Deep Learning: Advanced

Machine Learning in der Medizin

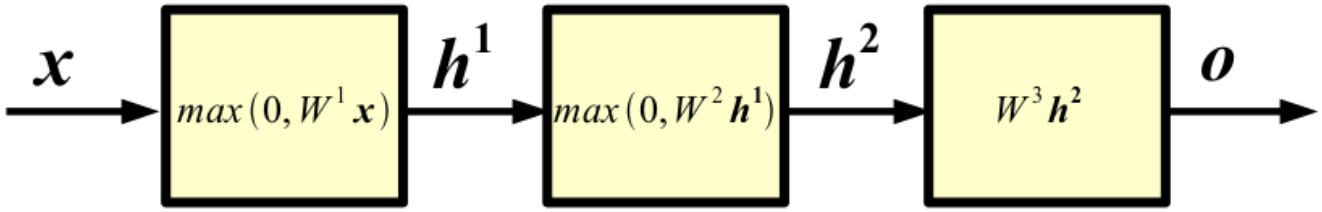
Asan Agibetov, PhD

asan.agibetov@meduniwien.ac.at

Medical University of Vienna
Center for Medical Statistics, Informatics and Intelligent Systems
Section for Artificial Intelligence and Decision Support
Währinger Strasse 25A, 1090 Vienna, OG1.06

Novermber 7, 2019

Recall: Neural Networks



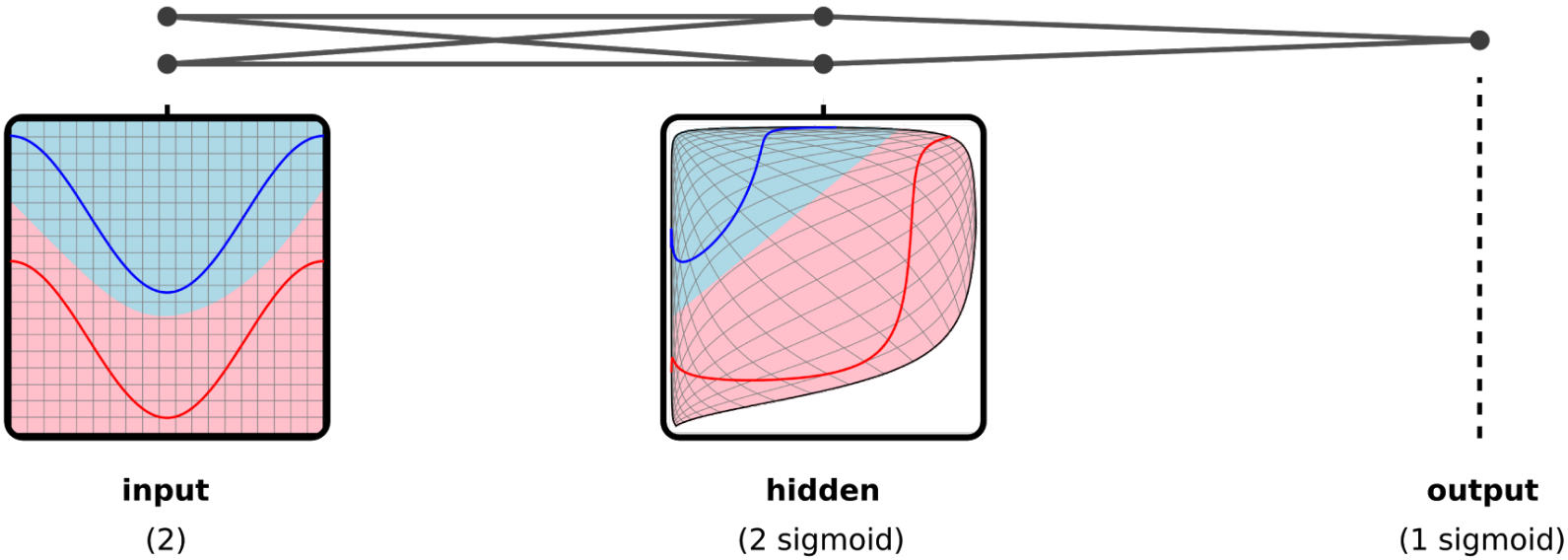
- Universal function approximation that maps input to output
  - $f : \mathbb{R}^n \mapsto \mathbb{R}^m$
- Class of functions considered to map input to output
  - composition of simpler (including non-linear [1]) functions
    - h^1 is non-linear $max(0, \mathbf{W} \cdot \vec{x} + \vec{b})$ aka ReLU
  - $f = o \circ h^2 \circ h^1 \circ x$

---

[1] composition of only linear function would be equivalent to one linear function

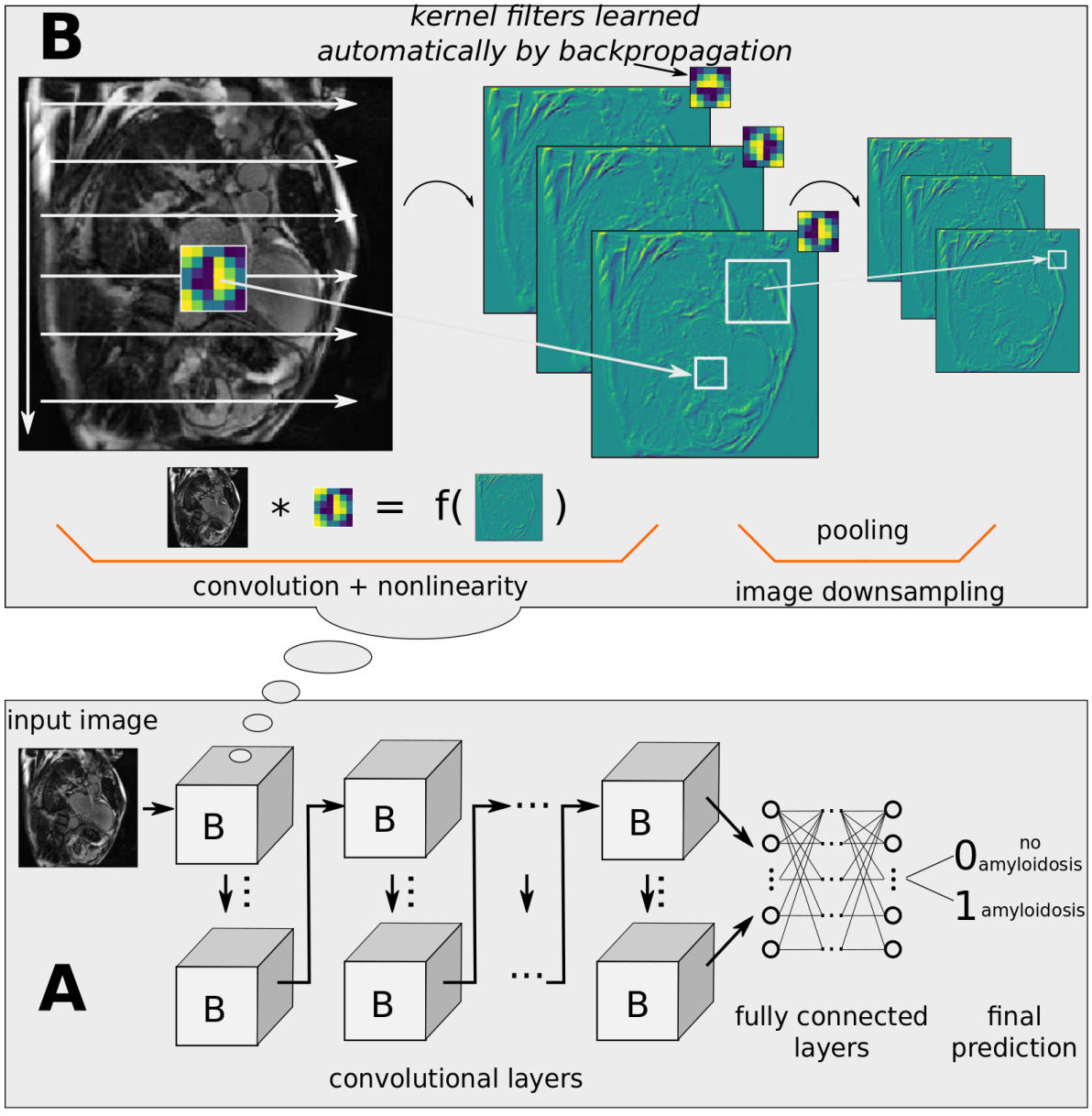
(Image credit) Course notes "An introduction to Deep Learning", Marc'Aurelio Ranzato

Composition of layers: input transformation to match output



(Image credit) Chris Olah's (OpenAI) blogpost on "Neural Networks, Types, and Functional Programming" (http://colah.github.io/posts/2015-09-NN-Types-FP/

Outputs from inputs: predicting outcomes from medical images



Recall: finding the best f with loss functions

Typical setup for optimization

- f can be parameterized with Θ ($f = \Theta \cdot x$ linear case)
- minimizing (learning) the loss function L_i over all training examples $1 \dots n$
- plus regularizations on:
  - $\lambda_2(f)$ - controls complexity of the function (usually norm $\|f\|$)
  - $\lambda_1(f, \Theta)$ - sparsity of the solution, where Θ parameters of f

$$f^* = \operatorname*{argmin}_f = \sum_{i=1}^{n} L_i(y, f(x)) + \lambda_2(f) + \lambda_1(f, \Theta)$$

- to find f^* you need to minimize complicated function
- **backpropagation** gives the gradients of that complicated function

Why do we need loss functions?

