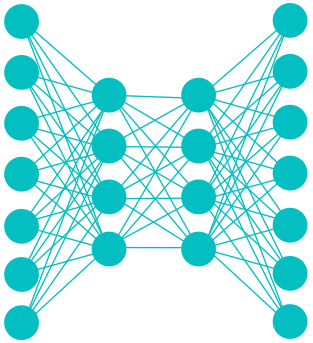
Lecture Notes for

Neural Networks and Machine Learning

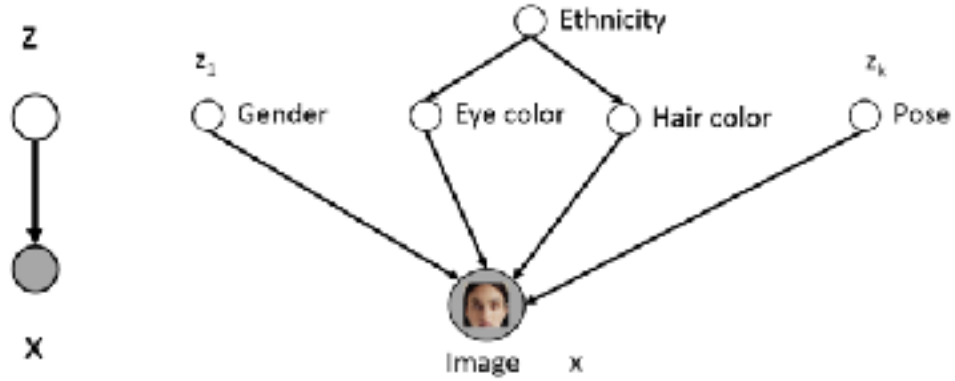
Generative Networks



Logistics and Agenda

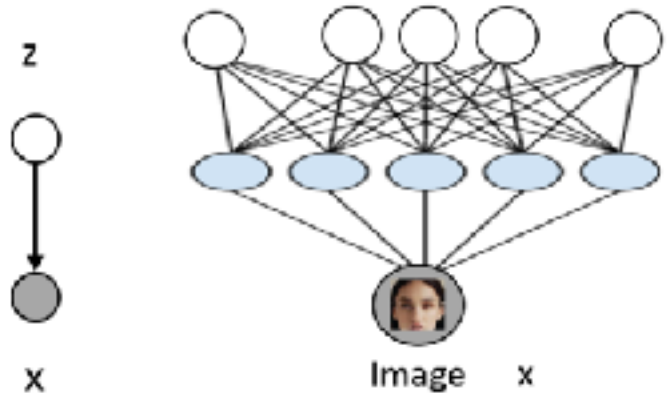
- Logistics
  - Lab Three is Posted
- Agenda
  - A historical perspective of generative Neural Networks
  - Variational Auto-Encoding
  - VAE in Keras Demo

Motivations: Latent Variables



$$p(\mathbf{x} \mid \mathbf{z})$$

Hard: **z** is expertly chosen



$$p(\mathbf{x} \mid \mathbf{z})$$

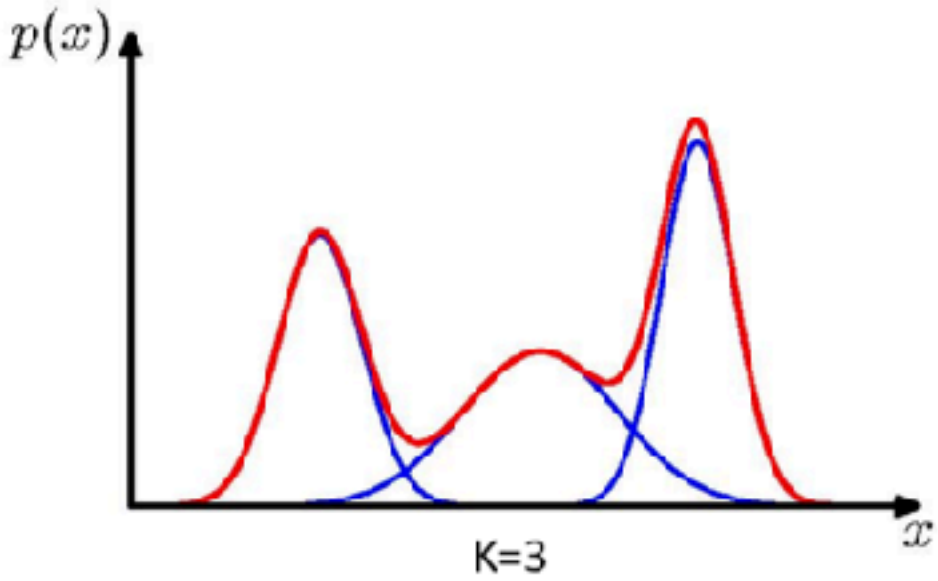
Not as Hard: **z** is trained, latent variables are uncontrolled

Want: $$p(\mathbf{x}) \approx \sum_{\mathbf{z}} p(\mathbf{z}) p_\theta(\mathbf{x} \mid \mathbf{z})$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Motivation: Mixtures for Simplicity

Want: $$p(\mathbf{x}) \approx \sum_{\mathbf{z}} p(\mathbf{z}) p_\theta(\mathbf{x} \mid \mathbf{z})$$



- Each latent variable mostly independent of other latent variables
- The sum of various mixtures can approximate most any ditribution
- Good choice for conditional is Normal Distribution
- Can parameterize $p(x|z)$ to be a Neural Network

$$p_\theta(\mathbf{x} \mid \mathbf{z} = k) = \mathcal{N}(\mathbf{x}, \mu_k, \Sigma_k)$$

