW}(\lambda),$$
$$z_n \mid \pi \overset{\text{iid}}{\sim} \pi$$
$$x_n \overset{\text{iid}}{\sim} \mathcal{N}(\mu^{(z_n)}, \Sigma^{(z_n)}),$$
$$y_n \mid x_n, \gamma \overset{\text{iid}}{\sim} \mathcal{N}(\mu(x_n; \gamma), \Sigma(x_n; \gamma)).$$



(d) GMM SVAE

M. Johnson, Composing graphical models with neural networks for structured representations and fast inference, https://arxiv.org/pdf/1603.06277.pdf

https://www.youtube.com/watch?v=btr1poCYIzw&t=60s

Limitations of VAE

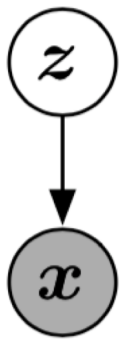
- When applied to image data, it results into blurry images.
- For images, it is sensitive to irrelevant variance, e.g., translations.
- Not applicable to discrete latent variables.
- Limited flexibility: converting the data distribution to fixed, single-mode prior distribution

GAN: GENERATIVE ADVERSARIAL NETWORK

GANs

- Introduced by Goodfellow et al., 2014
- Assumes implicit generative model
- Asymptotically consistent (unlike variational methods)
- No Markov chains needed
- Often regarded as producing the best samples (but,… no good way to quantify it)

Generator Network



$$z \sim \mathcal{N}(0, I)$$

$$\boldsymbol{x} = G(\boldsymbol{z}; \boldsymbol{\theta}^{(G)})$$

- Maps noise variable to the data space
- Must be differentiable but no invertibility is required

