Summary lecture

Machine Learning in der Medizin

Asan Agibetov, PhD

asan.agibetov@meduniwien.ac.at

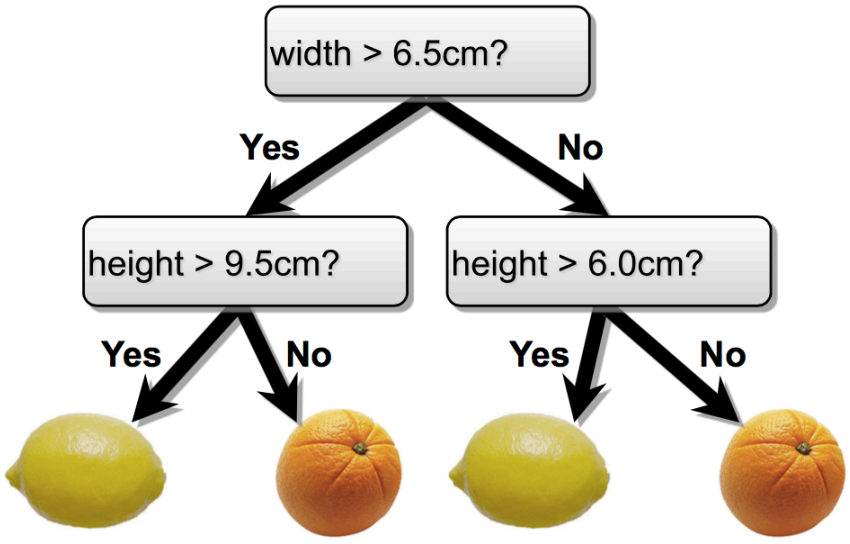
Medical University of Vienna
Center for Medical Statistics, Informatics and Intelligent Systems
Section for Artificial Intelligence and Decision Support
Währinger Strasse 25A, 1090 Vienna, OG1.06

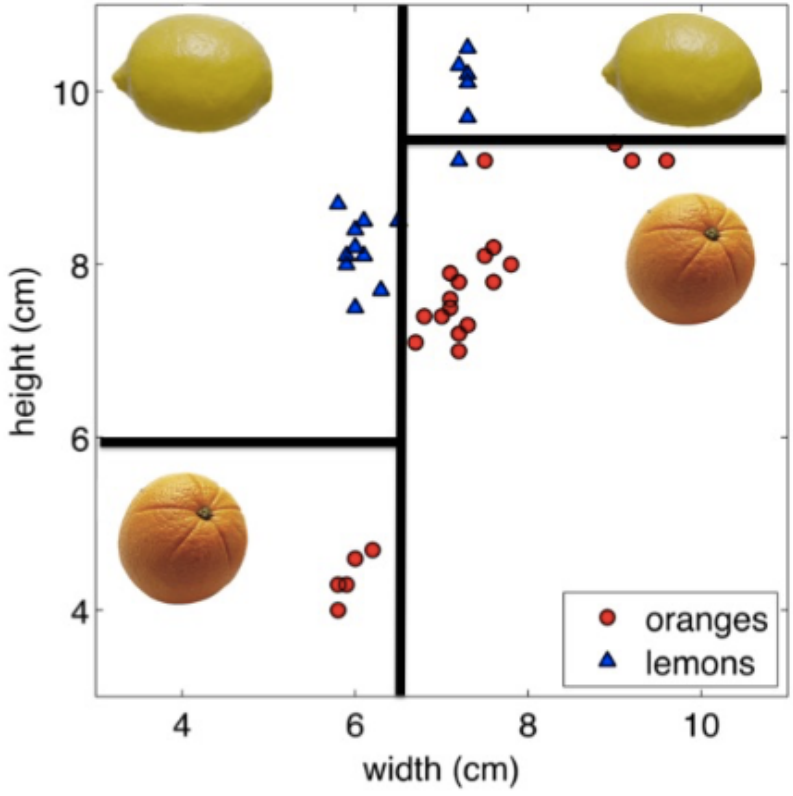
January 16, 2020

Summary

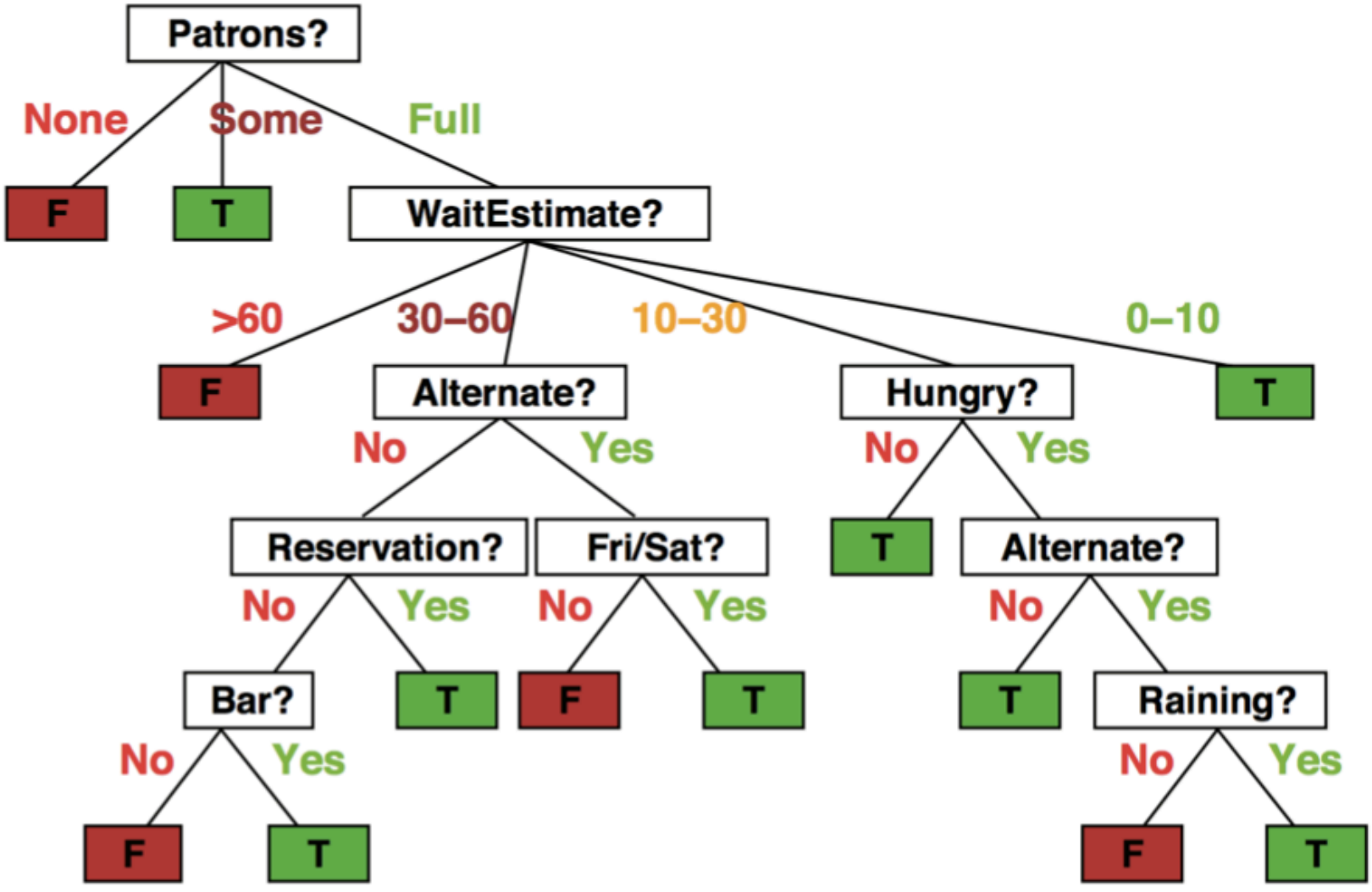
- Decision trees
  - partitioning feature space, entropy
  - medical applications
  - bias and variance trade-off, ensembles
- Deep Learning (DL) theory
  - function approximation, maximum likelihood principle
  - loss functions, cross-entropy minimization, regularization
  - gradient descent and backpropagation
- Main DL techniques
  - convolutional layers, embeddings, encoders (VAE, GANs), graph convolutional layers
- Biomedical applications of DL
- How to do DL
  - Python frameworks and cloud computation
- Possible exam questions
- What is next?

Decision trees





Classification with decision trees



Information theory: Quantifying uncertainty with entropy

```
Sequence 1:
0 0 0 1 0 0 0 0 0 0 0 1 ...
Sequence 2:
1 0 1 0 1 0 1 0 1 1 0 1 ...
```

Information theory: Quantifying uncertainty with entropy

```
Sequence 1:
0 0 0 1 0 0 0 0 0 0 0 1 ...
Sequence 2:
1 0 1 0 1 0 1 0 1 1 0 1 ...
```

$$H(X) = -\sum_{x} p(x) \log_2 p(x)$$

- Entropy of X - how surprised are we by a new value in the sequence?

Information theory: Quantifying uncertainty with entropy

```
Sequence 1:
0 0 0 1 0 0 0 0 0 0 0 1 ...
Sequence 2:
1 0 1 0 1 0 1 0 1 1 0 1 ...
```

$$H(X) = -\sum_{x} p(x) \log_2 p(x)$$

- Entropy of X - how surprised are we by a new value in the sequence?

Seq.	0	1	Entropy $H(X)$
1	10	2	$-\frac{10}{12}\log_2\frac{10}{12} - \frac{2}{12}\log_2\frac{2}{12} = 0.65$
2	5	7	$-\frac{5}{12}\log_2\frac{5}{12} - \frac{7}{12}\log_2\frac{7}{12} = 0.97$

Information gain (IG)

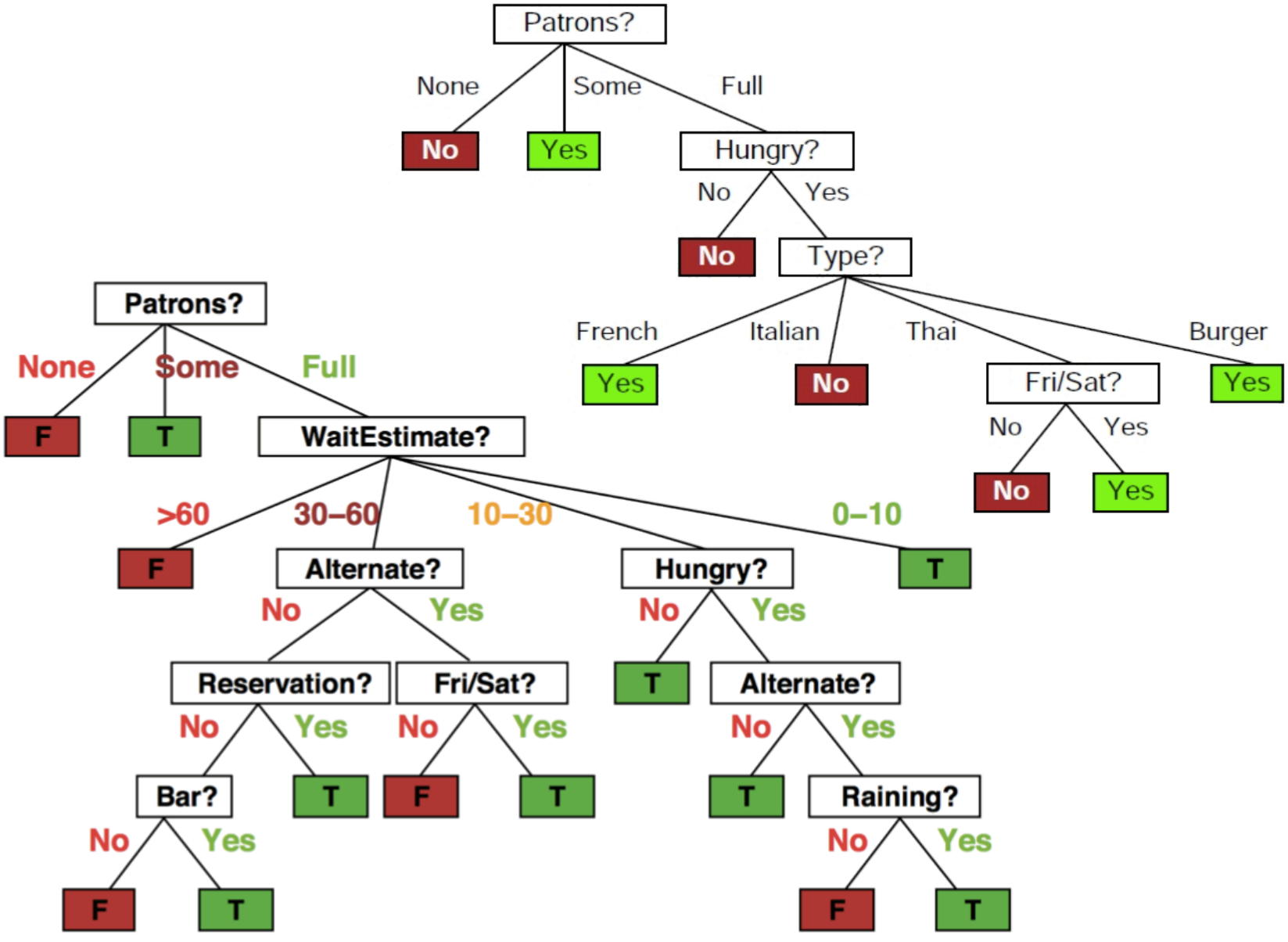
	disease	**not disease**
smoking	24/100	1/100
not smoking	25/100	50/100

- How much information about disease do we get by discovering whether the patient is smoking or not?

$$IG(Y|X) = H(Y) - H(Y|X)$$
$$\approx 0.25$$

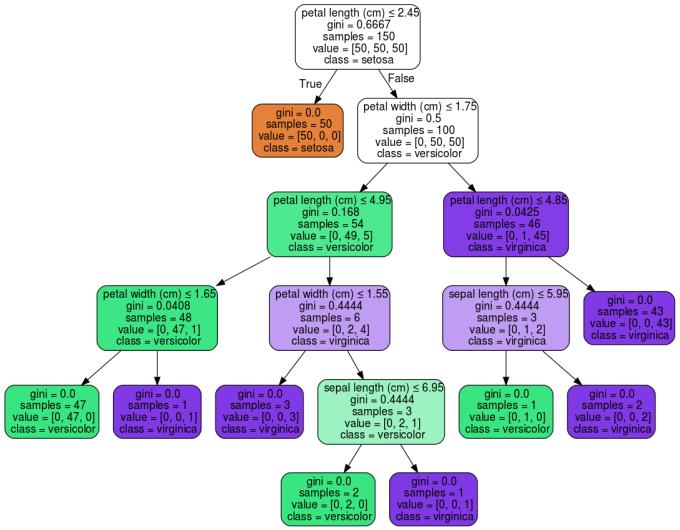
- $IG(Y|X) = 0$
  - if X is completely uninformative about Y
- $IG(Y|X) = H(Y)$
  - if X is completely informative about Y

Model selection

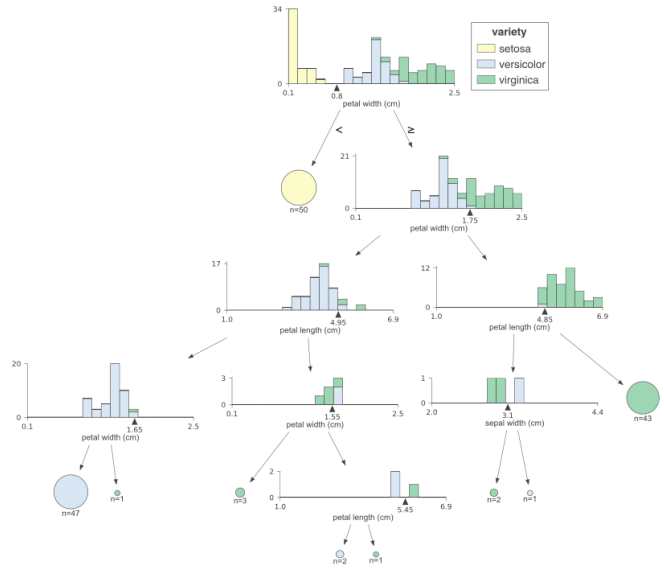


Interpreting decision trees (visualization)

- standard `scikit-learn` visualization



- more informative `dtree` visualization



Medical applications

Predicting cesarean delivery with decision tree models

Cynthia J. Sims, MD,[a] Leslie Meyn, BS,[a] Rich Caruana, PhD,[b, c] R. Bharat Rao, PhD,[d] Tom Mitchell, PhD,[b] and Marijane Krohn, PhD[a]

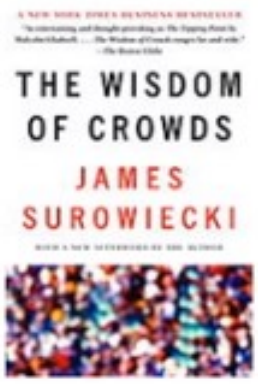
Pittsburgh, Pennsylvania, Los Angeles, California, and Princeton, New Jersey

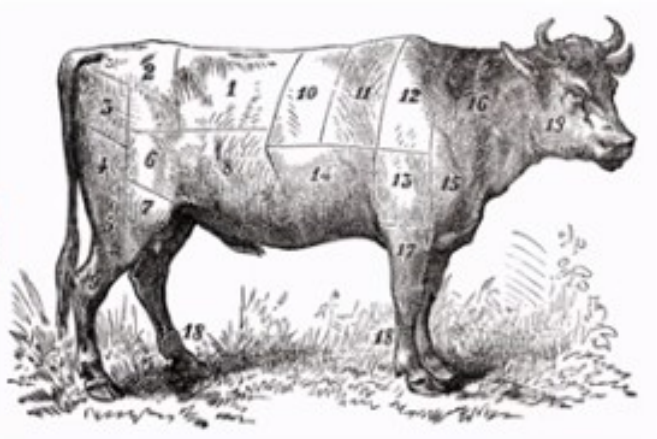
OBJECTIVE: The purpose of this study was to determine whether decision tree–based methods can be used to predict cesarean delivery.
STUDY DESIGN: This was a historical cohort study of women delivered of live-born singleton neonates in 1995 through 1997 (22,157). The frequency of cesarean delivery was 17%; 78 variables were used for analysis. Decision tree rule-based methods and logistic regression models were each applied to the same 50% of the sample to develop the predictive training models and these models were tested on the remaining 50%.
RESULTS: Decision tree receiver operating characteristic curve areas were as follows: nulliparous, 0.82; parous, 0.93. Logistic receiver operating characteristic curve areas were as follows: nulliparous, 0.86; parous, 0.93. Decision tree methods and logistic regression methods used similar predictive variables; however, logistic methods required more variables and yielded less intelligible models. Among the 6 decision tree building methods tested, the strict minimum message length criterion yielded decision trees that were small yet accurate. Risk factor variables were identified in 676 nulliparous cesarean deliveries (69%) and 419 parous cesarean deliveries (47.6%).
CONCLUSION: Decision tree models can be used to predict cesarean delivery. Models built with strict minimum message length decision trees have the following attributes: Their performance is comparable to that of logistic regression; they are small enough to be intelligible to physicians; they reveal causal dependencies among variables not detected by logistic regression; they can handle missing values more easily than can logistic methods; they predict cesarean deliveries that lack a categorized risk factor variable. (Am J Obstet Gynecol 2000;183:1198-206.)