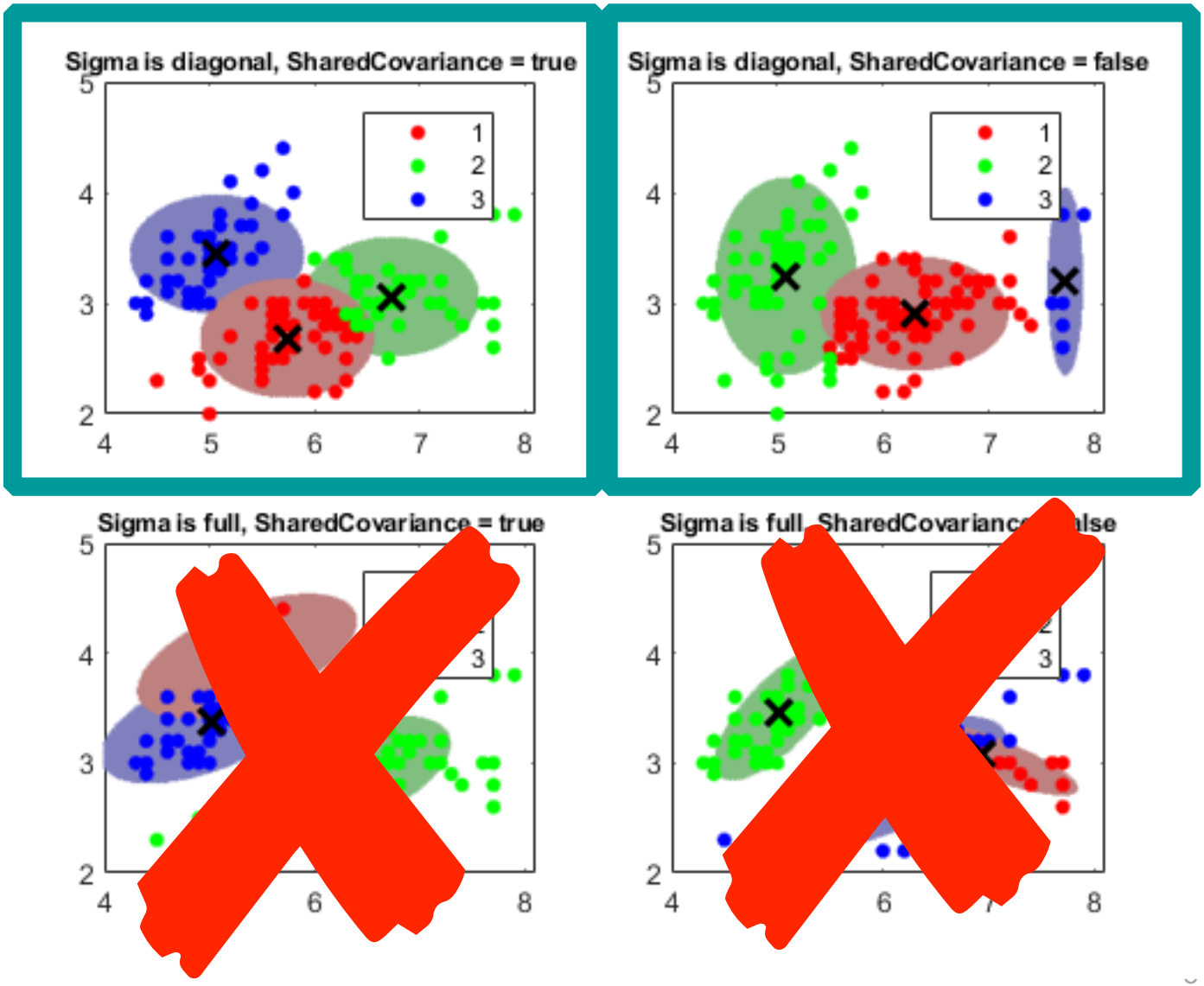
mean and covariance learned

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

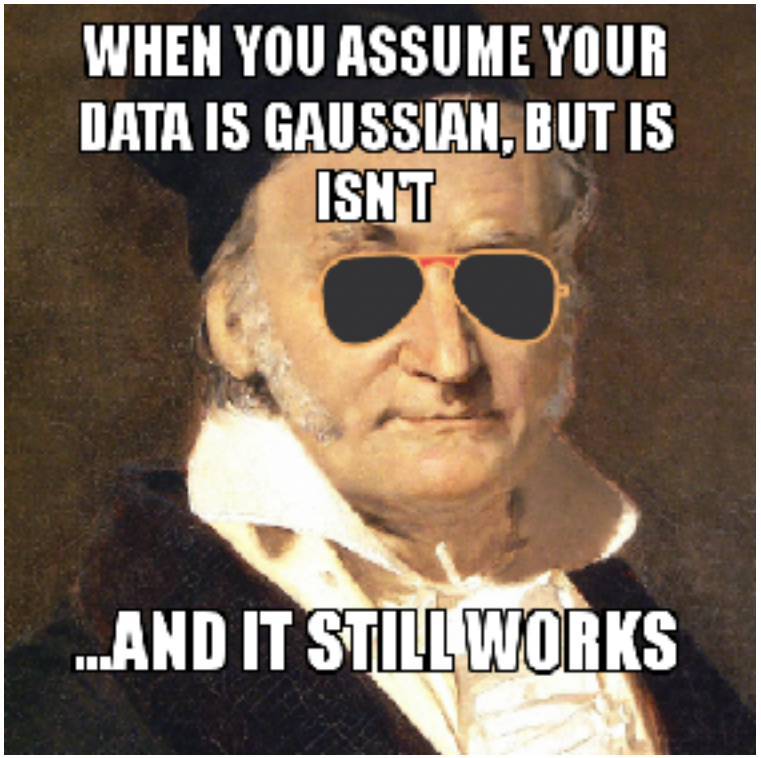
Motivation: Mixtures for Simplicity

$$= \mathcal{N}(\mathbf{x}, \mu_k, \Sigma_k)$$

mean and covariance learned



A History of Generative Networks



Taxonomy of Generative Models

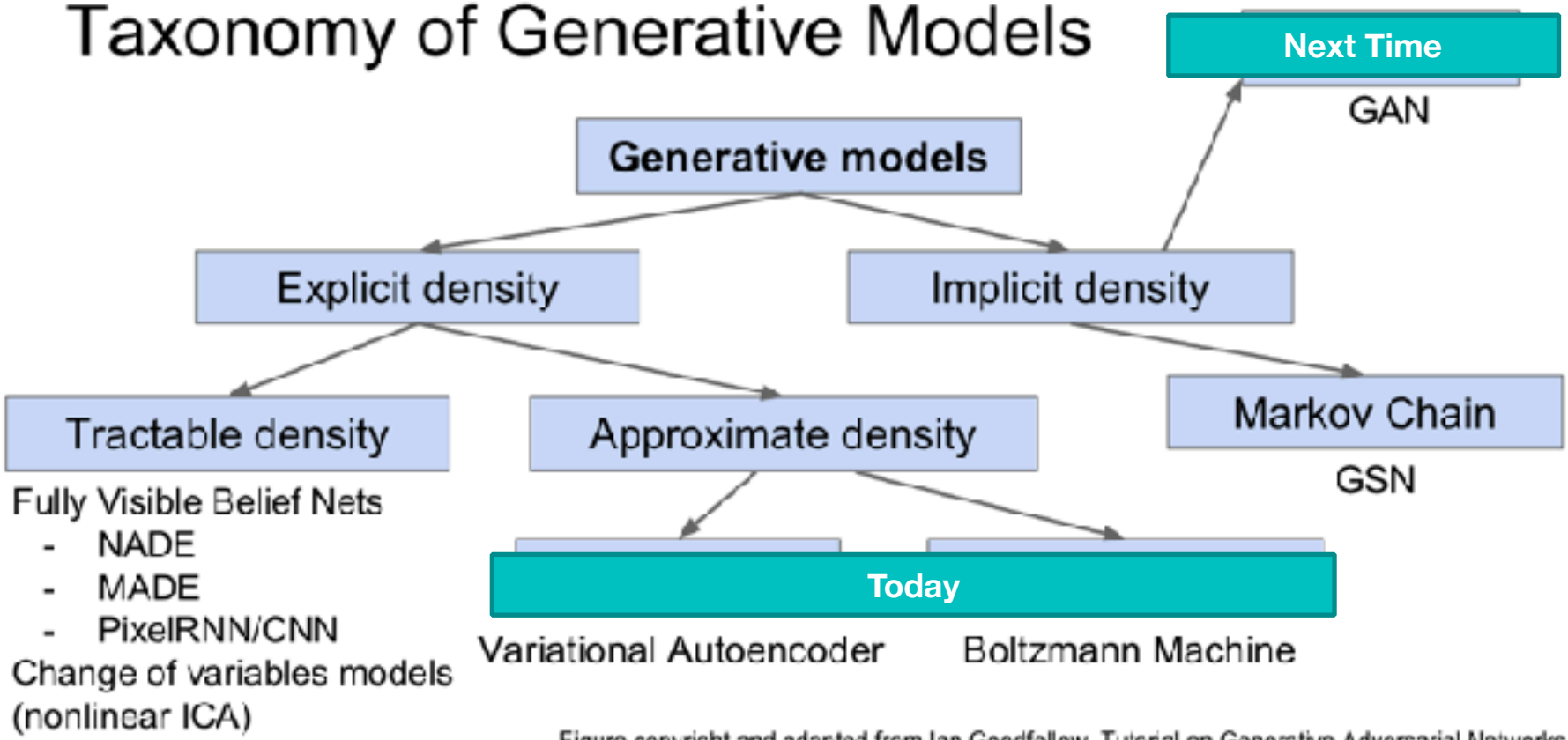


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Deep Generative Models, 2006

- Deep Belief Networks (DBNs)
- Iterative Layer Training (not feed forward)

energy function of the RBM

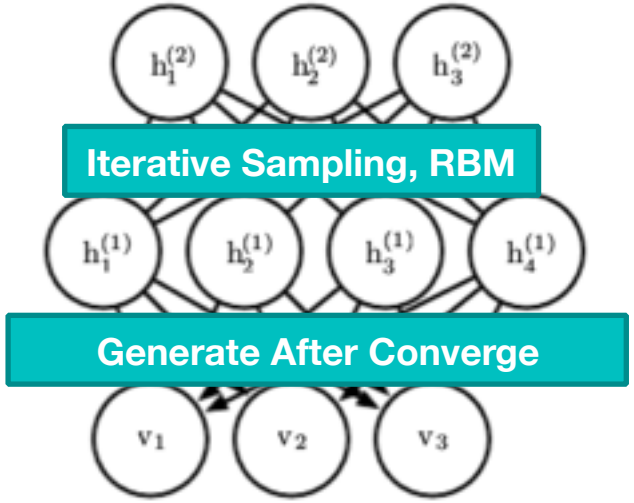
$$P(\boldsymbol{h}^{(l)}, \boldsymbol{h}^{(l-1)}) \propto \exp\left(\boldsymbol{b}^{(l)\top}\boldsymbol{h}^{(l)} + \boldsymbol{b}^{(l-1)\top}\boldsymbol{h}^{(l-1)} + \boldsymbol{h}^{(l-1)\top}\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l)}\right), \quad (20.17)$$

$$P(h_i^{(k)} = 1 \mid \boldsymbol{h}^{(k+1)}) = \sigma\left(b_i^{(k)} + \boldsymbol{W}_{:,i}^{(k-1)\top}\boldsymbol{h}^{(k+1)}\right) \forall i, \forall k \in 1, \ldots, l-2, \quad (20.18)$$

$$P(v_i = 1 \mid \boldsymbol{h}^{(1)}) = \sigma\left(b_i^{(0)} + \boldsymbol{W}_{:,i}^{(1)\top}\boldsymbol{h}^{(1)}\right) \forall i. \quad (20.19)$$

In the case of real-valued visible units, substitute

$$\mathbf{v} \sim \mathcal{N}\left(v; \boldsymbol{b}^{(0)} + \boldsymbol{W}^{(1)\top}h^{(1)}, \boldsymbol{\beta}^{-1}\right) \quad (20.20)$$



$$\mathbf{E}_{v \leftarrow p}[\log p(v)]$$

$p(v)$ is intractable

$$\mathbf{E}_{v \leftarrow p}\left[\mathbf{E}_{h^{(1)} \leftarrow p^{(1)}(h^{(1)}|v)}[\log p^{(2)}(h^{(1)})]\right]$$

optimize with constrastive divergence

i.e., approximation of EM, not Back Prop

To train a deep belief network, one begins by training an RBM to maximize $\mathbb{E}_{\mathbf{v} \sim p_{\text{data}}} \log p(\boldsymbol{v})$ using contrastive divergence or stochastic maximum likelihood. The parameters of the RBM then define the parameters of the first layer of the DBN. Next, a second RBM is trained to approximately maximize

