10707
Deep Learning: Spring 2020

Andrej Risteski

Machine Learning Department

Course review & wrap-up

Goals of this course

Convey levels of understanding of phenomena in deep learning

Fully **mathematical** understanding: satisfying mathematical theory. (*Rare*.)

"**Physics**" understanding: mathematical theory under strong structural assumptions, e.g. random inputs, independence. (*Semi-common*.)

"**Practical**" understanding: imprecise intuitions/conjectures from trial-and-error. (*Common*.)

Survey classical and recent topics in deep learning

Classical topics provide historical context and intuition for modern ones.

Breadth-first rather than depth-first search.

Mix of hands-on experience and rigorous thinking

Problem sets will force you to think mathematically about topics we cover.

Coding assignments will involve implementing algorithms taught in class.

Differences with prior offering of 10-707

Topic selection: while there is overlap, the topic selection is somewhat different from prior course. Check Schedule to for details.

More mathematical content: the course will be mathematically more demanding, and we will cover substantively more theory.

Assignments: problem sets, as well as coding assignments. Final exam instead of a project.

Course organization

Supervised learning: learning in the presence of labels

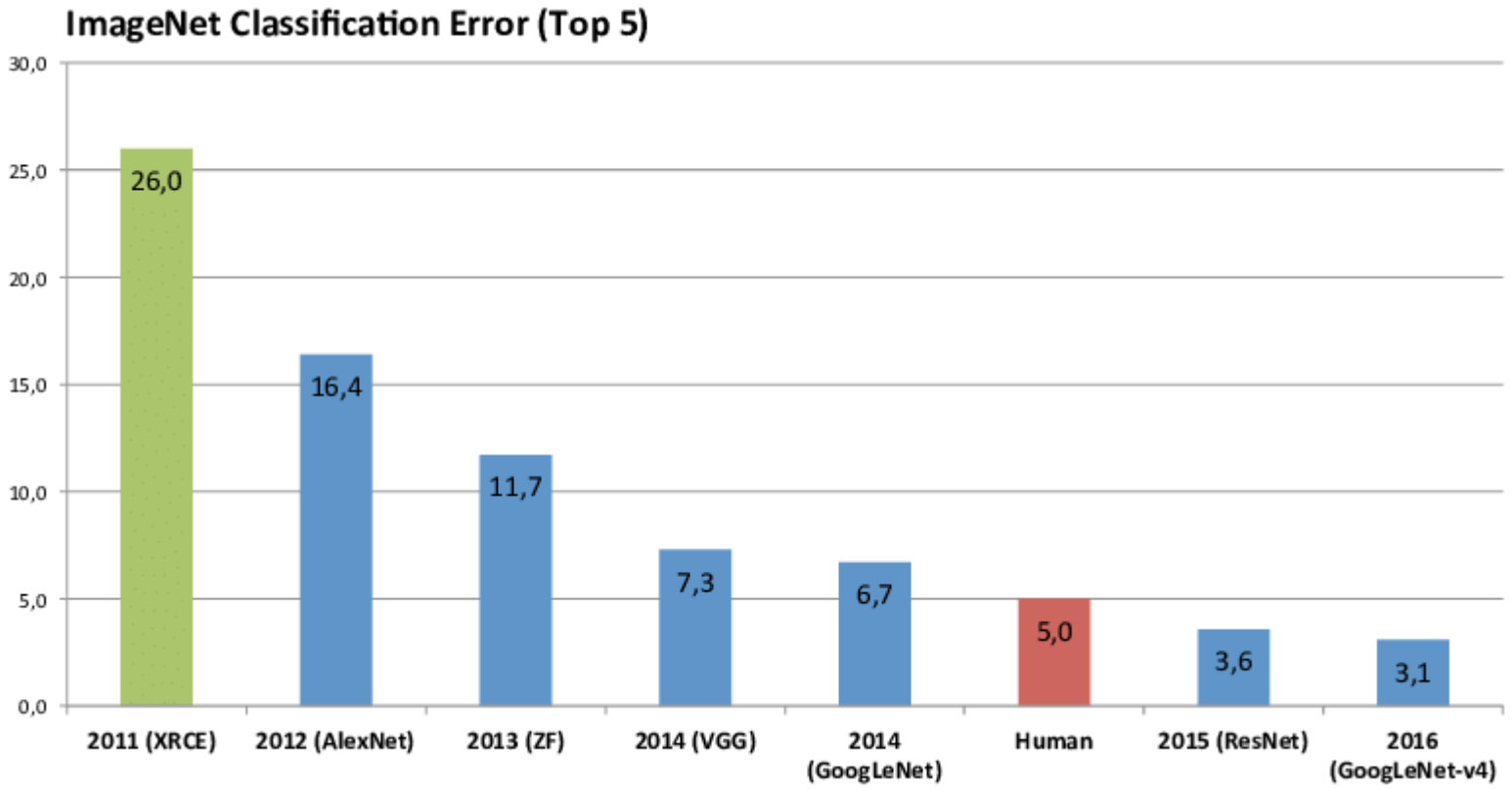
Unsupervised learning: learning in the absence of labels

Sequential data: models for text / language / translation

Other topics: robustness, causality, etc.

Part I: Supervised learning

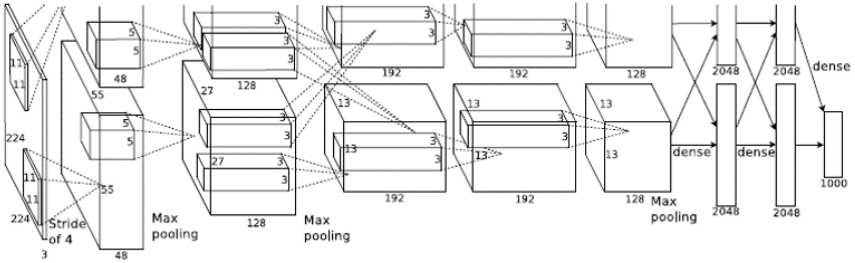
The start of the deep learning renaissance:



Part I: Supervised learning

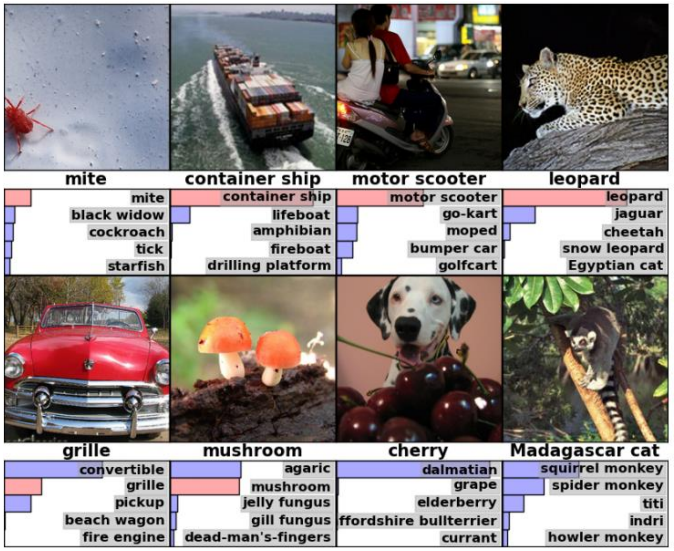
Deep Convolutional Nets for Vision (Supervised)

Krizhevsky, A., Sutskever, I. and Hinton, G. E., ImageNet Classification with Deep Convolutional Neural Networks, NIPS, 2012.



IMAGENET

1.2 million training images
1000 classes



Deep Nets for Speech (Supervised)

Hinton et. al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups, IEEE Signal Processing Magazine. 2012.

Part I: Supervised learning

Empirical risk minimization approach:
minimize a **training** loss l over a class of **predictors** $\mathcal{F}$:

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{(x,y):\text{training samples}} l(f(x), y)$$

Three pillars:

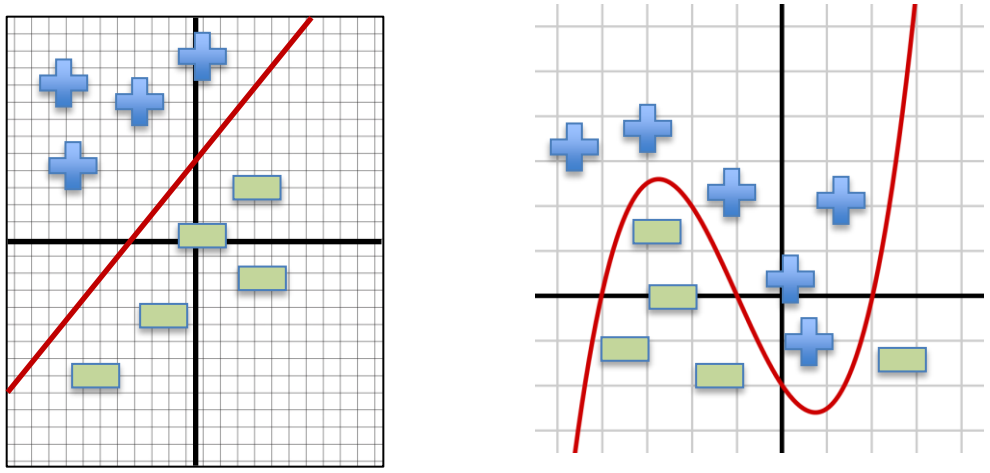
(1) How expressive is the class $\mathcal{F}$? (**Representational power**)

(2) How do we minimize the training loss efficiently? (**Optimization**)

(3) How does $\hat{f}$ perform on unseen samples? (**Generalization**)

Expressivity

What do we mean by expressivity?

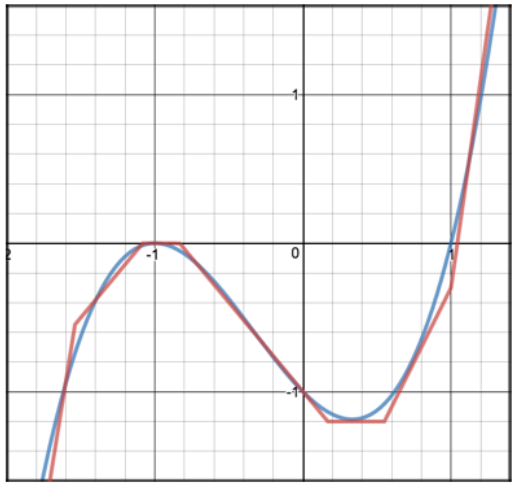


Expressive = functions in class can represent “complicated” functions

"Universal" expressivity of neural networks

(1): Neural networks are **universal approximators**: given any (reasonably nice) function f: $\mathbb{R}^d \to \mathbb{R}$, a **shallow** (3-layer) neural network with $\sim \left(\frac{1}{\epsilon}\right)^{\mathrm{d}}$ neurons can approximate it to within ϵ error.

"curse of dimensionality"



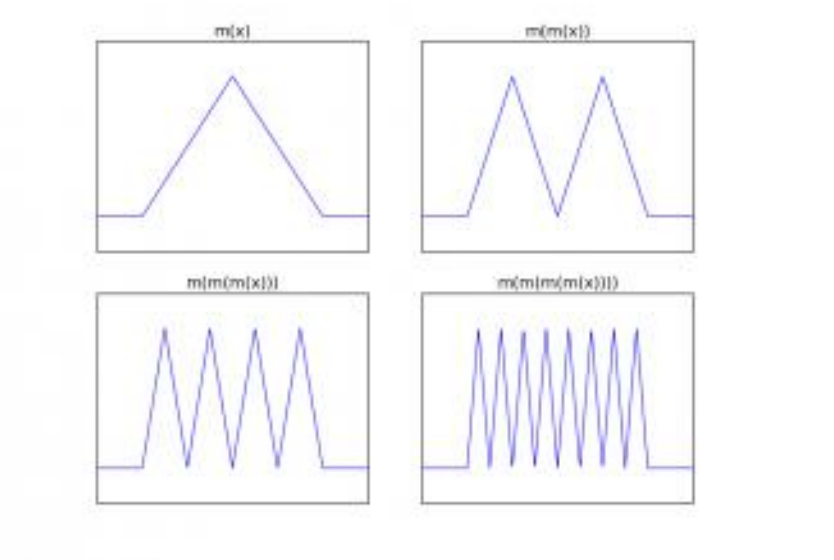
(2): Neural networks can **circumvent** the curse of dimensionality for functions w/ decaying Fourier coefficients:

shallow neural networks with $\sim \left(\frac{1}{\epsilon}\right)$ neurons can approximate them to within ϵ error.

Does depth help?

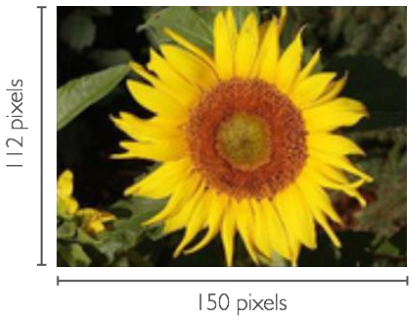
Universal approximation constructions yield good approximators that are **shallow**. Is there a benefit to using deeper neural networks?

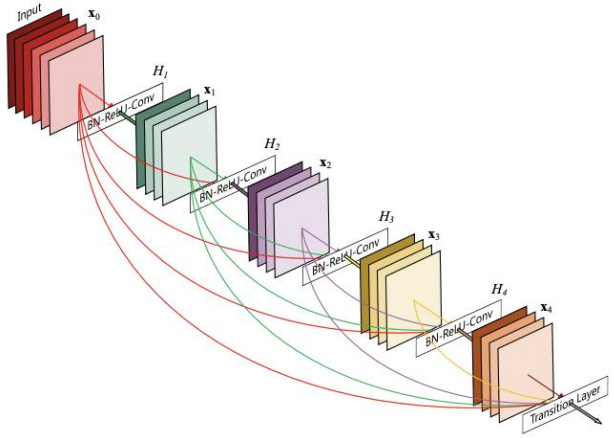
Depth separation theorems: for every $k \in \mathbb{N}$, there are functions representable by k^3 -layer neural networks of size $O(k^3)$, s.t. every $O(k)$ -depth network approximating them has size $\Omega(2^k)$.



Modern architectures

Applications in vision: architecture should make use of structure in visual data (pixel locality, invariance, texture, hierarchy, etc.). **Solution**: convolutional networks





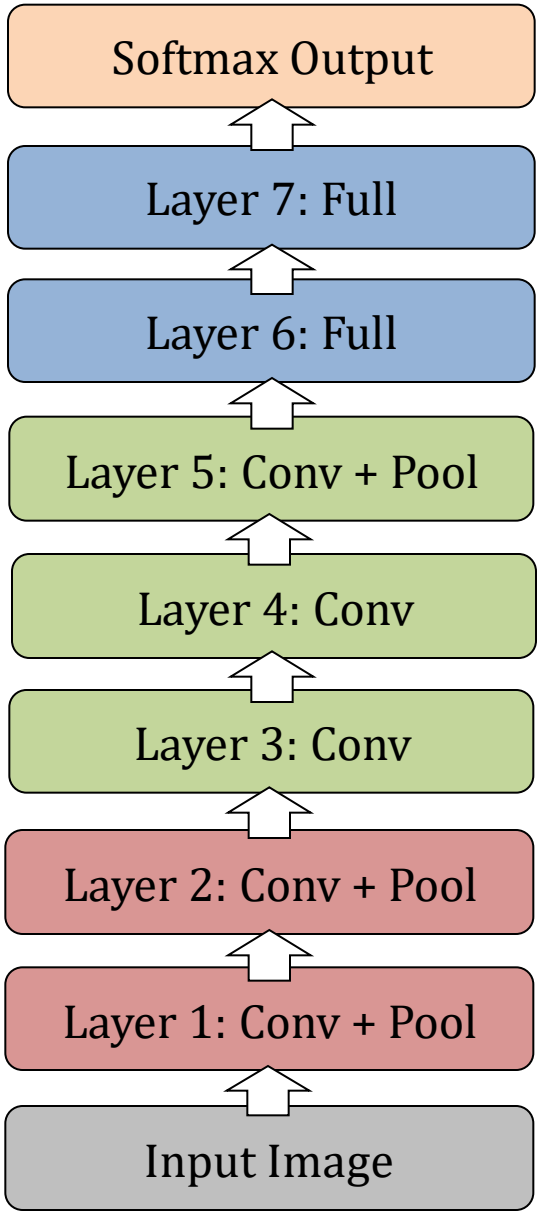
Modern architectures: large, deep networks have various training problems (vanishing gradients, poor generalization). **Solution(s)**: Residual networks (ResNets), dense convolutional networks (DenseNets).

AlexNet

8 layers total, ~60 million parameters

Trained on Imagenet
dataset [Deng et al. CVPR'09]

18.2% top-5 error

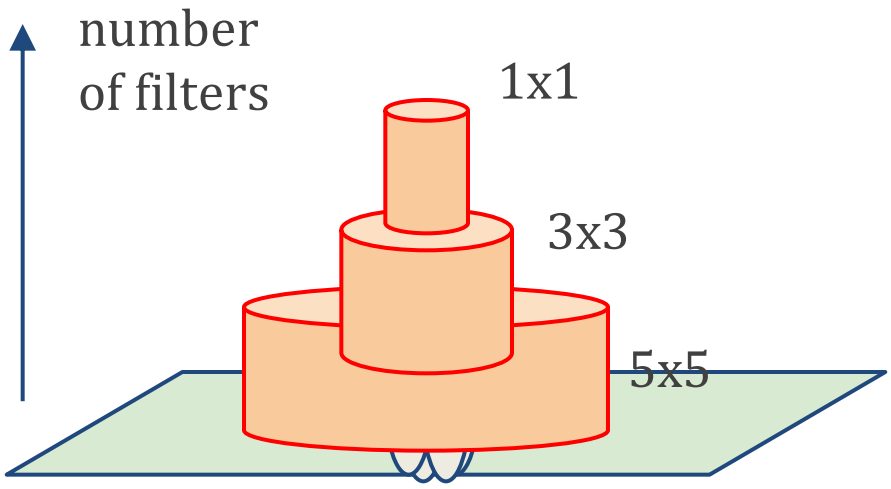


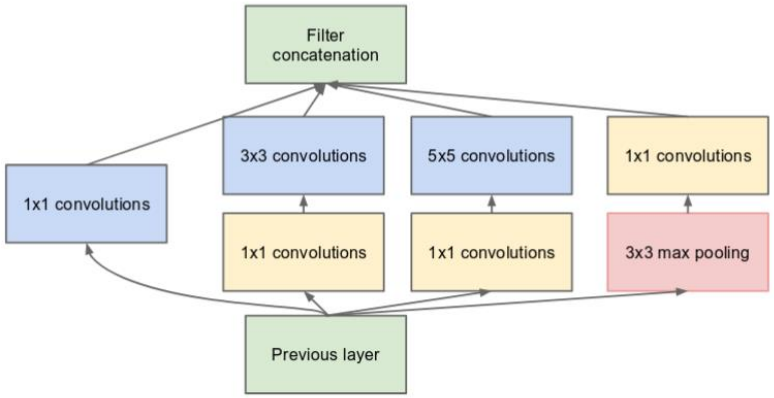
[From Rob Fergus' CIFAR 2016 tutorial]

GoogLeNet

GoogLeNet inception module:

- Multiple filter scales at each layer
- Dimensionality reduction to keep computational requirements down



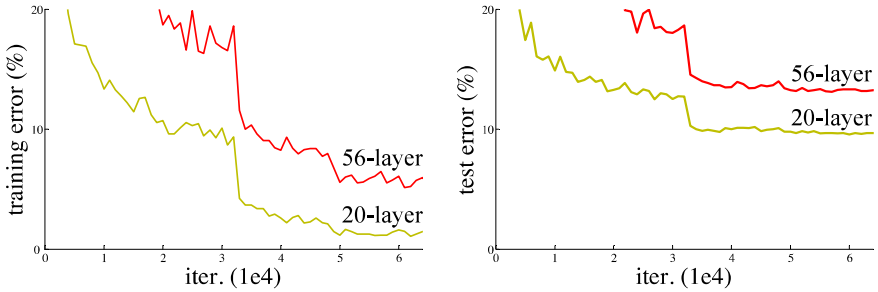


(b) Inception module with dimension reductions

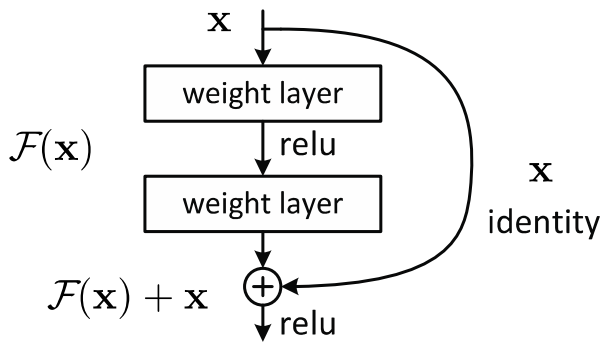
[Going Deep with Convolutions, Szegedy et al., arXiv:1409.4842, 2014]