Key words: Decision trees, machine learning, predicting cesarean delivery, statistical models

Ensembles

The Wisdom of Crowds

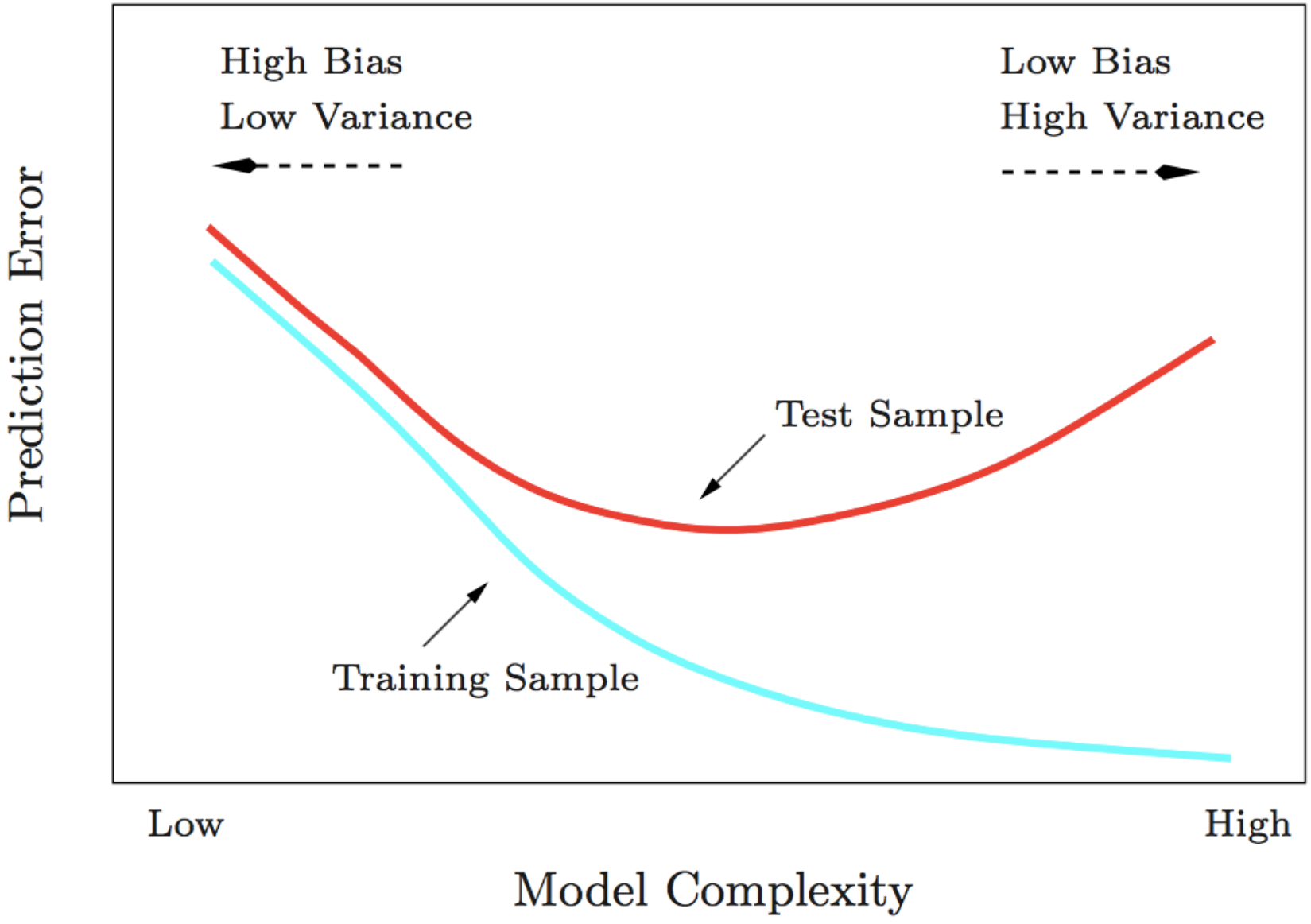




average of 800 guesses = 1,197

actual weight of the ox = 1,198

Bias/Variance tradeoff



How to reduce Variance without increasing Bias

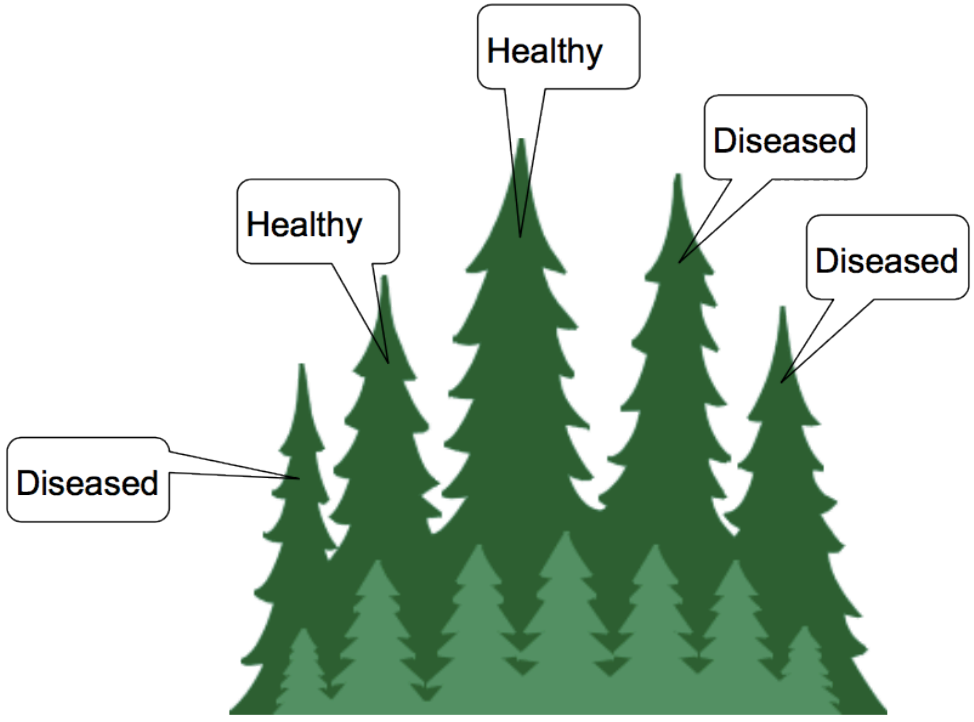
- If predictions are independent
  - *averaging* reduces variance

$$Var(\bar{X}) \approx \frac{Var(X)}{N}$$

- Average models reduce model variance
- However, we have only one training set
  - *How to get multiple models?* **Bagging (bootrstrap aggregation)**

Random Forest (Leo Breiman, 2001)

- Grow a *forest* of many trees (`R` default is 500)
  - each tree on an independent *bootstrap sample*
- At each node
  1. select m variables at *random* out of all M possible variables (independently for each node)
  2. find the best split on the selected m variables
- Grow trees to maximum depth
- Vote/average trees to get predictions

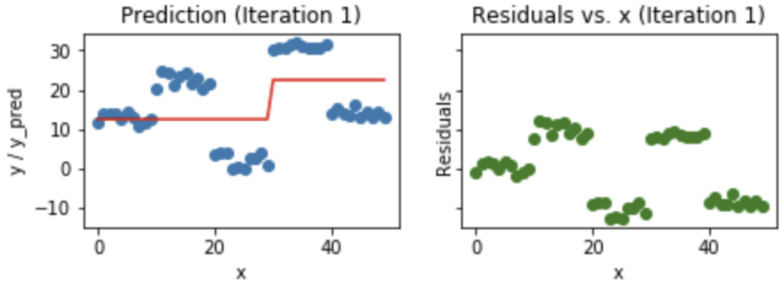


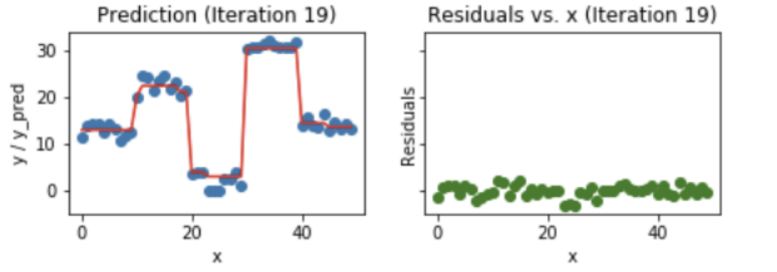
Reduce Bias AND decrease Variance?

- Bagging reduces variance by averaging
  - *however*, it has little effect on bias
- Can we average variance **and** reduce bias? **Boosting**

Gradient Boosting

- Gradient boosting - ensemble method, also known as "additive training"
  - optimization is in "function space" (i.e., parameters are functions, in our case - trees)
  - start from constant prediction, add a new function (tree) each time
  - each subsequent function improves errors of all previous functions





Deep Learning (DL) theory

- function approximation, maximum likelihood principle
- loss functions, cross-entropy minimization, regularization
- gradient descent and backpropagation

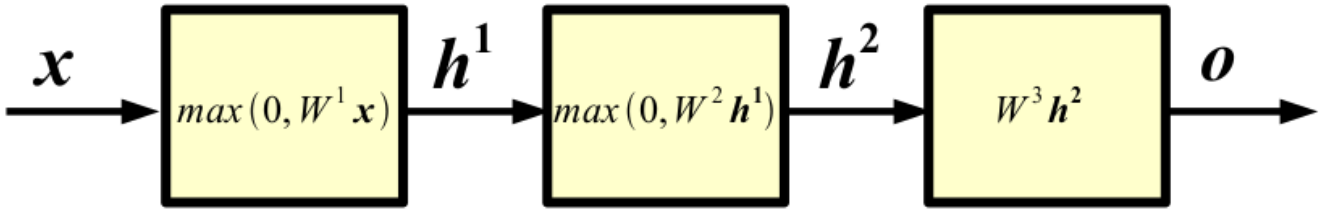
Learning function to model conditional probability (supervized case)

- 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

Neural Networks: Universal function approximation



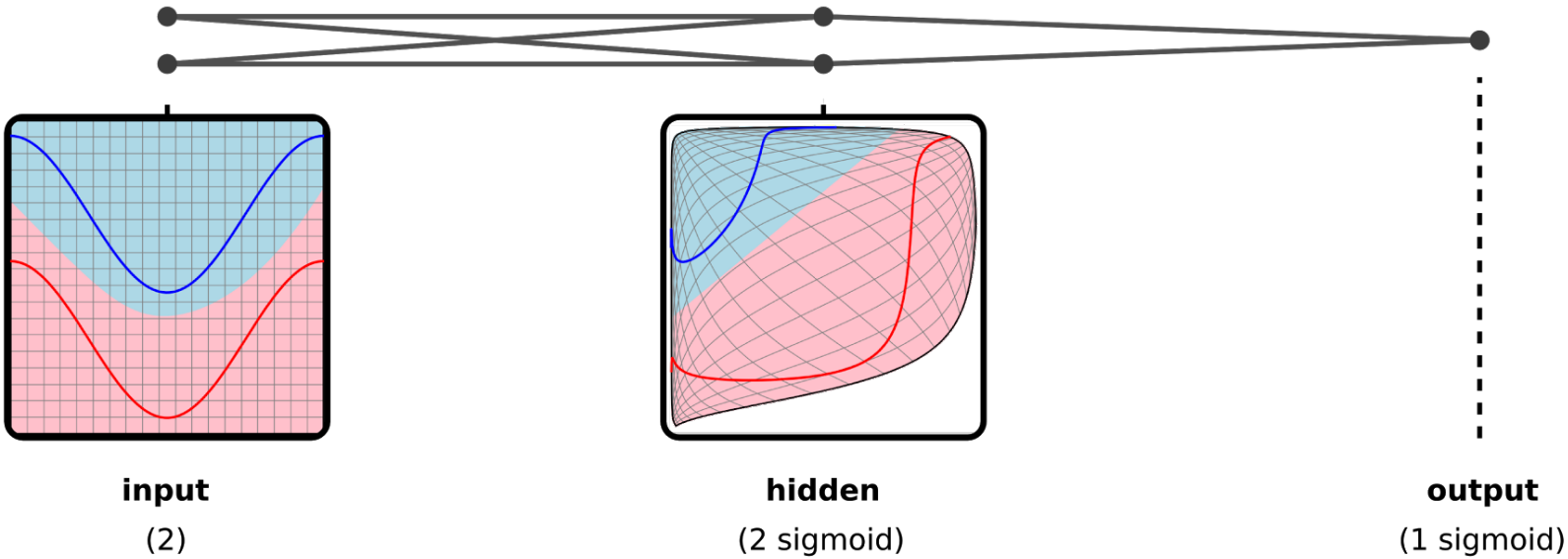
- Universal function approximation that maps input to output
  - $f : \mathbb{R}^n \mapsto \mathbb{R}^m$
- Class of functions considered to map input to output
  - composition of simpler (including non-linear [1]) functions
    - h^1 is non-linear $max(0, \mathbf{W} \cdot \vec{x} + \vec{b})$ aka ReLU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

---

[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 NN 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/

How to learn an optimal function f? 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

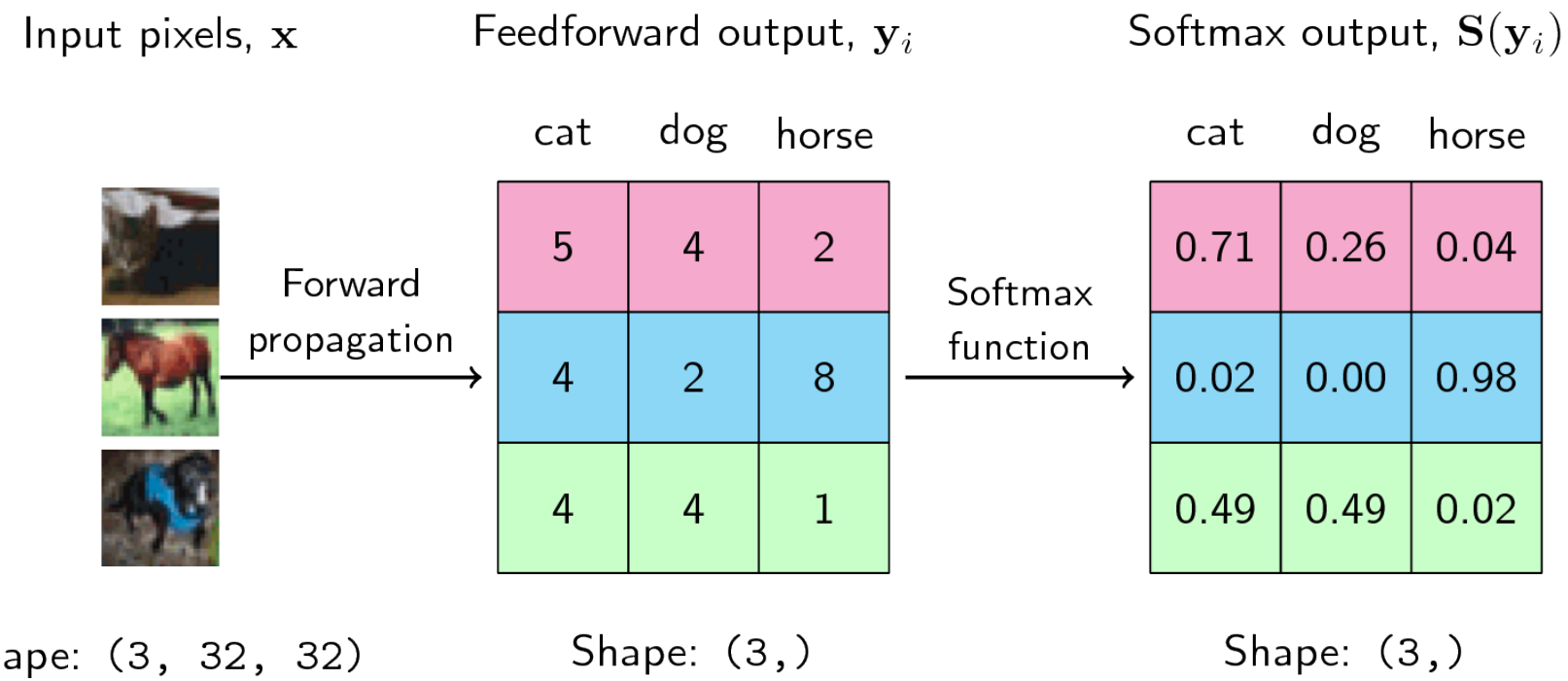
Probabilistic interpretation (supervized case)

- 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

How do I make my network output probabilities?

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

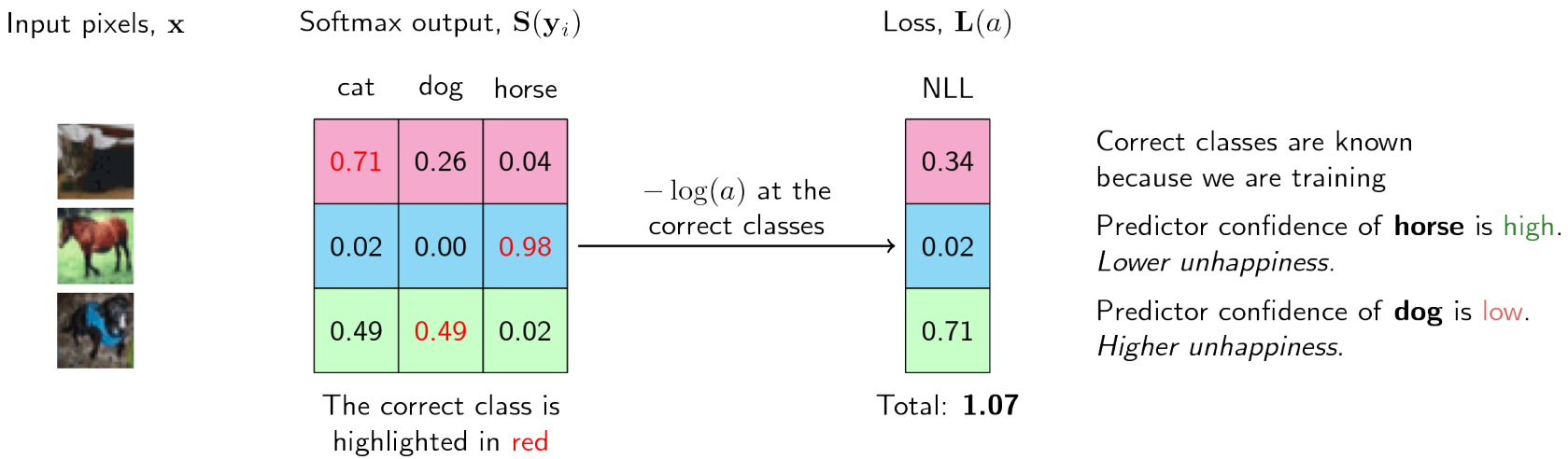
$$\text{softmax}(f_k) = \frac{e^{f_k}}{\sum_j e^{f_j}}.$$



(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



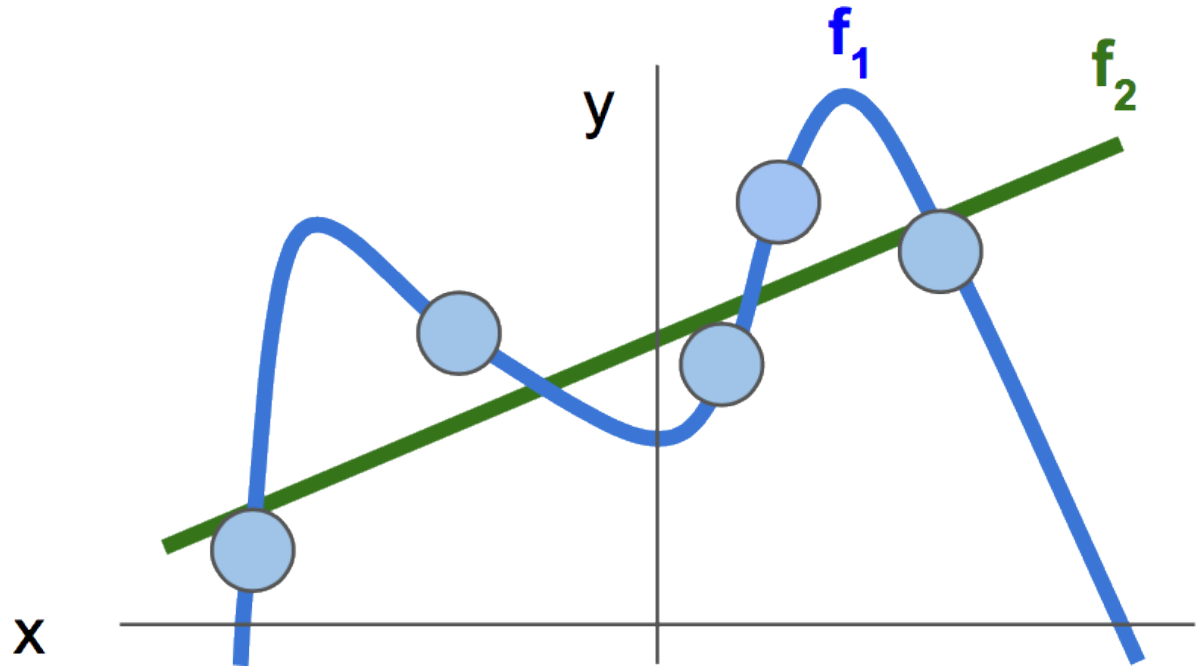
(Image credit) L.Miranda tech blog https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/

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

Why we need regularization?