- Suppose our training data set consists of two examples

Patient ($x^{(i)}$)	Age	Sex (M=0, W=1)	Outcome ($y^{(i)}$ = Cancer)
1	55	0	1
2	65	1	0

- I want to predict the probability of cancer given data, i.e., $p(y = \text{Cancer}|x)$, my ground truth
  - $p(y = \text{Cancer}|x^{(1)} = \{\text{Age} = 55, \text{Sex} = 0) = 1$
  - $p(y = \text{Cancer}|x^{(2)} = \{\text{Age} = 65, \text{Sex} = 1) = 0$
- I can approximate this conditional probability $p(y = \text{Cancer}|x)$ with a function $f_\Theta(x)$ parameterized with Θ

Why do we need loss functions?

- Suppose our training data set consists of two examples

Patient ($x^{(i)}$)	Age	Sex (M=0, W=1)	Outcome ($y^{(i)}$ = Cancer)
1	55	0	1
2	65	1	0

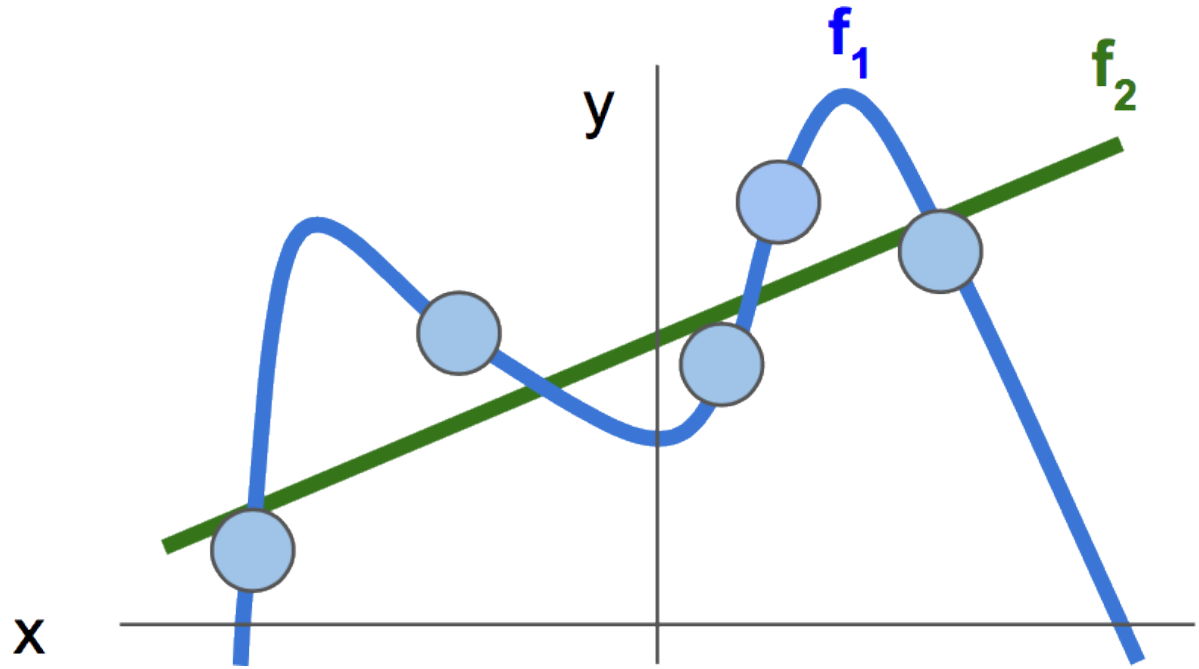
- Intuitively for each training example $\{(x^{(i)}), y^{(i)}\}$ my approximation $f_\Theta(x^{(i)})$ should come close enough to the true probability $y^{(i)}$
  - e.g., $f_\Theta(x^{(1)}) = \{\text{Age} = 55, \text{Sex} = 0) \approx 1$
  - For all $x^{(1)}, \ldots, x^{(n)}$, $f_\Theta(x^{(i)}) \approx p(y^{(i)} = \text{Cancer}|x^{(i)})$
- Loss functions allow us to measure how good is our approximation
  - minimize loss function minimize error between approximation and ouput

Disecting loss functions

$$J(\Theta) = \frac{1}{N} \sum_{i=1}^{N} L_i(f_\Theta(x^{(i)}), y^i) + \lambda(\Theta)$$

- $\frac{1}{m} \sum_{i=1}^{m} L_i(f(x^{(i)}), y^i)$
  - per example data loss: model predictions should match training data
- $\lambda(\Theta)$
  - regularization: prevent model from overfitting, prefer simpler solutions, improve generalization on the test set
- Parameters Θ that minimize $J(\Theta)$ define a model that match as closely as possible training data

Small parenthesis: why we need regularization?



- $\lambda(\Theta)$ regularization: prevent model from overfitting, prefer simpler solutions, improve generalization on the test set

$$J(\Theta) = \frac{1}{N}\sum_{i=1}^{N} L_i(f_\Theta(x^{(i)}), y^i) + \lambda(\Theta)$$

(Image credit) Course notes "CNN for Visual Recognition" (Stanford, Spring 2017)

Probabilistic interpretation (supervized case)

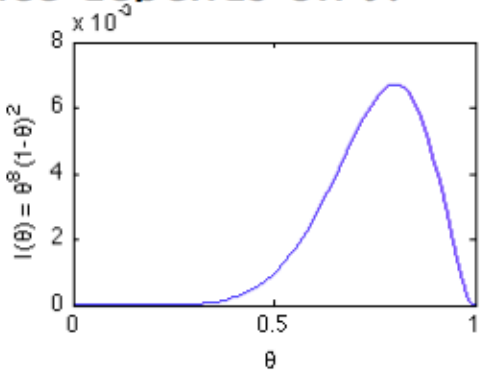
- Our model f_Θ approximates $p(y|x)$, e.g., $p(y = \text{Cancer}|\text{patient})$
- Call this approximated probability distribution $p_{\text{model}}(y|x;\Theta)$
  - The true probability $p(y|x)$ is not available (we don't have access to all possible cancer patients)
  - We only have access to N examples, from which we will try to estimate *empirically* $p(y|x)$ with f_Θ
  - Our best take is to maximize the probability for the correct class with maximum likelihood estimation

Maximum likelihood estimation (Pin flip)

- θ = P(up), 1-θ = P(down)
- Observe:

- Likelihood of the observation sequence depends on θ:

$$\begin{aligned} l(\theta) &= \theta(1-\theta)\theta(1-\theta)\theta\theta\theta\theta\theta\theta \\ &= \theta^8(1-\theta)^2 \end{aligned}$$



- Maximum likelihood finds

$$\arg\max_\theta l(\theta) = \arg\max_\theta \theta^8(1-\theta)^2$$

$$\frac{\partial}{\partial\theta} l(\theta) = 8\theta^7(1-\theta)^2 - 2\theta^8(1-\theta) = \theta^7(1-\theta)(8(1-\theta)-2\theta) = \theta^7(1-\theta)(8-10\theta)$$

