

A Hybrid: Deep Learning and Graphical Models

Kayhan Batmanghelich

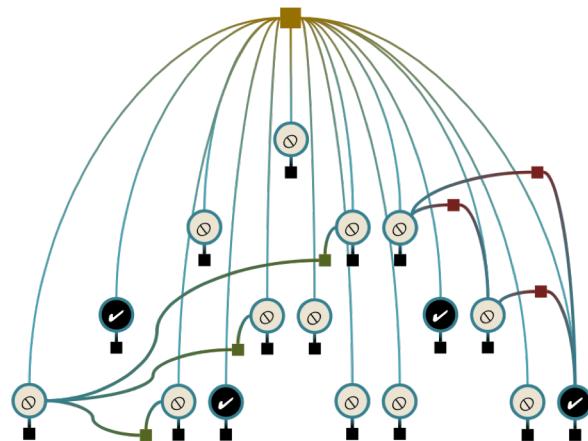
Outlines

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

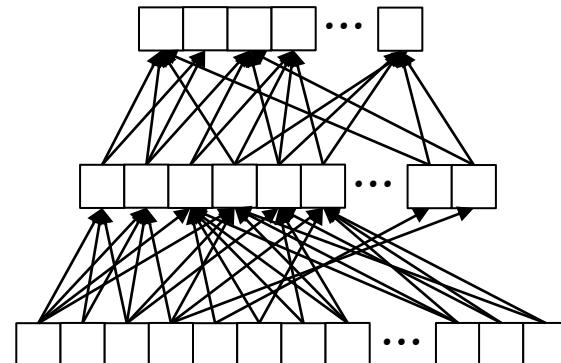
Motivation:

Hybrid Models

Graphical models let you encode domain knowledge



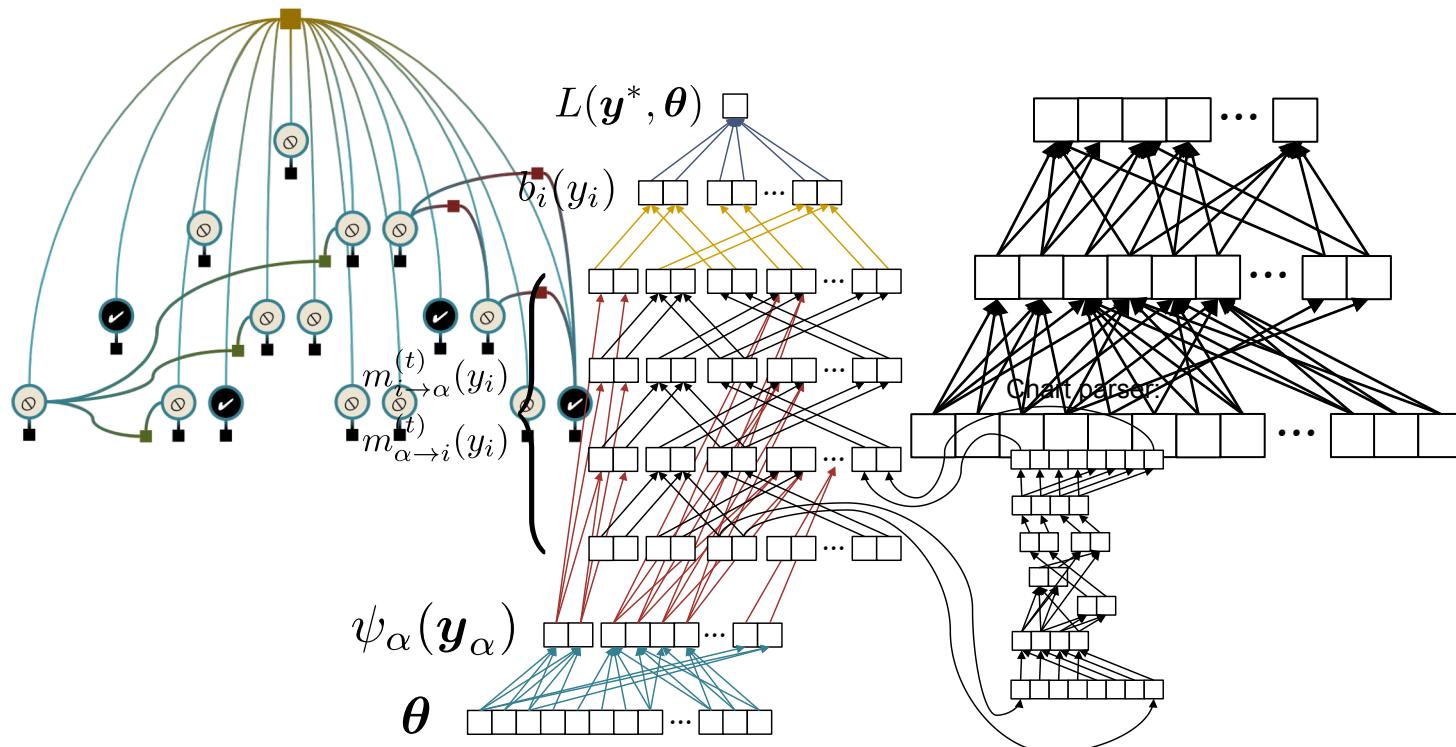
Neural nets are really good at fitting the data discriminatively to make good predictions



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

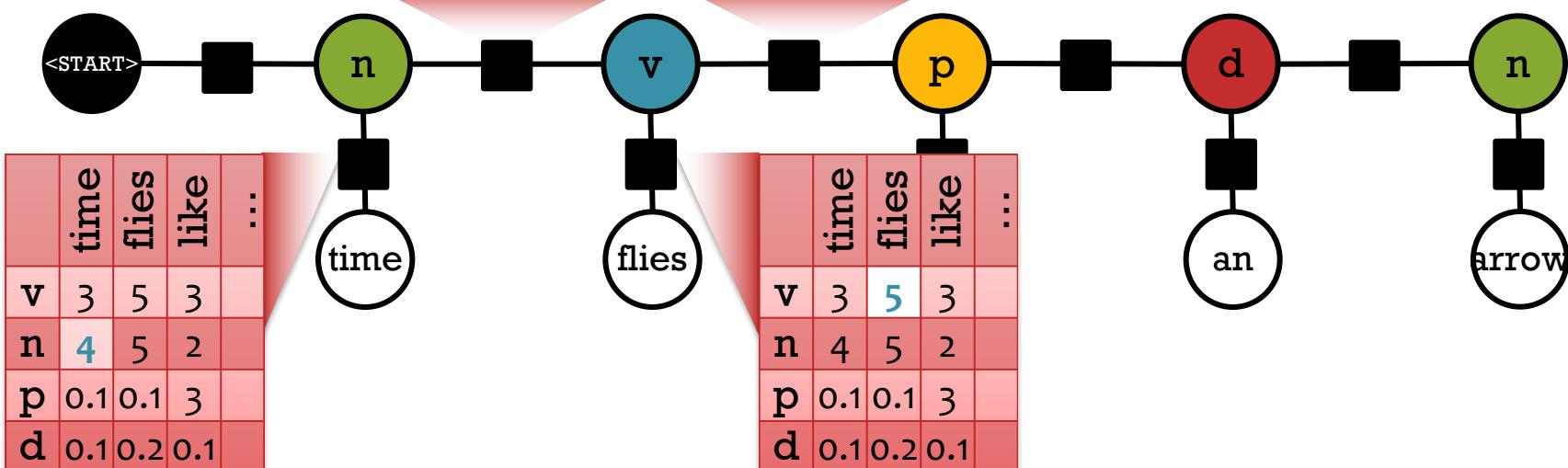
Markov Random Field (MRF)

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

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

	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



Hidden Markov Model

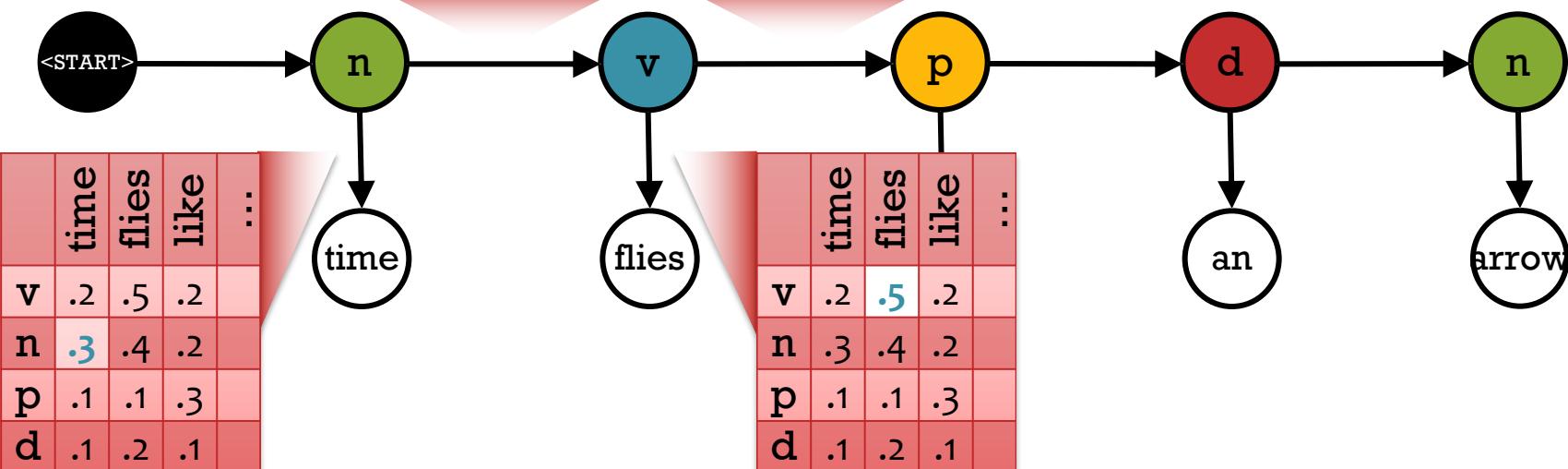
But sometimes we choose to make them probabilities.

Constrain each row of a factor to sum to one. Now $Z = 1$.