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

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

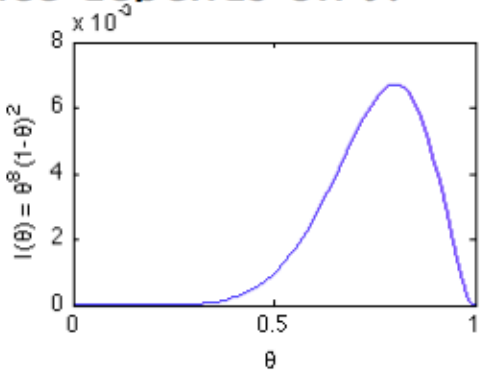
(Image credit) Course notes "CNN for Visual Recognition" (Stanford, Spring 2017)

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

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

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

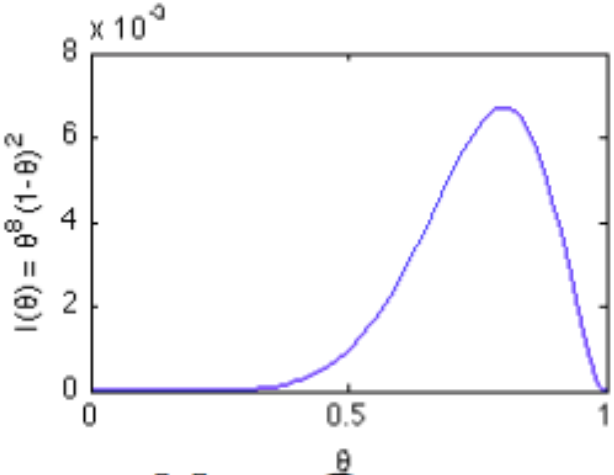
→ extrema at θ = 0, θ = 1, θ = 0.8

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

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

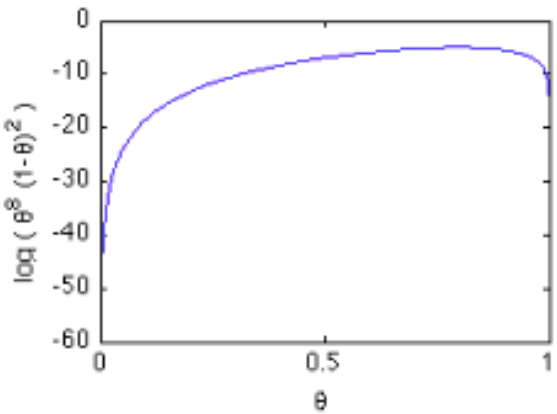
MLE Log transform

- 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
  $D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \log\left(\frac{p(x_i)}{q(x_i)}\right) = 0.$

Recap: 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

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

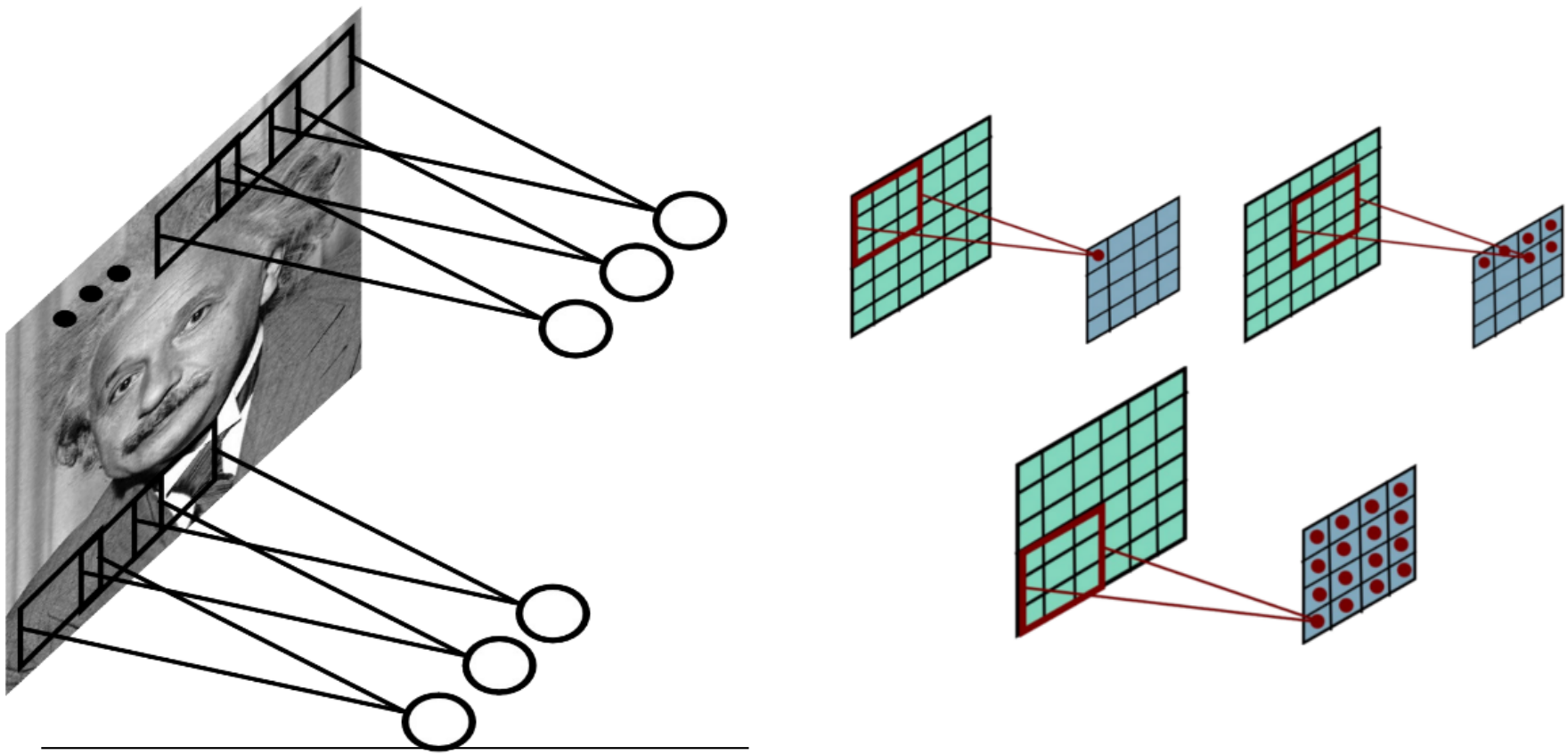
- to find f^* you need to minimize complicated function
- **backpropagation** gives the gradients of that complicated function

(Subset) Main techniques of DL

- Convolutional neural networks
- Embeddings
- Autoencodersi

Convolutional Layer

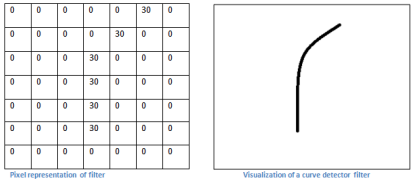
- shared weights across the whole image
- convolution takes advantage of
  - *stationarity* (similar statistics at different locations)
  - local spatial correlation



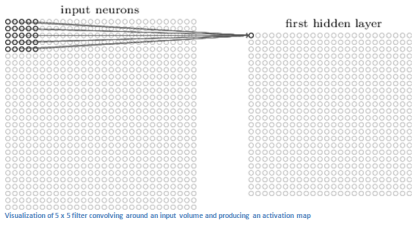
(Image credit) Course notes "An introduction to Deep Learning", Marc'Aurelio Ranzato

Convolutional layer activations

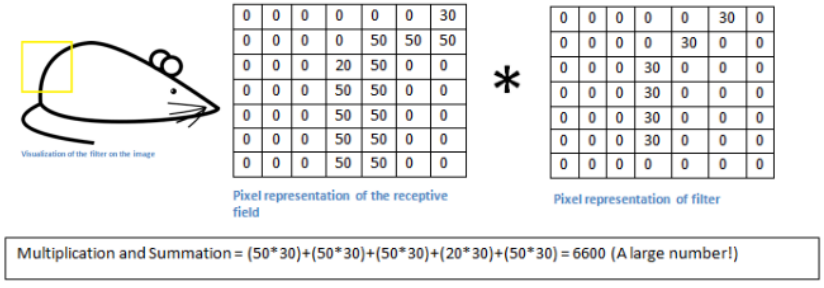
Receptive field (aka. filter, kernel)



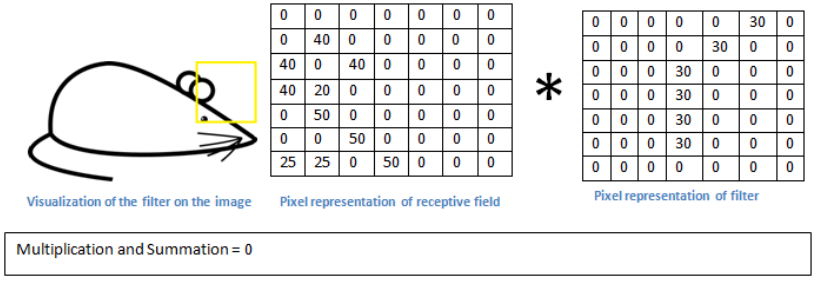
Swipes through input and outputs activation maps



activation maps - results of convolutions (sum of element-wise multiplications)



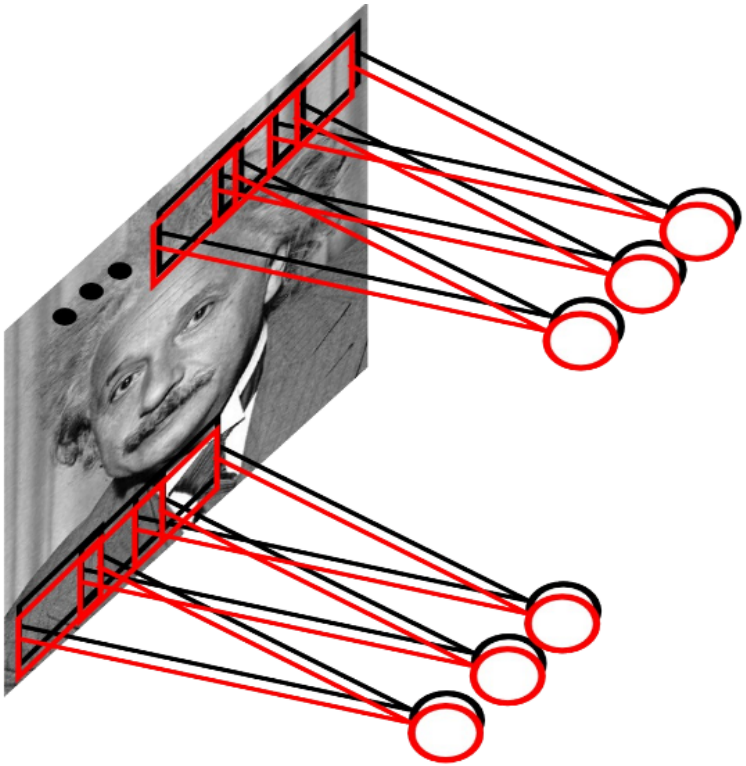
positive activation

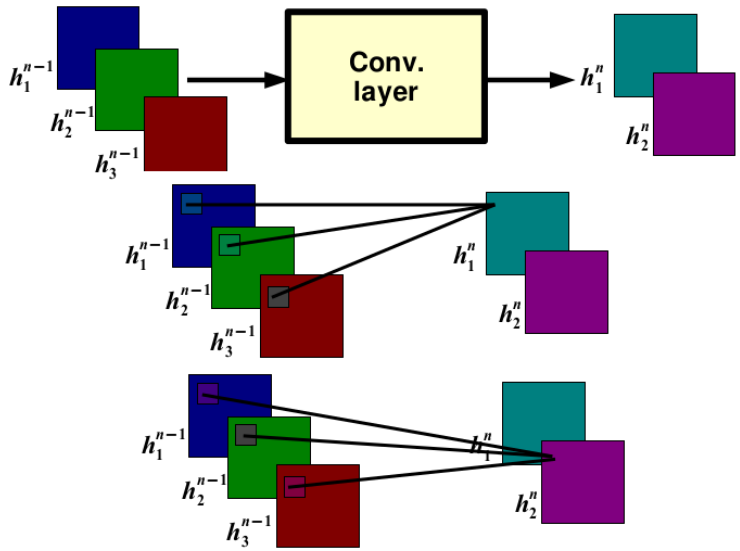


null activation

Multiple convolutional filters

$$h_j^n = max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n)$$

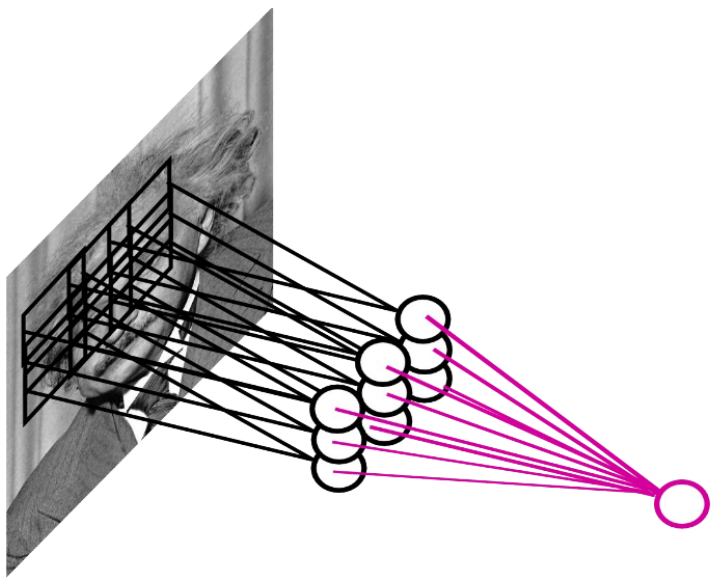




(Image credit) Course notes "An introduction to Deep Learning", Marc'Aurelio Ranzato

Pooling layer

$$h_j^n(x, y) = max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

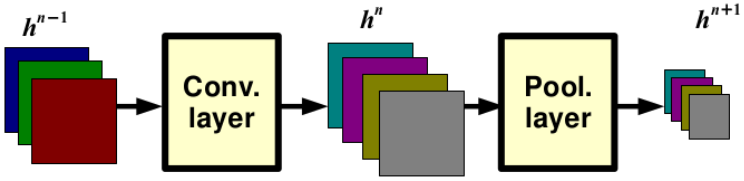


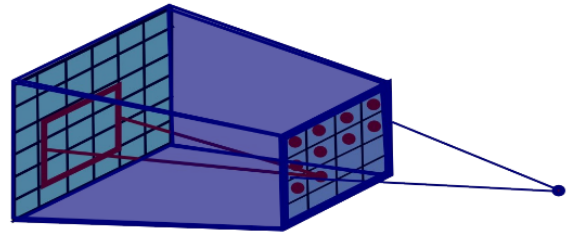
- Pooling layer goal: spatial robustness for feature extraction
- Assume our filter is *eye* dectector
  - Pooling layer makes *eye* detector robust to exact location of eye
- by *pooling* (e.g., taking max) filter responses at different locations
  - we gain robustness to the exact spatial location of features