1-D Example

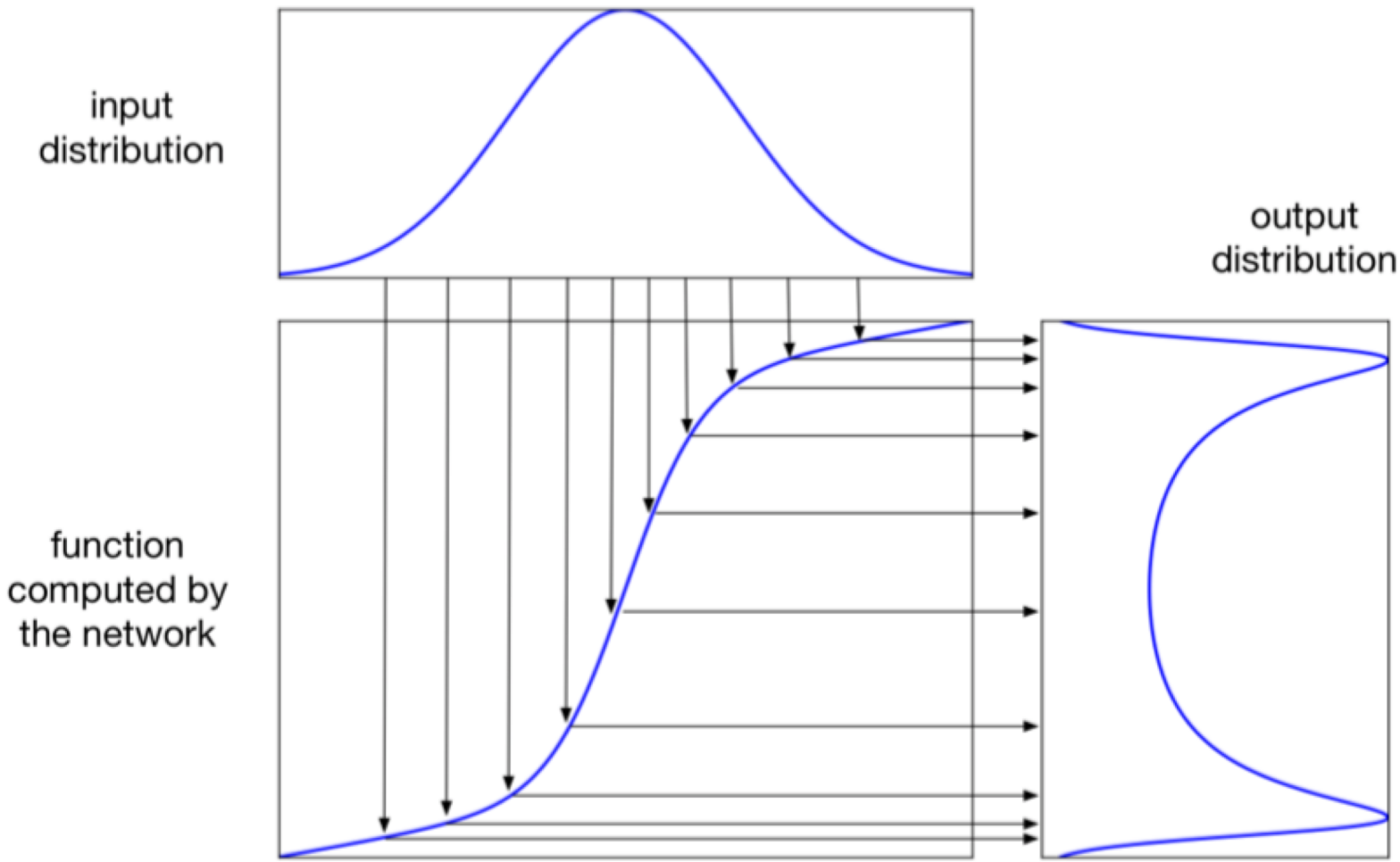


Figure: Roger Grosse

Learning

- Train D to discriminate between training examples and generated samples
- Train G to fool the discriminator