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

	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



(Bengio et al., 1992)

Hybrid: NN + HMM

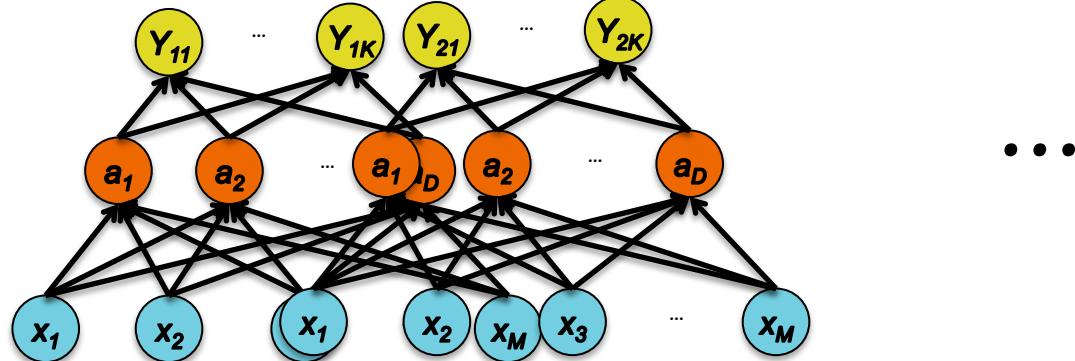
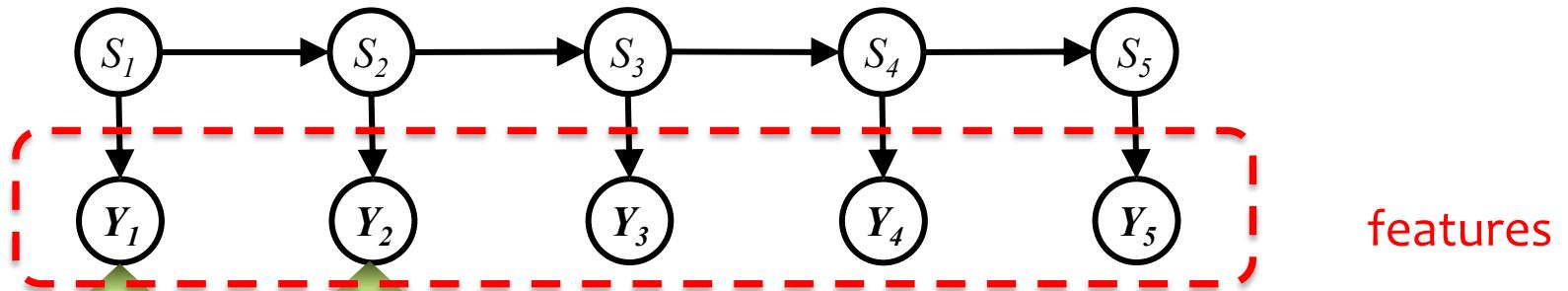
Discrete HMM state: $S_t \in \{/p/, /t/, /k/, /b/, /d/, \dots, /g/\}$

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

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

Gaussian emission:

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



What if we jointly learn the features?

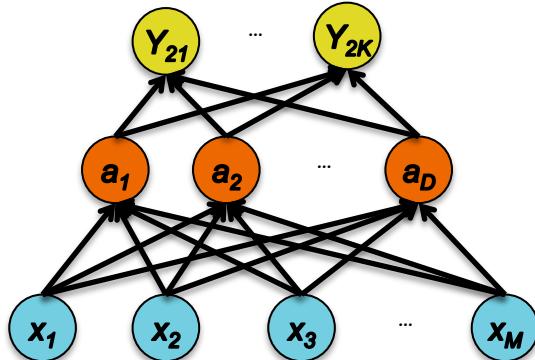
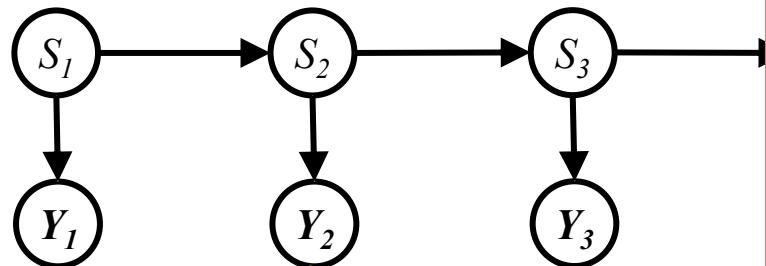
Hybrid: NN + HMM

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

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

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

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

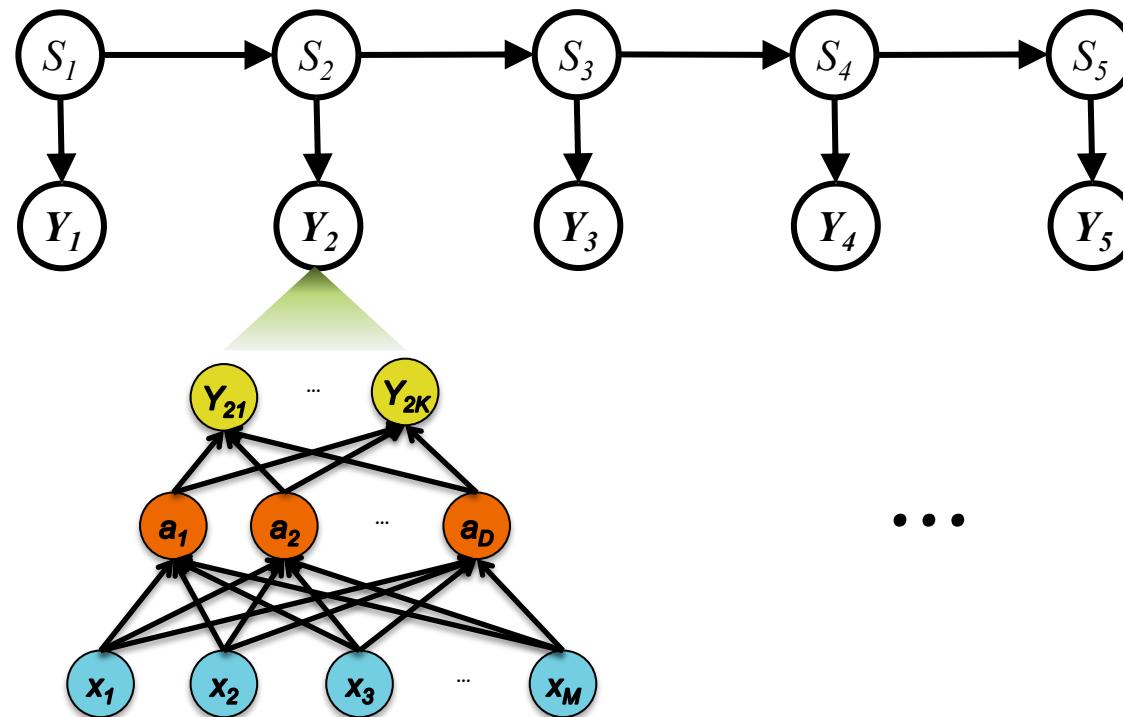


Lots of oddities to this picture:

- **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.

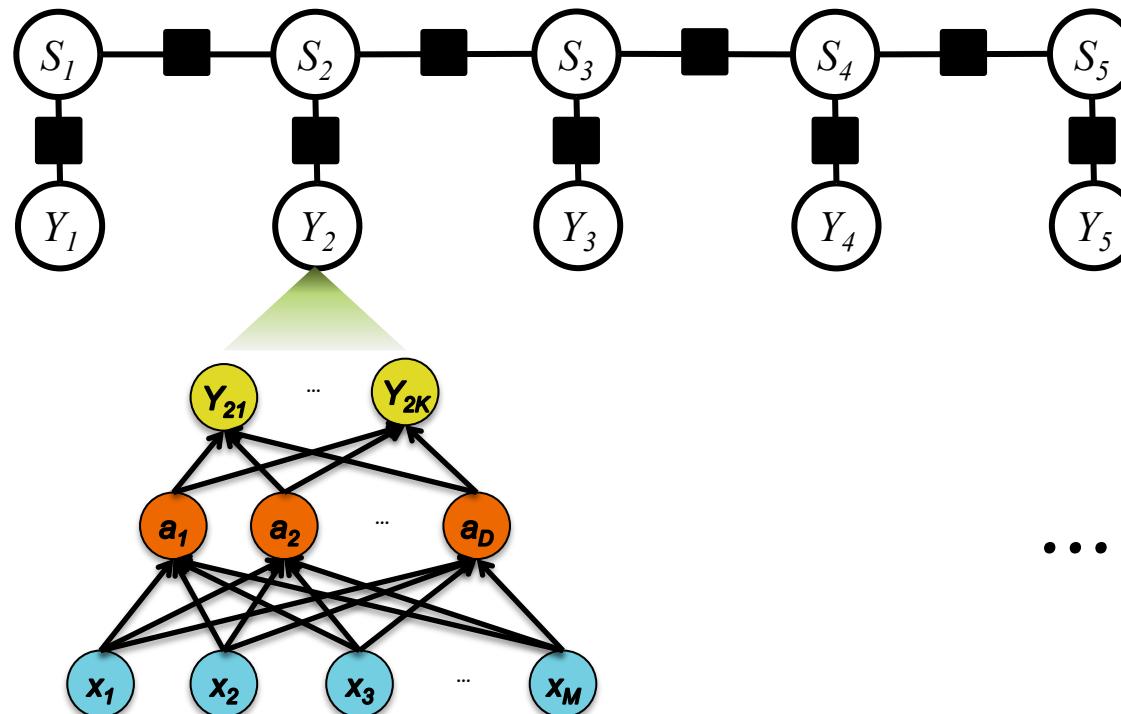
$$\alpha_{i,t} = P(Y_1^t \text{ and } S_t = i \mid \text{model}) = b_{i,t} \sum_j a_{ji} \alpha_{j,t-1}$$

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

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

Log-likelihood: a “feed-forward” objective function.

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



A Recipe for Graphical Models

Decision / Loss Function for Hybrid NN + HMM

1. Given training data:

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

2. Choose each of these

– Decision function

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

– Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{I}$$

How do we compute the gradient?

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

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

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

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

Log-likelihood: a “feed-forward” objective function.

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

Training

Backpropagation

Graphical Model and Log-likelihood

Neural Network

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

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

How to compute these partial derivatives?

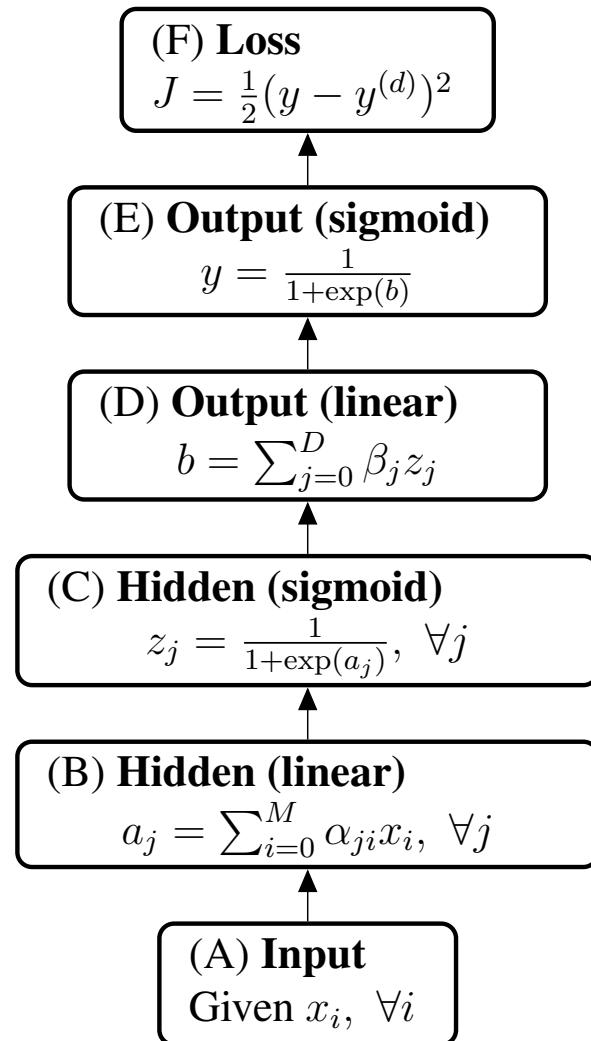
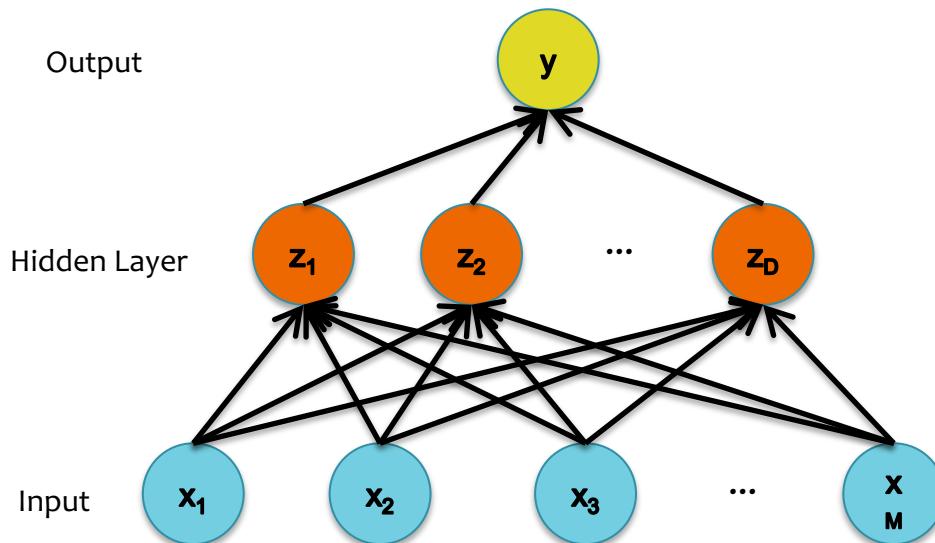
Chain Rule:

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

Training

Backpropagation

What does this picture actually mean?



