Lets talk about generative models

Will Grathwohl - African Masters in Machine Intelligence
June 20, 2020

Types of models in machine learning

- Most problems of interest are discrminative
- We want to make a prediction of a target y given an input x
- $p(\text{is this a dog} \mid \text{input image})$
- $p(\text{should I give this peron a loan} \mid \text{info about the person})$
- The input may be high or low dimensional but the *random* output is usually low

Discriminative modeling

- Parameterize a distribution over the y as a function of x
- Neural net $f : R^D \rightarrow R^K (k << D)$
- Parameterize $\log p(y \mid x) = \log p(y; f(x))$
- Example: Softmax (classification)
  - Model outputs logits $l = f(x)$
  - $p(y = i \mid x) = \frac{e^{l_i}}{\sum_j e^{l_j}}$

Discriminative modeling

- Parameterize a distribution over the y as a function of x
- Neural net $f : R^D \rightarrow R^K (k << D)$
- Parameterize $\log p(y \mid x) = \log p(y; f(x))$
- Example: Gaussian (regression)
  - Model outputs mean,std $\mu, \sigma = f(x)$
  - $p(y = i \mid x) = N(y \mid \mu, \sigma^2)$

Low dim data → simple probabilistic model

- The random variable here is usually small so we can capture its uncertainty with a simple model (gaussian, categorical, etc)
- When data is high dimensional we need to model uncertainty of high dimensional data — much harder!!!

High dim data → simple models don’t work

- Gaussians and categorical distributions cannot model complex high-dimensional distributions
- Real data is highly multimodal and so we need more flexible probability models to handle it
- The field of generative models focuses on these models and how to train them

High dim vs Low dim

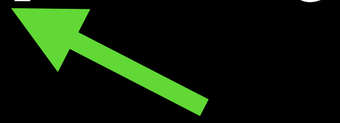
- For discriminative modelling most work involves making $f(x)$ flexible, but we pay little attention to modelling uncertainty in $p(y \mid f(x))$
- For generative modeling all we care about is uncertainty in $p(x)$

Training objectives

- We want to learn a parametric probability distribution $p_\theta(x)$ which closely matches the true data density $p(x)$
- We do this by minimizing KL-divergence between $p_\theta(x)$ and $p(x)$
- $KL(p_\theta \| p) = E_p[\log p(x) - \log p_\theta(x)]$

Generative modeling

- We want to learn a parametric probability distribution $p_\theta(x)$ which closely matches the true data density $p(x)$
- We do this by minimizing KL-divergence between $p_\theta(x)$ and $p(x)$
- $KL(p_\theta \| p) = E_p[\log p(x) - \log p_\theta(x)]$



Constant!

Generative modeling

- We want to learn a parametric probability distribution $p_\theta(x)$ which closely matches the true data density $p(x)$
- We do this by minimizing KL-divergence between $p_\theta(x)$ and $p(x)$
- $KL(p_\theta \| p) = -E_p[\log p_\theta(x)] + C$
- Equivalent to minimizing $E_p[\log p_\theta(x)]$ aka maximum likelihood

We need more expressive models

- I'll go over 4 classes of models which have successfully modelled complicated data like images, text and audio
- These are:
  - VAEs
  - Flows
  - Autoregressive Models
  - GANs

Latent variable models (VAEs)

- VAEs work by defining a *latent variable* model
- We define a model for the data x by adding additional variables that we marginalize out
- $p(x) = \int_z p(x, z) = \int_z p(x \mid z) p(z)$
- A simple example of this is a mixture of 2 Gaussians
- The $p(z) =$ Bernoulli on cluster ID
- $p(x \mid z) = N(x \mid \mu_z, \sigma_z^2)$

Latent variable models (VAEs)

- Incorporating the latent variable allows the model to be more expressive which still easy to write and sample from

VAEs

- In a VAE $p_\theta(x) = \int_z p_\theta(x, z)$
- $p_\theta(x, z) = p_\theta(x \mid z)p(z)$
- $p(z) = N(z \mid 0,1)$
- $p_\theta(x \mid z) = N(x \mid \mu_\theta(z), \sigma^2_\theta(z))$

VAEs (learning)

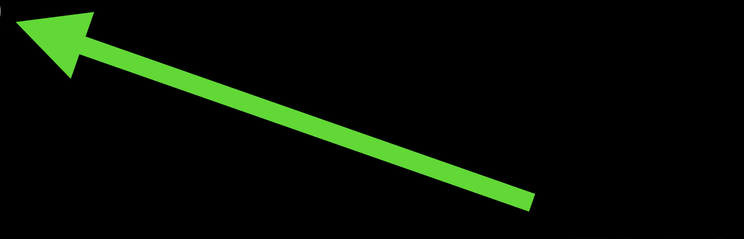
- We wish to maximize $E_p[\log p_\theta(x)]$ but this is hard since
- $$E_p[\log p_\theta(x)] = E_p\left[\log \int_z p_\theta(x \mid z) p(x)\right]$$
- Can’t compute this integral!!

Varational Inference

- We produce a lower bound on $\log p(x)$
- We note:
  - $\log p(x) = \log p(x) - KL(p(z|x)||p(z|x))$
  - $> \log p(x) - KL(p(z|x)||q(z|x))$
  - $= E_{p(z|x)}\left[\log p(x) + \log p(z|x) - \log q(z|x)\right]$
  - $= E_{p(z|x)}\left[\log p(x,z) - \log q(z|x)\right]$

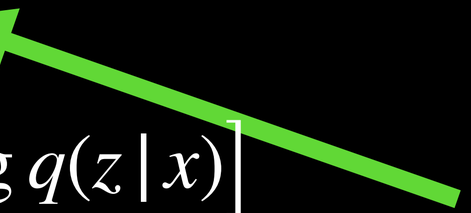
Varational Inference

- We produce a lower bound on $\log p(x)$
- We note:
  - $\log p(x) = \log p(x) - KL(p(z|x)||p(z|x))$
  - $> \log p(x) - KL(p(z|x)||q(z|x))$
  - $= E_{q(z|x)}\left[\log p(x) + \log p(z|x) - \log q(z|x)\right]$
  - $= E_{q(z|x)}\left[\log p(x,z) - \log q(z|x)\right]$



Varational Inference

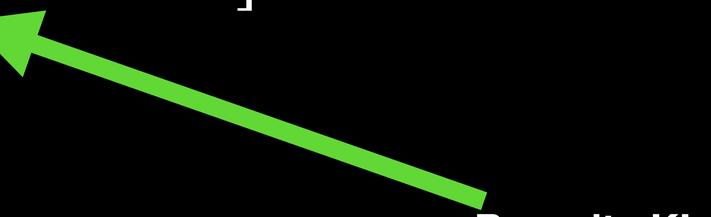
- We produce a lower bound on $\log p(x)$
- We note:
  - $\log p(x) = \log p(x) - KL(p(z|x)||p(z|x))$
  - $> \log p(x) - KL(p(z|x)||q(z|x))$
  - $= E_{q(z|x)}\left[\log p(x) + \log p(z|x) - \log q(z|x)\right]$
  - $= E_{q(z|x)}\left[\log p(x,z) - \log q(z|x)\right]$



This Is >0

Varational Inference

- We produce a lower bound on $\log p(x)$
- We note:
  - $\log p(x) = \log p(x) - KL(p(z|x)||p(z|x))$
  - $> \log p(x) - KL(p(z|x)||q(z|x))$
  - $= E_{q(z|x)}\left[\log p(x) + \log p(z|x) - \log q(z|x)\right]$
  - $= E_{q(z|x)}\left[\log p(x,z) - \log q(z|x)\right]$



Varational Inference

- We produce a lower bound on $\log p(x)$
- We note:
  - $\log p(x) = \log p(x) - KL(p(z|x)||p(z|x))$
  - $> \log p(x) - KL(p(z|x)||q(z|x))$
  - $= E_{q(z|x)}\left[\log p(x) + \log p(z|x) - \log q(z|x)\right]$
  - $= E_{q(z|x)}\left[\log p(x,z) - \log q(z|x)\right]$ ← **Tractable bound!**

Varational Inference

- We have:
  - $\log p(x) > E_{q(z|x)}\left[\log p(x,z) - \log q(z\,|\,x)\right]$ for any $q(z\,|\,x)$
  - Bound is tight when $p(z\,|\,x) = q(z\,|\,x)$
  - We parameterize as a neural net
  - $q_\phi(z\,|\,x) = N(z\,|\,\mu_\phi(x), \sigma^2_\phi(x))$

Taking a step back

- We have:
  - Our model $p_\theta(x \mid z)p(z)$
  - Our inference distribution $q_\phi(z \mid x)$
  - Lower bound on likelihood:
    - $\log p(x) > E_{q(z|x)}\left[\log p(x, z) - \log q(z \mid x)\right]$
  - Train θ, ϕ together both to maximize the bound!

