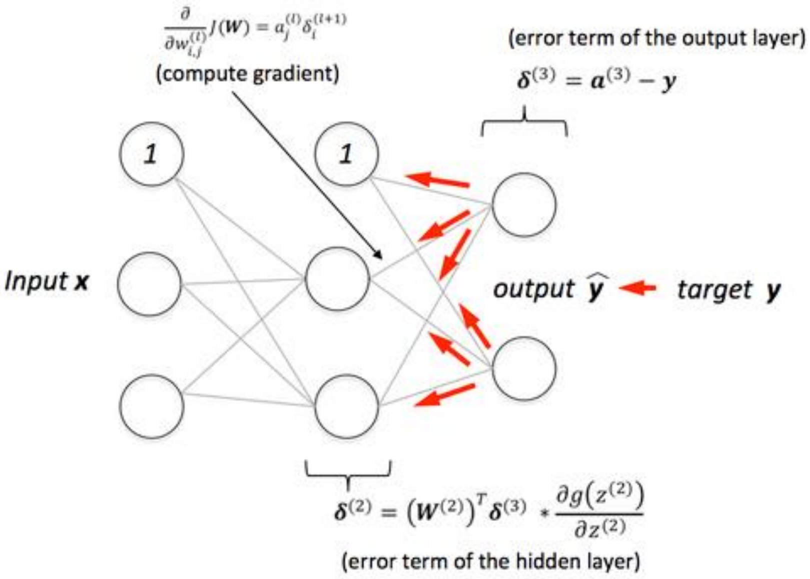
Inspire…Educate…Transform.

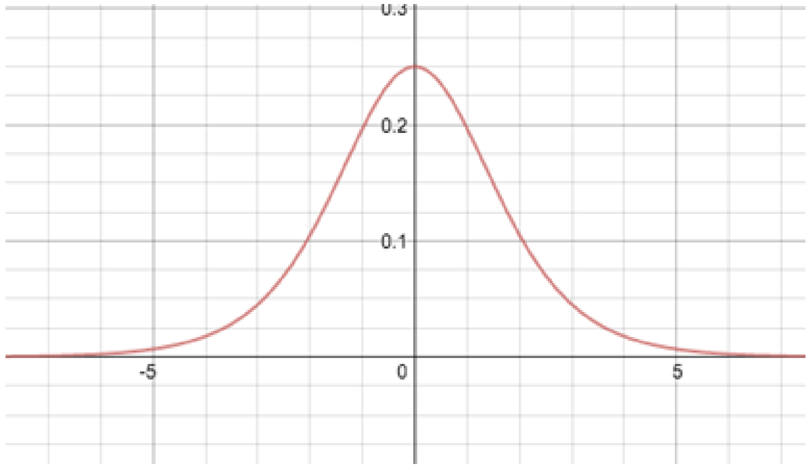
Deep MLPs, Deep Unsupervised nets, Embeddings

Dr. K. V. Dakshinamurthy

President, International School of Engineering

Issues with deep learning: Vanishing gradients

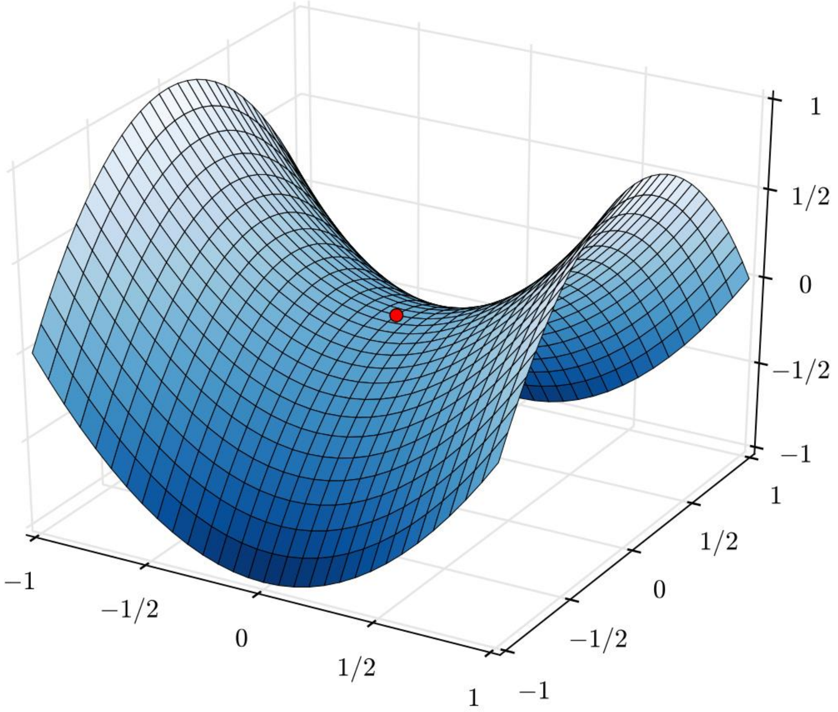

$\frac{\partial}{\partial w_{i,j}^{(l)}} J(W) = a_j^{(l)} \delta_i^{(l+1)}$
(compute gradient)
(error term of the output layer)
$\delta^{(3)} = a^{(3)} - y$
1
1
Input x
output $\hat{y}$
target y
$\delta^{(2)} = (W^{(2)})^T \delta^{(3)} * \frac{\partial g(z^{(2)})}{\partial z^{(2)}}$
(error term of the hidden layer)



Derivative of a sigmoid

Overfitting

- As we carefully chose loss functions, local minima is not a big issue
- But, high dimensional spaces are filled with saddle points
- Vanishing gradients coupled with overfitting to local minima or saddle points is a deadly combo!



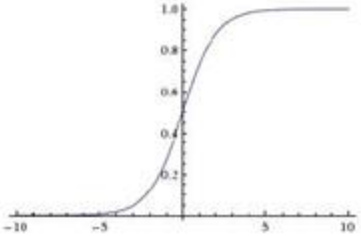
Multiple approaches

Improve training	Activation function, initialization, pre-processing, gradient descent, learning rate
Further accuracy	Ensembles
Minimize overfit	Regularization

Activation function

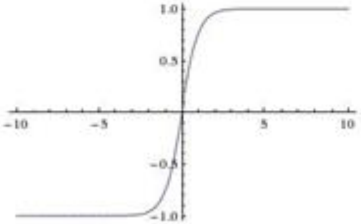
Sigmoid

$\sigma(x) = 1/(1 + e^{-x})$



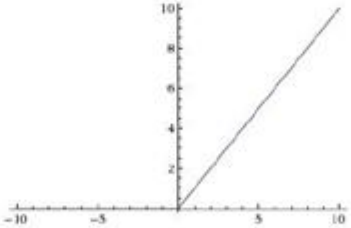
tanh

tanh(x)



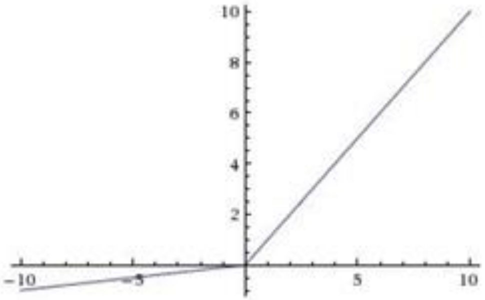
ReLU

max(0,x)



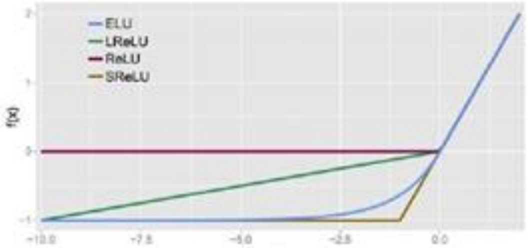
Leaky ReLU

max(0.1x, x)



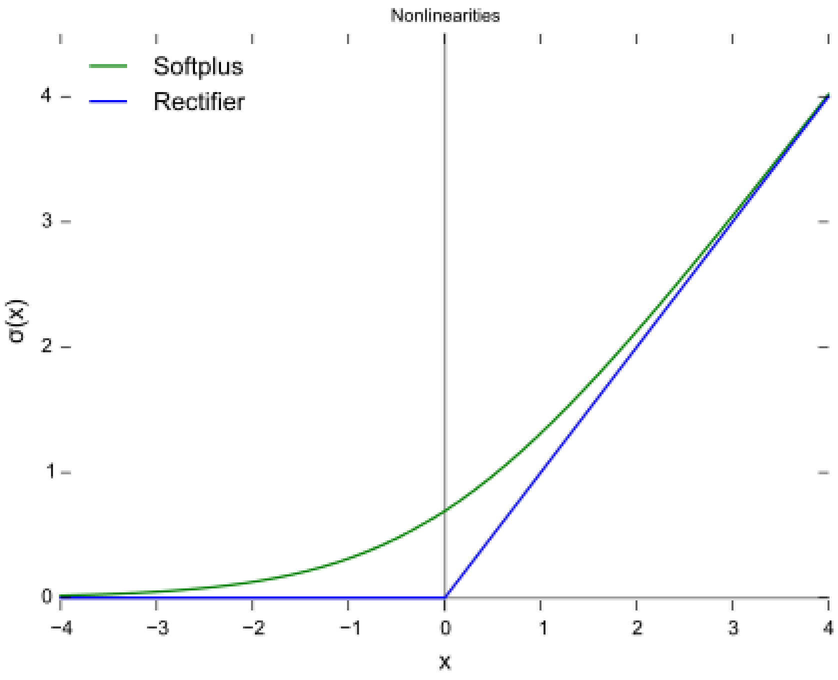
ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$



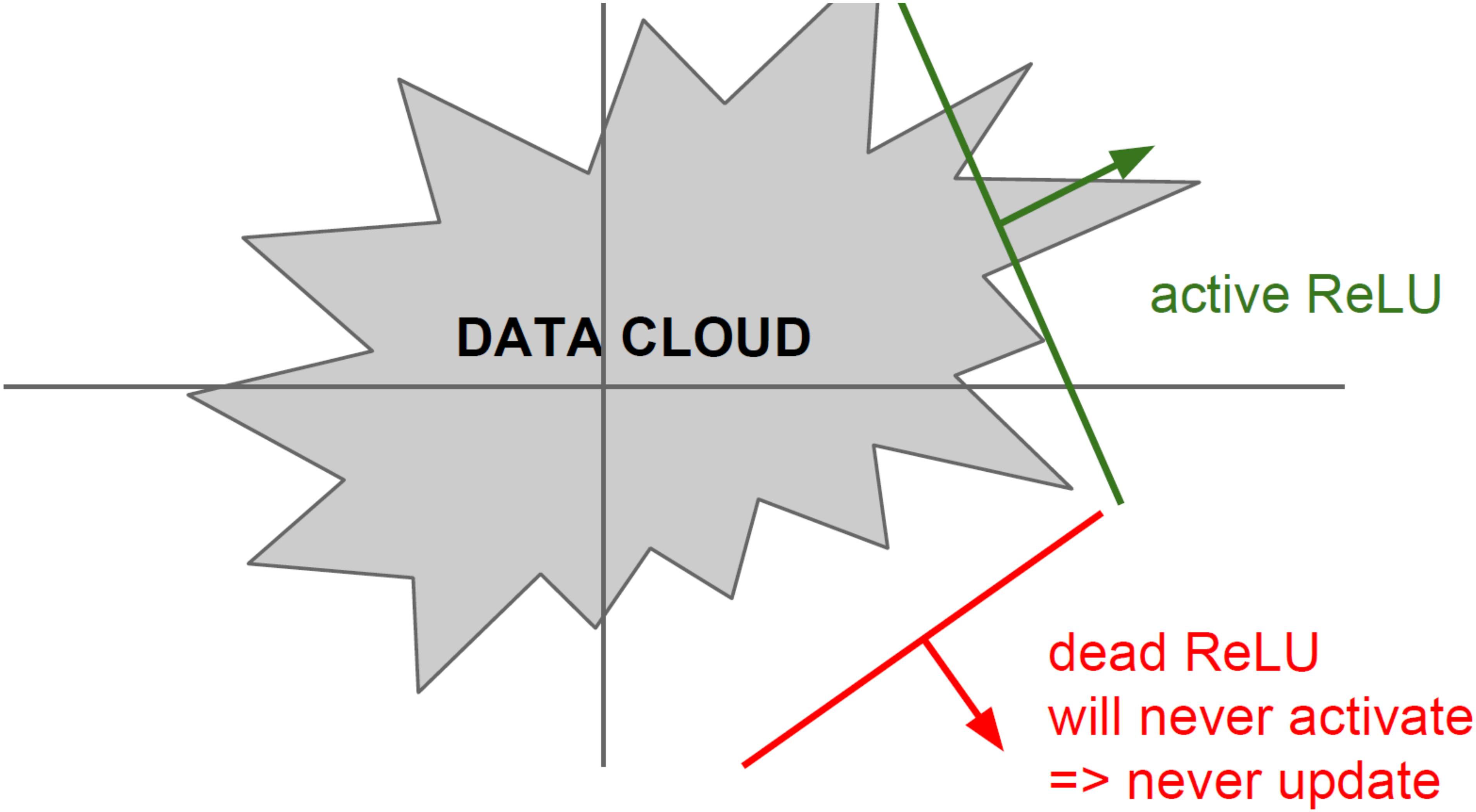
Rectified linear functions

- Does not vanish 50% of the times
- More biologically plausible
- Computationally faster
- Most popular activation function for deep neural networks
- Efficient gradient propagation: No vanishing gradient problem or exploding effect



https://en.wikipedia.org/wiki/Rectifier_%28neural_networks%29

RELU issues



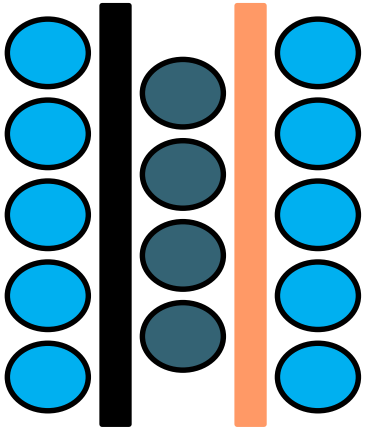
active ReLU

dead ReLU
will never activate
=> never update

Lessons

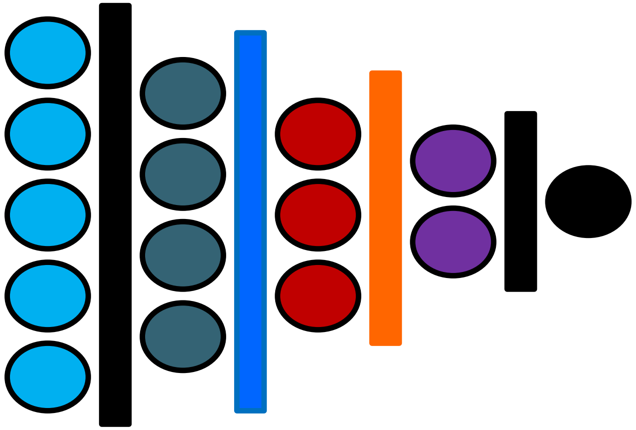
- USE RELU as a default. Check the learning rate and dead neurons
- If they bother you, try Leaky RELU, ELU
- Tanh may work.
- Sigmoid won’t if the net has more than two layers
- Activation of the last layer versus the rest are different