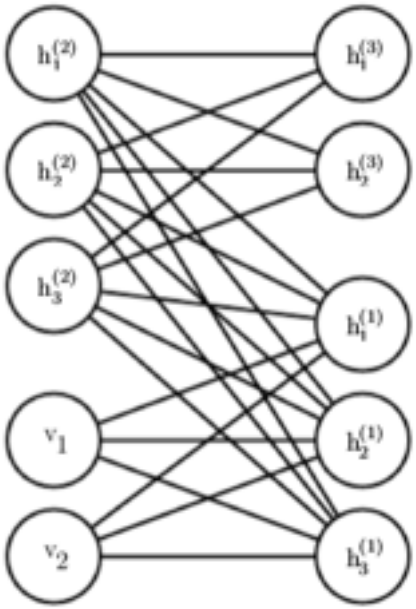
$$\mathbb{E}_{\mathbf{v} \sim p_{\text{data}}} \mathbb{E}_{\mathbf{h}^{(1)} \sim p^{(1)}(\boldsymbol{h}^{(1)}|\boldsymbol{v})} \log p^{(2)}(\boldsymbol{h}^{(1)}) \quad (20.21)$$

where $p^{(1)}$ is the probability distribution represented by the first RBM and $p^{(2)}$ is the probability distribution represented by the second RBM. In other words, the second RBM is trained to model the distribution defined by sampling the hidden units of the first RBM, when the first RBM is driven by the data. This

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

Generative Models, 2009

- Deep Boltzmann Machine



$$P\left(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}\right) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(-E(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}; \boldsymbol{\theta})\right). \tag{20.24}$$

To simplify our presentation, we omit the bias parameters below. The DBM energy function is then defined as follows:

$$E(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}; \boldsymbol{\theta}) = -\boldsymbol{v}^\top \boldsymbol{W}^{(1)} \boldsymbol{h}^{(1)} - \boldsymbol{h}^{(1)\top} \boldsymbol{W}^{(2)} \boldsymbol{h}^{(2)} - \boldsymbol{h}^{(2)\top} \boldsymbol{W}^{(3)} \boldsymbol{h}^{(3)}. \tag{20.25}$$

We now develop the mean field approach for the example with two hidden layers. Let $Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v})$ be the approximation of $P(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v})$. The mean field assumption implies that

$$Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v}) = \prod_j Q(h_j^{(1)} \mid \boldsymbol{v}) \prod_k Q(h_k^{(2)} \mid \boldsymbol{v}). \tag{20.29}$$

Not tractable: Can only optimize the Evidence lower bound, ELBO

Approximate via MCMC like **Gibbs Sampling**

$$\mathrm{KL}(Q \| P) = \sum_{\boldsymbol{h}} Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v}) \log\left(\frac{Q(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v})}{P(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} \mid \boldsymbol{v})}\right). \tag{20.30}$$

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

Image Generation from Samples

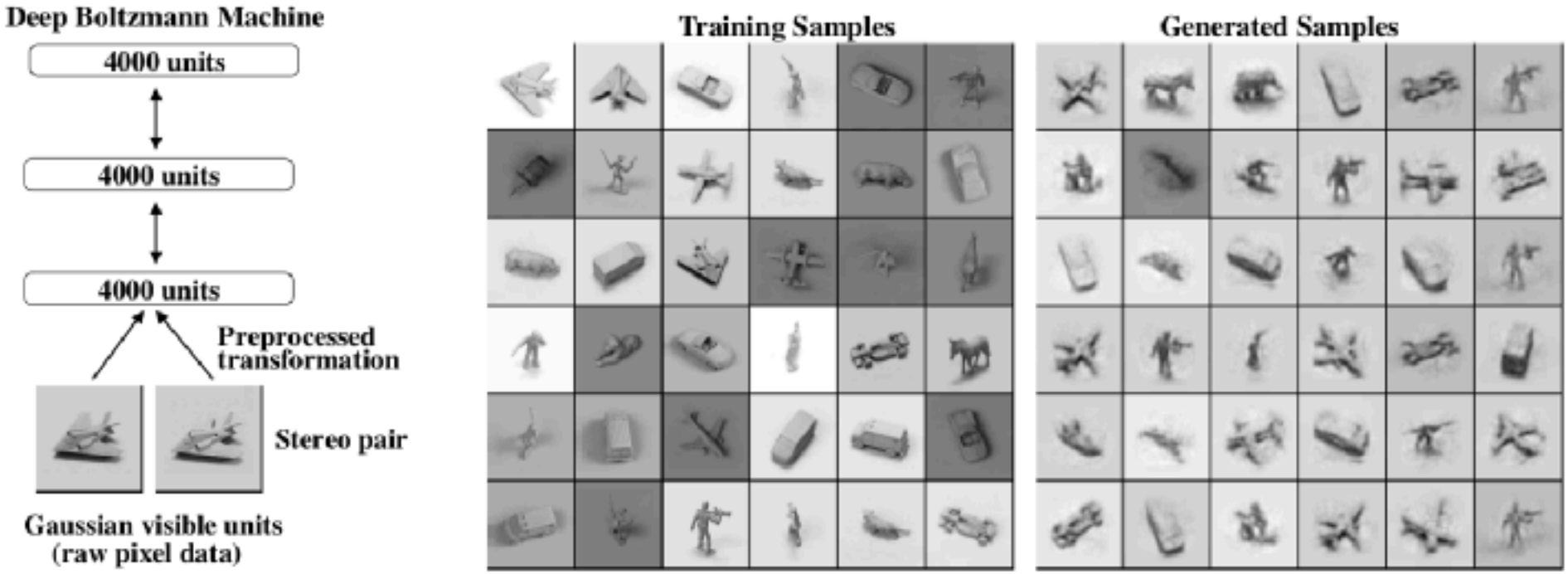


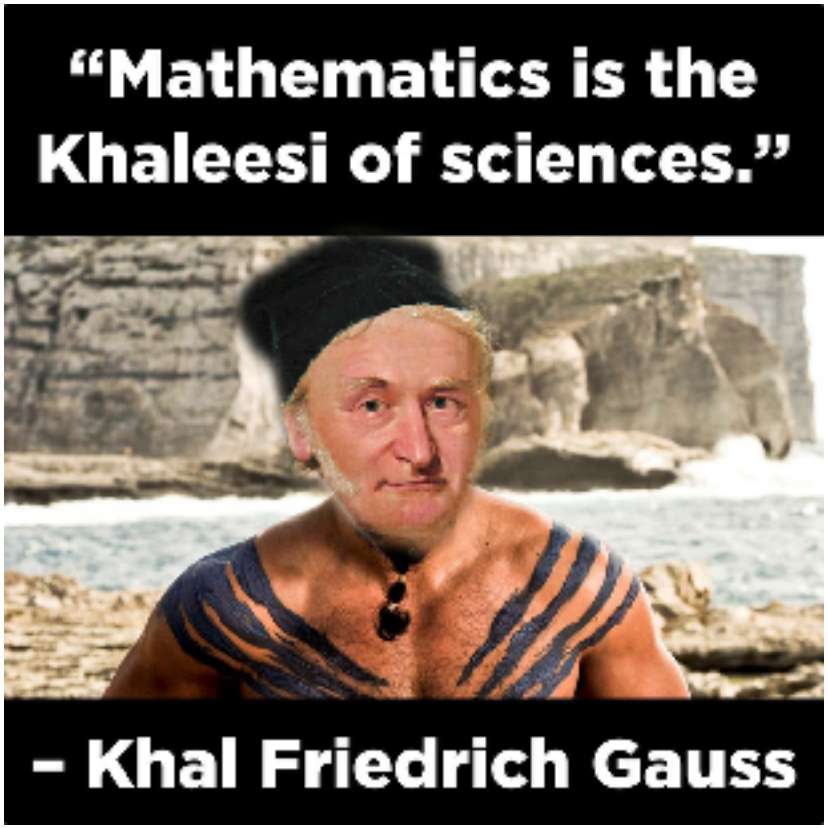
Figure 5: **Left:** The architecture of deep Boltzmann machine used for NORB. **Right:** Random samples from the training set, and samples generated from the deep Boltzmann machines by running the Gibbs sampler for 10,000 steps.

Salakhutdinov, Ruslan, and Geoffrey Hinton. "Deep boltzmann machines." In Artificial intelligence and statistics, pp. 448-455. 2009.