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: $\nabla \ell(f_{\theta}(x_i), y_i)$

Forward computation

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

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

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

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

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

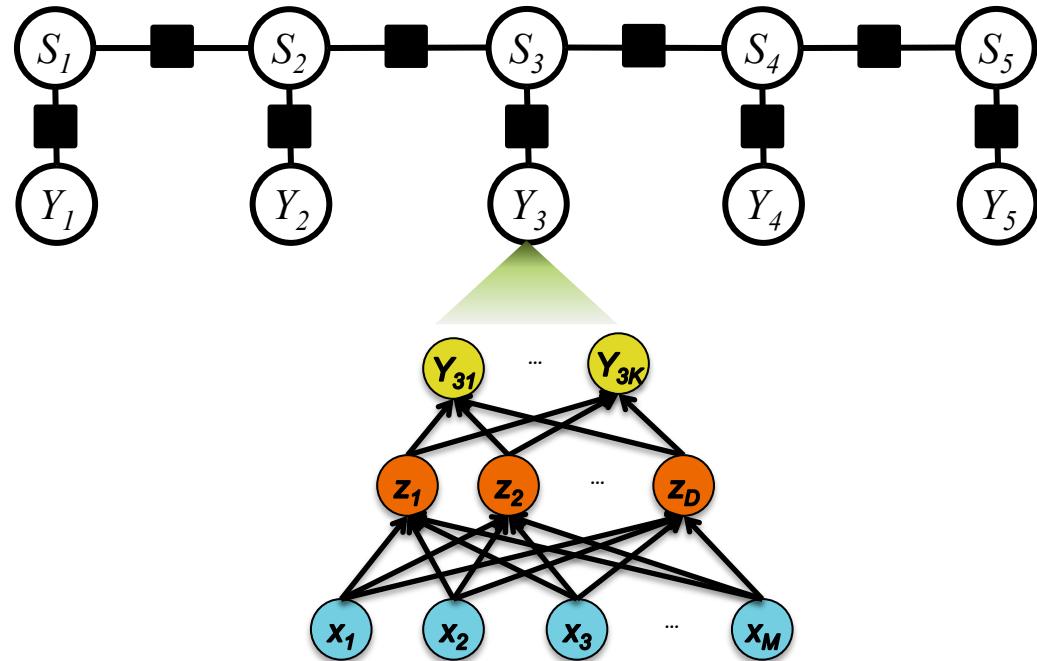
$b_{i,t} = \dots$ (emissions)

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

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

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

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



Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_{\theta}(x_i), y_i)$

Forward computation

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

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

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

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

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

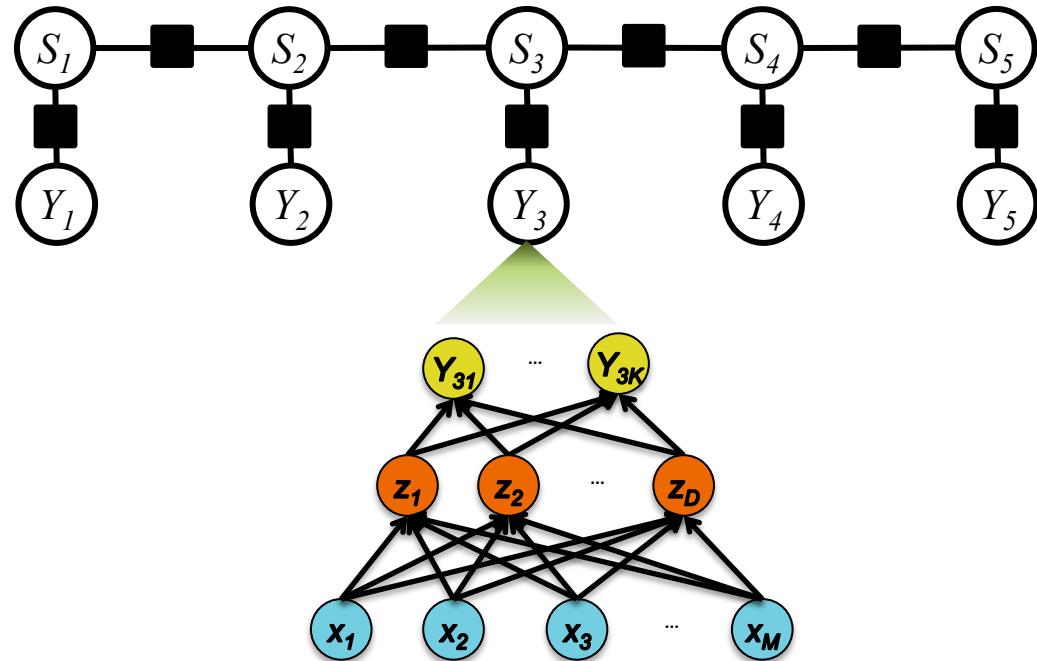
$b_{i,t} = \dots$ (emissions)

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

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

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

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



Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_{\theta}(x_i), y_i)$

Forward computation

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

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

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

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

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

$b_{i,t} = \dots$ (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

$$\begin{aligned} \frac{dJ}{db_{i,t}} &= \frac{\partial \alpha_{F_{model}, T}}{\partial \alpha_{i,t}} \frac{\partial \alpha_{i,t}}{\partial b_{i,t}} = (\sum_j \frac{\partial \alpha_{j,t+1}}{\partial \alpha_{i,t}} \frac{\partial L_{model}}{\partial \alpha_{j,t+1}}) (\sum_j a_{ji} \alpha_{j,t-1}) \\ &= (\sum_j b_{j,t+1} a_{ji} \frac{\partial \alpha_{F_{model}, T}}{\partial \alpha_{j,t+1}}) (\sum_j a_{ji} \alpha_{j,t-1}) = \beta_{i,t} \frac{\alpha_{i,t}}{b_{i,t}} = \frac{\gamma_{i,t}}{b_{i,t}} \end{aligned}$$

Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_{\theta}(x_i), y_i)$

Forward computation

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

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

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

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

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

$b_{i,t} = \dots$ (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

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

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

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

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

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

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

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

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

Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_{\theta}(x_i), y_i)$

Forward computation

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

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

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

$\gamma_{i,t} = \dots$ (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

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

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

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

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

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

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

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

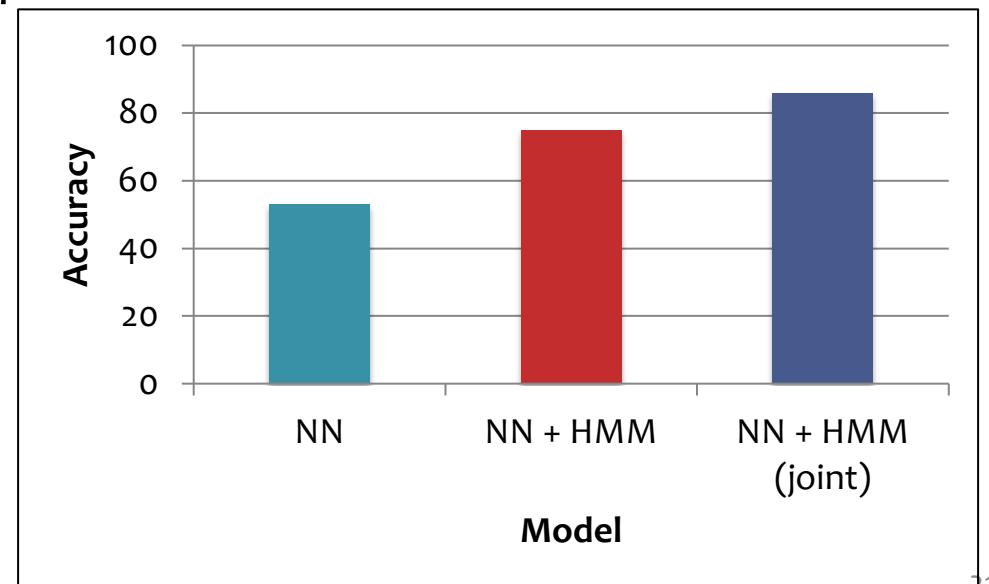
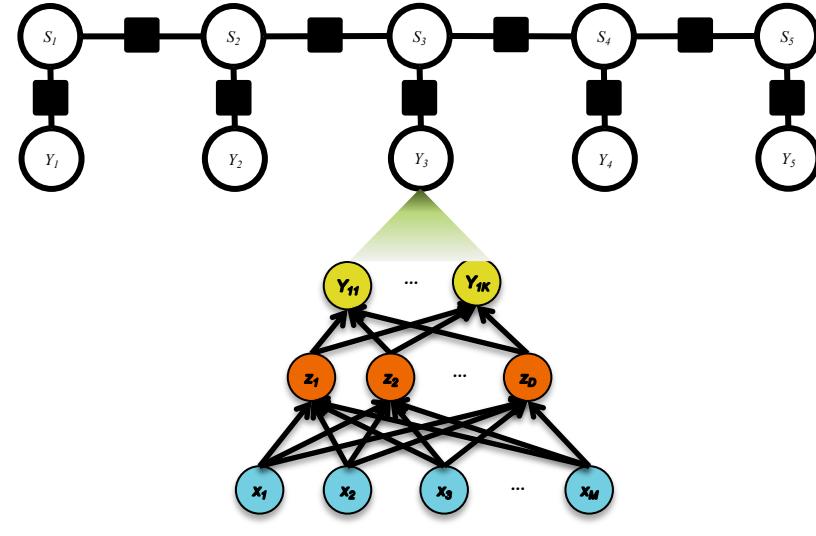
$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$