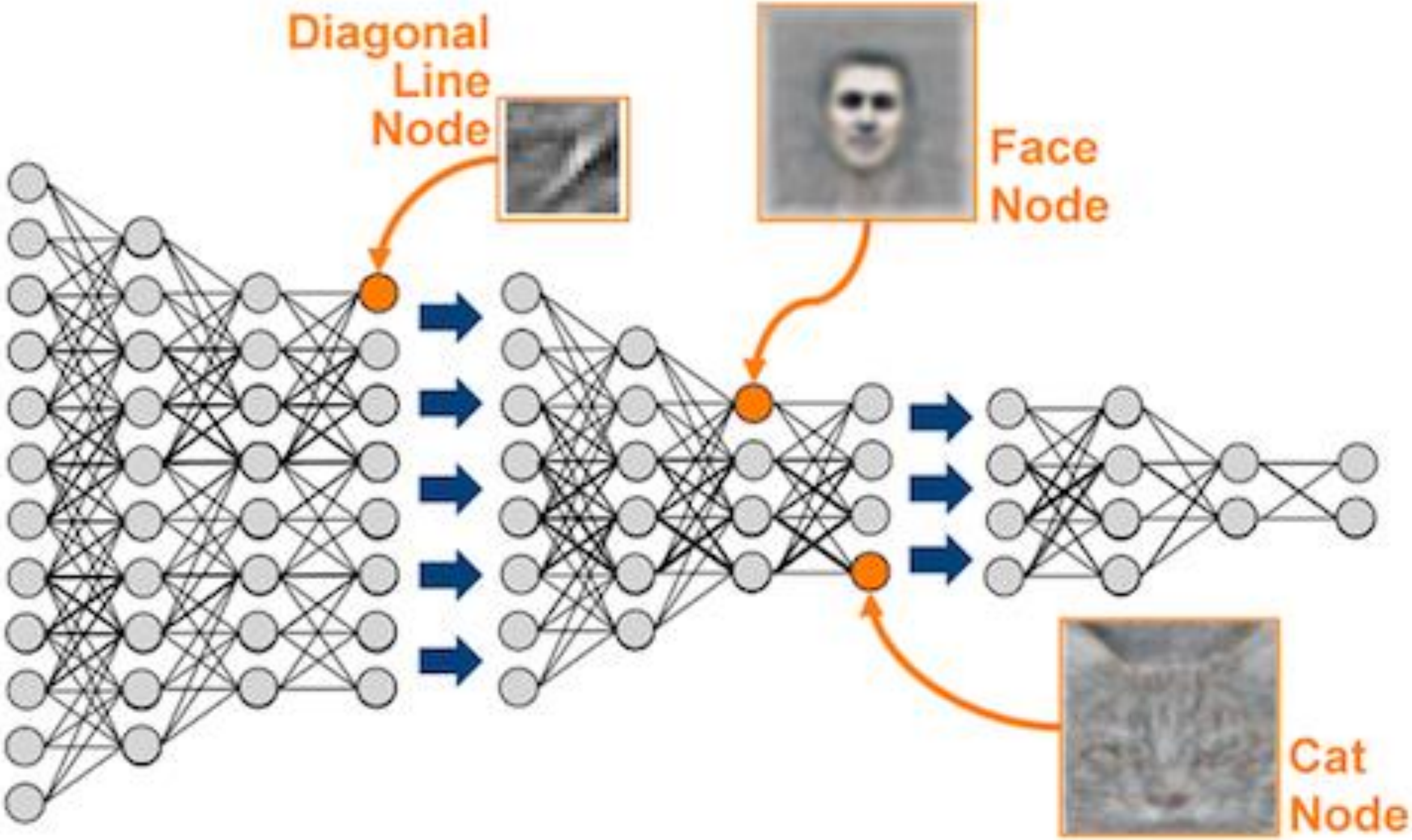
Weight initialization

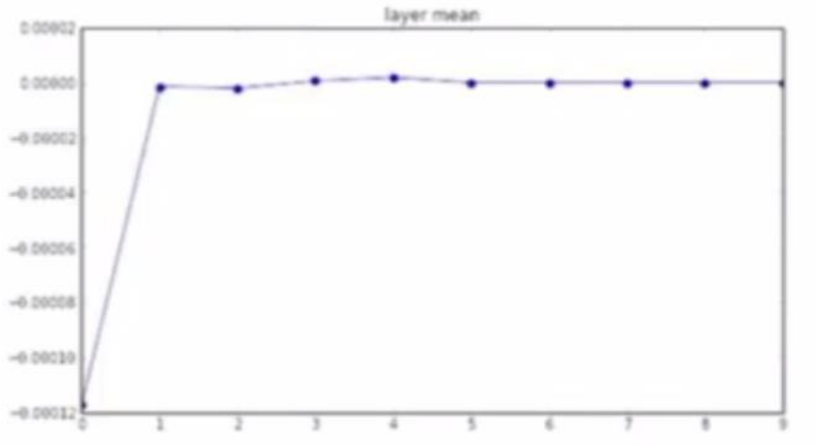


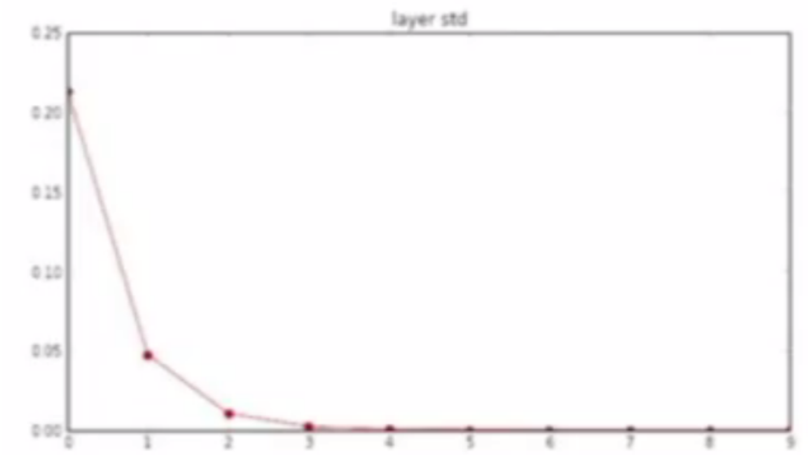
Dense Connection

Dense Connection




Diagonal Line Node
Face Node
Cat Node





Random initialization fails for deep nets
The above graphs are activations

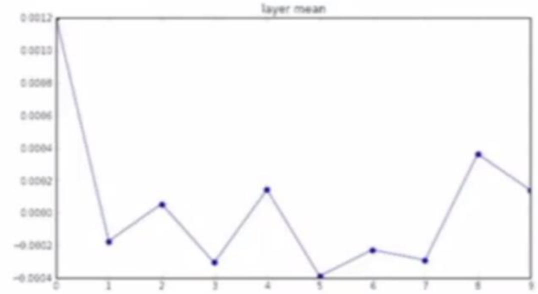
Xavier initialization

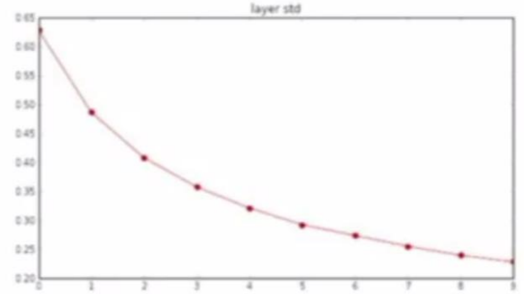
- Initialize each neuron’s weight vector with zero mean and variance

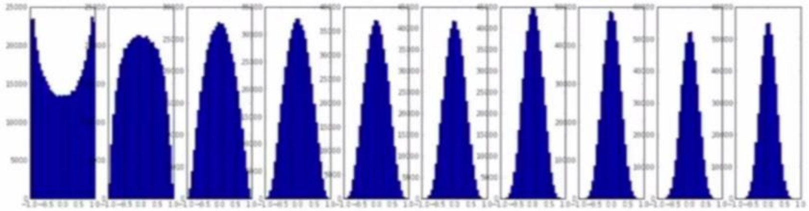
$$\mathrm{Var}(W_i) = \frac{1}{n_{\text{in}}}$$

- where n_{in}in is the number of its inputs into that node

- It fails for RELU though it works for tanh







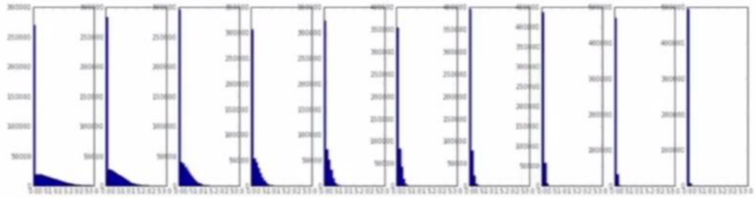
Modified Xavier initialization

- The recommended heuristic is to initialize each neuron’s weight vector is with zero mean and

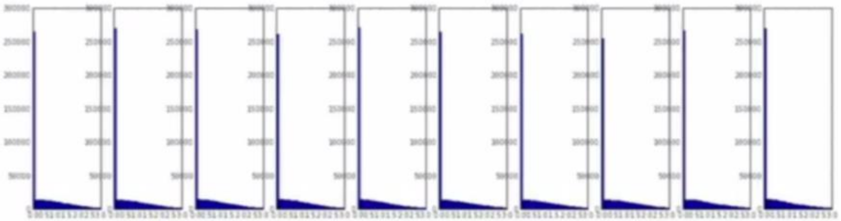
$$\mathrm{Var}(W) = \frac{2}{n_{\text{in}}}$$

- This accounts for the fact that 50% of the nodes go to zero

Unmodified

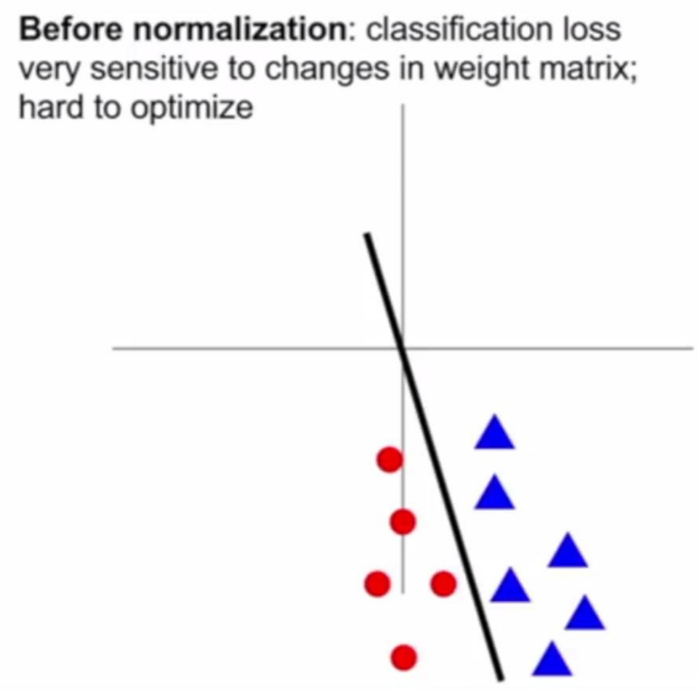


Modified

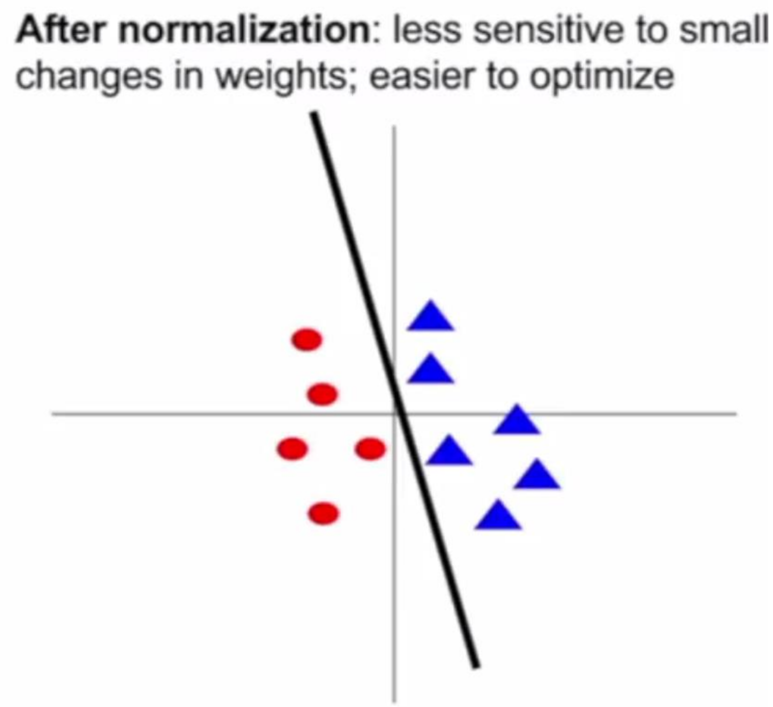


Pre-Processing

- Normalizing the data with a zero mean allows more play for the classifier



Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize



After normalization: less sensitive to small changes in weights; easier to optimize

After the first layer

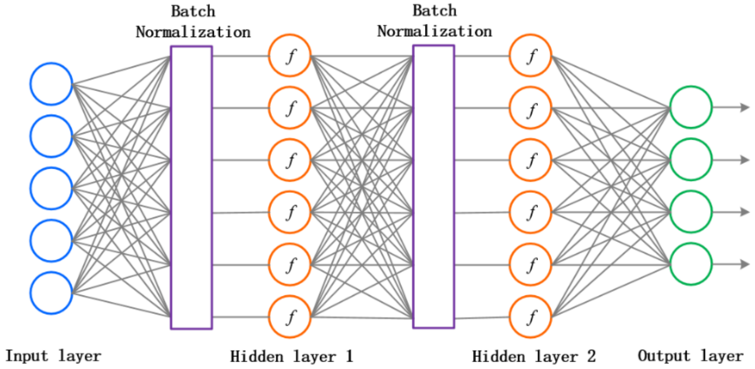
- All the good work of normalization evaporates
  - Summation operation will destroy zero centricity (-2,0,2) can get linearly transformed to (4,6,8)
  - Activation could destroy the distribution
- Solution
  - Re-normalize!