Modest beginnings

Bigger and better

Problems with VAE

- We cannot exactly compute $\log p(x)$ we can only lower bound it
- For some applications we need exact computation
- Next model addresses that

Normalizing flows

- Normalizing flows present a simple generative process for data
- Sample $z \sim p(z)$ (simple $p(z)$ such as a Gaussian)
- Apply *invertible* function $x = F(x)$
- *If* F is invertible we can use the change-of-variables formula to write
- $\log p(x) = \log p(F^{-1}(x)) + \log |\nabla_x F^{-1}(x)|$

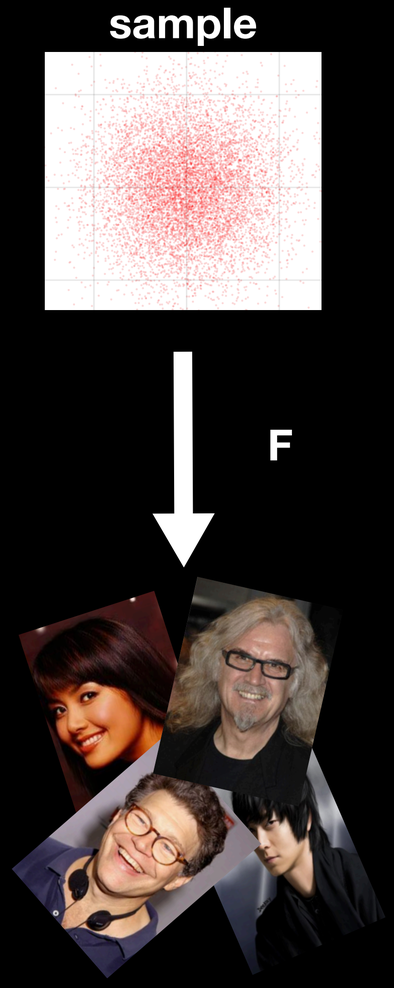
generative modeling with the change of variables formula

we can define a simple generative model as

$$z \sim p(z)$$
$$x = F_\theta(z)$$

if F is invertible…

$$\log p(x) = \log p(F_\theta^{-1}(x)) + \log \left| \frac{\partial F_\theta^{-1}}{\partial x} \right|$$



Normalizing flows

- We let F be a neural network F_θ and train its parameters to maximize likelihood
- F_θ cannot be *any* neural network:
  - F_θ must be invertible (for c.o.v. to hold)
  - $\log |\nabla_x F^{-1}(x)|$ must be easy to compute (for efficiency)

A Solution

affine coupling layers (Dinh et al., 2016)

split input: $x \rightarrow x_a, x_b$

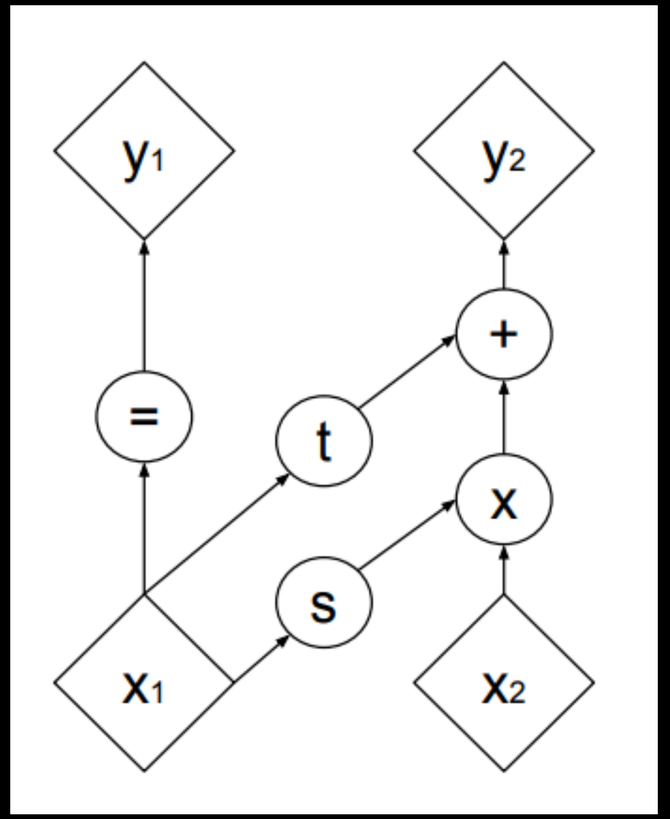
generate transform: $s, t = f(x_a)$

apply:

$$y_a = x_a$$

$$y_b = s \cdot x_b + t$$

recombine: $y = y_a, y_b$



previous solution

affine coupling layers (Dinh et al., 2016)

split input: $x \rightarrow x_a, x_b$

generate transform: $s, t = f(x_a)$ | $s, t = f(y_a)$

apply:

$$y_a = x_a$$
$$y_b = s \cdot x_b + t$$

$$x_a = y_a$$
$$x_b = (y_b - t)/s$$

recombine: $y = y_a, y_b$

lower-triangular Jacobian:

$$\log \left| \frac{\partial F}{\partial x} \right| = \sum_i \log s_i$$

efficient log-det

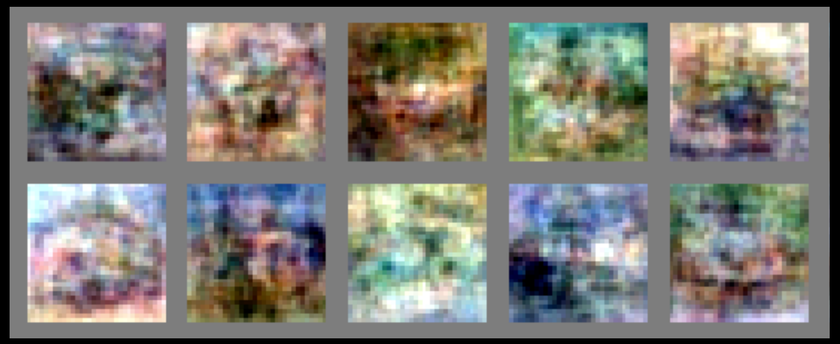
easy to invert

Difficulties

- Many ways to split input
- Invertibility can affect expressive power
- Having tractable log-det also limits expressive power greatly
- Most research on flows works on building neural networks which have these properties but are more flexible

progress

NICE (Dinh et al., 2014)



Real-NVP (Dinh et al., 2016)

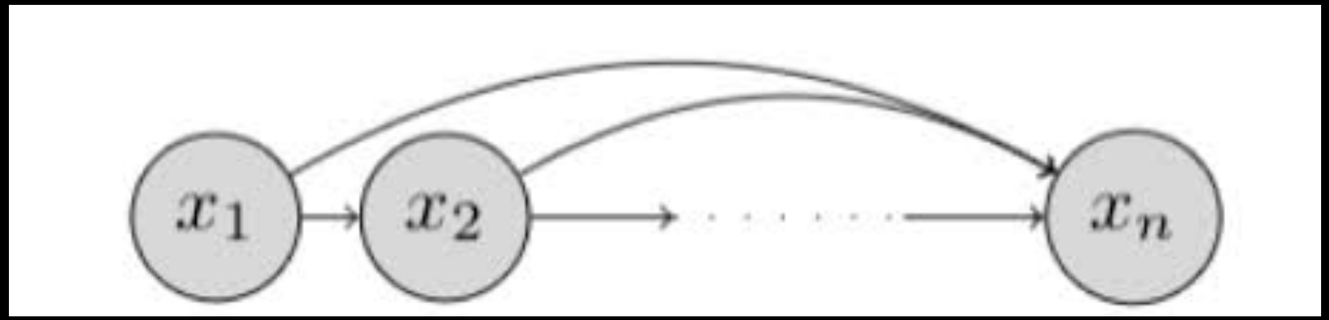
Glow (Kingma et al., 2018)

Progress

- Generally lagging behind VAE and other models but has the nicest properties, so its a trade-off
- If you need exact likelihoods and fast sampling then flows are the best thing out there

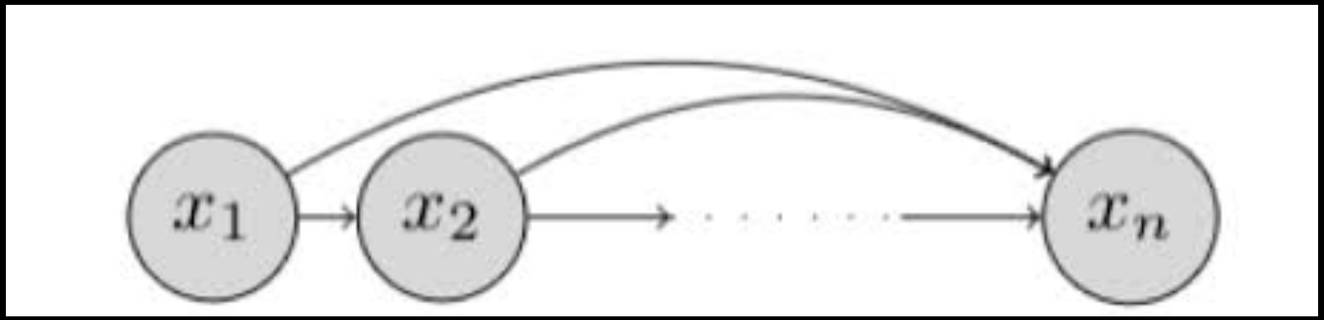
Autoregressive Models

- Autoregressive models take a different approach they note for any distribution of n-dimensional data $p(x_1, \ldots, x_n)$ it can be factorized
- $p(x_1, \ldots, x_n) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2, x_1) \cdots p(x_n \mid x_1, \ldots, x_{n-1})$