Residual Networks

Really, really deep convnets do not train well,
E.g. CIFAR10:



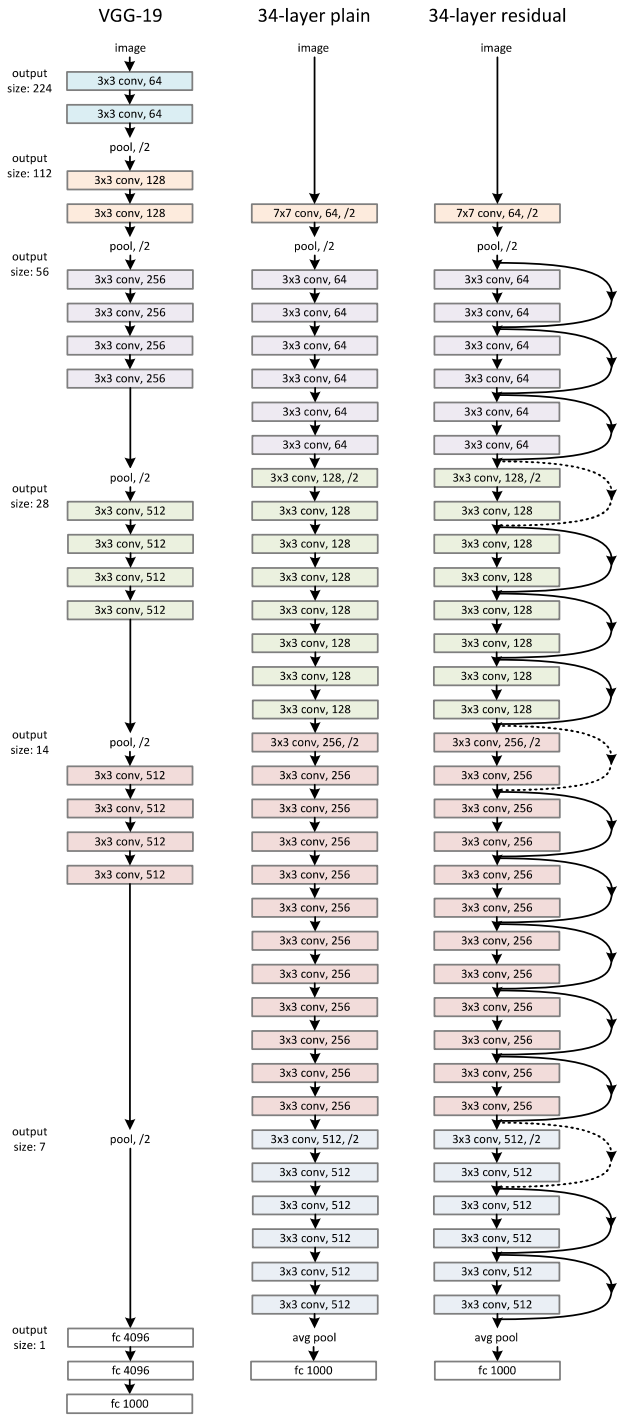
Key idea: introduce "identity shortcut"



method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

With ensembling, 3.57% top-5 test error on ImageNet



[He, Zhang, Ren, Sun, CVPR 2016]

Dense Convolutional Networks

Information in ResNets is only carried implicitly, through addition.

Idea: explicitly forward output of layer to *__all__* future layers (by **concatenation**).

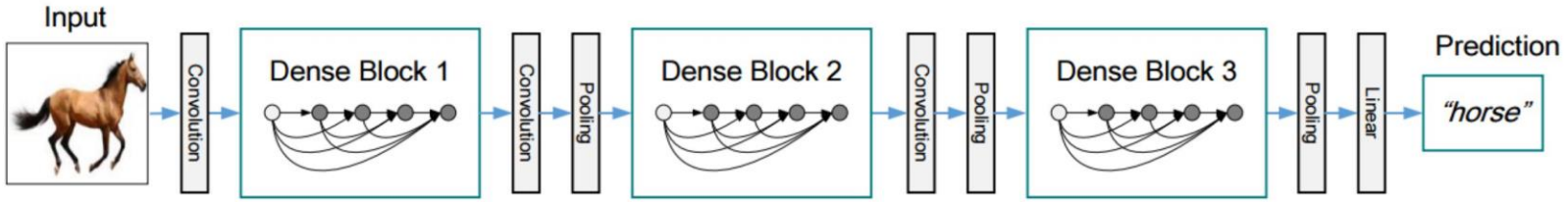
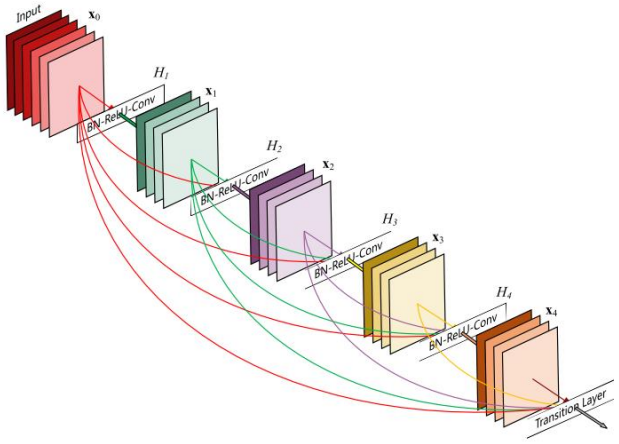


Figure 2. A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature map sizes via convolution and pooling.

Intuition: helps vanishing gradients; encourage reuse features (& hence reduce parameter count);



Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 36$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Full architecture for Imagenet

[Huang, Liu, Weinberger, van der Maaten, CVPR 2017]

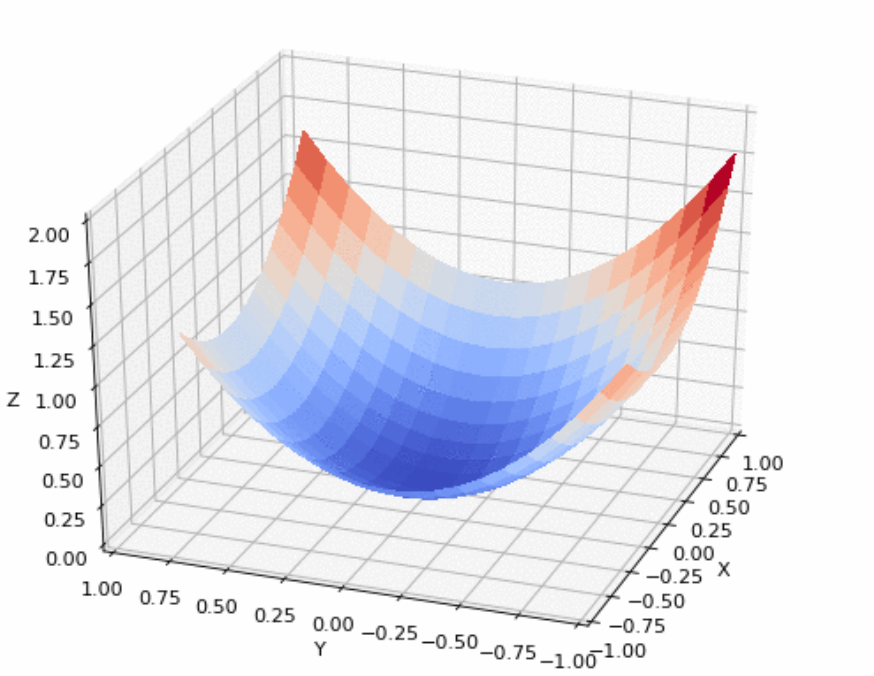
Optimization

How difficult is it to minimize the training loss?

The algorithm: gradient descent (& friends: stochastic, momentum, etc.)

Follow the negative gradient (greedy direction of most improvement)

Remember/reuse previous gradients



Optimization

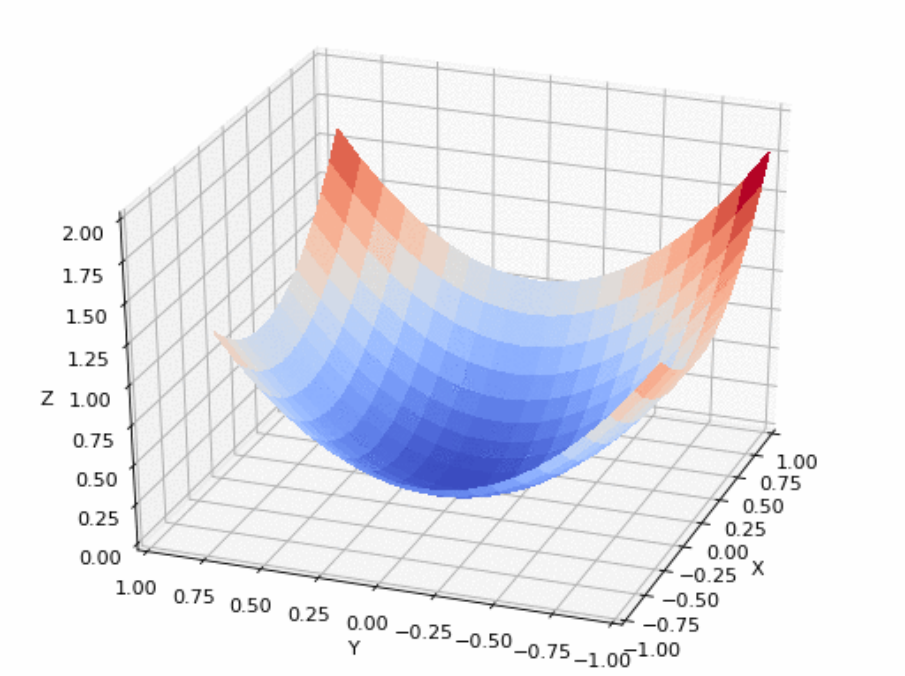
The typical training task in ML can be cast as: $\min_{x \in \mathbb{R}^d} \quad f(x)$

Most algorithms we will look at are iterative: they progressively pick points $x_1, x_2, \ldots$ that are supposed to bring "improvement".

$$x_{t+1} = x_t - \eta \nabla f(x_t)$$

Gradient descent

Optimization

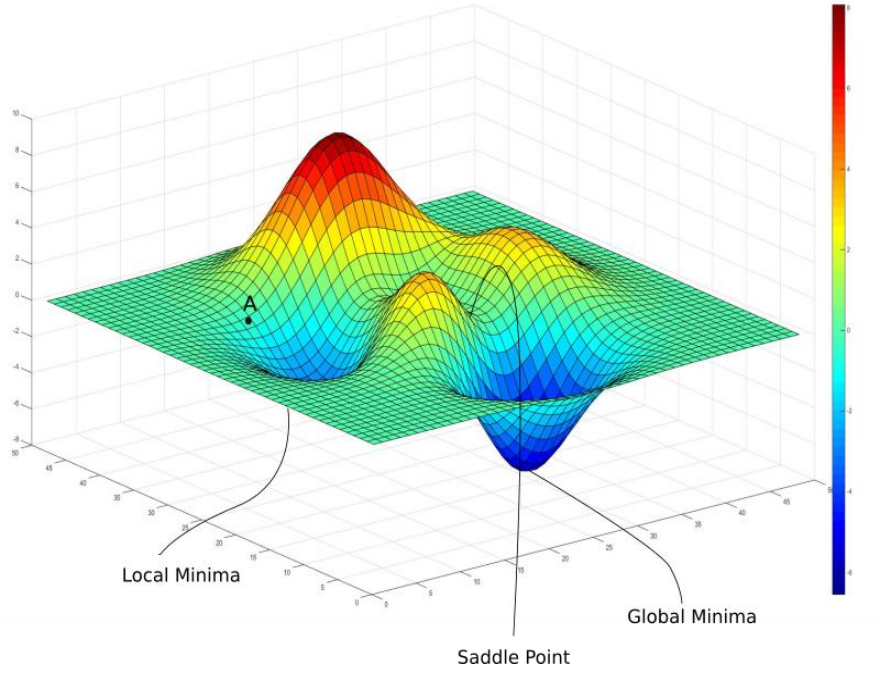


What can we hope for, in the case that $\eta \to 0$?

We stop moving when $\nabla f(x_t) \approx 0$: these care called **stationary points**.

What kinds of stationary points are there?

Optimization



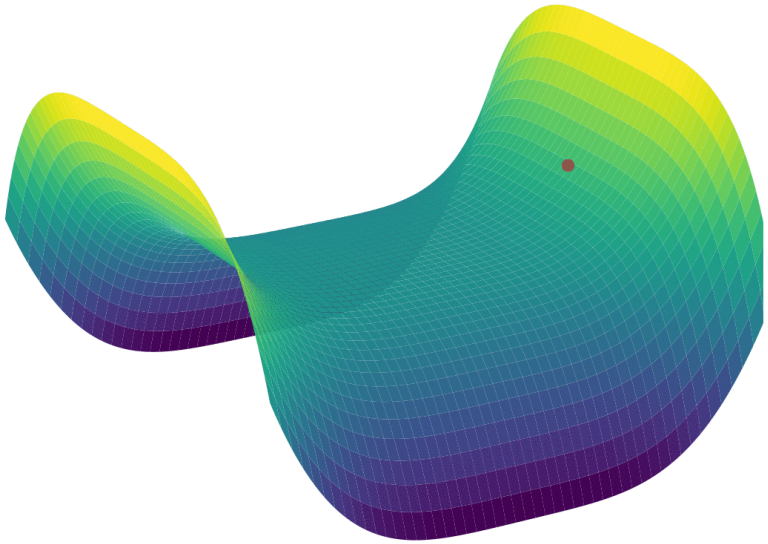
Global minimum: actual minimizer, namely $f(\hat{x}) \leq f(x), \forall x \in \mathbb{R}^d$

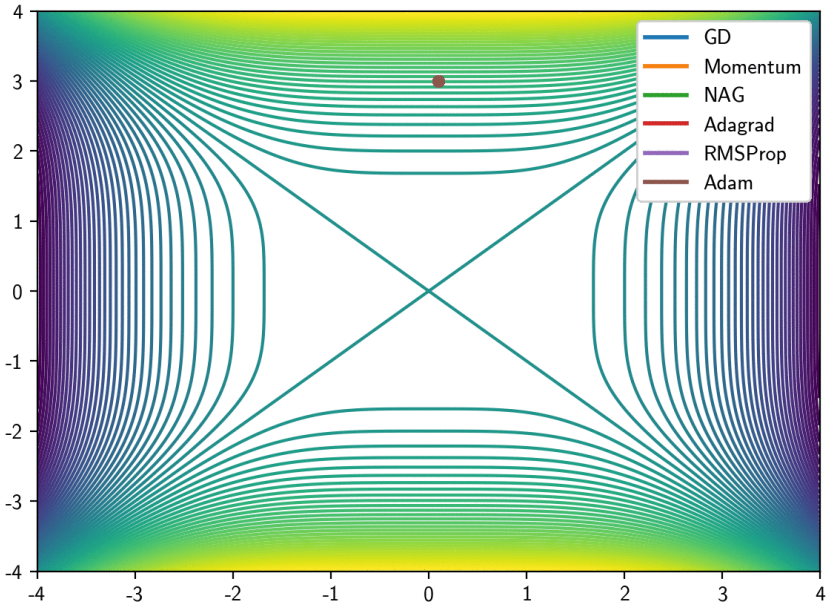
Local minimum: $f(\hat{x}) \leq f(x),\ \forall x \text{ s.t. } ||x - \hat{x}|| \leq \epsilon$ for some $\epsilon > 0$

Local maximum: $f(\hat{x}) \geq f(x),\ \forall x \text{ s.t. } ||x - \hat{x}|| \leq \epsilon$ for some $\epsilon > 0$

Saddle points: stationary point that is *not* a local min/max.

Optimization



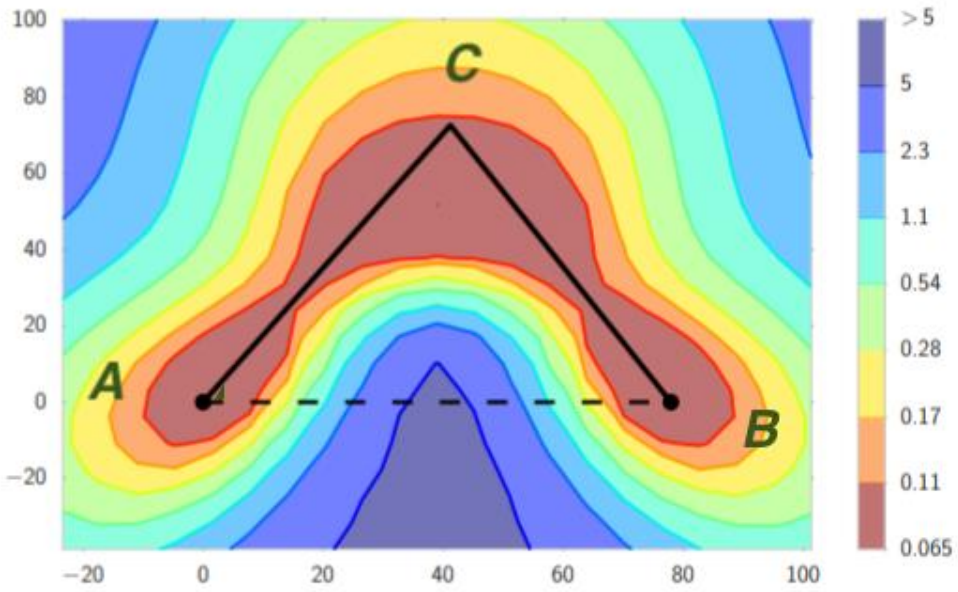


https://tangbinh.github.io/01/04/Optimizers.html

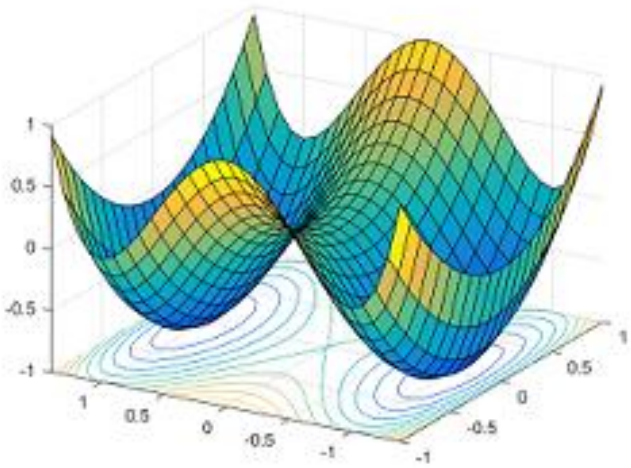
Loss function $f(x, y) = y^4 - x^4$, saddle point at (0,0), minimized at x=∞.

Surprises: mode connectivity

(*Freeman and Bruna, 2016*, *Garipov et al. 2018*, *Draxler et al. 2018*): Local minima can be connected by simple paths of near-same cost.



Surprising!
Remember our intuition, permutations should give rise to isolated solutions



Some explanations due to [*Kuditipudi et al '19*]: robustness of the network to "dropping out" some neurons can ensure this.

Surprises: lottery hypothesis

(*Frankle, Carbin ICLR '20, Best paper award):* Typical neural nets have **much** smaller subnetworks that "could have been trained" in isolation to give comparable results.

More precisely, the following works reliably:

1. Initialize a network and train it.
2. "Prune" superfluous structure. (e.g. smallest weights)
3. Reset unpruned weights to values in 1.
4. Repeat 2+3.

B/w 15% and 1% of the original network!

The small network won the "**initialization lottery**" – it could've been trained to a good network w/o rest of weights.

We're training networks that are way larger than in principle necessary!

Generalization

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{(x,y):\text{training samples}} l(f(x), y)$$

Basic question: if (x, y) are sampled i.i.d from distribution $\mathcal{D}$,

is $\mathbb{E}_{(x,y)\sim\mathcal{D}} l\left(\hat{f}(x), y\right)$ comparable to training loss?

Traditional view: "**gap**" between training loss and expected loss on $\mathcal{D}$ depends on "**complexity**" of class $\mathcal{F}$ (the more complex, the more training data is necessary). Basic VC dimension bounds on neural networks.