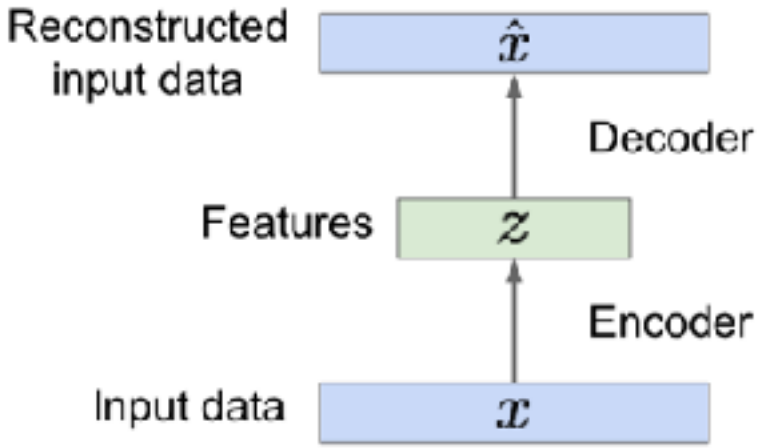
Contemporary Modeling

- DBNs and DBMs have mostly been abandoned
  - Mathematics detracts from popular understanding
  - Often methods using sampling are not scalable
  - Cannot directly use Gradients (no Back Prop) 😢
- Evidence Lower Bound (ELBO) considered "good enough" because … we can't do better computationally
- Popular method for calculating generative networks with ELBO approximation:
  - Variational Auto Encoding
    - Guaranteed NOT to find global minimum
    - But scalable and will converge in finite time

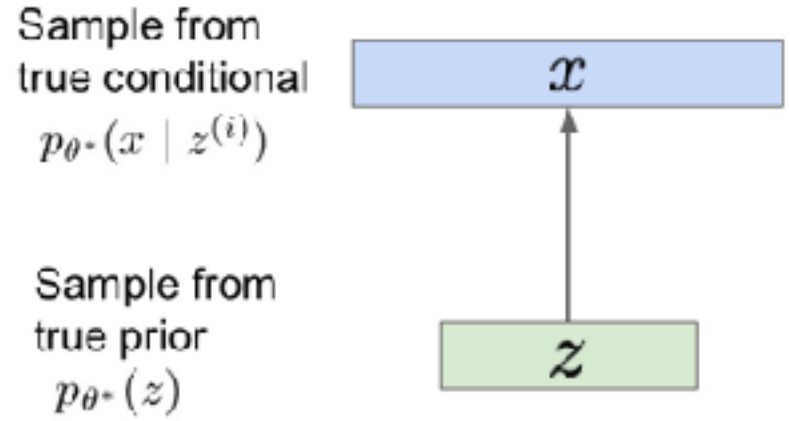
Variational Auto Encoding



Aside: Auto Encoding



Once trained, is it possible to generate data?



- Does this work for simple auto encoding?
  - Nope.
- Learned space is not continuous
- How to sample from the latent space?
- Features could be highly correlated
- Need to define some constraints on the latent space…

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

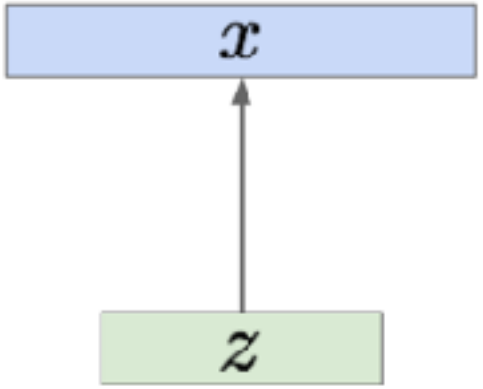
Reasonable constraints for p(z)?

Sample from true conditional

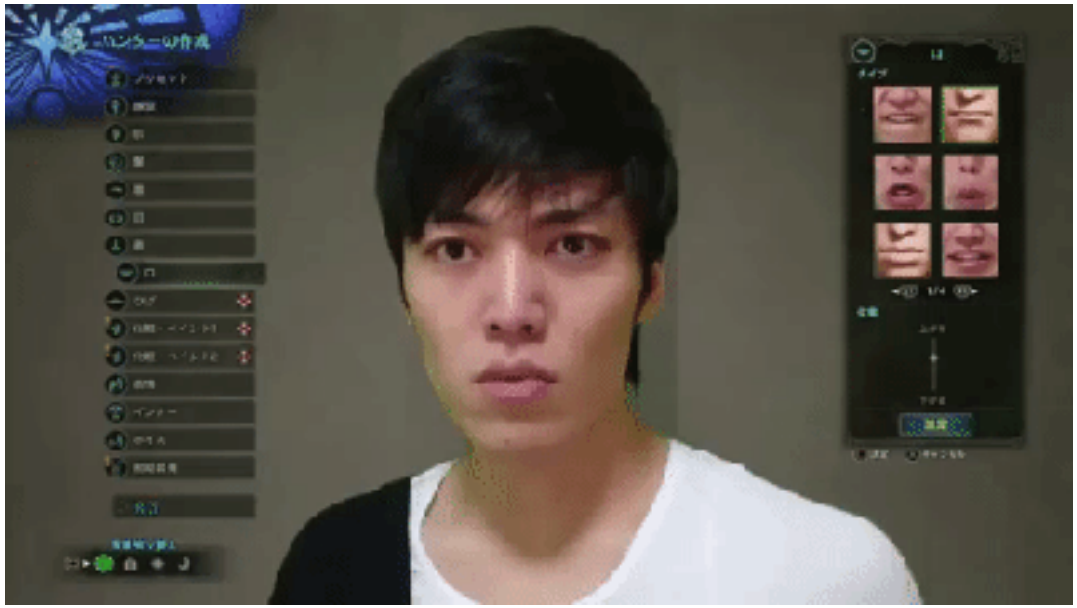
$p_{\theta^*}(x \mid z^{(i)})$

Sample from true prior

$p_{\theta^*}(z)$



- Should be simple, easily to sample from: Normal
- Each component should be independent: Covariance
  - Encourages features that are semantic



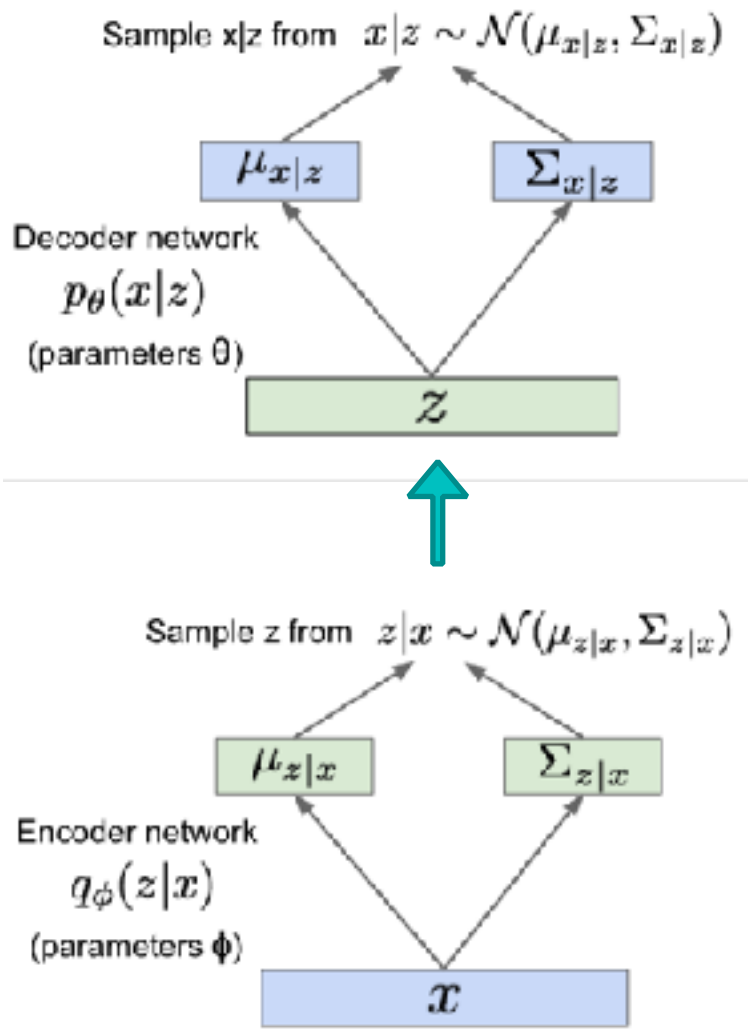
Optimizing

- We want generated samples from latent space to be as close as possible to the true p(x)…
- How can we optimize this?

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

- We can't! Intractable computation for every "z"
- So let's define this with variational inference:
  - Only needs to work for z with observed $x^{(i)}$
  - Encode observed $x^{(i)}$ using network q,
    - so $q = p(z \mid x^{(i)})$
  - Optimize decoder to minimize reconstruction

Need a new formulation



if we optimize **z**, this is MLE

$$\log p(x^{(i)}) = \mathbf{E}_{\mathbf{z} \leftarrow q(z|x)} \left[\log p(x^{(i)}) \right]$$

$$= \sum_{z \in \mathcal{Z} | x^{(i)}} q(z \,|\, x^{(i)}) \cdot \log p(x^{(i)})$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Need a new formulation

$$\log p(x^{(i)}) = \mathbf{E}_{\mathbf{z} \leftarrow q(z|x)} \left[\log p(x^{(i)}) \right]$$

$$= \mathbf{E}_{\mathbf{z}} \left[\log \frac{p(x^{(i)} | z) p(z)}{p(z | x^{(i)})} \frac{q(z | x^{(i)})}{q(z | x^{(i)})} \right]$$

$$= \mathbf{E}_{\mathbf{z}} \left[\log p(x^{(i)} | z) \right] + \mathbf{E}_{\mathbf{z}} \left[\log \frac{p(z)}{q(z | x^{(i)})} \right] + \mathbf{E}_{\mathbf{z}} \left[\log \frac{q(z | x^{(i)})}{p(z | x^{(i)})} \right]$$

$$= \mathbf{E}_{\mathbf{z}} \left[\log p(x^{(i)} | z) \right] - \mathbf{E}_{\mathbf{z}} \left[\log \frac{q(z | x^{(i)})}{p(z)} \right] + \mathbf{E}_{\mathbf{z}} \left[\log \frac{q(z | x^{(i)})}{p(z | x^{(i)})} \right]$$

$$= \mathbf{E}_{\mathbf{z}} \left[\log p(x^{(i)} | z) \right] - D_{KL} \left[q(z | x^{(i)}) || p(z) \right] + D_{KL} \left[q(z | x^{(i)}) || p(z | x^{(i)}) \right]$$

$$\log p(x^{(i)}) \geq \mathbf{E}_{\mathbf{z}} \left[\log p(x^{(i)} | z) \right] - D_{KL} \left[q(z | x^{(i)}) || p(z) \right]$$ Maximize Lower Bound

What have we really done here? Is this still MLE?