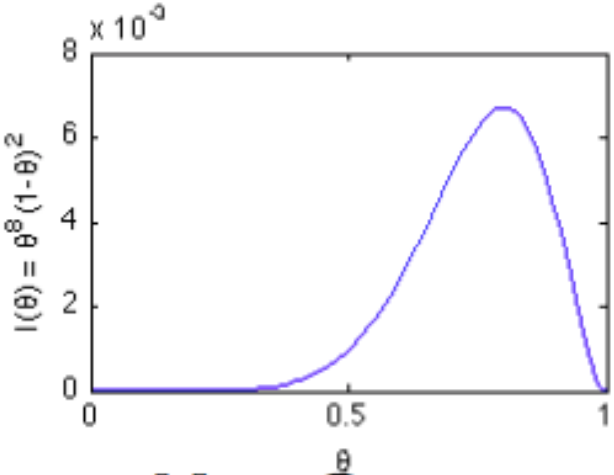
→ extrema at θ = 0, θ = 1, θ = 0.8

→ Inspection of each extremum yields θ_{MI} = 0.8

(Image credit) CS287-FA13, USC Berkley EECS lecture slides

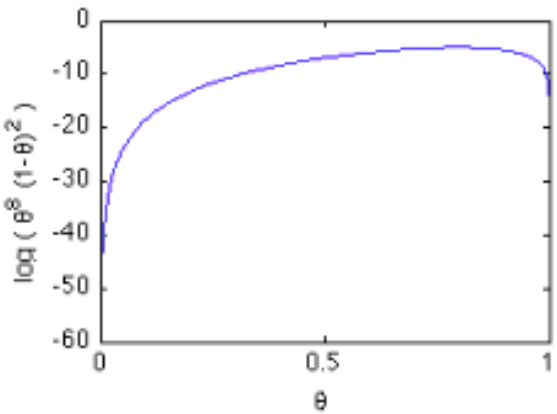
MLE Log transform

- Likelihood



Not Concave

- log-likelihood



Concave

- argmax $l(\Theta) = \text{argmax} \log l(\Theta)$
  - since log non-negative and monotonically increasing
- Concave functions are easier to optimize

(Image credit) CS287-FA13, USC Berkley EECS lecture slides

Maximum likelihood estimation for neural networks

- An important assumption is that all N examples are assumed to be independantly and identically distributed, i.e., each has a probability $1/N$ to be picked.

$$\Theta = \text{argmax}_\Theta p_{\text{model}}(y|x;\Theta) = \text{argmax}_\Theta \prod_{i=1}^{N} p_{\text{model}}(y^{(i)}|x^{(i)};\Theta)$$

- Take log of argmax function (since non-negative and monotonically increasing)
  - log of a product is the sum of log's (e.g., $\log(2 \times 4) = \log 8 = 3 = 1 + 2 = \log 2 + \log 4$.

$$\Theta = \text{argmax}_\Theta \sum_{i=1}^{N} \log p_{\text{model}}(y^{(i)}|x^{(i)};\Theta)$$

Improving stability with negative log-likelihood

- We can scale the sum (does not change argmax)

$$\Theta = \text{argmax}_\Theta \sum_{i=1}^{N} \frac{1}{N} \log p_{\text{model}}(y^{(i)}|x^{(i)}; \Theta)$$

$$= \text{argmax}_\Theta \sum_{i=1}^{m} p_{\text{data}}(x^{(i)}, y^{i}) \log p_{\text{model}}(y^{(i)}|x^{(i)}; \Theta)$$

- In fact, we want to maximize the *expected* loss (average error of my prediction to the true distribution)

$$\text{argmax}_\Theta E_{(x,y)\sim p_{\text{data}}} \left[\log p_{\text{model}}(y^{(i)}|x^{(i)}; \Theta)\right]$$

- minimizing negative expectation is the same as maximizing expectation

$$\text{argmin}_\Theta - E_{(x,y)\sim p_{\text{data}}} \left[\log p_{\text{model}}(y^{(i)}|x^{(i)}; \Theta)\right]$$

Negative log-likelihood

- In most AI papers you will see this term, also refered to as *data loss*
  - All it means is that we use a model to approximate ground truth labels (supervized case)
  - Find parameters Θ of my neural network that fit my training data

$$\Theta = \text{argmin}_{\Theta} - E_{(x,y)\sim p_{\text{data}}} \left[\log p_{\text{model}}(y^{(i)}|x^{(i)}; \Theta) \right]$$

Connection to information theory

- It turns out negative log-likelihood is also connected to the *cross-entropy* quantity from the information theory

$$H(p, q) = -E_p \left[\log q\right].$$

For discrete probability distributions this means

$$H(p, q) = -\sum_x p(x) \log q(x).$$

- Compare it with

$$\Theta = \text{argmin}_\Theta - E_{(x,y)\sim p_{\text{data}}} \left[\log p_{\text{model}}(y^{(i)}|x^{(i)}; \Theta)\right]$$

Cross-entropy loss

- Suppose the conditional probability distribution $p(y|x)$ of your examples are always either 0 or 1. That is the labels of your data points are used as ground truth

$$\begin{aligned} p(y=1|x_{\text{cancer}}) &= 1 \\ p(y=1|x_{\text{no-cancer}}) &= 0 \\ p(y=0|x_{\text{no-cancer}}) &= 1 \\ p(y=0|x_{\text{no-cancer}}) &= 0 \end{aligned}$$

- You would like your neural network f to output a probability distribution q for each example, that matches p

$$\begin{aligned} H(p,q) &= -\sum_x p(y|x) \log q(y|x) \\ &= -(p(y=0|x_{\text{no-cancer}}) \log q(y=0|x_{\text{no-cancer}}) \\ &+ p(y=1|x_{\text{no-cancer}}) \log q(y=1|x_{\text{no-cancer}})) \\ &= -1 \log q(y=0|x_{\text{no-cancer}}) - 0 \log q(y=1|x_{\text{no-cancer}}). \end{aligned}$$

Cross-entropy loss

$$
\begin{aligned}
p(y = 1|x_{\text{cancer}}) &= 1 \\
p(y = 1|x_{\text{no-cancer}}) &= 0 \\
p(y = 0|x_{\text{no-cancer}}) &= 1 \\
p(y = 0|x_{\text{no-cancer}}) &= 0
\end{aligned}
$$

$$
\begin{aligned}
H(p, q) &= -\sum_x p(y|x) \log q(y|x) \\
&= -(p(y = 0|x_{\text{no-cancer}}) \log q(y = 0|x_{\text{no-cancer}}) \\
&+ p(y = 1|x_{\text{no-cancer}}) \log q(y = 1|x_{\text{no-cancer}})) \\
&= -1 \log q(y = 0|x_{\text{no-cancer}}) - 0 \log q(y = 1|x_{\text{no-cancer}}).
\end{aligned}
$$

- Notice that $H(0, 0) = 0$ and $H(1, 1) = 0$
  - cross-entropy is minimized when f outputs probabilities close to ground truth

Comparing two distributions

- *Cross-entropy* uses Kullback-Leibler divergence $D_{KL}(p||q)$ of p from q (relative entropy of q with respect to p) to compare two distributions.