Batch normalization

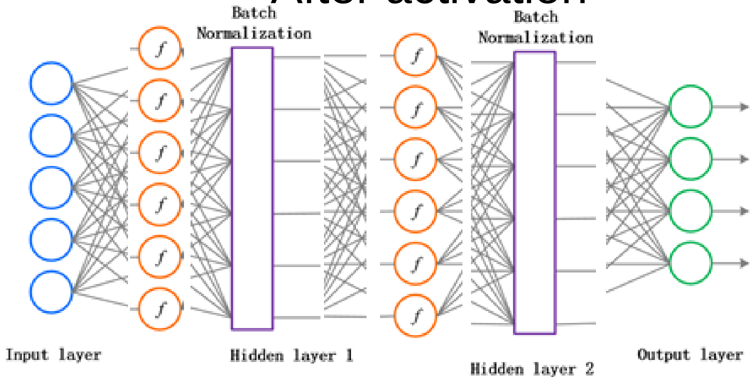
- Normalize the output of each hidden layer either after summation or after activation.
- People reported both winning in different data sets.
- Use the mini batch to normalize. Compute for every mini batch, the mean and variance and use the transformation

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

After summation



After activation



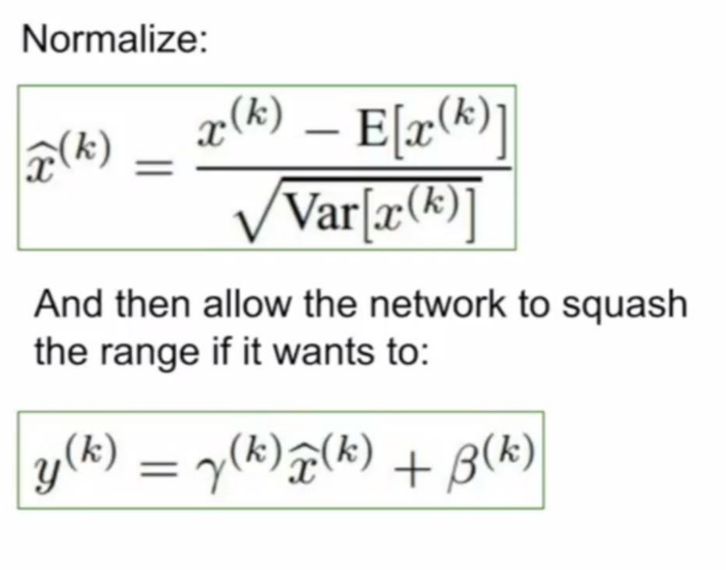
- After normalizing, we allow them to readjust to any center and any variance.

- Gamma and Beta are learnable parameters of the BatchNorm layer and make it basically possible to say “Hey!! I don’t want zero mean/unit variance input, give me back the raw input - it’s better for me.”

- If gamma = sd(x) and beta = mean(x), the original activation is restored.

- This is, what makes BatchNorm really powerful. We initialize the BatchNorm Parameters to transform the input to zero mean/unit variance distributions but during training they can learn that any other distribution might be better.

- Some amount of saturation may be good.

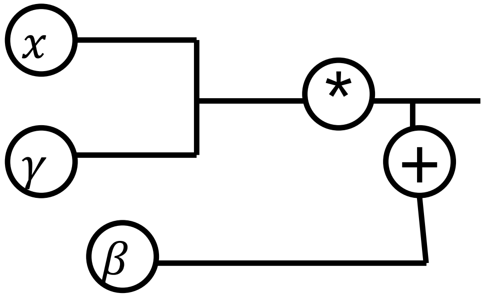


Normalize:

$$\boxed{\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}}$$

And then allow the network to squash the range if it wants to:

$$\boxed{y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}}$$



Gradient descent hacks

Exponential moving average

- $wx_n + w(1-w)x_{n-1} + w(1-w)^2x_{n-2} \dots$

- The same can be written as

$$wx_n + (1-w)s_n$$

s_n *is the average computed until now. If* w *is large, you give lot of importance to the current measurement and vice versa*

Momentum = 1-w

GD Hacks: How about different learning rates for different w

- If the average update of a weight until now is high, change it less; if it is less, increase the update
  - Average update done until now is EMA of gradient square! (why not gradient; positives might cancel negatives)
  - Square root of EMA of gradient square is called RMS prop.

Adam (adaptive moment) update

- Adam update combines momentum and RMS Prop.

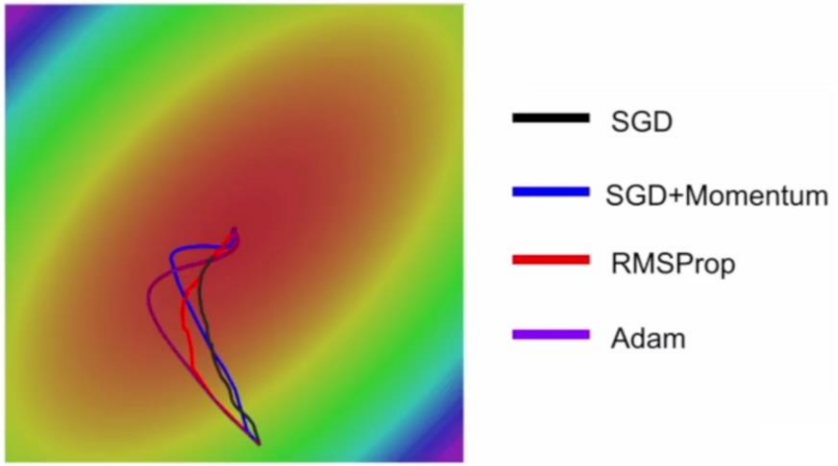
- It has two hyper parameters. Momentum (beta 1) and squared momentum (beta 2).

$$v_1 = 0$$

$$v_{t+1} = \beta_1 v_t + (1 - \beta_1).grad\ (momentum)$$
$$r_{t+1} = \beta_2 r_t + (1 - \beta_2).grad^2\ (RMS\ Prop)$$

$$w_{t+1} = w_t - \alpha.\frac{v_{t+1}}{\sqrt{r_{t+1}} + \epsilon}$$

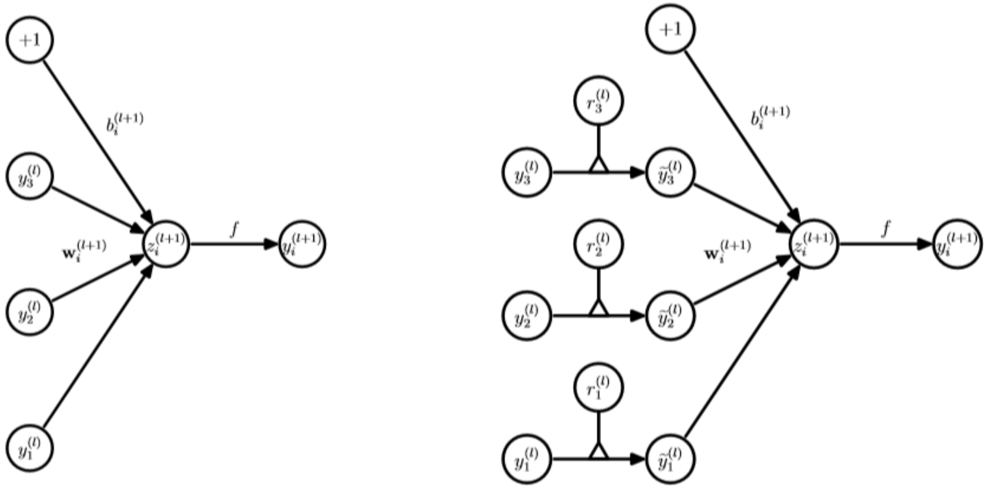
- It is common to implement beta corrections to both $v_{t+1} and\ r_{t+1}$in the above equation.



Adam with beta1 = 0.9,
beta2 = 0.999, and learning_rate = 1e-3 or 5e-4
is a great starting point for many models!

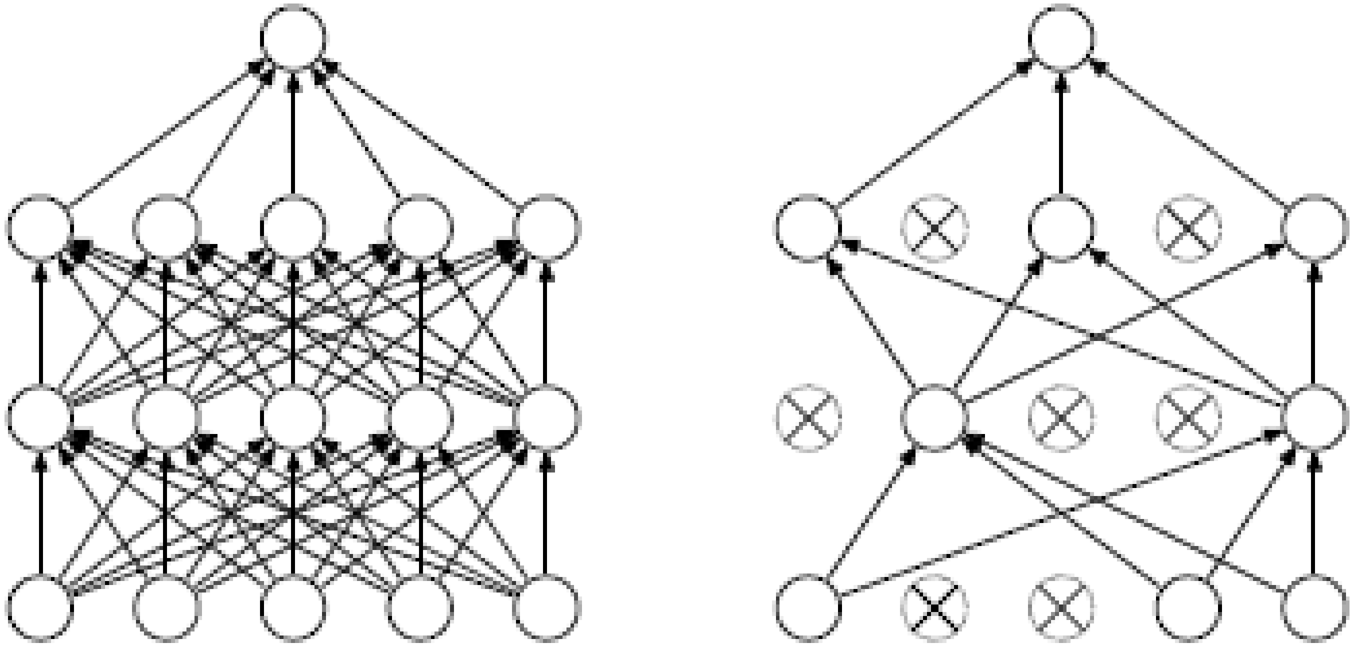
Regularization

Drop out



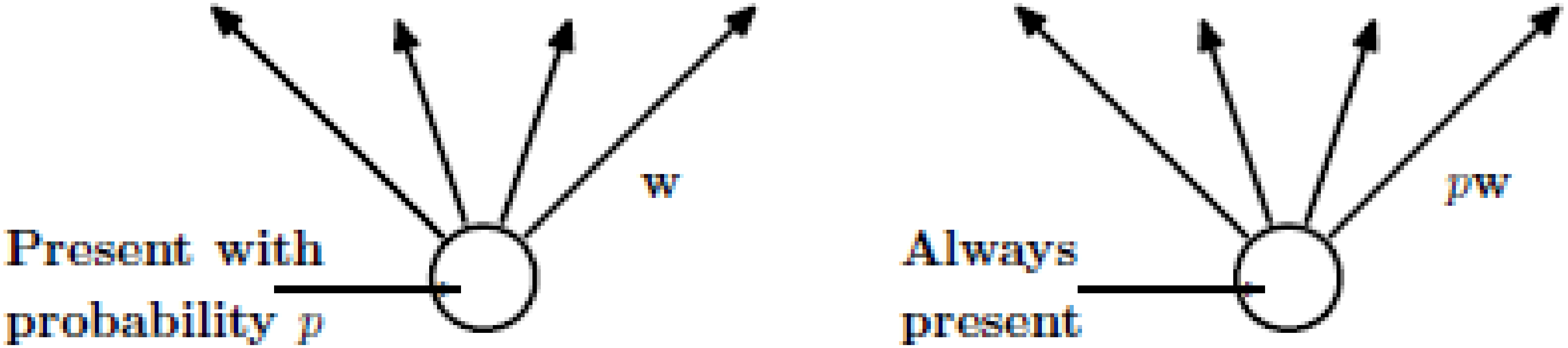
(a) Standard network

(b) Dropout network



(a) Standard Neural Net

(b) After applying dropout.