The Loss Function

Maximize through
Error of Reconstruction
This is just negative cross entropy

Here we
assume p(z) is Normal
because we like Normal

$$D_{KL}[N(\mu(X),\Sigma(X))\|N(0,1)] = \frac{1}{2}\left(\mathrm{tr}(\Sigma(X)) + \mu(X)^T\mu(X) - k - \log\det(\Sigma(X))\right)$$

$$D_{KL}[N(\mu(X),\Sigma(X))\|N(0,1)] = \frac{1}{2}\left(\sum_k \Sigma(X) + \sum_k \mu^2(X) - \sum_k 1 - \log\prod_k \Sigma(X)\right)$$

$$= \frac{1}{2}\left(\sum_k \Sigma(X) + \sum_k \mu^2(X) - \sum_k 1 - \sum_k \log\Sigma(X)\right)$$

$$\geq \mathbf{E}_{\mathbf{z}}\left[\log p(x^{(i)}\,|\,z)\right] - D_{KL}\left[q(z\,|\,x^{(i)})\,||\,p(z)\right]$$

$$= \frac{1}{2}\sum_k \left(\Sigma(X) + \mu^2(X) - 1 - \log\Sigma(X)\right)$$

https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/

The Loss Function

$$\geq \mathbf{E}_{\mathbf{z}}\left[\log p(x^{(i)} \mid z)\right] - D_{KL}\left[q(z \mid x^{(i)}) \mid\mid p(z)\right]$$

Maximize through
Error of Reconstruction
This is just negative cross entropy

Here we
assume p(z) is Normal
because we like Normal

$$= \frac{1}{2}\sum_{k}\left(\Sigma(X) + \mu^2(X) - 1 - \log \Sigma(X)\right)$$

covariance is not numerically stable because of underflow

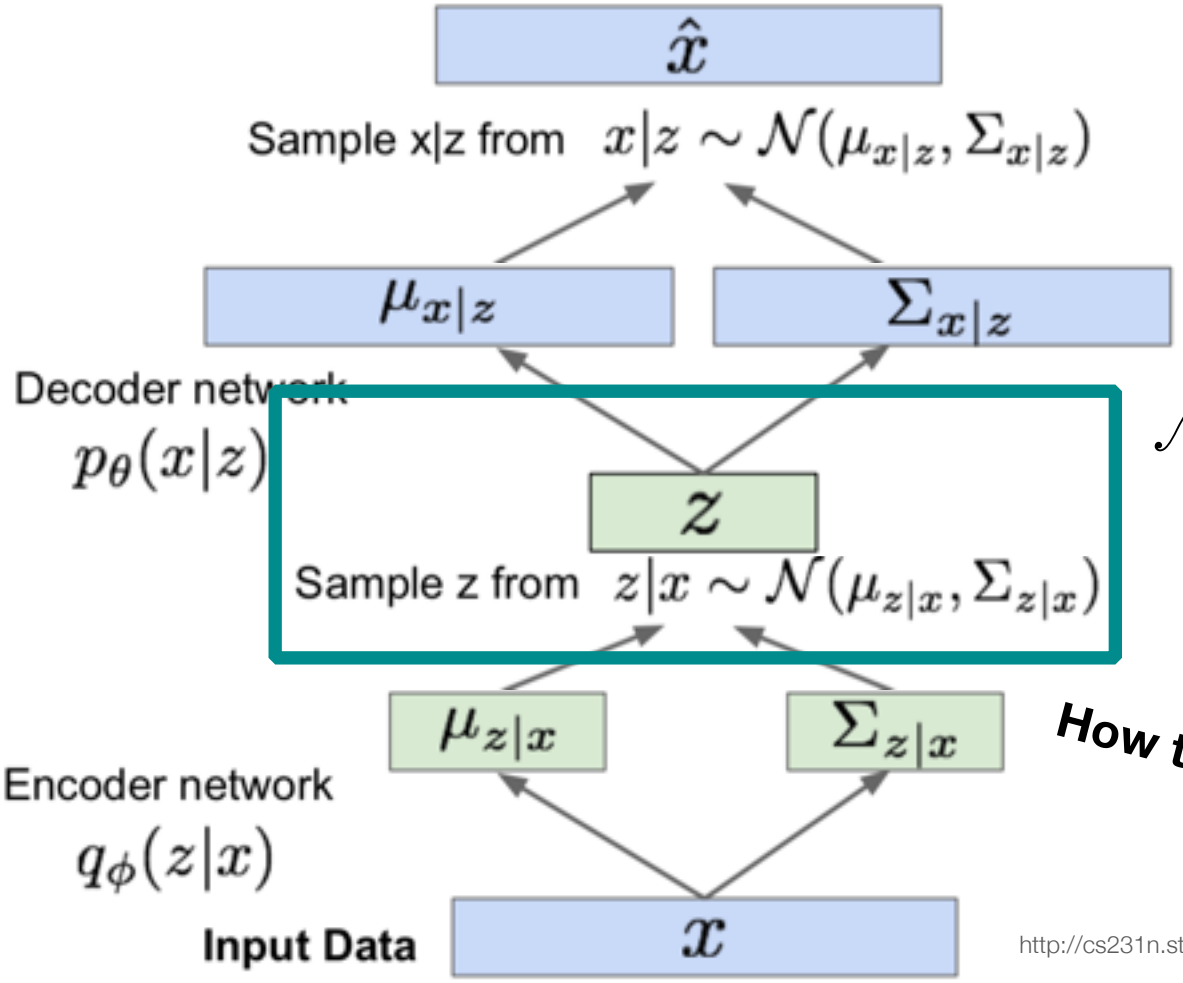
$$D_{KL}[N(\mu(X), \Sigma(X)) \| N(0,1)] = \frac{1}{2}\sum_{k}\left(\exp(\Sigma(X)) + \mu^2(X) - 1 - \Sigma(X)\right)$$

so we actually optimize log variance
(remember we assume diagonal covariance)

https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/

Back Propagating

$$\geq \mathbf{E}_{\mathbf{z}}\left[\log p(x^{(i)}|z)\right] - D_{KL}\left[q(z|x^{(i)})||p(z)\right]$$



This is partially differentiable by chain rule…

$$\mathcal{N}(\mu, \Sigma) = z$$
$$= \mu(x^{(i)}) + \Sigma(x^{(i)}) \cdot \mathcal{N}(0,1)$$

How to back propagate?

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

The Loss Function Implementation

```
# Encode the input into a mean and variance parameter
z_mean, z_log_variance = encoder(input_img)

# Draw a latent point using a small random epsilon
z = z_mean + exp(z_log_variance) * epsilon

# Then decode z back to an image
reconstructed_img = decoder(z)

# Instantiate a model
model = Model(input_img, reconstructed_img)
```

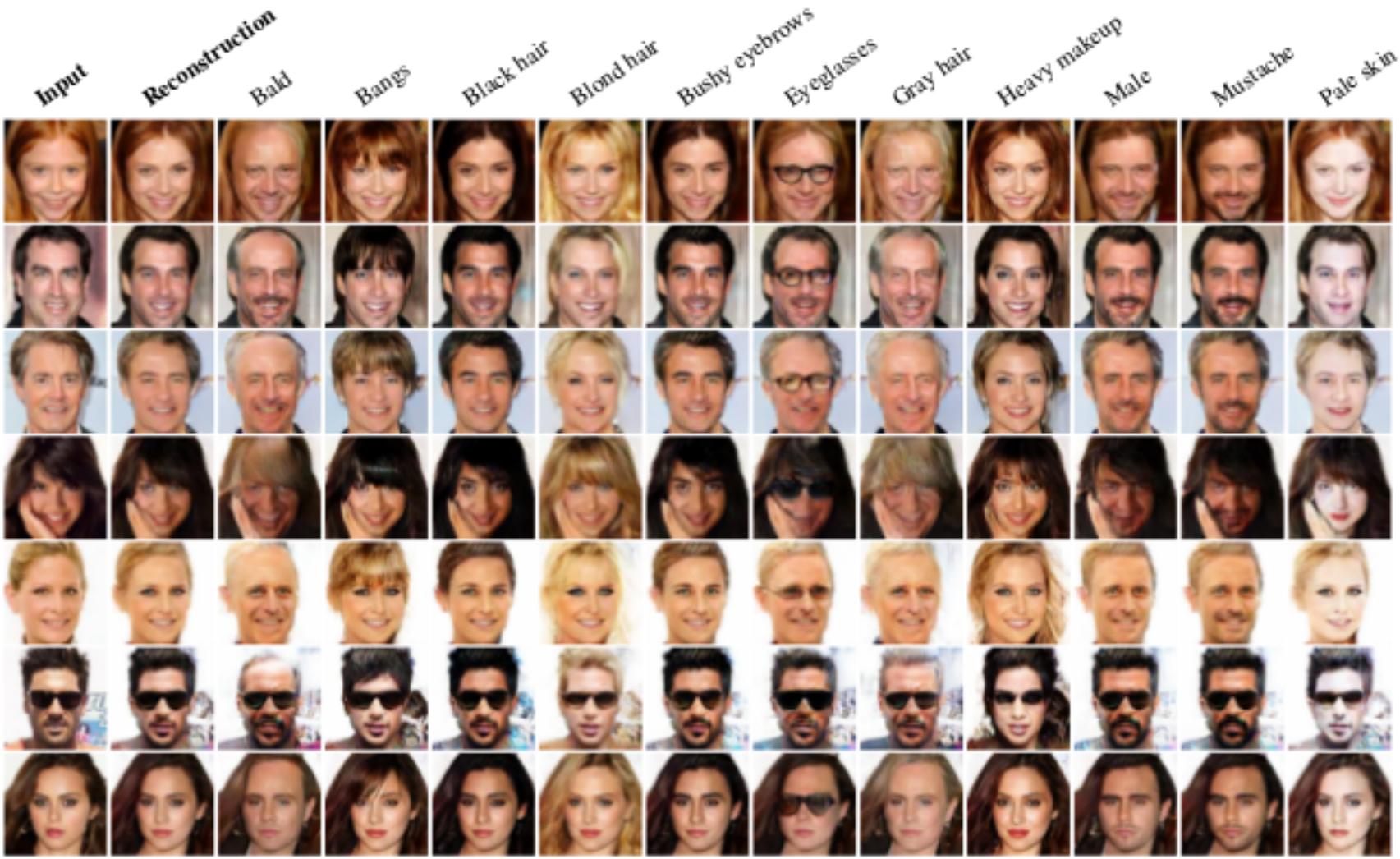
$$\mathcal{N}(\mu, \Sigma) = z$$
$$= \mu(x^{(i)}) + \Sigma(x^{(i)}) \cdot \mathcal{N}(0,1)$$

```
def vae_loss(self, x, z_decoded):
    x = K.flatten(x)
    z_decoded = K.flatten(z_decoded)
    xent_loss = keras.metrics.binary_crossentropy(x, z_decoded)
    kl_loss = -5e-4 * K.mean(
        1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
    return K.mean(xent_loss + kl_loss)
```

