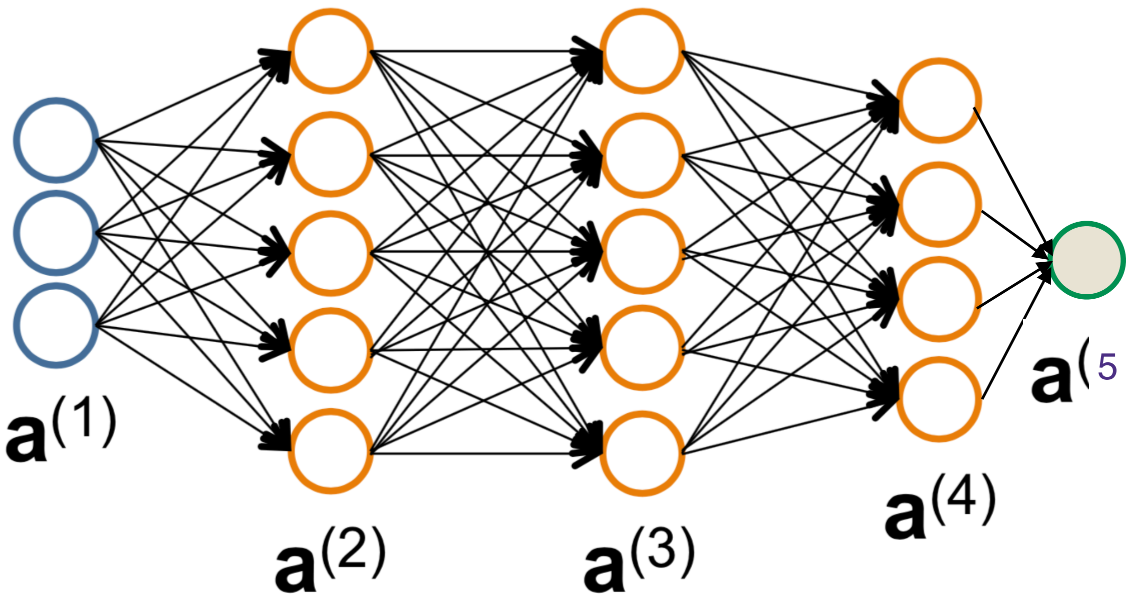
Structured Neural Networks

UNIVERSITY *of* WASHINGTON

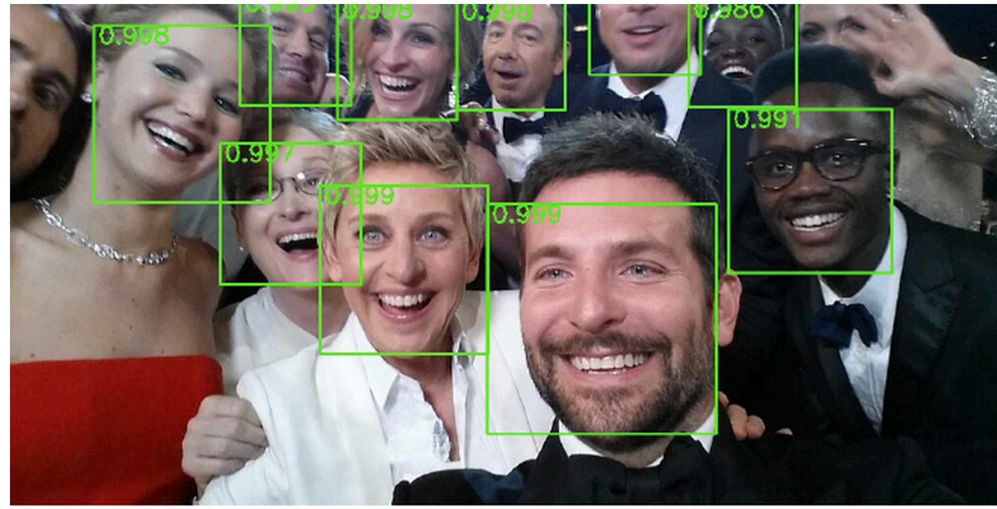
Neural Network Architecture

The neural network architecture is defined by the number of layers, and the number of nodes in each layer, but also by allowable edges.

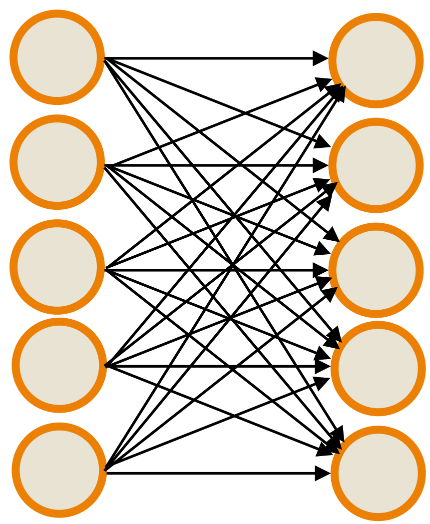


Neural Network Architecture

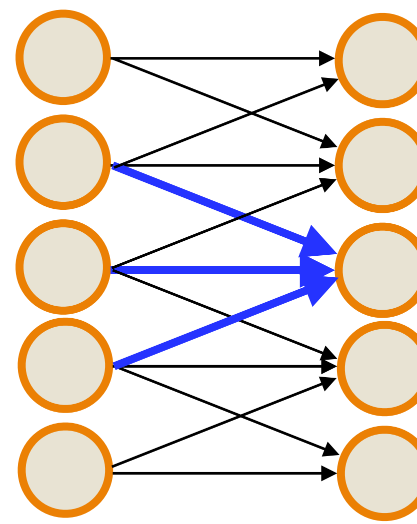
Objects are often localized in space so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.



Similarly, to identify edges or other local structure, it makes sense to only look at local information



vs.



Convolution of images (2d convolution)

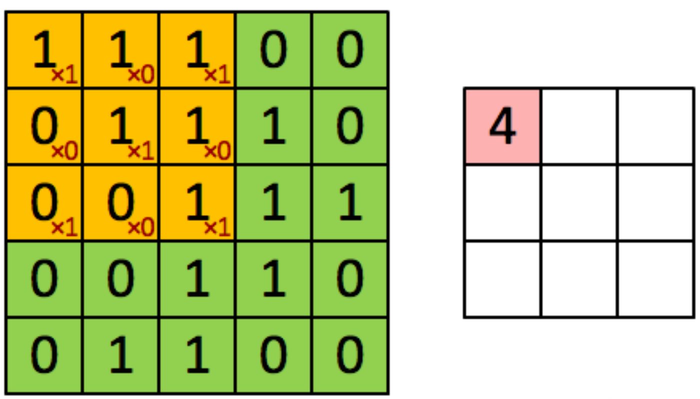
$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image I