Classical view of generalization

Meta-"theorem" of generalization: with probability $1-\delta$ over the choice of a training set of size m, we have

$$\sup_{f\in\mathcal{F}} \left| \widehat{\mathbb{E}}\, l(f(x),y) - \mathbb{E}\, l(f(x),y) \right| \leq O\left(\sqrt{\frac{\text{complexity}(\mathcal{F}) + \ln 1/\delta}{m}}\right)$$

Some measures of complexity:

(Effective) number of elements in $\mathcal{F}$

VC (Vapnik-Chervonenkis)

Rademacher complexity

Covering dimension

Bounding the VC dimension of a neural network

We will recursively use this to bound VC dimension of neural nets with binary activation function.

We will show: VC-dimension of **fully connected net** with **N** edges in total is O(**N log N**)

Main idea: growth function behaves nicely wrt to compositions and concatenations.

Generalization

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{(x,y):\text{training samples}} l(f(x), y)$$

Basic question: if (x, y) are sampled i.i.d from distribution $\mathcal{D}$, is $\mathbb{E}_{(x,y)\sim\mathcal{D}} l\left(\hat{f}(x), y\right)$ comparable to training loss?

Modern view:

Modern networks don't seem to fit this paradigm (size is large enough to fit random labels).

There are instances where larger models don't overfit, but have *better* generalization (**double-descent** phenomenon).

Conjectures and speculations: **shallow vs flat** minima.

Problems with the standard view I

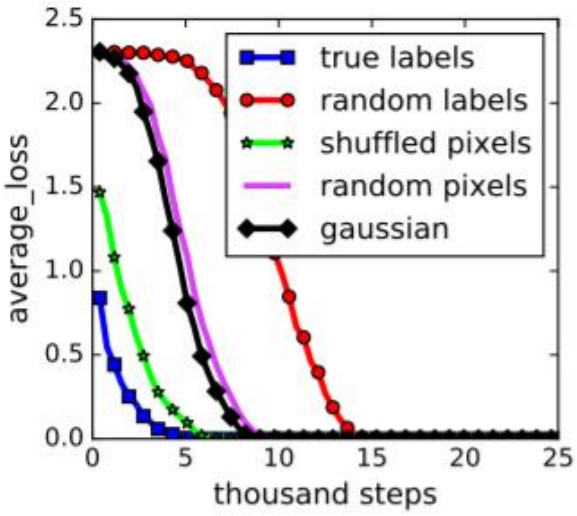
Modern models are **very** complex, by any measure of complexity. (*Not uncommon*: # of params in neural net >> # training samples)

Observations in seminal paper of *Zhang-Bengio-Hardt-Recht-Vinyals '17:*

Modern architectures can fit **random noise!**

Can achieve 0 training error even if we:

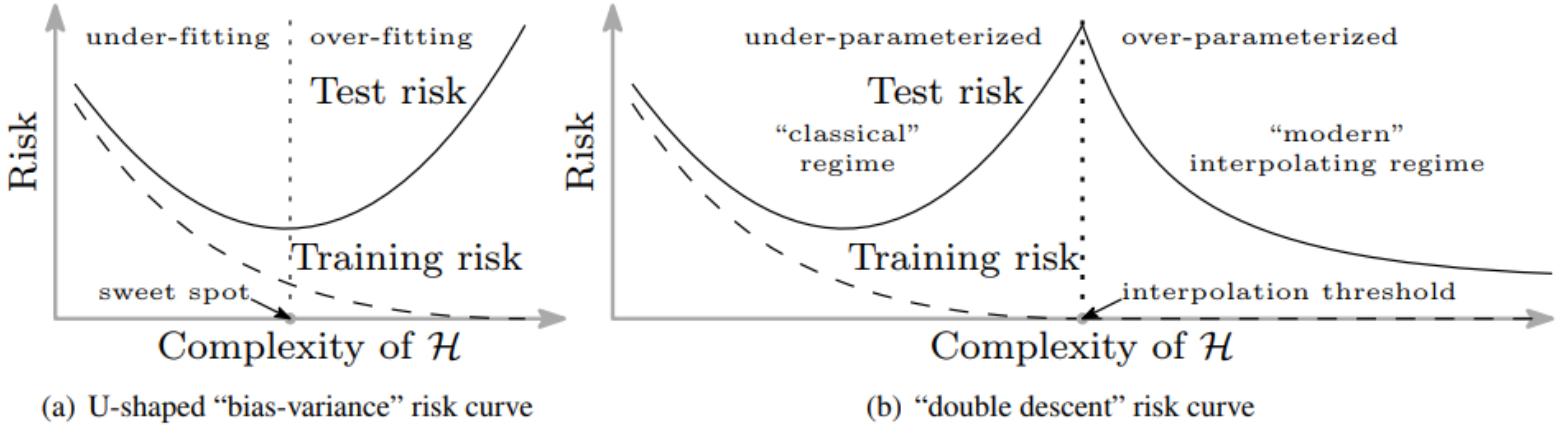
- Randomly permute pixels by a fixed permutation.
- Randomly permute pixels using a different permutation for each image.
- Replace the true labels by random labels;



(a) learning curves

Problems with the standard view II

Classical view of the "model complexity"/"over-fitting" curve is broken, as observed in a seminal paper by *Belkin-Hsu-Ma-Mandal '18*



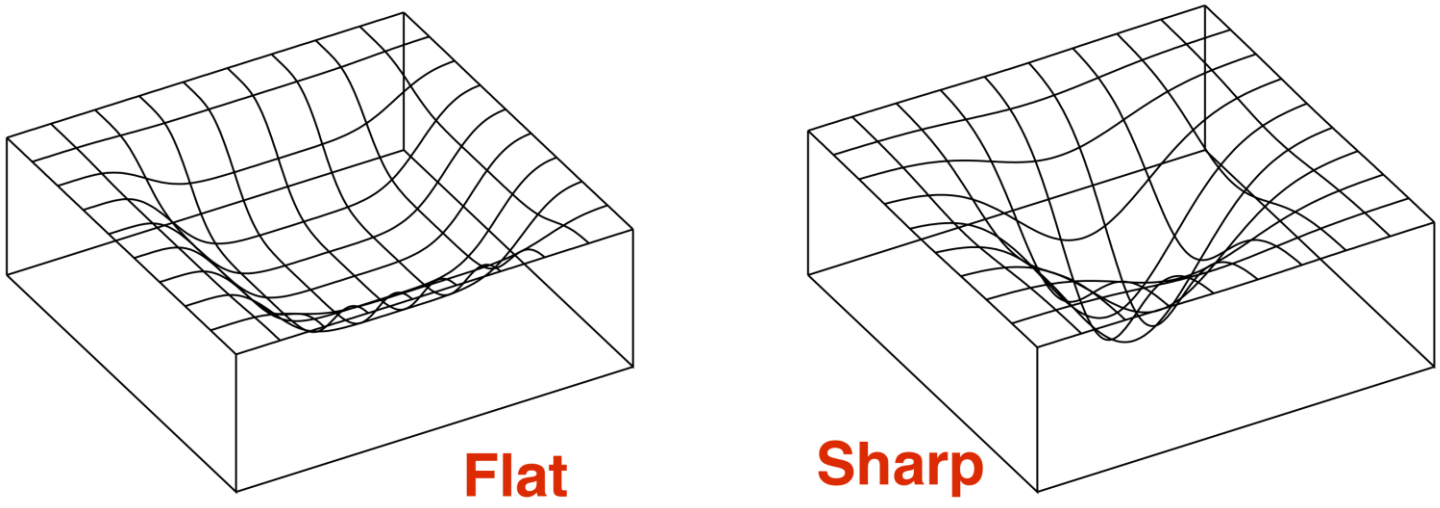
(a) U-shaped "bias-variance" risk curve

(b) "double descent" risk curve

There are cases where the model gets bigger, yet the (test!) loss goes down, sometimes even lower than in the "underparametrized" regime.

Flat vs sharp minima

The practical observation: “flat” minima generalize better.
(Hochreiter and Schmidhuber 1995, Keskar et al 2016, Hinton and Camp 1993)



Original intuition (Hinton and Camp 1993): if the minimum occupies “volume” V, the number of distinct “minima regions” is at most total_volume/V.
Hence, the “cardinality” of the neural nets corresponding to flat minima is on the order of total_volume/V.

Part II: Unsupervised learning

Learning from data **without** labels.

What can we hope to do:

Task A: Fit a parametrized **structure** (e.g. clustering, low-dimensional subspace, manifold) to data to reveal something meaningful about data. (**Structure learning**)

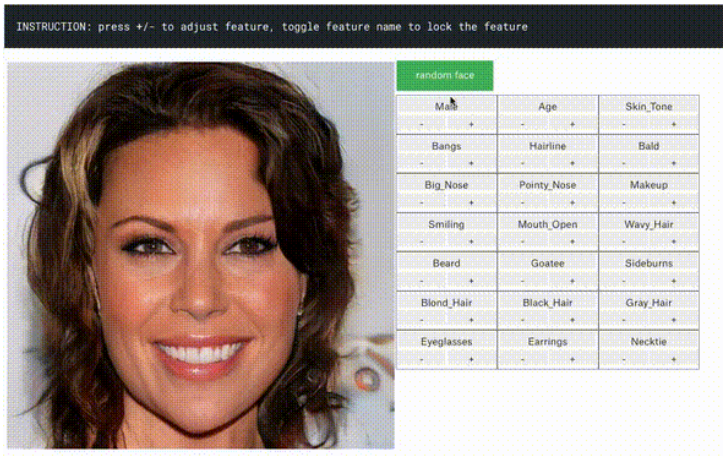
Task B: Learn a (parametrized) **distribution** *close* to data generating distribution. (**Distribution learning**)

Task C: Learn a (parametrized) distribution that implicitly reveals an **"embedding"/"representation"** of data for downstream tasks. (**Representation/feature learning**)

Entangled! The "structure" and "distribution" often reveals an embedding.

Part II: Unsupervised learning

Photorealistic image/video generation, learning to generate new samples, complex conditioning



(Video from Guan et. al ’18)

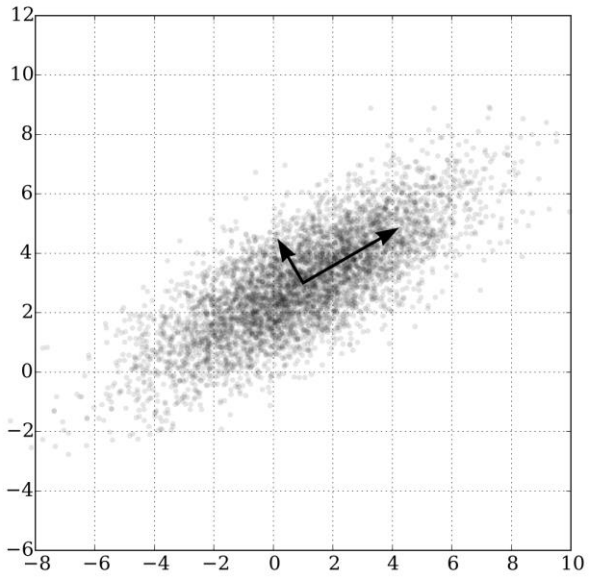
Extracting complex features for downstream applications, domain adaptation

(Video from Zhu et. al ’17)

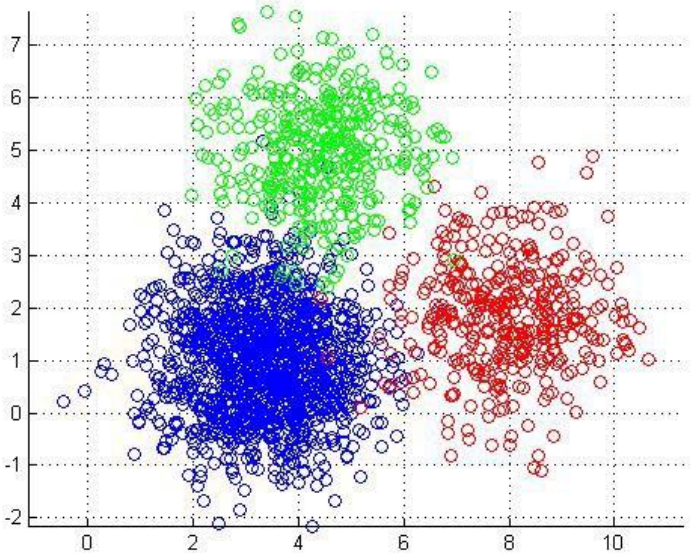
Structure learning

Fit a parametrized **structure** (e.g. clustering, low-dimensional subspace) to data to reveal something meaningful about data.

PCA(principal component analysis), direction of highest variance



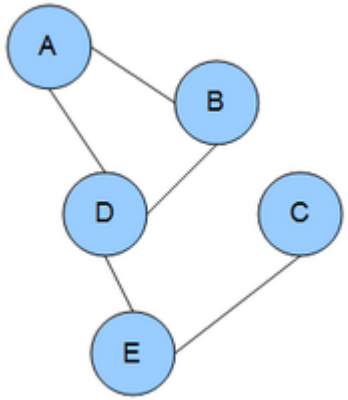
Clustering



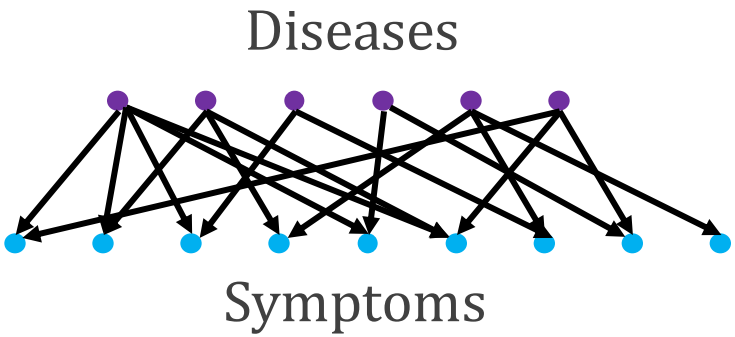
Distribution learning

Some typical choices of parametrized distributions:

Classical choices: fully-observed graphical models (undirected and directed), latent-variable graphical models (mixture models, sparse coding, topic models).



Markov Random Fields: sparse independence structure: "A is independent of other vars, given B, D"

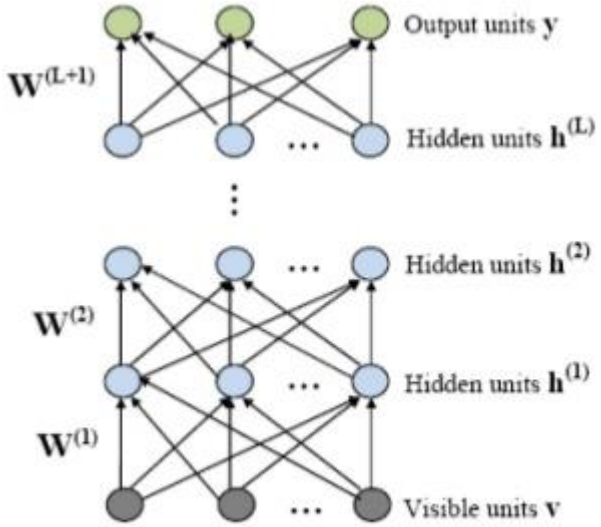


Latent variable models: data is "simple" conditioned on some unobserved (latent) variables

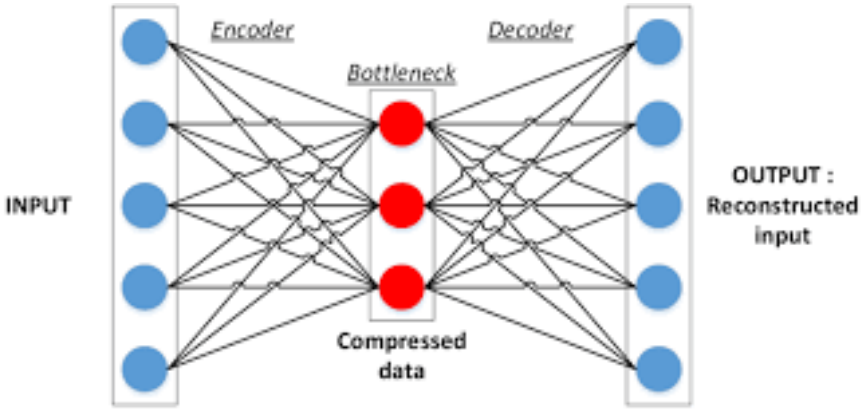
Distribution learning

Some typical choices of parametrized distributions:

Semi-modern choices: deep Boltzmann machines, deep belief networks, (variational) auto-encoders, energy models.



Deep Boltzmann machines, belief networks: graphical model analogues of deep neural networks.

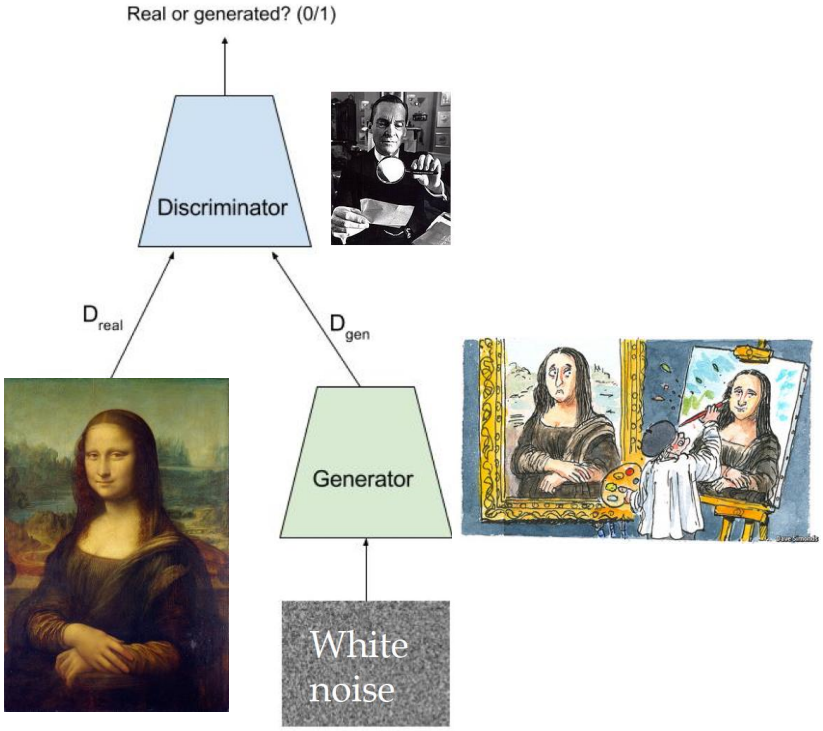


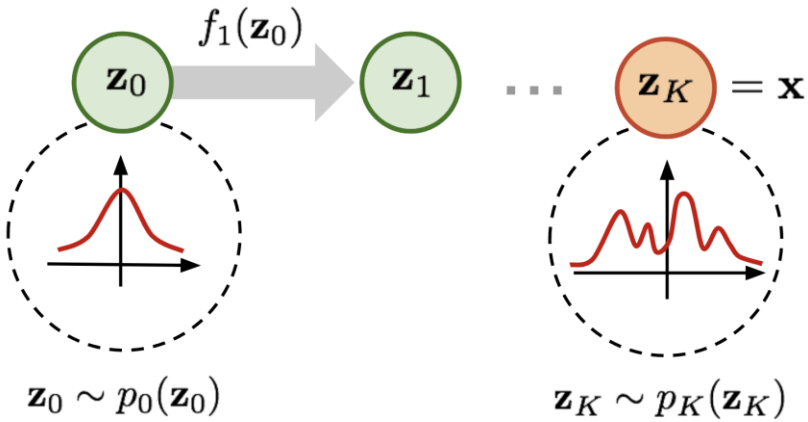
(Variational) autoencoders: model data by enforcing a latent space "bottleneck:

Distribution learning

Some typical choices of parametrized distributions:

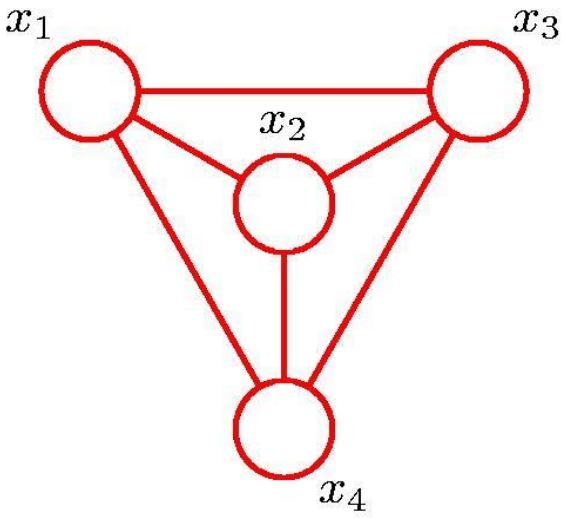
Modern choices: generative adversarial networks, autoregressive models (pixelRNN, pixelCNN), flow models, etc.