(a) At training time

(b) At test time

Practical tips

- Network size: n/p
- Learning and momentum: As there is significant noise we need to compensate. 10-100 times the learning rate of normal nets and 0.95-0.99 momentum.
- Max norm regularization
- Keep rate of 0.5 to 0.8 for hidden and 0.8-0.9 for inputs

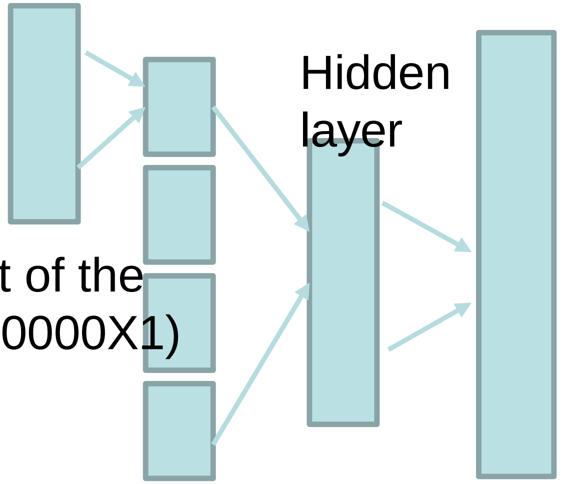
Embedding or Summarization or Vectorization

A SERENDIPITY

Setting up neural embedding

a	o_1	$\longrightarrow$	E	$\longrightarrow$	e_1
glass	o_{3852}	$\longrightarrow$	E	$\longrightarrow$	e_{3852}
of	o_{6163}	$\longrightarrow$	E	$\longrightarrow$	e_{6163}
orange	o_{6257}	$\longrightarrow$	E	$\longrightarrow$	e_{6257}

Softmax (10000 nodes (one hot))



Hidden layer

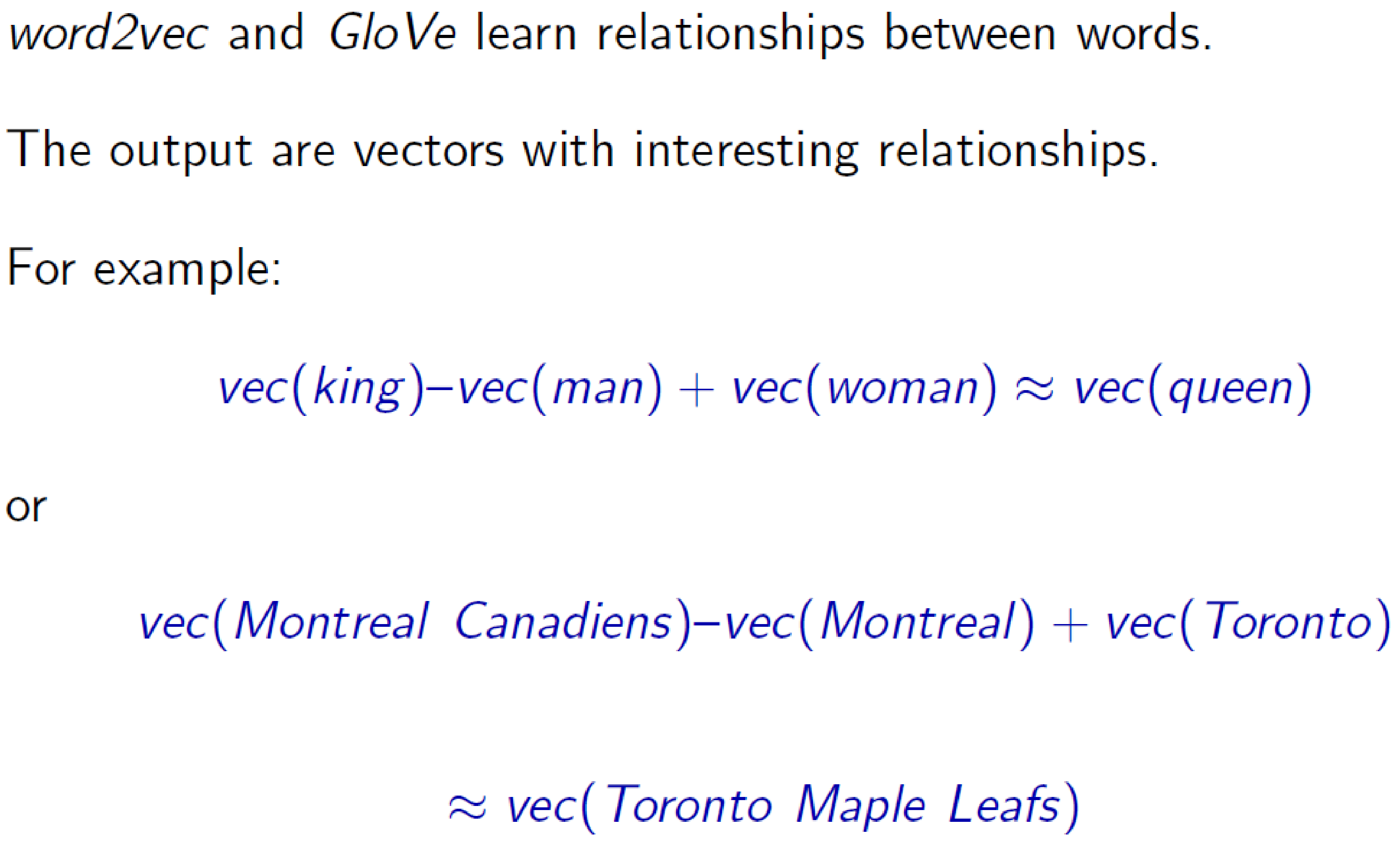
One hot of the word (10000X1)

Goal is to learn the embedding

1200 (4X300) nodes

If we assume a 300 dimensional embedding, 4 words lead to 1200 nodes

Word2Vec worked very well!



word2vec and *GloVe* learn relationships between words.

The output are vectors with interesting relationships.

For example:

$$vec(king) - vec(man) + vec(woman) \approx vec(queen)$$

or

$$vec(Montreal\ Canadiens) - vec(Montreal) + vec(Toronto)$$

$$\approx vec(Toronto\ Maple\ Leafs)$$

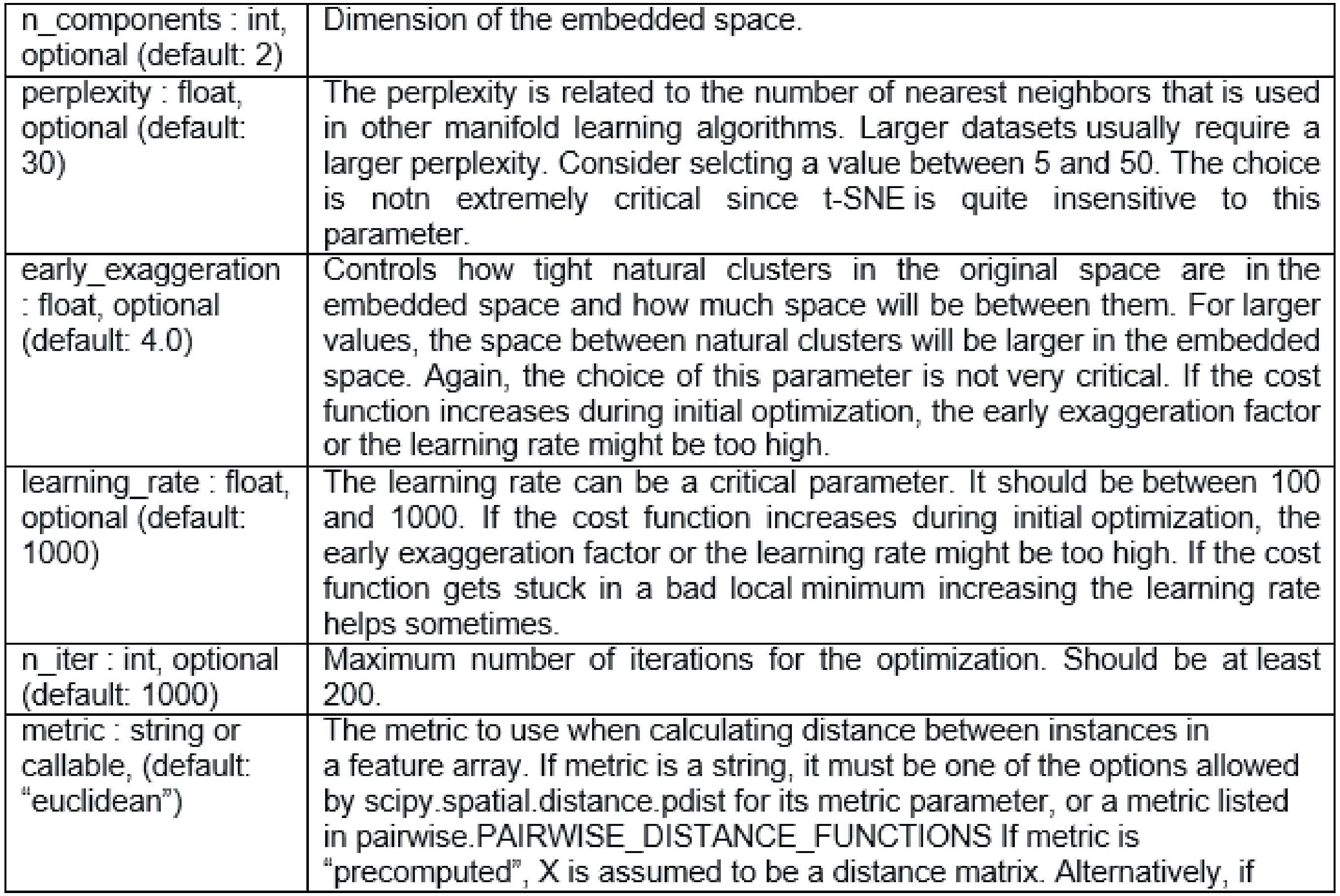
Every categorical attribute can be embedded

- Let us say, country is an attribute and there are 25 nations in your database. How do you feed them to the model
  - Assigning a number based on alphabetical order: We are telling the model that USA is more similar to Uganda than to Canada; confusing the model
  - One hot encoding: All countries are equidistant from each other: Equally bad
  - Intelligently assign values that represent the similarities and dissimilarities: Very difficult to scale
  - Learn them!

T-SNE of F7 layer

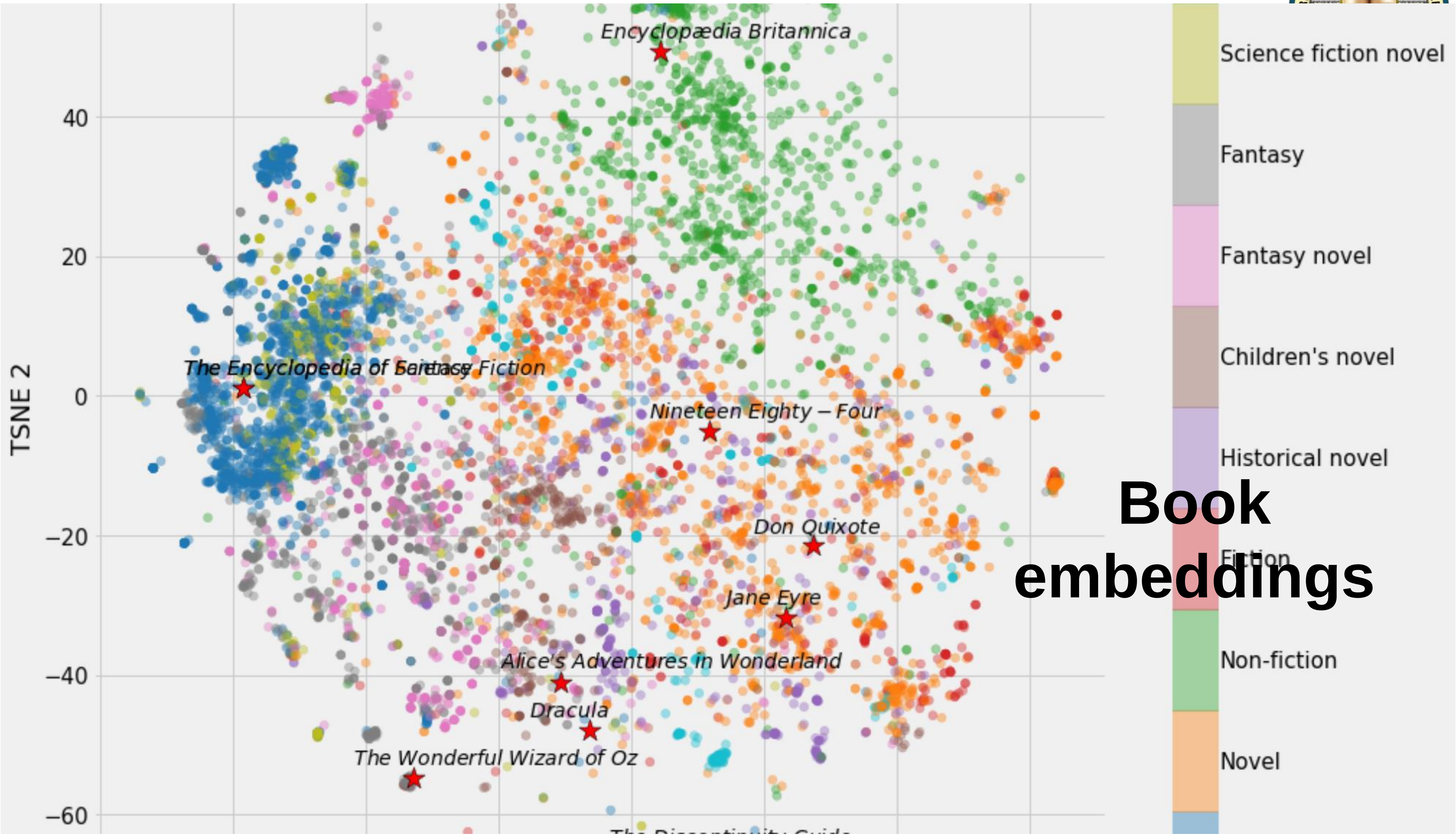
http://jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf