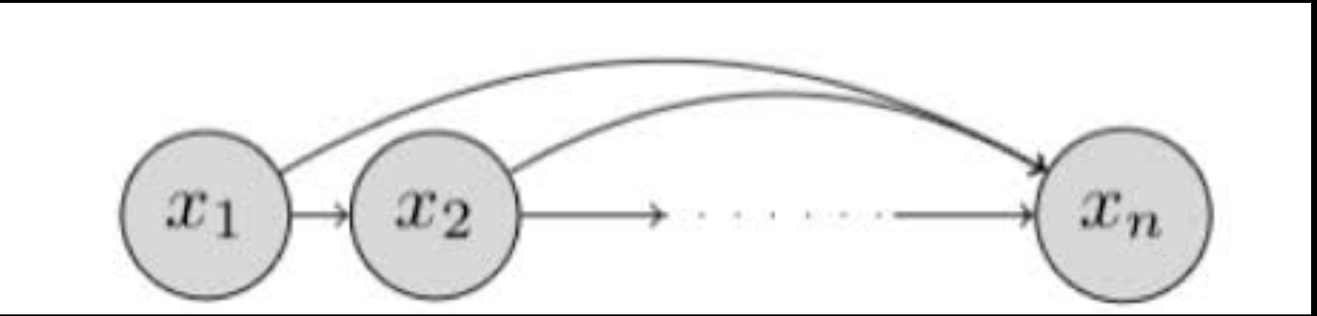
Autoregressive Models

- Can be done for *any* distribution and *any* ordering of variables



Autoregressive Models

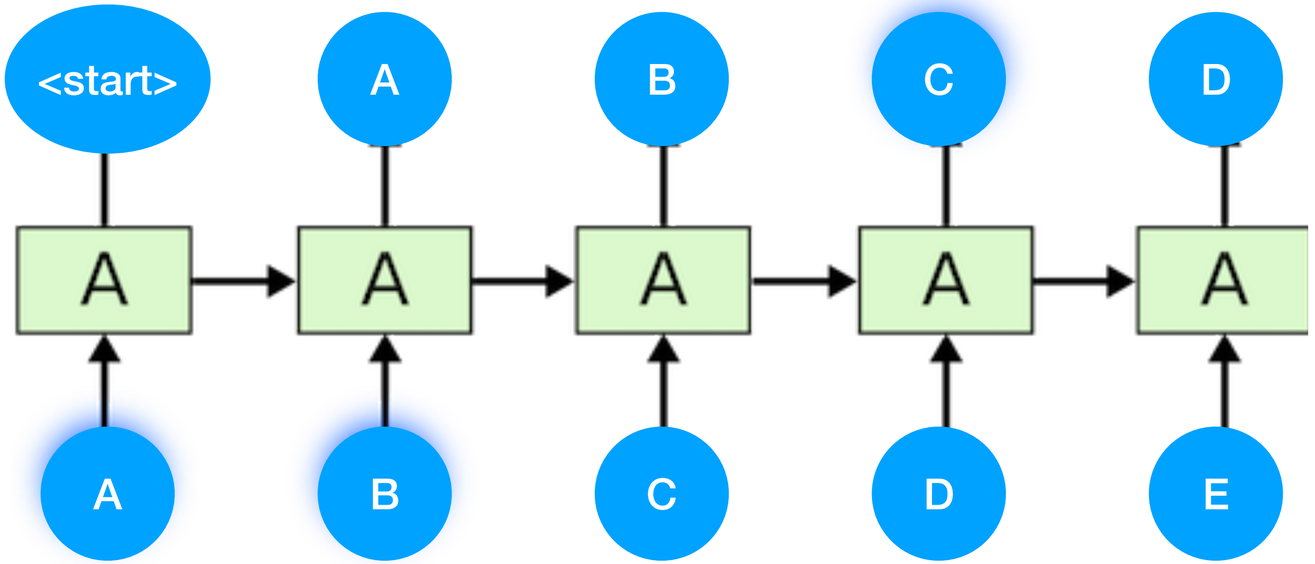
- Autoregressive models model $p(x)$ by learning each individual conditional for *some* ordering of dimensions



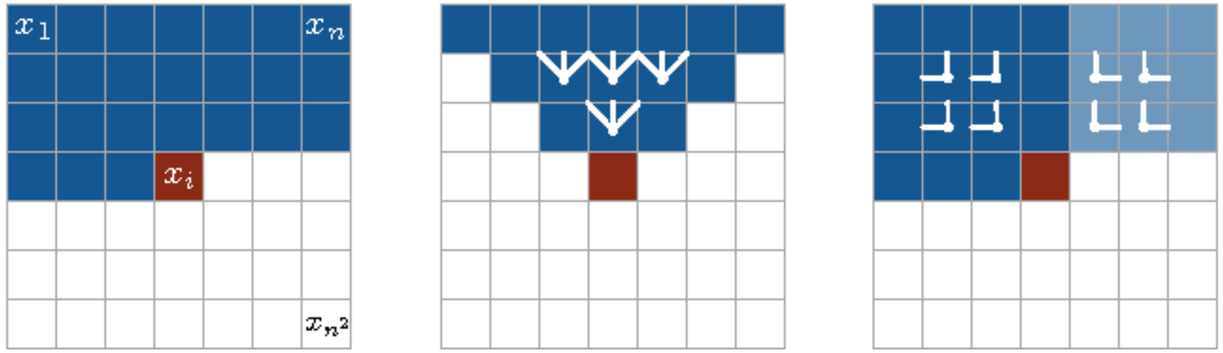
Autoregressive Models

- This is good for spatially structured data such as images or text
- Can be implemented with RNNs or Transformers

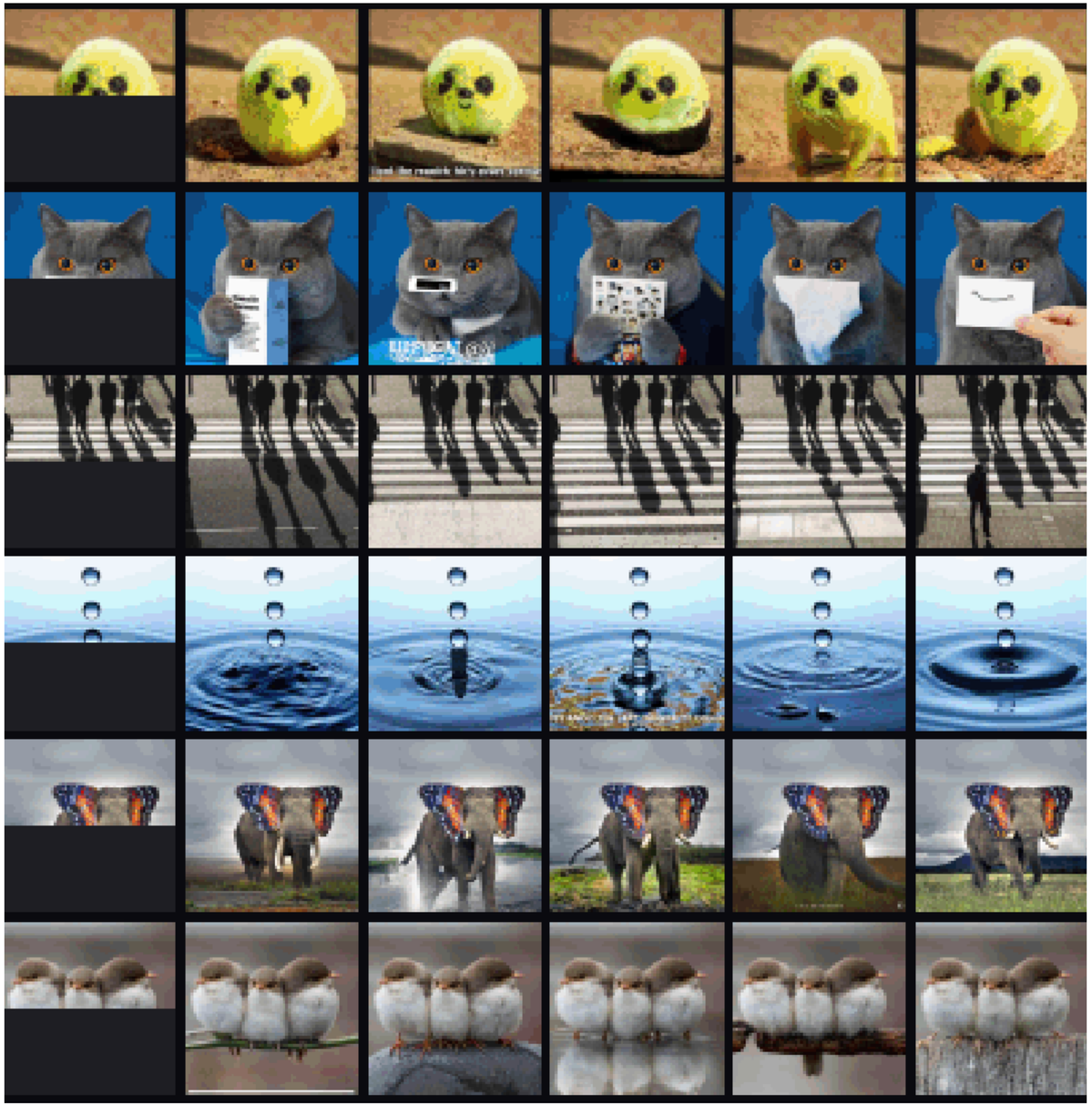
Visualized (RNN)



$$p(x) = \prod_i p_\theta(x_i \mid x_{<i})$$



Pixel Recurrent Neural Networks
Aaron van den Oord, Nal Kalchbrenner, Koray Kavukcuoglu



Autoregressive Models

- Exact likelihoods
- Efficient likelihood computation
- Slow sampling $O(n)$ vs $O(1)$ for flow and VAE
- Less interpretable — no leaning of latent variables
- Hard to pull out representations

GANs and implicit models

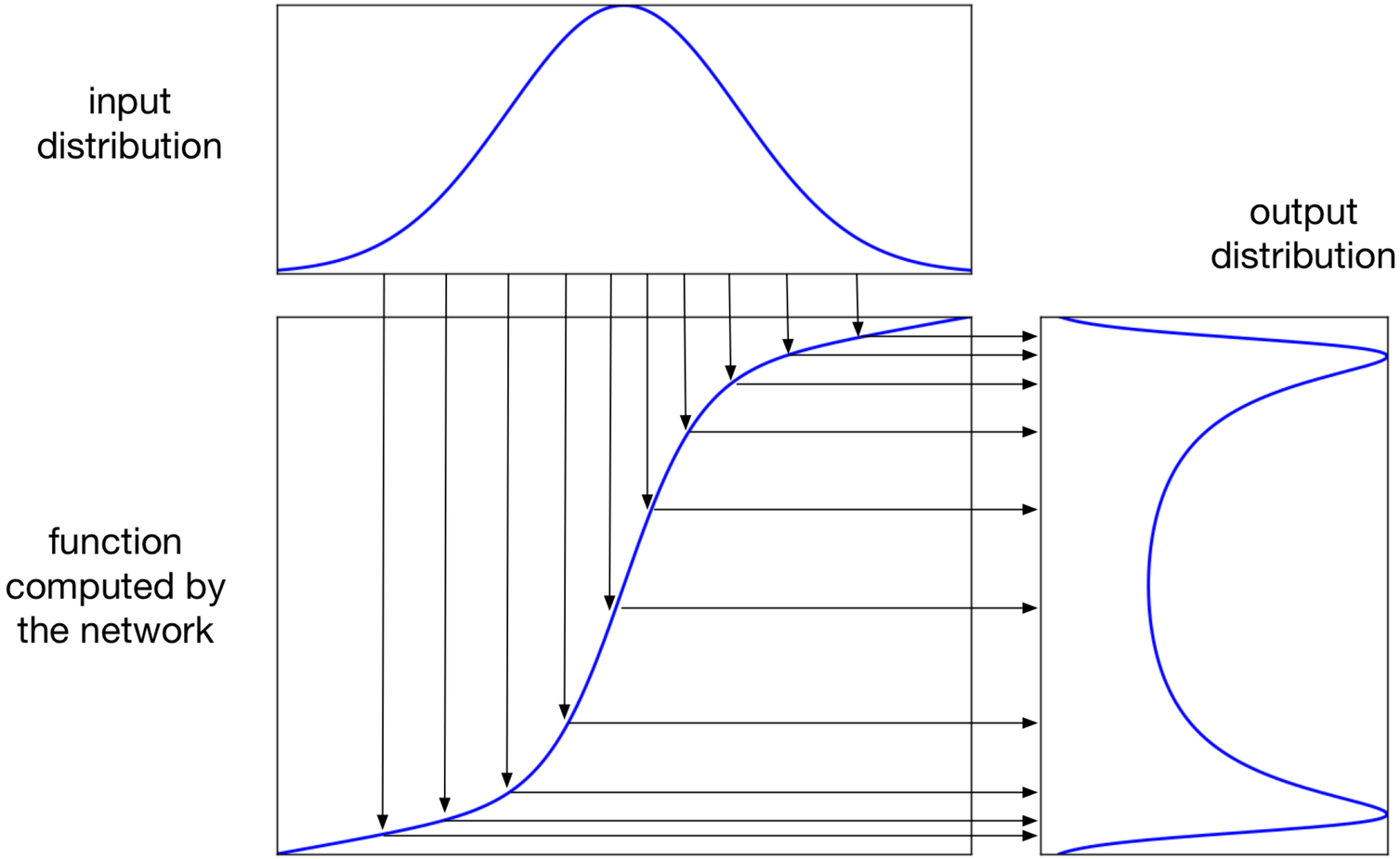
- GANs define a data distribution in a slightly different way
- Like a VAE we sample $z \sim p(z)$ (Gaussian)
- Then pass z through a neural net to produce the data $x = g_\theta(z)$
- Here we do not restrict the structure of g_θ like we do in the Flow
- We also do not add output noise
- This means that we cannot compute $\log p(x)$