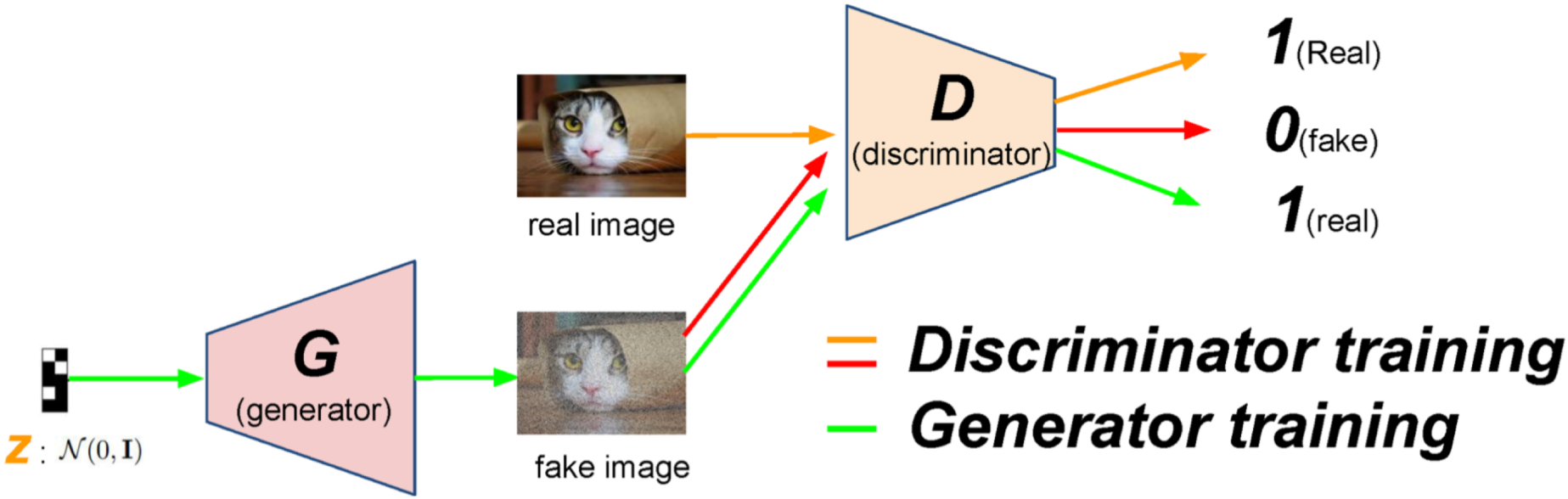


Figure courtesy: Kim's slides

Minimax Game

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Fixing G

$$\int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))dx$$

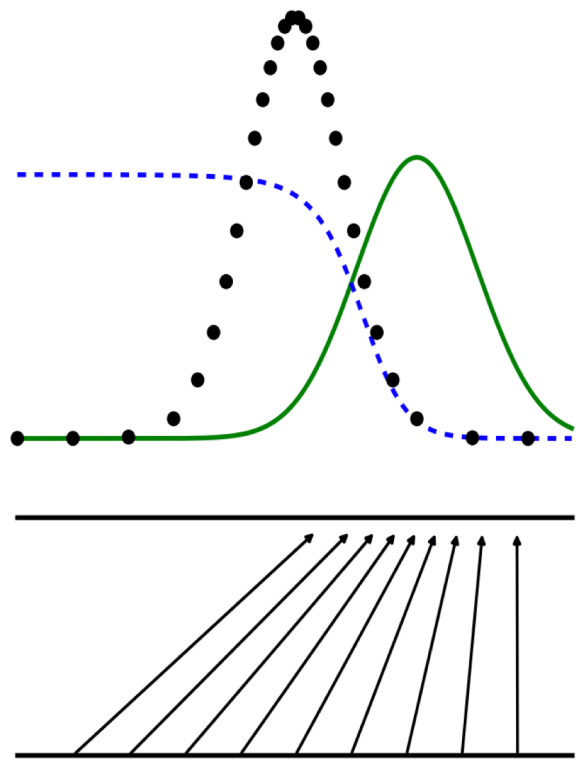
Taking derivative of

$a\log(y) + b\log(1 - y)$

Optimal D: $D(x) = \dfrac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}$

Discriminator Strategy

$$D(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}$$



Minimax Game

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Substituting

$$D(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}$$

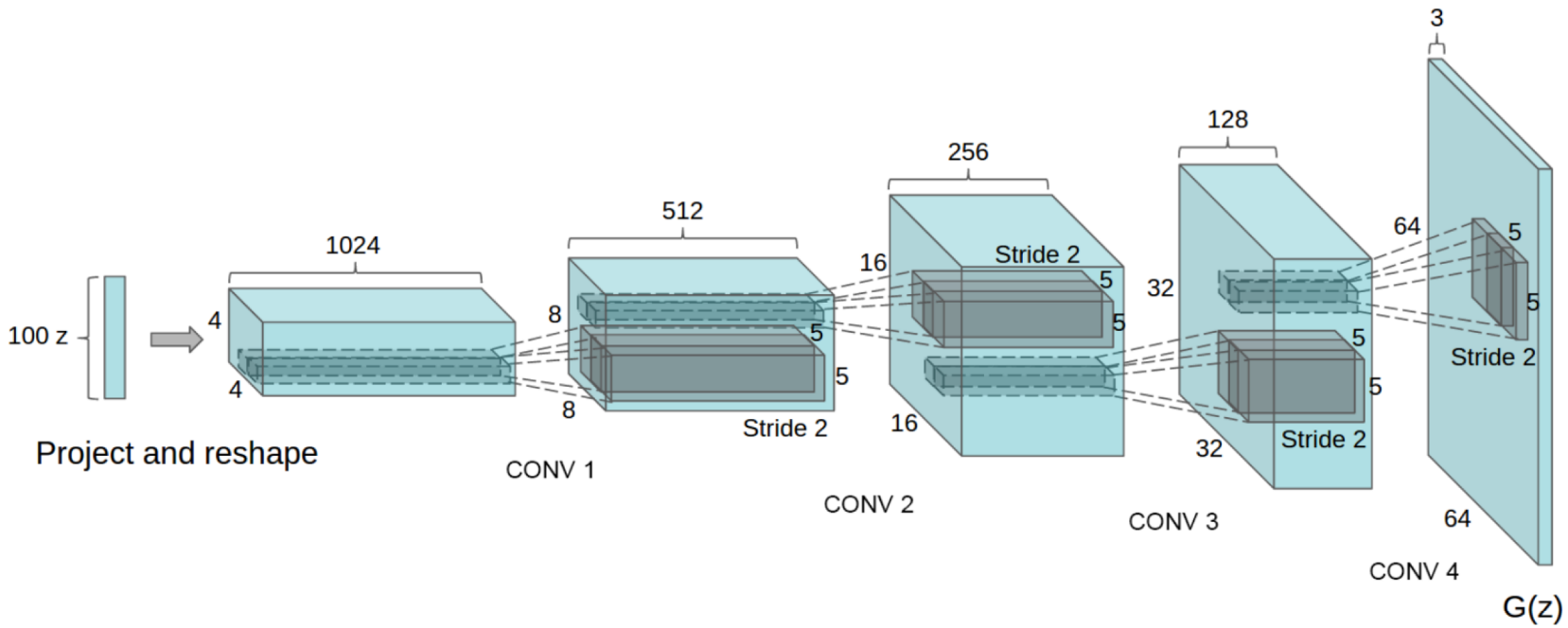
$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right]$$