$$H(p, q) = H(p) + D_{KL}(p||q).$$

- The KL-divergence $D_{KL}(p||q)$ is defined as

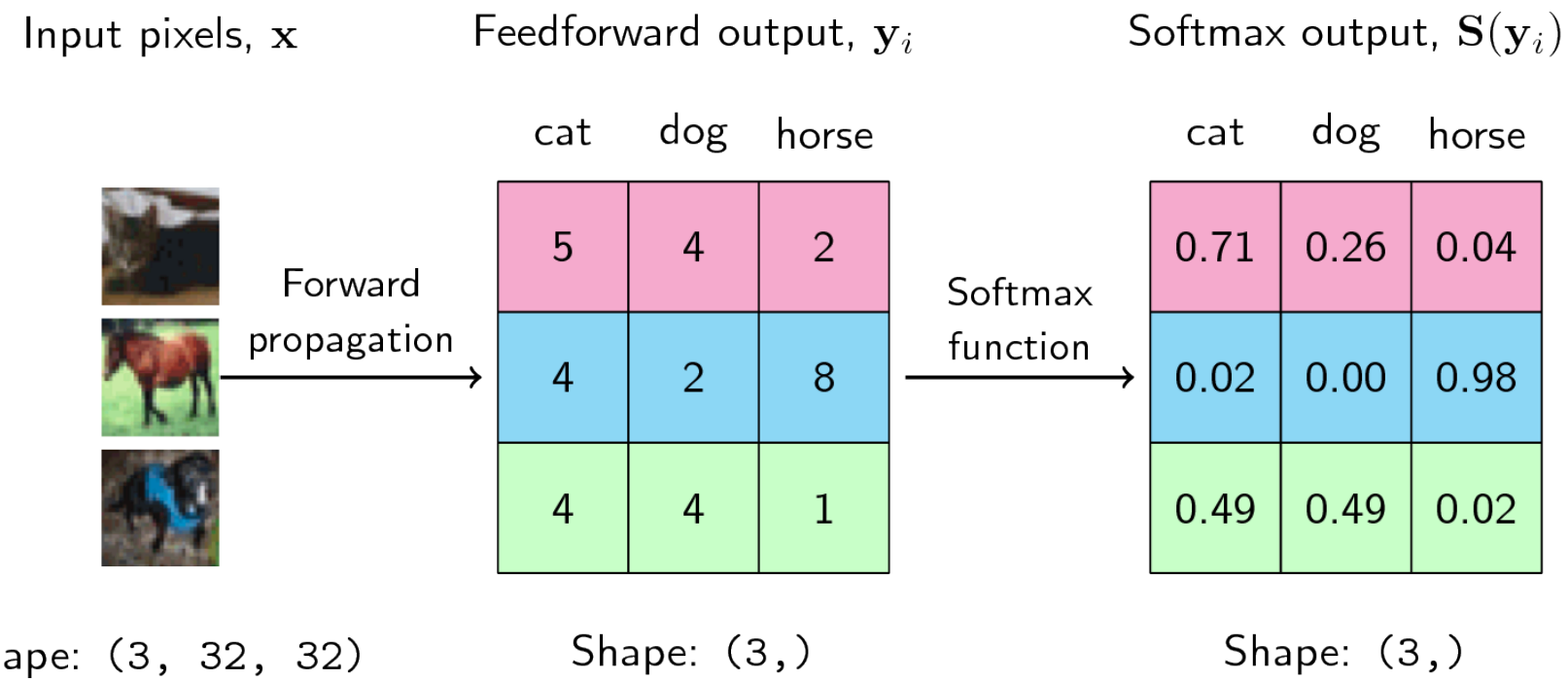
$$\begin{aligned} D_{KL}(p||q) &= -\sum_{i=1}^{N} p(x_i) \log\left(\frac{q(x_i)}{p(x_i)}\right) = \sum_{i=1}^{N} p(x_i) \log\left(\frac{p(x_i)}{q(x_i)}\right) \\ &= E\left[\log\left(\frac{p(x_i)}{q(x_i)}\right)\right] \geq 0. \end{aligned}$$

- Obviously, if the distributions match perfectly, i.e., $\frac{p(x_i)}{q(x_i)} = 0$, then the divergence is zero
  $D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \log\left(\frac{p(x_i)}{q(x_i)}\right) = 0.$

How do I make my network output probabilities?

- Softmax is simply a vector of normalized logits (generalization of sigmoid function)

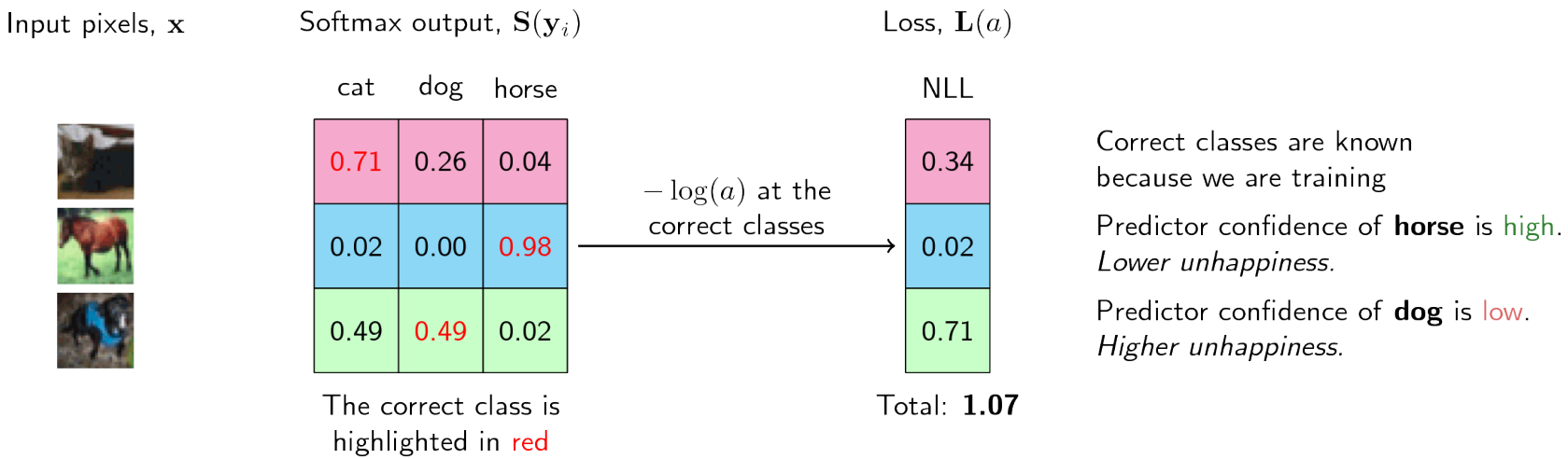
$$\text{softmax}(f_k) = \frac{e^{f_k}}{\sum_j e^{f_j}}.$$



(Image credit) L.Miranda tech blog
https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-

How do I measure how good is my network at outputing probabilities?

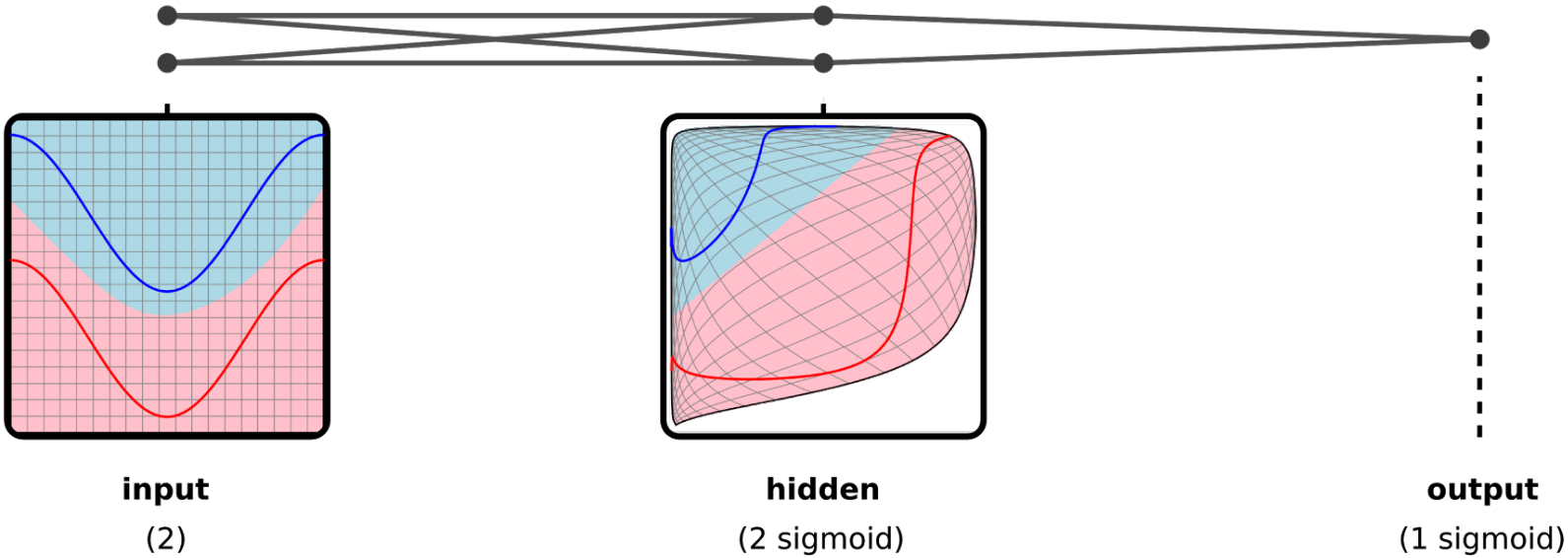
- Use cross-entropy (negative log-likelihood) to *adjust* classifier's probabilities



(Image credit) L.Miranda tech blog https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/

Bottom line: NN input to output transformation

- $f : \mathbb{R}^n \mapsto \mathbb{R}^m$: Universal function approximation that maps input to output



- Can make my NN output probabilities
- Can use loss functions to match my training data
  - approximate true data distribution

(Image credit) Chris Olah's (OpenAI) blogpost on "Neural Networks, Types, and Functional Programming" (http://colah.github.io/posts/2015-09-NN-Types-FP/

Questions?