Graphical Models

Recall: graph contains a set of nodes connected by edges.



In a probabilistic graphical model, each node represents a random variable, links represent "probabilistic dependencies" between random variables.

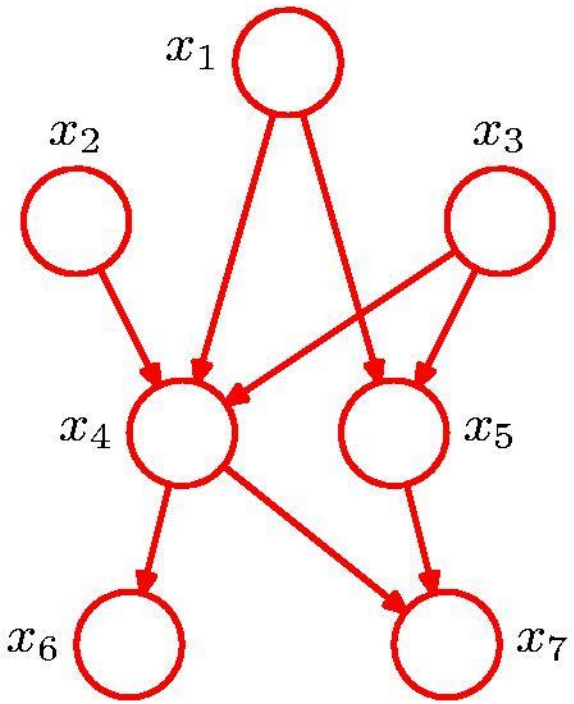
Graph specifies how joint distribution over all random variables decomposes into a **product** of factors, each factor depending on a subset of the variables.

Two types of graphical models:

- **Bayesian networks**, also known as Directed Graphical Models (the links have a particular directionality indicated by the arrows)
- **Markov Random Fields**, also known as Undirected Graphical Models (the links do not carry arrows and have no directional significance).

Bayesian networks

The joint distribution defined by the graph is given by the product of a conditional distribution for each node conditioned on its parents:



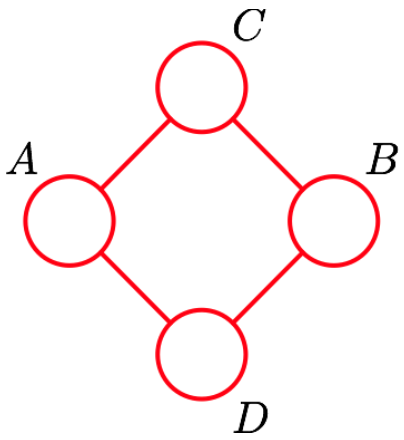
$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k|\mathrm{pa}_k)$$

where pa_k denotes a set of parents for the node x_k.

Each of the conditional distributions will typically have some parametric form. (e.g. product of Bernoullis in the noisy-OR case)

Important restriction: There must be **no directed cycles!** (i.e. graph is a DAG)

Undirected graphical models or Markov Random Fields (MRFs)



More generally, we'd like to be able to define distribution in terms of "local" interactions.

The correct way to formalize this is in terms of **maximal cliques** C (clique = fully connected subset of nodes).

$$p(x) \propto \Pi_C \phi_C(x_C)$$

For example, the joint distribution above factorizes as:

$$p(A,B,C,D) \propto \phi_{AC}(A,C)\phi_{BC}(B,C)\phi_{AC}(B,D)\phi_{AD}(A,D)$$

Modeling pros/cons of directed and undirected models

MRFs

- Hard to draw samples ✗

(In fact, #P-hard provably, even in Ising models)

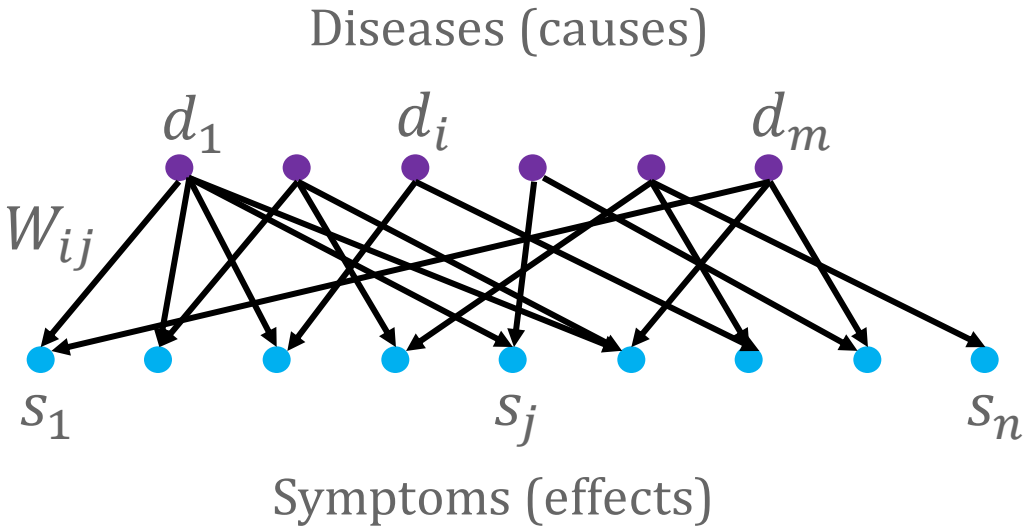
- Models independence structure directly ✓
- Captures soft constraints/energy

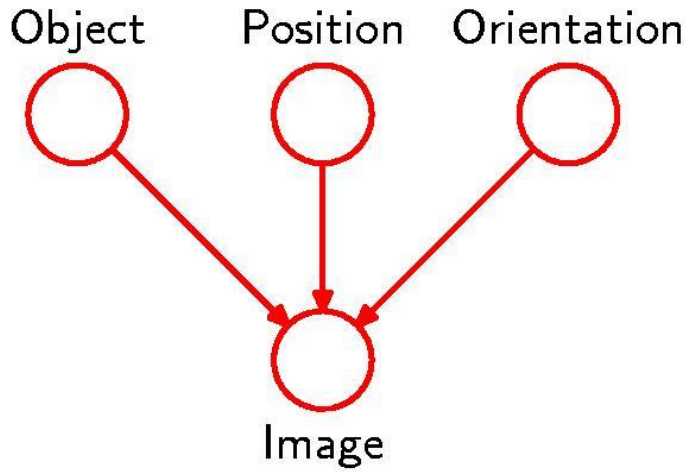
Bayesian networks

- Easy to draw samples ✓
- Models independence structure indirectly ✗
- Captures “causal structure”

Latent variable models

More often than not, we need to model part of the data that is **not observable**. We already saw examples of this:



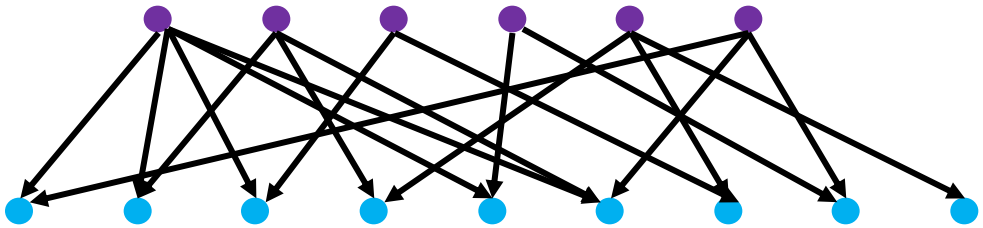


This is also a natural way to extract **features/representation**: the latent variables contain "meaningful" information.

Bayesian networks with latent variables

Simple, but powerful paradigm:
single-layer Bayesian networks, where top nodes are latent.

Latent variables Z



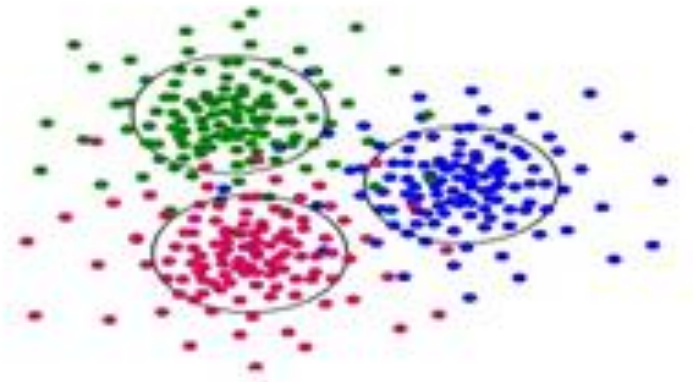
Observable variables X

$$p_\theta(X, Z) = p_\theta(Z)\, p_\theta(X|Z)$$

Example 1: Mixture distributions

Mixture models: observables = points; latent = clustering

To draw a sample (X,Z):
Sample *Z* from a categorial distr. on K components with parameters $\{\pi_i\}$
Sample X from the corresponding component in the mixture.



$$\forall k : \pi_k \geqslant 0 \qquad \sum_{k=1}^{K} \pi_k = 1$$

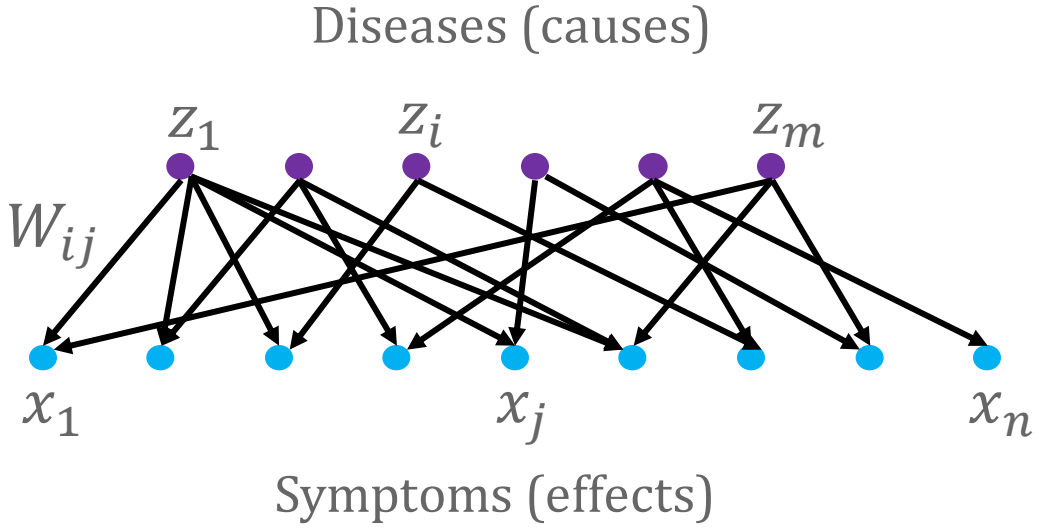
$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Component

Mixing coefficient

Example 2: Noisy-OR networks

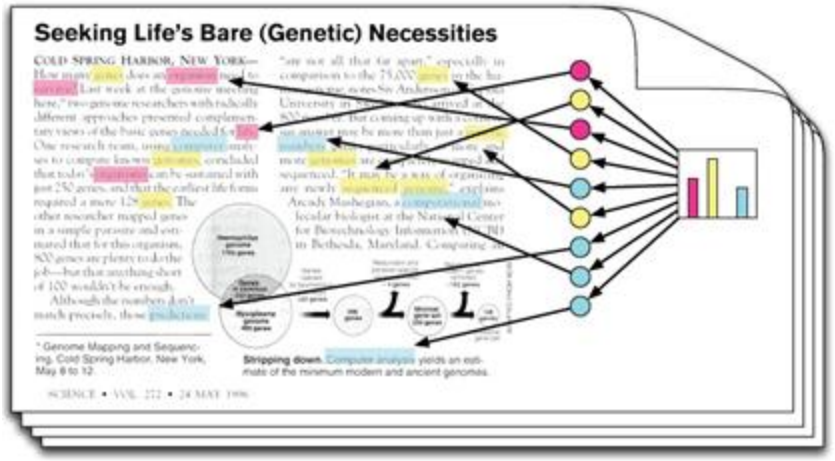
$x_i, z_j \in \{0,1\}$
$W_{ij} \geq 0$



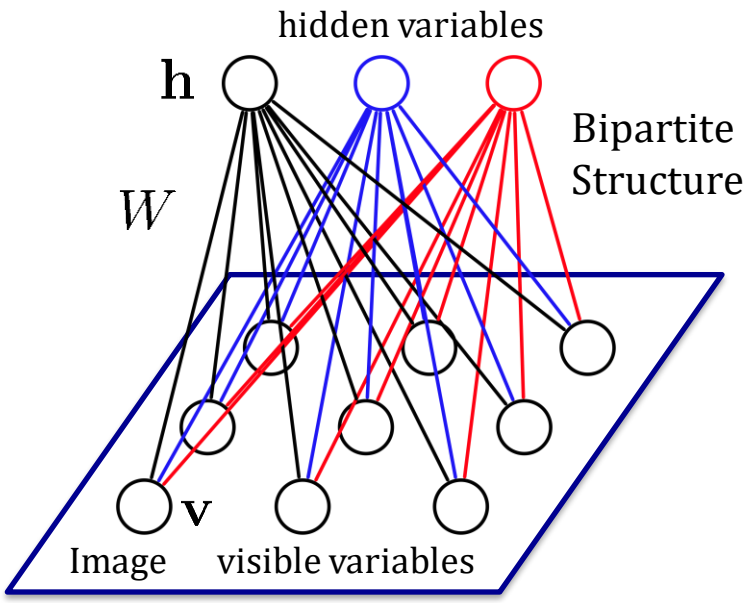
- Sample each z_i is on **independently** with prob. ρ
- When z_i is on, it **activates** x_j with probability $1 - \exp(-W_{ij})$.
  - x_j is **on** if one of z_i's **activates** x_j

Example 3: Topic models (LDA)

Latent Dirichlet Allocation: famous model for modeling topic structure of documents of text. (Blei, Ng, Jordan '03)



Restricted Boltzmann Machines



The **posterior** over the hidden variables is easy to sample from!
(Conditional independence!)

$$p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x}) \qquad p(h_j = 1|\mathbf{x}) = \frac{1}{1 + \exp(-(b_j + \mathbf{W}_j.\mathbf{x}))}$$

Factorizes

Similarly:

$$p(\mathbf{x}|\mathbf{h}) = \prod_k p(x_k|\mathbf{h}) \qquad p(x_k = 1|\mathbf{h}) = \frac{1}{1 + \exp(-(c_k + \mathbf{h}^\top \mathbf{W}_{.k}))}$$

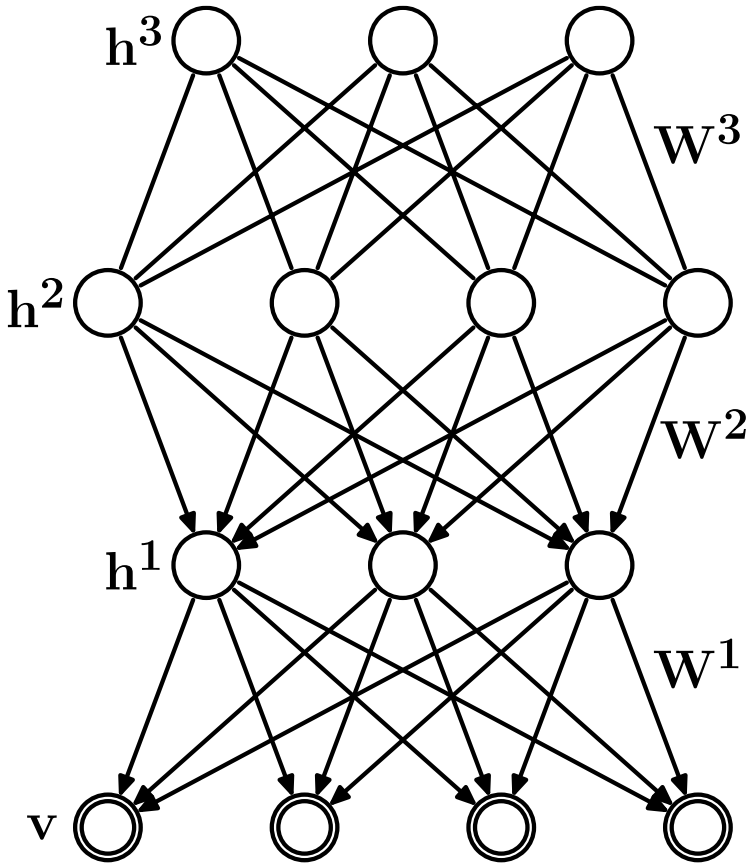
Deep Belief Network

Approximate Inference

$Q(\mathbf{h}^3|\mathbf{h}^2)$

$Q(\mathbf{h}^2|\mathbf{h}^1)$

$Q(\mathbf{h}^1|\mathbf{v})$



Generative Process

$P(\mathbf{h}^2, \mathbf{h}^3)$

$P(\mathbf{h}^1|\mathbf{h}^2)$

$P(\mathbf{v}|\mathbf{h}^1)$

$$Q(\mathbf{h}^t|\mathbf{h}^{t-1}) = \prod_j \sigma\left(\sum_i W^t h_i^{t-1}\right)$$

$$P(\mathbf{h}^{t-1}|\mathbf{h}^t) = \prod_j \sigma\left(\sum_i W^t h_i^t\right)$$

Canonical tasks with graphical models

Inference

Given values for the parameters θ of the model, *sample/calculate* marginals (e.g. sample $p_\theta(x_1), p_\theta(x_4, x_5), p_\theta(z|x)$, etc.)

Learning

Find values for the parameters θ of the model, that give a *high likelihood* for the observed data. (e.g. canonical way is solving maximum likelihood optimization

$$\max_{\theta \in \Theta} \sum_{i=1}^{n} \log p(x_i)$$

Other methods exist, e.g. method of moments (matching moments of model), but less used in deep learning practice.

Canonical tasks with graphical models

Inference

Inference is hard in undirected fully-observable models (due to partition function); easy in fully-observable Bayesian nets.
It's easy for RBM's, hard for latent-variable Bayesian nets (again, implicit normalizing factor is hard.)

Learning

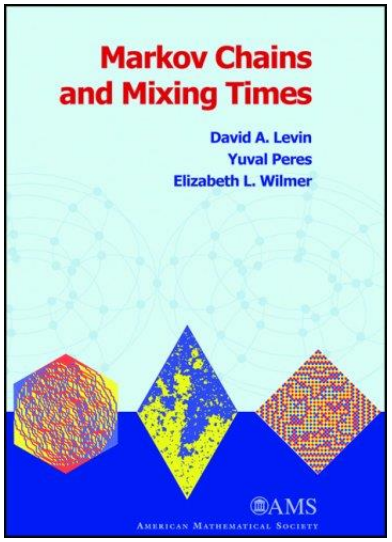
We will derive "iterative"/"incremental" learning algorithms, using inference algorithms as subroutines.
We will, in particular see how a technique called "variational methods" can be used. (Next time, we see how MCMC methods can be used.)

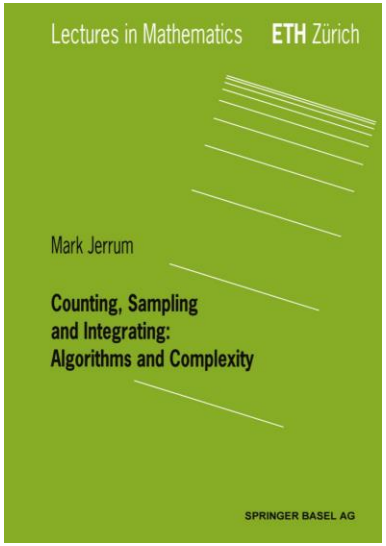
Algorithmic approaches

When faced with a difficult to calculate probabilistic quantity (partition function, difficult posterior), there are two families of approaches:

MARKOV CHAIN MONTE CARLO

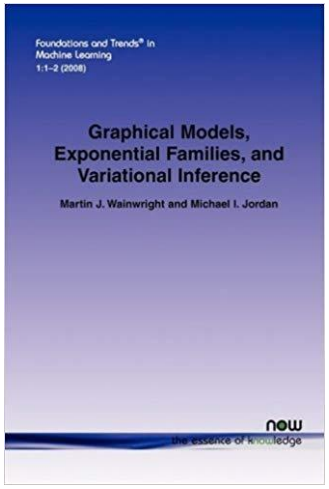
- Random walk w/ equilibrium distribution the one we are trying to sample from.
- Well studied in TCS.