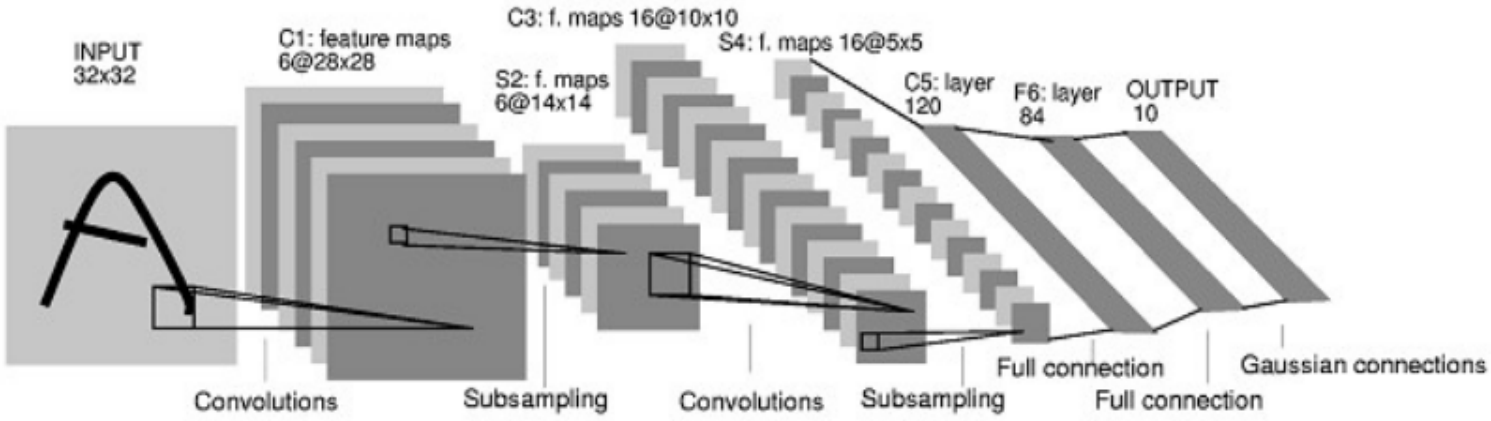
---

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

ConvNets architecture



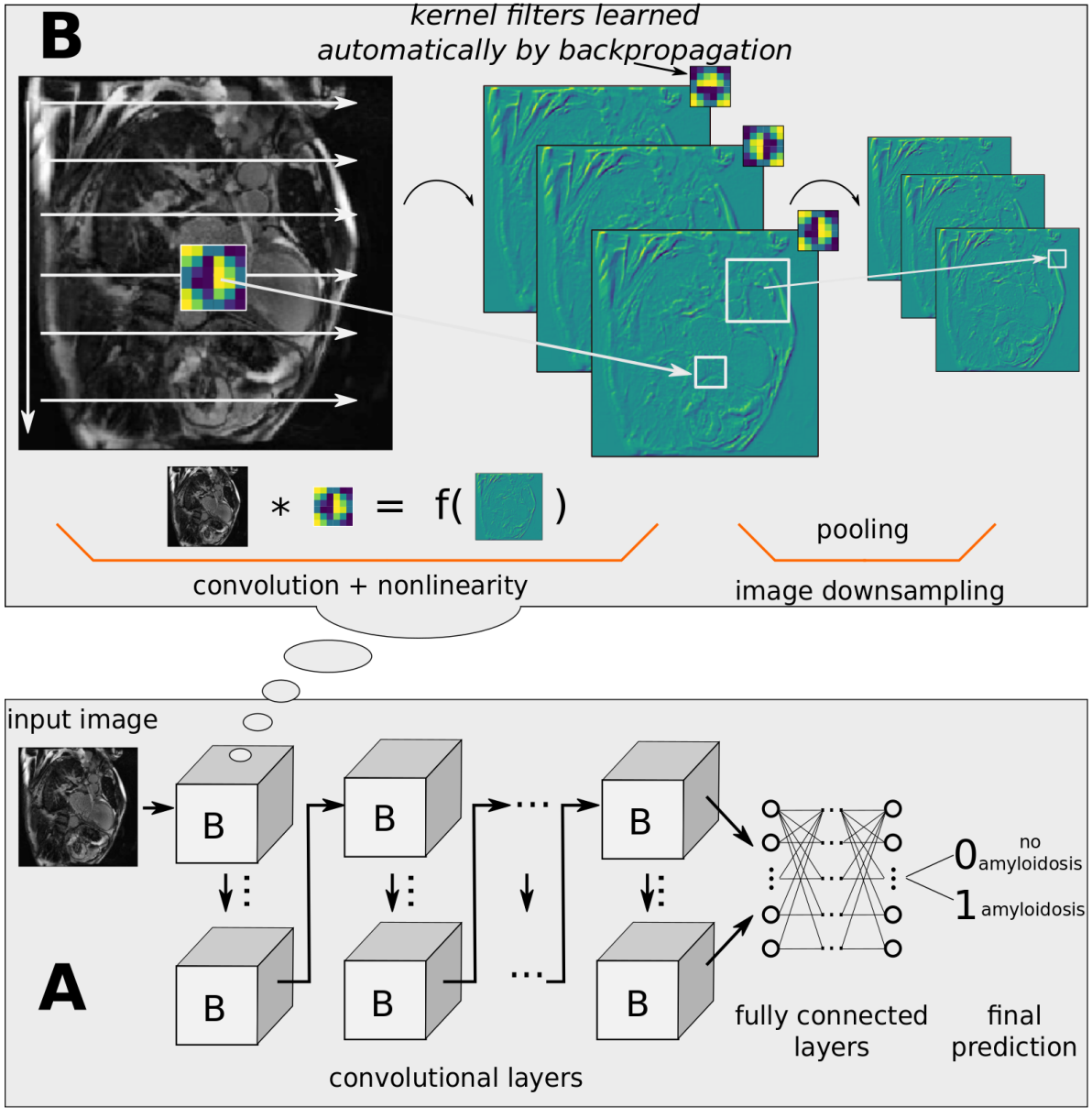




(Image credit (top)) Course notes "An introduction to Deep Learning", Marc'Aurelio Ranzato

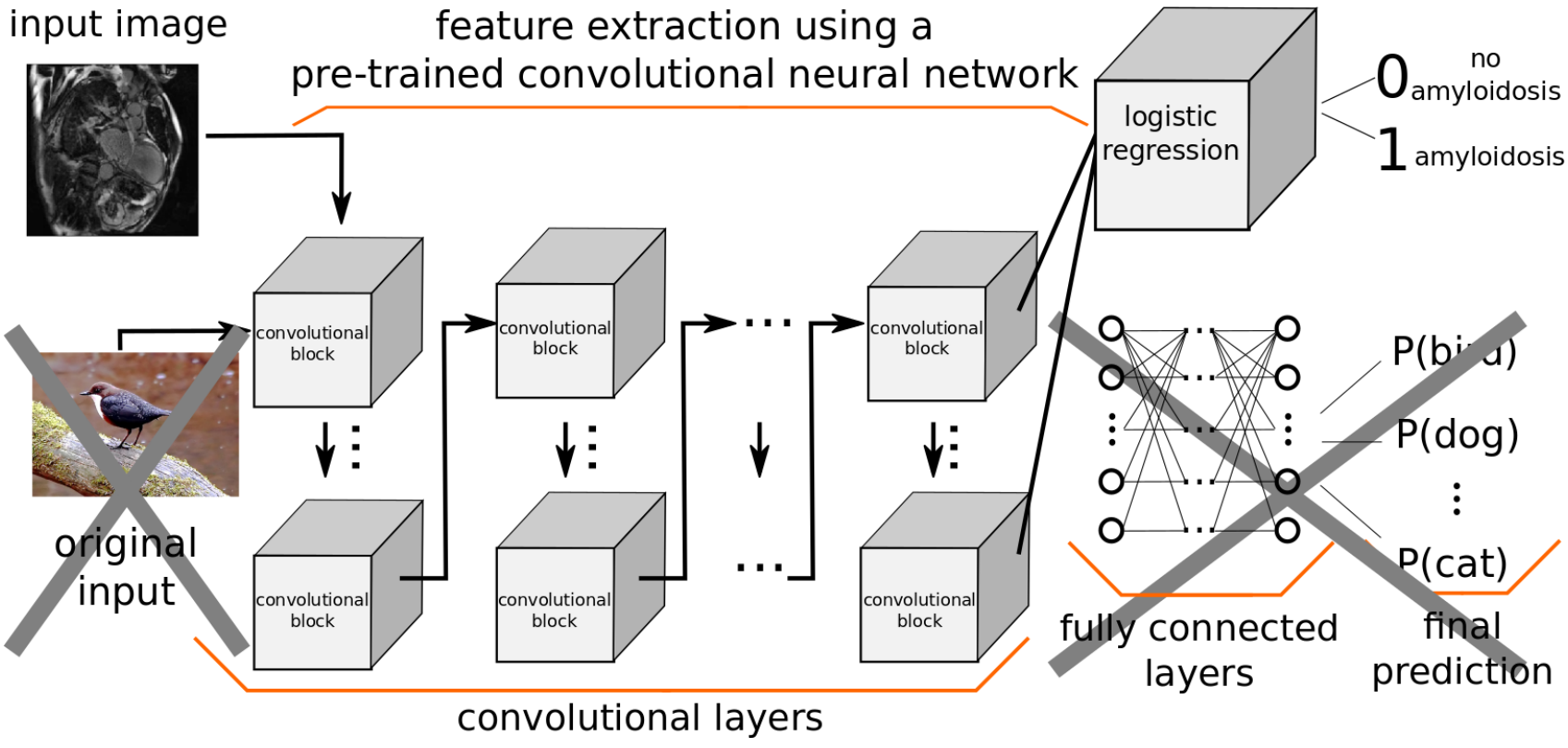
(Image credit (bottom)) LeCun et al. "Gradient based learning applied to document recognition" IEEE 1988

Diagnosis of medical images

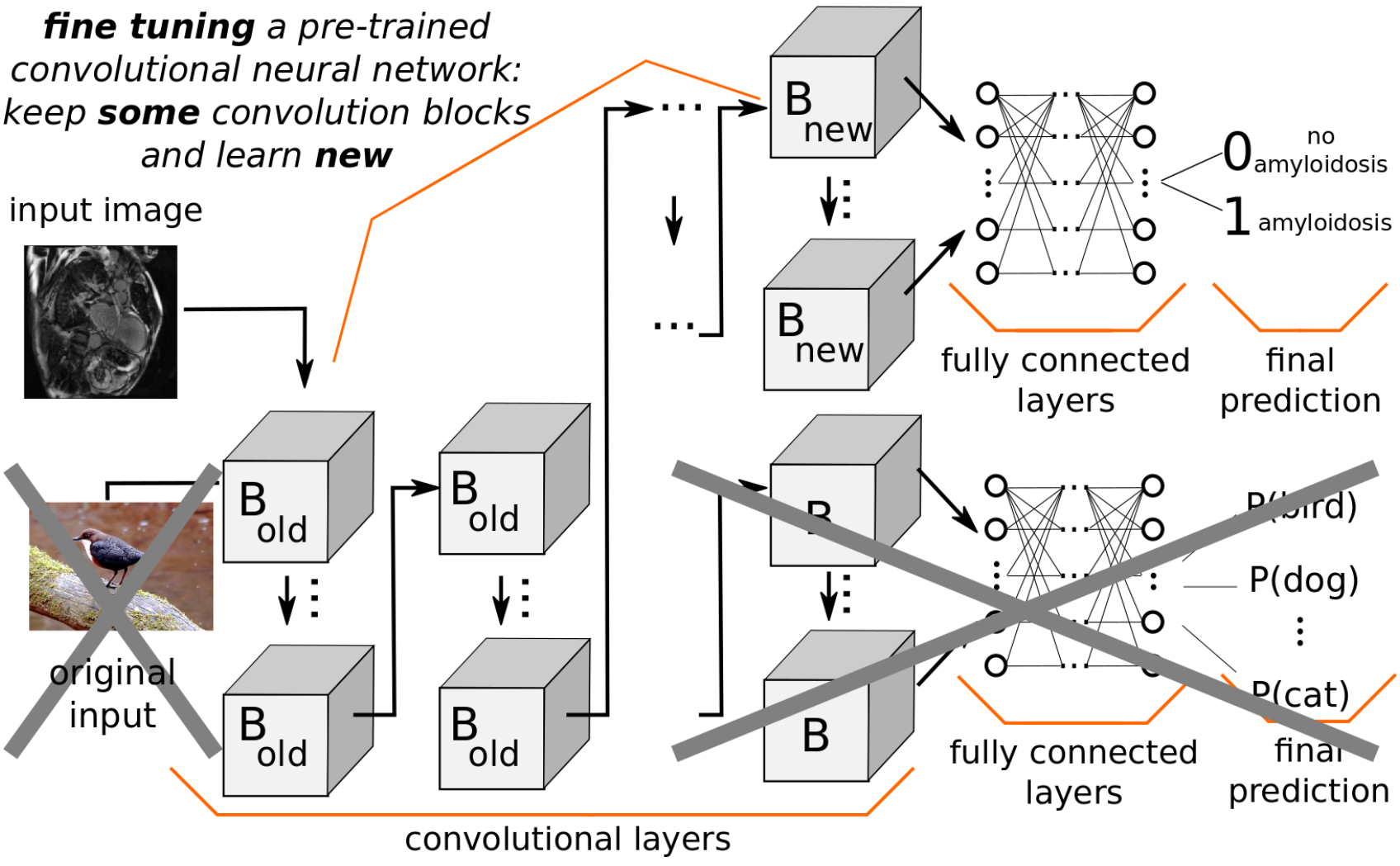


Transfer learning

Feature extraction



Fine tuning

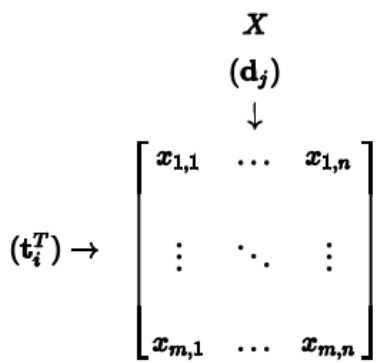

fine tuning *a pre-trained convolutional neural network: keep* ***some*** *convolution blocks and learn* ***new***
input image
original input
B old
B old
B old
B old
B new
B new
B
B
fully connected layers
final prediction
0 no amyloidosis
1 amyloidosis
P(bird)
P(dog)
P(cat)
fully connected layers
final prediction
convolutional layers

Embeddings: representing symbolic data

- Goal: represent symbolic data in a continuous space
  - easily measure relatedness (distance measures on vectors)
  - Linear Algebra and DL to perform complex reasonng
- Challenges
  - discrete nature, easy to count, but not obvious how to represent
  - cannot use backprop (no gradients) on discrete units
  - embedded continuous space can be potentially very large (high-dimensional vector spaces)
  - data not associated to a regular grid structure like image (e.g., text)

Latent Semantic Analysis [2]

- Problem: Find similar documents in a corpus
- Solution:
  - construct term/document matrix - (normalized) occurrence counts



term-document matrix

Example
doc1: the cat is furry
doc2: dogs are furry

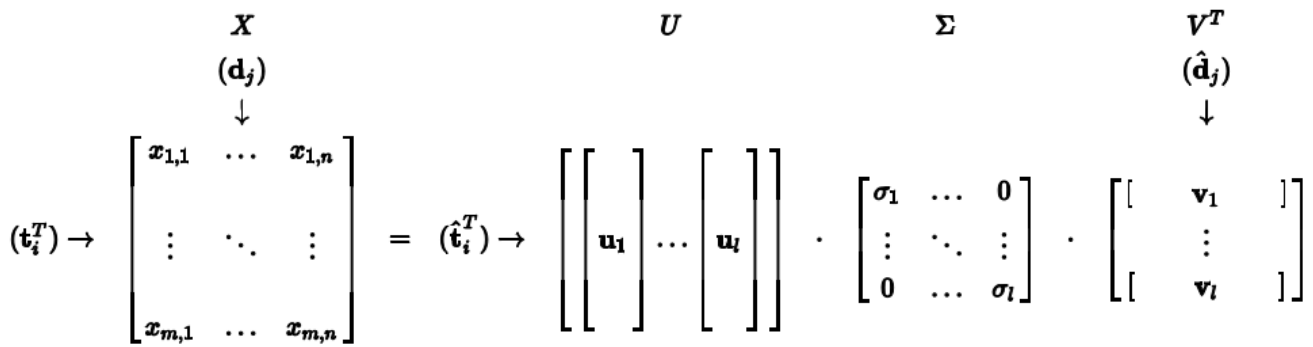
	doc1	doc2
are	0	1
cat	1	0
dogs	0	1
furry	1	1
is	1	0
the	1	0

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

[2] Deerwester et al. "Indexing by Latent Semantic Analysis", JASIS, 1990

Latent Semantic Analysis (contd.)

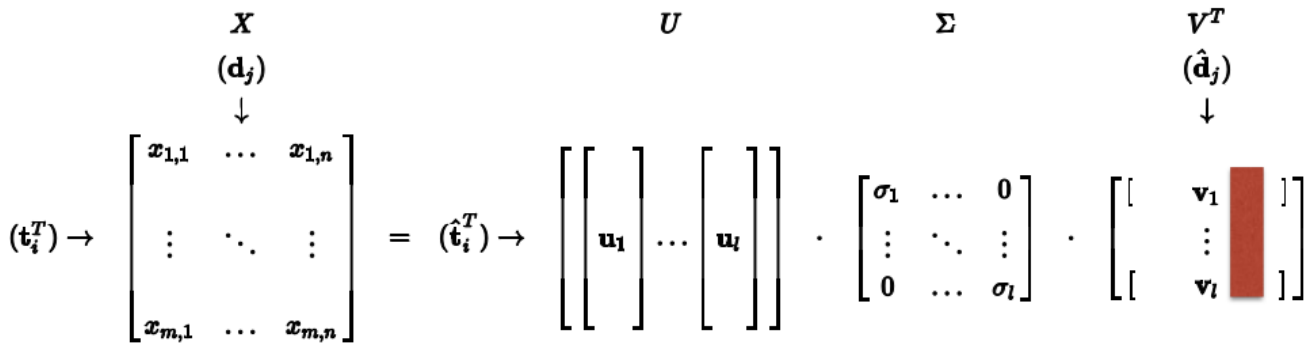
- Problem: Find similar documents in a corpus
- Solution:
  - construct term/document matrix - (normalized) occurrence counts
  - perform SVD (singular value decomposition)
  - $x_{i,j}$ # times word i appears in document j



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

Latent Semantic Analysis (contd.)

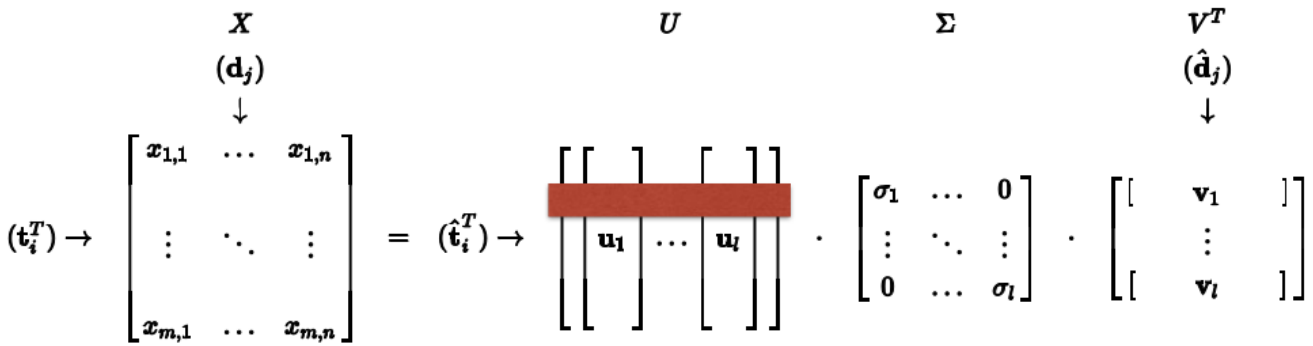
- Each column of V^T - representation of a document in the corpus
- Each column is D-dimensional vector
  - we can use it to compare and retrieve documents
  - $x_{i,j}$ # times word i appears in document j



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

Latent Semantic Analysis (contd.)