GANs and implicit models

- The distribution of $p(x)$ exists *implicitly* but we have no hope of ever computing it
- We have a generative model which we can only use to draw samples
- Why?
  - Much more flexibility
  - Smaller models
  - More efficient

Generative Adversarial Networks

A 1-dimensional example:



How to Train?

- We can train a classifier $D(x)$ to classify samples as being generated by the generator or real data
- The value of this classifier converges to $\log \frac{p_\theta(x)}{p_{\text{data}}(x)}$
- We train the generator to *fool* the discriminator
- Forms an adversarial game between two players (generator and discriminator)
- This game has a stable equilibrium when $D(x) = 0$ and $p_\theta(x) = p_{\text{data}}(x)$

Minimax Game

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

$$J^{(G)} = -J^{(D)}$$

-Equilibrium is a saddle point of the discriminator loss

-Resembles Jensen-Shannon divergence

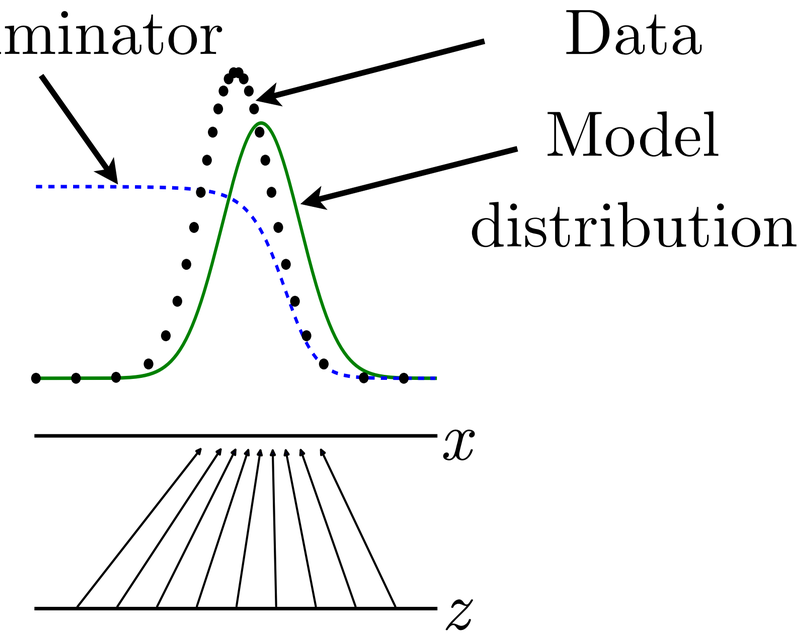
-Generator minimizes the log-probability of the discriminator being correct

Discriminator Strategy

Optimal $D(\boldsymbol{x})$ for any $p_{\text{data}}(\boldsymbol{x})$ and $p_{\text{model}}(\boldsymbol{x})$ is always

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

Estimating this ratio using supervised learning is the key approximation mechanism used by GANs



A new challenger approaches…

- Recently, a new class of generative model has become popular
- These are Energy-Based Models (EBMs)

energy-based models

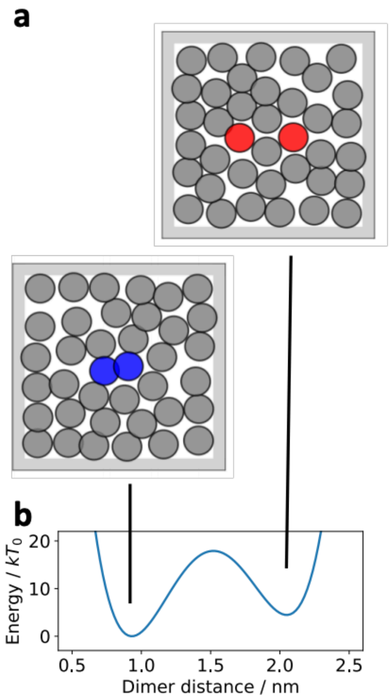
- an energy-based model (EBM) is any model which parameterizes a density with:

$$p_\theta(x) = \frac{e^{-E_\theta(x)}}{Z(\theta)} \qquad Z(\theta) = \int_x e^{-E_\theta(x)} dx$$

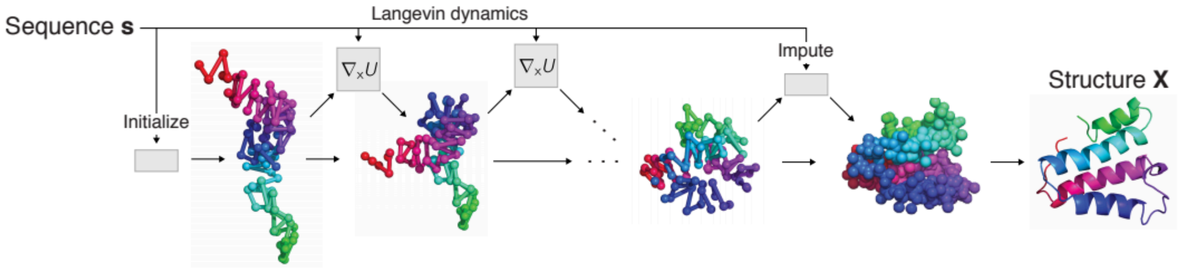
- where $E_\theta : \mathbf{R}^D \to \mathbf{R}$ fully specifies the model

energy-based models

- long been popular to model systems in physics, chemistry, biology, etc…



Noe et al.

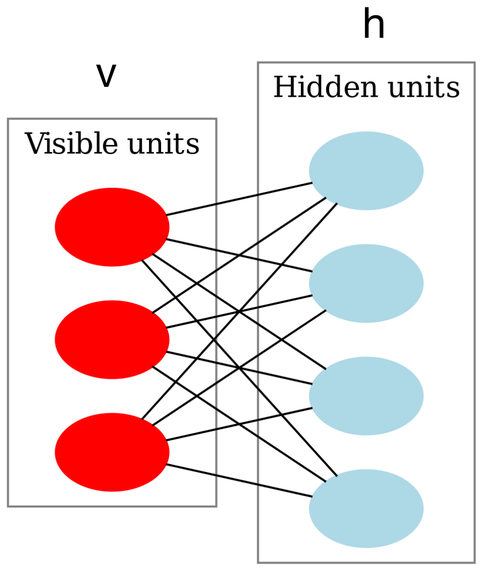


Ingraham et al.

energy-based models

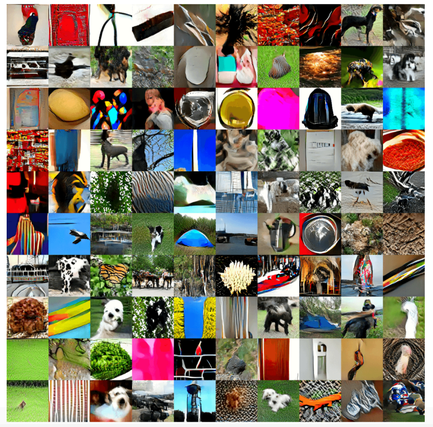
- foundational to deep learning as well with RBMs

$$E(v,h) = -a^T v - b^T v - v^T W h, \qquad p(v,h) = \frac{e^{-E(v,h)}}{Z}$$



energy-based models

- recently re-popularized for generative modeling with great success



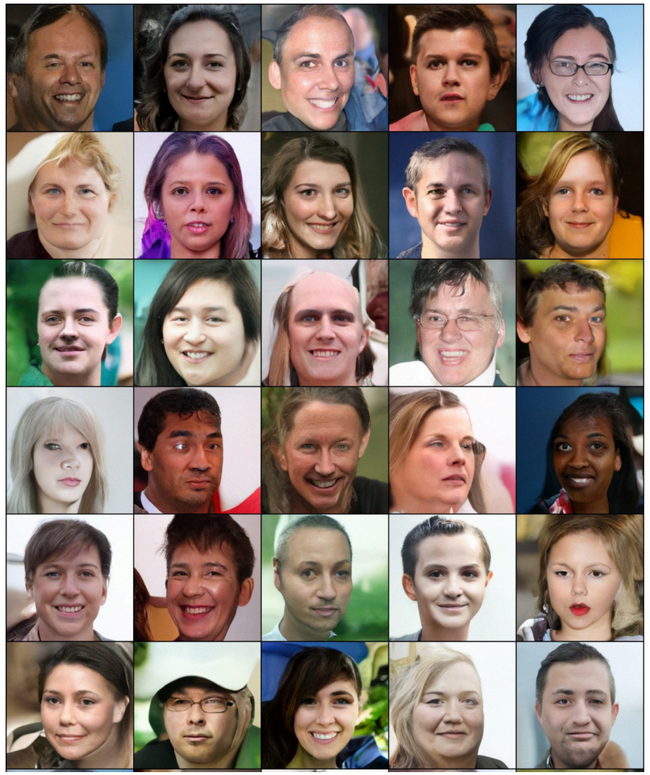
Du & Mordatch (2019)