VARIATIONAL METHODS

- Based on solving an optimization problem.
- Very popular in practice.
- Comparatively poorly understood



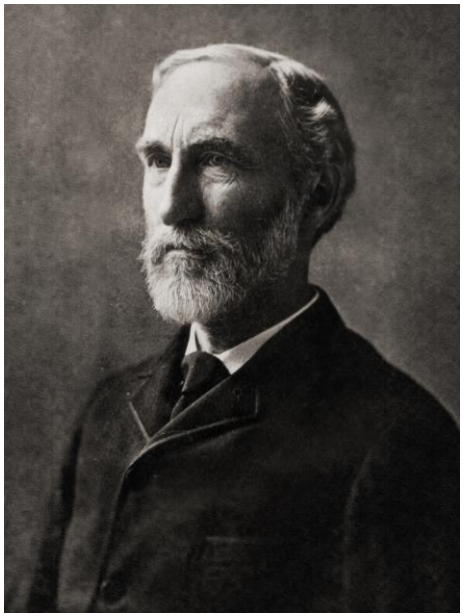
Variational methods for partition functions

Gibbs variational principle: Let $p(x) = \frac{1}{Z}\exp\big(E(x)\big)$ be a distribution over a domain $\mathcal{X}$. Then, Z is the solution to the following optimization problem:

$$\log\ Z = \max_{q\text{: distribution over } \mathcal{X}} H(q) + \mathbb{E}_{x\sim q}[E(x)]$$

Find the distribution that has both high entropy, and high expected energy value

$$H(q) \coloneqq -\sum_{x\in\mathcal{X}} q(x)\log q(x)$$



Variational methods for posterior distributions

ELBO (Evidence Lower Bound): Let $p(z, x)$ be a joint distribution over latent variables and observables. Then:

$$\log\ p(x) = \max_{q(z|x)\text{: distribution over } \mathcal{Z}} H(q|z) + \mathbb{E}_{q(z|x)}[\log p(x, z)]$$

Write, by Bayes rule, $p(z|x) = \frac{p(x,z)}{p(x)}$. Then, the formula above follows by Gibbs variational principle with $\mathrm{E}(\mathrm{x}) = \log p(x, z)$. **Argmax** = p(z|x)!

Gibbs variational principle: Let $p(x) = \frac{1}{Z}\exp\big(E(x)\big)$ be a distribution over a domain $\mathcal{X}$. Then, Z is the solution to the following optimization problem:

$$\log Z = \max_{q\text{: distribution over } \mathcal{X}} H(q) + \mathbb{E}_{x\sim q}[E(x)]$$

Expectation-maximization/ variational inference

The canonical algorithm for learning a single-layer latent-variable Bayesian network is an iterative algorithm as follows.

Consider the max-likelihood objective, rewritten as in the previous slide:

$$\max_{\theta \in \Theta} \max_{\{q_i(z|x_i)\}} \sum_{i=1}^{n} H(q_i(z|x_i)) + \mathbb{E}_{q_i(z|x_i)}[\log p_\theta(x_i, z)]$$

Algorithm maintains iterates $\theta^t, \{q_i^t(z|x_i)\}$, and updates them iteratively

(1) **Expectation (E)-step**:

Keep θ^t fixed, set $\{q_i^{t+1}(z|x_i)\}$, s.t. they maximize the objective above.

(2) **Maximization (M)-step**:

Keep $\{q_i^t(z|x_i)\}$ fixed, set θ^{t+1} s.t. it maximizes the objective above.

Clearly, every step cannot make the objective worse!

Does *not* mean it converges to global optimum – could, e.g. get stuck in a local minimum.

Sampling via random walks

Goal: Sample from distribution given up to constant of proportionality.

Definition: A set of random variables $(X_1, X_2, \dots, X_T)$ is **Markov** if $\forall t: P(X_t|X_{<t}) = P(X_t|X_{t-1})$

It is homogeneous if $P(X_t|X_{t-1})$ doesn't depend on t.

We can describe a homogeneous Markov process on a discrete domain $\mathcal{X}$ by a **transition matrix** $T \in \mathbb{R}_+^{|\mathcal{X}|\times|\mathcal{X}|}: T_{ij} = P(X_{t+1} = j \,|X_t = i)$

Clearly, $\forall i, \sum_j \; T_{ij} = 1$. We will also call such process a Markov Chain/ Markov random walk.

Stationary distribution

Stationary distribution: a distribution $\pi = (\pi_1, \ldots \pi_{|\mathcal{X}|})$ is stationary for a Markov walk if $\pi T = \pi$.

Many Markov Chains have unique stationary distributions: after taking many steps, starting with any distribution, we get to the same distribution

$$\forall p_0, \lim_{t \to \infty} p_0 T^t = \pi$$

Name of the game: if we wish to sample from some π, design a Markov Chain which has π as stationary distribution.

If we run chain long enough (??), we can draw samples from something close to π

Gibbs sampling

Repeat:

Let current state be $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$

Pick $i \in [n]$ uniformly at random.

Sample x $\sim P(X_i = x | \boldsymbol{x}_{-i})$

Update state to $\boldsymbol{y} = (x_1, x_2, \dots, x_{i-1}, x, x_{i+1}, \dots, x_n)$

Why does it work? Metropolis-Hastings with appropriate kernel!

Let

$$
\begin{aligned}
q(\mathbf{x}, \mathbf{y}) &= q(\overbrace{(x_1, \dots, x_n)}^{\mathbf{x}}, \overbrace{(x_1, \dots, x_{i-1}, x, x_{i+1}, x_n)}^{\mathbf{y}}) \\
&\doteq \frac{1}{n} P(X_i = x | X_j = x_j, \forall j \neq i) \\
&= \frac{1}{n} \frac{P(\mathbf{y})}{P(X_j = x_j, \forall j \neq i)}
\end{aligned}
$$

Langevin dynamics

Consider sampling from $p(x) = \frac{1}{Z}\exp\big(-f(x)\big)$ with support $\mathbb{R}^d$, f(x) is differentiable and we can efficiently take gradients. (e.g. f(x) is parametrized by a neural network).

A natural random walk:

Gradient descent Gaussian noise

Limit (as $\eta \to 0$) of: $$x_{t+1} = x_t - \eta \nabla f(x_t) + \sqrt{2\eta}\xi_k$$

$$\xi_k \sim N(0, I)$$

Stationary (equilibrium) distr.

$$p(x) = \frac{1}{Z}\exp\big(-f(x)\big)$$

CD-k Algorithm

For each training example $\mathbf{x}$

- Generate a negative sample $\tilde{\mathbf{x}}$ using k steps of Gibbs sampling, starting at the data point $\mathbf{x}$
- Update model parameters:

$$\mathbf{W} \Longleftarrow \mathbf{W} + \alpha \left(\mathbf{h}(\mathbf{x}) \, \mathbf{x}^{\top} - \mathbf{h}(\tilde{\mathbf{x}}) \, \tilde{\mathbf{x}}^{\top} \right)$$

$$\mathbf{b} \Longleftarrow \mathbf{b} + \alpha \left(\mathbf{h}(\mathbf{x}) - \mathbf{h}(\tilde{\mathbf{x}}) \right)$$

$$\mathbf{c} \Longleftarrow \mathbf{c} + \alpha \left(\mathbf{x} - \tilde{\mathbf{x}} \right)$$

Gradients we derived before

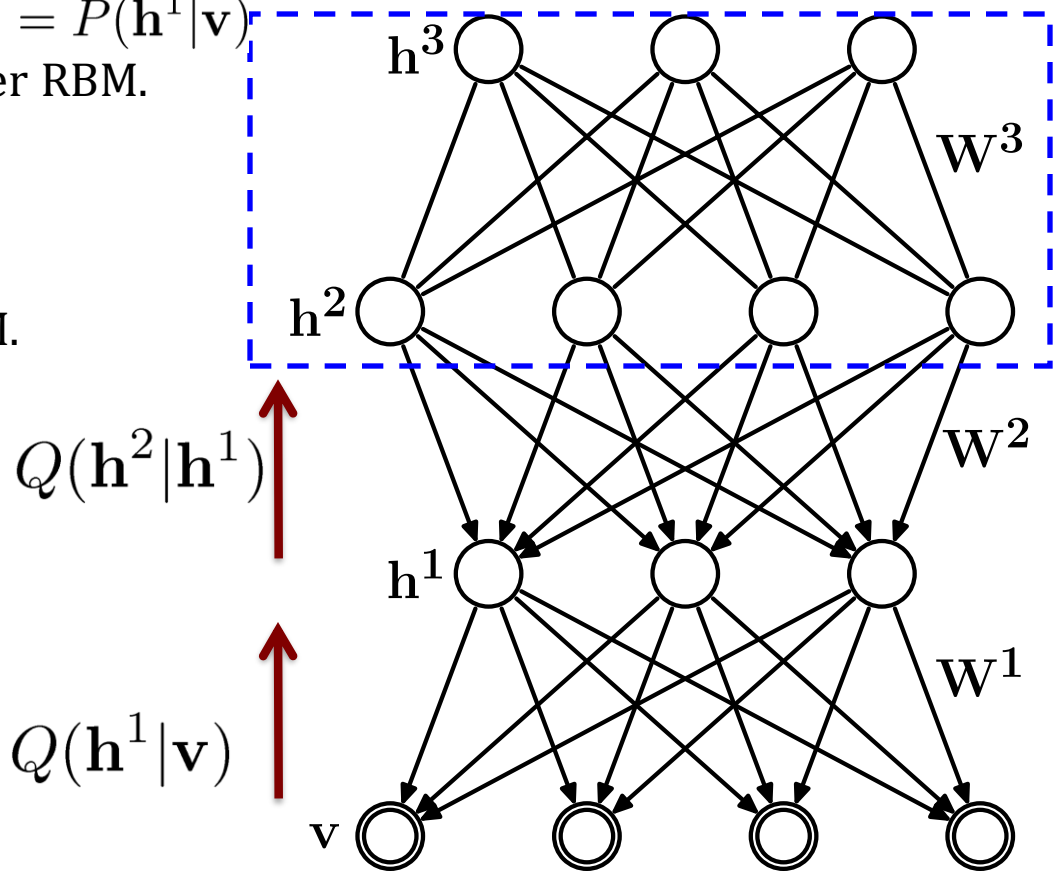
- Go back to 1 until stopping criteria

Step size

DBN Layer-wise Training

- Learn an RBM with an input layer v=x and a hidden layer h.
- Treat inferred values $Q(\mathbf{h}^1|\mathbf{v}) = P(\mathbf{h}^1|\mathbf{v})$ as the data for training 2nd-layer RBM.
- Learn and freeze 2nd layer RBM.
- Proceed to the next layer.

Unsupervised Feature Learning.



The simplest of representation learners

Sparse coding: learn features, s.t. each input can be written as a *sparse linear combination* of some of these features.

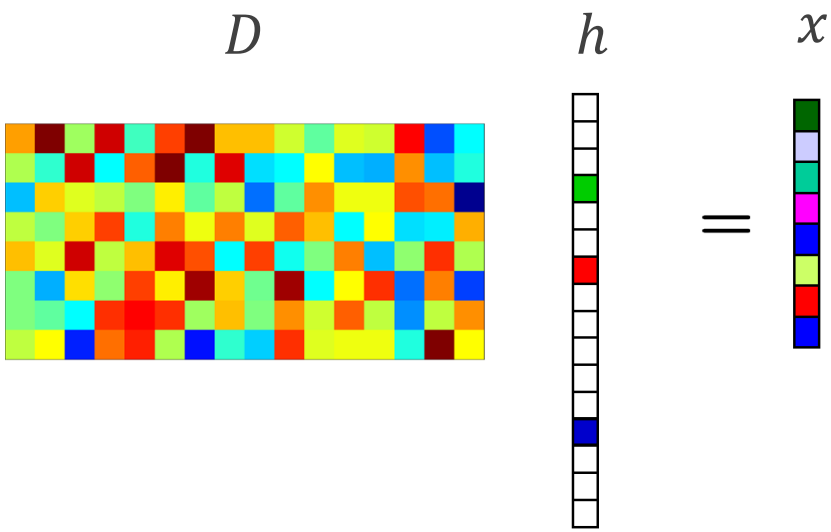
Originally made famous by *Olshausen and Field*, '96 as a model for how early visual processing works (edge detection etc.)

Autoencoders: learn encoding with some constraints (e.g. functional form, sparsity, denoising ability) from which the inputs can be approximately reconstructed.

Sparse coding

Goal: learn a *dictionary* D of features, s.t. each sample x is (approximately) writeable as a *sparse* (i.e. mostly zeros) linear combination of these features.

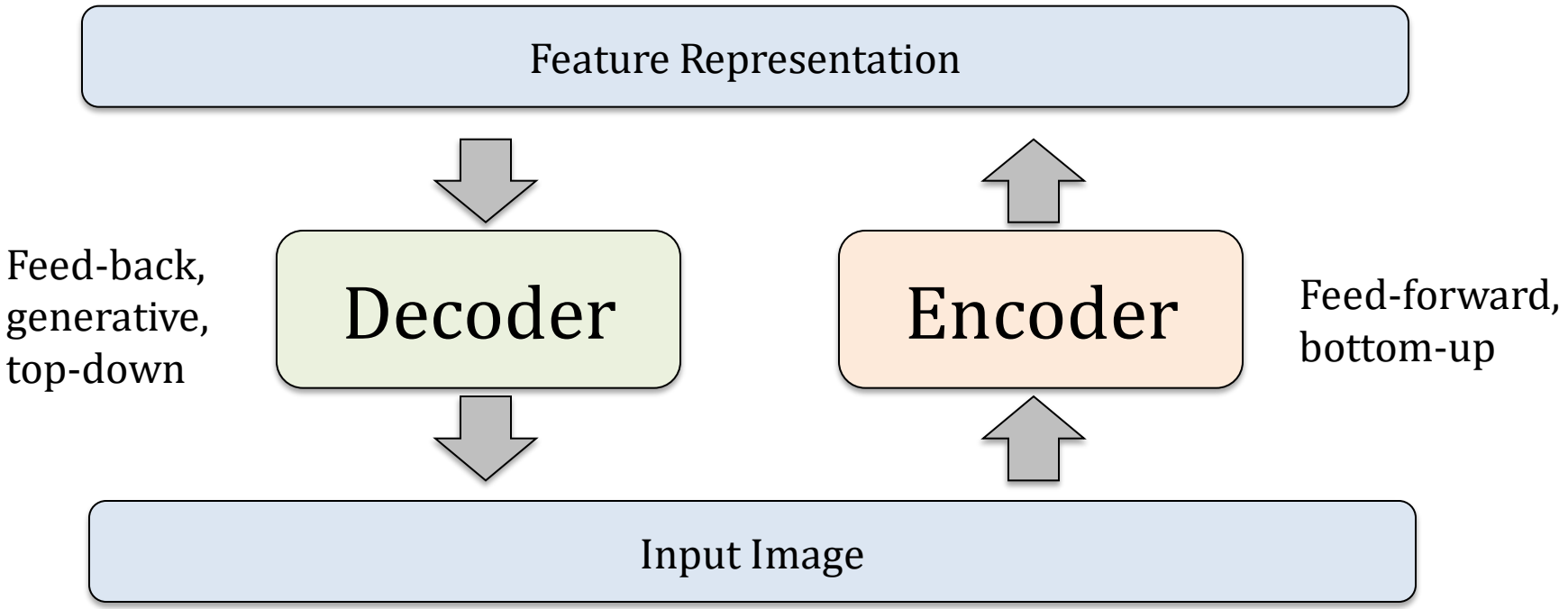
$$\forall x: \quad x \approx Dh, \qquad ||h||_0 \text{ small}$$



h is the representation of sample x

Autoencoders

The idea behind autoencoders: learn features, s.t. input is reconstructable from them

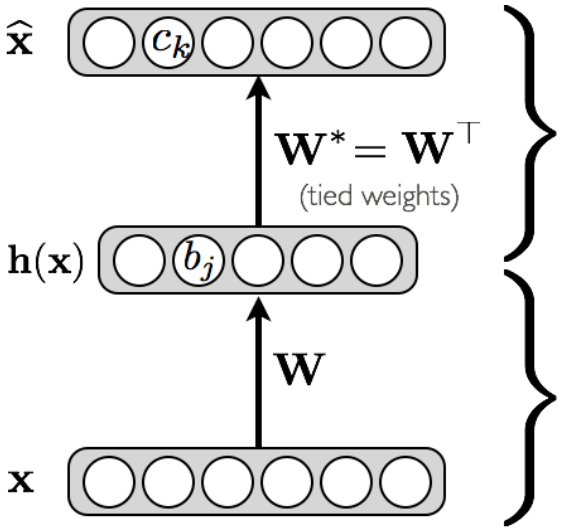


- Details of what goes insider the encoder and decoder matter!
- Need *constraints* to **avoid learning an identity.**

Autoencoders

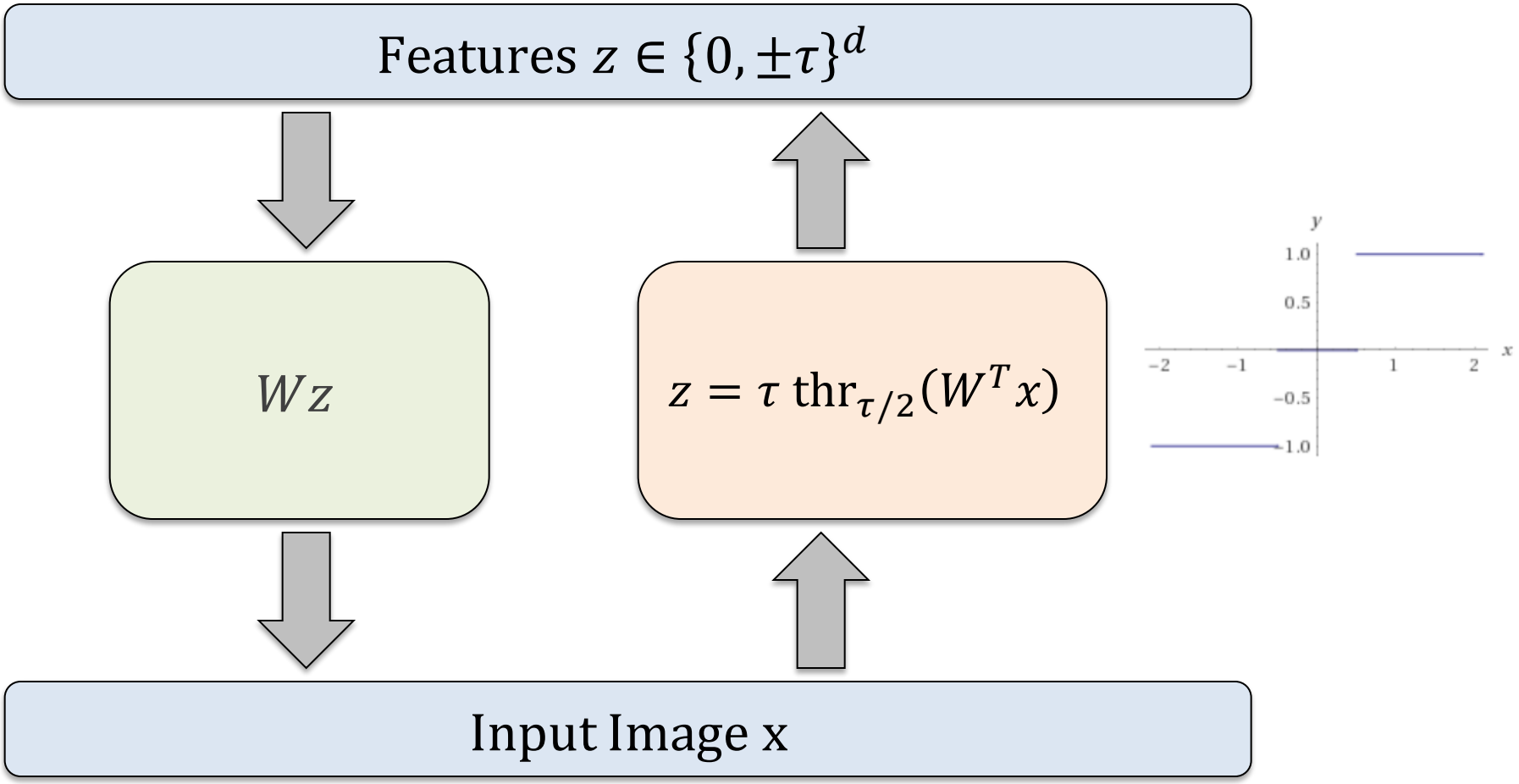
Some way to prevent identity:

- *Weight tying* of encoder/decoder. (Often magical!)
- *Smaller dimension* for latent variables
- Enforce *sparsity* of the latent representation
- Encourage decoder to be robust to adding noise to x. (*Denoising autoencoder*)
- Encode to distribution rather than pointmass. (*Variational autoencoder*)



Intuitions for weight tieing

Setup: sgn activations with weight tieing



Intuitions for weight tieing

Claim: if true x's satisfy $x = Wz + \epsilon$, for W orthogonal, $\left|\left|\epsilon\right|\right|_2 \leq \frac{\tau}{4}$, the above combination of encoder/decoders give a reconstruction error of at most $\left|\left|\epsilon\right|\right|_2$

Same calculation as doing one ISTA step!