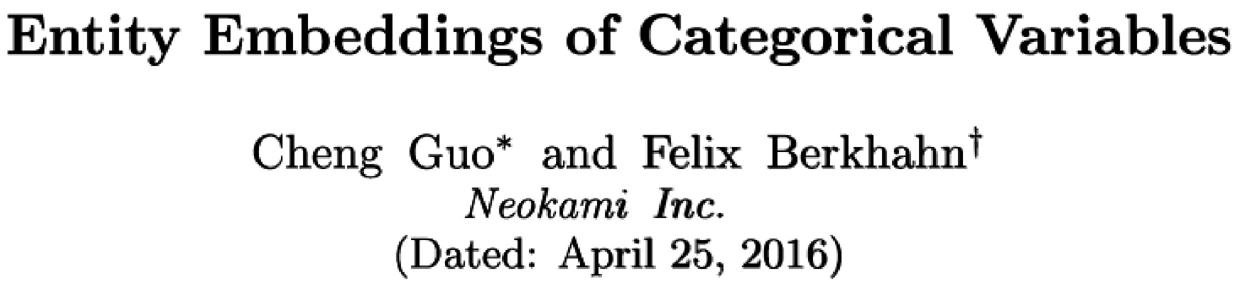
- t-sne is a dimensionality reduction technique that stands for t-distributed stochastic neighbor embedding
- How does it work
  - We find virtual 2-D coordinates that preserve the distance in the high dimensional space



n_components : int, optional (default: 2)	Dimension of the embedded space.
perplexity : float, optional (default: 30)	The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. Consider selcting a value between 5 and 50. The choice is notn extremely critical since t-SNE is quite insensitive to this parameter.
early_exaggeration : float, optional (default: 4.0)	Controls how tight natural clusters in the original space are in the embedded space and how much space will be between them. For larger values, the space between natural clusters will be larger in the embedded space. Again, the choice of this parameter is not very critical. If the cost function increases during initial optimization, the early exaggeration factor or the learning rate might be too high.
learning_rate : float, optional (default: 1000)	The learning rate can be a critical parameter. It should be between 100 and 1000. If the cost function increases during initial optimization, the early exaggeration factor or the learning rate might be too high. If the cost function gets stuck in a bad local minimum increasing the learning rate helps sometimes.
n_iter : int, optional (default: 1000)	Maximum number of iterations for the optimization. Should be at least 200.
metric : string or callable, (default: "euclidean")	The metric to use when calculating distance between instances in a feature array. If metric is a string, it must be one of the options allowed by scipy.spatial.distance.pdist for its metric parameter, or a metric listed in pairwise.PAIRWISE_DISTANCE_FUNCTIONS If metric is "precomputed", X is assumed to be a distance matrix. Alternatively, if

Book embeddings

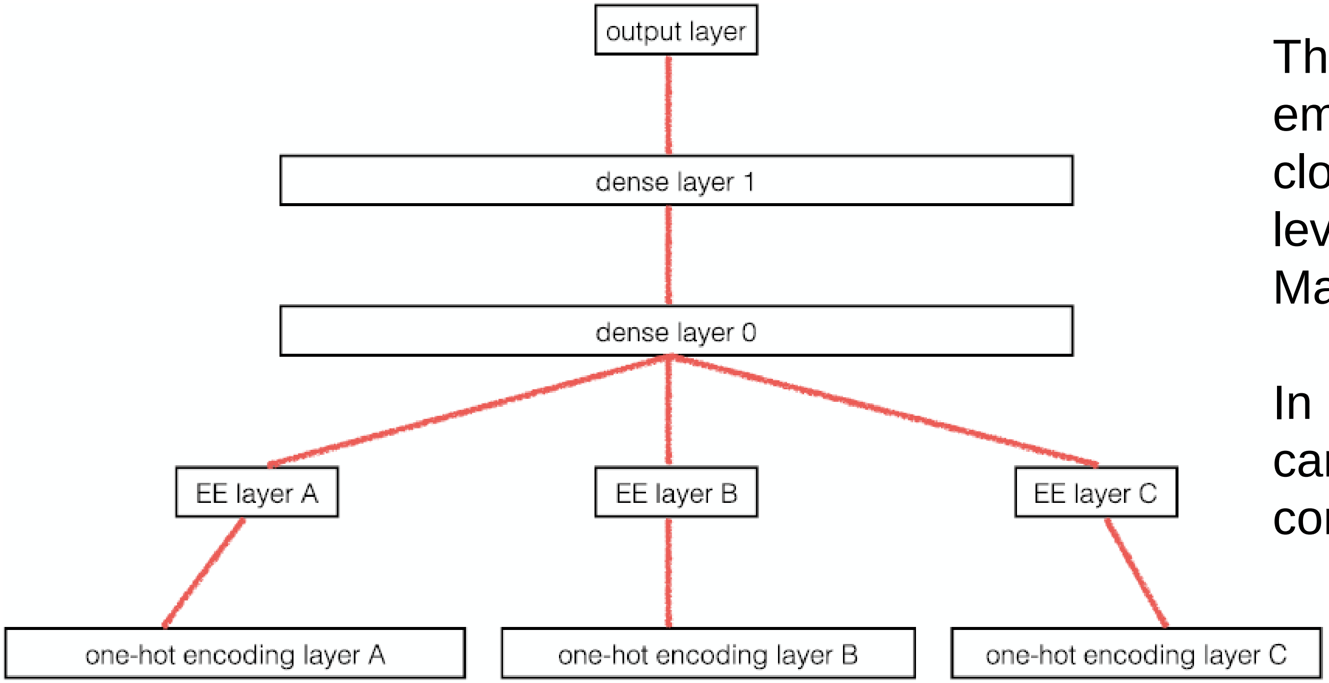




Entity Embeddings of Categorical Variables

Cheng Guo[*] and Felix Berkhahn[†]
Neokami Inc.

(Dated: April 25, 2016)

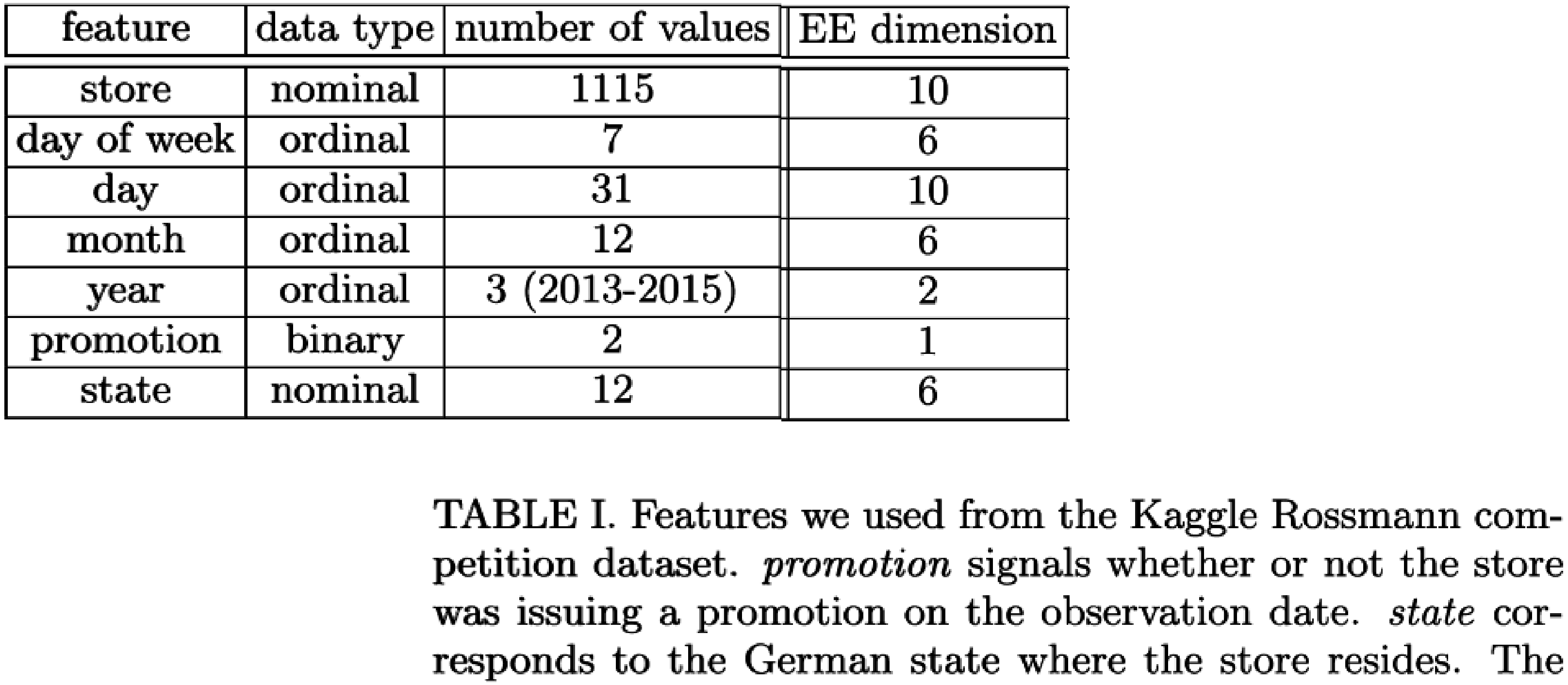


The purpose of embedding is to get closer categorical levels to the similar Mapping

In practice, embeddings can be just linear combinations

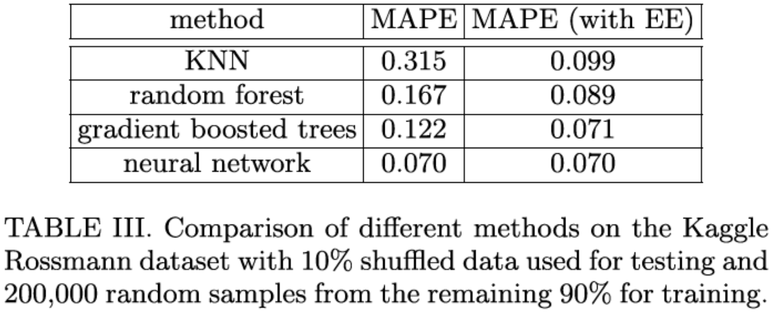
Rossman stores competition in Kaggle

- Forecast the sales as a function of



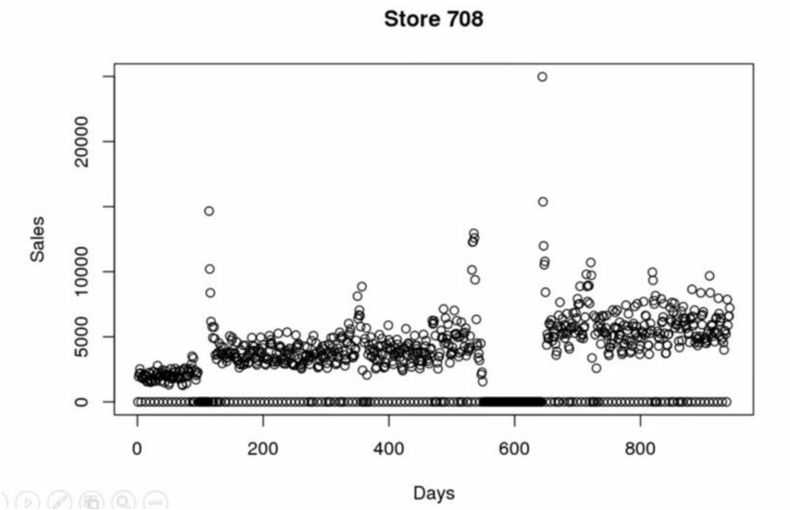
feature	data type	number of values	EE dimension
store	nominal	1115	10
day of week	ordinal	7	6
day	ordinal	31	10
month	ordinal	12	6
year	ordinal	3 (2013-2015)	2
promotion	binary	2	1
state	nominal	12	6

TABLE I. Features we used from the Kaggle Rossmann competition dataset. *promotion* signals whether or not the store was issuing a promotion on the observation date. *state* corresponds to the German state where the store resides. The



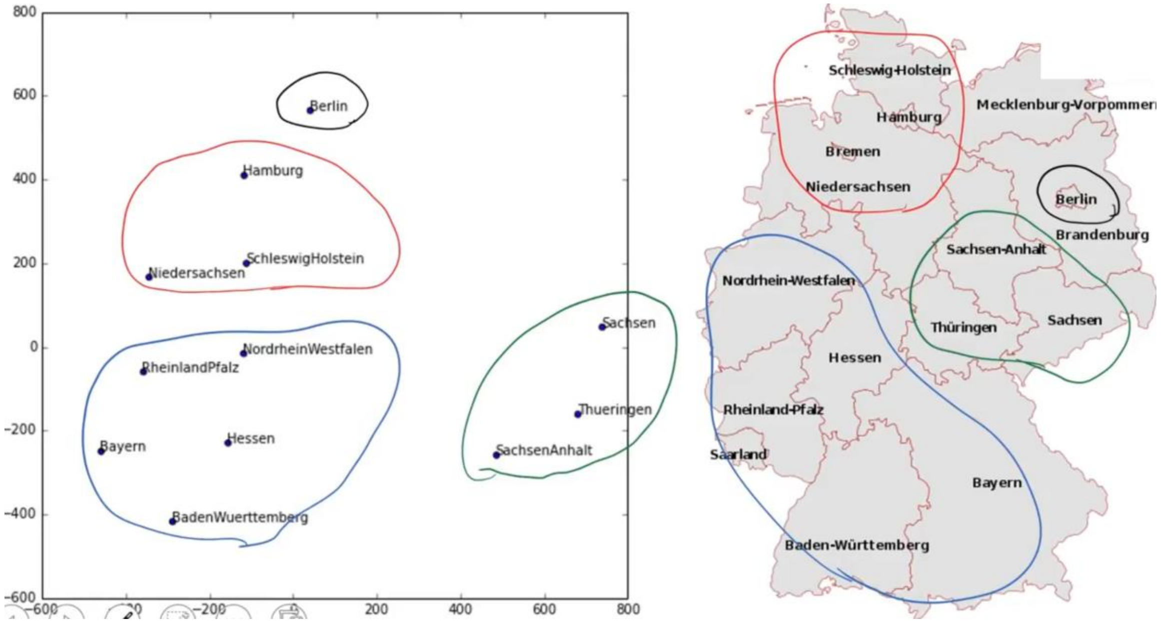
method	MAPE	MAPE (with EE)
KNN	0.315	0.099
random forest	0.167	0.089
gradient boosted trees	0.122	0.071
neural network	0.070	0.070