(a) MNIST (b) CelebA (c) CIFAR-10

Song & Ermon (2019)

Hot off the presses

- recently re-popularized for generative modeling with great success

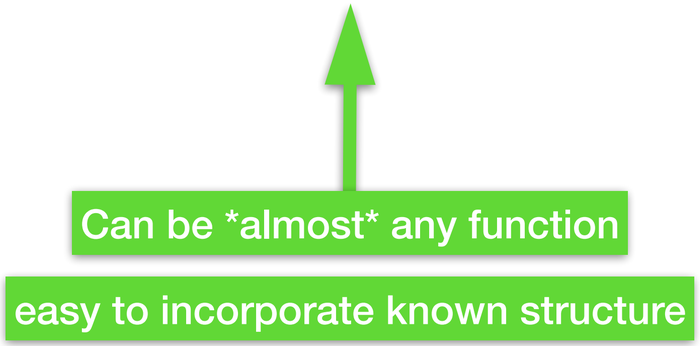


energy-based models

- an energy-based model (EBM) parameterizes a density using its unnormalized log-density function

$$p_\theta(x) = \frac{e^{-E_\theta(x)}}{Z(\theta)} \qquad Z(\theta) = \int_x e^{-E_\theta(x)} dx$$

- where $E_\theta : \mathbf{R}^D \to \mathbf{R}$



energy-based models

- an energy-based model (EBM) parameterizes a density using its unnormalized log-density function

$$p_\theta(x) = \frac{e^{-E_\theta(x)}}{Z(\theta)} \qquad Z(\theta) = \int_x e^{-E_\theta(x)} dx$$

- where $E_\theta : \mathbf{R}^D \to \mathbf{R}$

Can be *almost* any function

easy to incorporate known structure

inctractable to compute or even estimate

cannot efficeintly compute likelihoods or draw samples

training EBMs

- given we cannot compute likelihoods we cannot train to maximize likelihood (obviously)
- we must be a bit more clever
  - approximate ML gradient

approximate likelihood gradient

- while $\log p_\theta(x)$ does not have a nice form, $\frac{\partial \log p_\theta(x)}{\partial \theta}$ can be written more simply as

$$\frac{\partial \log p_\theta(x)}{\partial \theta} = \mathbf{E}_{p_\theta(x')}\left[\frac{\partial E_\theta(x)}{\partial \theta}\right] - \frac{\partial E_\theta(x)}{\partial \theta}$$

approximate likelihood gradient

- while $\log p_\theta(x)$ does not have a nice form, $\frac{\partial \log p_\theta(x)}{\partial \theta}$ can be written more simply as

$$\frac{\partial \log p_\theta(x)}{\partial \theta} = \mathbf{E}_{p_\theta(x')}\left[\frac{\partial E_\theta(x)}{\partial \theta}\right] - \frac{\partial E_\theta(x)}{\partial \theta}$$

expectation over model distribution

need samples to estimate

approximate likelihood gradient

- while $\log p_\theta(x)$ does not have a nice form, $\frac{\partial \log p_\theta(x)}{\partial \theta}$ can be written more simply as

$$\frac{\partial \log p_\theta(x)}{\partial \theta} = \mathbf{E}_{p_\theta(x')}\left[\frac{\partial E_\theta(x)}{\partial \theta}\right] - \frac{\partial E_\theta(x)}{\partial \theta}$$

- this estimator is used to train RBMs

approximate likelihood gradient

- while $\log p_\theta(x)$ does not have a nice form, $\frac{\partial \log p_\theta(x)}{\partial \theta}$ can be written more simply as

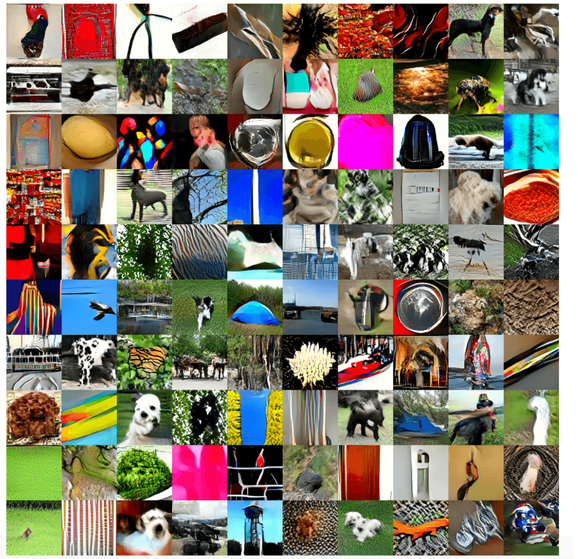
$$\frac{\partial \log p_\theta(x)}{\partial \theta} = \mathbf{E}_{p_\theta(x')}\left[\frac{\partial E_\theta(x)}{\partial \theta}\right] - \frac{\partial E_\theta(x)}{\partial \theta}$$

- recent work as successfully approximated this with samples using Langevin Dynamics

$$x_t = x_{t-1} - \alpha \frac{\partial E_\theta(x_{t-1})}{\partial x_{t-1}} + \mathcal{N}\left(0, \sqrt{2\alpha}\right)$$

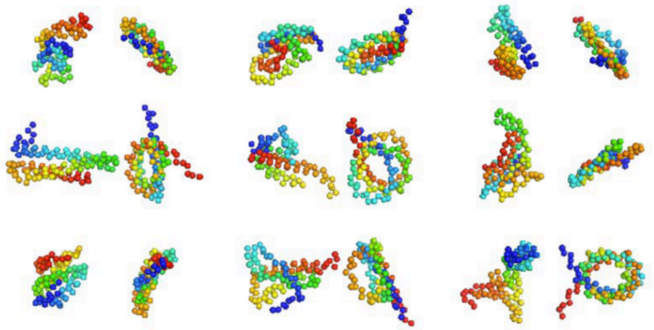
recent successes

- image data, CNN energy-function



Du & Mordatch (2019)

- protein folding, bio-inspired neural-network energy-function



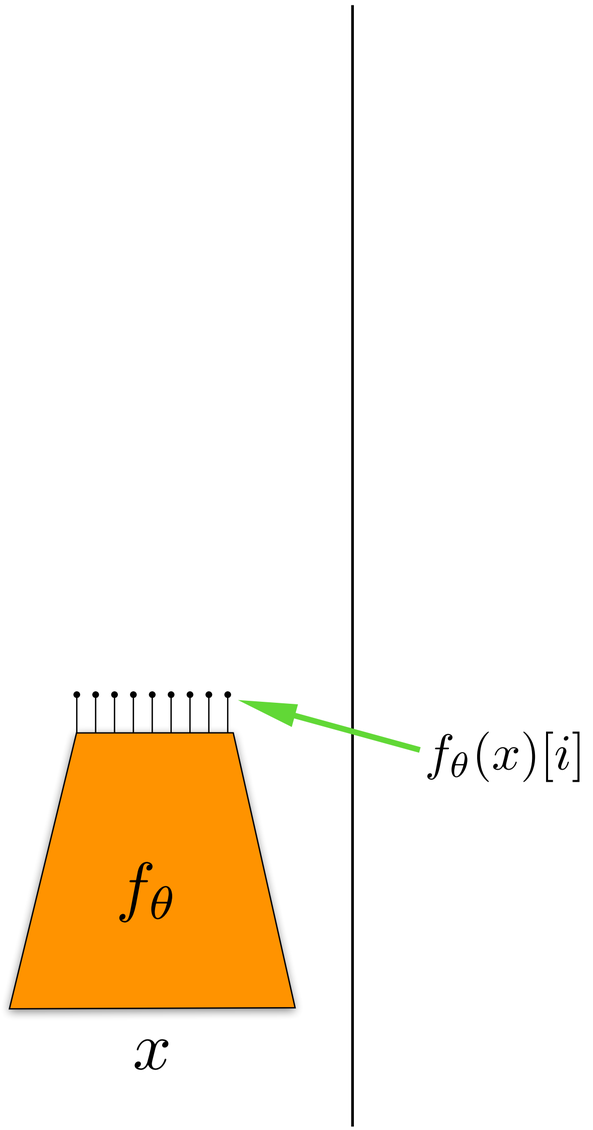
Ingraham et al. (2019)

One cool applications

- Recent work shows that EBMs can improve discriminative models

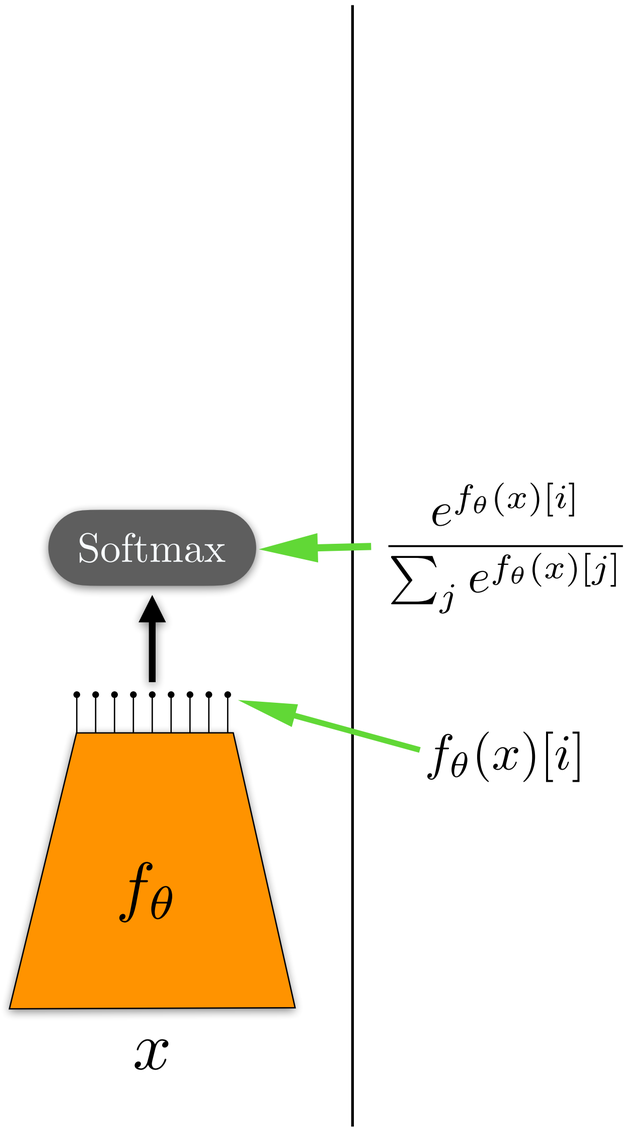
architectures for classification

- classification tasks solved by modeling $p(y \mid x)$ and maximizing likelihood
- we do this with a function $f_\theta : \mathbf{R}^D \to \mathbf{R}^K$
- we pass these K outputs through a softmax function to obtain $p(y \mid x)$