$$\mathbf{E}_{\mathbf{z}}\left[\log p(x^{(i)} \mid z)\right] - D_{KL}\left[q(z \mid x^{(i)}) \mid\mid p(z)\right]$$
$$= \mathbf{E}_{\mathbf{z}}\left[\log p(x^{(i)} \mid z)\right] - \sum_{k} \exp(\Sigma(x^{(i)}) + \mu(x^{(i)})^2 - 1 - \Sigma(x^{(i)})$$

VAE Examples

VAE Examples



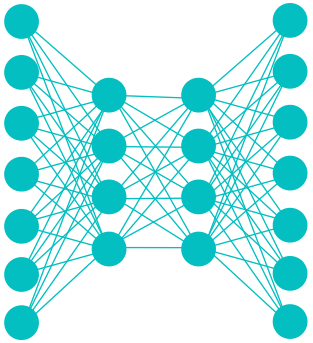
Lecture Notes for

Neural Networks and Machine Learning

Generative Networks

Next Time:
GANs

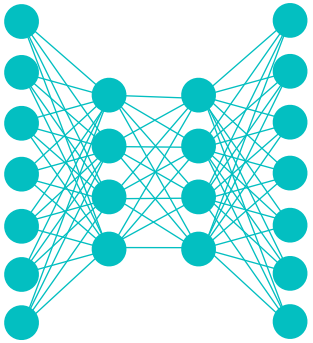
Reading: Chollet CH8



Lecture Notes for

Neural Networks and Machine Learning

Generative Adversarial Networks



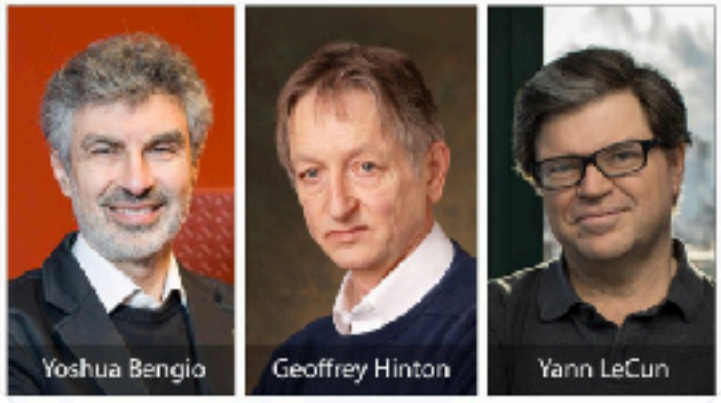
Logistics and Agenda

- Logistics
  - None!
  - How is the ChEMBL filtering going?
- Agenda
  - Working with ChEMBL
  - VAE Demo
  - Simple Generative Adversarial Networks

Working with ChEMBL

Official ACM @TheOfficialACM

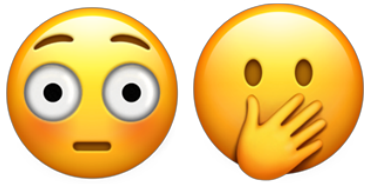
Yoshua Bengio, Geoffrey Hinton and Yann LeCun, the fathers of #DeepLearning, receive the 2018 #ACMTuringAward for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing today. bit.ly/2HVJtdV



Olivier Grisel liked

Volodymyr Kuleshov @volkuleshov · 11h

Congratulations! I hope Jurgen Schmidhuber is not too upset that he wasn't included. Could lead to some awkward questions at the award ceremony ;)



Working with ChEMBL Tables

- **Assays**: need assay type to be binding affinity
- **Activities**:
  - Has **IC50** column, **molregno**, and target (**record_id**)
  - Should calculate “top” records here
  - Do filtering here to find needed compounds
- **Compound_Structures**
  - Has **molregno** and **canonical_smiles** representation
- Building final dataset is simply a join operation on calculated fingerprints and binary IC50 results

Working with ChEMBL

activities

activity_id	assay_id	doc_id	record_id	molregno	star	standard_value	standard_units	stan	standard_type
31863	54505	6424	206172	180094	>	100000	nM	1	IC50
31864	83907	6432	208970	182268	=	2500	nM	1	IC50
31865	88152	6432	208970	182268	>	50000	nM	1	IC50
31866	83907	6432	208987	182855	=	9000	nM	1	IC50

assays

assay_id	doc_id	description	assay_type
1	11087	The compou	B
2	684	Compound w	F
3	15453		B
4	17841	Binding affin	B

compound_structures

molregno	mo	sta	sta	canonical_smiles
1		InC	OV	Cc1cc(ccc1C(=O)c2ccccc2Cl)N3N=CC(=O)NC3=O
2		InC	ZJ	Cc1cc(ccc1C(=O)c2ccc(cc2)C#N)N3N=CC(=O)NC3=O
3		InC	YO	Cc1cc(cc(C)c1C(O)c2ccc(Cl)cc2)N3N=CC(=O)NC3=O
4		InC	PS	Cc1ccc(cc1)C(=O)c2ccc(cc2)N3N=CC(=O)NC3=O
5		InC	KE	Cc1cc(ccc1C(=O)c2ccc(Cl)cc2)N3N=CC(=O)NC3=O

Back to VAEs

Geoffrey Hinton @geoffreyhinton · 23h

Replying to @JeffDean @ylecun and @TheOfficialACM

Thanks to my graduate students and postdocs whose work won a Turing award. Thanks to my visionary mentors Inman Harvey, David Rumelhart and Terry Sejnowski. And thanks to Jeff Dean for creating the brain team that turns basic research in neural nets into game-changing products.

40 313 2,208

Yann LeCun @ylecun · 20h

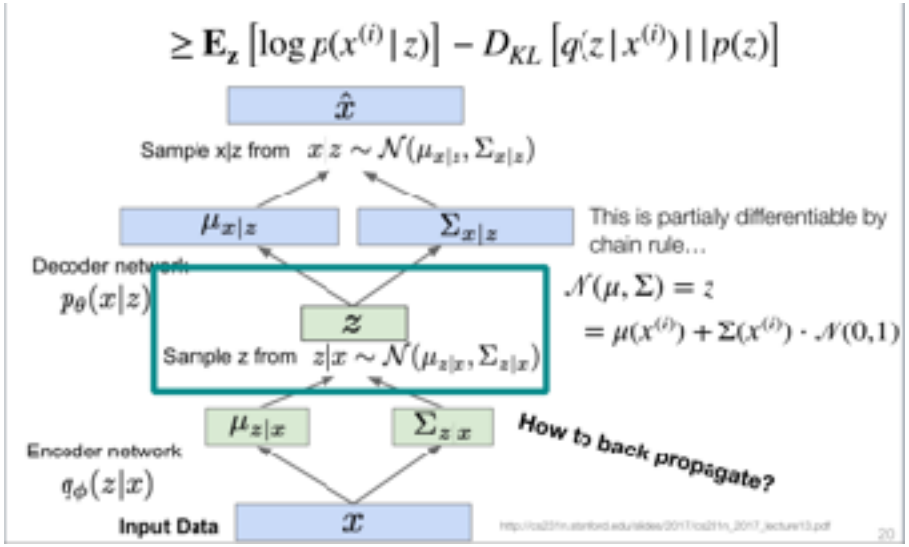
Congratulations Geoff, from one of your former postdocs ;-)

3 11 312

This is why we like Hinton.