- Each row of U - vectorial representation of a word in the dictionary
  - aka **embedding**
  - $x_{i,j}$ # times word i appears in document j



Word embeddings

- Convert words (symbols) into a D dimensional vector
  - D becomes a hyperparameter
- In the embedded space
  - compare words (compare vectors)
  - apply machine learning (DL) to represent sequence of words
  - use weighted sum of embeddings (linear combination of vectors) for document retrieval

Bi-gram

- Bi-gram models probability of a word given the preceding one

$$p(w_k|w_{k-1}), w_k \in V$$

- Bi-gram model builds a matrix of counts

$$c(w_k|w_{k-1}) = \begin{bmatrix} c_{1,1} \dots c_{1,|V|} \\ \dots c_{i,j} \dots \\ c_{|V|,1} \dots c_{|V|,|V|} \end{bmatrix}$$

- $c_{i,j}$ number of times word i is preceded by word j

Factorized bi-gram

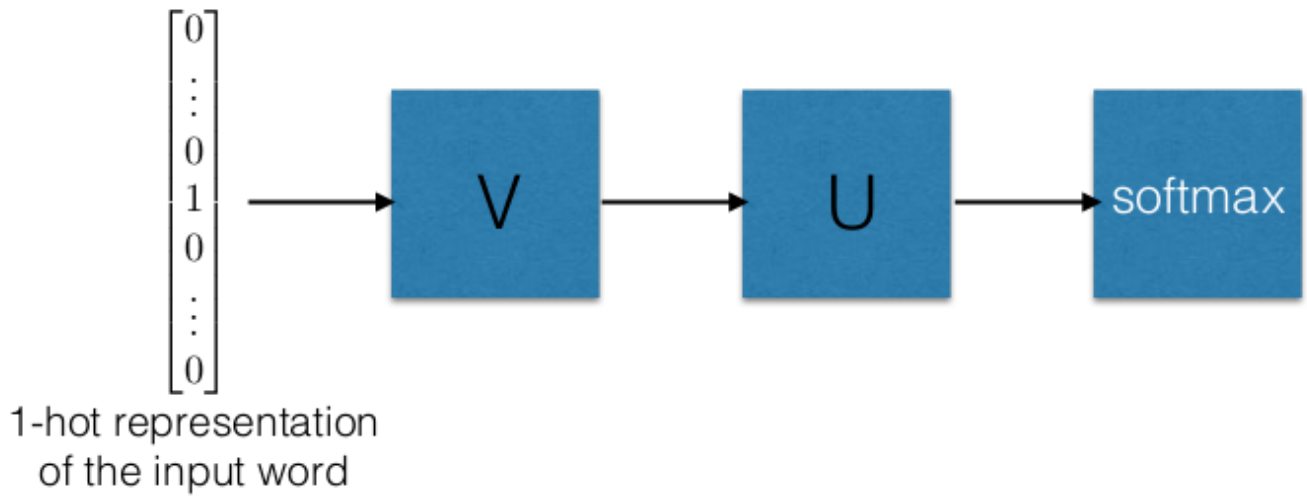
- factorize (via SVD) the bigram matrix
  - reduce # of parameters to learn
  - become more robus to noise (entries with low counts)

$$c(w_k|w_{k-1}) = \begin{bmatrix} c_{1,1} \dots c_{1,|V|} \\ \dots c_{i,j} \dots \\ c_{|V|,1} \dots c_{|V|,|V|} \end{bmatrix} = UV$$

- rows of $U \in \mathbb{R}^{|V| \times D}$ - "output" word embeddings
- columns of $V \in \mathbb{R}^{D \times |V|}$ - "input" word embeddings

Factorize bi-gram with NN

- we could express UV factorization with a two layer (linear) neural network

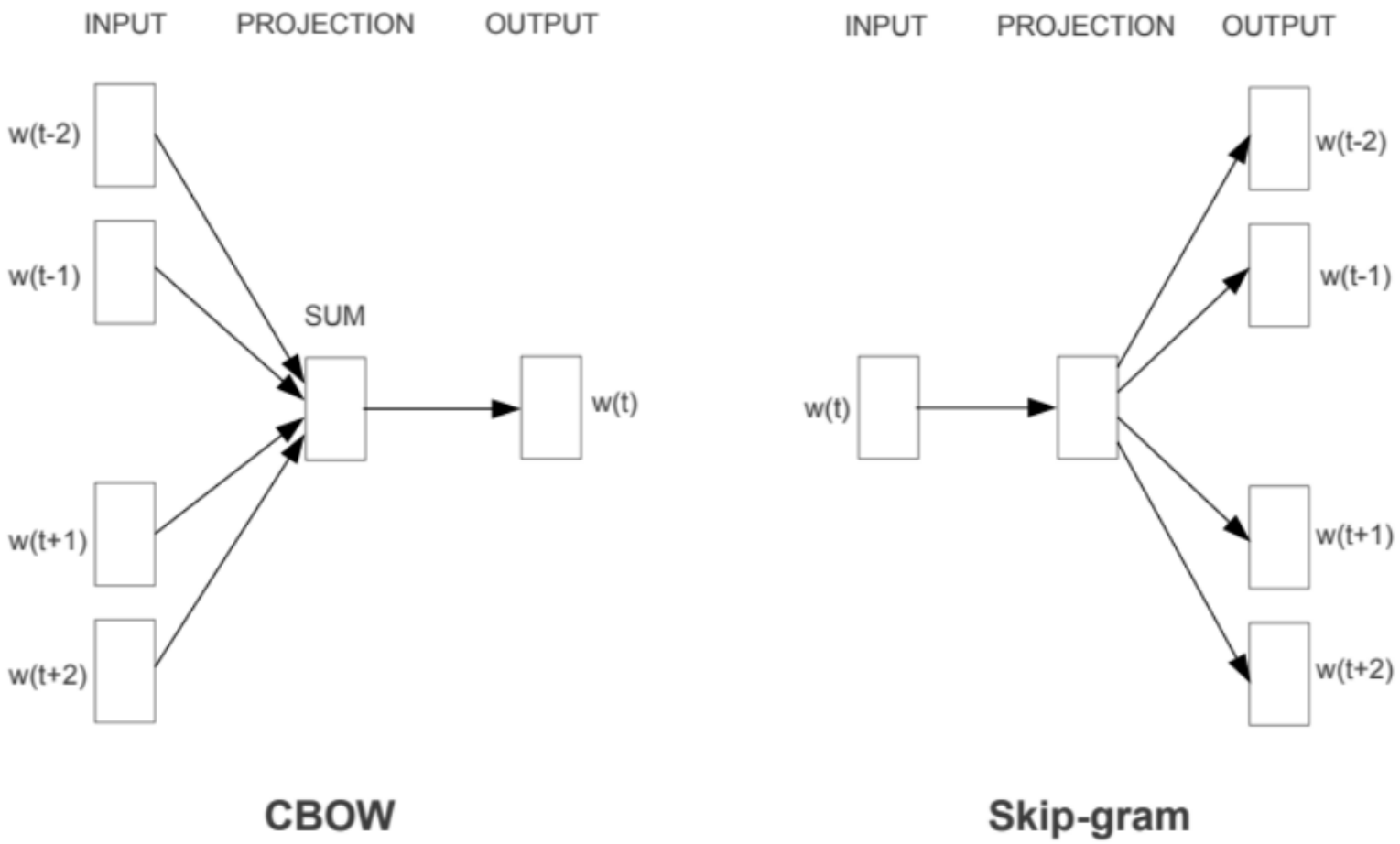


(Image credit) Course notes "An introduction to Deep Learning", Marc'Aurelio Ranzato

Recap

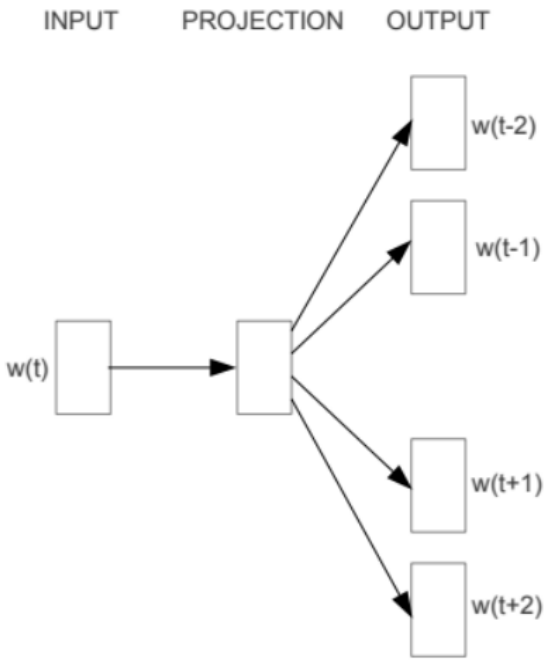
- LSA learns word embeddings via co-occurences across documents
- bi-gram learns word embeddings that only take into account next word
- if you combine the two you get **word2vec**
  - use more context around the word ($n \geq 2$)
  - but look for context only around the word of interest

Word2Vec: Continuous bag of words



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

Word2Vec: Skip-gram

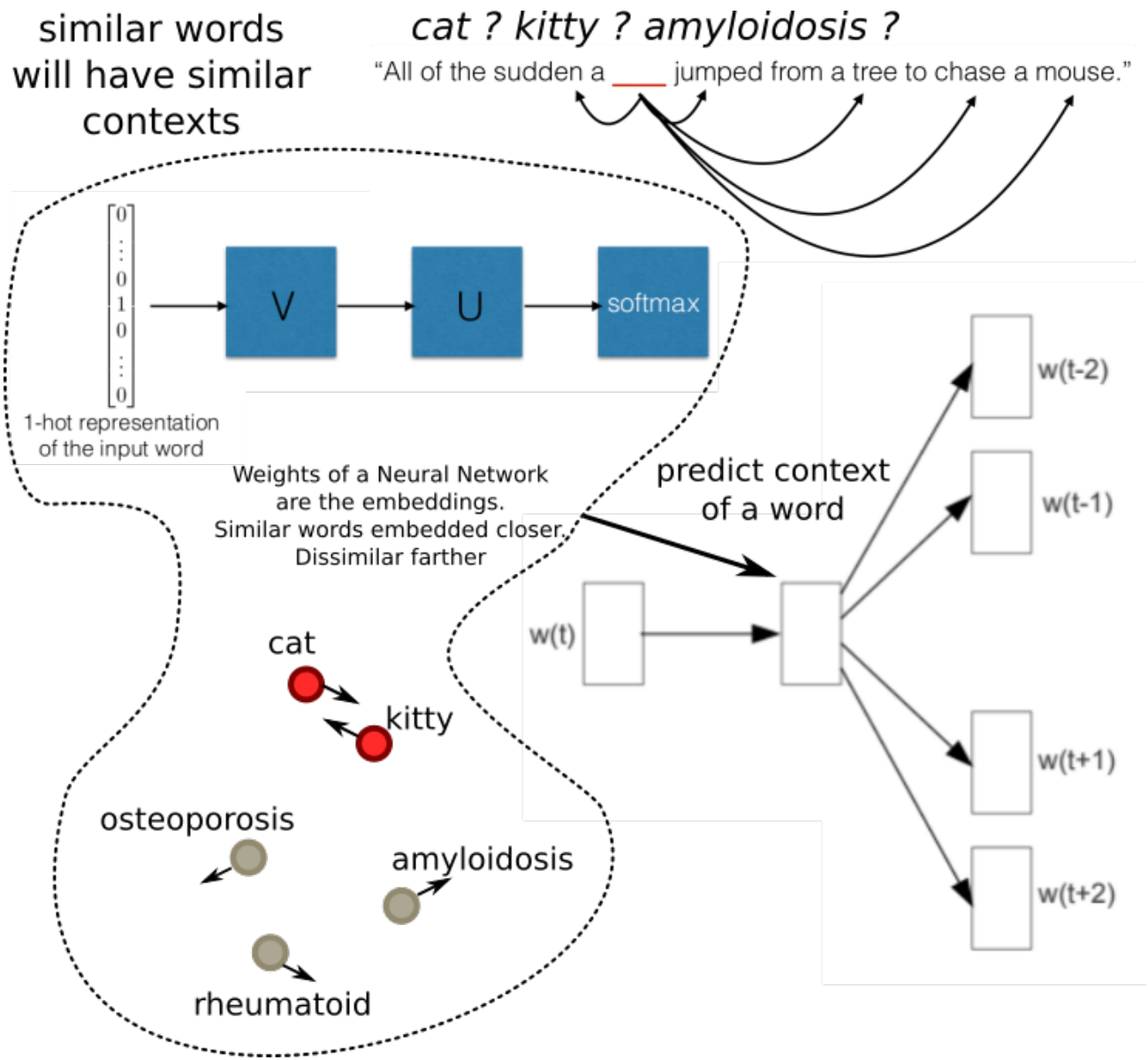


Skip-gram

- similar to *bi-gram*, but predict N preceding and N following words
- words with same context will have similar embedding (e.g., cat & kitty)
- input projection - look-up table
- bulk of computation - prediction of words in context
- learning by cross-entropy minimization via SGD

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

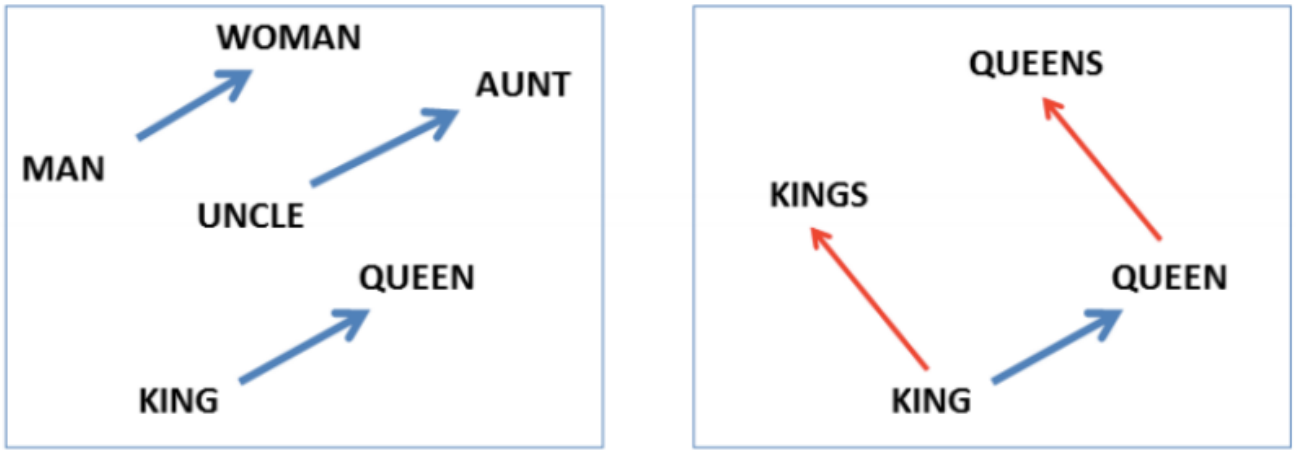
Continous representations for discrete data



Mikolov et al., "word2vec" NIPS 2014

Word2Vec: Linguistic regularities in Word Vector Space

- word vector space implictly encodes linguistic regularities



- distributed word representations implicitly contain syntactic and semantic information
  - **KING** is similar to **QUEEN** as **MAN** is similar to **WOMAN**
  - **KINGS** is similar to **KINGS** as **MAN** is similar to **MEN**

(Image credit) "Efficient estimation of word representations", Mikolov et al. NIPS 2013

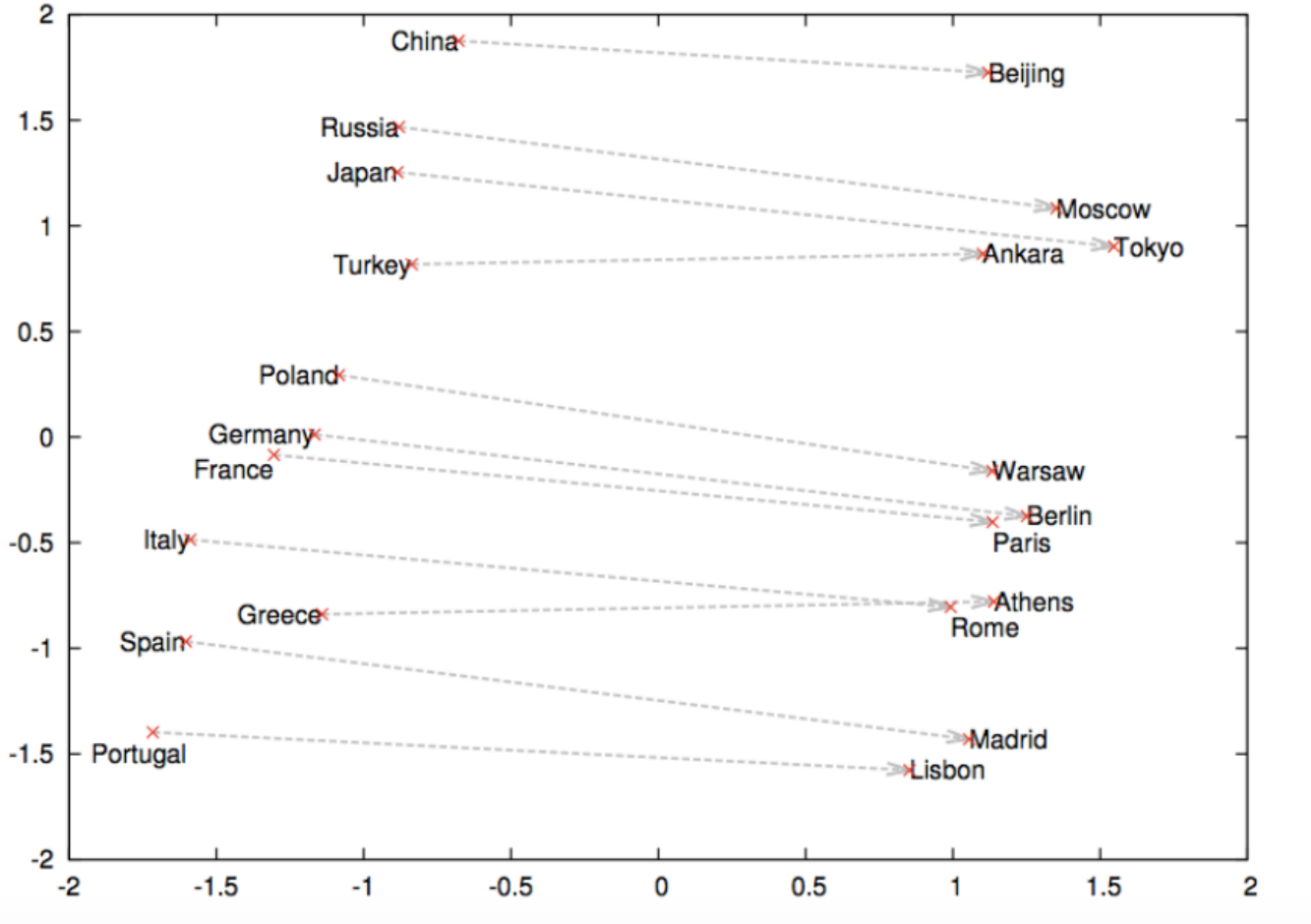
Word2Vec: Vector operations in Word Vector Space

- vector operations (addition/subtraction) give intuitive results

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

(Image credit) "Efficient estimation of word representations", Mikolov et al. NIPS 2013

Word2Vec: Linguistic regularities in Word Vector Space (contd.)