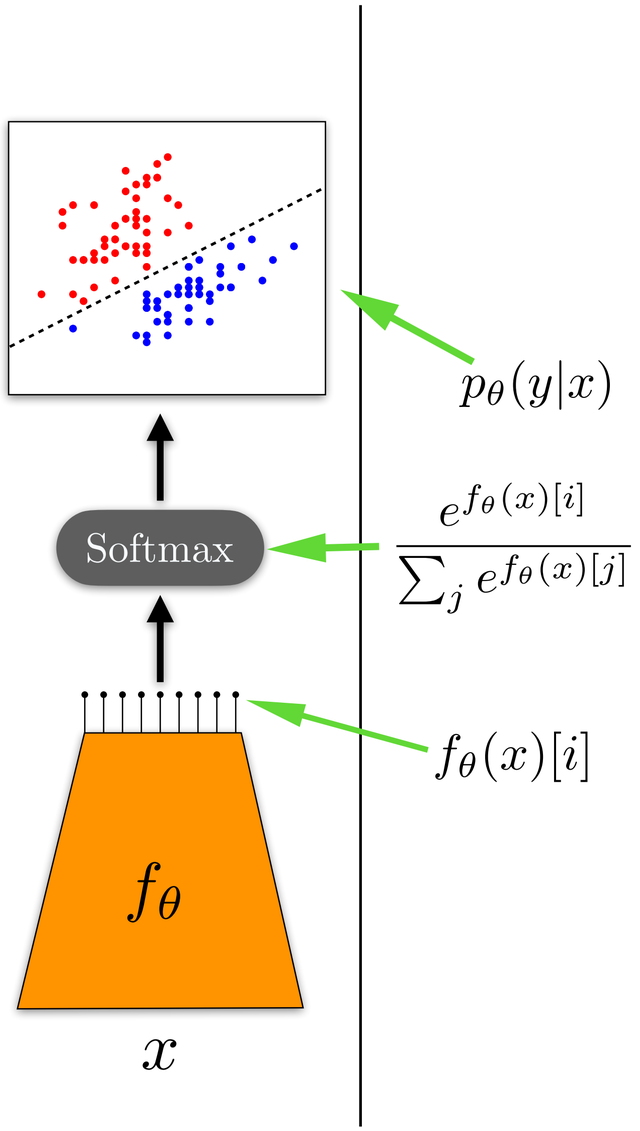
architectures for classification

- classification tasks solved by modeling $p(y \mid x)$ and maximizing likelihood
- we do this with a function $f_\theta : \mathbf{R}^D \to \mathbf{R}^K$
- we pass these K outputs through a softmax function to obtain $p(y \mid x)$



architectures for classification

- classification tasks solved by modeling $p(y \mid x)$ and maximizing likelihood
- we do this with a function $f_\theta : \mathbf{R}^D \to \mathbf{R}^K$
- we pass these K outputs through a softmax function to obtain $p(y \mid x)$



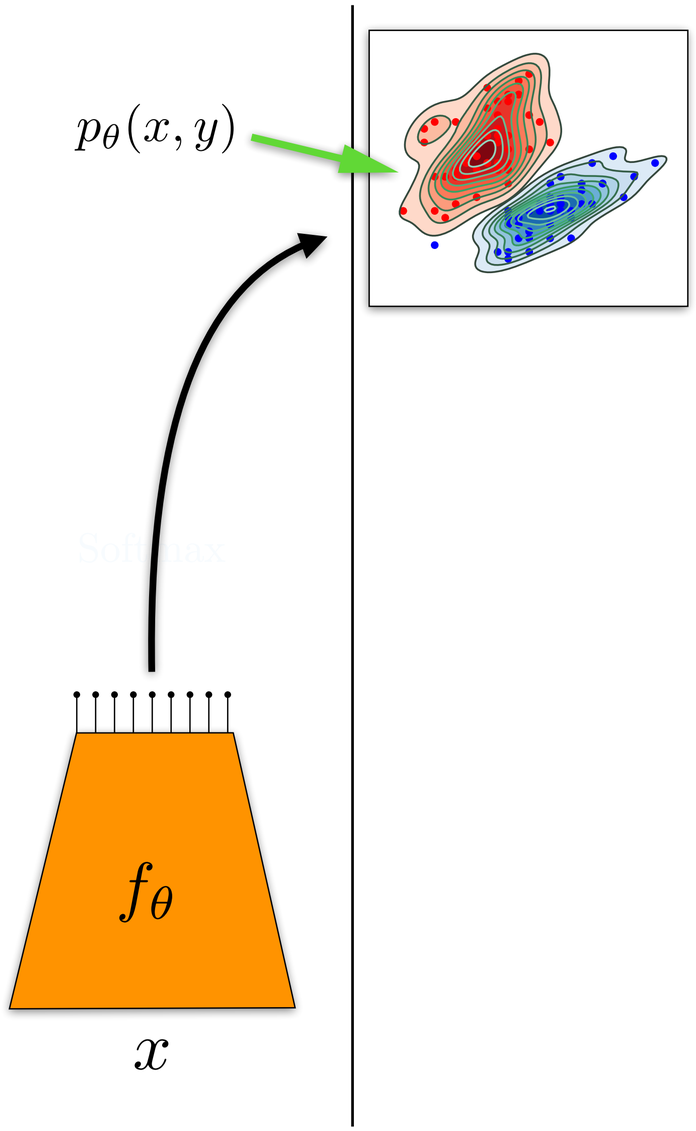
a different perspective

- lets redefine what these outputs mean

$$p_\theta(x, y) = \frac{e^{f_\theta(x)[y]}}{Z(\theta)}$$

- this is an EBM with energy

$$E_\theta(x, y) = -f_\theta(x)[y]$$



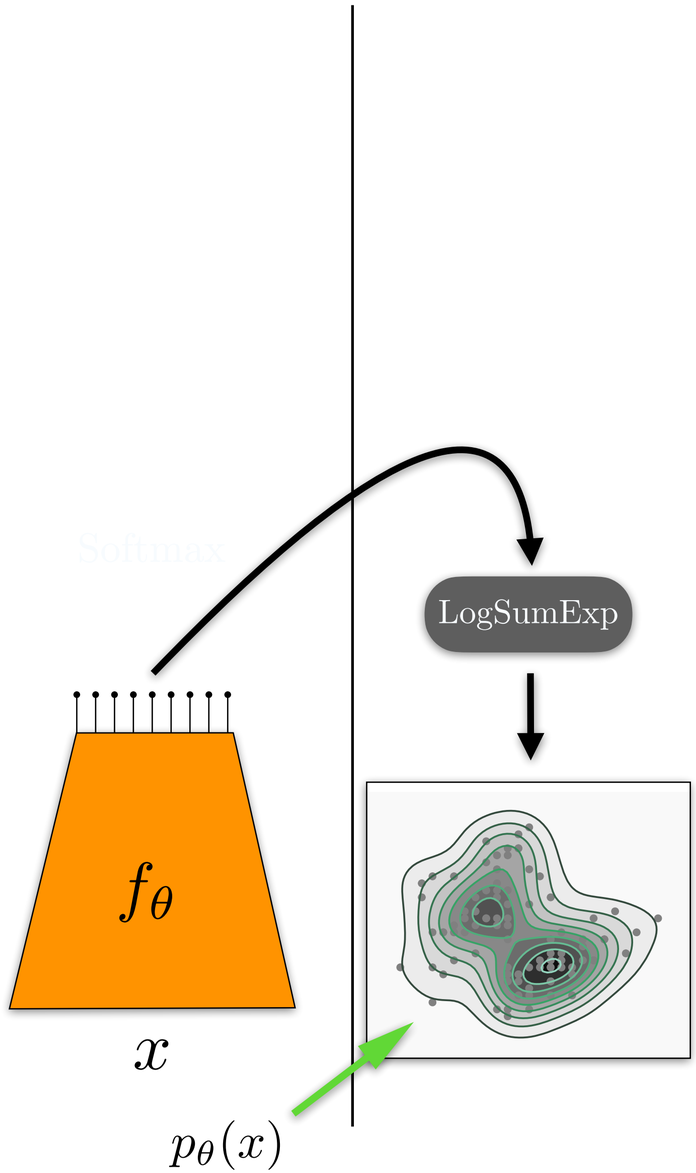
a different perspective

- summing out y we obtain

$$p_\theta(x) = \sum_y p_\theta(x, y) = \frac{\sum_y e^{f_\theta(x)[y]}}{Z(\theta)}$$