- Small break

Supervised vs. Unsupervised

- Supervised: Given a dataset $D = \{(x, y)\}$ of inputs x with targets y, learn to predict y from x (MLE)
  - $L(D) = \sum_{(x,y)\in D} -\log p(y|x)$
  - Clear task: learn to predict y from x
- Unsupervised: Given a dataset $D = \{x\}$ of inputs x learn to predict... what?
  - $L(D)$?
  - We want a single task that will allow the network generalize to many other tasks (which ones?)

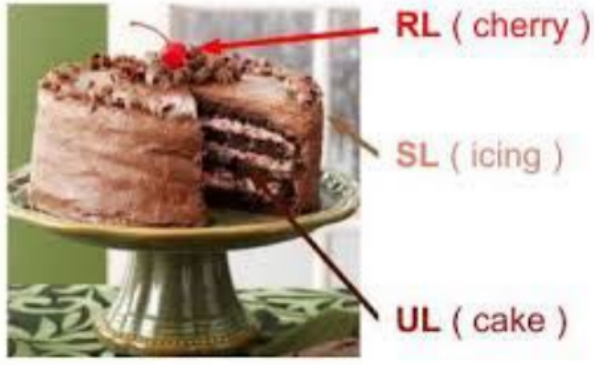
Can we learn data?

- One way: **density modeling**
  - $L(D) = \sum_{x \in D} -\log p(x)$
  - Goal: learn *true* distribution (**everything** about data)
- Issues:
  1. curse of dimensionality (complex interactions between variables)
  2. focus on low-level details (pixel correlations, word N-Grams), rather than on "high-level" structure (image contents, semantics)
  3. even if we learn underlying structure, how access and exploit that knowledge for future tasks (*representation learning*)

What's wrong with learning tasks?

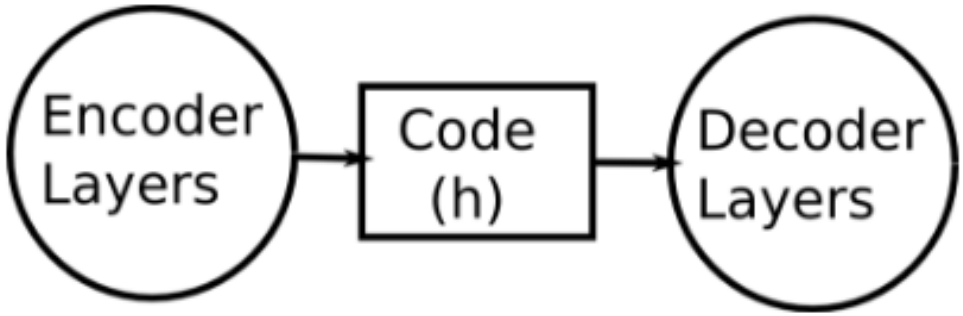
- We want rapid *generalization* to *new tasks* and *situations*
  - just as humans who learn skills and apply them to tasks, rather than the other way around
  - "Stop learning tasks, start learning skills" - Satinder Singh @ Deepmind
- Is there enough *information* in the *targets* to learn transferable skills?
  - *targets* for supervised learning contain *far less* information than the input data
  - unsupervised learning gives us an essentially **unlimited** supply of information about the world

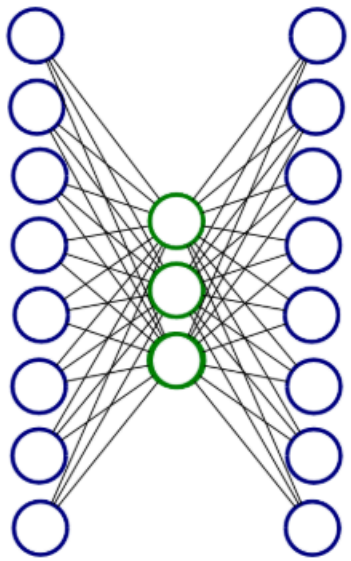
you can't eat just the cherry, nor just the filling....
you gotta eat a whole slice!



picture/metaphor credit: Y. LeCun

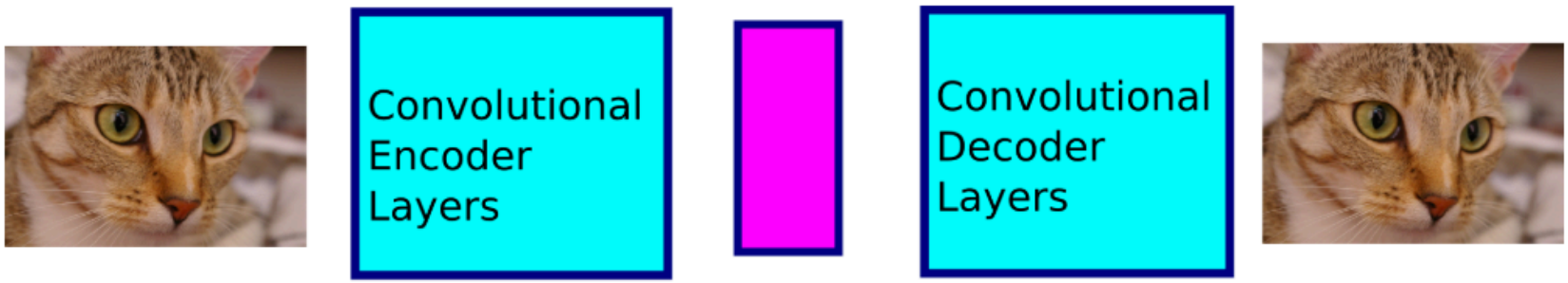
Autoencoders: learning data distributions





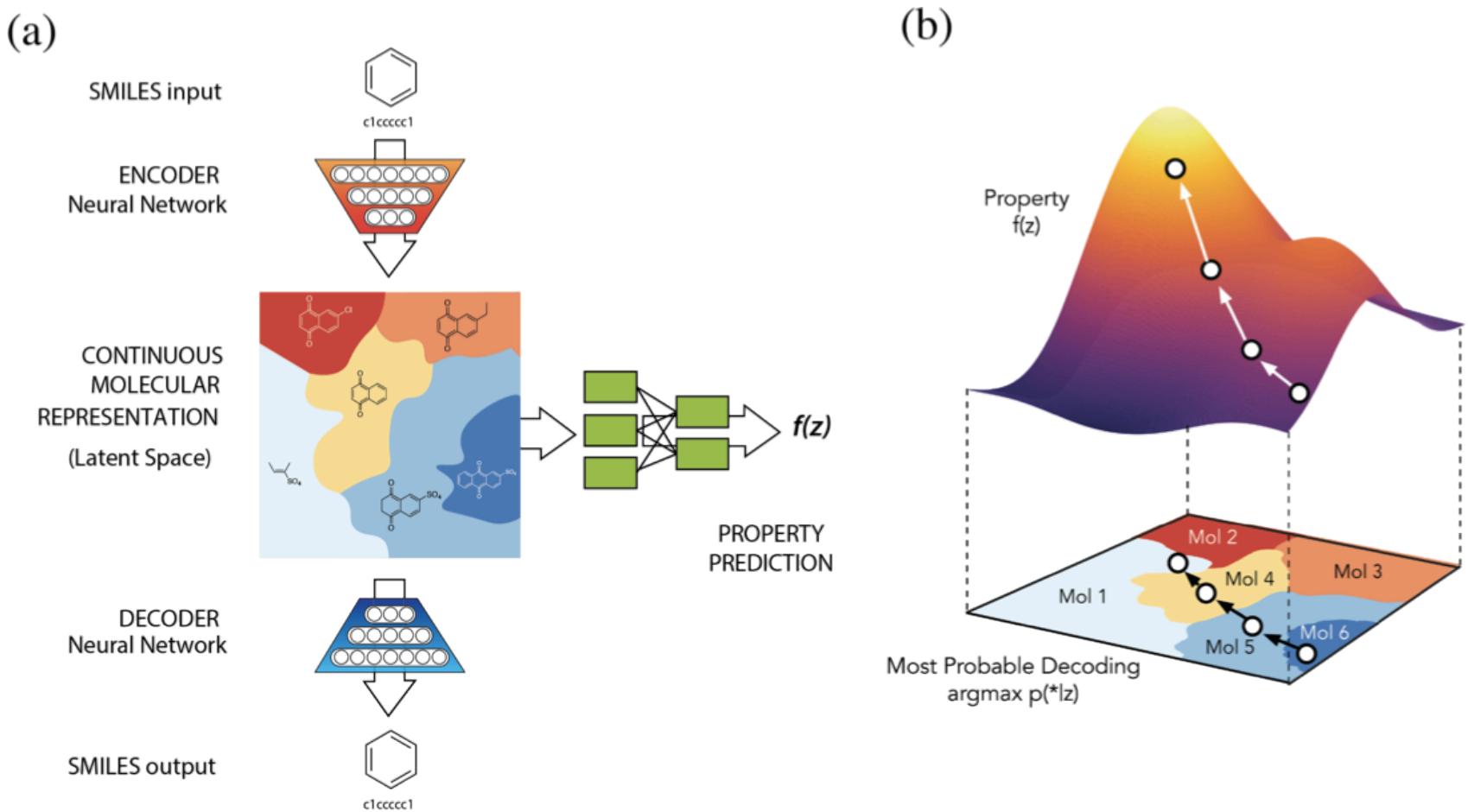
(Images credit) B. Irving Autoencoder tutorial
https://github.com/benjaminirving/mlseminars-autoencoders