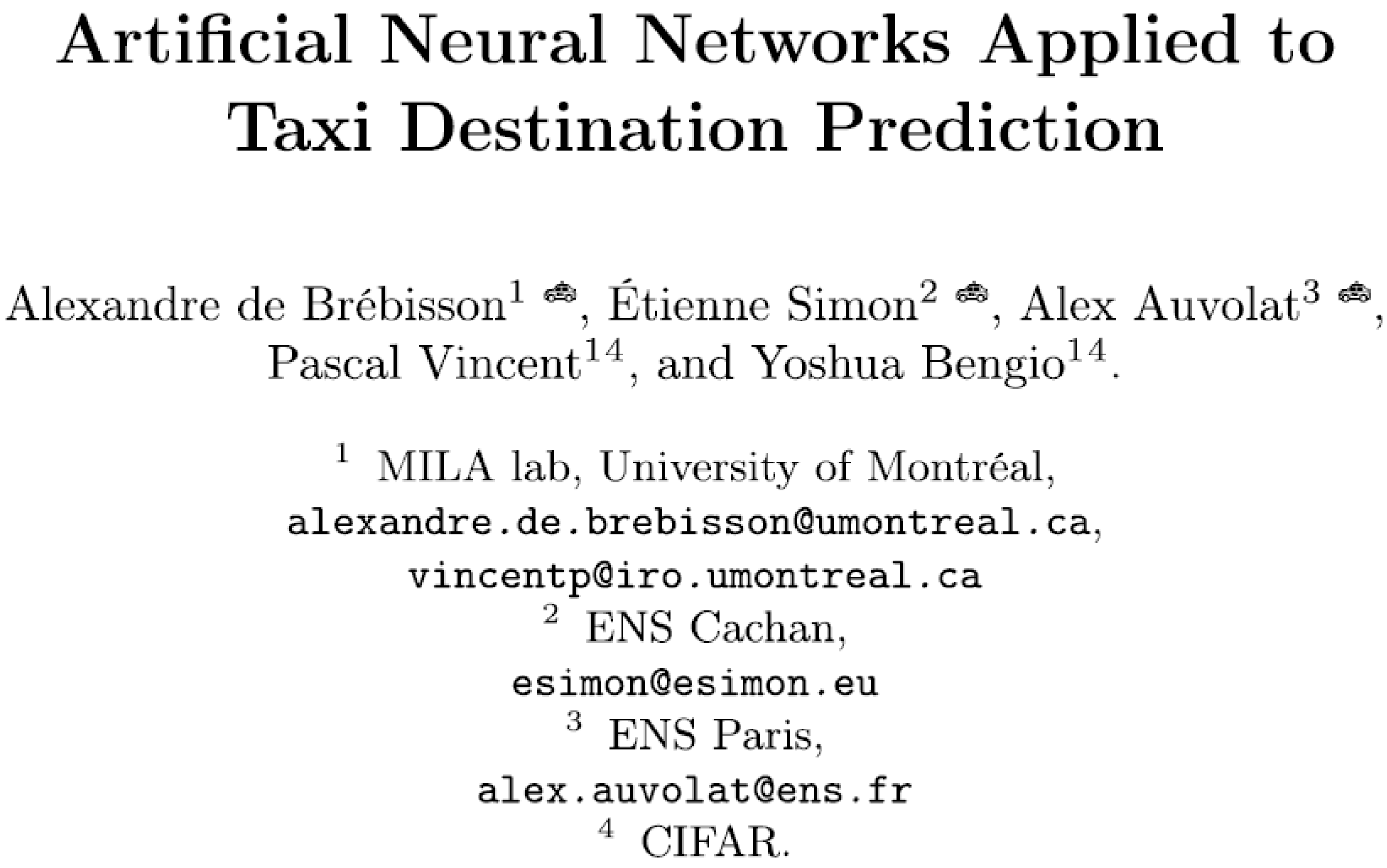
TABLE III. Comparison of different methods on the Kaggle Rossmann dataset with 10% shuffled data used for testing and 200,000 random samples from the remaining 90% for training.



Feature representations: Cat2Vec

Level	Representation
Bayern	1,0,0,0,0,0,0,0,0,0,0, 0
BadenWuerttemberg	0,1,0,0,0,0,0,0,0,0,0, 0
Berlin	0,0,1,0,0,0,0,0,0,0,0, 0
Hamburg	
Hessen	

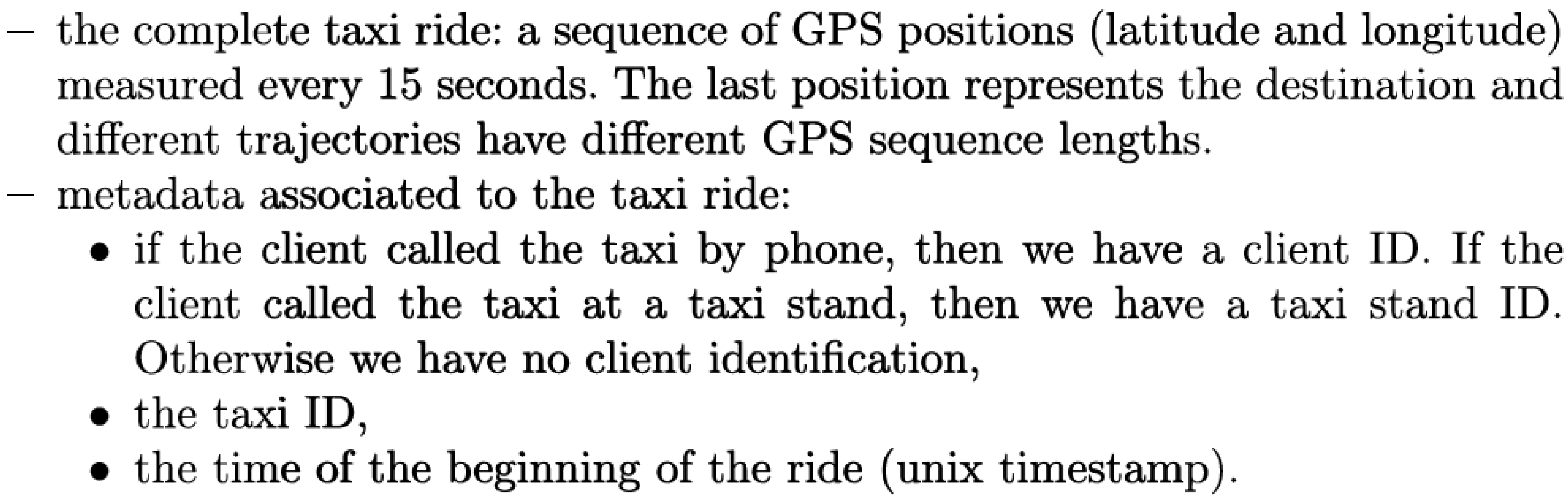




Artificial Neural Networks Applied to Taxi Destination Prediction

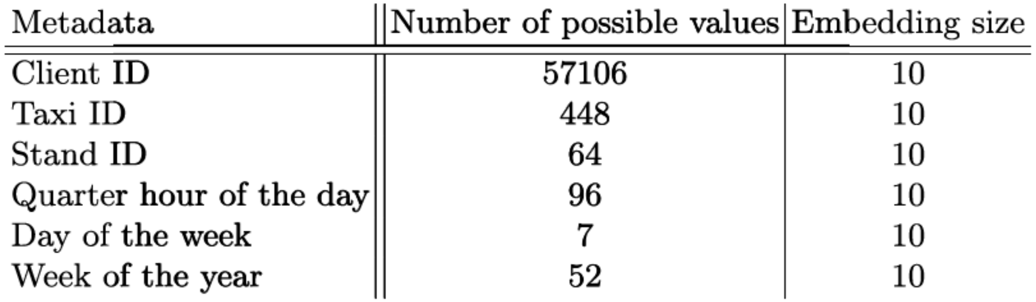
Alexandre de Brébisson[1] 🚕, Étienne Simon[2] 🚕, Alex Auvolat[3] 🚕,
Pascal Vincent[1 4], and Yoshua Bengio[1 4].

[1] MILA lab, University of Montréal,
alexandre.de.brebisson@umontreal.ca,
vincentp@iro.umontreal.ca
[2] ENS Cachan,
esimon@esimon.eu
[3] ENS Paris,
alex.auvolat@ens.fr
[4] CIFAR.



- the complete taxi ride: a sequence of GPS positions (latitude and longitude) measured every 15 seconds. The last position represents the destination and different trajectories have different GPS sequence lengths.
- metadata associated to the taxi ride:
  - if the client called the taxi by phone, then we have a client ID. If the client called the taxi at a taxi stand, then we have a taxi stand ID. Otherwise we have no client identification,
  - the taxi ID,
  - the time of the beginning of the ride (unix timestamp).

Table 1: Metadata values and associated embedding size.



Metadata	Number of possible values	Embedding size
Client ID	57106	10
Taxi ID	448	10
Stand ID	64	10
Quarter hour of the day	96	10
Day of the week	7	10
Week of the year	52	10

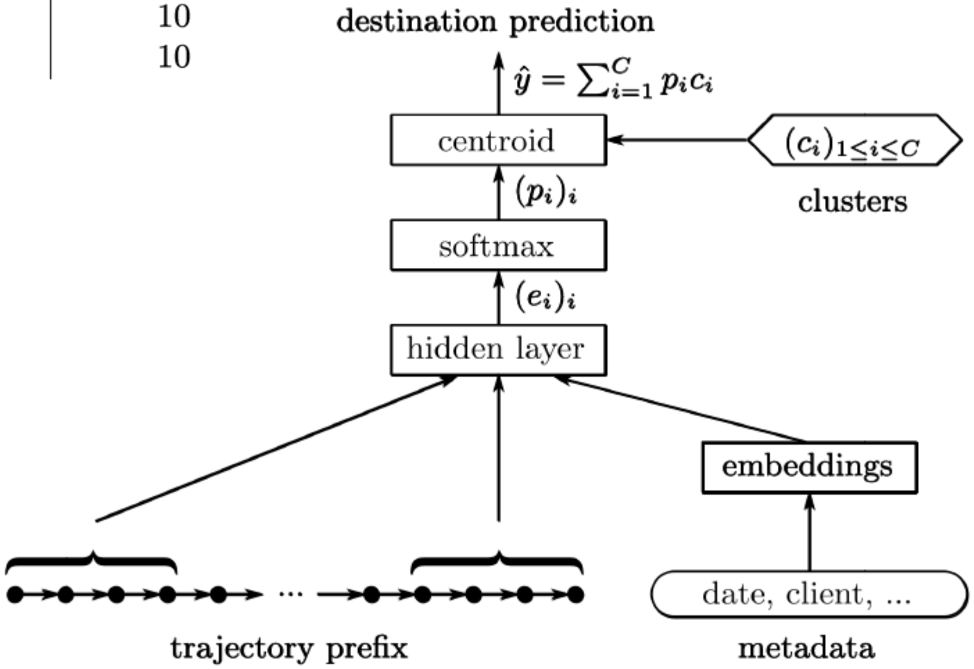
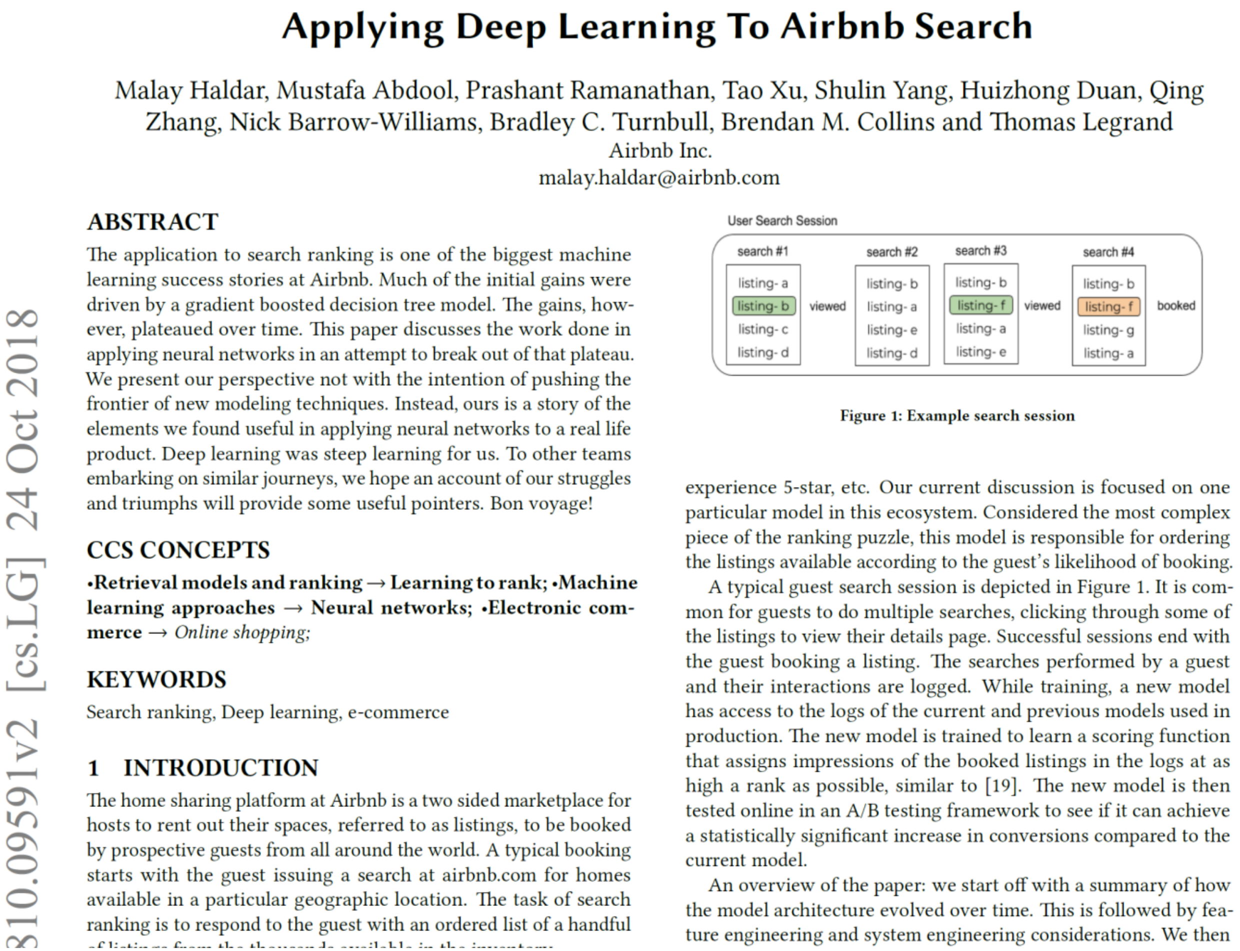


Fig. 1: Architecture of the winning model.

https://www.linkedin.com/in/malayhaldar/



Applying Deep Learning To Airbnb Search

Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C. Turnbull, Brendan M. Collins and Thomas Legrand
Airbnb Inc.
malay.haldar@airbnb.com

ABSTRACT

The application to search ranking is one of the biggest machine learning success stories at Airbnb. Much of the initial gains were driven by a gradient boosted decision tree model. The gains, however, plateaued over time. This paper discusses the work done in applying neural networks in an attempt to break out of that plateau. We present our perspective not with the intention of pushing the frontier of new modeling techniques. Instead, ours is a story of the elements we found useful in applying neural networks to a real life product. Deep learning was steep learning for us. To other teams embarking on similar journeys, we hope an account of our struggles and triumphs will provide some useful pointers. Bon voyage!