1	0	1
0	1	0
1	0	1

Filter K

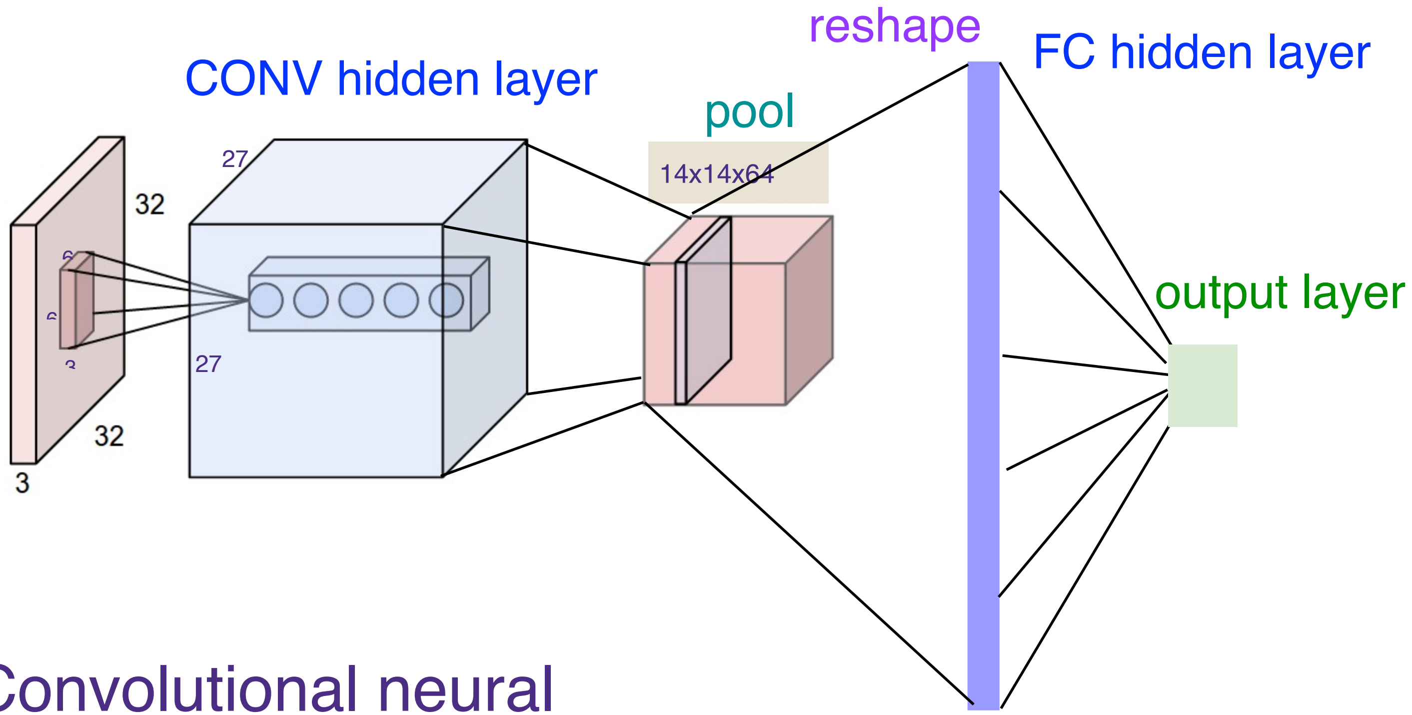


Image

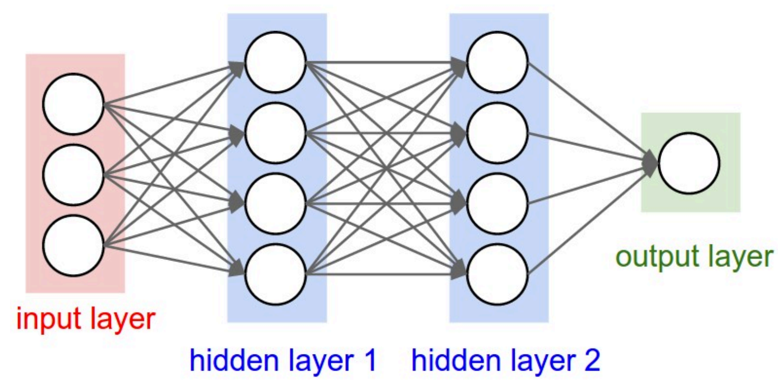
Convolved Feature

$I * K$

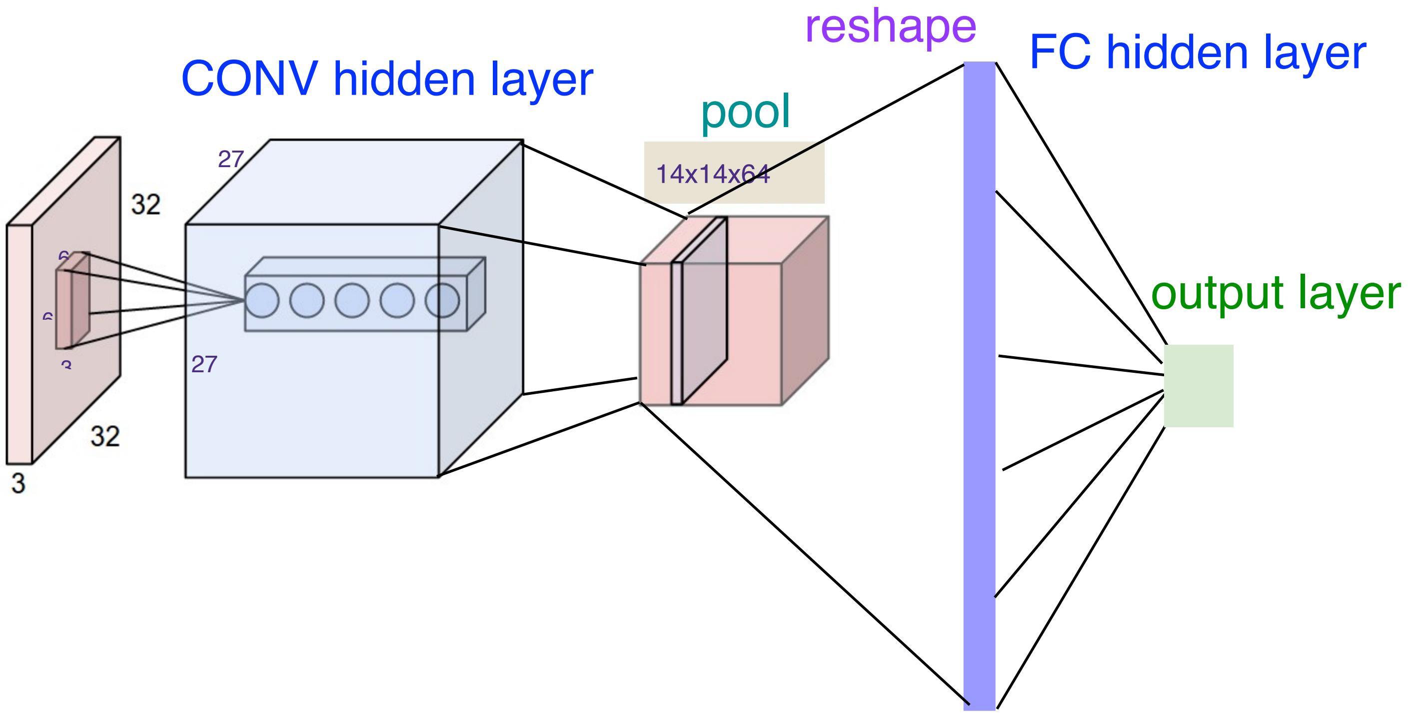
Learning Features with Convolutional Networks



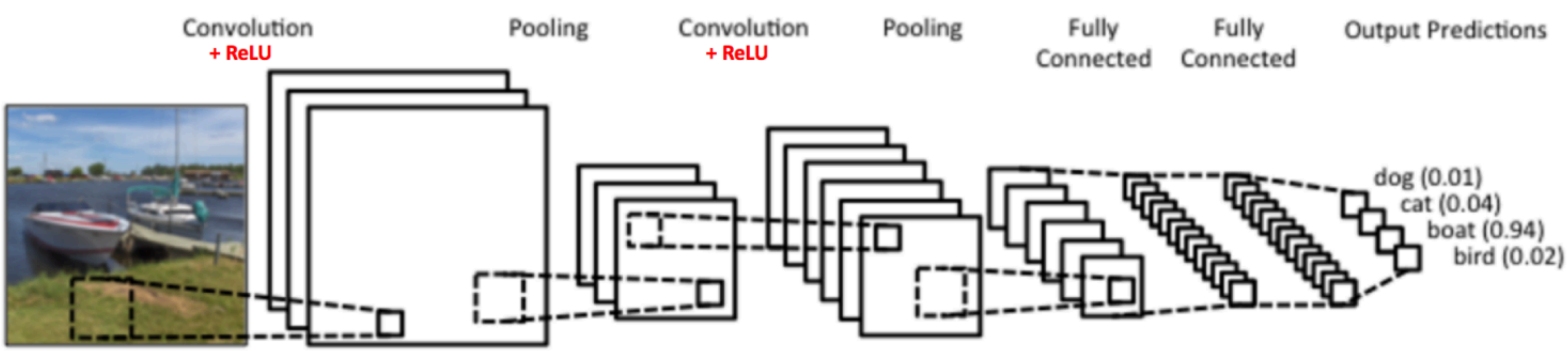
Recall: Convolutional neural networks (CNN) are just regular fully connected (FC) neural networks with some connections removed.
Train with SGD!

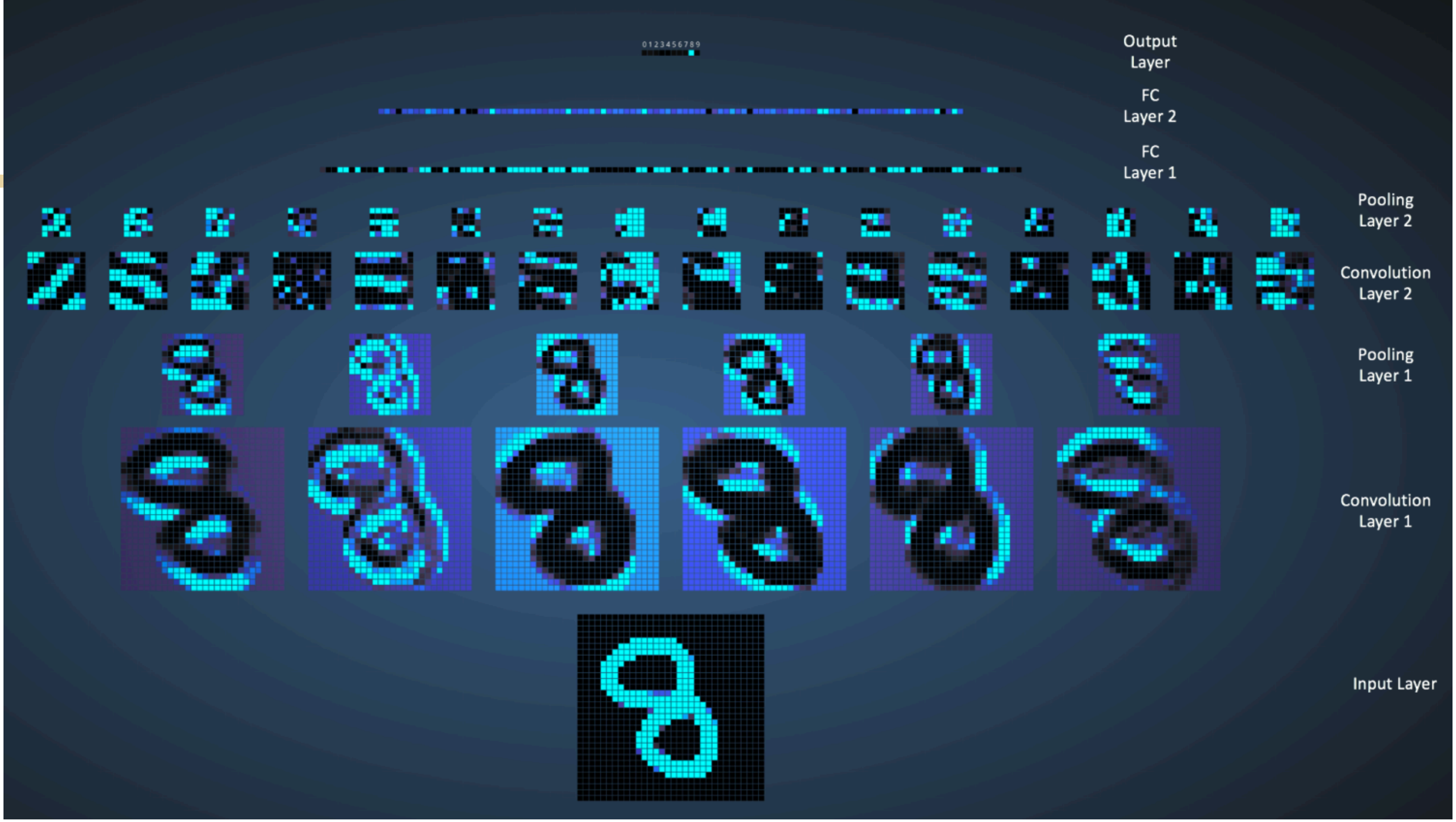


Training Convolutional Networks

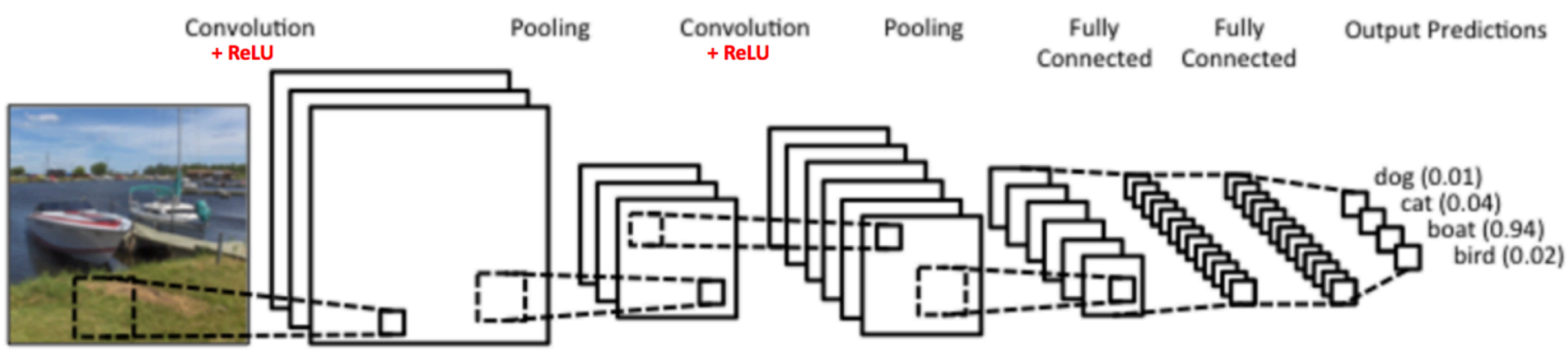


Real example network: LeNet





Real example network: LeNet



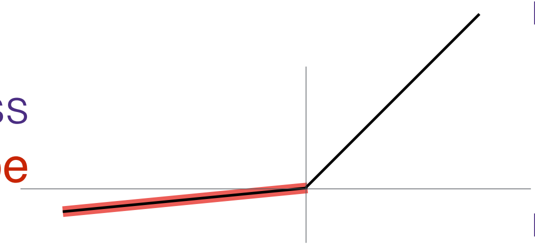
Real networks

Residual Network of [HeZhangRenSun'15]