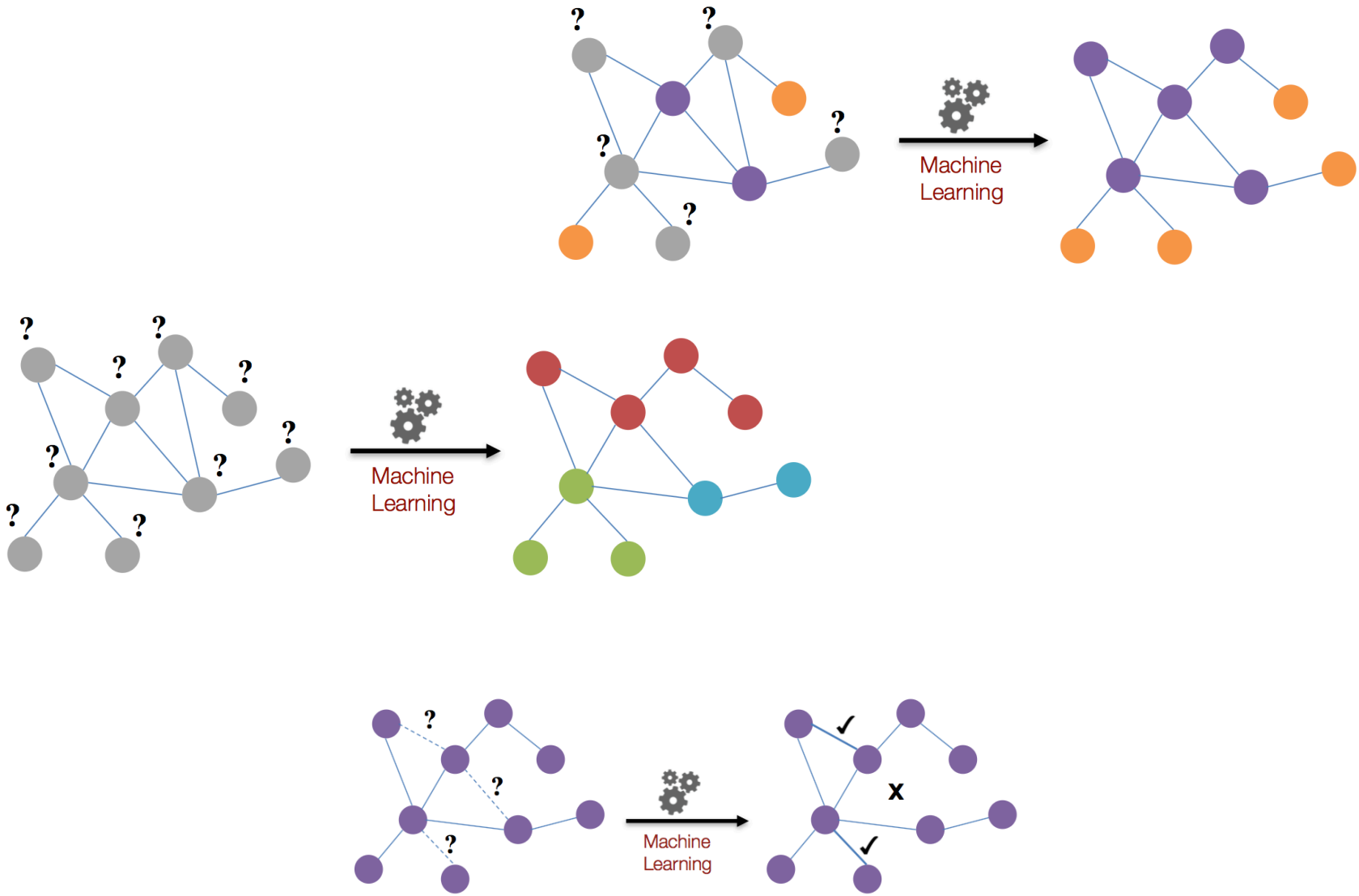


(Image credit) "Efficient estimation of word representations", Mikolov et al. NIPS 2013

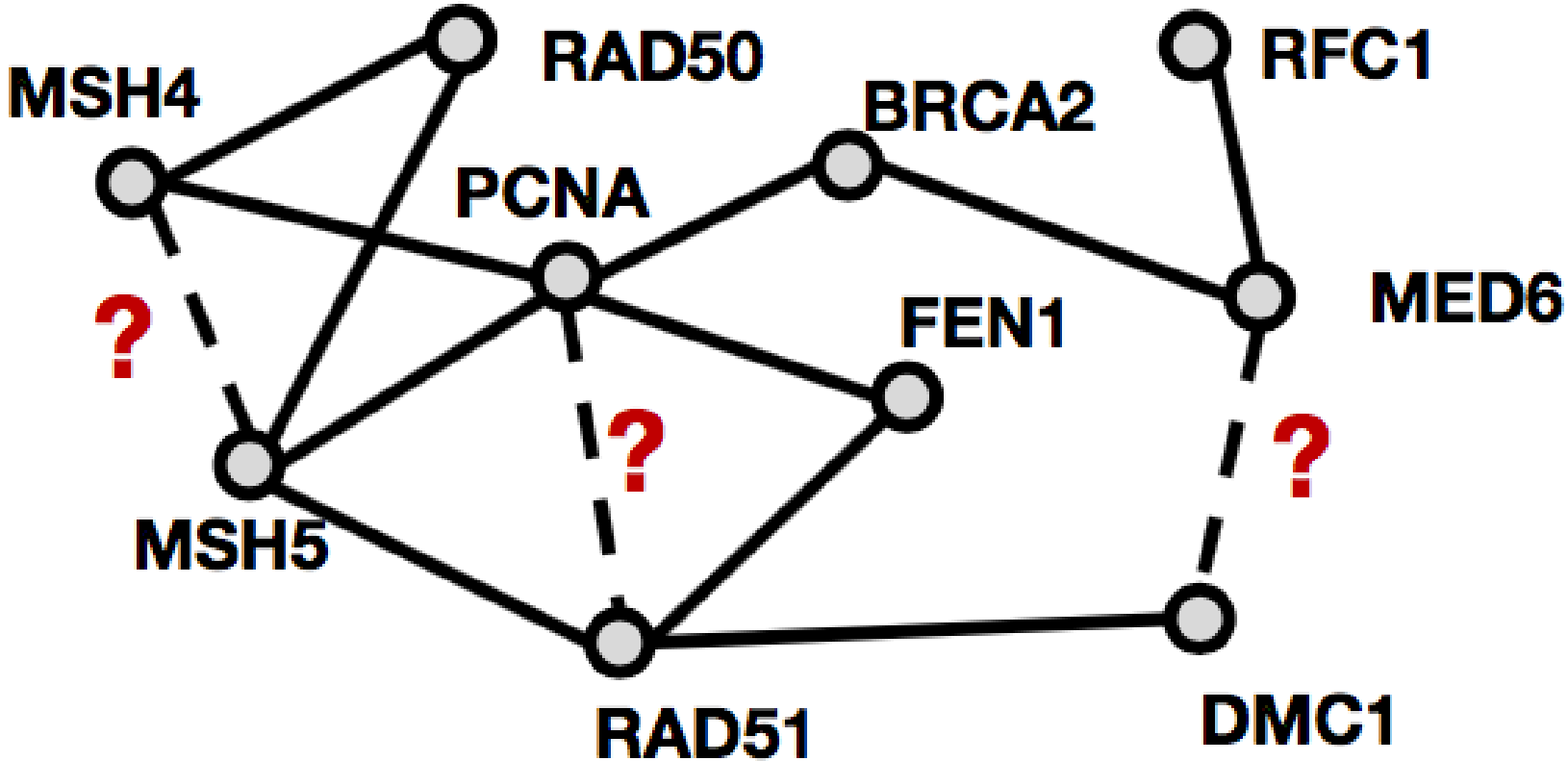
Recap

- Embeddings of words (from 1-hot to distributed representation):
  - understand similarity between words
  - plug them within any parametric ML model
- Several ways to learn word embeddings
  - **word2vec** among the most popular ones
- **word2vec** leverages large amounts of *unlabeled* data

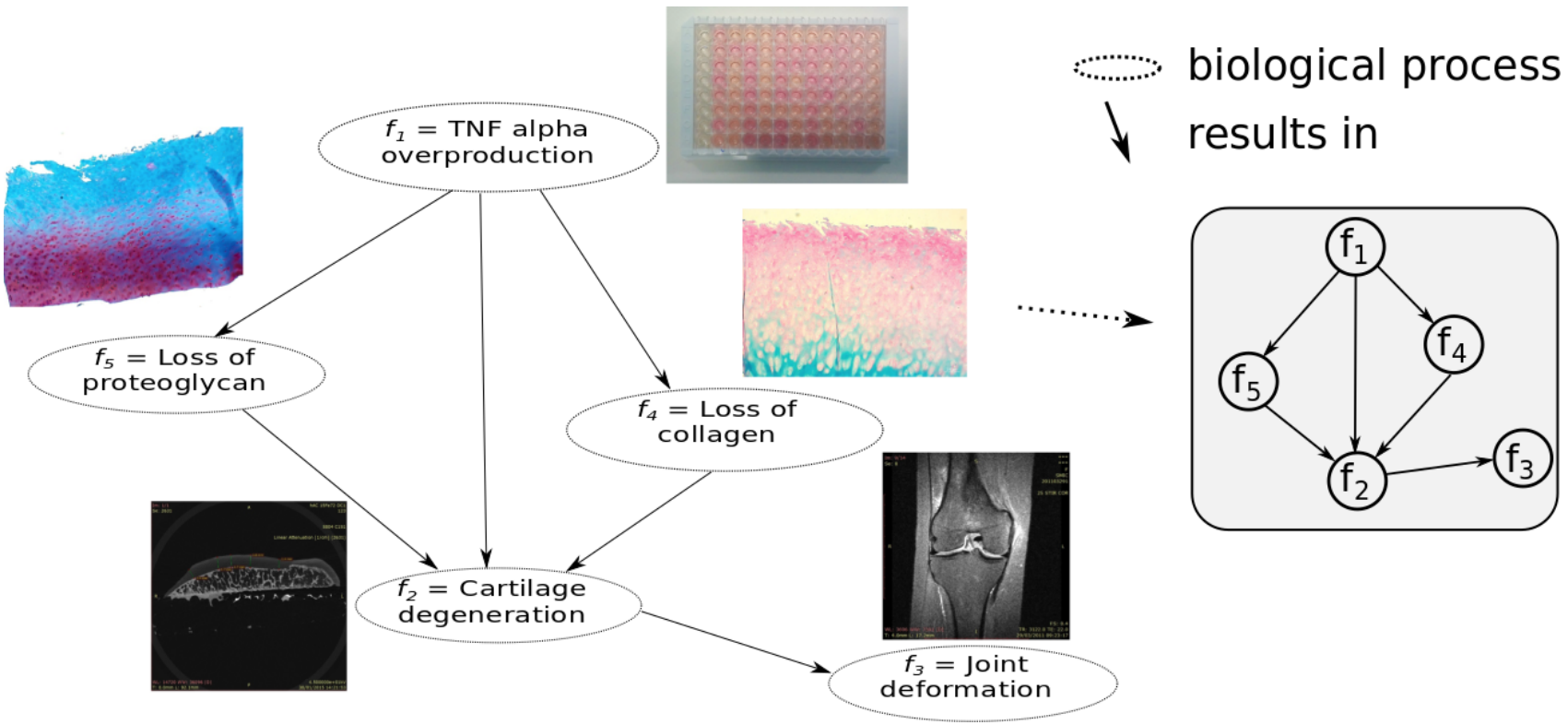
AI for network biology



AI for network biology: link prediction

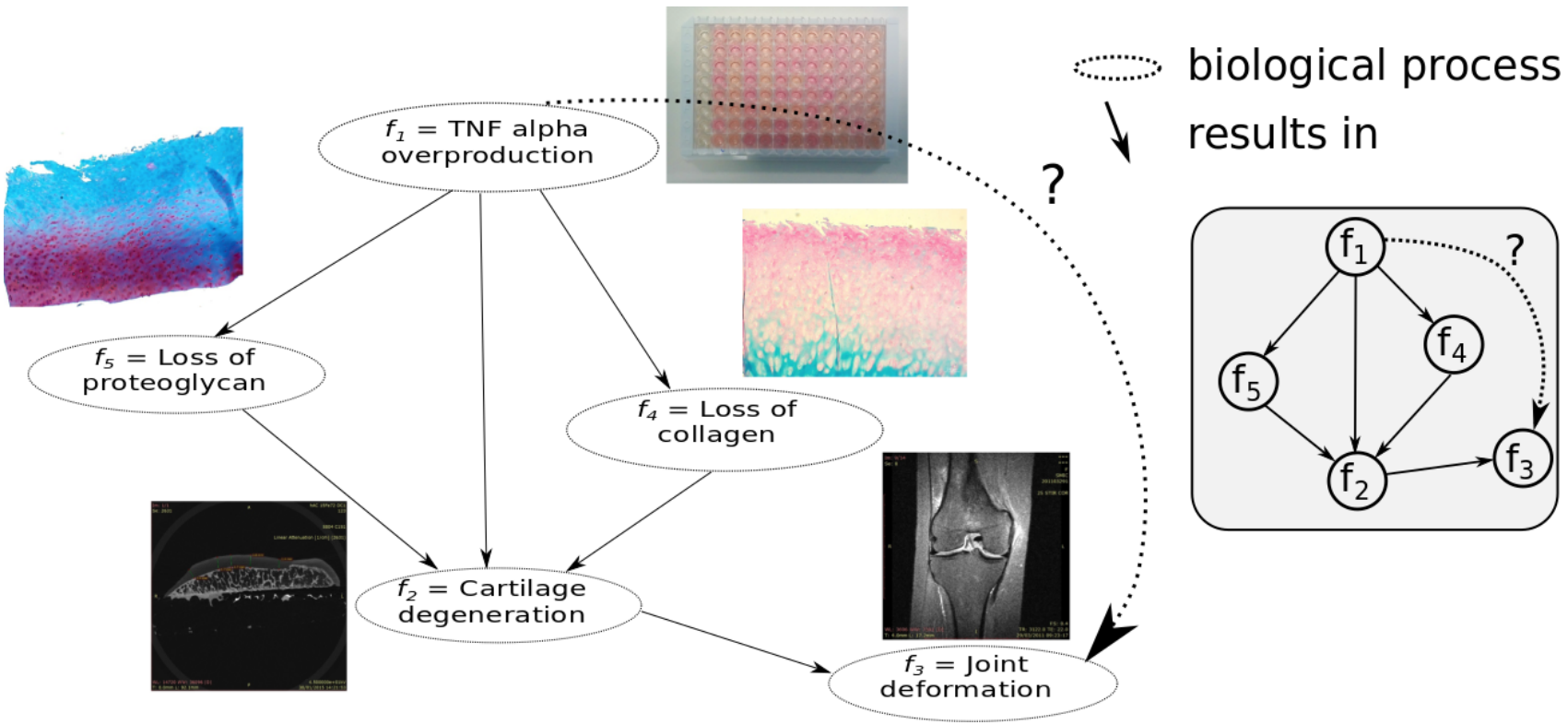


Representing biological knowledge



Agibetov et al., "Shared hypothesis testing", J. Biomed. Sem., 2018

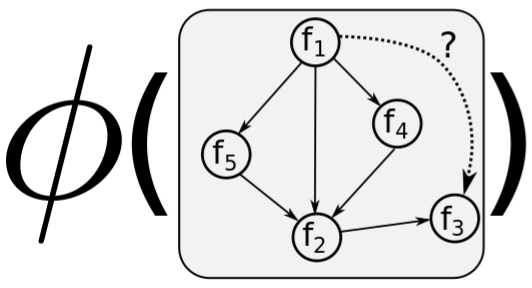
Biological link prediction



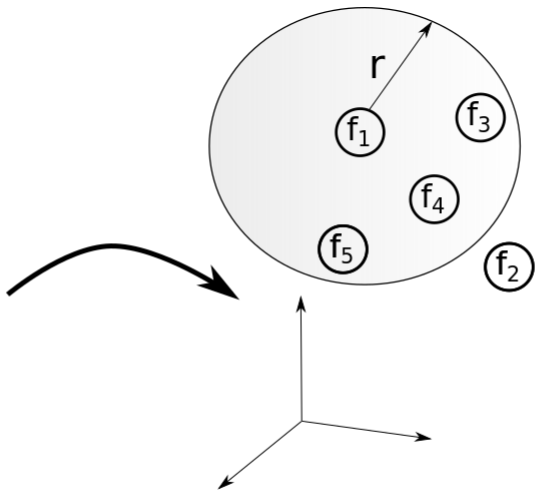
Agibetov et al., "Shared hypothesis testing", J. Biomed. Sem., 2018

Link prediction as distance based inference in embedding space

How to learn ϕ ?



Graph domain



Embedding space $\mathbb{R}^n$

inference in embedding space based on distance

Points within r-ball centered around fixed point are connected in the graph domain

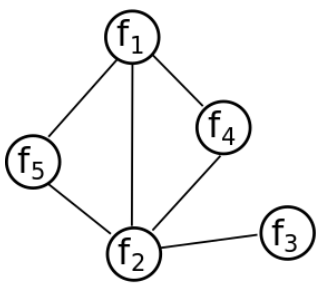
$d(\phi(f_1), \phi(f_3)) \leq r \implies$

f1 and f3 have a link

Learning graph embeddings

- Learn *link estimate* $Q(u, v) \mapsto [0, 1]$ (u, v node pairs) and approximate graph structure (connectivity) with MLE (maximum likelihood estimation)
  - $Pr(G) = \prod_{(u,v) \in E_{train}} Q(u, v) \prod_{(u,v) \notin E_{train}} 1 - Q(u, v)$

- If Q perfect estimator then $Pr(x) = 1$ iff $x = G$ (i.e., graph can be fully reconstructed)
- Q can be *trained* to estimate links at different *orders*, i.e., approximate A^n.



1-order paths

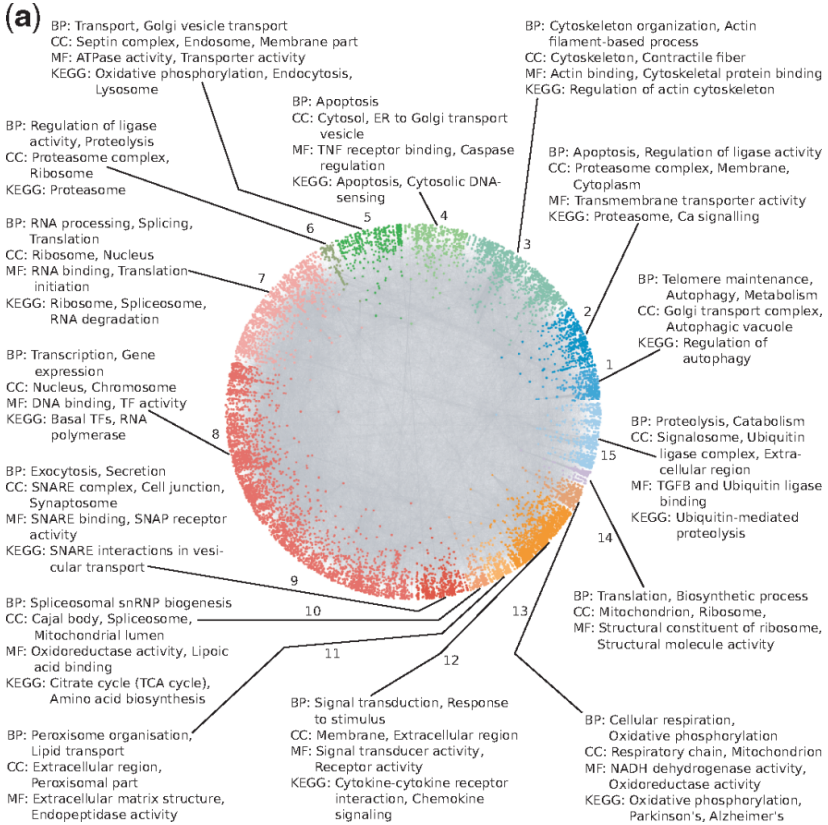
$$A = \begin{array}{c} \\ \mathsf{f}_1 \\ \mathsf{f}_2 \\ \mathsf{f}_3 \\ \mathsf{f}_4 \\ \mathsf{f}_5 \end{array} \begin{array}{c} \begin{array}{ccccc} \mathsf{f}_1 & \mathsf{f}_2 & \mathsf{f}_3 & \mathsf{f}_4 & \mathsf{f}_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