Hybrid: NN + HMM

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

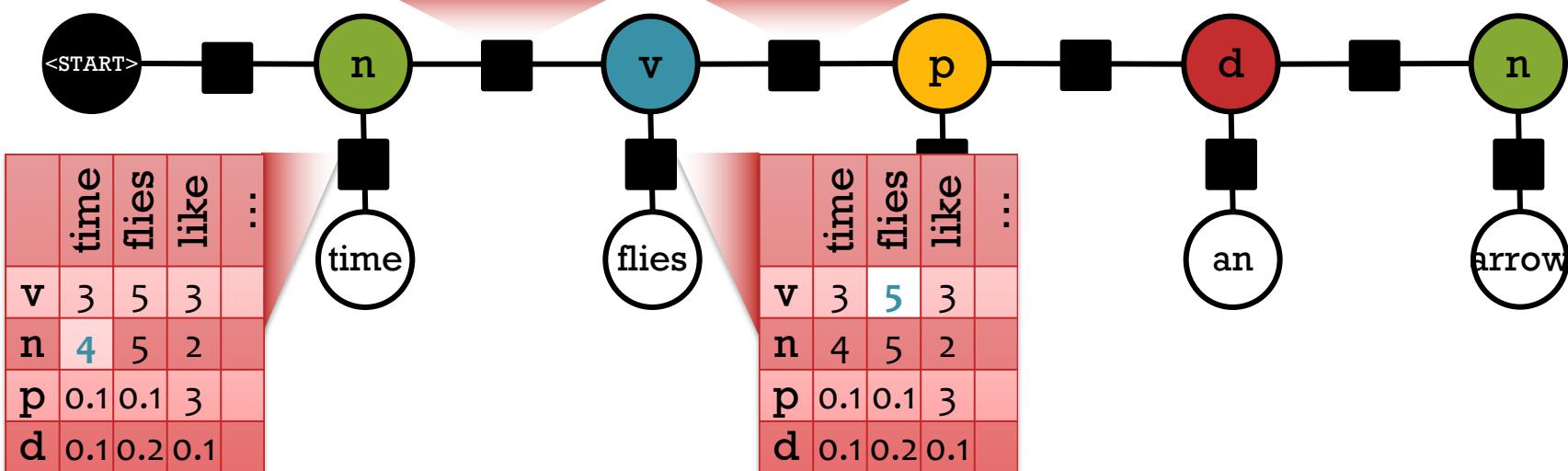
Markov Random Field (MRF)

Joint distribution over tags Y_i and words X_i

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

	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



Recall...

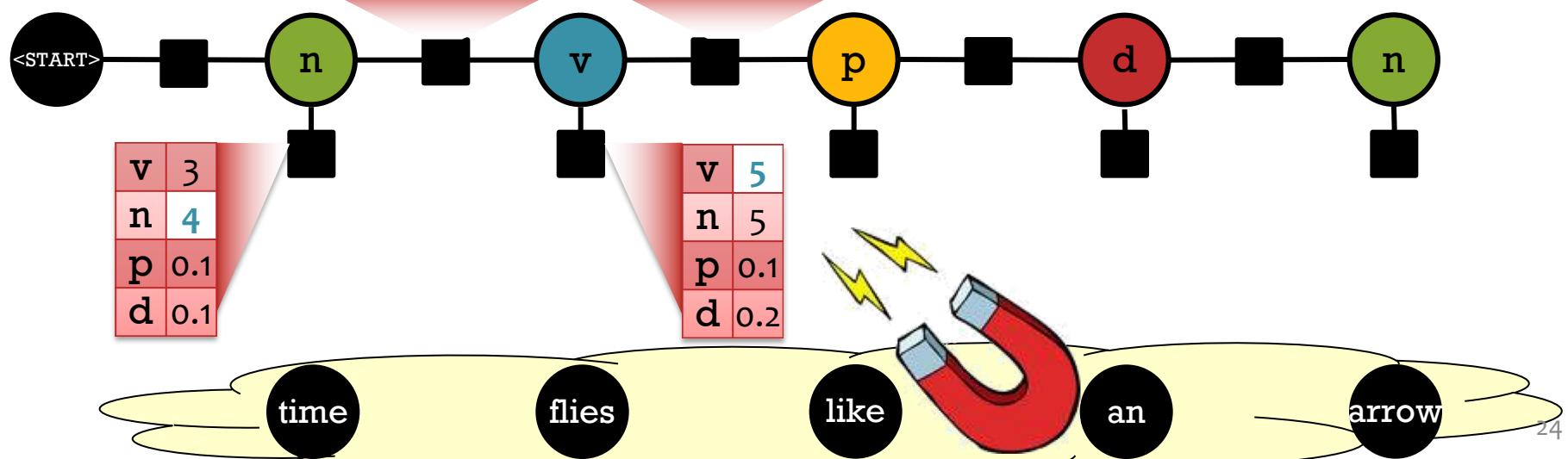
Conditional Random Field (CRF)

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

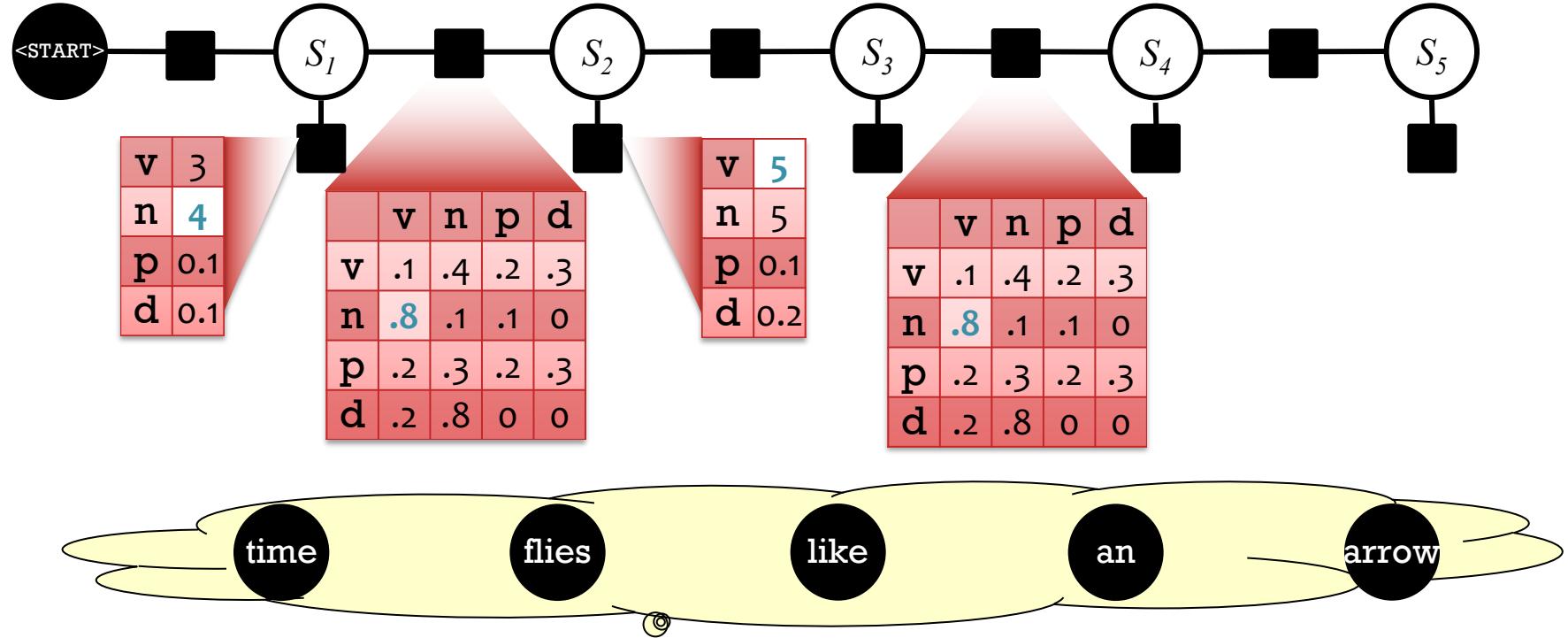
$$p(n, v, p, d, n \mid \text{time, flies, like, an, arrow}) \frac{1}{Z} (4 * 8 * 5 * 3 * \dots)$$