CCS CONCEPTS

•Retrieval models and ranking → Learning to rank; •Machine learning approaches → Neural networks; •Electronic commerce → *Online shopping;*

KEYWORDS

Search ranking, Deep learning, e-commerce

1 INTRODUCTION

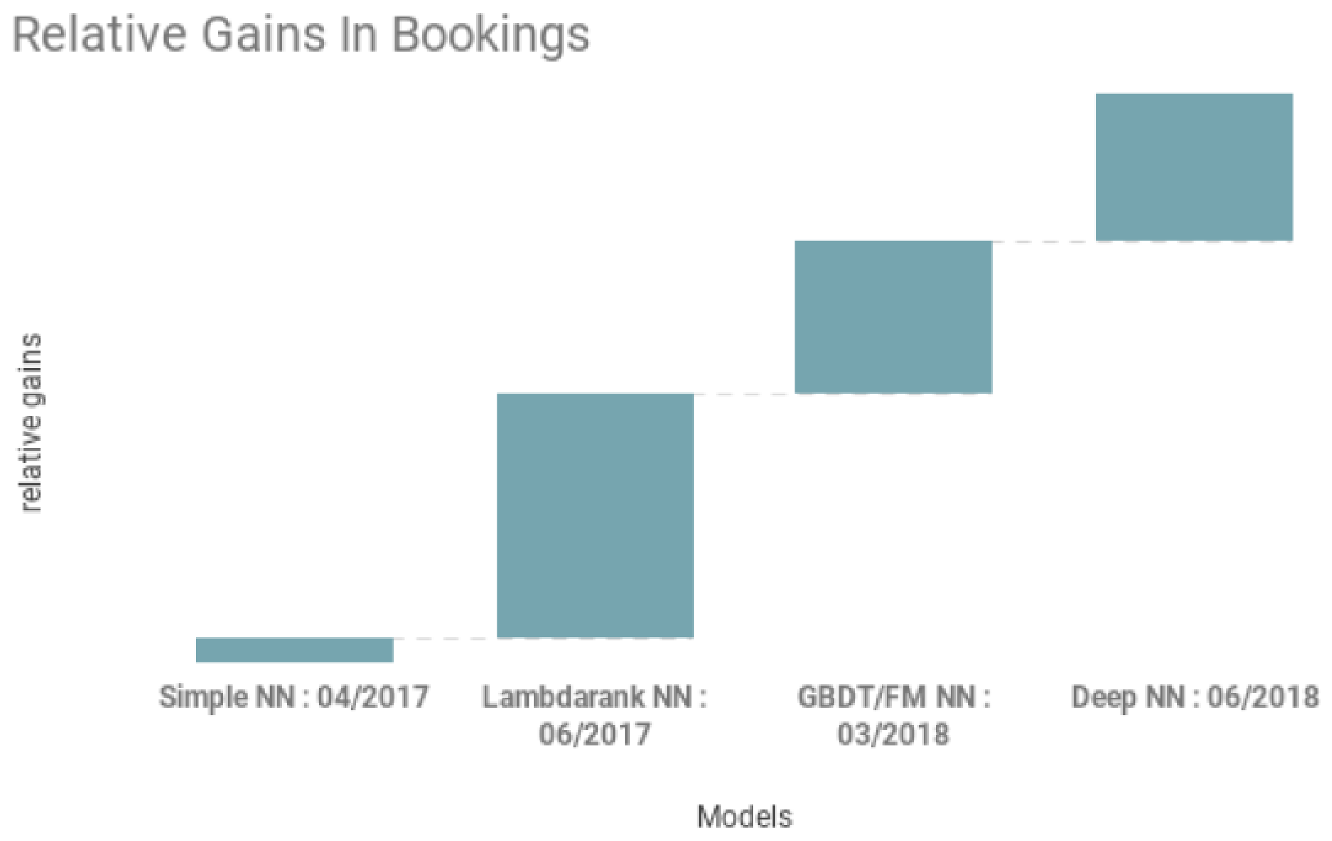
The home sharing platform at Airbnb is a two sided marketplace for hosts to rent out their spaces, referred to as listings, to be booked by prospective guests from all around the world. A typical booking starts with the guest issuing a search at airbnb.com for homes available in a particular geographic location. The task of search ranking is to respond to the guest with an ordered list of a handful of listings from the thousands available in the inventory

Figure 1: Example search session

experience 5-star, etc. Our current discussion is focused on one particular model in this ecosystem. Considered the most complex piece of the ranking puzzle, this model is responsible for ordering the listings available according to the guest's likelihood of booking.

A typical guest search session is depicted in Figure 1. It is common for guests to do multiple searches, clicking through some of the listings to view their details page. Successful sessions end with the guest booking a listing. The searches performed by a guest and their interactions are logged. While training, a new model has access to the logs of the current and previous models used in production. The new model is trained to learn a scoring function that assigns impressions of the booked listings in the logs at as high a rank as possible, similar to [19]. The new model is then tested online in an A/B testing framework to see if it can achieve a statistically significant increase in conversions compared to the current model.

An overview of the paper: we start off with a summary of how the model architecture evolved over time. This is followed by feature engineering and system engineering considerations. We then


Relative Gains In Bookings
relative gains
Simple NN : 04/2017
Lambdarank NN : 06/2017
GBDT/FM NN : 03/2018
Deep NN : 06/2018
Models

Don’t be a hero: Start simple

First architecture that we finally managed to get online was a simple single hidden layer NN with 32 fully connected ReLU activations that proved booking neutral against the GBDT model.

NN was fed by the same features as the GBDT model. Training objective for the NN was also kept invariant w.r.t the GBDT model: minimizing the L2 regression loss where booked listings are assigned a utility of 1.0 and listings that are not booked a utility of 0.0.

Value of the whole exercise was that it validated that the entire NN pipeline was production ready and capable of serving live traffic.

Find how experts are solving similar problems and copy shamelessly

Not being a hero got us to a start, but not very far. In time we would adapt Karpathy’s advice to: don’t be a hero, in the beginning.

Our first breakthrough came when we combined a NN with the idea behind Lamdarank [2].

When you have data, do DNN

we were able to deprecate all that complexity by simply scaling the training data 10x and moving to a DNN with 2 hidden layers.

Typical configuration of the network: an input layer with a total of 195 features after expanding categorical features to embeddings, feeding the first hidden layer with 127 fully connected ReLUs, and then the second hidden layer with 83 fully connected ReLUs.

Features feeding the DNN were mostly simple properties of the listings such as price, amenities, historical booking count, etc, fed directly with minimal feature engineering.

Smart features do help

Exceptions include features output from another model:

Price of listings that have the Smart Pricing feature enabled, supplied by a specialized model [24].

Similarity of the listing to the past views of the user, computed based on co-view embeddings [9].

These models tap into data that isn’t directly part of the search ranking training examples, providing the DNN with additional information.

Embeddings don’t work without data

Each listing at Airbnb has a corresponding unique id. One of the exciting new opportunities unlocked by NNs was to use these listing ids as features. .e idea was to use the listing ids as index into an embedding, which allowed us to learn a vector representation per listing, encoding their unique properties.

The reason why such an established technique fails at Airbnb is because of some unique properties of the underlying marketplace.

Embeddings need substantial amounts of data per item to converge to reasonable values. When items can be repeated without constraints, such as online videos or words in a language, there is no limit to the amount of user interaction an item can have. Obtaining large amounts of data for the items of interest is relatively
easy.

Listings, on the other hand, are subjected to constraints from the physical world. Even the most popular listing can be booked at most 365 times in an entire year. Typical bookings per listing are much fewer. This is the fundamental limitation.

But, they do work

For example, given the query ”San Francisco” and a listing near the Embarcadero, we created an ID 71829521 to build our categorical feature. Categorical features are then mapped to an embedding, which feed the NN.

During training, the model infers the embedding with back propagation which encodes the location preference for the neighbourhood.

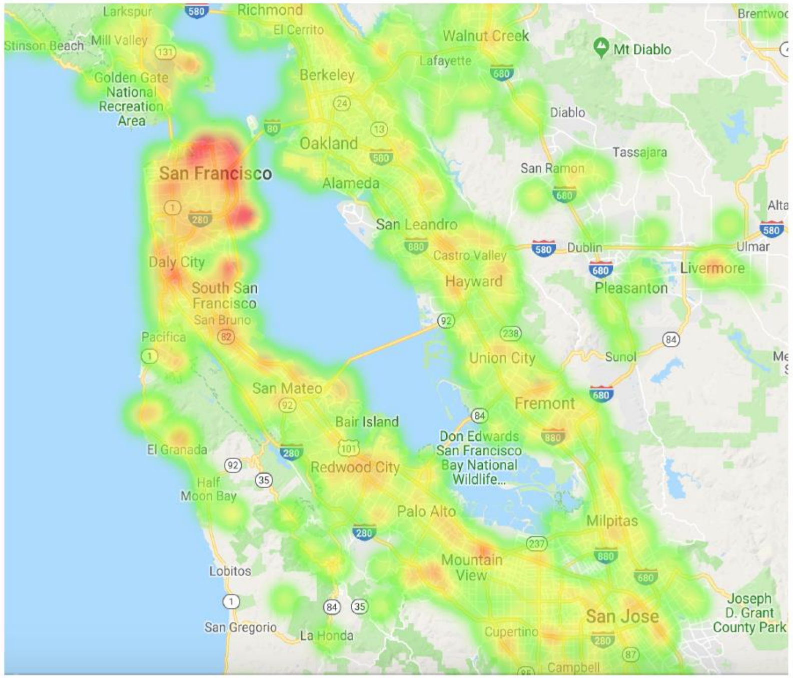
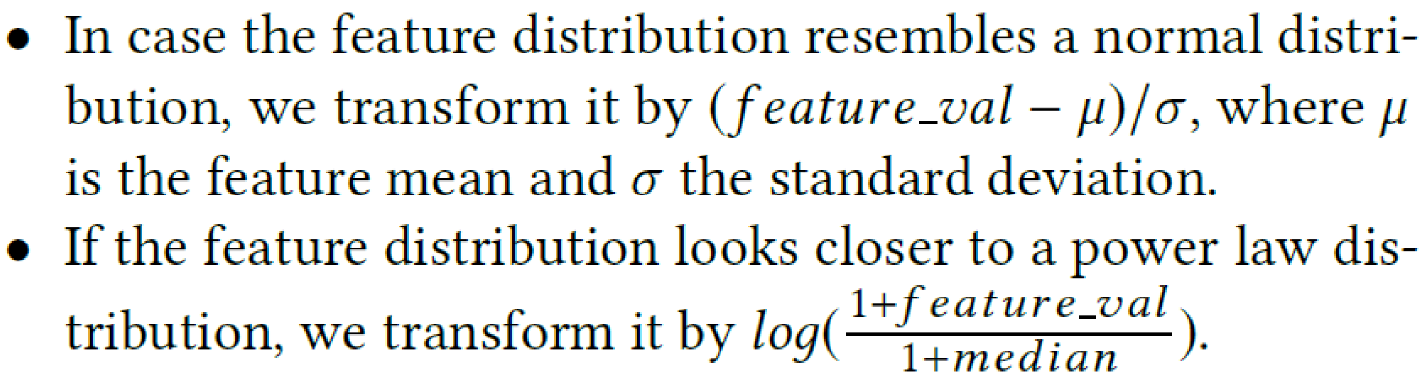


Figure 13: Location preference learnt for the query ”San Francisco”

Feature engineering

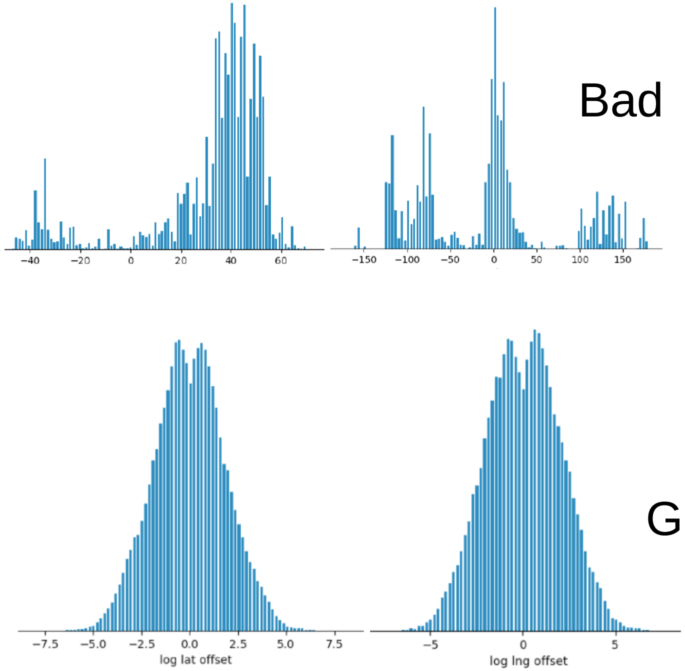
- Normalize



- In case the feature distribution resembles a normal distribution, we transform it by $(feature\_val - \mu)/\sigma$, where μ is the feature mean and σ the standard deviation.
- If the feature distribution looks closer to a power law distribution, we transform it by $log(\frac{1+feature\_val}{1+median})$.

Feature engineering

- DNNs prefer smooth inputs. So, give them that



Bad

Good

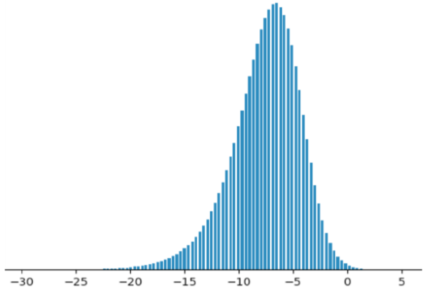


Figure 8: Distribution of output layer.