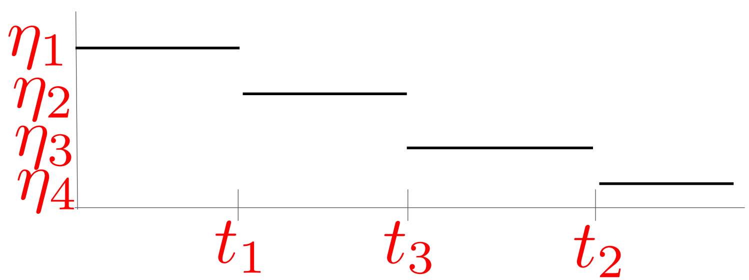
Modern networks have dozens of parameters to tune.

Data augmentation?
Batch norm?

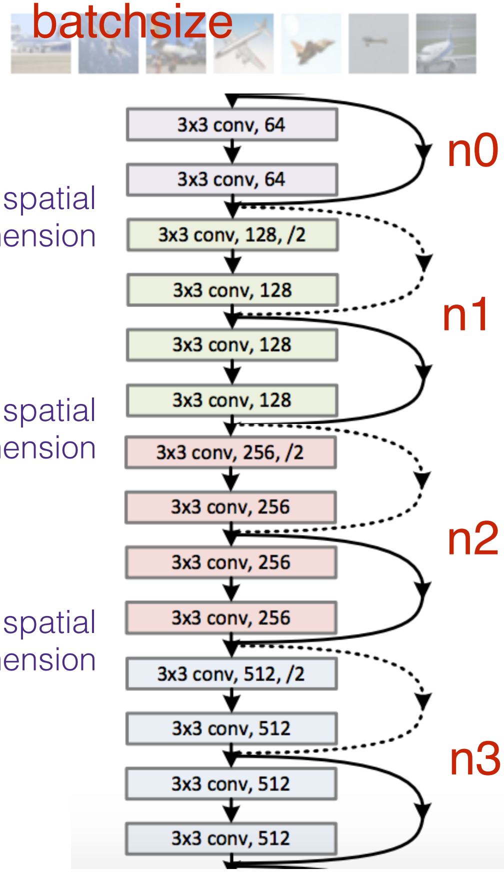
RELU leakiness
slope



Learning rate schedule



batchsize



Reduce spatial dimension

Reduce spatial dimension

Reduce spatial dimension

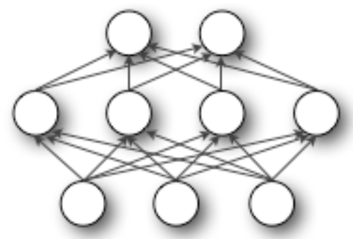
n0 layers of f0 filters

n1 layers of f1 filters

n2 layers of f2 filters

n3 layers of f3 filters

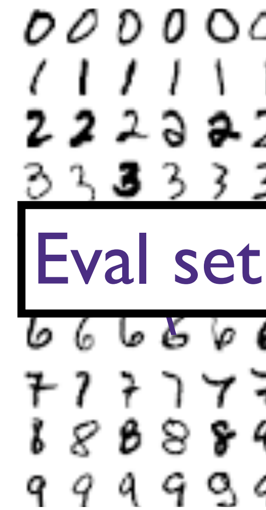
Hyperparameter Optimization



$N_{out} = 10$

N_{hid}

$N_{in} = 784$



Hyperparameters   Eval-loss

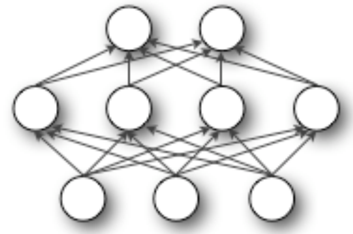
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

ℓ_2-penalty $\lambda \in [10^{-6}, 10^{-1}]$

\# hidden nodes $N_{hid} \in [10^1, 10^3]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$

$\widehat{f}$

Hyperparameters

Eval-loss

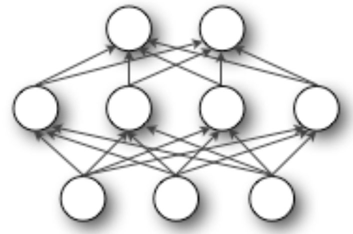
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

ℓ_2-penalty $\lambda \in [10^{-6}, 10^{-1}]$

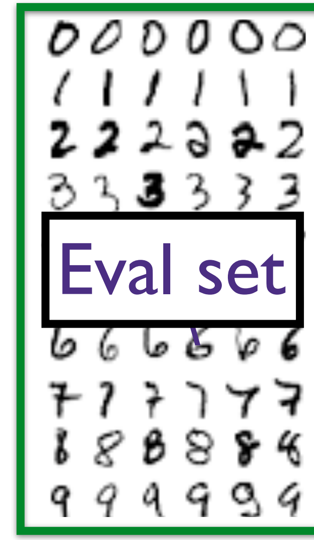
$\#$ hidden nodes $N_{hid} \in [10^1, 10^3]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$



$\widehat{f}$

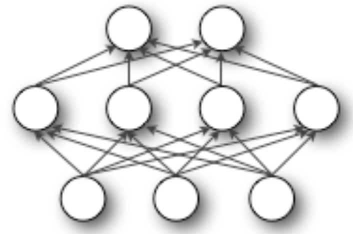
Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

ℓ_2-penalty $\lambda \in [10^{-6}, 10^{-1}]$

hidden nodes $N_{hid} \in [10^1, 10^3]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$

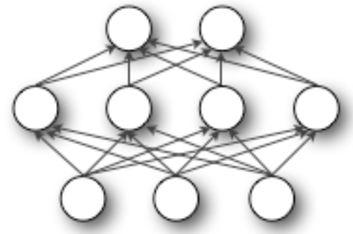
Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	0.1196
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$	0.0834
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$	0.0242
$(10^{-1.8}, 10^{-5.6}, 10^{1.7})$	0.029

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

ℓ_2-penalty $\lambda \in [10^{-6}, 10^{-1}]$

hidden nodes $N_{hid} \in [10^1, 10^3]$



$N_{out} = 10$

N_{hid}

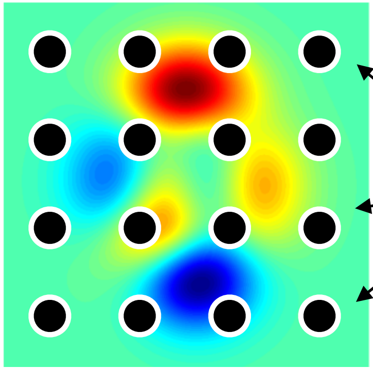
$N_{in} = 784$

Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	0.1196
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$	0.0834
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$	0.0242
$(10^{-1.8}, 10^{-5.6}, 10^{1.7})$	0.029

How do we choose hyperparameters to train and evaluate?

How do we choose hyperparameters to train and evaluate?

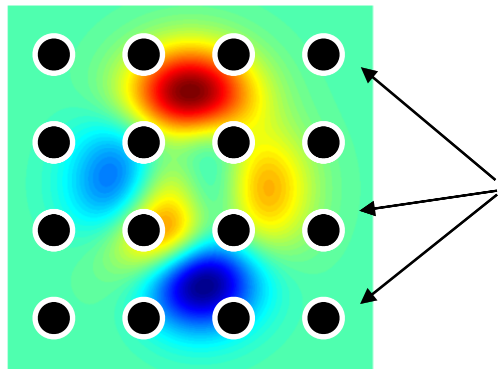
Grid search:



Hyperparameters on 2d uniform grid

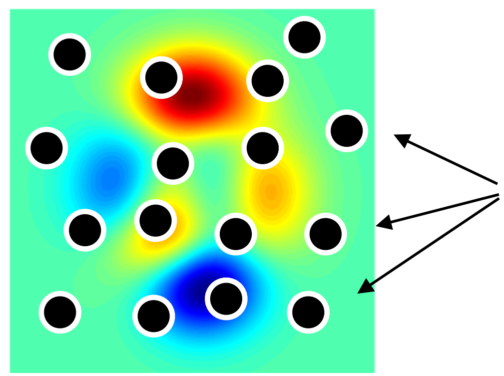
How do we choose hyperparameters to train and evaluate?

Grid search:



Hyperparameters on 2d uniform grid

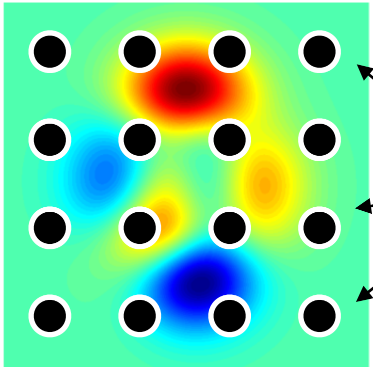
Random search:



Hyperparameters randomly chosen

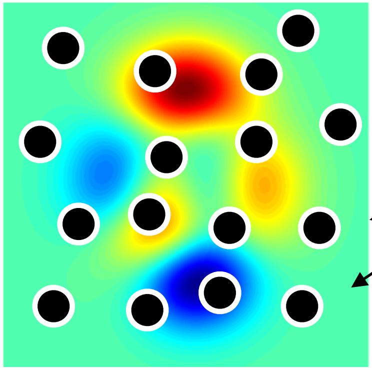
How do we choose hyperparameters to train and evaluate?