	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



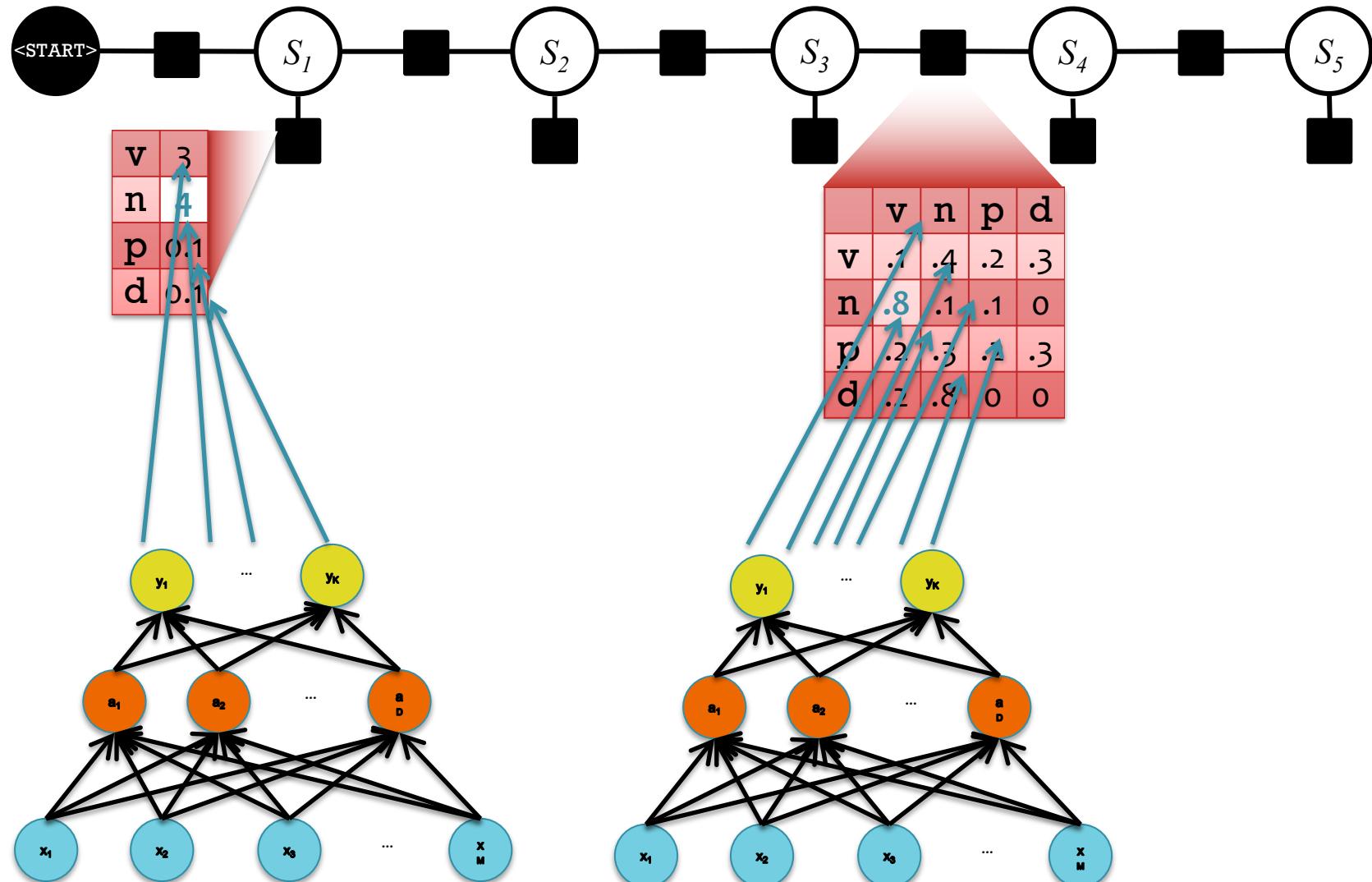
Hybrid: Neural Net + CRF



- 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

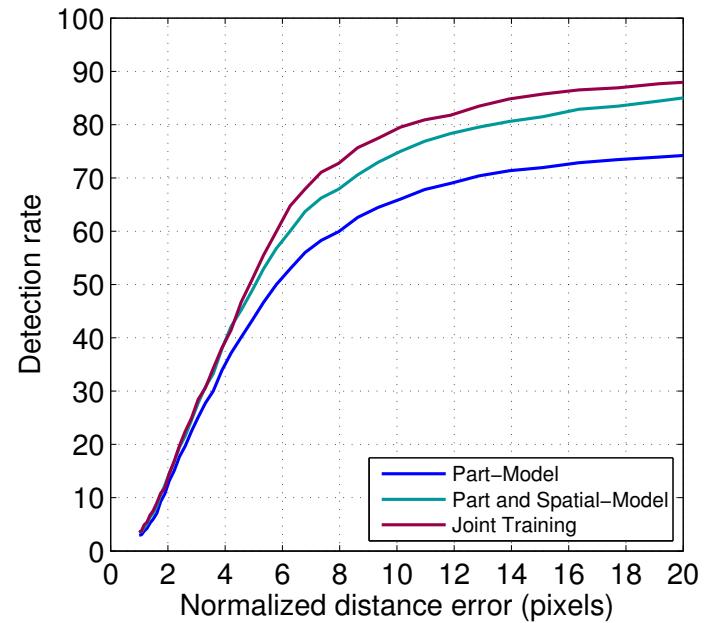
Forward computation



Hybrid: CNN + MRF

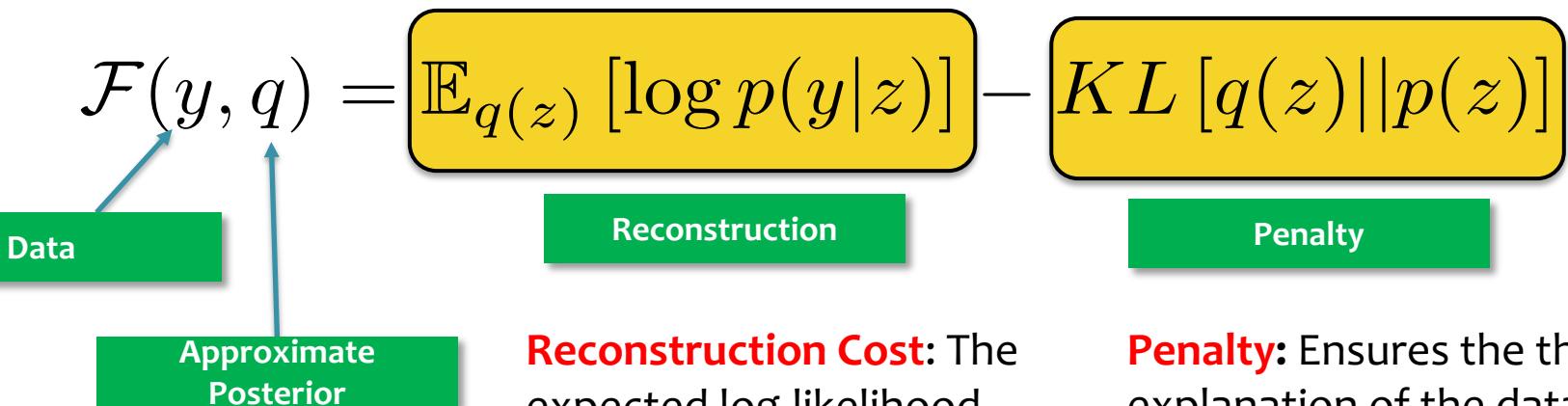
Experimental Setup:

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



VAE: VARIATIONAL AUTO- ENCODER

Recap: Interpreting the Lower Bound (ELBO)

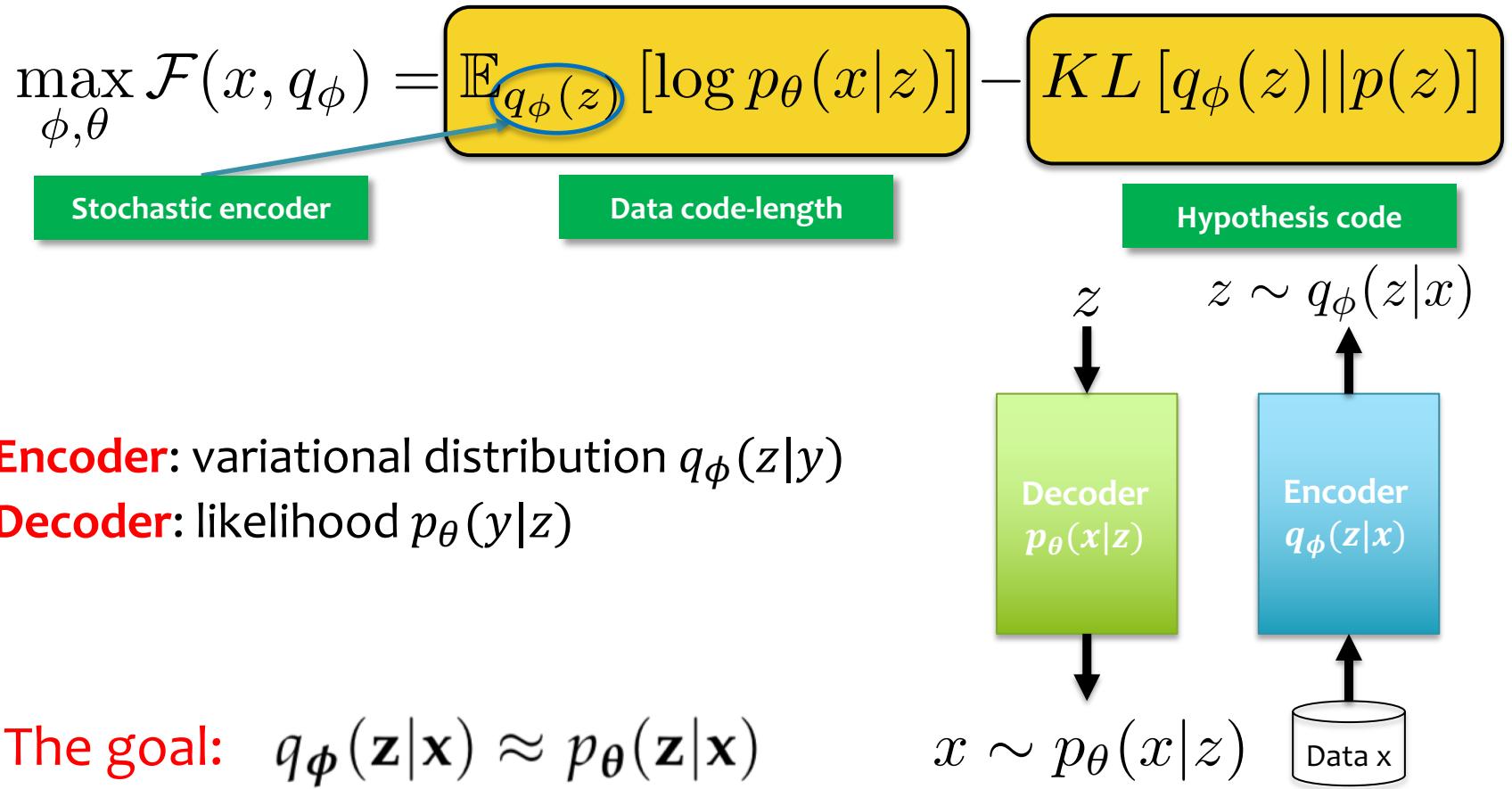


Approximate posterior distribution $q(z)$: Best match to true posterior $p(z|y)$, one of the unknown inferential quantities of interest to us.

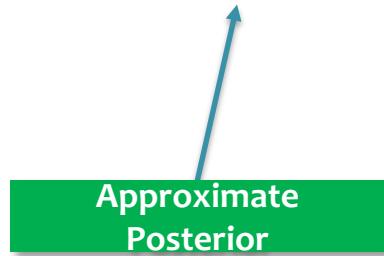
Reconstruction Cost: The expected log-likelihood measure how well samples from $q(z)$ are able to explain the data y .

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

Recap: Decoder Encoder View



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



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

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

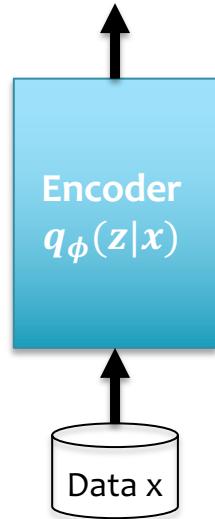


Amortization
(sharing parameters)

Example:

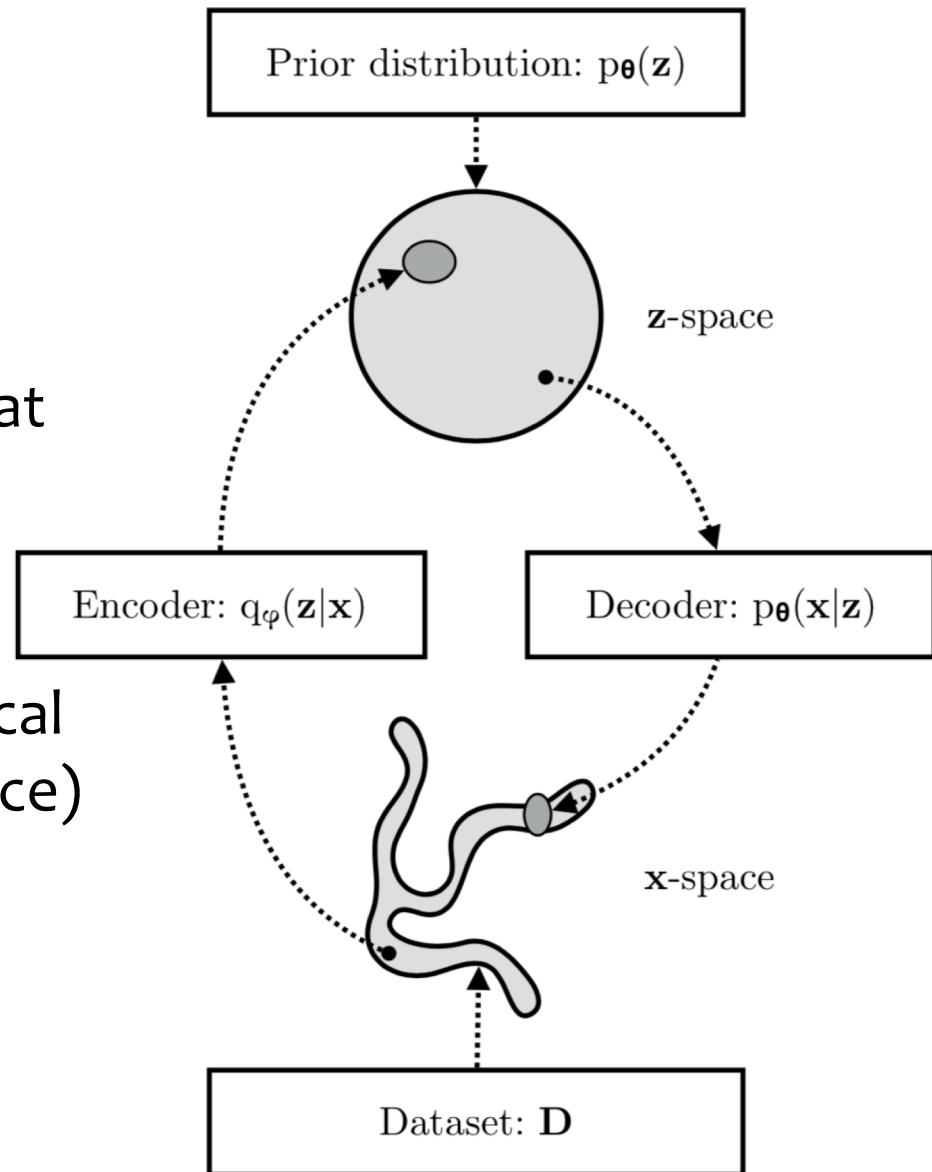
$$(\mu, \log \sigma) = \text{EncoderNeuralNet}_\phi(\mathbf{x})$$