$$KL\left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_g}{2}\right) \qquad KL\left(p_g \middle\| \frac{p_{\text{data}} + p_g}{2}\right)$$

GANs minimizes the Jensen–Shannon divergence between two distributions.

More Stable Version

- DCGAN: Most "deconvs" are batch normalized



Redford et al., "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks"

DCGANs for LSUN Bedrooms

(Radford et al 2015)

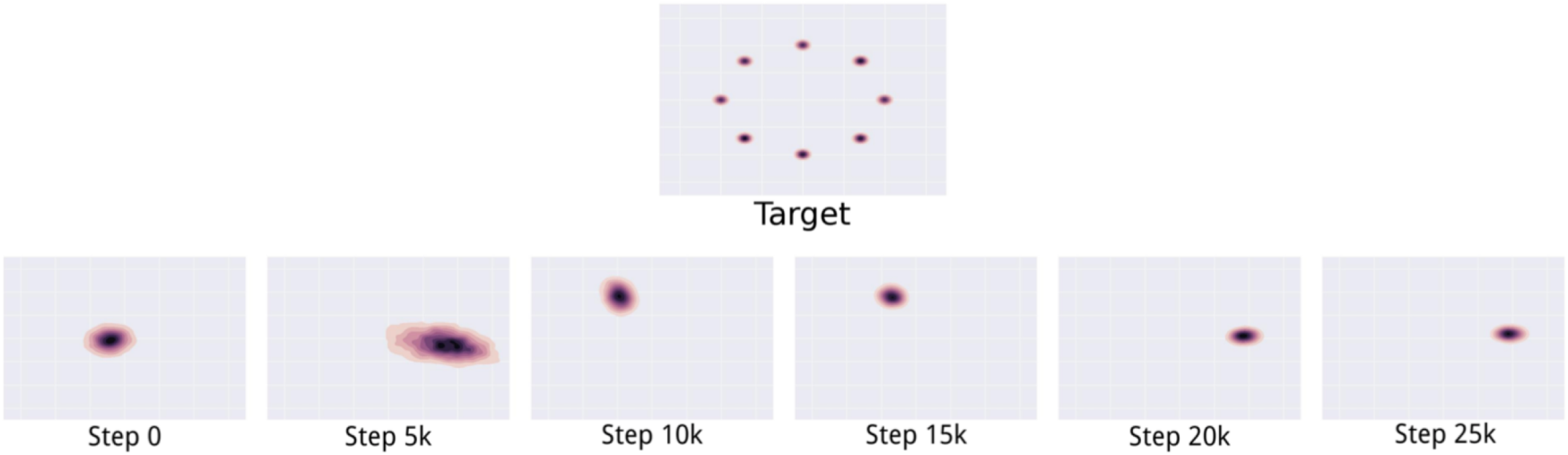
Redford et al., "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks"

Practical Issues

Strong intra-batch correlation

Practical Issues

Mode Collapse



(Metz et al 2016)

(Goodfellow 2016)

Practical Issues

(Goodfellow 2016)