Grid search:



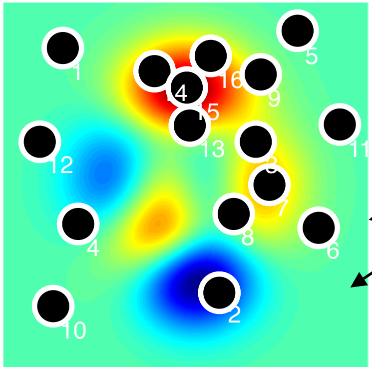
Hyperparameters on 2d uniform grid

Random search:



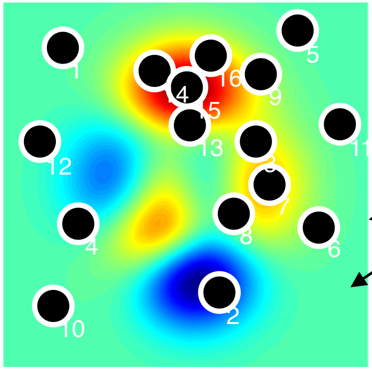
Hyperparameters randomly chosen

Black-box Optimization:



Hyperparameters ***adaptively*** chosen

Black-Box Optimization:



Hyperparameters ***adaptively*** chosen

How does it work?

Recent work attempts to speed up hyperparameter evaluation by stopping poor performing settings before they are fully trained.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. arXiv:1406.3896, 2014.

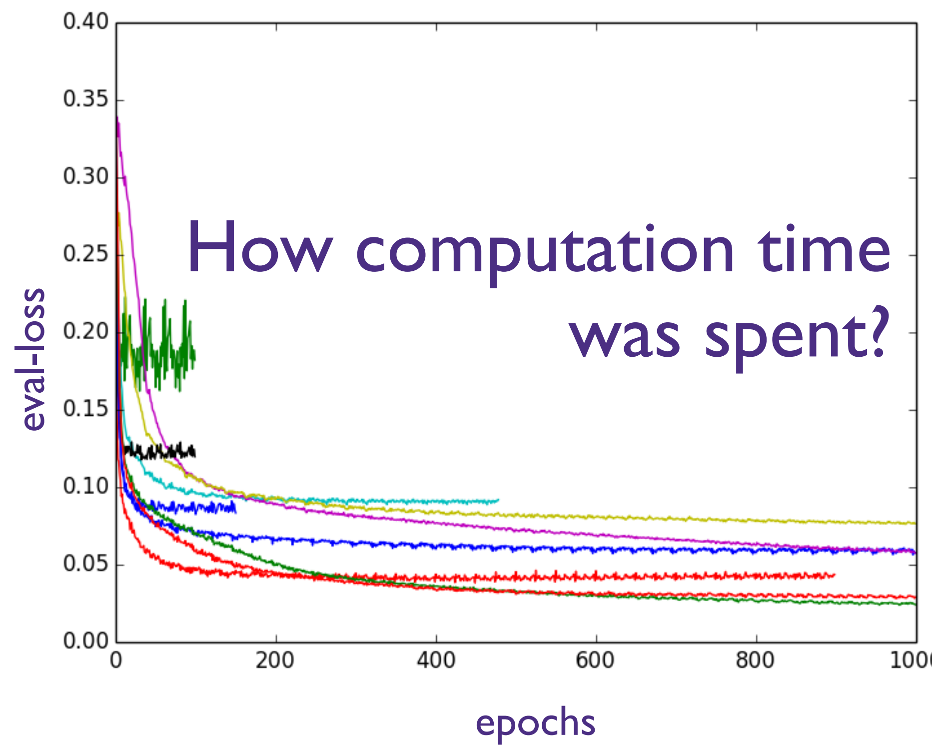
Alekh Agarwal, Peter Bartlett, and John Duchi. Oracle inequalities for computationally adaptive model selection. COLT, 2012.

Domhan, T., Springenberg, J. T., and Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, 2015.

András György and Levente Kocsis. Efficient multi-start strategies for local search algorithms. JAIR, 41, 2011.

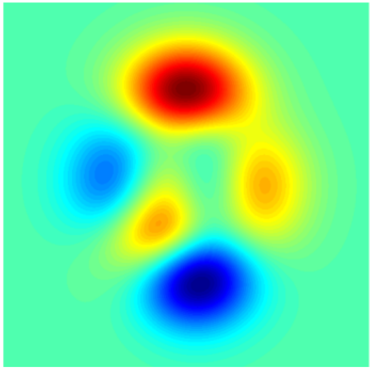
Li, Jamieson, DeSalvo, Rostamizadeh, Talwalkar. Hyperband: Hyperband: A Novel Bandit-Based Approach to Hyperparameter. JMLR 2018.

Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	0.1196
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$	0.0834
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$	0.0242
$(10^{-1.8}, 10^{-5.6}, 10^{1.7})$	0.029



Hyperparameter Optimization

In general, hyperparameter optimization is non-convex optimization and little is known about the underlying function (only observe validation loss)



Your time is valuable, computers are cheap:
Do not employ "grad student descent" for hyper parameter search.
Write modular code that takes parameters as input and automate this embarrassingly parallel search.

Tools for different purposes:

- Very few evaluations: use random search (and pray) or be *clever*
- Few evaluations and long-running computations: see refs on last slide
- Moderate number of evaluations (but still exp(#params)) and high accuracy needed: use Bayesian Optimization
- Many evaluations possible: use random search. Why overthink it?

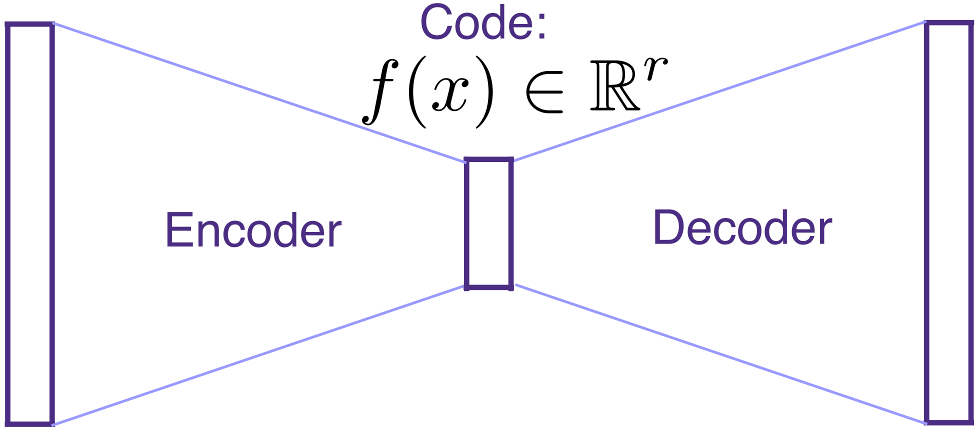
Unsupervised Learning revisited

Autoencoders

Find a low dimensional representation for your data by predicting your data

Code:
$f(x) \in \mathbb{R}^r$

Input:
$x \in \mathbb{R}^d$



Output:
$\widehat{x} = g(f(x)) \in \mathbb{R}^d$

$$\underset{f,g}{\text{minimize}} \sum_{i=1}^{n} \|x_i - g(f(x_i))\|_2^2$$

Three major applications
1. De-noising
2. Feature extraction
3. Manifold learning

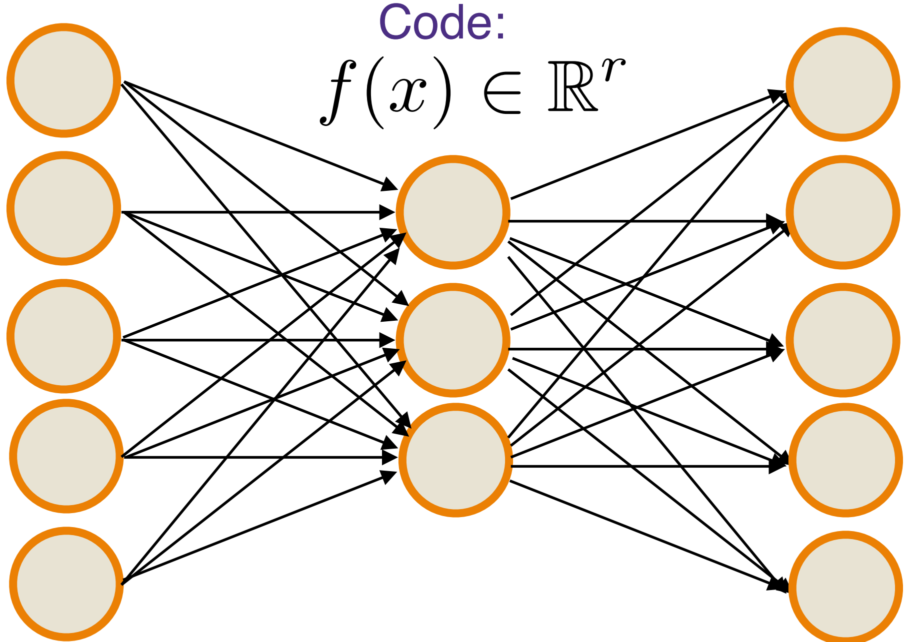
Autoencoders

Code:

$f(x) \in \mathbb{R}^r$

Input:

$x \in \mathbb{R}^d$



Output:

$\widehat{x} = g(f(x)) \in \mathbb{R}^d$

$$\underset{f,g}{\text{minimize}} \sum_{i=1}^{n} \|x_i - g(f(x_i))\|_2^2$$

What if $f(X) = Ax$ and $g(y) = By$?