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



Mapping the Distributions

- 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.



Learning

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

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

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

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

Reparametrization Trick

Changes of variables:

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

Examples (white board):

Reparametrization Trick

Changes of variables:

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

What is it good for?

\mathbf{z} is deterministic



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

ϵ is not a function of ϕ

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

MC estimation with
a single sample

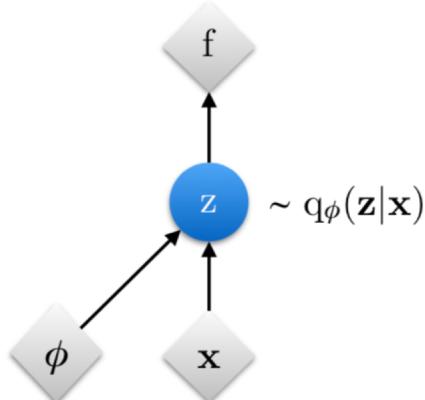
$$\simeq \nabla_{\phi} f(\mathbf{z})$$

Reparametrization Trick

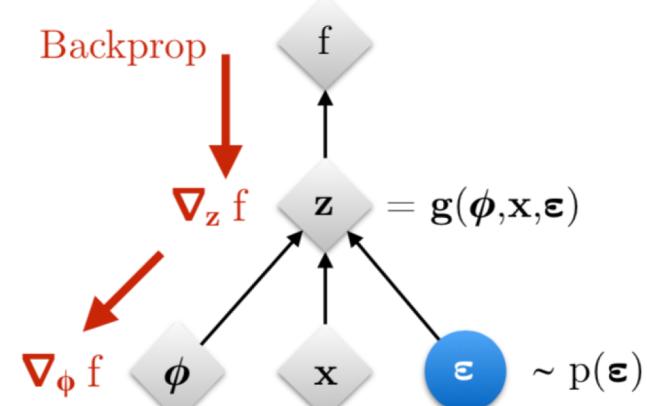
Changes of variables:

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

Original form



Reparameterized form



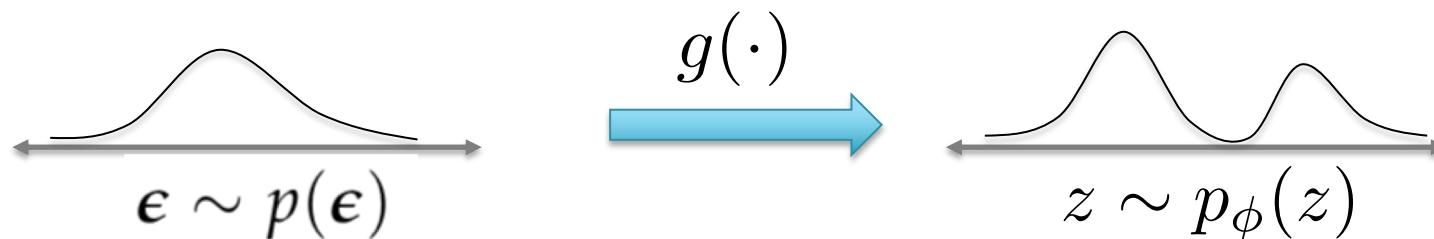
Back to the Learning...

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

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

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

Let's revisit the change of variables:



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

An Example of the Encoder

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

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

$$\mathbf{z} = \mu + \sigma \odot \epsilon$$

Remember the change of the variable:

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

Some Results

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

Variations

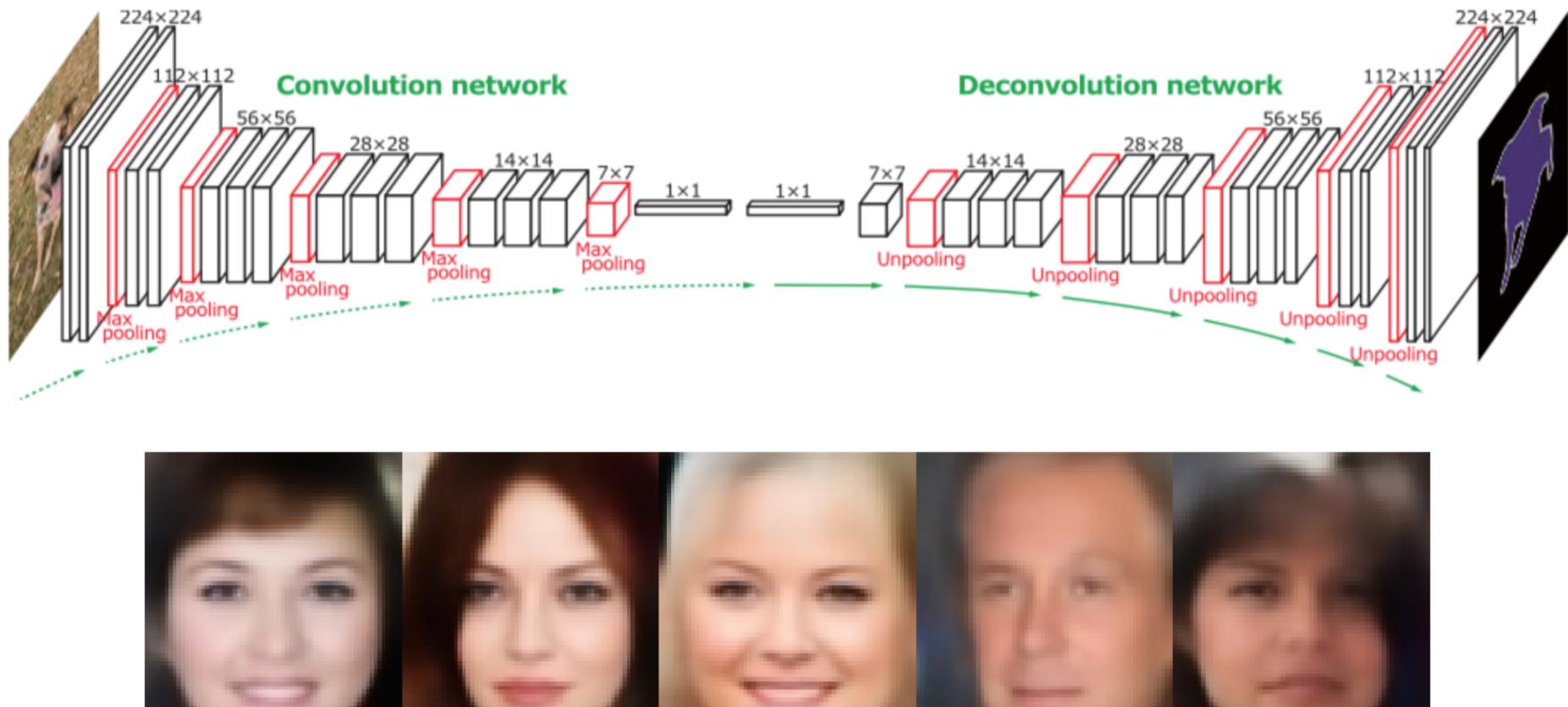
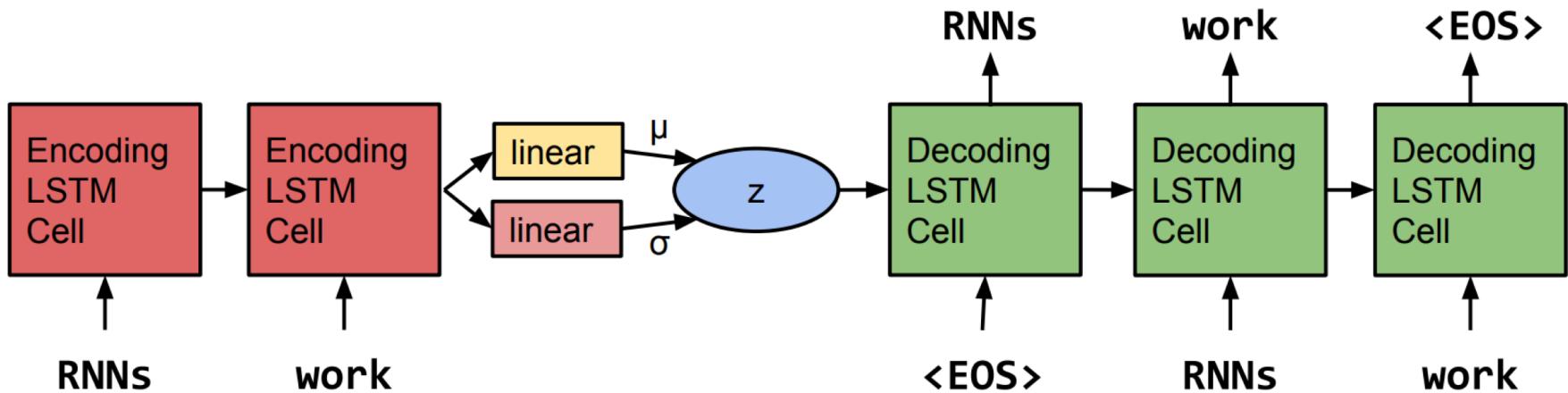


Fig Credit: <https://arxiv.org/pdf/1607.07539.pdf>

Fig Credit: Learning Deconvolution Network for Semantic Segmentation
<http://arxiv.org/abs/1505.04366>.

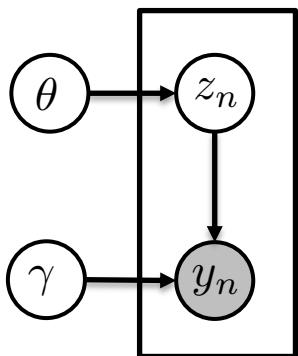
Variations



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

How to use this technique in a Graphical Model?