Last Time

- Variational auto encoding

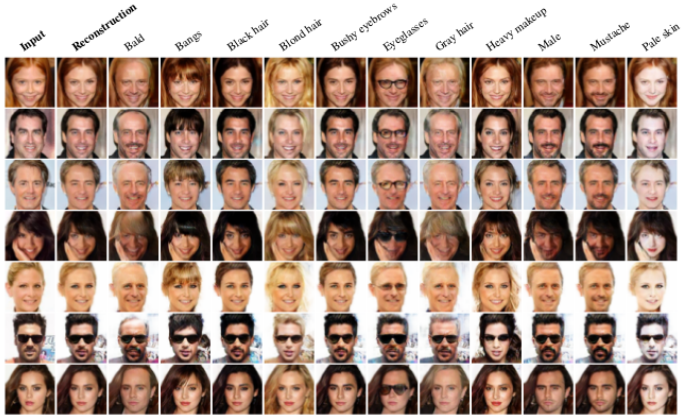



```
# Encode the input into a mean and variance parameter
z_mean, z_log_variance = encoder(input_img)

# Draw a latent point using a small random epsilon
z = z_mean + exp(z_log_variance) * epsilon

# Then decode z back to an image
reconstructed_img = decoder(z)

# Instantiate a model
model = Model(input_img, reconstructed_img)
```

$$\mathcal{N}(\mu, \Sigma) = z$$
$$= \mu(x^{(i)}) + \Sigma(x^{(i)}) \cdot \mathcal{N}(0,1)$$

```
def vae_loss(self, x, z_decoded):
    x = K.flatten(x)
    z_decoded = K.flatten(z_decoded)
    xent_loss = keras.metrics.binary_crossentropy(x, z_decoded)
    kl_loss = -5e-4 * K.mean(
        1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
    return K.mean(xent_loss + kl_loss)
```

$$\mathbf{E}_{\mathbf{z}}\left[\log p(x^{(i)}|z)\right] - D_{KL}\left[q(z|x^{(i)})||p(z)\right]$$
$$= \mathbf{E}_{\mathbf{z}}\left[\log p(x^{(i)}|z)\right] - \sum \exp(\Sigma(x^{(i)}) + \mu(x^{(i)})^2 - 1 - \Sigma(x^{(i)}))$$



VAEs in Keras

Implementation from Book on MNIST

Demo by Francois Chollet

Follow Along: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/8.4-generating-images-with-vaes.ipynb

The Simple GAN

Latent Variable Approximation with GANS

- **Idea**: transform random input data into target of interest
  - Like an image!
- Rather than training an encoder with variational inference, we need a transformation protocol
  - Transformer will be a…
  - Neural Network (of course!)
- Transformer is a “complex” sampling method
- How to optimize and provide training examples?
  - Use two Neural Networks!
  - … Deep Learning is like the chiropractory of the machine learning…

Generative Adversarial Network

- Generator: $x = g_w(z)$
- Discriminator: $\{0,1\} = d_w(x)$
- Mini-max, turn-based game:

$$\hat{w} = \arg \min_{g} \max_{d} v(g,d)$$

- Nice differentiable choice for v (zero sum game):

$$v(g,d) = \mathbf{E}_{x \leftarrow p_{data}} \left[\log d(x)\right] + \mathbf{E}_{x \leftarrow g(z)} \left[\log(1 - d(x))\right]$$

only portion dependent on g
generator minimizes

everything dependent on d
discriminator maximizes

Taking turns

$$v(g,d) = \mathbf{E}_{x \leftarrow p_{data}}\left[\log d(x)\right] + \mathbf{E}_{x \leftarrow g(z)}\left[\log(1 - d(x))\right]$$

$$\hat{w} = \arg\min_{g}\max_{d} v(g,d)$$

- If only the problem was convex! This would be easy to solve…
- But v(g,d) is not convex, not even close
  - Saddle points, saddle points everywhere
- Taking turns on the gradient descent will probably never reach equilibrium
  - There is no convergence guarantee
  - But practically we like when discriminator == chance

Practical Objectives

- Discriminator objective, gradient ascent:

$$\max_d \mathbf{E}_{x \leftarrow p_{data}} \left[\log d(x)\right] + \mathbf{E}_{x \leftarrow g(z)} \left[\log(1 - d(x))\right]$$

- Generator objective, gradient descent:

$$\min_g \mathbf{E}_{x \leftarrow g(z)} \left[\log(1 - d(x))\right]$$

- But **practically** generator is really hard to train, amplified by a decreasing gradient
- Goodfellow tried to solve this mathematically through a number of different formulations
  - Nothing seemed to work, and then…

Practical Objectives

- Original Generator objective, gradient descent:

$$\min_g \mathbf{E}_{x \leftarrow g(z)} \left[\log(1 - d(x)) \right]$$

- Goodfellow et al. gave up on using rigorous mathematics (intractable) and relied on heuristic:
  - Instead of minimizing when discriminator is right
  - Why not maximize when its wrong?
    - not trying to win, just make the other person lose

$$\max_g \mathbf{E}_{x \leftarrow g(z)} \left[\log(d(x)) \right]$$ **No longer a zero sum game?!**

Final Loss Functions

- Discriminator:

$$\max_{d} \mathbf{E}_{x \leftarrow p_{data}} \left[\log d(x)\right] + \mathbf{E}_{x \leftarrow g(z)} \left[\log(1 - d(x))\right]$$

Same as minimizing the binary cross entropy of the model with: (1) labeled data and (2) generated data!

- Generator:

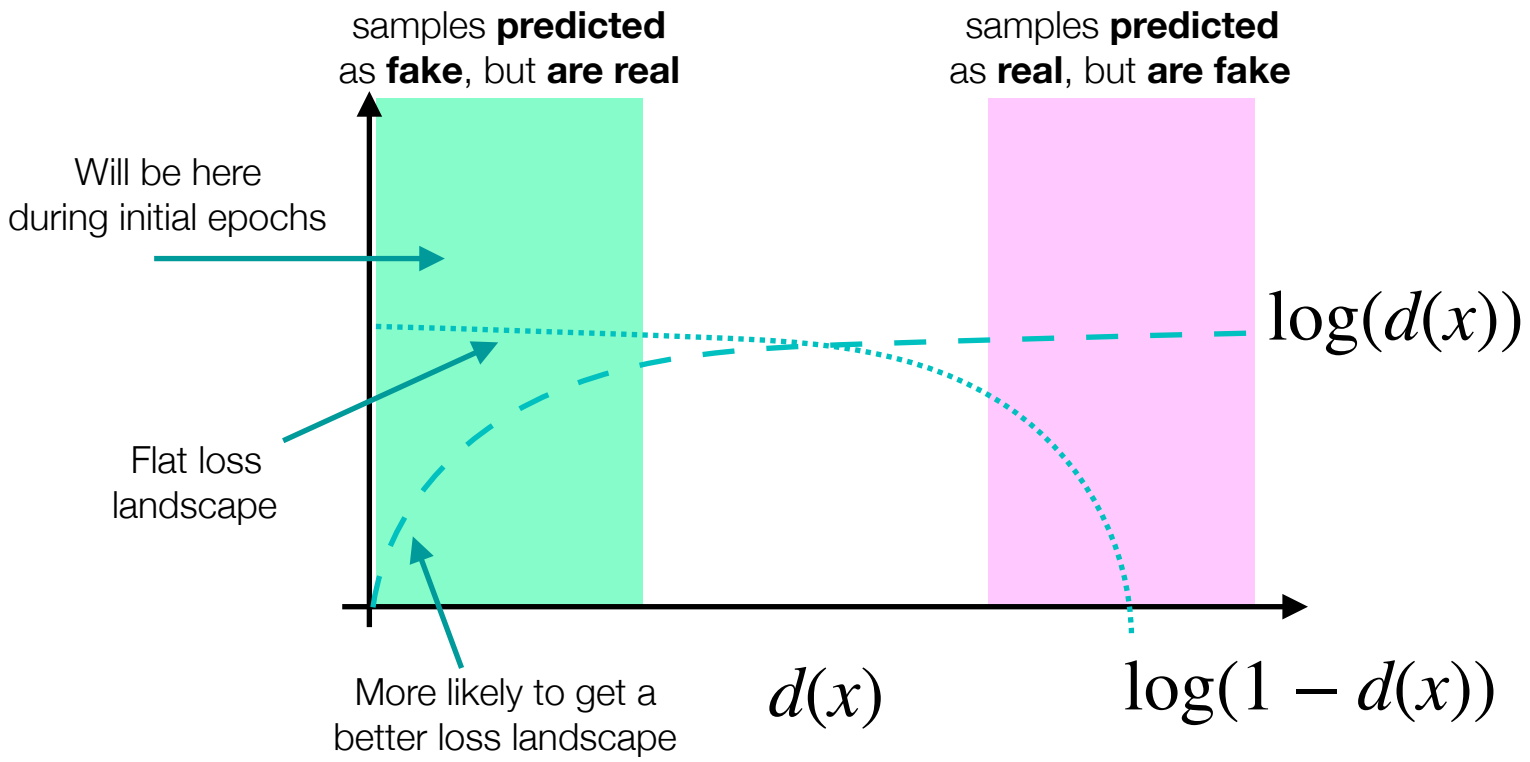
$$\max_{g} \mathbf{E}_{x \leftarrow g(z)} \left[\log(d(x))\right]$$

Freeze Discriminator Weights

Same as minimizing the binary cross entropy of "mislabeled" generated data!