Generative models

Related application: Generating new samples (see variational autoencoders (VAE) or generative adversarial networks (GAN)

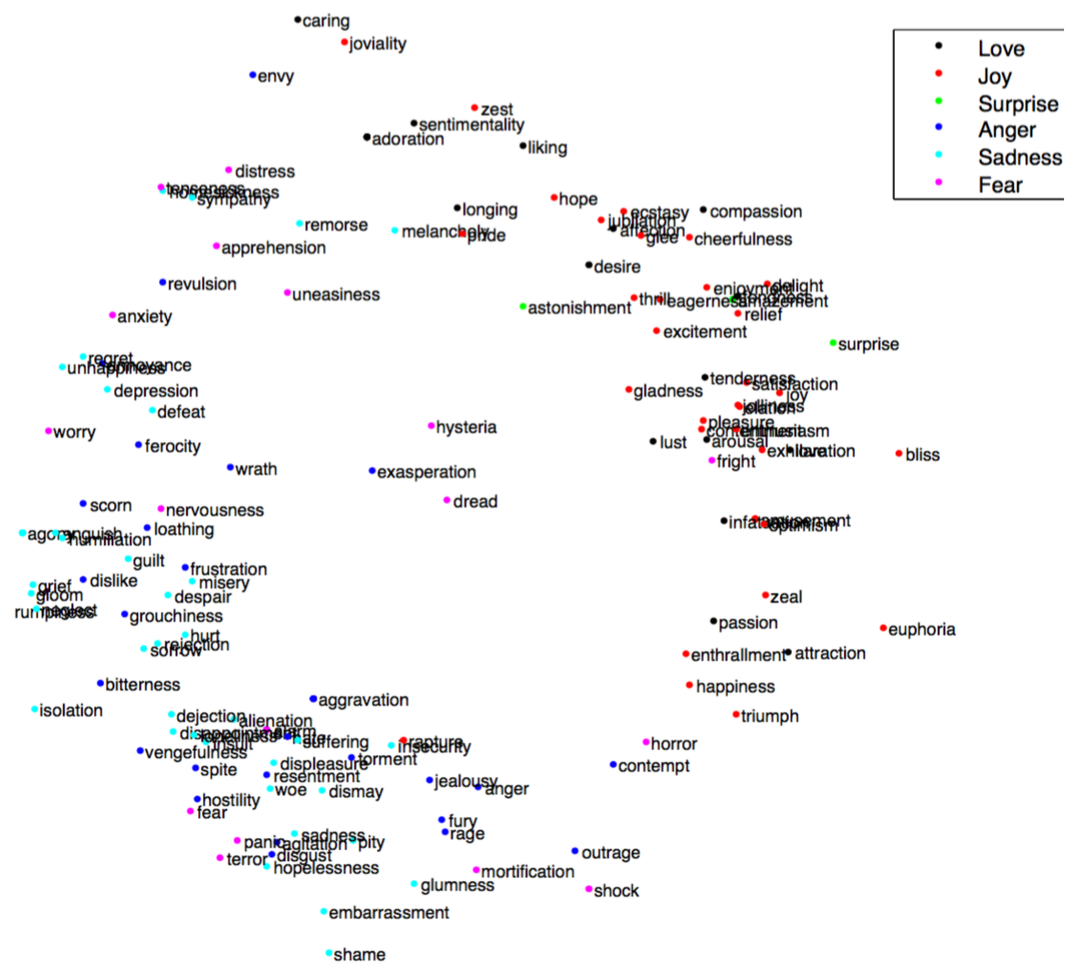
Basic Text Modeling

Word embeddings, word2vec

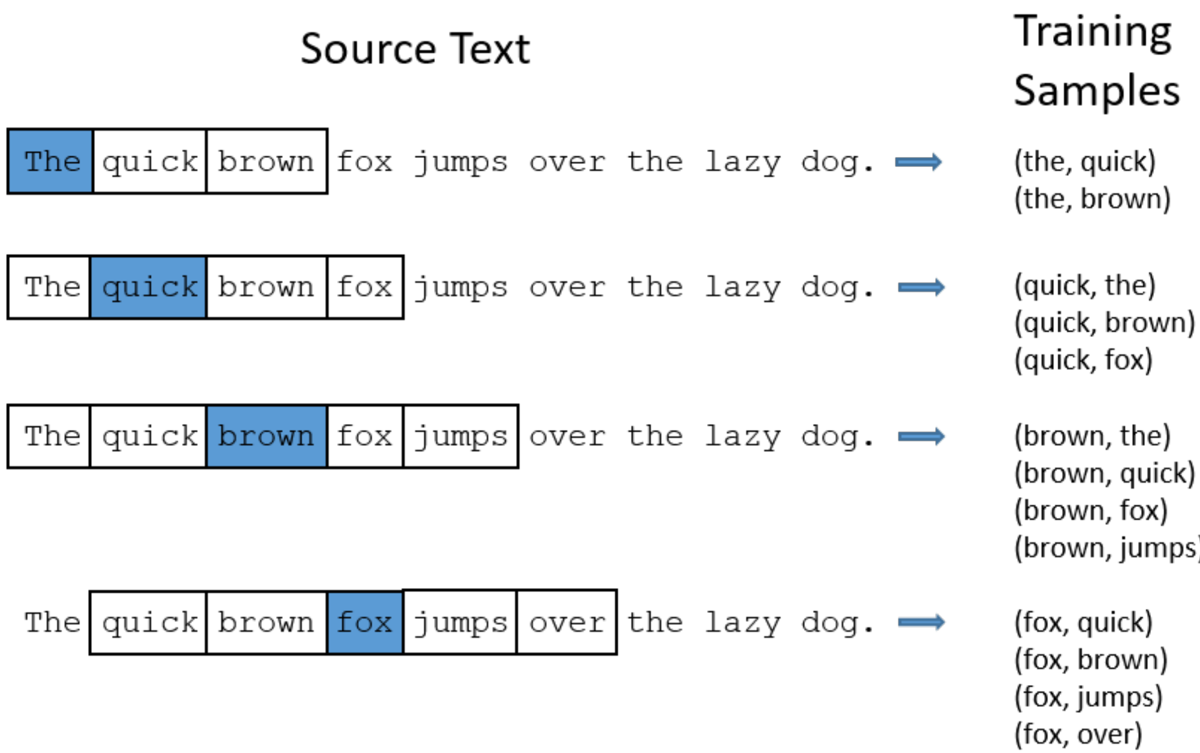
Can we **embed words** into a latent space?

This embedding came from directly querying for relationships.

word2vec is a popular unsupervised learning approach that just uses a text corpus (e.g. nytimes.com)

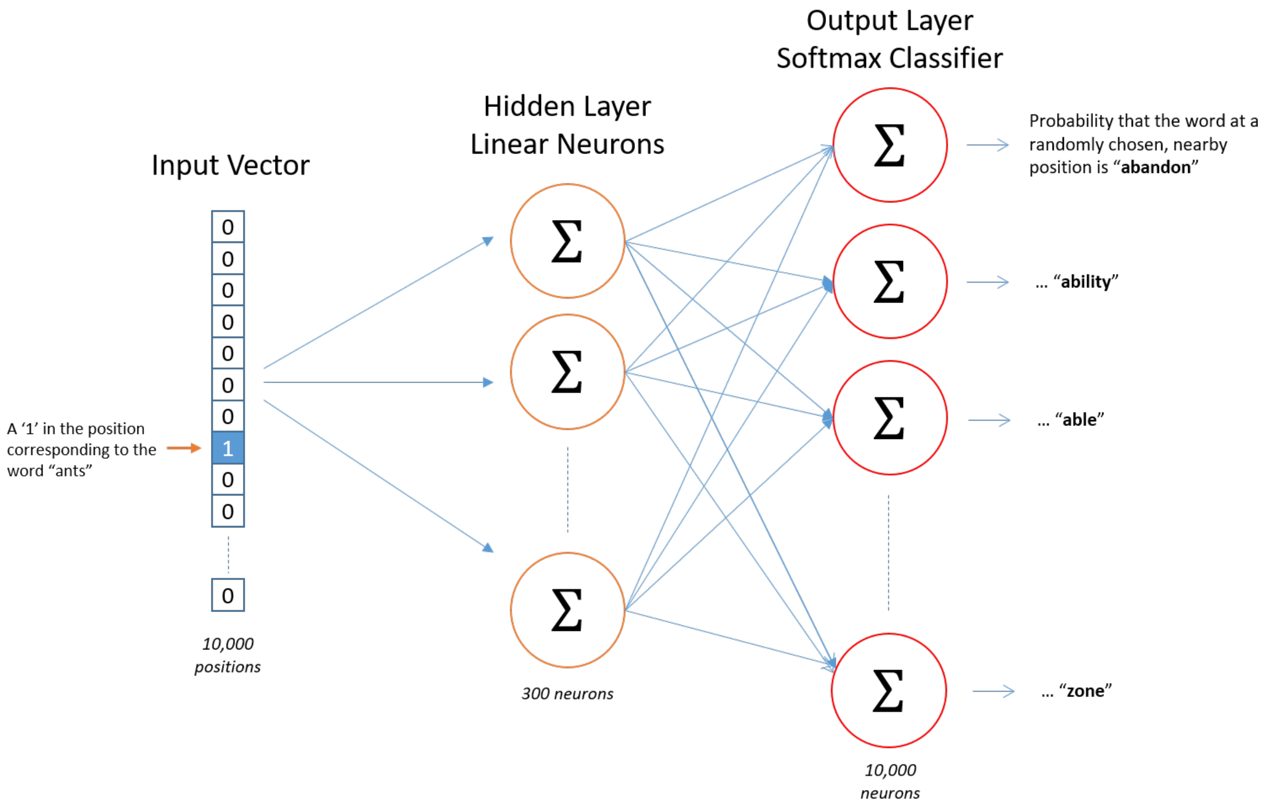


Word embeddings, word2vec



slide: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

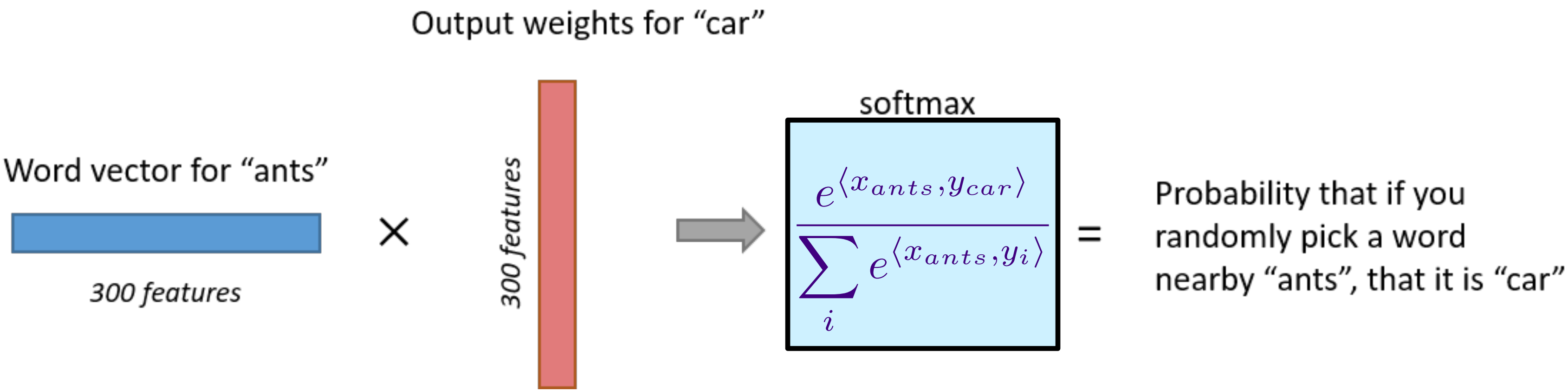
Word embeddings, word2vec



Training neural network to predict co-occuring words. Use first layer weights as embedding, throw out output layer