- which is an EBM with energy

$$E_\theta(x) = -\,\text{LogSumExp}_y(f_\theta(x)[y])$$

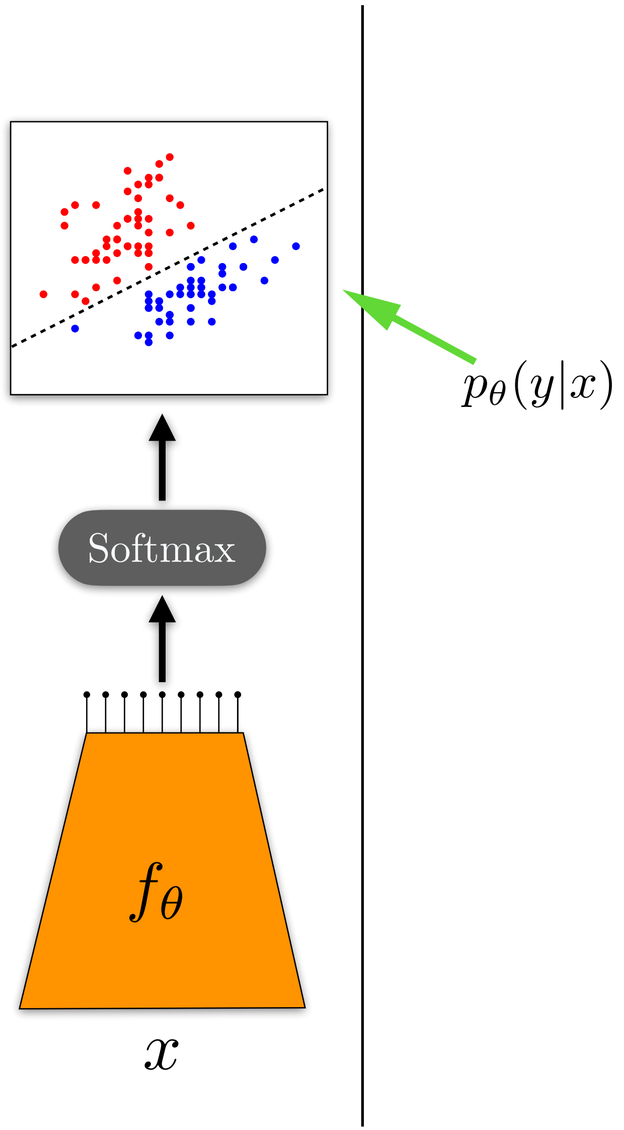


a different perspective

- when we compute $p_\theta(y \mid x) = \frac{p_\theta(x, y)}{p_\theta(x)}$ we get

$$p_\theta(y \mid x) = \frac{e^{f_\theta(x)[y]}}{\sum_{y'} e^{f_\theta(x)[y']}}$$

standard softmax!!



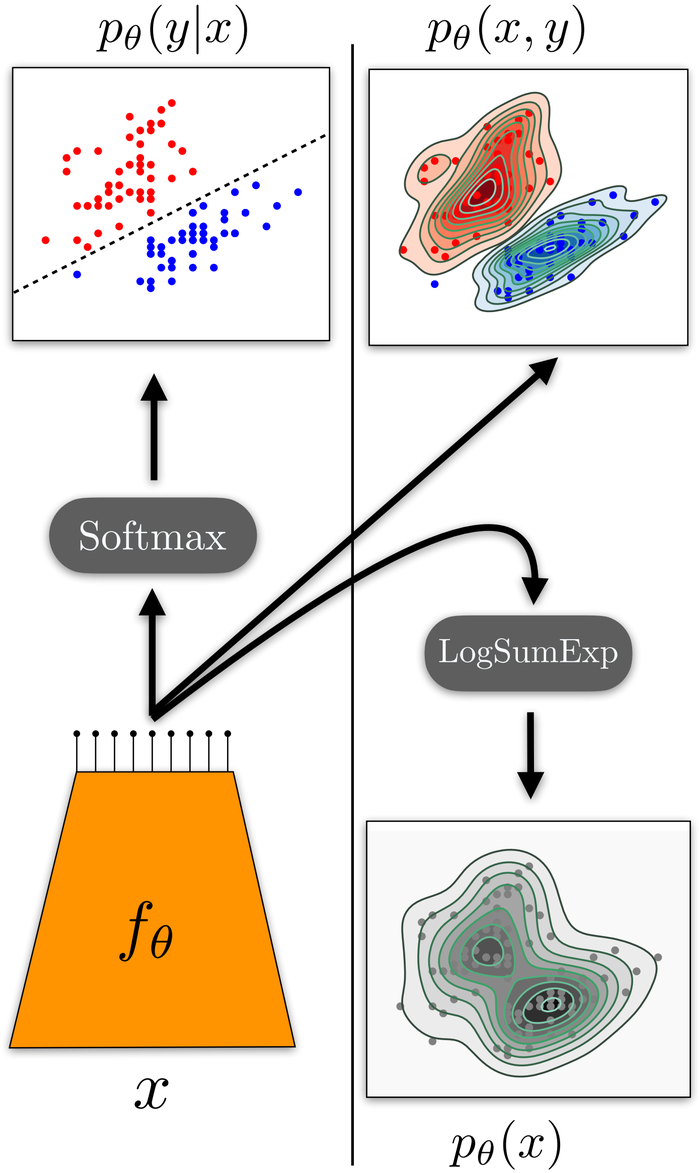
all together now

- without changing our architecture we have found models of

$$p_\theta(x, y) = \frac{e^{f_\theta(x)[y]}}{Z(\theta)}$$

$$p_\theta(x) = \frac{\sum_y e^{f_\theta(x)[y]}}{Z(\theta)}$$

$$p_\theta(y \mid x) = \frac{e^{f_\theta(x)[y]}}{\sum_{y'} e^{f_\theta(x)[y']}}$$



all together now

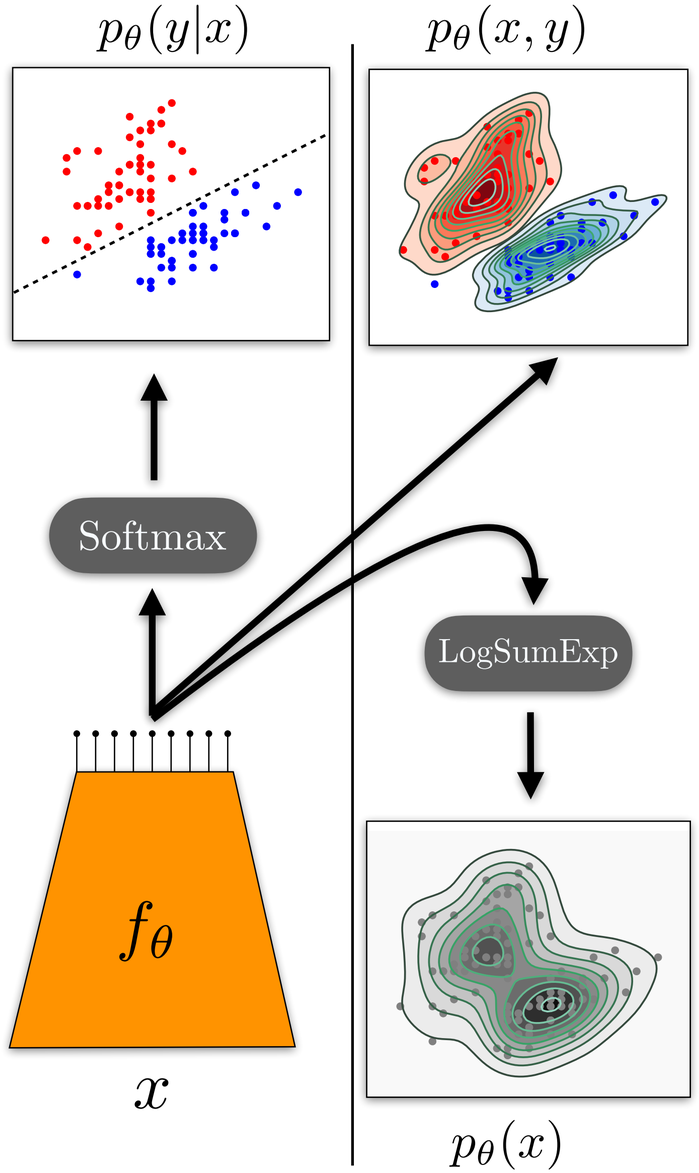
- without changing our architecture we have found models of

$$p_\theta(x, y) = \frac{e^{f_\theta(x)[y]}}{Z(\theta)}$$

$$p_\theta(x) = \frac{\sum_y e^{f_\theta(x)[y]}}{Z(\theta)}$$

$$p_\theta(y \mid x) = \frac{e^{f_\theta(x)[y]}}{\sum_{y'} e^{f_\theta(x)[y']}}$$