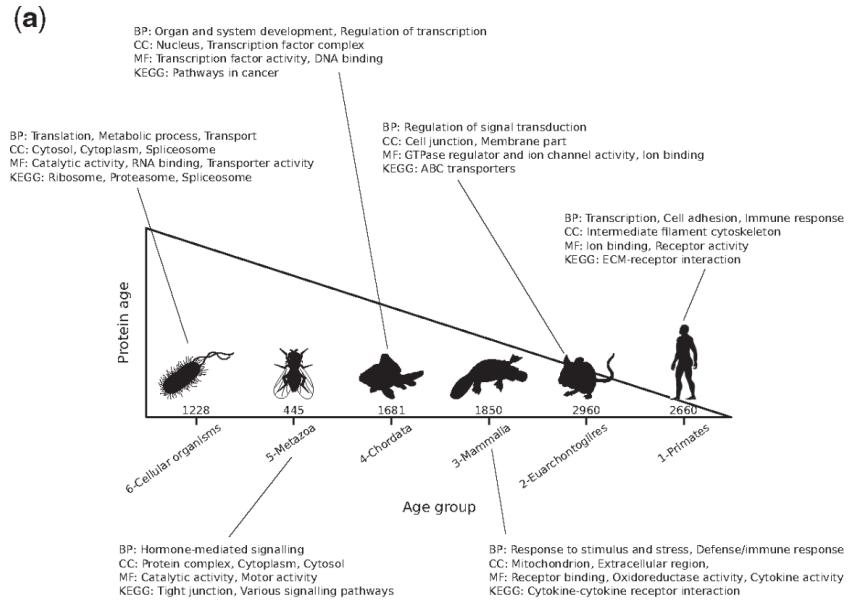
2-order paths

$$A(A) = A^2 = \begin{array}{c} \\ \mathsf{f}_1 \\ \mathsf{f}_2 \\ \mathsf{f}_3 \\ \mathsf{f}_4 \\ \mathsf{f}_5 \end{array} \begin{array}{c} \begin{array}{ccccc} \mathsf{f}_1 & \mathsf{f}_2 & \mathsf{f}_3 & \mathsf{f}_4 & \mathsf{f}_5 \end{array} \\ \begin{bmatrix} 3 & 2 & 1 & 1 & 1 \\ 2 & 4 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 2 & 2 \end{bmatrix} \end{array}$$

Haija, et al., "Graph likelihood", CIKM17, NeurIPS 2018

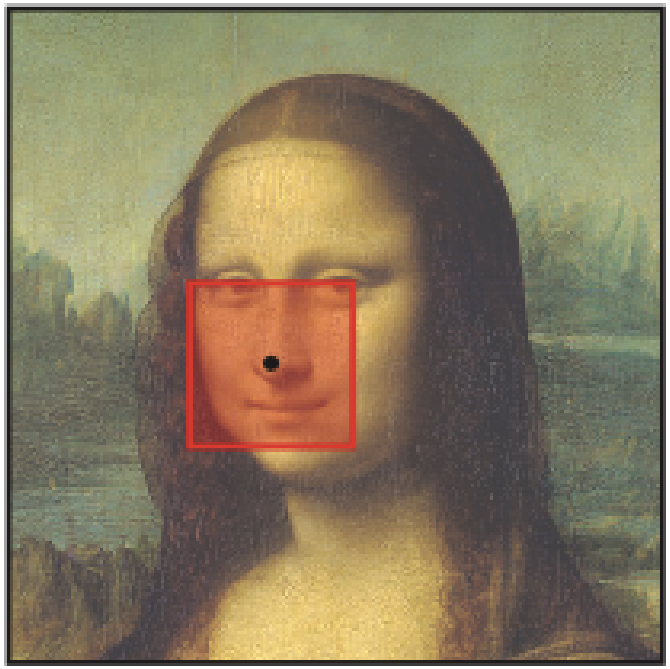
Clusters of proteins and age groups from hyperbolic coordinates



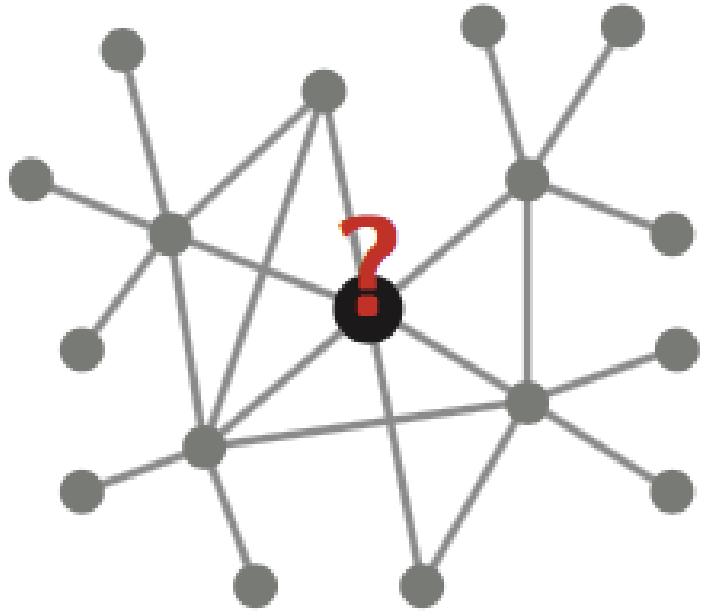


Lobato et al. Bioinformatics 2018

Graph Convolutional Neural Networks (GCN)



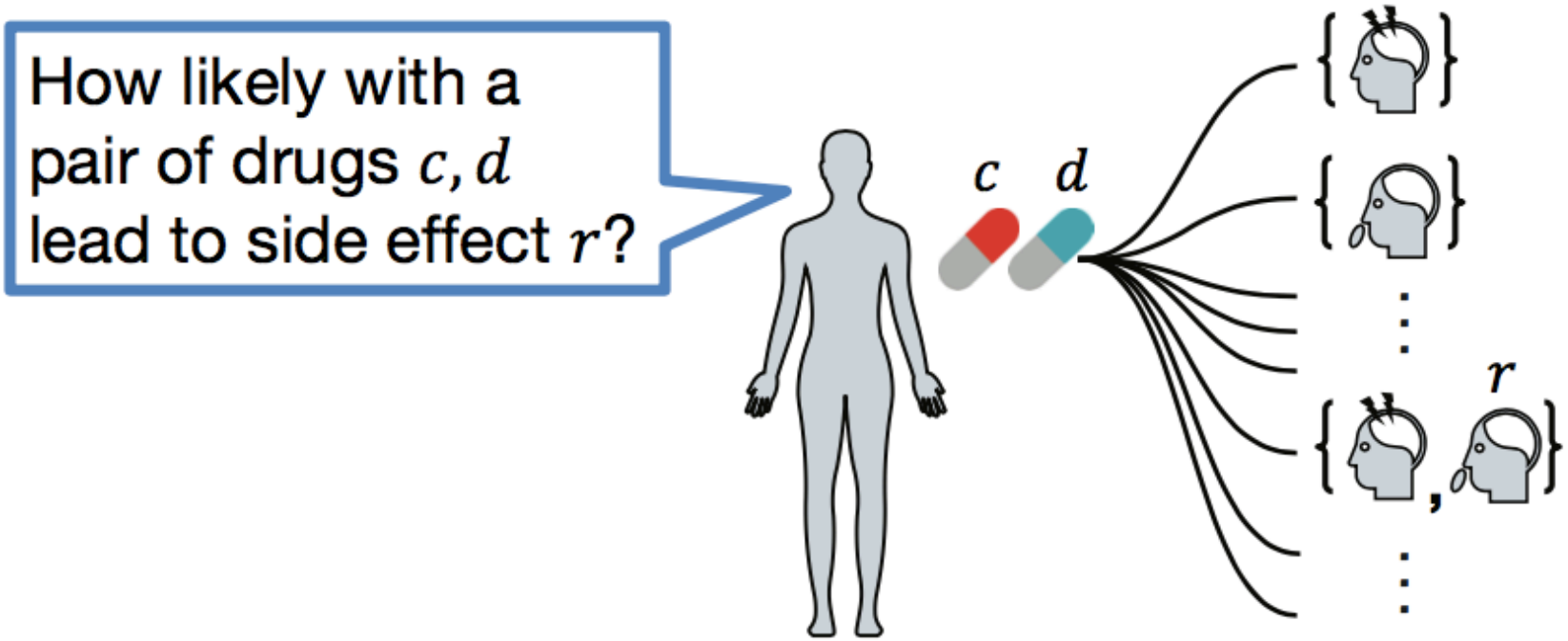
Image



Graph

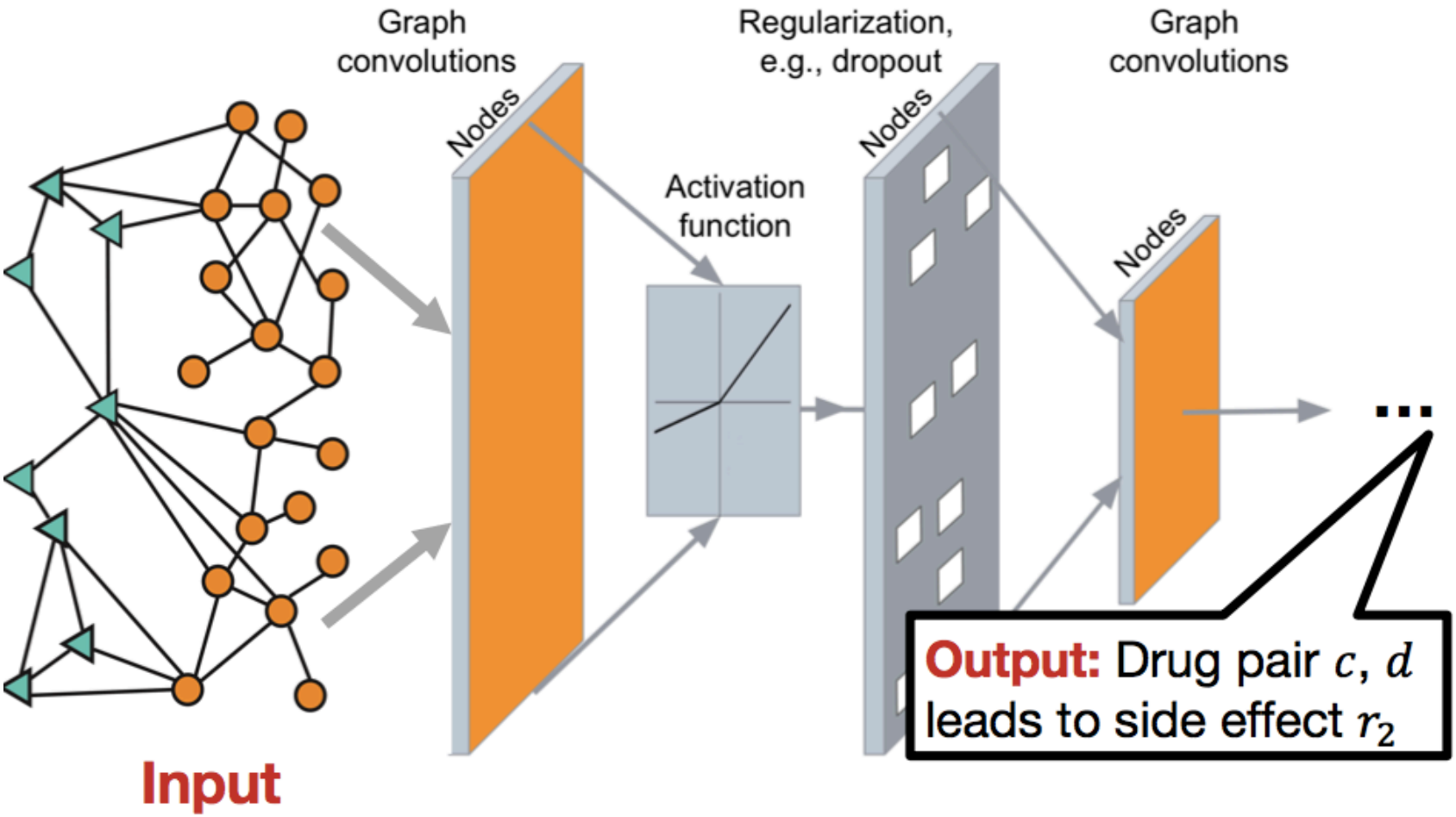
Bronstein. "GCN Tutorial". 2019

Polypharmacy and side effects link prediction



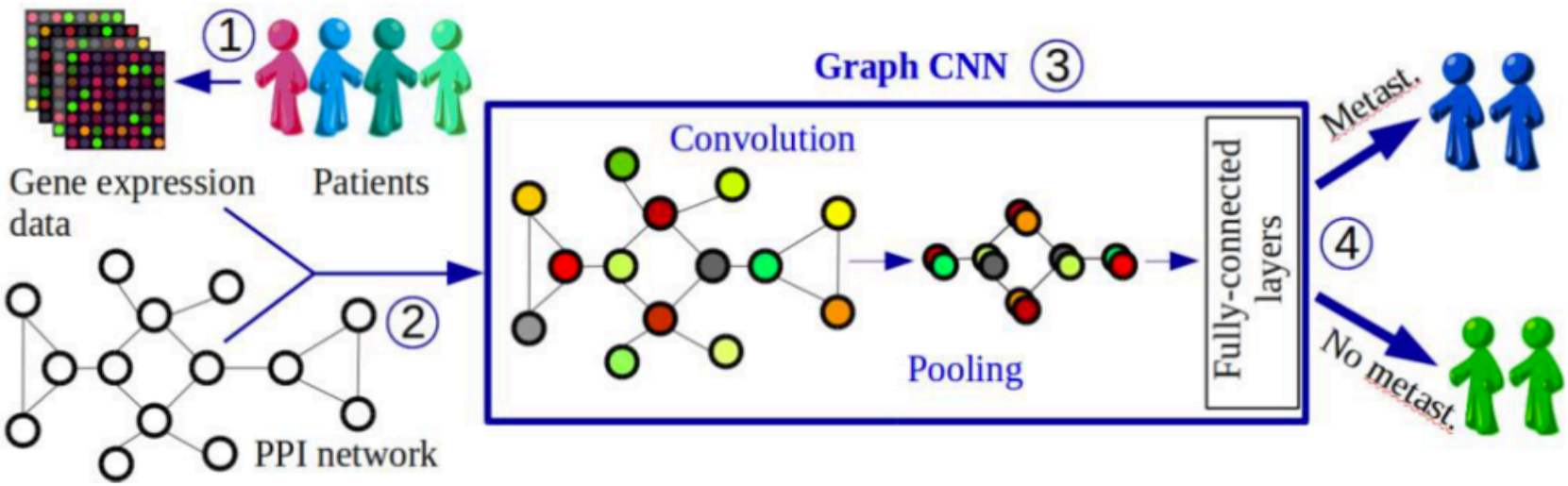
Zitnik et al. Oxford. 2018

GCN applications



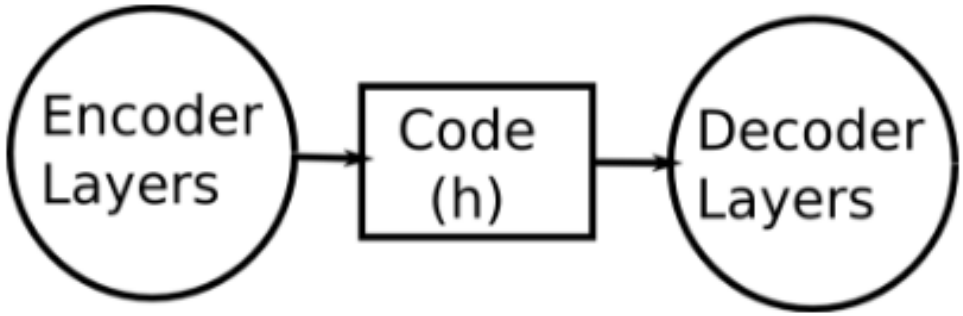
Zitnik et al. Bioinformatics. 2018

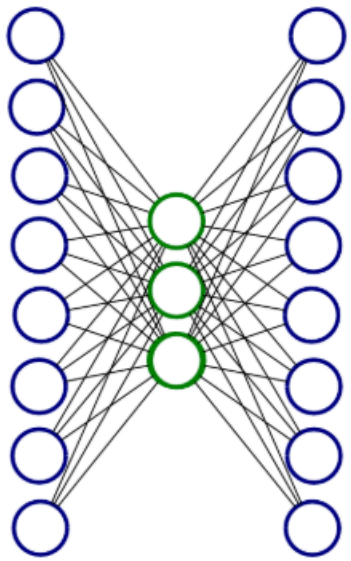
Predicting metastatic events in breast cancer



Chereda H. et al. "Utilizing Molecular Network Information via Graph Convolutional Neural Networks to Predict Metastatic Event in Breast Cancer." Stud. Health. Technol. Inform. 2019

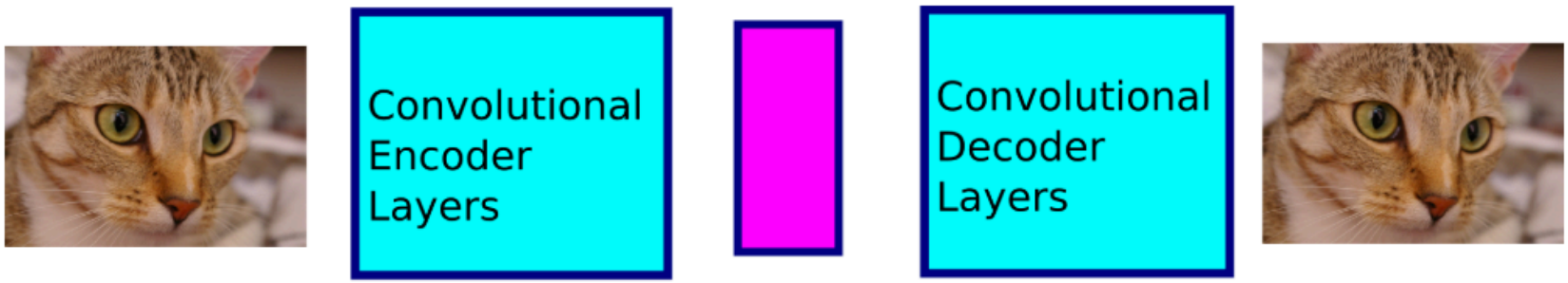
Autoencoders: learning data distributions