slide: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

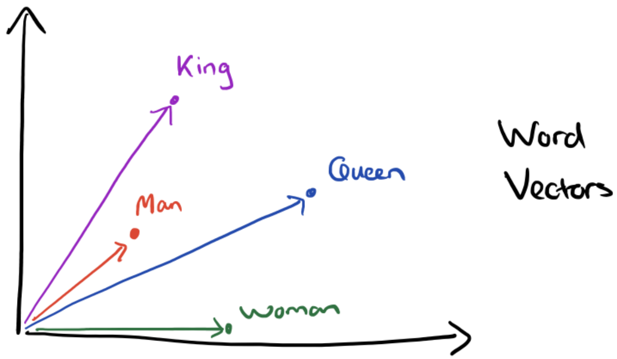
Word embeddings, word2vec



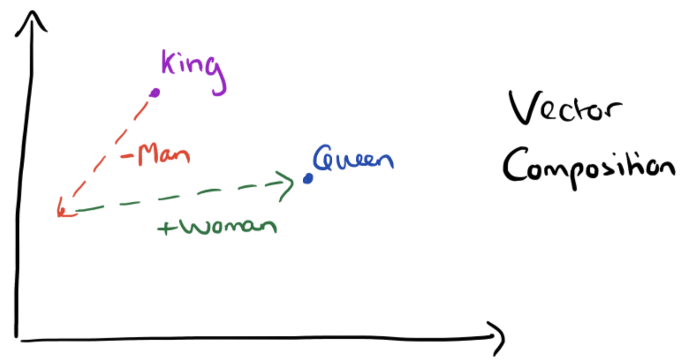
Training neural network to predict co-occuring words. Use first layer weights as embedding, throw out output layer

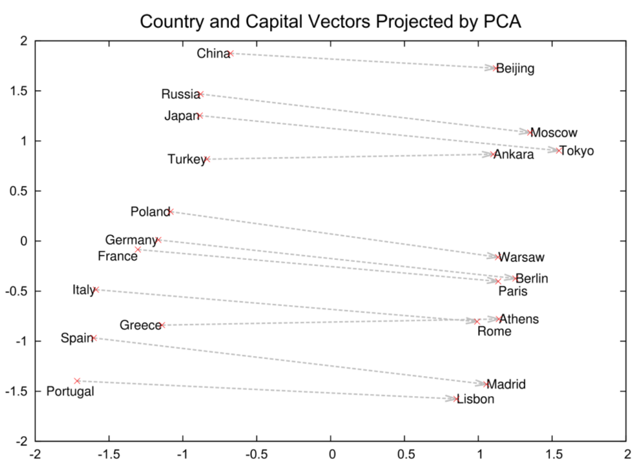
slide: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

word2vec outputs



king - man + woman = queen

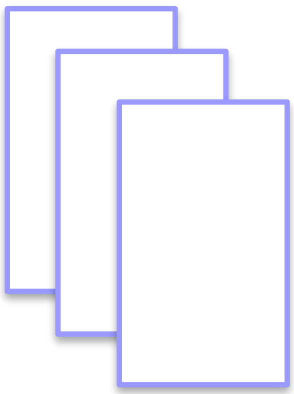




country - capital

slide: https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

Bag of Words



n documents/articles with lots of text

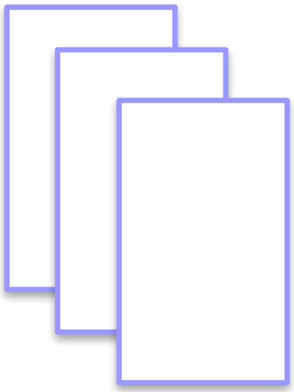
Questions:
- How to get a feature representation of each article?
- How to cluster documents into topics?

Bag of words model:

*i*th document: $x_i \in \mathbb{R}^D$

$x_{i,j} =$ proportion of times jth word occurred in ith document

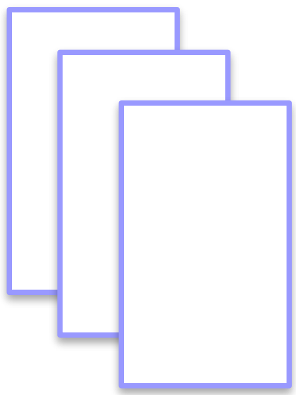
Bag of Words



n documents/articles with lots of text

- **Can we embed each document into a feature space?**

Bag of Words



n documents/articles with lots of text

- **Can we embed each document into a feature space?**

Bag of words model:

i*th* document: $x_i \in \mathbb{R}^D$

$$x_{i,j} = \text{ proportion of times } j\text{th word occurred in } i\text{th document}$$

Given vectors, run k-means or Gaussian mixture model to find k clusters/topics

Nonnegative matrix factorization (NMF)

$A \in \mathbb{R}^{m \times n}$ $\quad A_{i,j} = \text{frequency of } j\text{th word in document } i$

Nonnegative Matrix factorization: $\min_{W \in \mathbb{R}_+^{m \times d}, H \in \mathbb{R}_+^{n \times d}} \|A - WH^T\|_F^2$

d is number of topics

Also see latent Dirichlet factorization (LDA)

Nonnegative matrix factorization (NMF)

$A \in \mathbb{R}^{m \times n}$ $\quad A_{i,j} = \text{frequency of } j\text{th word in document } i$

Nonnegative Matrix factorization:

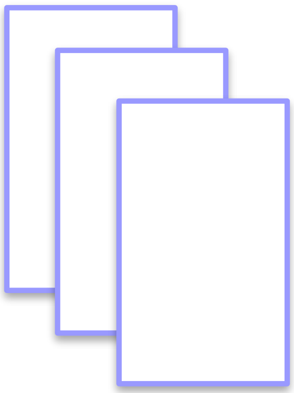
$$\min_{W \in \mathbb{R}_+^{m \times d}, H \in \mathbb{R}_+^{n \times d}} \|A - WH^T\|_F^2$$

d is number of topics

Each column of H represents a cluster of a topic,
Each row W is some weights a combination of topics

Also see latent Dirichlet factorization (LDA)

TF*IDF



n documents/articles with lots of text

How to get a feature representation of each article?

1. For each document *d* compute the proportion of times word *t* occurs out of all words in *d*, i.e. **term frequency**

$$TF_{d,t}$$

2. For each word *t* in your corpus, compute the proportion of documents out of *n* that the word *t* occurs, i.e., **document frequency**

$$DF_t$$

3. Compute score for word *t* in document *d* as $TF_{d,t} \log(\frac{1}{DF_t})$

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$

ratebeer

Two Hearted Ale - Input ~2500 natural language reviews

http://www.ratebeer.com/beer/two-hearted-ale/1502/2/1/

3.8 AROMA 8/10 APPEARANCE 4/5 TASTE 8/10 PALATE 3/5 OVERALL 15/20
fonefan (25678) - VestJylland, DENMARK - JAN 18, 2009

Bottle 355ml.
Clear light to medium yellow orange color with a average, frothy, good lacing, fully lasting, off-white head. Aroma is moderate to heavy malty, moderate to heavy hoppy, perfume, grapefruit, orange shell, soap. Flavor is moderate to heavy sweet and bitter with a average to long duration. Body is medium, texture is oily, carbonation is soft. [250908]

4 AROMA 8/10 APPEARANCE 4/5 TASTE 7/10 PALATE 4/5 OVERALL 17/20
Ungstrup (24358) - Oamaru, NEW ZEALAND - MAR 31, 2005

An orange beer with a huge off-white head. The aroma is sweet and very freshly hoppy with notes of hop oils - very powerful aroma. The flavor is sweet and quite hoppy, that gives flavors of oranges, flowers as well as hints of grapefruit. Very refreshing yet with a powerful body.

Reviews for each beer	Bag of Words weighted by TF*IDF	Get 100 nearest neighbors using cosine distance	Non-metric multidimensional scaling	Embedding in d dimensions

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$

Two Hearted Ale - Weighted Bag of Words:

Reviews for each beer

Bag of Words weighted by TF*IDF

Get 100 nearest neighbors using cosine distance

Non-metric multidimensional scaling

Embedding in d dimensions

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$

Weighted count vector for the ith beer:

$$z_i \in \mathbb{R}^{400{,}000}$$

Cosine distance:

$$d(z_i, z_j) = 1 - \frac{z_i^T z_j}{||z_i|| \, ||z_j||}$$

Two Hearted Ale - Nearest Neighbors:

Bear Republic Racer 5
Avery IPA
Stone India Pale Ale (IPA)
Founders Centennial IPA
Smuttynose IPA
Anderson Valley Hop Ottin IPA
AleSmith IPA
BridgePort IPA
Boulder Beer Mojo IPA
Goose Island India Pale Ale
Great Divide Titan IPA
New Holland Mad Hatter Ale
Lagunitas India Pale Ale
Heavy Seas Loose Cannon Hop3
Sweetwater IPA

Reviews for each beer

Bag of Words weighted by TF*IDF

Get 100 nearest neighbors using cosine distance

Non-metric multidimensional scaling

Embedding in d dimensions

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$

Find an embedding $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ such that

$||x_k - x_i|| < ||x_k - x_j||$ whenever $\underline{d(z_k, z_i)} < \underline{d(z_k, z_j)}$

for all 100-nearest neighbors.

(10^7 constraints, 10^5 variables)

distance in 400,000 dimensional "word space"

Solve with hinge loss and stochastic gradient descent.

(20 minutes on my laptop) (d=2,err=6%) (d=3,err=4%)

Could have also used local-linear-embedding, max-volume-unfolding, kernel-PCA, etc.