Convolutional autoencoders



(Images credit) B. Irving Autoencoder tutorial
https://github.com/benjaminirving/mlseminars-autoencoders

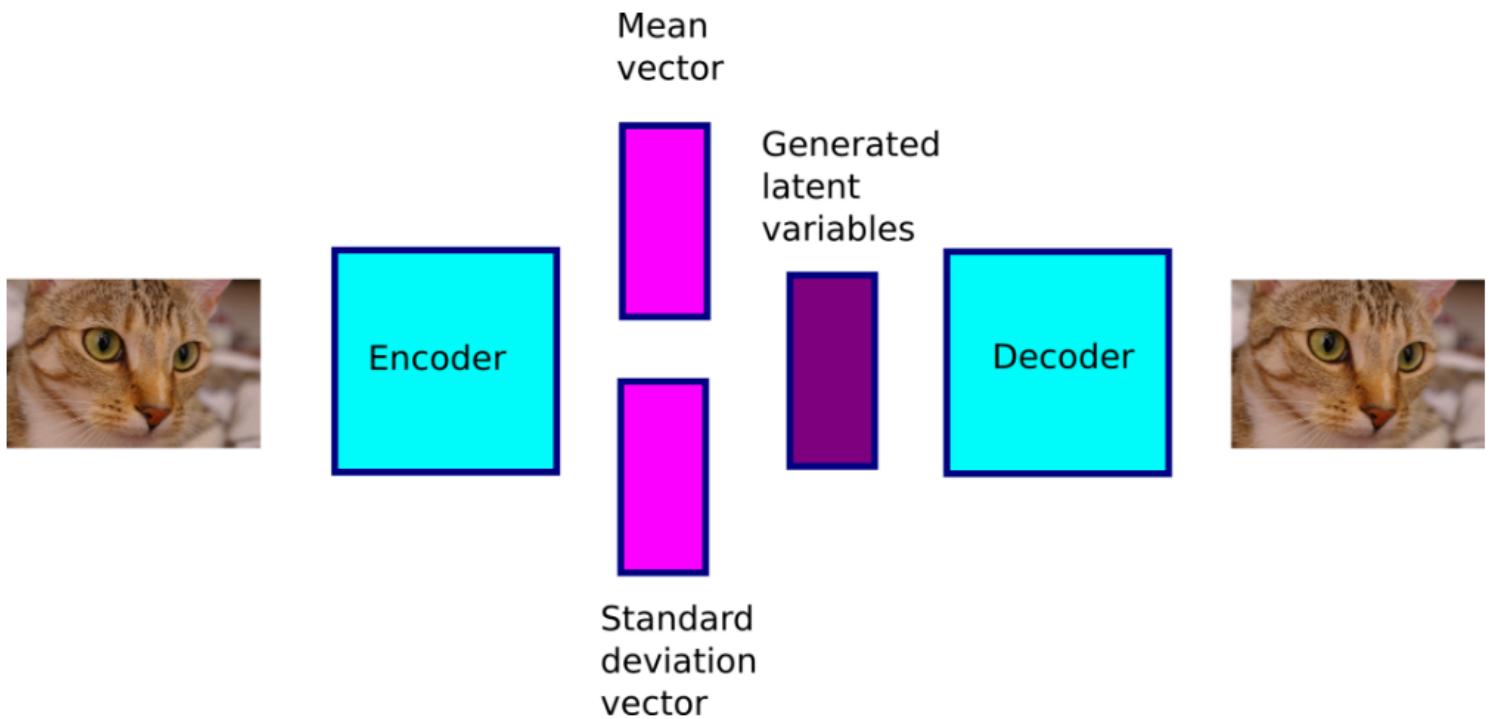
Can we have more control over latent space?



(Image credit) Gomez-Bombarelli "Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules" arxiv. 2017

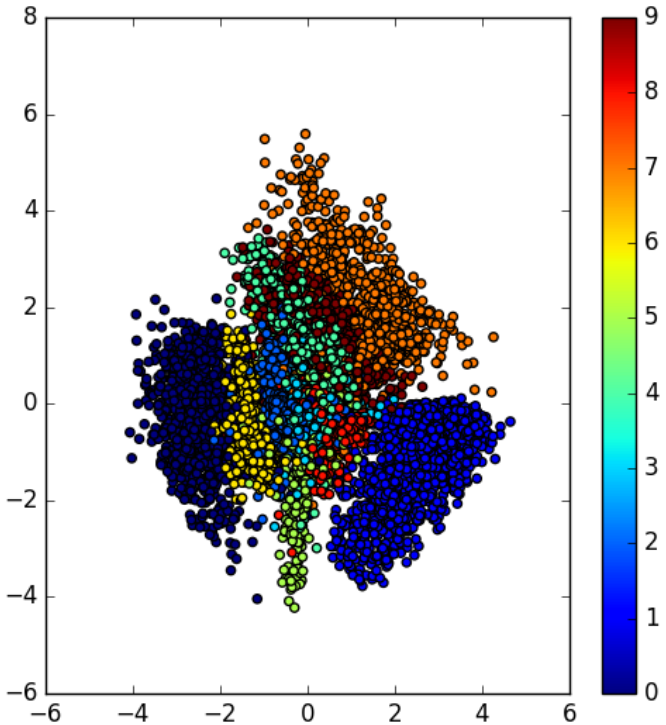
Variational auto encoder (VAE)

- The holy grail of VAE is to have a neural network that can extract very few (interpretable) features from a high dimensional space
- Our hope is obviously that these learned features will generalize well outside of the domain of your training data.



(Images credit) B. Irving Autoencoder tutorial
https://github.com/benjaminirving/mlseminars-autoencoders

VAE on MNIST



- Sampling from the latent space generates *new* images
  - Why does 5 look similar to 3 and 8?

(Images credit) A. Kristaldi's tech blog
https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/

How does VAE work?