Encoder produces: $\text{thr}_{\tau/2}\ (W^T x) = \text{thr}_{\tau/2}(W^T(Wz + \epsilon))$

$= \text{thr}_{\tau/2}(z + W^T\epsilon)$

As $\langle W_{\cdot,k}, \epsilon\rangle \leq \left|\left|\epsilon\right|\right|_2$, encoder produces $z_i + \delta_i, |\delta_i| \leq \frac{\tau}{4}$.

If $|z_i| = \tau$, input to thr is $z_i + \delta_i \in \tau \pm \frac{\tau}{4}$, hence encoder produces z_i

If $z_i = 0$, input to thr is $z_i + \delta_i \in \pm\frac{\tau}{4}$, hence encoder produces 0.

Hence, $\left|\left|\hat{x} - x\right|\right|_2 = ||Wz - x||_2 = \epsilon$, which is small!

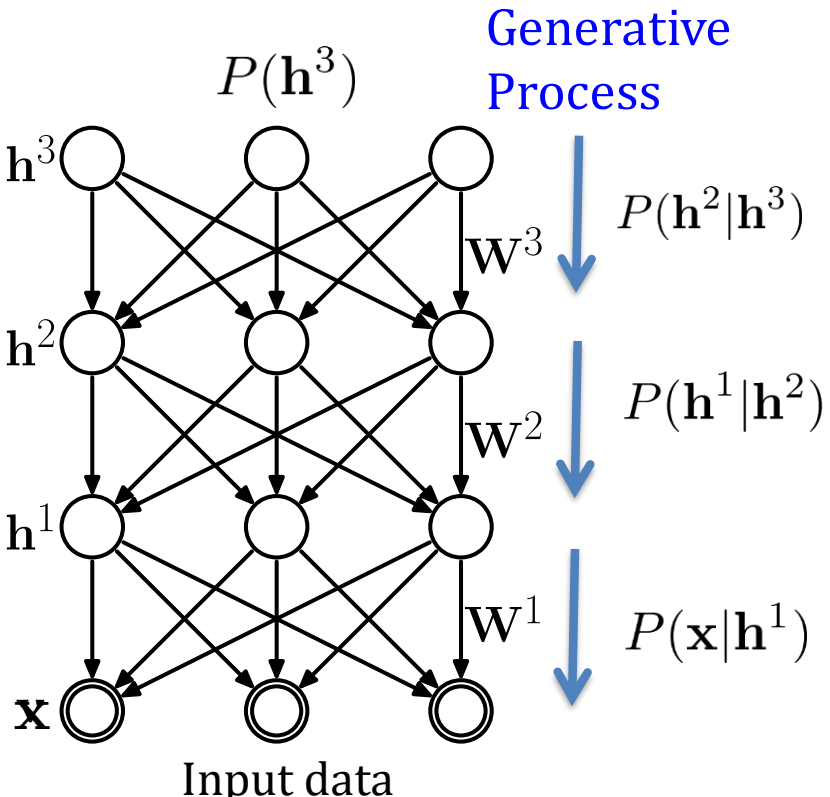
Good reconstruction!!

Variational autoencoders

"*Decoder/generator*": directed Bayesian network with Gaussian layers

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{\mathbf{h}^1,\ldots,\mathbf{h}^L} p(\mathbf{h}^L|\boldsymbol{\theta})p(\mathbf{h}^{L-1}|\mathbf{h}^L,\boldsymbol{\theta})\cdots p(\mathbf{x}|\mathbf{h}^1,\boldsymbol{\theta})$$

Each term may denote a complicated nonlinear relationship



Typically, directed layers are parametrized as:

$$p(\boldsymbol{h}^{L-1}|\boldsymbol{h}^L,\boldsymbol{\theta}) = \mathcal{N}(\mu_{\boldsymbol{\theta}}(\boldsymbol{h}^L), \Sigma_{\theta}(\boldsymbol{h}^L))$$

Gaussians, means/covariances functions (e.g. one-layer neural net) of previous layer and model parameters θ.

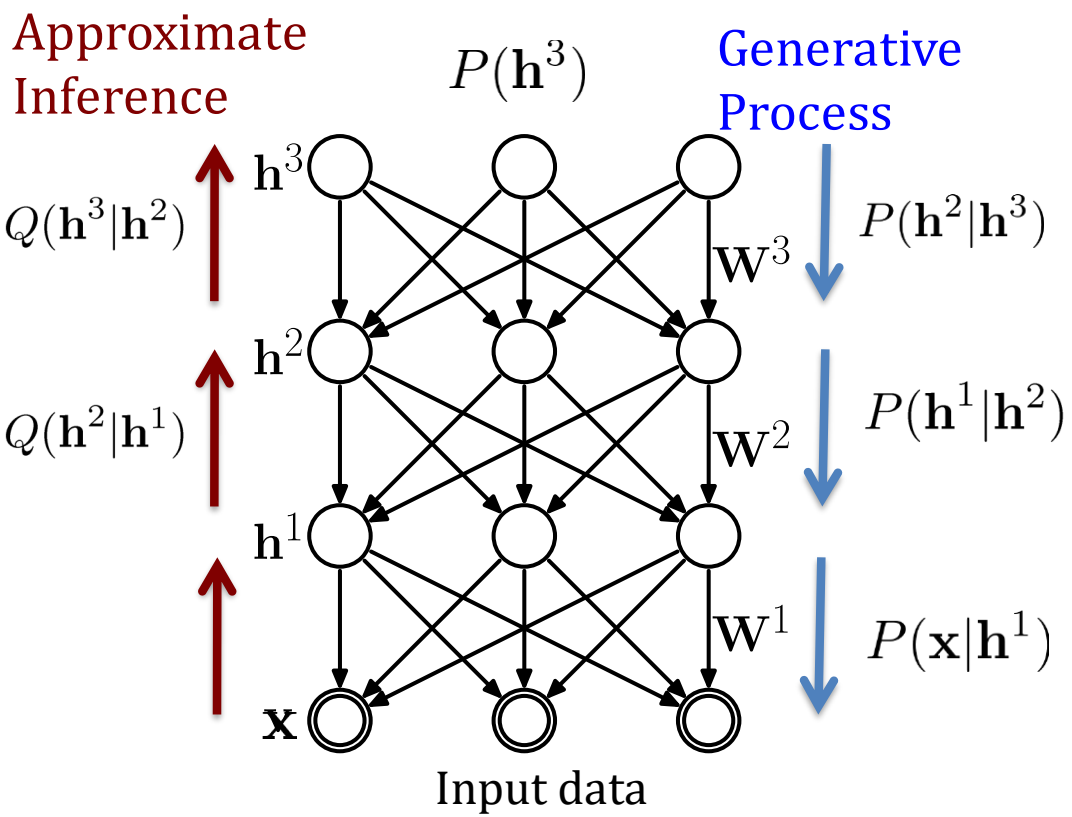
Easy to sample!

Encoder: A "recognition network"

The encoder is defined in terms of an analogous factorization:

$$q(\boldsymbol{h}|\boldsymbol{x},\boldsymbol{\theta}) = q(\boldsymbol{h}^1|\boldsymbol{x},\boldsymbol{\theta})q(\boldsymbol{h}^2|\boldsymbol{h}^1,\boldsymbol{\theta}) \ldots q(\boldsymbol{h}^L|\boldsymbol{h}^{L-1},\boldsymbol{\theta})$$

Each term may denote a complicated nonlinear relationship



Typically, directed layers are parametrized as:

$$q\left(\boldsymbol{h}^l\middle|\boldsymbol{h}^{l-1},\boldsymbol{\theta}\right) = \mathcal{N}\left(\mu_{\boldsymbol{\theta}}\left(\boldsymbol{h}^{\boldsymbol{l-1}}\right), \Sigma_{\boldsymbol{\theta}}\left(\boldsymbol{h}^{\boldsymbol{l-1}}\right)\right)$$

Means/covariances fns (e.g. one-layer neural net) of previous layer and parameters $\boldsymbol{\theta}$.

Why is this called an "encoder"?

Max-likelihood can be written as:

$$\max_{\theta \in \Theta} \max_{\{q(h^L|x)\}} \sum_{i=1}^{n} H(q(h^L|x)) + \mathbb{E}_{q(h^L|x)}[\log p(x, h^L)]$$

Let's rewrite the ELBO a bit:

$$H\big(q(h^L|x)\big) + \mathbb{E}_{q(h^L|x)}[\log p(x, h^L)] = \mathbb{E}_{q(h^L|x)}[\log p(x, h^L) - \log q(h^L|x)]$$

$$= \mathbb{E}_{q(h^L|x)}[\log p(h^L) + \log p(x|h^L) - \log q(h^L|x)]$$

$$= \mathbb{E}_{q(h^L|x)} \log p(x|h^L) - \mathbb{E}_{q(h^L|x)} \log \frac{q\big(h^L|x\big)}{p(h^L)}$$

$$= \mathbb{E}_{q(h^L|x)} \log p(x|h^L) - KL(q(h^L|x)\big|\big|p(h^L)\big)$$

"Reconstruction" error

Use q as a "probabilistic" encoder,

Use p as a "probabilistic" decoder,

$x \to h^L \to x$

"Regularization towards prior"

How to train?

Max-likelihood can be written as:

$$\max_{\theta\in\Theta} \max_{\{q_\theta(h^L|x)\}} \sum_x \mathbb{E}_{q_\theta(h^L|x)} \log \frac{p_\theta(x,h^L)}{q_\theta(h^L|x)}$$

As usual: we need to be able to take gradients in θ

The solution: write the expectation $\mathbb{E}_{q_\theta(h^L|x)} \log \frac{p_\theta(x,h^L)}{q_\theta(h^L|x)}$ as an expectation over a distribution not dependent on θ.

Kingma-Welling '13: reparametrization trick!

Main idea: a sample from $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ can be generated as follows

Sample $\mathbf{x} \sim \mathcal{N}(0, \mathrm{I})$.

Output $\boldsymbol{y} = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{1/2}\boldsymbol{x}$.

Desiderata for representations

What do we want out a representation?

Many possible answers here. First, a few uncontroversial desiderata:

Interpretability: if the derived features are semantically meaningful, and interpretable by a human, they can be easily evaluated.
(e.g. noisy-OR: "features" are diseases a patient has)

Sparsity of a representation is an important subcase: "explanatory" features for sample can be examined if there are a small number of them.

Downstream usability: the features are "useful" for downstream tasks. Some examples:

Improving label efficiency: if, for a task, a linear (or otherwise "simple") classifier can be trained on features and it works well, smaller # of labeled samples are needed.

Desiderata for representations

Obvious issue: interpretability and "usefulness" are not easily mathematically expressed. We need some "proxies" that induce such properties.

This is a lot more contraversial – here we survey some general desiderata, proposed as early as *Bengio-Courville-Vincent '14:*

Hierarchy/compositionality: video/images/text/ are expected to have hierarchical structure – depth helps induce such structure.

Semantic clusterability: features of the same "semantic class" (e.g. images in the same category) are clustered.

Linear interpolation: in representation space, linear interpolations produce meaningful data points (i.e. "latent space is convex"). Sometimes called *manifold flattening*.

Disentangling: features capture "independent factors of variation" of data. (*Bengio-Courville-Vincent '14).* Has been very popular in modern unsupervised learning, though many potential issues with it.

The idea behind GANs

Matching a distribution on images is hard because we don't have good measures of "**distance**" between images.
(Intuitively, two images could be very different in pixel space, while "**semantically**" being the same image.)

Why don't we simultaneously train a "distance" metric as we are training the model?

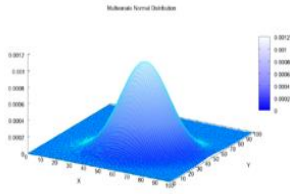
As a result, we will no longer be fitting the "maximum likelihood" model, but instead trying to learn some distribution close to the distribution of the input images in a learned metric.

This is (one of many) models which are "**likelihood-free**": we won't be able to explicitly write a likelihood for the model, but (importantly) we will efficiently be able to draw samples from the model!

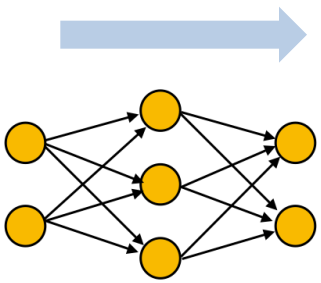
The GAN paradigm (Goodfellow et al. '14)

Goal: **Learn** a distribution close to some distribution we have few samples from. (Additionally, we will be able to sample efficiently from distribution.)

Approach: **Fit** distribution P_g parametrized by **neural network g**



$Z \sim N(0, I_{k\times k})$



Neural network $g(\cdot)$

$X = g(Z)$

W-GAN formalization (Arjovsky et al. '17)

Min-max problem:

- Min-player: generators $g \in G$; Max-player: discriminators f $\in F$.
- Samples from image distr. P_{real}. Unif. distribution over samples: $P_{samples}$
- P_g - generator distribution: $Z \sim N(0, I) \rightarrow g(Z)$

Training loss:

$$\min_{g \in G} \max_{f \in F} \left| \mathbb{E}_{P_g}[f] - \mathbb{E}_{P_{\text{samples}}}[f] \right|$$

Difference of expectation of f on **samples vs generated** images

Tension: strength of discriminators

Small discriminators ⇒ mode collapse:

Generator w/ support size $\approx m$ **fools** neural net discriminators with **≤ m parameters.** [Arora et al'17, Arora-Risteski-Zhang 'ICLR18]

Large discriminators ⇒ poor generalization:

Loss with small # samples differs a lot from loss with infinite # samples.

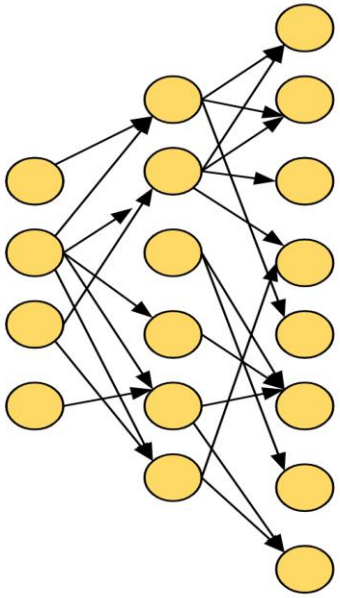
$$d_F(P_{samples}, P_g) \not\approx d_F(P_{real}, P_g)$$

Sweet spot for natural distributions

Let P_{real} **itself** be generated by neural net. ($P_{real} = P_g$, $g \in G$)

Let G = { **1-to-1 neural networks** of bounded size }

Design **small** discriminators F w/ good distinguishing power.



- Less general than arbitrary neural-net generators
- Allows data to lie on **low-dim. manifold**.

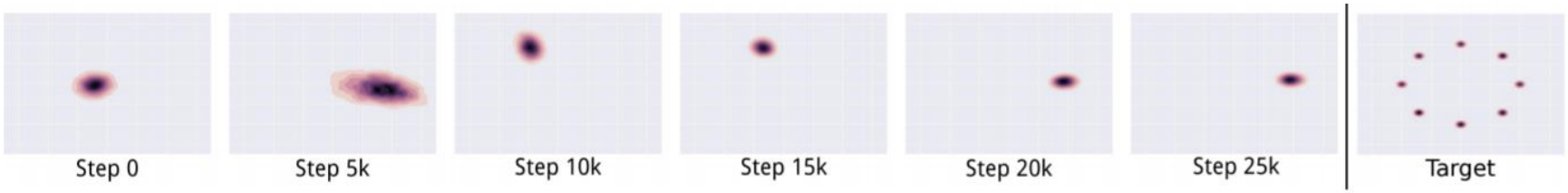
Common training problems

Unstable training: the problem is a min-max problem (also called saddle point problem) – typically optimization is much less stable than pure minimization.

Vanishing gradient: if the discriminator is too good, the generator gradients have a propensity to be small. (This is concerning, as to be taking gradients of the Wasserstein/JS/… objective, the discriminator needs to be optimal.)

Less of a problem with more modern GANs than with DC-GAN.

Mode collapse: the training only recovers some of the modes of the underlying distribution. **(NOT** clear if this is a statistical or algorithmic problem.)



How do we evaluate GANs

Since we cannot evaluate the likelihood of the input data under a generator, evaluation is hard.

(Disproportionately) frequently, the evaluation is done by visually comparing samples – this cannot exclude issues like memorization, mode collapse, etc.

*Can we test for some common **failure modes**?*

Diagnosing small support size: bday paradox

Birthday Paradox:
If there are **23** people in a group, **>** ½ chance that two of them share a birthday.

General version: Suppose a distribution is uniform over N images. Then
Pr[sample of size $\sqrt{N}$ has a duplicate image] > ½.

Birthday paradox test* [Arora-Risteski-Zhang **ICLR'18] :** If a sample of size **s** has **duplicate** images with prob. > ½, then distribution essentially* has only **s^2 distinct images**.

Implementation: Draw sample of size **s**; heuristically flag possible near-duplicates. Use human in the loop to verify duplicates.

Interpolation

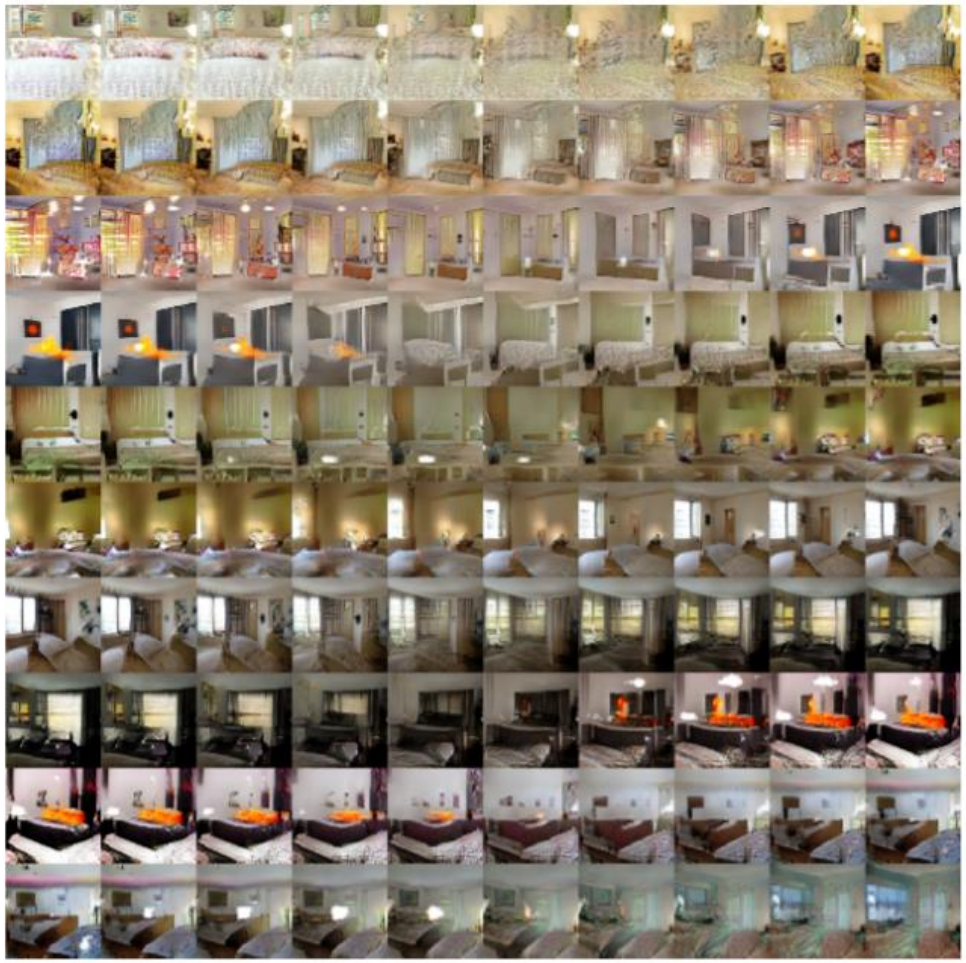


Figure 4: Top rows: Interpolation between a series of 9 random points in Z show that the space learned has smooth transitions, with every image in the space plausibly looking like a bedroom. In the 6th row, you see a room without a window slowly transforming into a room with a giant window. In the 10th row, you see what appears to be a TV slowly being transformed into a window.