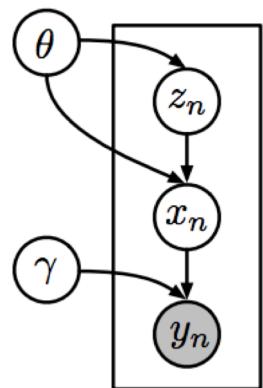
Example: Mixture Model



$$\begin{aligned}\pi &\sim \text{Dir}(\alpha), \\ (\mu_k, \Sigma_k) &\stackrel{\text{iid}}{\sim} \text{NIW}(\lambda), \\ z_n | \pi &\stackrel{\text{iid}}{\sim} \pi \\ y_n | z_n, \{(\mu_k, \Sigma_k)\}_{k=1}^K &\stackrel{\text{iid}}{\sim} \mathcal{N}(\mu_{z_n}, \Sigma_{z_n}).\end{aligned}$$



(b) GMM



$$\begin{aligned}\pi &\sim \text{Dir}(\alpha), \\ (\mu_k, \Sigma_k) &\stackrel{\text{iid}}{\sim} \text{NIW}(\lambda), \\ z_n | \pi &\stackrel{\text{iid}}{\sim} \pi \\ x_n &\stackrel{\text{iid}}{\sim} \mathcal{N}(\mu^{(z_n)}, \Sigma^{(z_n)}), \\ y_n | x_n, \gamma &\stackrel{\text{iid}}{\sim} \mathcal{N}(\mu(x_n; \gamma), \Sigma(x_n; \gamma)).\end{aligned}$$



(d) GMM SVAE

<https://www.youtube.com/watch?v=btr1poCYIzw&t=6os>

Limitations of VAE

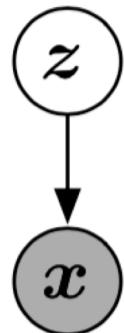
- When applied to image data, it results into blurry images.
- For images, it is sensitive to irrelevant variance, e.g., translations.
- Not applicable to discrete latent variables.
- Limited flexibility: converting the data distribution to fixed, single-mode prior distribution

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

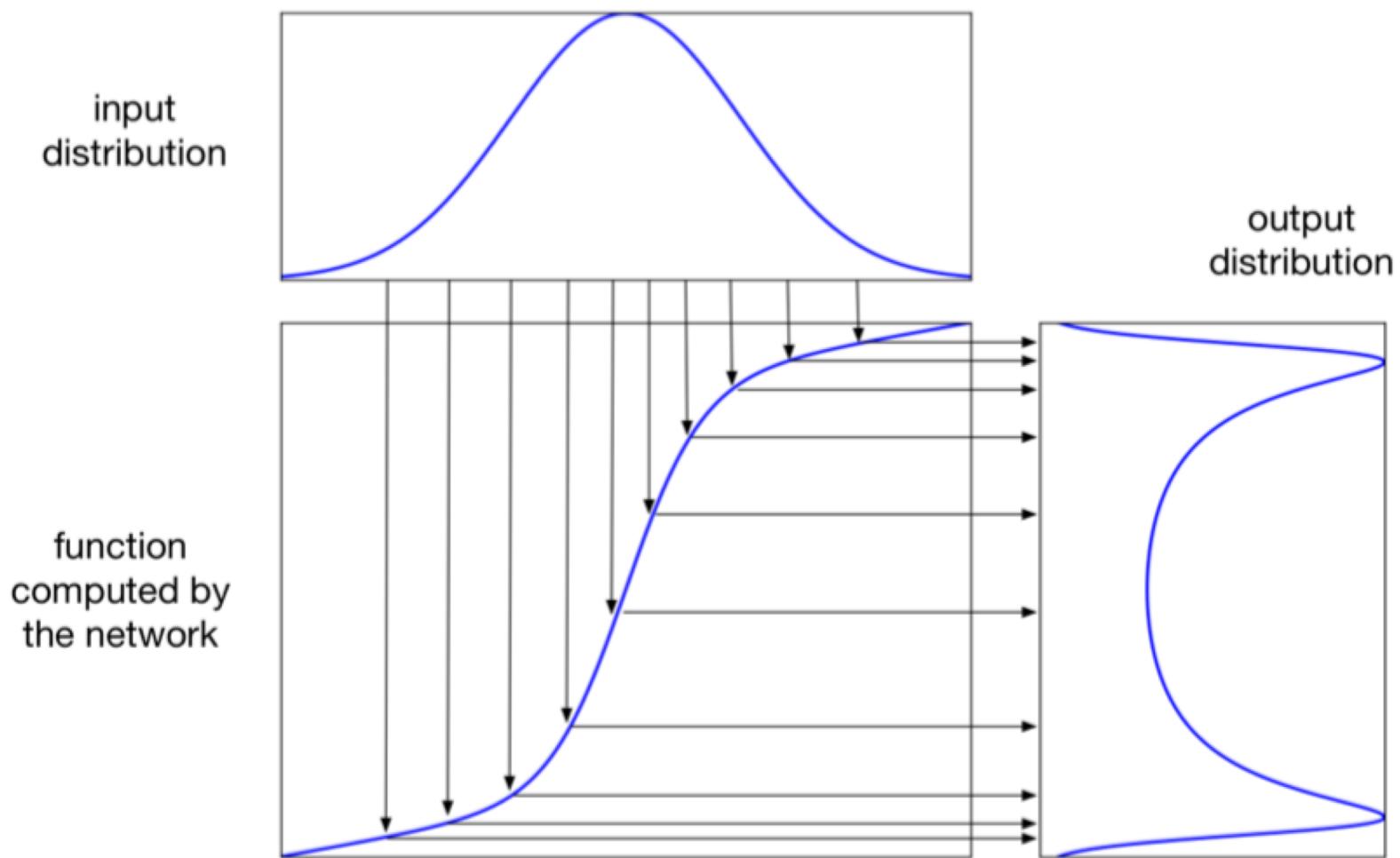


$$z \sim \mathcal{N}(0, I)$$

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

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

1-D Example



Learning

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

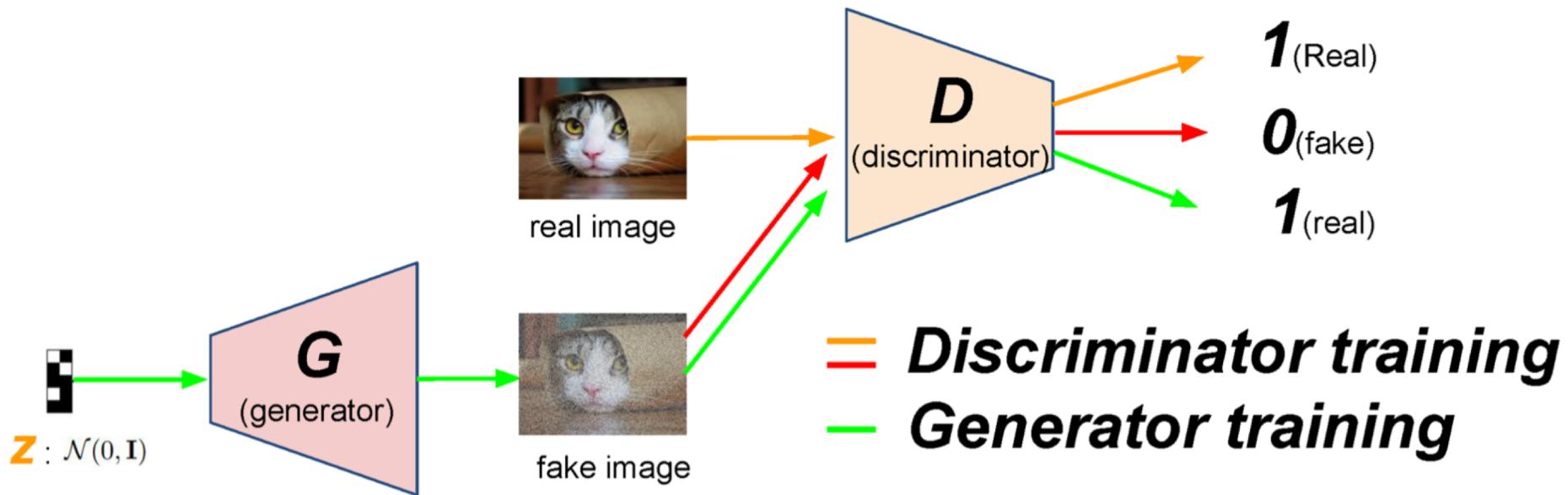


Figure courtesy: Kim's slides

Minimax Game

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

Fixing G

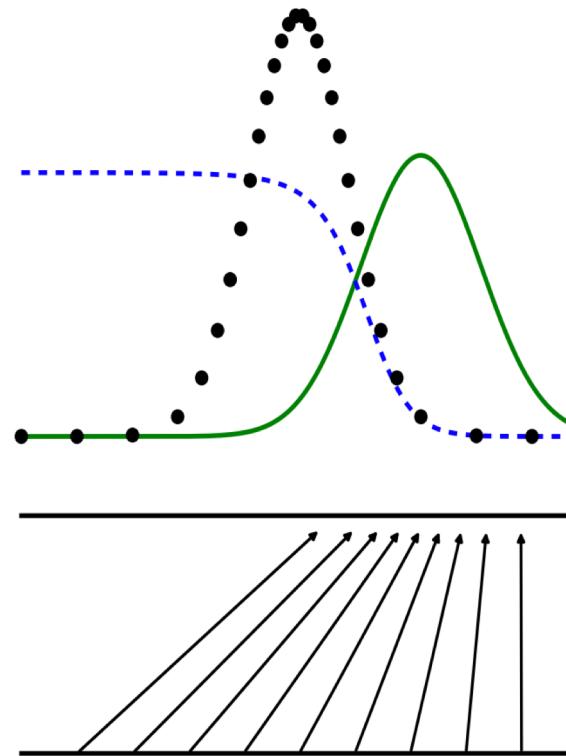
$$\int_x p_{\text{data}}(\mathbf{x}) \log(D(\mathbf{x})) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x})) dx$$