We want to model $P(x)$ via latent variables z_i

$$P(x) = \int P(x, z)dz = \int P(x|z)p(z)dz.$$

- Now, we don't know the distribution of latent variables z, however we could approximate it with $p(z|x)$.
- Intuitive explanation: we wanna make our latent variable likely under our data,
  - i.e., restrict portions of z that are only strictly relevant to our data x

Variational inference

- To infer the distribution $P(z|x)$ we use variational inference, which uses optimization to infer $P(z|x)$.
- To model true distribution $P(z|x)$ we will use a simpler distribution, i.e., Gaussian, and we will use the KL divergence metric.
- Let us use $Q(z|x)$, and to make it as close as possible to the true distribution $P(z|x)$ we will use the KL divergence metric.

$$
\begin{aligned}
D_{KL}[Q(z|x)||P(z|x)] &= \sum_z Q(z|x) \log\left(\frac{P(z|x)}{Q(z|x)}\right) \\
&= E\left[\log\left(\frac{P(z|x)}{Q(z|x)}\right)\right] \\
&= E\left[\log Q(z|x) - \log P(z|x)\right].
\end{aligned}
$$

Rewrite KL divergence with Bayes' rule

Now, let us rewrite with the Bayes' rule:

$$
\begin{aligned}
D_{KL}[Q(z|x)||P(z|x)] &= E\left[\log Q(z|x) - \log P(z|x)\right] \\
&= E\left[\log Q(z|x) - \log \frac{P(x|z)P(z)}{P(x)}\right] \\
&= E\left[\log Q(z|x) - (\log(P(x|z) + \log P(z) - \log P(x))\right] \\
&= E\left[\log Q(z|x) - \log(P(x|z) - \log P(z) + \log P(x)\right].
\end{aligned}
$$

Rewrite further

- Since the expectation is over z and $P(x)$ does not depend on z, we could move it outside of the expectation.

$$D_{KL}[Q(z|x)||P(z|x)] = E[\log Q(z|x) - \log(P(x|z) - \log P(z) + \log P(x)]$$
$$= E[\log Q(z|x) - \log(P(x|z) - \log P(z)] + \log P(x)$$
$$D_{KL}[Q(z|x)||P(z|x)] - \log P(x) = E[\log Q(z|x) - \log(P(x|z) - \log P(z)]$$

And further

By looking closely, we can see that the last term can also be rewritten to include another KL divergence term. But before let's multiply by -1 the whole expression

$$\log P(x) - D_{KL}[Q(z|x)||P(z|x)] = E[\log(P(x|z) - (\log Q(z|x) - \log P(z))]$$
$$= E[\log(P(x|z)] - E[\log Q(z|x) - \log P(z)]$$
$$= E[\log(P(x|z)] - D_{KL}[Q(z|x)||\log P(z)]$$

Rewritten objective to minimize

- The final objective than becomes

$$\log P(x) - D_{KL}[Q(z|x)||P(z|x)] = E[\log(P(x|z)] - D_{KL}[Q(z|x)||\log P(z)]$$

So we have

1. $P(x)$ distribution of data
2. $Q(z|x)$ projection of our data x onto latent variable space
3. z latent variable
4. $P(x|z)$ posterior that generates data given latent variable

More observations

- Then we notice that $D_{KL}[Q(z|x)||P(z|x)]$ cannot be computed since we do not know $P(z|x)$, however,
  - since it is always non-negative, we can discard it and maximize the lower bound

$$\log P(x) \geq E\left[\log(P(x|z)\right] - D_{KL}\left[Q(z|x)||\log P(z)\right]$$

- $E\left[\log(P(x|z)\right]$ is the standard *loss* term
- $D_{KL}\left[Q(z|x)||\log P(z)\right]$ divergence between the approximated and latent prior (of our choice)

Choosing priors

- We will use a simple prior on our latent variables, with $P(z) = N(0, 1)$.
- Assume that we have k Gaussians, then the divergence between the Gaussians can be easily computed using the following formulas

$$
\begin{aligned}
& D_{KL}\left[N(\mu(x), \Sigma(x)) || N(0, 1)\right] \\
&= \frac{1}{2}\left(\mathrm{tr}(\Sigma(x)) + \mu(x)^T \mu(x) - k - \log\det(\Sigma(x))\right) \\
&= \frac{1}{2}\sum_k \left(\Sigma(x) + \mu(x)^2 - 1 - \log\Sigma(x)\right).
\end{aligned}
$$

Further numerical tricks for stability

The numerical trick is to take the exponent of the covariance matrix

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

Video time!

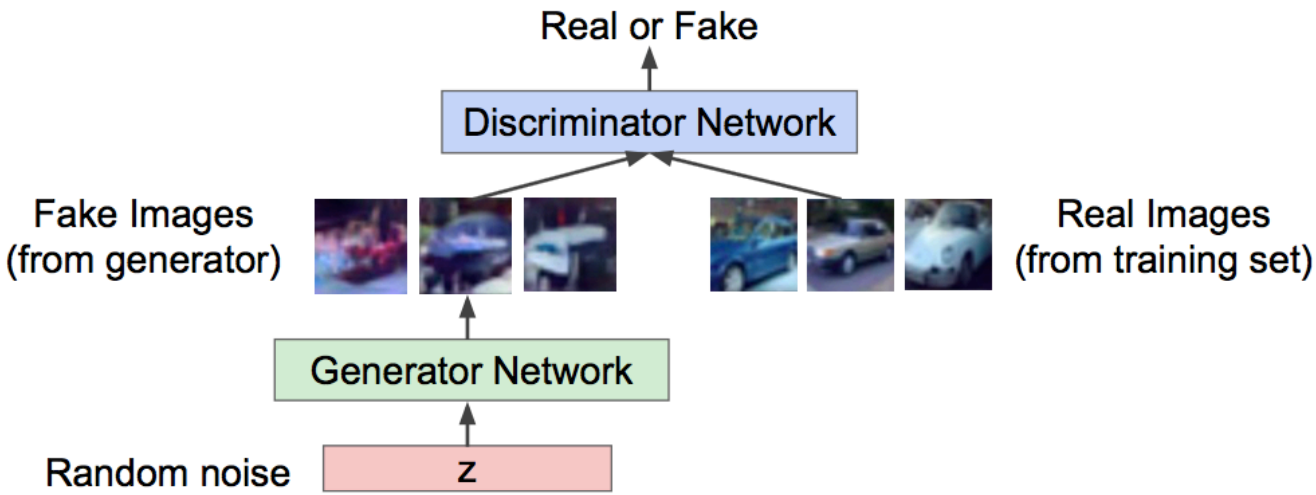
- Videos are good

Generative adversarial networks (GAN)