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

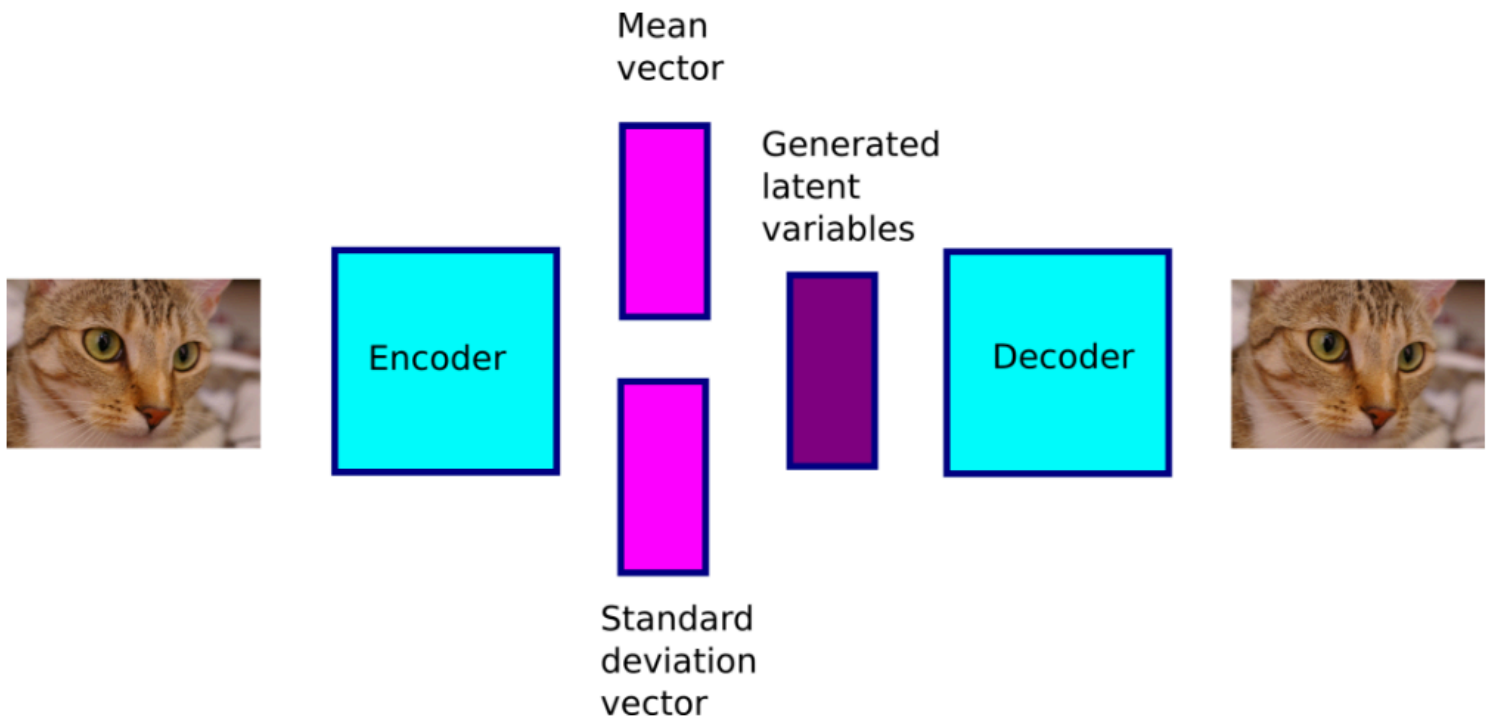
Convolutional autoencoders



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

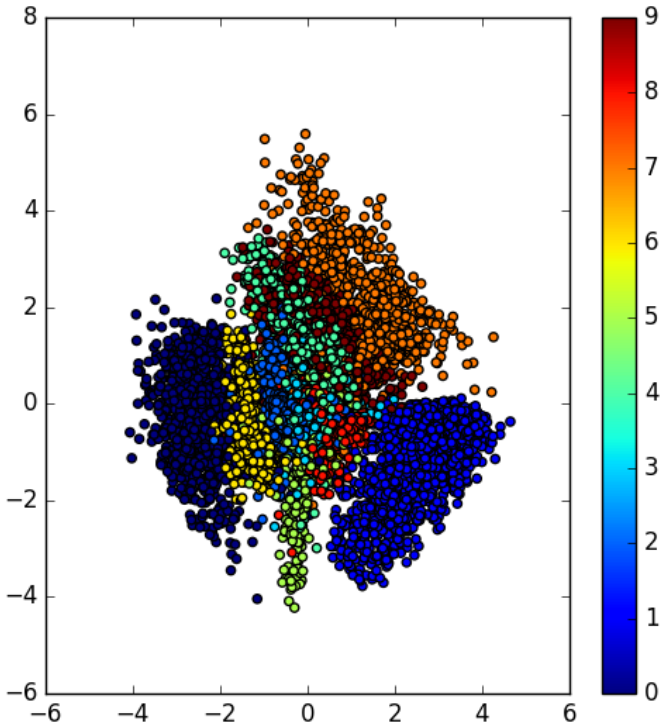
Variational auto encoder (VAE)

- 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

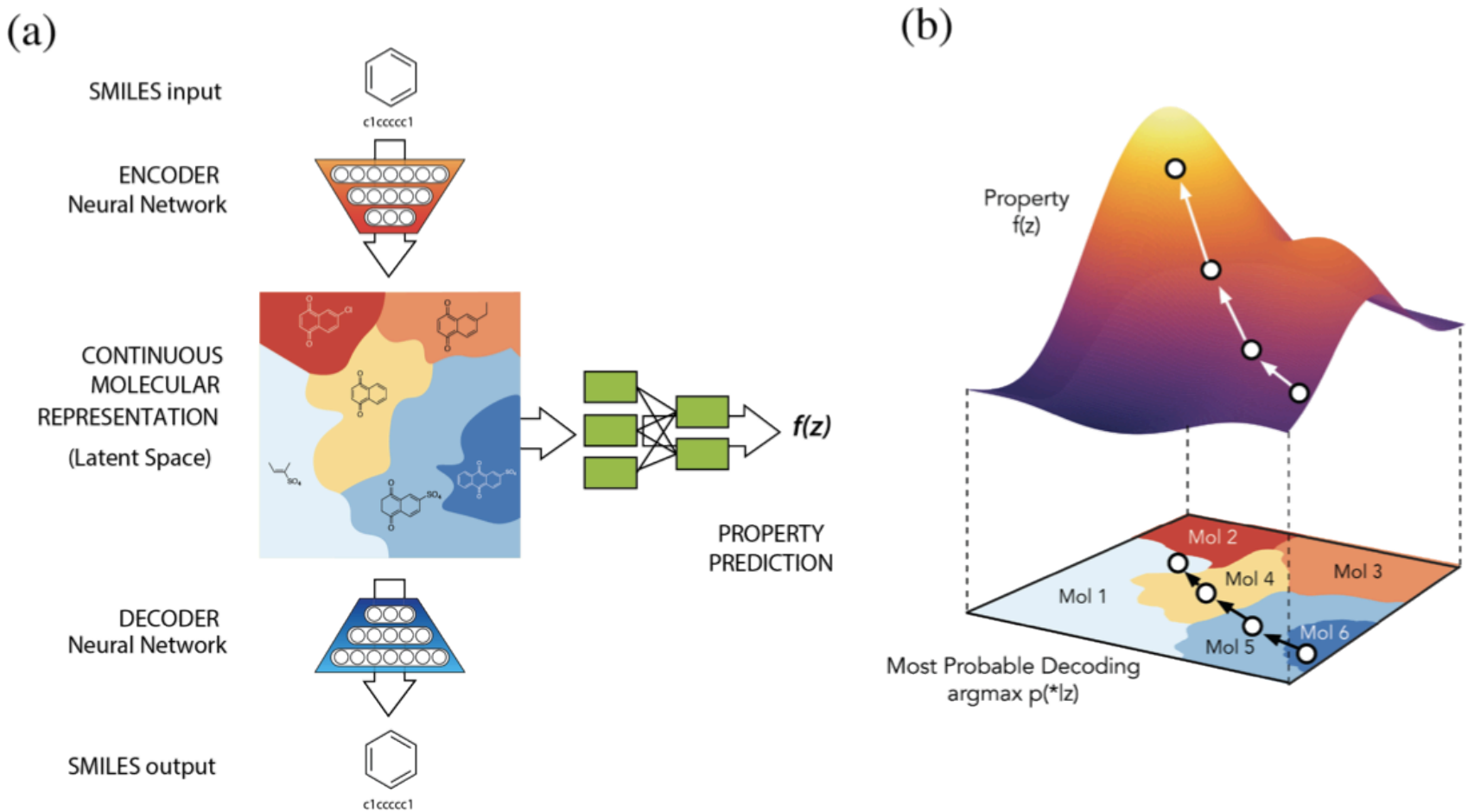
VAE on MNIST



- 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/

Generating molecules with desired chemical properties



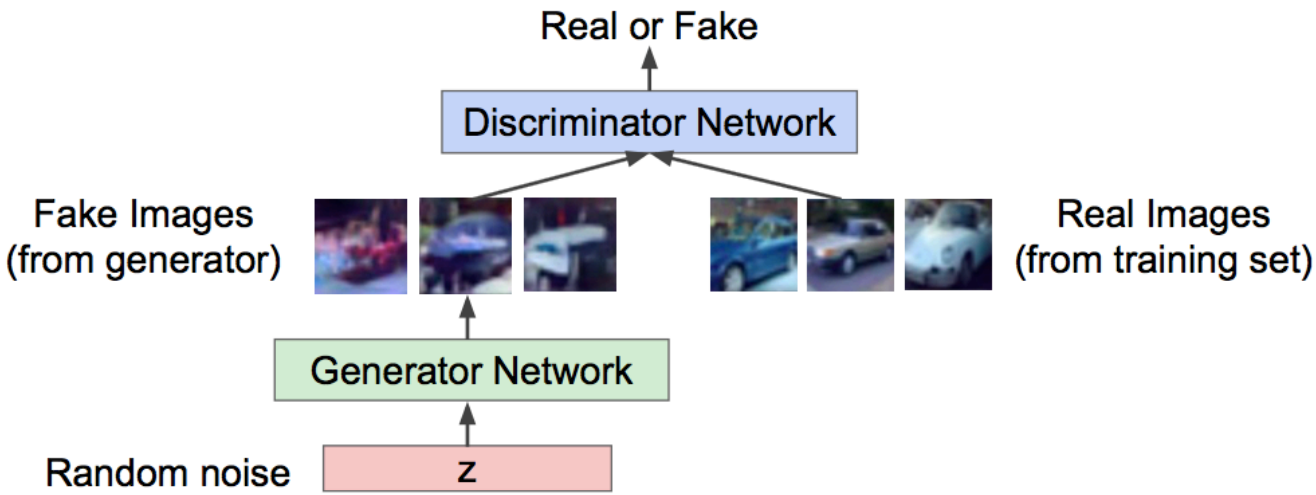
(Image credit) Gomez-Bombarelli "Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules" arxiv. 2017

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

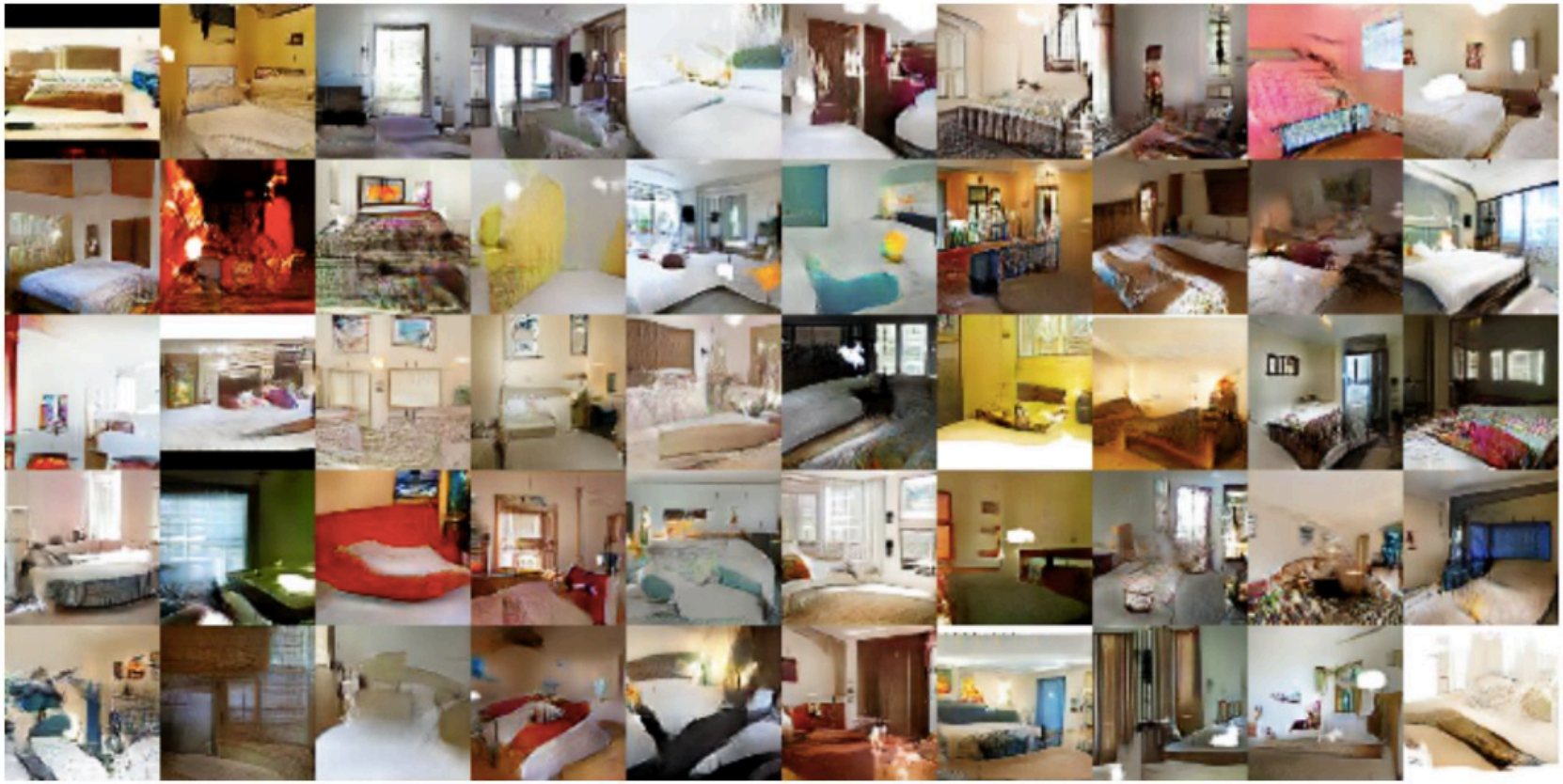


(Image credit) Course notes "CNN for Visual Recognition" (Stanford, Spring 2017)

Difference from VAE

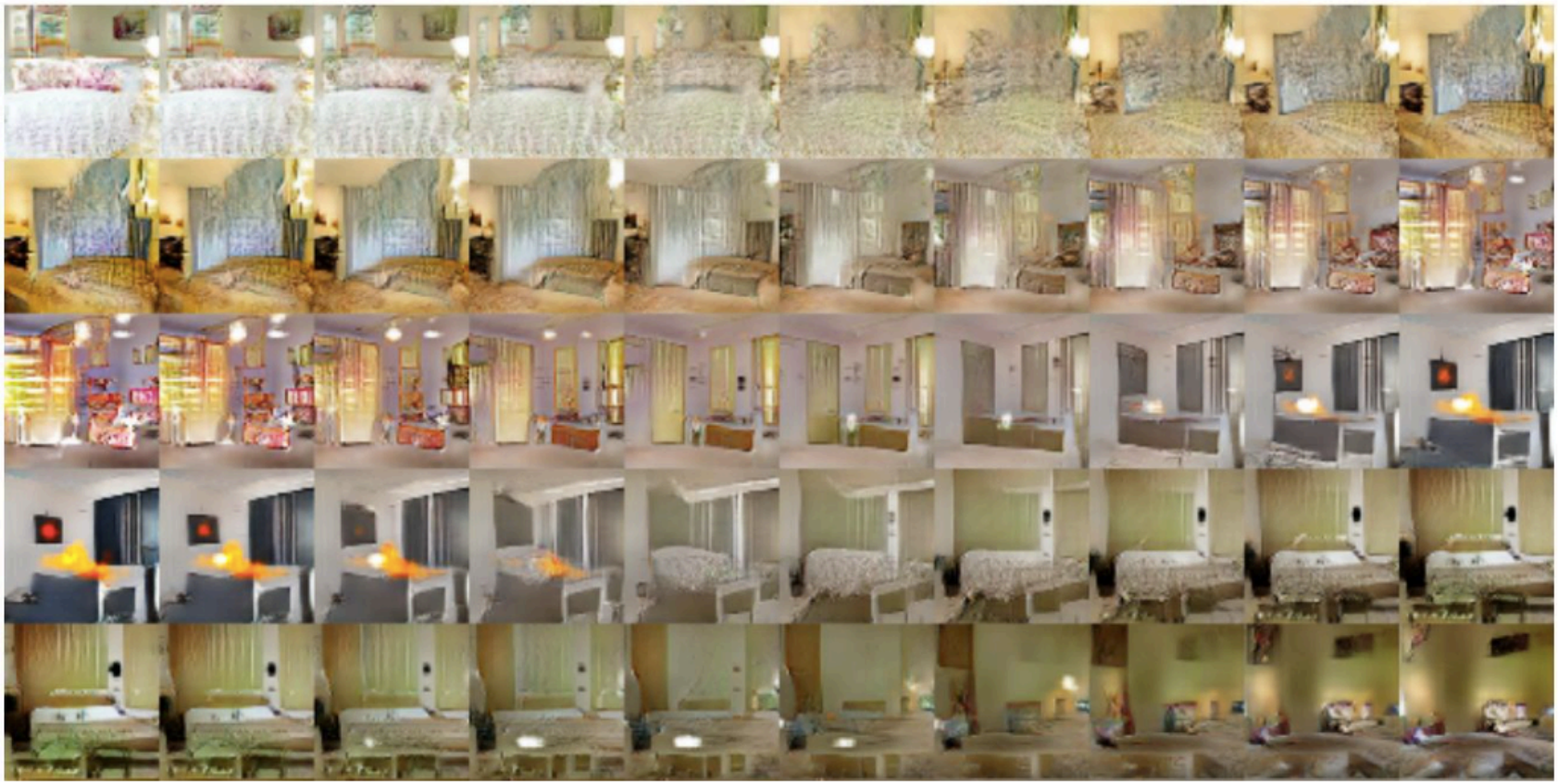
- In VAE we try to minimize a lower bound of an intractable likelihood function
- With GANs we use a game-theoretic approach instead
  - learn to generate from training distribution by
  - sampling from a simple distribution, e.g., random noise, learn transformation to training distribution
- Two networks
  - 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)

Deep Learning frameworks

- Goals:
  1. Easily build big computation graphs
  2. Easily compute gradients in computational graphs (automatic gradient computation)
  3. Run it all efficiently on GPU (wrap low level NVIDIA and Linear Algebra libraries (e.g., cuDNN, cuBLAS))
- Academia/Industry open source frameworks
  - Caffe (UC Berkeley) $\mapsto$ Caffe2 (Facebook)
  - Torch (NYU/Facebook) $\mapsto$ PyTorch (Facebook)
  - Theano (U Montreal) $\mapsto$ TensorFlow (Google)
- Industry (not necessarily open source) frameworks
  - Paddle (Baidu), CNTK (Microsoft), MXNet (Amazon), and others...
- High-level frameworks
  - Keras (Theano, TensorFlow or CNTK as backend)
    - good for beginners

Possible exam questions

- One question on decision trees
  - use information gain metric to choose the best split in the decision tree
- One question on convolutional neural networks or GCN
  - explain the difference of a convolutional layer on a regular grid vs. on a graph
- One question on embeddings or VAE or LSTM
  - you will be presented with a modified version of VAE and you will be asked to explain its advantages over the vanilla VAE

What is next

- (upcoming) Full course "Aktuelle Themen der Medizinischen Informatik - Deep Learning Notebooks"
  - More in-depth, interactive introduction to Deep Learning
  - More examples of DL for medical imaging, network biology with hands on sessions
  - New topics including: deep learning on medical text (fastText, ELMO, BERT), adversarial training, explainable AI, deep learning on tabular data, connections of DL methods to traditional statistical approaches, functional programming and topological data analysis
- Starts Feb 25 through May 12
  - 10 lectures, Tuesdays 09:15 - 11:30
  - format of lecture 90 minutes theory, 45 minutes hands-on (bring your laptops)
  - 3 exam dates (May-June), exam duration 90 minutes OR a project
- Location AID Seminarraum (Waehringer Strasse 25A, 1090)
  - MUW Studienabteilung building