Reviews for each beer

Bag of Words weighted by TF*IDF

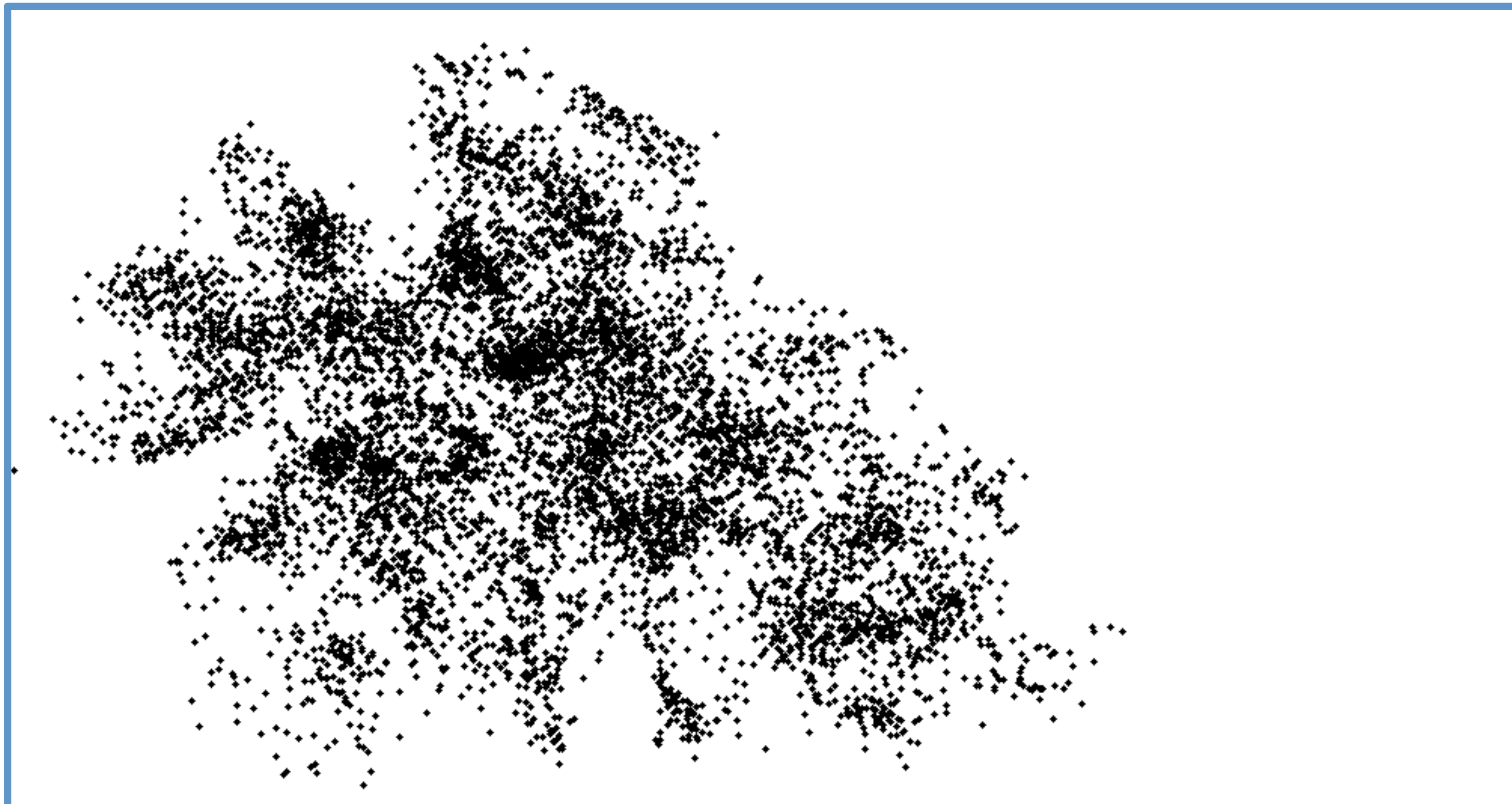
Get 100 nearest neighbors using cosine distance

Non-metric multidimensional scaling

Embedding in d dimensions

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$



Reviews for each beer

Bag of Words weighted by TF*IDF

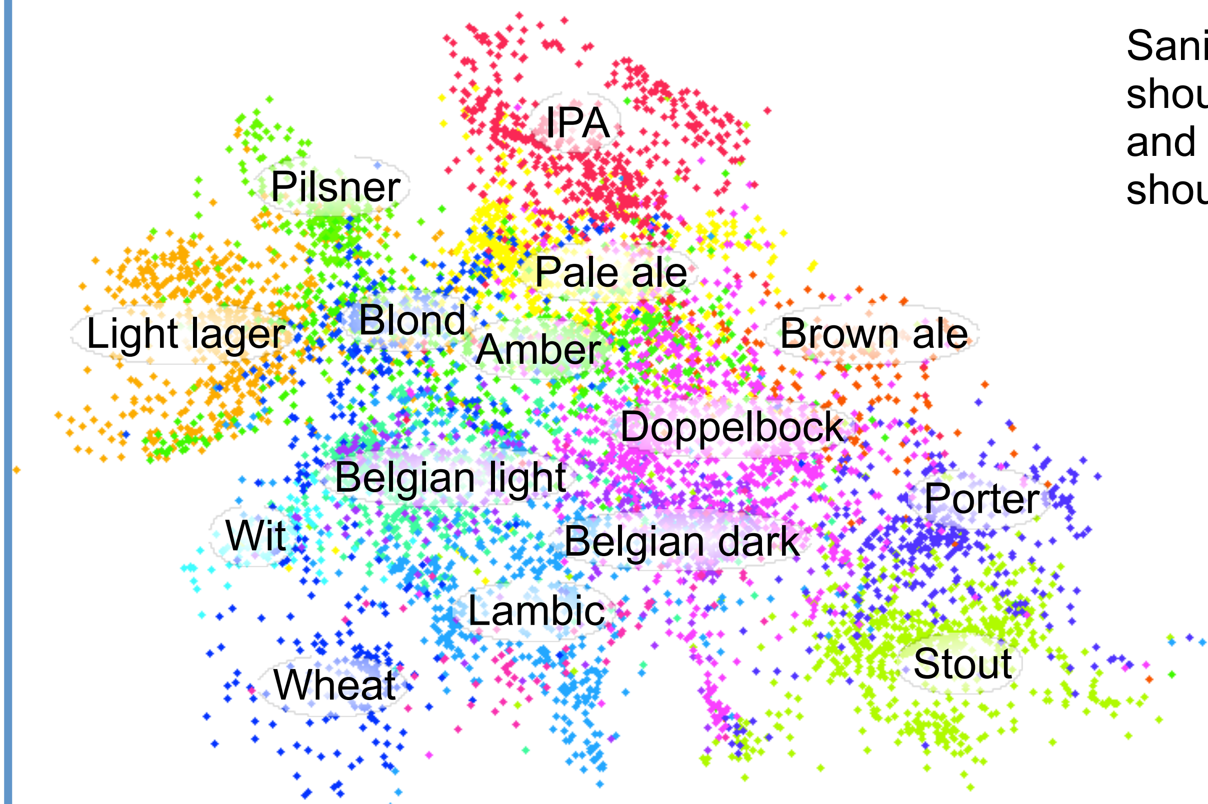
Get 100 nearest neighbors using cosine distance

Non-metric multidimensional scaling

Embedding in d dimensions

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$

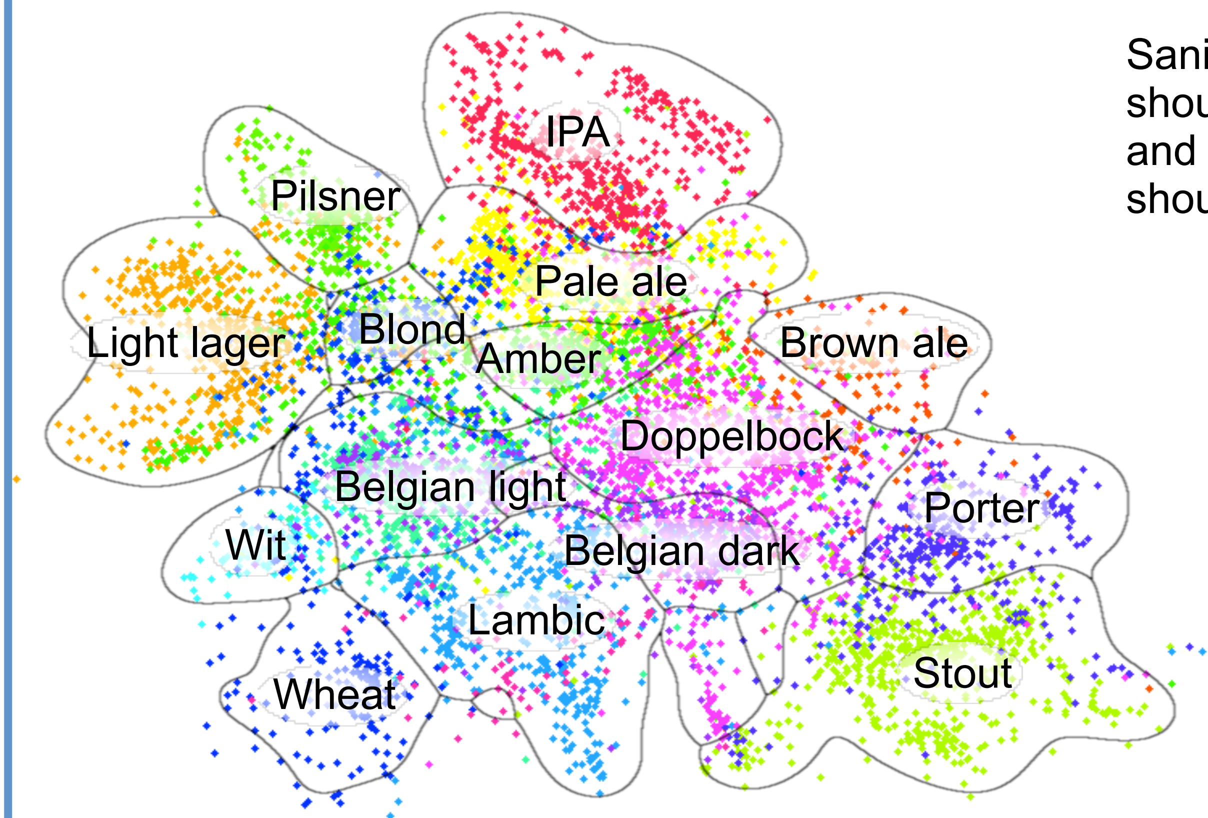


Sanity check: styles should cluster together and similar styles should be close.