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

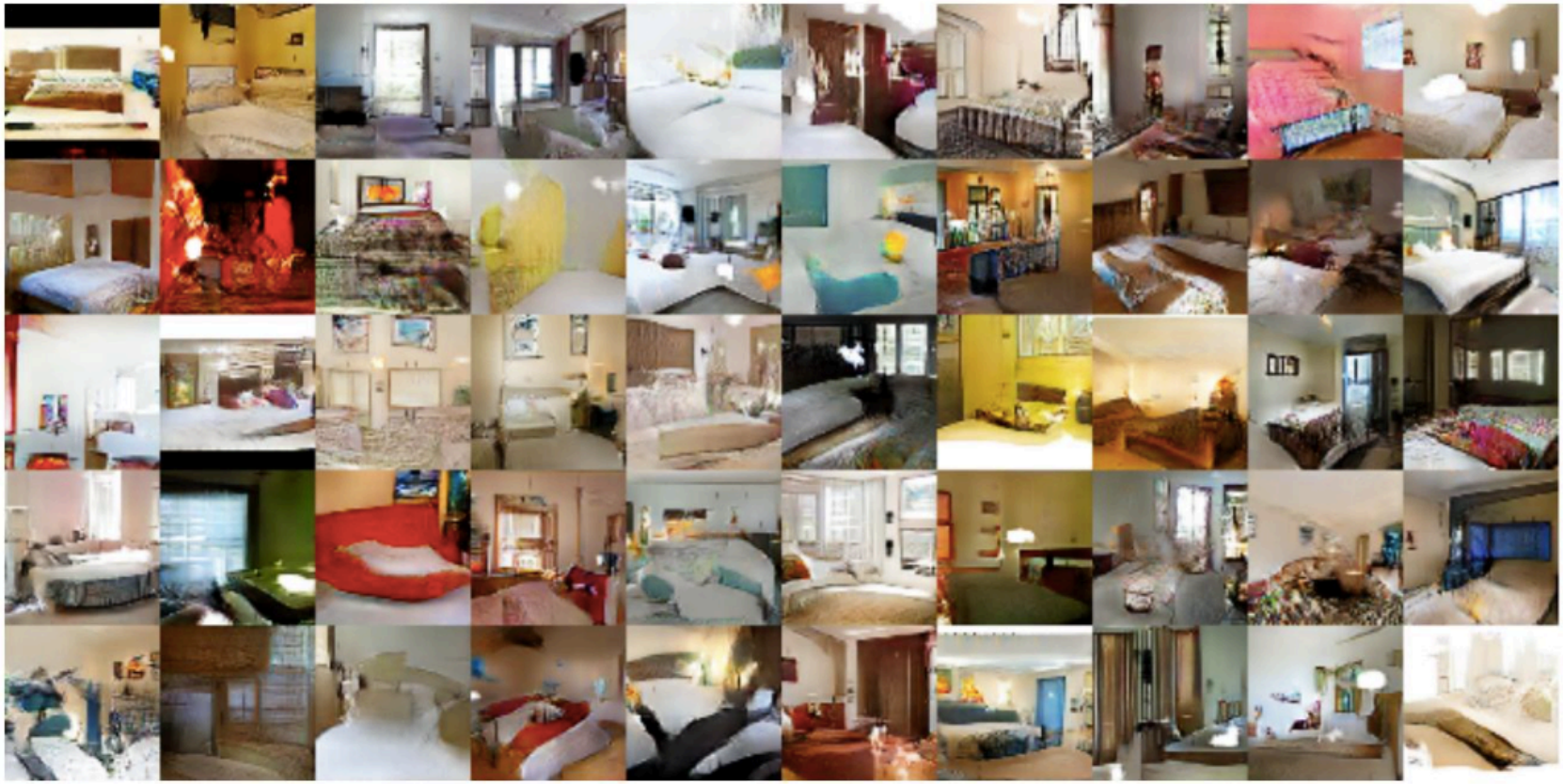


(Image credit) Course notes "CNN for Visual Recognition" (Stanford, Spring 2017)

Difference from VAE

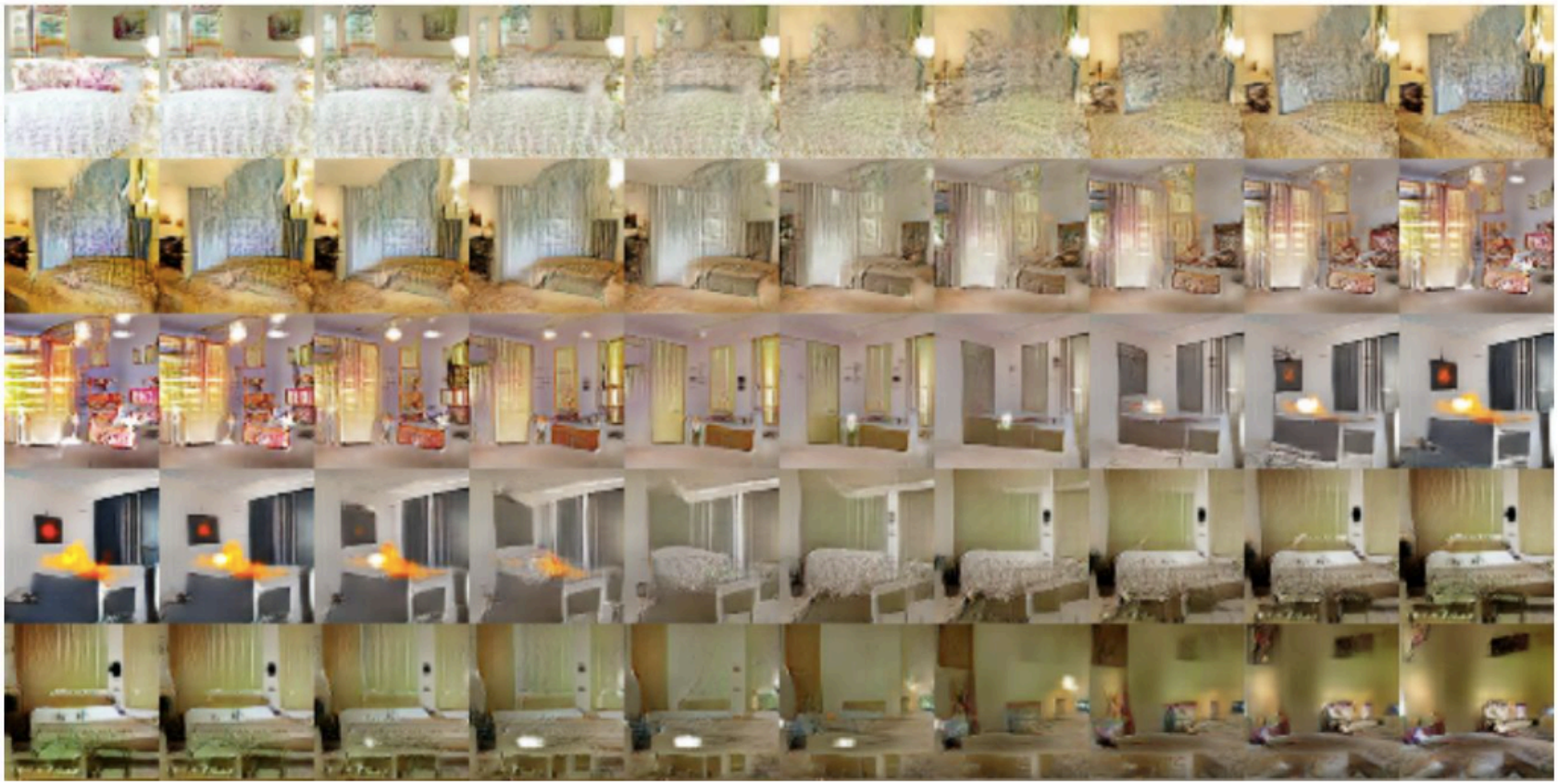
- In VAE we try to minimize a lower bound of an intractable likelihood function
- With GANs we use a game-theoretic approach instead
  - learn to generate from training distribution by
  - sampling from a simple distribution, e.g., random noise, learn transformation to training distribution
- Two networks
  - Generator network: try to fool the discriminator by generating real-looking images
  - Discriminator network: try to distinguish between real and fake images

CoNvolutional GAN



(Image credit) Course notes "CNN for Visual Recognition" (Stanford, Spring 2017)

Interpolation between images with (conv) GANs



(Image credit) Course notes "CNN for Visual Recognition" (Stanford, Spring 2017)

How does GAN work?

- The model is trained jointly in a minimax game (minimax objective function)

$$\min_{\Theta_g} \max_{\Theta_d} \left[E_{x \sim p_{data}} \log D_{\Theta_d}(x) + E_{z \sim p(z)} \log(1 - D_{\Theta_d}(G_{\Theta_g}(z))) \right]$$

- $D_{\Theta_d}(x)$
  - discriminator output for real data x (likelihood in $(0, 1)$)
- $D_{\Theta_d}(G_{\Theta_g}(z))$
  - discriminator output for generated fake data $G_{\Theta_g}(z)$

Training GANs

We use gradient ascent on discriminator (maximization)

$$\max_{\Theta_d} \left[E_{x \sim p_{data}} \log D_{\Theta_d}(x) + E_{z \sim p(z)} \log(1 - D_{\Theta_d}(G_{\Theta_g}(z))) \right]$$

- Discriminator $D_{\Theta_d}(G_{\Theta_g}(z))$ wants to **maximize objective** such that $D_{\Theta_d}(x)$ is close to 1 (real) and $D_{\Theta_d}(G_{\Theta_g}(z))$ is close to 0 (fake)

And we use gradient descent (minimization) on generator

$$\min_{\Theta_g} \left[E_{z \sim p(z)} \log(1 - D_{\Theta_d}(G_{\Theta_g}(z))) \right]$$

Training GANs

- Generator G_{Θ_g} wants to **minimize objective** such that $D_{\Theta_d}(G_{\Theta_g}(z))$ is close to 1 (discriminator is fooled into thinking generated $G_{\Theta_g}(z)$ is real)
- i.e., we **minimize** likelihood of discriminator being correct

In practice, we use gradient ascent on the generator with a slightly different objective function

$$\max_{\Theta_g} \left[E_{z \sim p(z)} \log(D_{\Theta_d}(G_{\Theta_g}(z))) \right]$$

- i.e., we **maximize** likelihood of discriminator being wrong

This choice allows for higher gradient signal for bad samples.

Video time!

- Videos are awesome