- we have found a Joint Energy Model inside of your classifier…a hidden JEM 😉

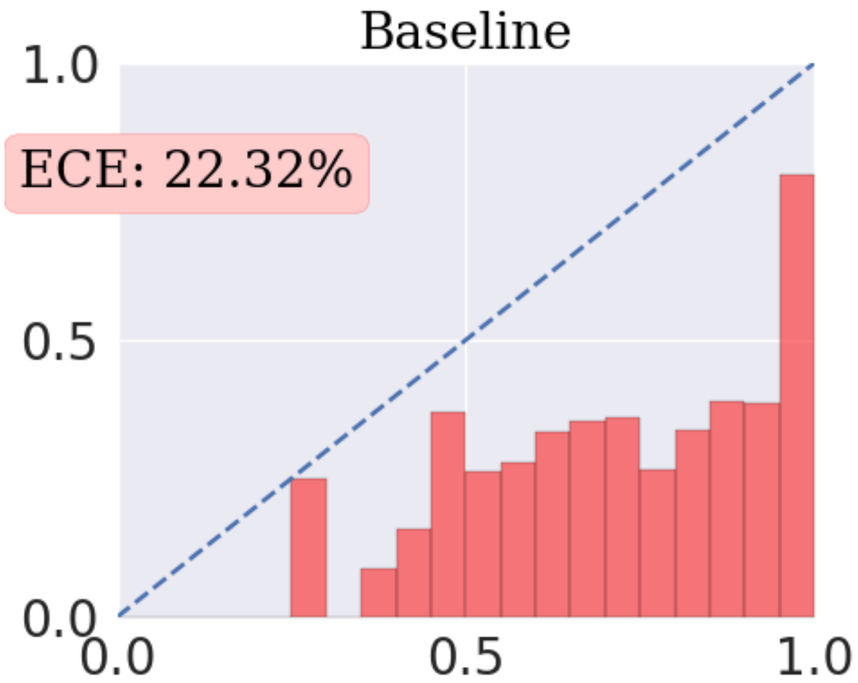


results: hybrid modeling

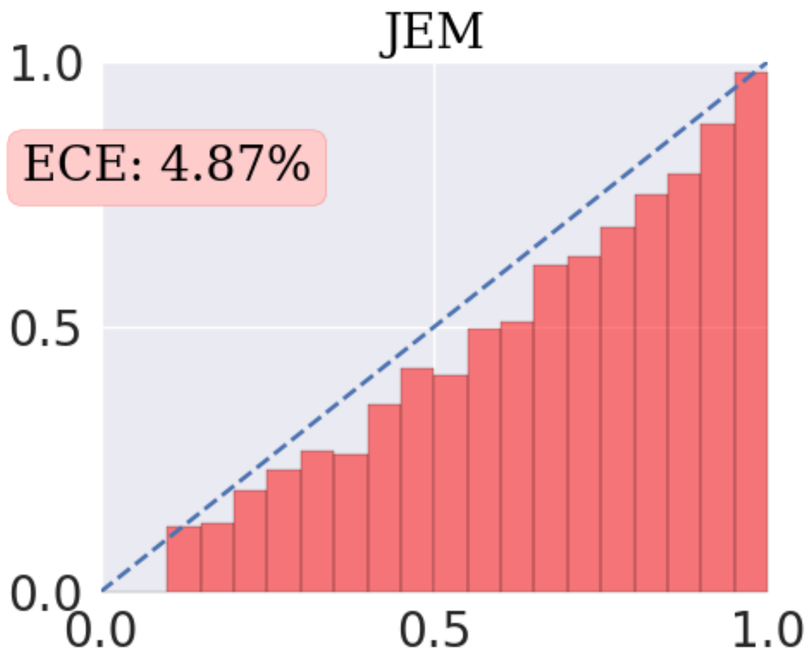
Class	Model	Accuracy% ↑	IS↑	FID↓
Hybrid	Residual Flow	70.3	3.6	46.4
	Glow	67.6	3.92	48.9
	IGEBM	49.1	8.3	**37.9**
	JEM $p(\mathbf{x}\|y)$ factored	30.1	6.36	61.8
	JEM (Ours)	**92.9**	**8.76**	38.4
Disc.	Wide-Resnet	95.8	N/A	N/A
Gen.	SNGAN	N/A	8.59	25.5
	NCSN	N/A	8.91	25.32

CIFAR10 Quantitative Results

results: calibration



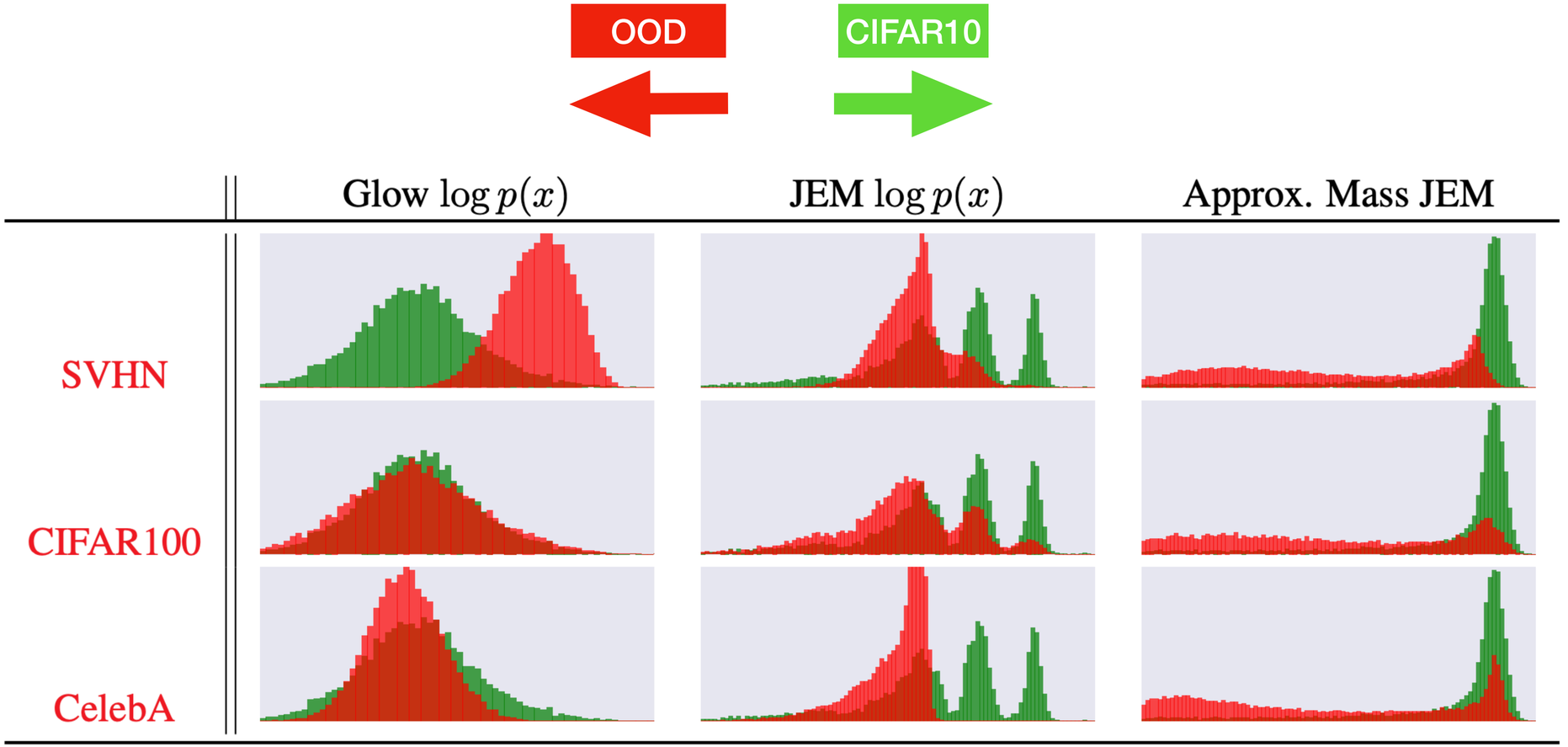
(74.2% Accuracy)



(72.2% Accuracy)

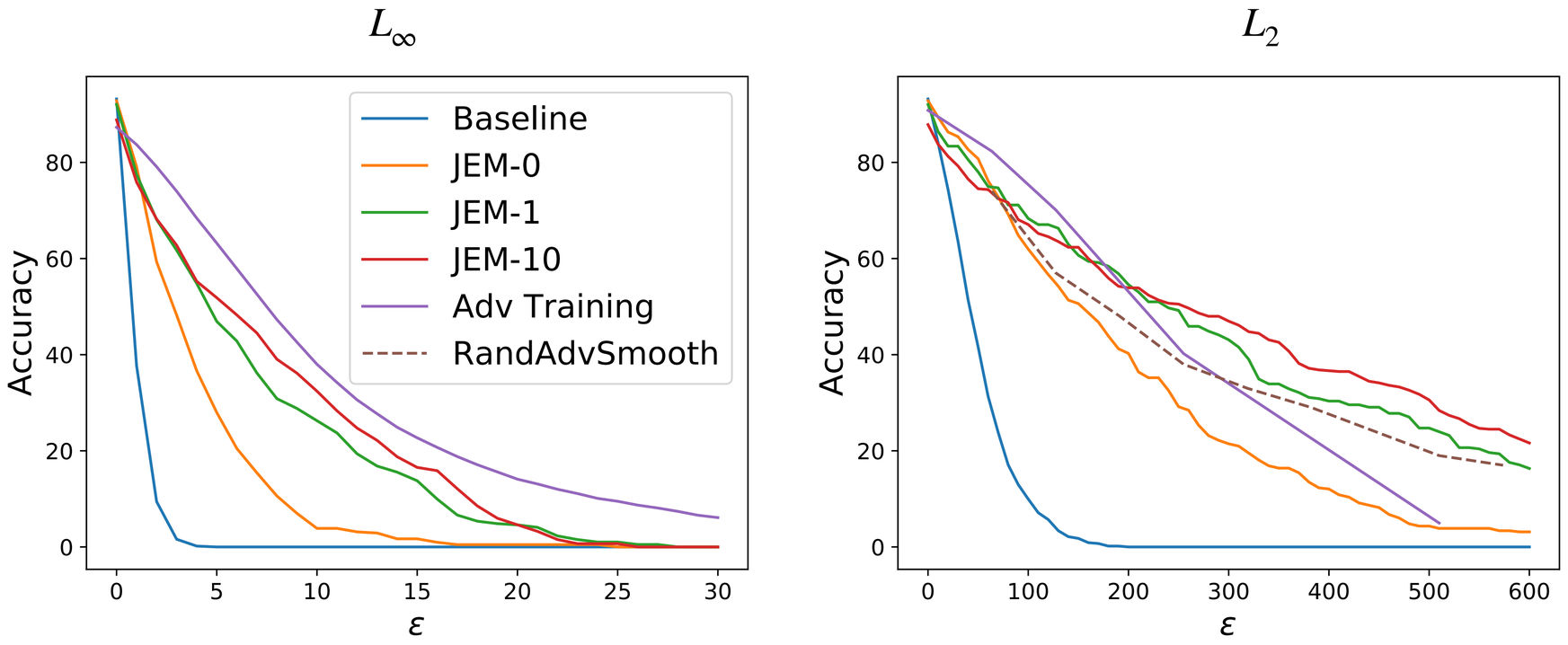
CIFAR100

results: out-of-distribution detection



results: adversarial robustness

- does this make more robust models?



EBMs

- Lots of potential, but field is still new and growing! Lots of room to make progress!

- Autoregressive Models: $x_i \sim p_\theta(x_i \mid x_{<i}), \quad p(x) = \prod_i p_\theta(x_i \mid x_{<i})$
  - Pros: Exact likelihoods, easy to train
  - Cons: O(N) layers to evaluate or sample, need to order dimensions
- Variational Autoencoders: $x \sim p_\theta(x \mid z), \quad p(x) = \int p(x \mid z) p(z) dz$
  - Pros: Cheap to evaluate and sample, low-D latents
  - Cons: Factorized likelihood gives noisy samples
- Explicitly normalized models: $x = f_\theta(z), \quad p(x) = p(z) \left| \det \left(\nabla_z f_\theta \right) \right|^{-1}$
  - Pros: Exact likelihoods, easy to train by maximum likelihood
  - Cons: Must cripple layers to maintain tractability, need huge models
- Implicit models (GANs): $x = f_\theta(z), \quad p(x) \approxeq D(x)$
  - Pros: Cheap to sample, no factorization, can use any architecture
  - Cons: Hard to train, likelihood not available, not easy to invert

- EBMs
  - Pros: Flexible, unrestricted architectures, natural for physical models, biology
  - Cons: intractable to compute likelihood, sample, hard to train, surrogate objectives are not well understood, lots more…

Conclusion

- We have introduced the main approaches today for Generative modelling and discussed their benefits and trade-offs