Why maximize incorrect?

- Training two networks is inherently unstable, need heuristic
- Let's assume generator is not any good for initial epochs!



How to Train your (dra) GAN

Every time Ian Goodfellow has a tutorial, It should be called:

"GAN-splaining with Ian"

—Geoffrey Hinton, Probably

Simple Training Approach

- Some caveats on why training will be difficult:
  - The latent space discovered by the Generator has almost no guarantees regarding continuity and structure (different than VAEs)
  - Convergence of GAN is not possible, only equilibrium
    - even saying discriminator is chance is not enough!
  - Each iteration through, the entire loss function changes because generator changes
    - Also we are sampling different points from generator…

Simple Training Approach

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

  end for

  - Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  - Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Does this work?! **Not really! Why?**

Goodfellow et al. “Generative Adversarial Networks” (NeurIPS 2014)

A bag of tricks from F. Chollet

The process of training GANs and tuning GAN implementations is notoriously difficult. There are a number of known tricks you should keep in mind. Like most things in deep learning, it's **more alchemy than science**: **these tricks are heuristics, not theory-backed guidelines**. They're supported by a level of intuitive understanding of the phenomenon at hand, and they're known to work well empirically, although not necessarily in every context.

—Francois Chollet, DL with Python

- Use tanh for generator output, not a sigmoid
  - We typically normalize inputs to a NN to be from [-1, 1], generator needs to mirror this squashing
- Sample from Normal Distribution
  - Everyone else is doing it (even though no assumption of Gaussian in GAN formulation)
- Random is more robust
  - GANs get stuck a lot
- Sparse gradients are not your friend here
  - No max pooling, no ReLU 😳
- Make encoder and decoder symmetric
  - Unequal pixel coverage (checkerboards)

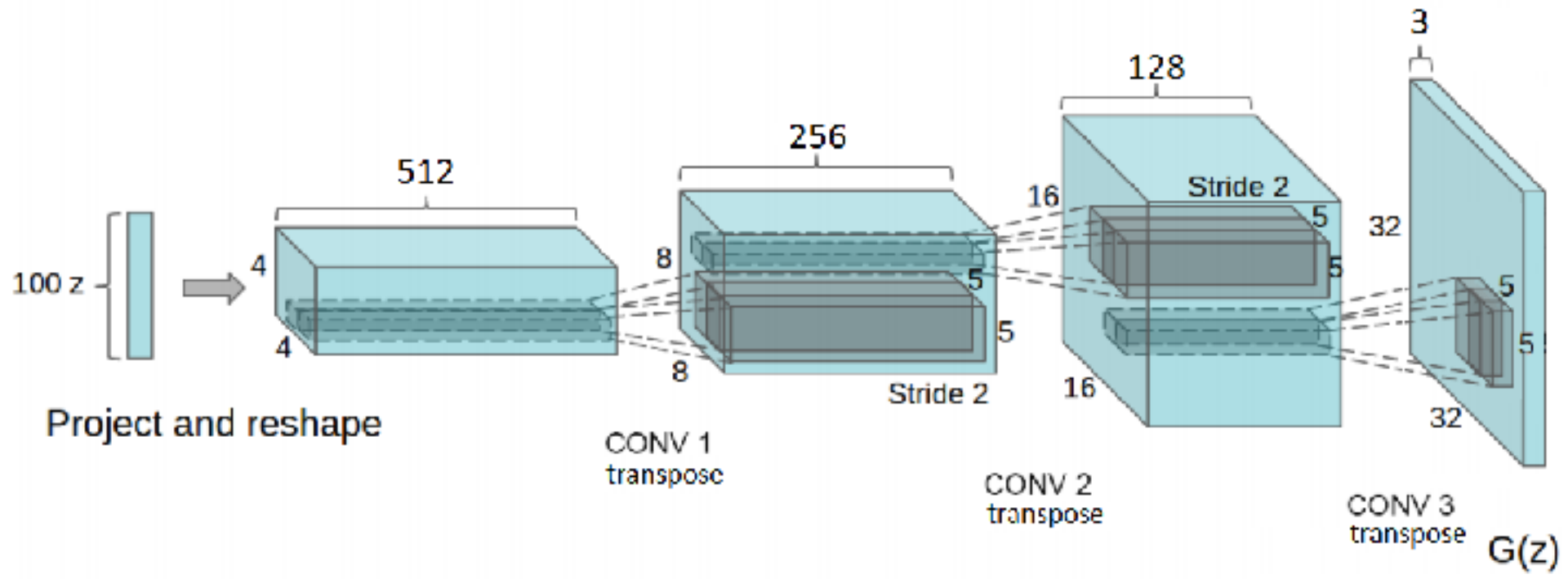
Some Results of Generation

Goodfellow et al. "Generative Adversarial Networks" (NeurIPS 2014)

Highlight: Nearest Training Sample

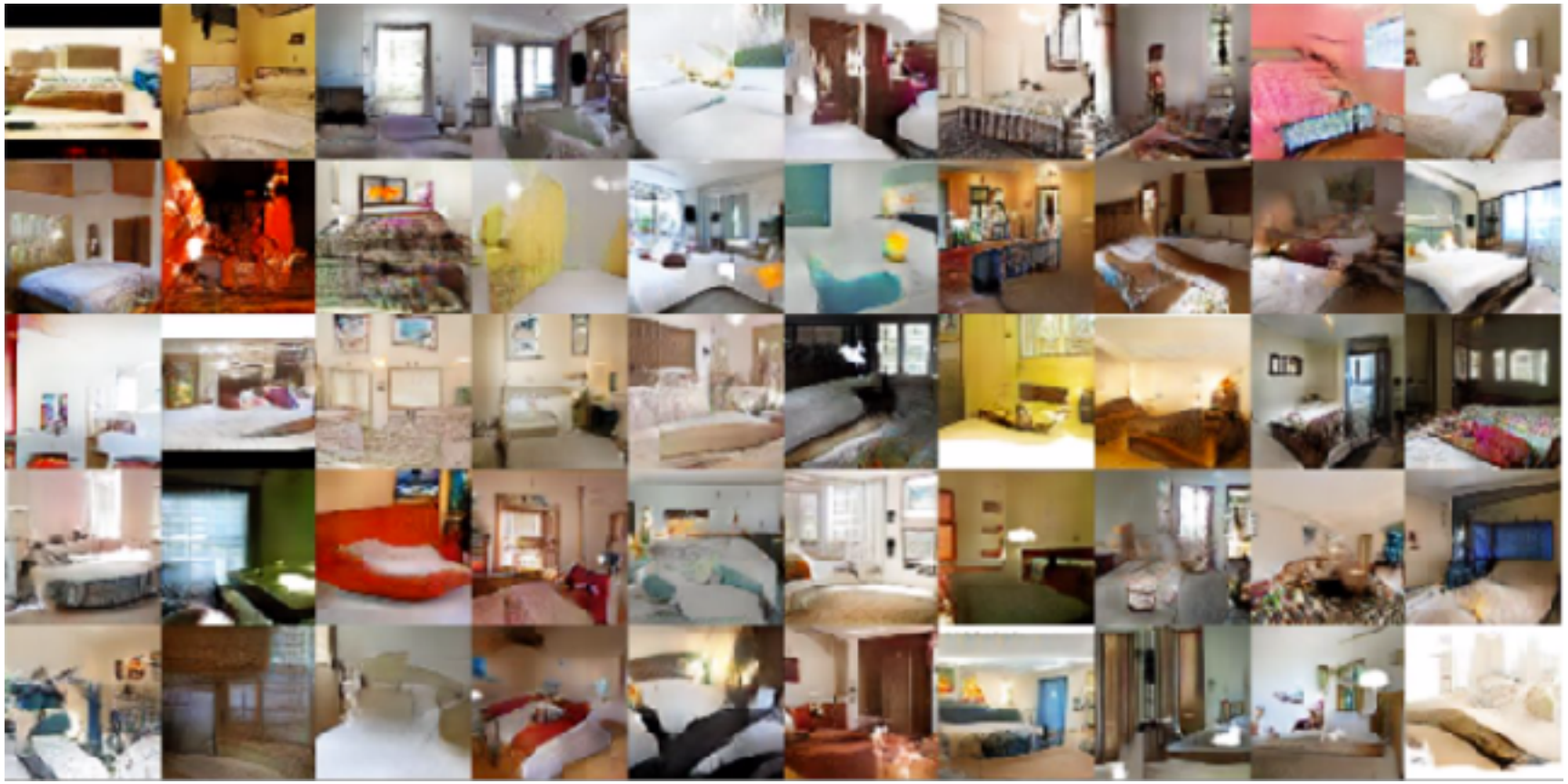
Deep Convolutional GANs

- Just need to reshape and use upsampling



https://sthalles.github.io/intro-to-gans/

Some Results of Generation



Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." *arXiv preprint arXiv:1511.06434* (2015).

GANs in Keras

Implementation from Book on "Frogs from CIFAR"

Demo by Francois Chollet

Follow Along: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/8.5-introduction-to-gans.ipynb

Lecture Notes for

Neural Networks and Machine Learning

GANs

Next Time:
Practical GANs

Reading: Chollet CH8