Reviews for each beer → Bag of Words weighted by TF*IDF → Get 100 nearest neighbors using cosine distance → Non-metric multidimensional scaling → Embedding in d dimensions

BeerMapper - Under the Hood

Algorithm requires feature representations of the beers $\{x_1, \ldots, x_n\} \subset \mathbb{R}^d$



Sanity check: styles should cluster together and similar styles should be close.

Reviews for each beer → Bag of Words weighted by TF*IDF → Get 100 nearest neighbors using cosine distance → Non-metric multidimensional scaling → Embedding in d dimensions

Sequences and Recurrent Neural Networks

UNIVERSITY of WASHINGTON

Time-dependent data

$x_t \in \mathbb{R}$: AAPL stock price at time t

Prediction model: $p(x_{t+1} | x_t, x_{t-1}, x_{t-2}, \ldots)$

Time-dependent data

$x_t \in \mathbb{R}$: AAPL stock price at time t

$h_t \in \mathbb{R}^d$: hidden latent state of AAPL

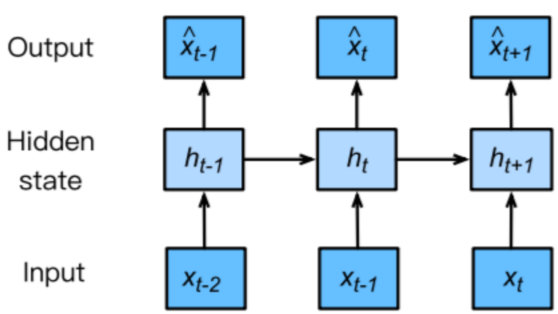
$$\text{Prediction model: } p(x_{t+1} | x_t, x_{t-1}, x_{t-2}, \dots) \approx p(x_{t+1} | x_t, h_{t+1})$$

Time-dependent data

$x_t \in \mathbb{R}$: AAPL stock price at time t

$h_t \in \mathbb{R}^d$: hidden latent state of AAPL

$$\text{Prediction model: } p(x_{t+1}|x_t, x_{t-1}, x_{t-2}, \ldots)$$
$$\approx p(x_{t+1}|x_t, h_{t+1})$$



$$h_{t+1} = g(h_t, x_t)$$

Hidden state and g never observed, but learned!

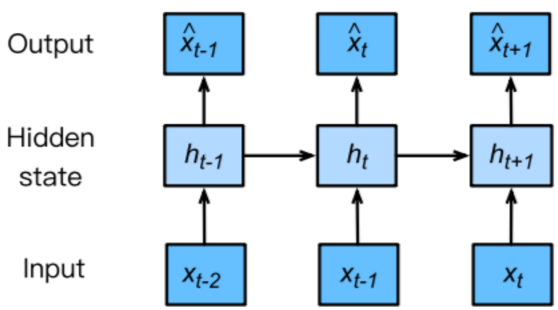
Zhang et al. “Dive into Deep Learning”

Time-dependent data

$x_t \in \mathbb{R}$: AAPL stock price at time t

$h_t \in \mathbb{R}^d$: hidden latent state of AAPL

$$\text{Prediction model: } p(x_{t+1} | x_t, x_{t-1}, x_{t-2}, \dots)$$
$$\approx p(x_{t+1} | x_t, h_{t+1})$$



$$h_{t+1} = g(h_t, x_t)$$

Hidden state and g never observed, but learned!

Explicit:

$$h_{t+1} = \sigma(Ah_t + Bx_t)$$
$$\widehat{x}_{t+1} = Ch_{t+1} + Dx_t$$

$$\sum_t (x_t - \widehat{x}_t)^2$$

Zhang et al. “Dive into Deep Learning”

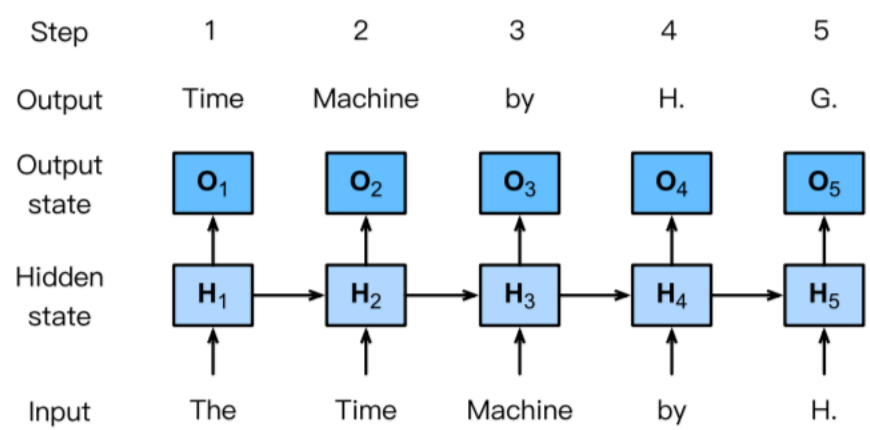
Time-dependent data

$$\text{Prediction model: } p(x_{t+1}|x_t, x_{t-1}, x_{t-2}, \dots) \approx p(x_{t+1}|x_t, h_{t+1})$$

$$h_{t+1} = g(h_t, x_t)$$

Hidden state and g never observed, but learned!

Model also works with text!



Zhang et al. “Dive into Deep Learning”

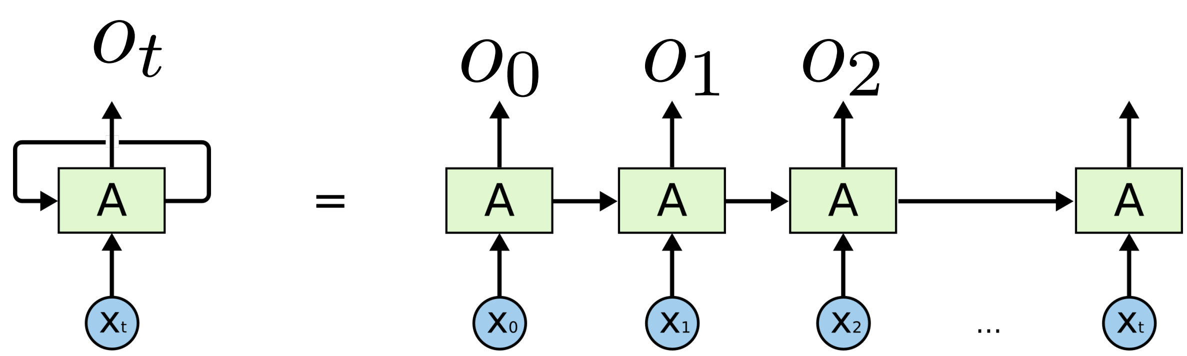
Time-dependent data

$$\text{Prediction model: } p(x_{t+1}|x_t, x_{t-1}, x_{t-2}, \dots)$$
$$\approx p(x_{t+1}|x_t, h_{t+1})$$

$$h_{t+1} = g(h_t, x_t)$$

Hidden state and g never observed, but learned!

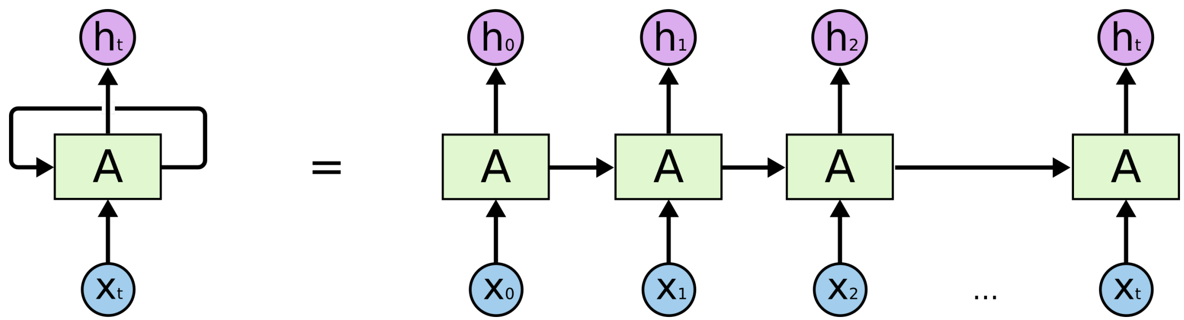
Recurrent Neural Network



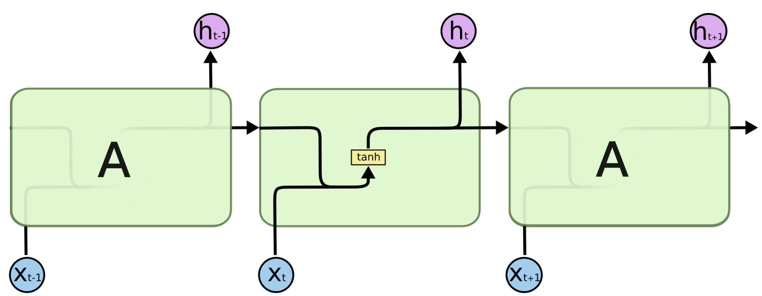
Slide: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Variable length sequences

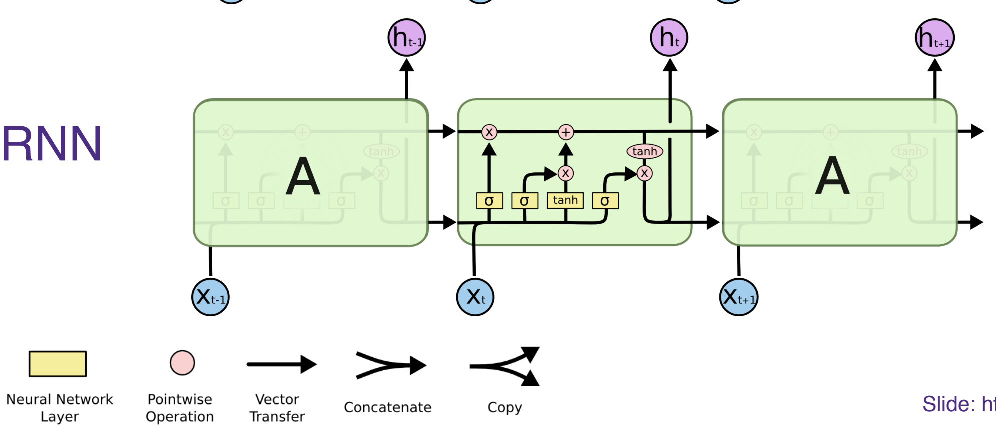
Recurrent Neural Network



Standard RNN



Gated RNN
LSTM



And MORE!
- Transformers
- Attention
- Bi-directional

Slide: http://colah.github.io/posts/2015-08-Understanding-LSTMs/