If linearly interpolating in latent space gives rise to meaningful images (without sharp transitions), unlikely GAN is just memorizing.

Figure from
Radford, Metz, Chintala '16.

Inception score

Suppose we use trained network – the *Inception* architecture as a **labeler** for images. Inception gives probability over labels y for sample x: p(y|x).

Desirable features of generator: the Inception classifier should be "sure" about the label for most images (p(y|x) should have *low entropy*), and the classes it generates should be diverse (p(y) = $\mathbb{E}_{x\sim P_g} p(y|x)$ should have *high entropy*)

Thus, we want $H\big(p(y|x)\big)$ to be **low**, $H(p(y))$ is **high**.

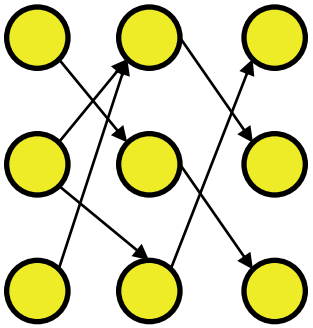
Consider the expression: $\mathbb{E}_{x\sim P_g} KL(p(y|x)||p(y))$

Inception score: $\exp\big(\mathbb{E}_{x\sim P_g} KL\big(p(y|x)\big|\big|p(y)\big)\big)$

Middle ground: "Invertible GANs/Flow models"

Can we "marry" likelihood models w/ GANs?

Suppose generator g: $\mathbb{R}^d \to \mathbb{R}^d$ were **invertible**.



Recall from the prev. lecture: if we denote by $\phi(z)$ the density of z under the standard Gaussian, by the change of variables formula:

$$P_g(x) = \phi(g^{-1}(x)) \left| \det\left(J_x\left(g^{-1}(x)\right)\right| \right.$$

Hence, we can write down the likelihood in terms of the parameters of g^{-1} under this model!

Choosing invertible transforms

Try 3: *NICE (Non-linear Independent Component Estimation)*

$$z_{1:\frac{d}{2}} = x_{1:\frac{d}{2}}$$

$$z_{\frac{d}{2}+1,d} = x_{\frac{d}{2}+1,d} \odot \exp\left(s_\theta\left(x_{1:d/2}\right)\right) + t_\theta\left(x_{1:d/2}\right)$$

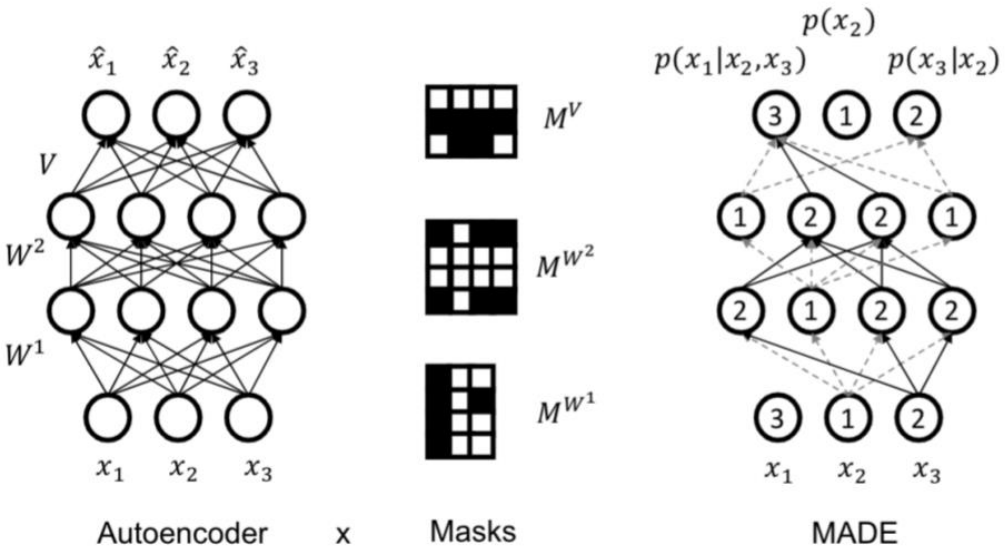
$$J_x\left(f_\theta(x)\right) = \begin{bmatrix} I & 0 \\ \frac{\partial \mathbf{z}_{d/2:d}}{\partial \mathbf{x}_{1:d/2}} & \operatorname{diag}\left(\exp(\boldsymbol{s_\theta}\left(\boldsymbol{x_{1:d/2}}\right)\right) \end{bmatrix}$$

The determinant of a triangular matrix is the product of the diagonals!

$$\text{Hence, } \det\ J_x\left(f_\theta(x)\right) = \Pi_i \exp\left(s_\theta\left(x_{1:d/2}\right)\right)_i$$

If t_θ, s_θ is say, a neural net, easy to evaluate and take derivatives.

MADE: Masked Autoencoder for Distribution Estimation



To zero out connections: mask via entrywise multiplication, e.g. for one layer:

$$\begin{aligned} \mathbf{h}(\mathbf{x}) &= \mathbf{g}(\mathbf{b} + (\mathbf{W} \odot \mathbf{M}^{\mathbf{W}})\mathbf{x}) \\ \hat{\mathbf{x}} &= \text{sigm}(\mathbf{c} + (\mathbf{V} \odot \mathbf{M}^{\mathbf{V}})\mathbf{h}(\mathbf{x})) \end{aligned}$$

To construct appropriate masks: for each node k, pick index m(k) uniformly at random in [1, D-1]: denoting which inputs node depends on.

$$M^{\mathbf{W}}_{k,d} = 1_{m(k) \geq d} = \begin{cases} 1 & \text{if } m(k) \geq d \\ 0 & \text{otherwise,} \end{cases} \qquad M^{\mathbf{V}}_{d,k} = 1_{d > m(k)} = \begin{cases} 1 & \text{if } d > m(k) \\ 0 & \text{otherwise,} \end{cases}$$

PixelCNN

Convolutional architecture, suitable for more complex image domains.

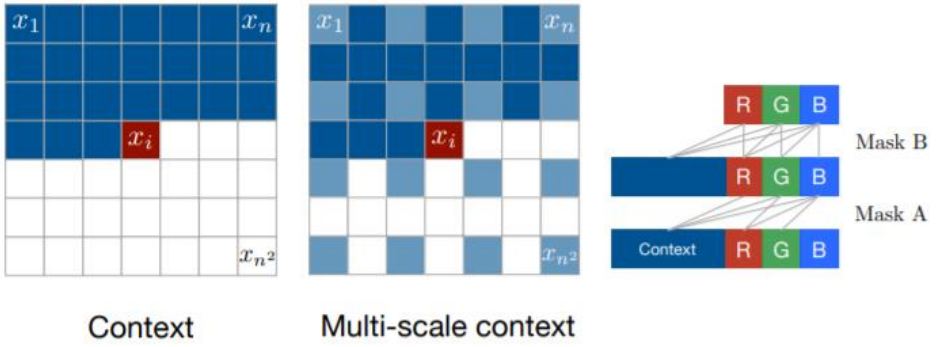


Figure 2. **Left**: To generate pixel x_i one conditions on all the previously generated pixels left and above of x_i. **Center**: To generate a pixel in the multi-scale case we can also condition on the subsampled image pixels (in light blue). **Right**: Diagram of the connectivity inside a masked convolution. In the first layer, each of the RGB channels is connected to previous channels and to the context, but is not connected to itself. In subsequent layers, the channels are also connected to themselves.

Figure from Pixel Recurrent Neural Networks, van den Oord '16

Obvious generalization – only implementation detail: channels are also generated "auto-regressively". $p(x_i|x_{<i})$ is factorized as (Mask A)

$$p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G})$$

In upper layers, we use Mask B, in which value of channel can depend on value of same channel below. (This does the correct thing: e.g. G in layer two only depends on R in the input; if we used mask A, G would not depend on current input at all.)

Recurrent neural networks

The problem with the previous approaches: number of parameters depend on total length.

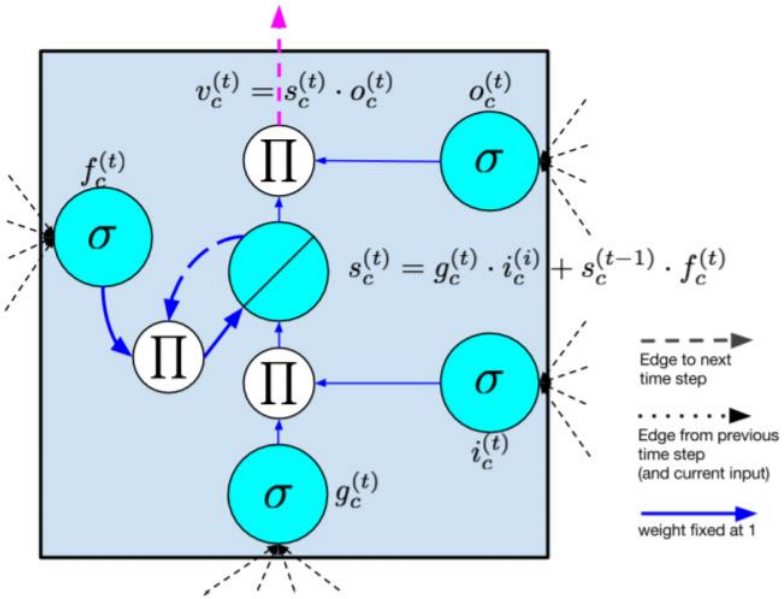
Recurrent neural networks are a way to *weight-tie* the parameters of an autoregressive model, s.t. it can be extended to arbitrary length sequences.

$$h_i = \tanh(W_{hh} h_{i-1} + W_{xh} x_i)$$

$$o_i = W_{hy} h_i$$

o_i specifies parameters for $p(x_i | x_{<i})$, e.g. $\text{softmax}(y_i)$

LSTM (Long Short-Term Memory)



Ingredients:

Input node: $\boldsymbol{g}^{(t)} = \phi(W^{\mathrm{gx}}\boldsymbol{x}^{(t)} + W^{\mathrm{gh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_g)$

Input gate: $\boldsymbol{i}^{(t)} = \sigma(W^{\mathrm{ix}}\boldsymbol{x}^{(t)} + W^{\mathrm{ih}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i)$

Input node will be "gated" by pointwise multiplication with input gate: how input "flows"

Forget gate: $\boldsymbol{f}^{(t)} = \sigma(W^{\mathrm{fx}}\boldsymbol{x}^{(t)} + W^{\mathrm{fh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_f)$

Will gate the "internal state"

Output gate: $\boldsymbol{o}^{(t)} = \sigma(W^{\mathrm{ox}}\boldsymbol{x}^{(t)} + W^{\mathrm{oh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_o)$

How output "flows"

Internal state: $\boldsymbol{s}^{(t)} = \boldsymbol{g}^{(t)} \odot \boldsymbol{i}^{(i)} + \boldsymbol{s}^{(t-1)} \odot \boldsymbol{f}^{(t)}$

Combine gated version of prev. state w/ forget gate, along with gated input node.

Hidden state: $\boldsymbol{h}^{(t)} = \phi(\boldsymbol{s}^{(t)}) \odot \boldsymbol{o}^{(t)}.$

Self-supervised/predictive learning

Given unlabeled data, design supervised tasks that induce a good representation for downstream tasks.

No good mathematical formalization, but the intuition is to "force" the predictor used in the task to learn something "semantically meaningful" about the data.

Self-supervised/predictive learning

- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the future from the recent past.
- Predict the past from the present.
- Predict the top from the bottom.
- Predict the occluded from the visible
- Pretend there is a part of the input you don't know and predict that.

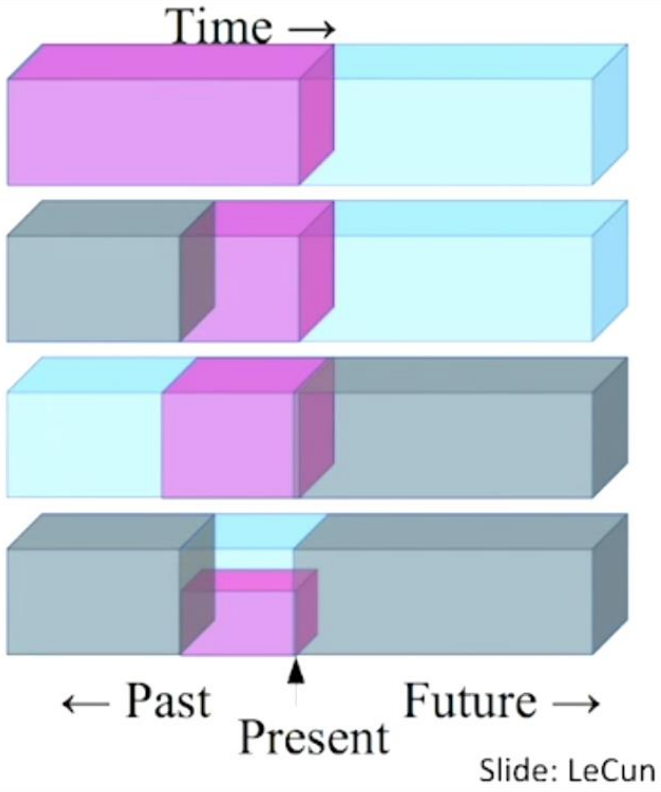
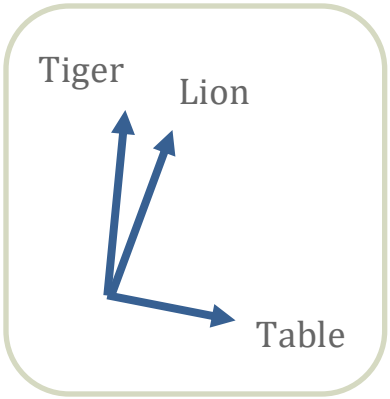


Figure by Yann LeCun

Word embeddings

Semantically meaningful **vector representations** of words



Example: Inner product (possibly scaled, i.e. cosine similarity) correlates with word similarity.

Word embeddings via predictive learning

Basic task: predict the next word, given a few previous ones.

In other words, optimize for

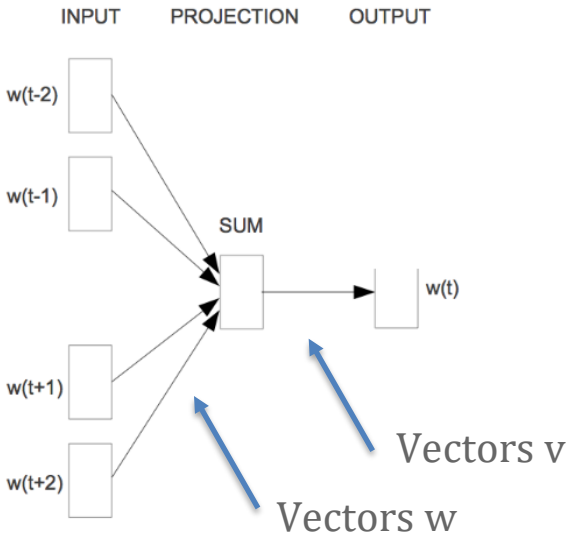
$$\max_{\theta} \sum_{t} \log \; p_\theta(x_t | x_{t-1}, x_{t-2}, \ldots, x_{t-L})$$

Word embeddings via predictive learning

Related: predict *middle* word in a sentence, given *surrounding* ones

$$\max_{\theta} \sum_{t} \log p_\theta(x_t | x_{t-L}, \dots, x_{t-1}, x_{t+1}, \dots, x_{t+L})$$

CBOW (Continuous Bag of Words): proposed by *Mikolov et al.* '13



Parametrization is chosen s.t.

$$p_\theta(x_t | x_{t-L}, \dots, x_{t-1}, x_{t+1}, \dots, x_{t+L}) \propto$$

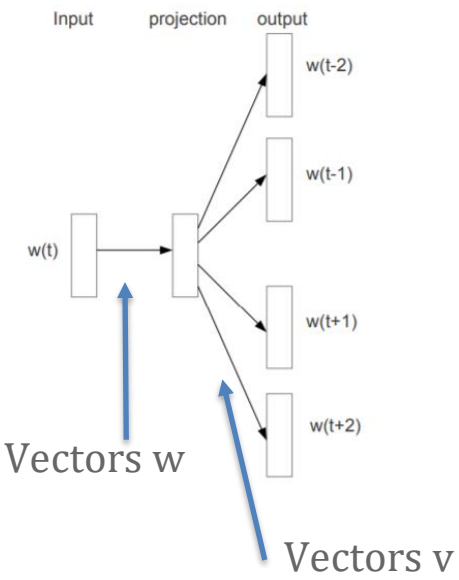
$$\exp\left(v_{x_t}, \sum_{i=t-L}^{t+L} w_{t_i} \right)$$

Word embeddings via predictive learning

Related: predict surrounding words, *given* middle word

$$\max_{\theta} \sum_{t} \sum_{i=t-L, i \neq t}^{t+L} \log \; p_\theta(x_i | x_t)$$

Skip-Gram: (also) proposed by *Mikolov et al.* '13



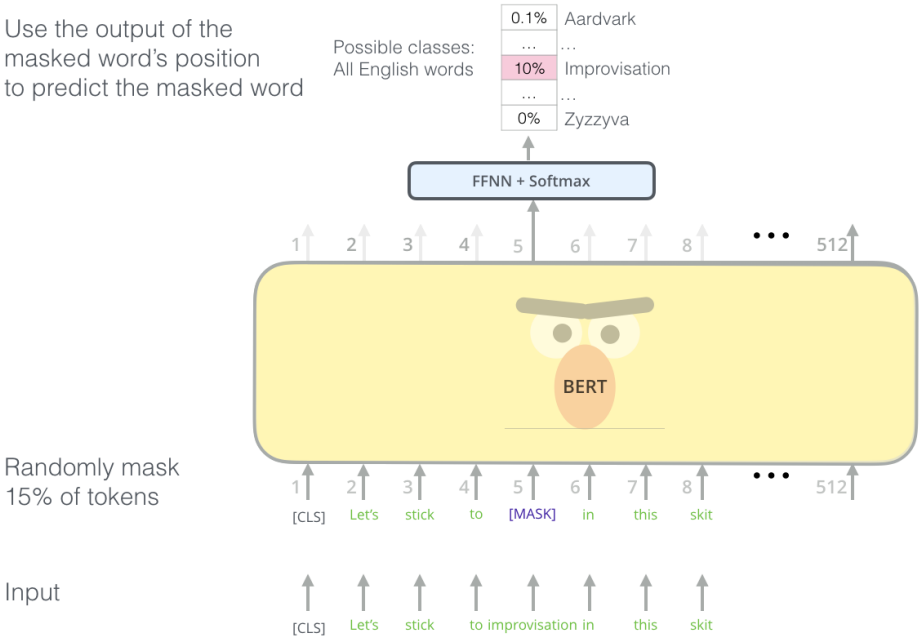
Parametrization is s.t. $p_\theta(x_i | x_t) \propto \exp\left(v_{x_i}, w_{x_t}\right)$

In practice, lots of other tricks are tacked on to deal with the slowest part of training: the softmax distribution (partition function sums over entire vocabulary).
Common ones are *negative sampling, hierarchical softmax*, etc.

Word embeddings via predictive learning

Related: predict random *15%* of the words, given the rest

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. '18.



Pretty much all-across-the-board best-performing representations for most downstream tasks. (Pre fine-tuning, of course.)

Semantic similarity

Observation: similar words tend to have larger (renormalized) inner products (also called cosine similarity).

Precisely, if we look at the word embeddings for words i,j

$\left\langle \frac{w_i}{||w_i||}, \frac{w_j}{||w_j||} \right\rangle = \cos(w_i, w_j)$ tends to be larger for similar words i,j

Example: the nearest neighbors to "Frog" look like

0. *frog*
1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

3. litoria

4. leptodactylidae

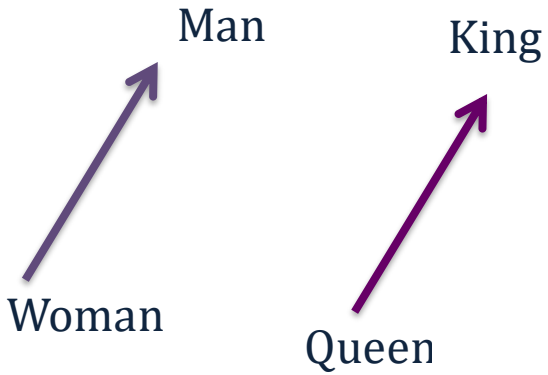
5. rana

7. eleutherodactylus

To solve semantic similarity query like "which is the most similar word to", output the word with the highest cosine similarity.