Taking derivative of
 $a \log(y) + b \log(1 - y)$

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

Discriminator Strategy

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



Minimax Game

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

Substituting

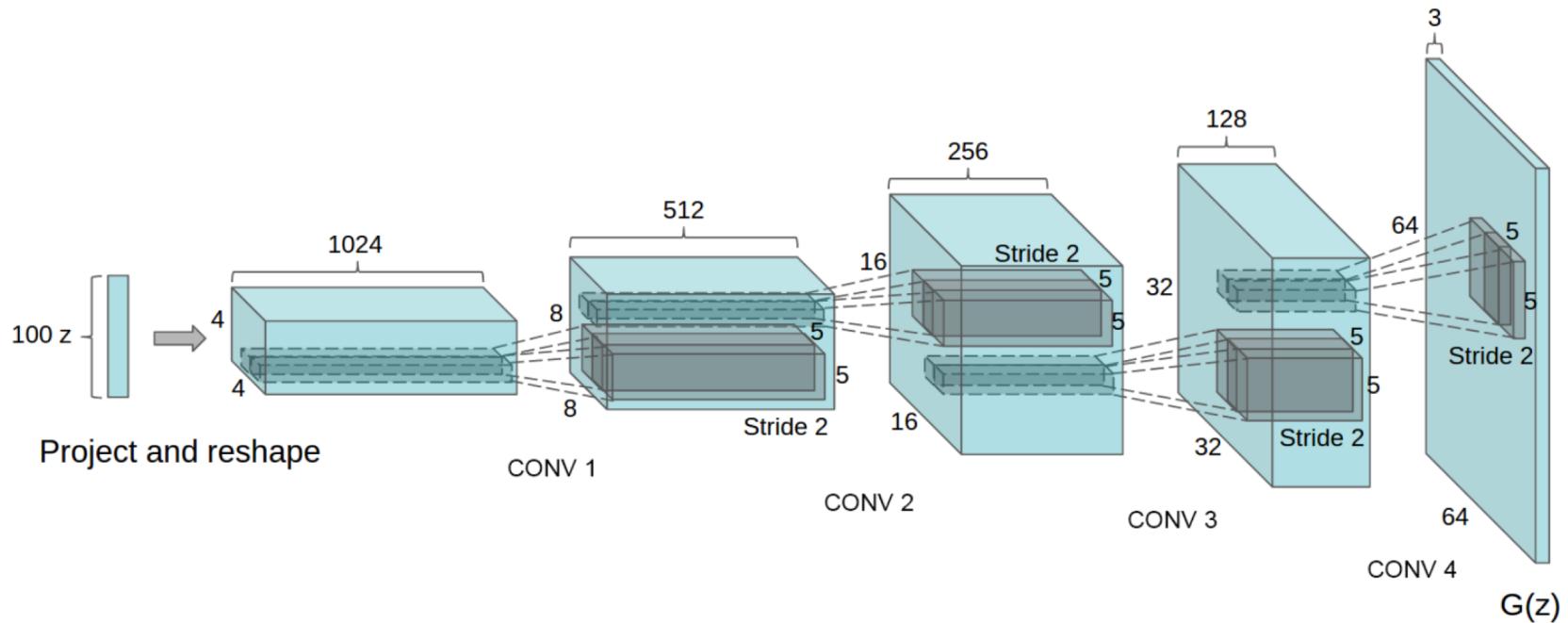
$$D(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}$$

$$\boxed{\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(\mathbf{x})}{P_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} \right]} + \boxed{\mathbb{E}_{\mathbf{x} \sim p_g} \left[\log \frac{p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} \right]}$$
$$KL \left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_g}{2} \right) \qquad \qquad \qquad KL \left(p_g \middle\| \frac{p_{\text{data}} + p_g}{2} \right)$$

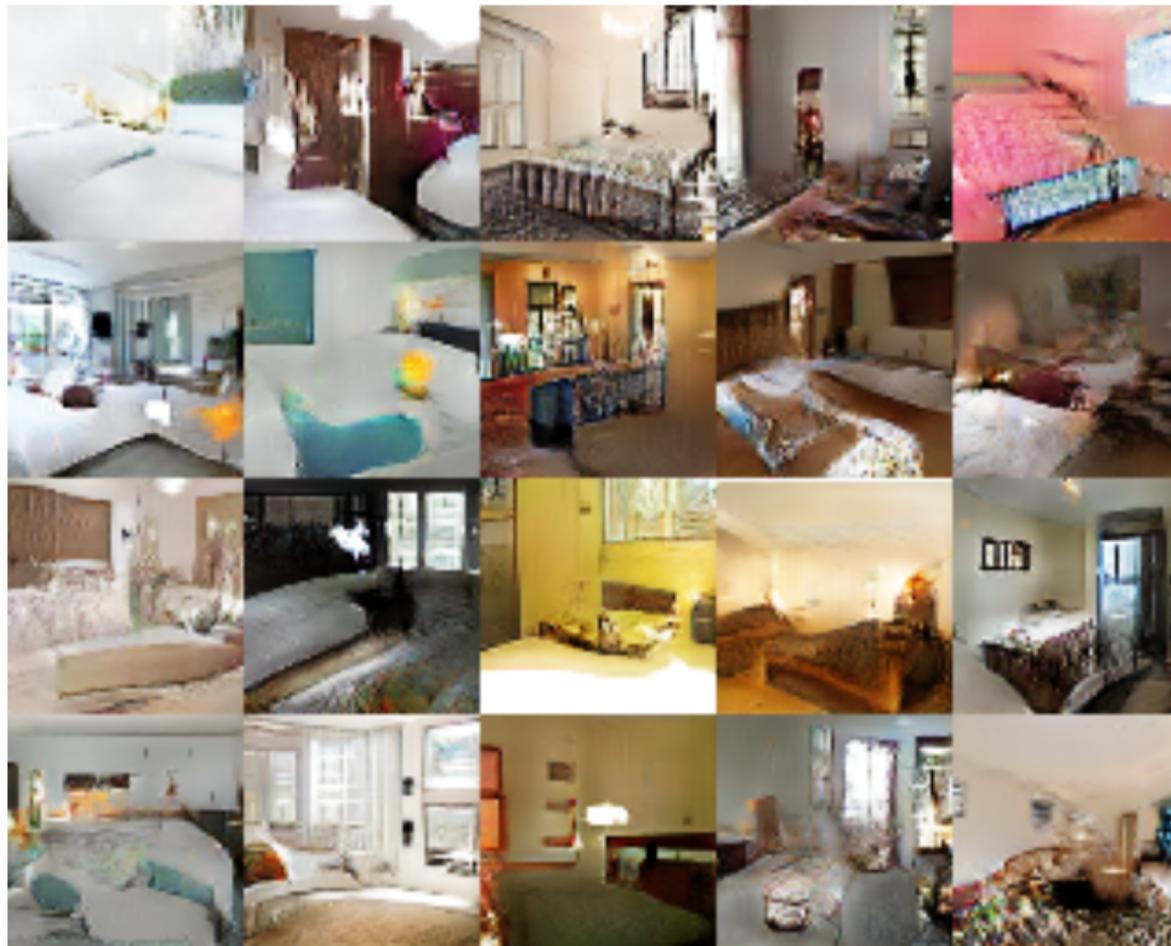
GANs minimizes the Jensen–Shannon divergence between two distributions.

More Stable Version

- DCGAN: Most “deconvs” are batch normalized



DCGANs for LSUN Bedrooms

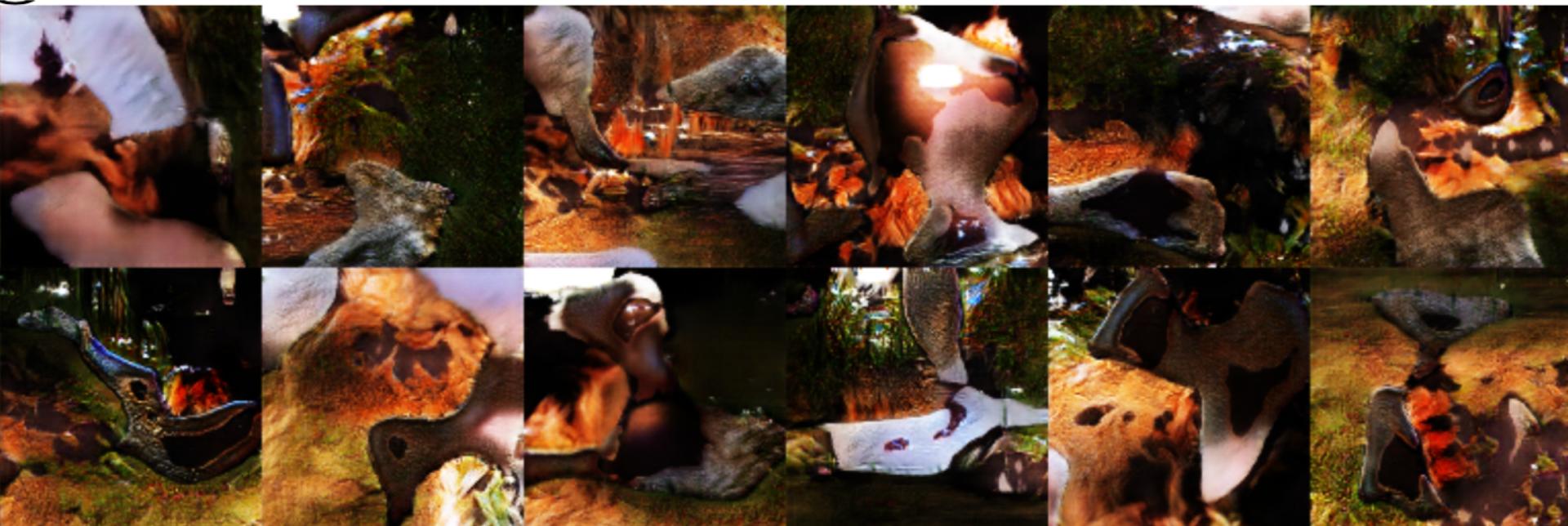


(Radford et al 2015)

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

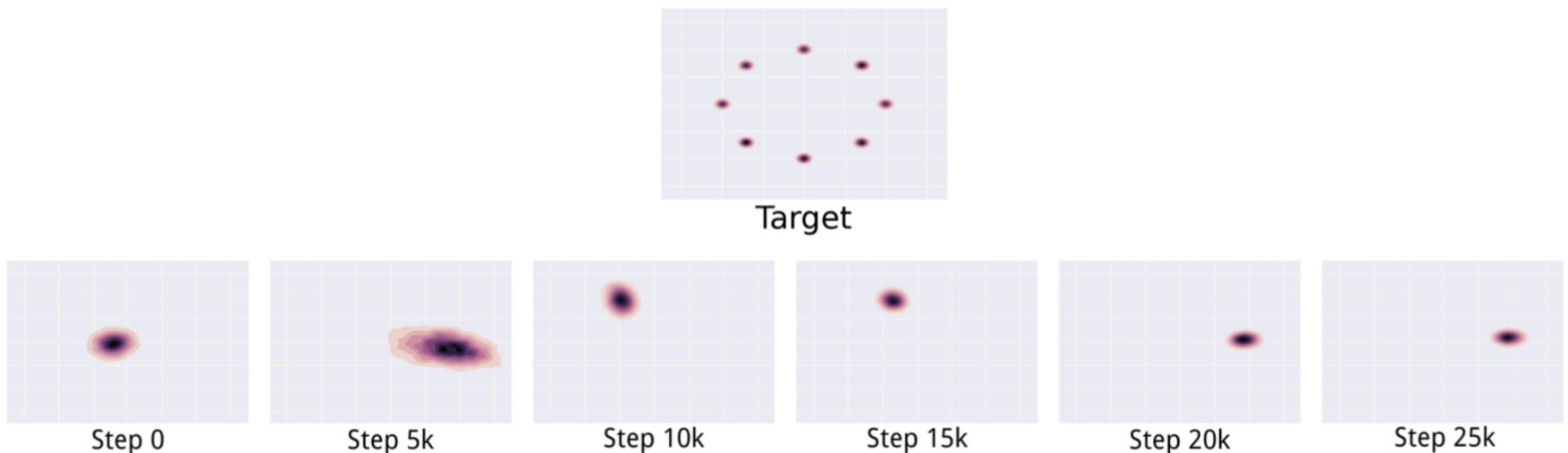
Practical Issues

Strong intra-batch correlation



Practical Issues

Mode Collapse



(Metz et al 2016)

(Goodfellow 2016)

Practical Issues