Analogies

Observation: You can solve *analogy* queries by linear algebra.



Precisely, w = queen will be the solution to:

$$\text{argmin}_w \|v_w - v_{\text{king}} - (v_{\text{woman}} - v_{\text{man}})\|^2$$

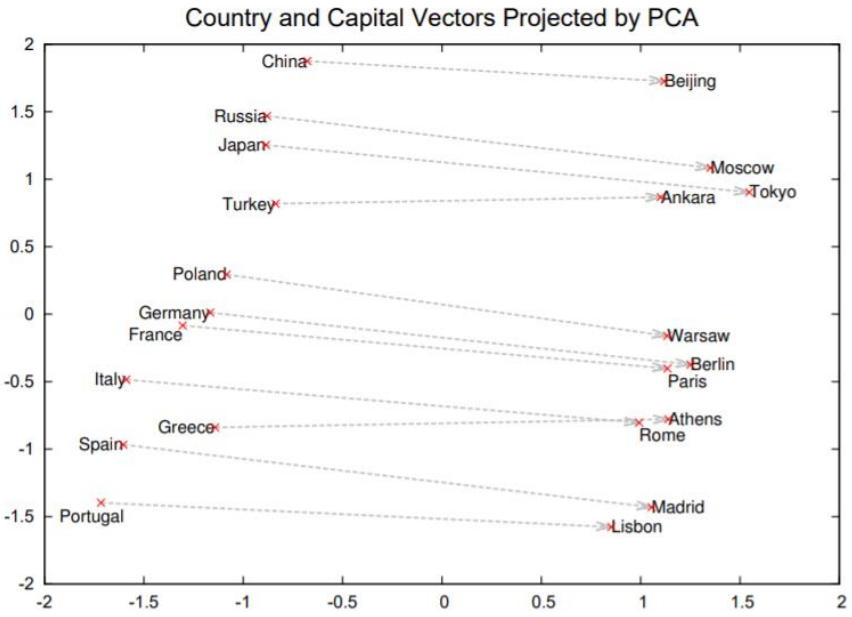


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Inpainting

The most obvious analogy to word embeddings: predict parts of image from remainder of image.

Pathak et al. '16: Context Encoders: Feature Learning by Inpainting

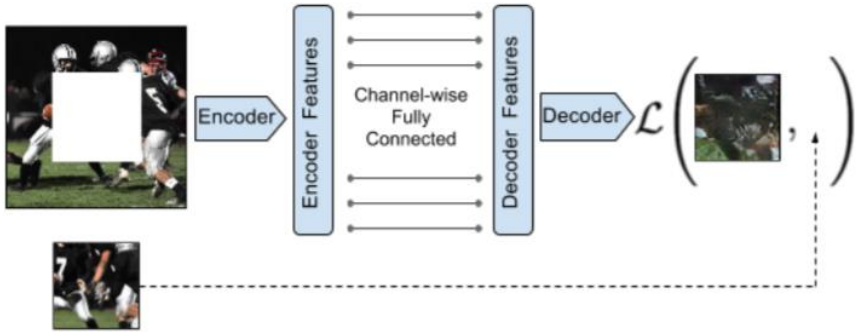


Figure 2: Context Encoder. The context image is passed through the encoder to obtain features which are connected to the decoder using channel-wise fully-connected layer as described in Section 3.1. The decoder then produces the missing regions in the image.

Architecture:
An encoder E takes a part of image, constructs a representation.

A decoder D takes representation, tries to reconstruct missing part.

Much trickier than in NLP:

As we have seen, meaningful losses for vision are much more difficult to design.

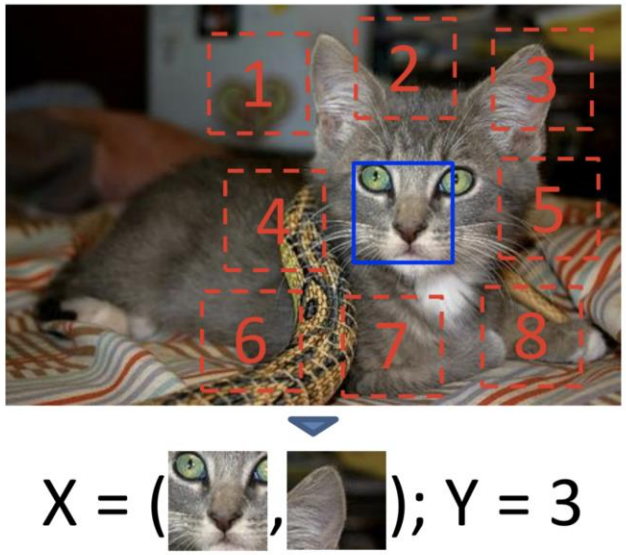
Choice of region to mask out is much more impactful.

Jigsaw puzzles

In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

Doersch et al. '16: Unsupervised Visual Representation Learning by Context Prediction

Task: Predict ordering of two randomly chosen pieces from the image.



Representation: penultimate layer of a neural net used to solve task.

Intuition: understanding relative positioning of pieces of an image requires some understanding of how images are composed.

Predicting rotations

In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

Gidaris et al. '18: Unsupervised representation learning via predicting image rotations

Task: predict one of 4 possible rotations of an image.

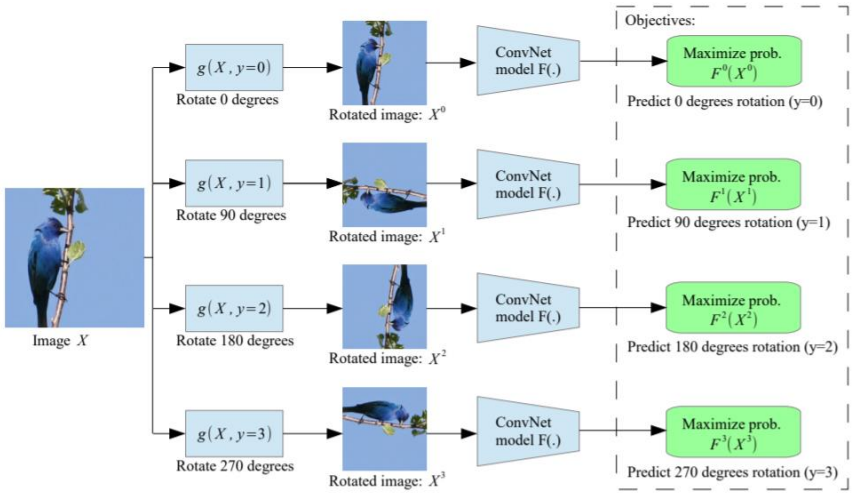


Figure 2: Illustration of the self-supervised task that we propose for semantic feature learning. Given four possible geometric transformations, the 0, 90, 180, and 270 degrees rotations, we train a ConvNet model $F(.)$ to recognize the rotation that is applied to the image that it gets as input. $F^y(X^{y^*})$ is the probability of rotation transformation y predicted by model $F(.)$ when it gets as input an image that has been transformed by the rotation transformation y^*.

Representation: penultimate layer of a neural net used to solve task.

Intuition: a rotation is a global transformation. ConvNets are much better at capturing local transformations (as convolutions are local), so there is no obvious way to "cheat".

Contrastive divergence

Another natural idea: if features are "semantically" relevant, a "distortion" of an image should produce similar features. Some instances of distortions:

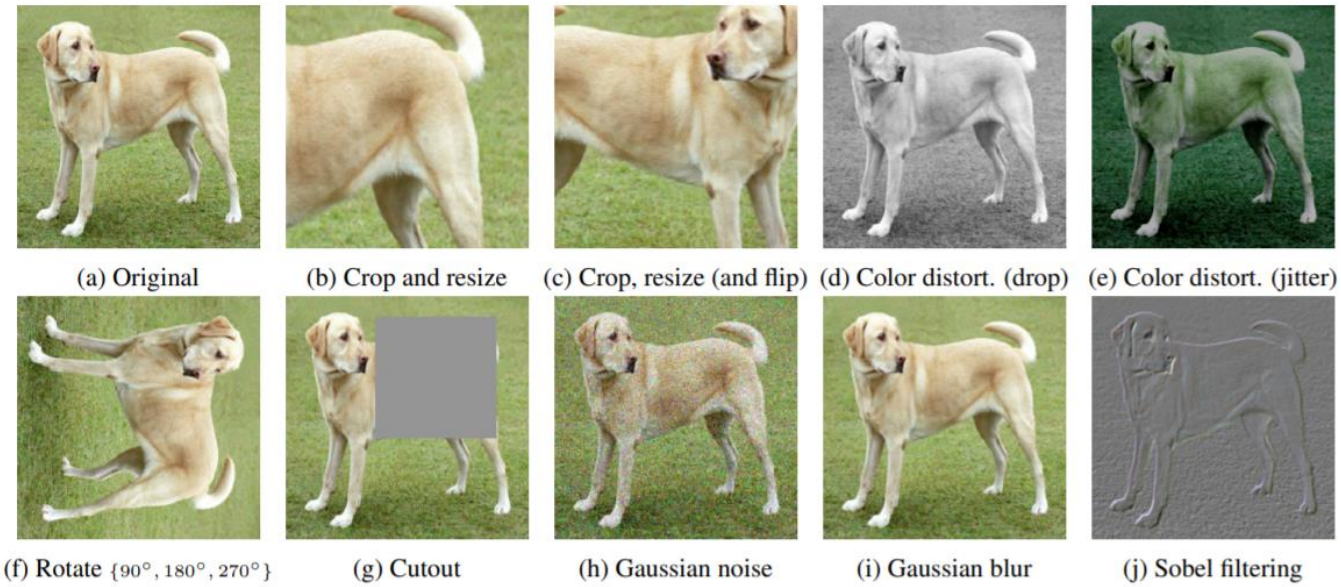
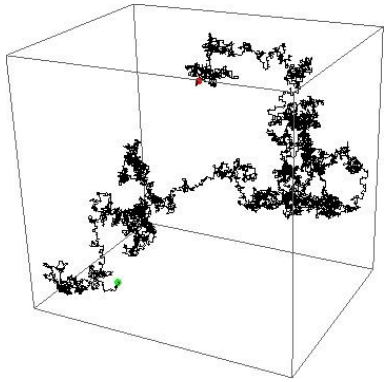


Figure 4. Illustrations of the studied data augmentation operators. Each augmentation can transform data stochastically with some internal parameters (e.g. rotation degree, noise level). Note that we *only* test these operators in ablation, the *augmentation policy used to train our models* only includes *random crop (with flip and resize)*, *color distortion*, and *Gaussian blur*. (Original image cc-by: Von.grzanka)

A generative model for language with semantics [Arora, Li, Liang, Ma, Risteski'15]

Corpus is generated sequentially, t-th word produced at time t.



Process driven by **discourse vector** doing a random walk in a d-dim. "discourse space"(d << N = vocab size).

Parameters:

Each word has (time invariant) latent vector v_w

If discourse vector at time t is c_t,

$$\Pr[w \text{ is output}] \propto \exp(v_w \cdot c_t)$$

(Similar to "loglinear topic model" of Mnih-Hinton'07)

Inner products capture PMI

Let P(w,w') be the probability that w, w' appear consecutively.

Thm:

$$\log p(w, w') = \frac{1}{2d}\|v_w + v_{w'}\|^2 - 2\log Z \pm \epsilon$$

$$\log p(w) = \frac{1}{2d}\|v_w\|_2^2 - \log Z \pm \epsilon \qquad \epsilon = o(1)$$

$$\log(\mathrm{PMI}(w, w')) = \frac{1}{2d}\langle v_w, v_{w'}\rangle \pm 3\epsilon$$

(Norm of word vec determines frequency; spatial orientation "meaning")

Hence, low dimensional vectors approximately recover PMI!

Note: $||v_w|| = \theta(\sqrt{d})$ whp, so $\frac{1}{2d}||v_w + v_{w'}||^2 = \Theta(1)$, so error is low order.

Note: The introduction of the scalar s allows word probability to vary.

Word embeddings

The main issue: embeddings are context-independent.

Seems intuitively wrong: determining the meaning of "bank" in

"*Walking along the river bank*" and "*A bank accepts deposits*"

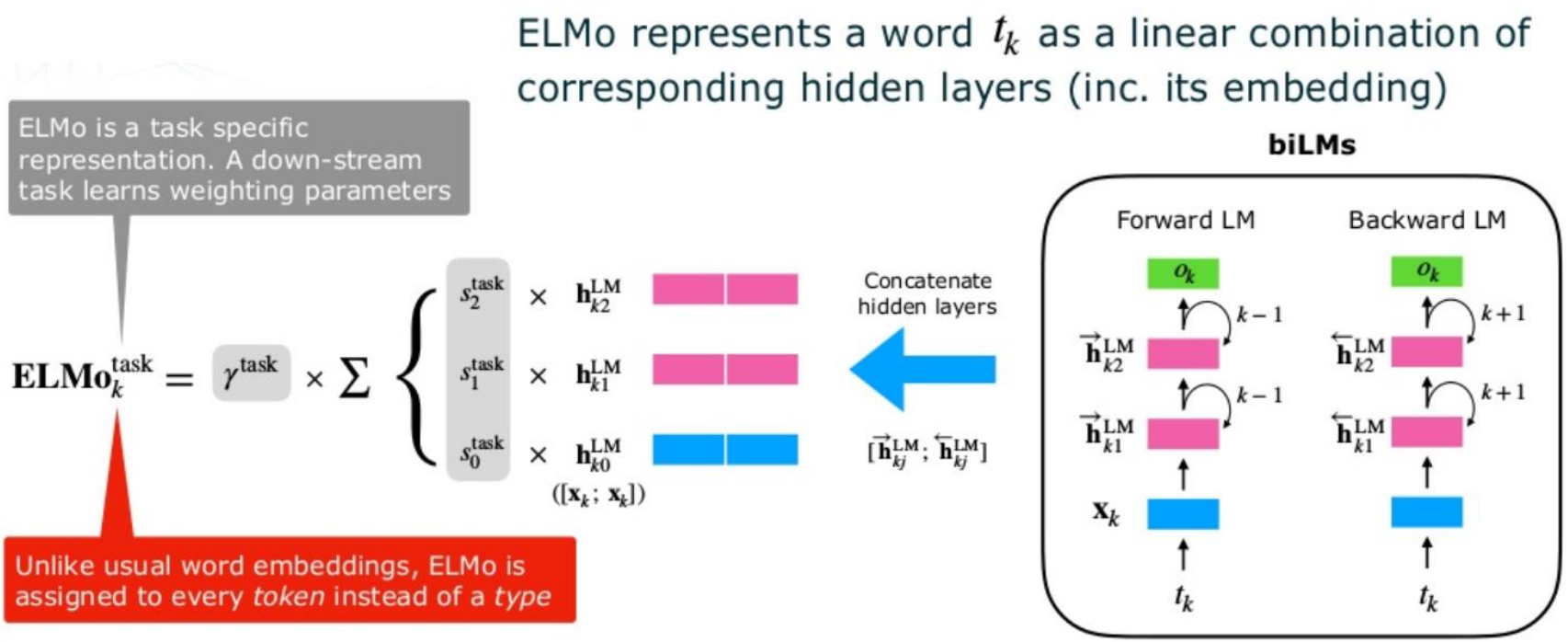
depends on the surrounding context ("river" and "deposits" being the strong indicators of meaning).

One way to handle this? Recurrent neural nets!

The parade of Sesame Street: ELMo

Peters et al '18: **ELMo** (Embeddings from Language Models)

Train one (stacked) LSTM model to model $p_\theta(x_t|x_{<t})$ and another LSTM to model $p_\theta(x_t|x_{>t})$. Concatenate two representations. (For downstream tasks, linearly combine tasks with learned weights.)



Slide from Matthew Peters

Attention

A more "differentiable" variant of this:

$$\text{Attention}(q, K, V) = \sum_i \text{similarity}(q, K_i) V_i$$

Vector

Matrices w/ rows keys/values

Linear combination of "most similar" values

The simplest notion of similarity: inner product.

$$\text{Attention}(q, K, V) = \sum_i \text{softmax}(\langle q, K_i \rangle) V_i$$

Produces distribution over keys

Produces distribution over values

Multiple queries combined in matrix Q:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V$$

Transformers: using attention

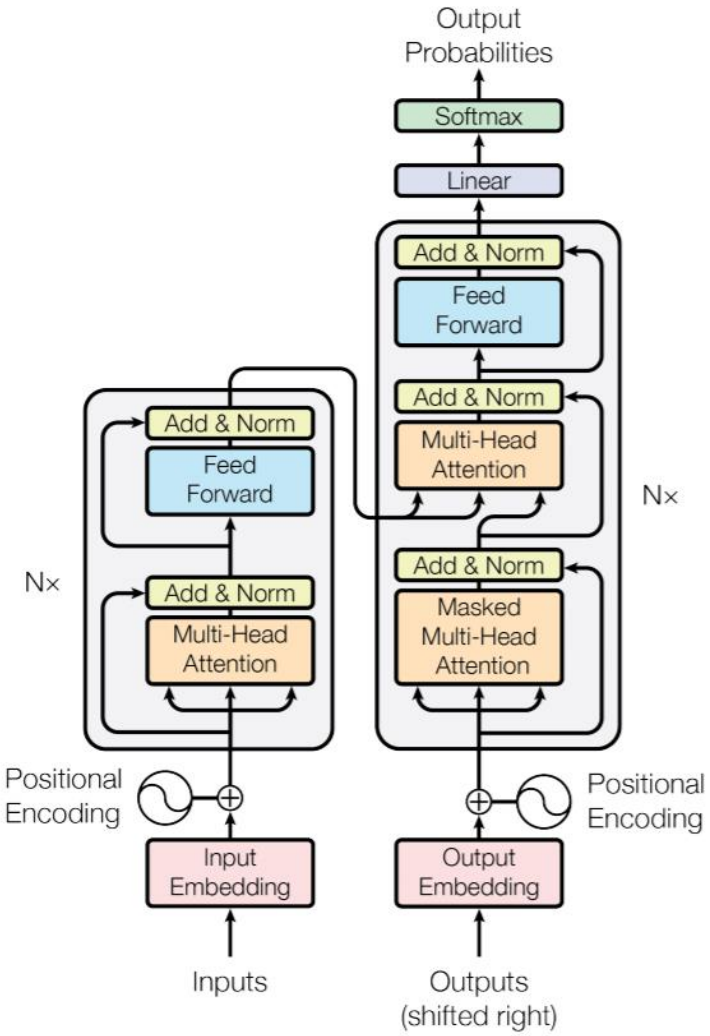
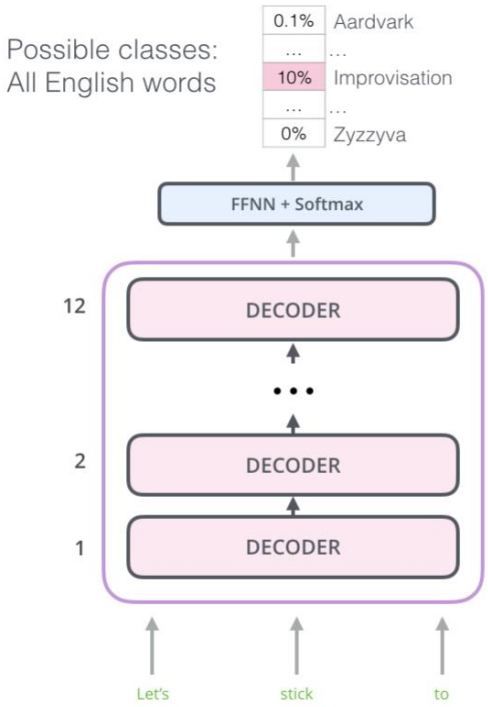


Figure from Vaswani et al ’17: “Attention is all you need”.

Using transformers for feature learning: the obvious way

The obvious way to use this architecture to simply extract features: use a decoder only!



GPT2: Transformer-based architecture due to OpenAI

Figure from http://jalammar.github.io/illustrated-bert/

The less obvious way: BERT

Something is still amiss: the decoder from transformers still predicts the next word. *Still unidirectional*!

The insight of Devlin et al. '18: BERT (Bidirectional Encoder Representations from Transformers) – use an **encoder** instead!

But how? What are trying to predict?

Predict random *15%* of the words, given the rest

Other topics

Adversarial robustness (Elan's lecture)

Deep reinforcement learning (10-703, Elan's lecture)

Graph neural networks (Dealing with graph structured data)

Meta-learning, learning-to-learn (Learning to adapt fast for subsequent tasks.)

Hyperparameter optimization. (Optimization over discrete domains.)

Causality in Deep Learning.

Deep learning for scientific applications.

….