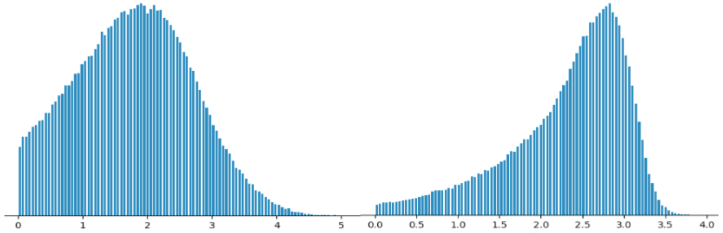


Figure 9: Example distributions from second hidden layer.

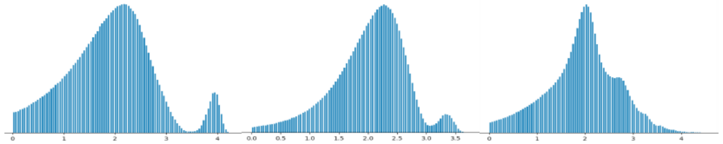


Figure 10: Example distributions from first hidden layer.

Adam works!!

Initialization. Out of sheer habit, we started our first model by initializing all weights and embeddings to zero, only to discover that is the worst way to start training a neural network.

After surveying different techniques, our current choice is to use Xavier initialization [5] for the network weights and random uniform in the range(-1, 1) for embeddings.

Learning rate. An overwhelming range of strategies confronted us here, but for our application we found it hard to improve upon the performance of Adam [12] with its default se.ings. Currently we use a variant LazyAdamOptimizer [7], which we found faster when training with large embeddings.

Issue with traditional ML: Poor representations and summarization of unstructured information

In modern AI, we are solving these problems

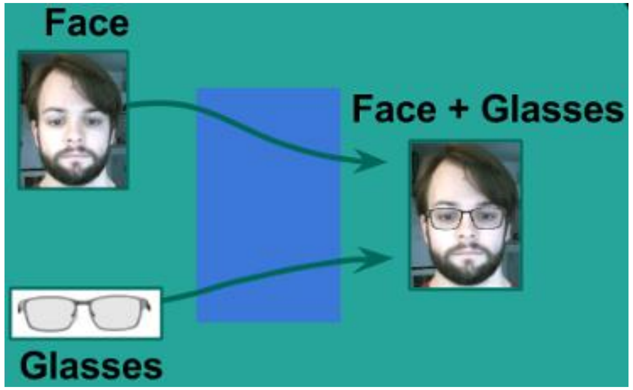
- DNNs summarize text, audio, image and categorical attributes

Categorical attributes (DNN)

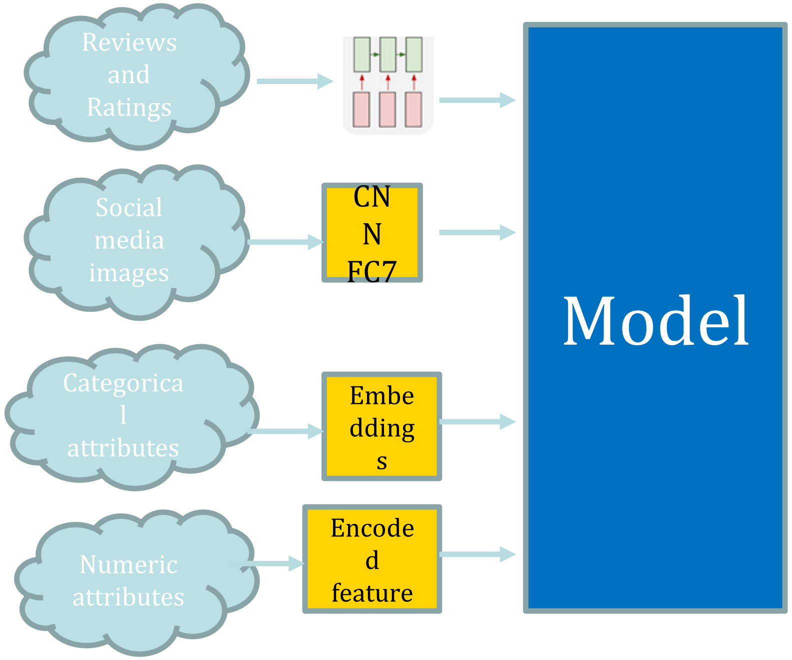
Images (CNN)

data (text, audio, video, time series)

King – Man + Woman = Queen



Inclusive Models + More Natural/Simple Architectures

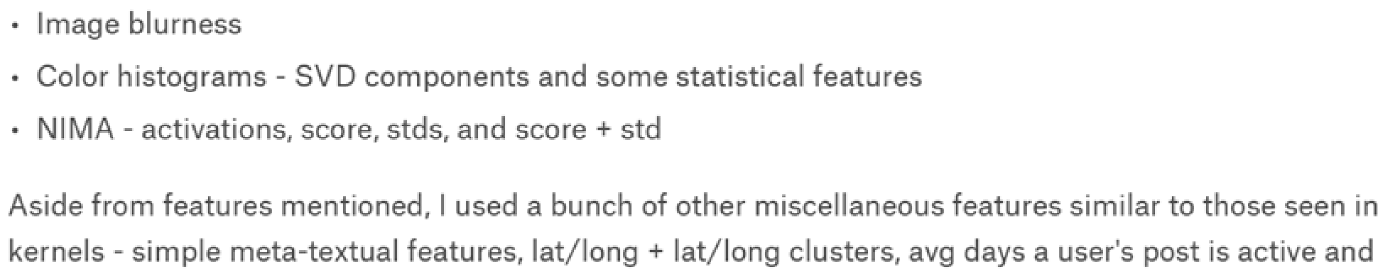


Kaggle Avito contest

- Looking at the ad, can you predict the leads
  - Engineer features
    - Structured data (product, catetory etc.)
    - Text title and description
    - Image of the product

4th place
LGB (Light gradient boosting) features

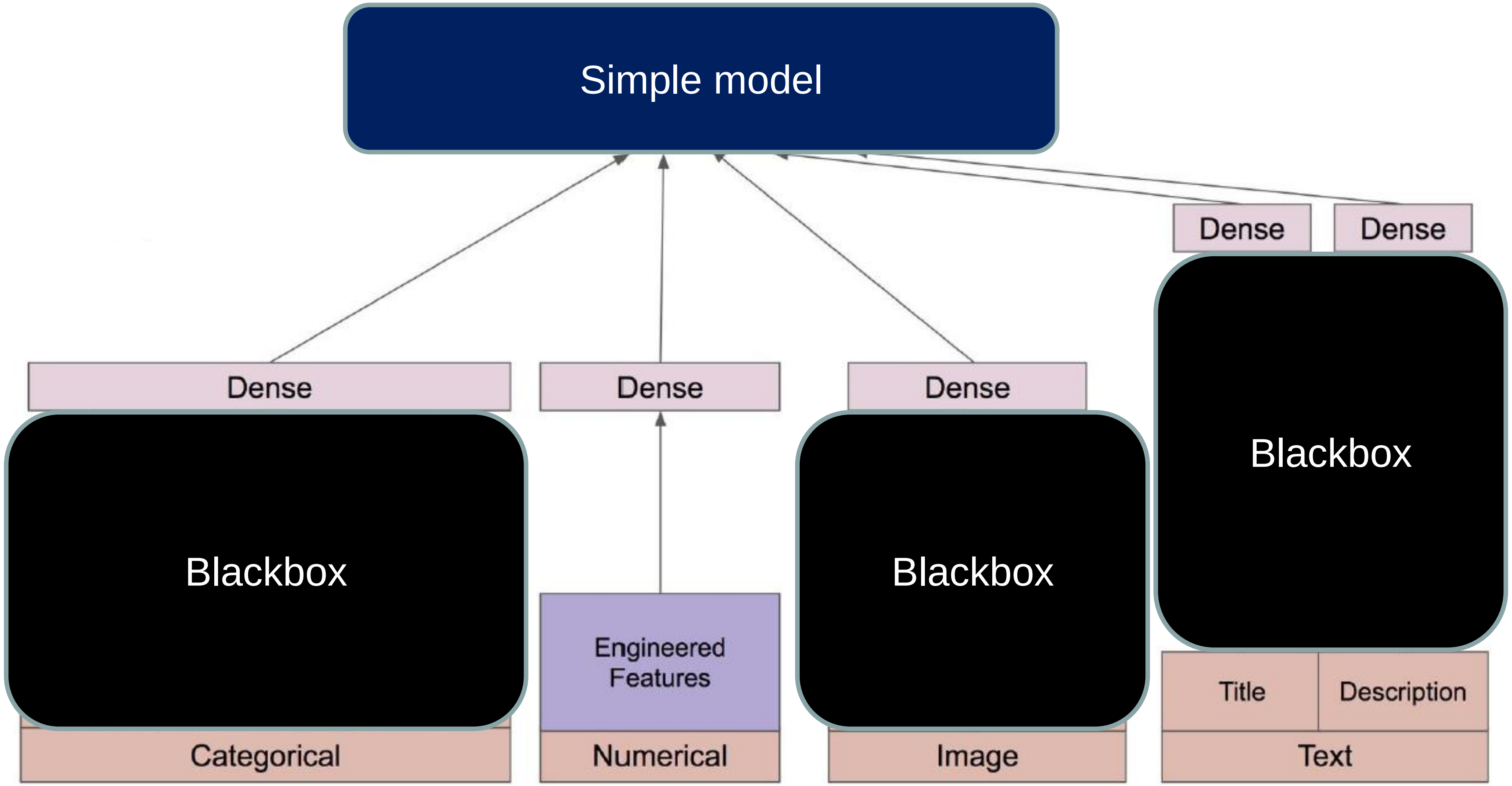
- TfidfVectorizer
- Title_noun aggregates: I extract all nouns from each title, normalized them, removed duplicates and sorted them alphabetically, and then used the resulting key as a basis for aggregation
- Titlenounadjs aggregates: same as above, but adding adjectives as well
- Title_cluster: using title tf-idf features, I run SVD with 500 components and form 30,000 k-means clusters on these components. k-means is very slow, so I used mini-batch k-means and limited it to 500 components / 30k clusters even though I think more of both may have performed better.
- Text_cluster: same as above, but based on a concatenation of title, description, and the param fields.



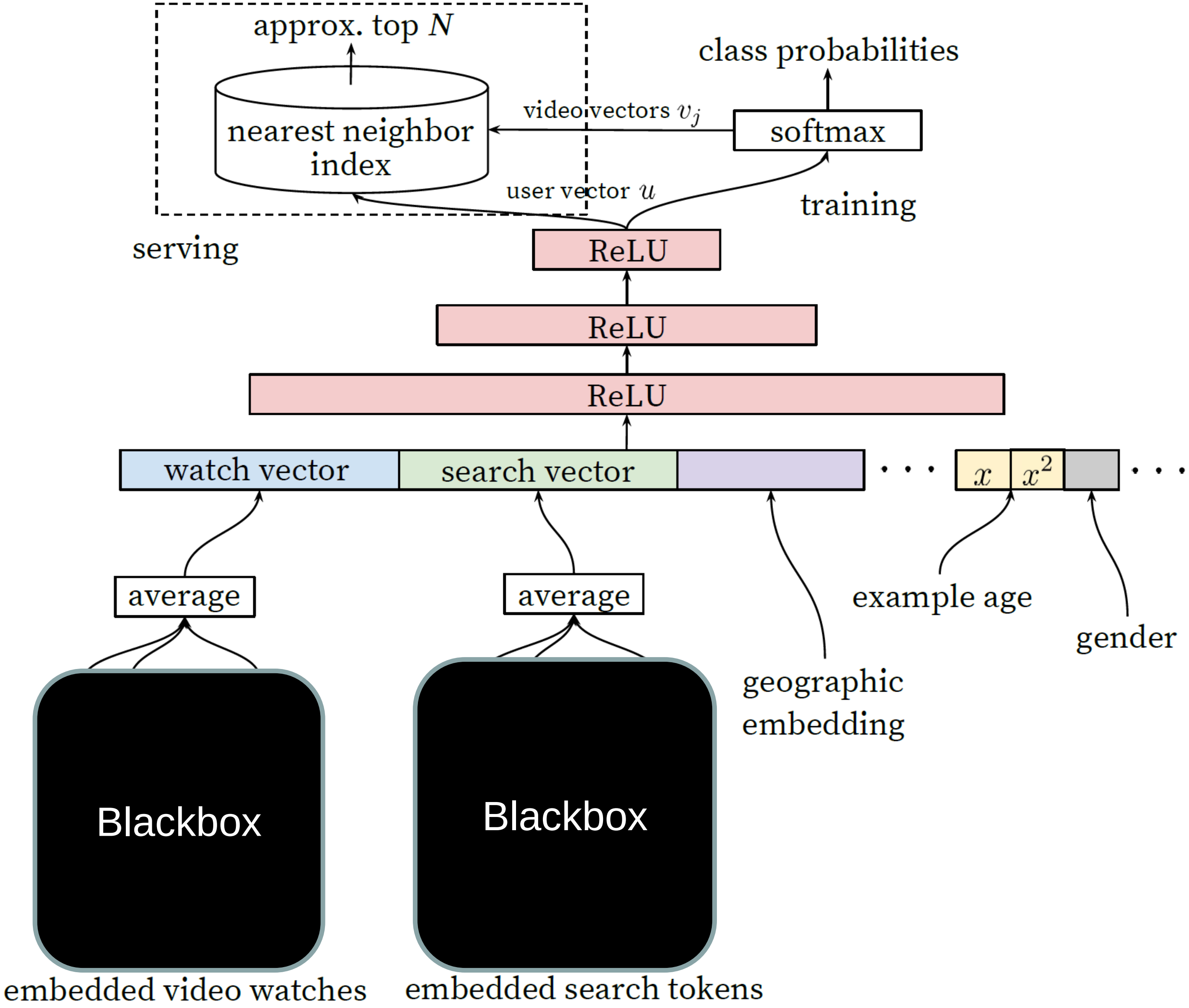
- Image blurness
- Color histograms - SVD components and some statistical features
- NIMA - activations, score, stds, and score + std

Aside from features mentioned, I used a bunch of other miscellaneous features similar to those seen in kernels - simple meta-textual features, lat/long + lat/long clusters, avg days a user's post is active and

Kaggle Avito winner (a true inclusive architecture)



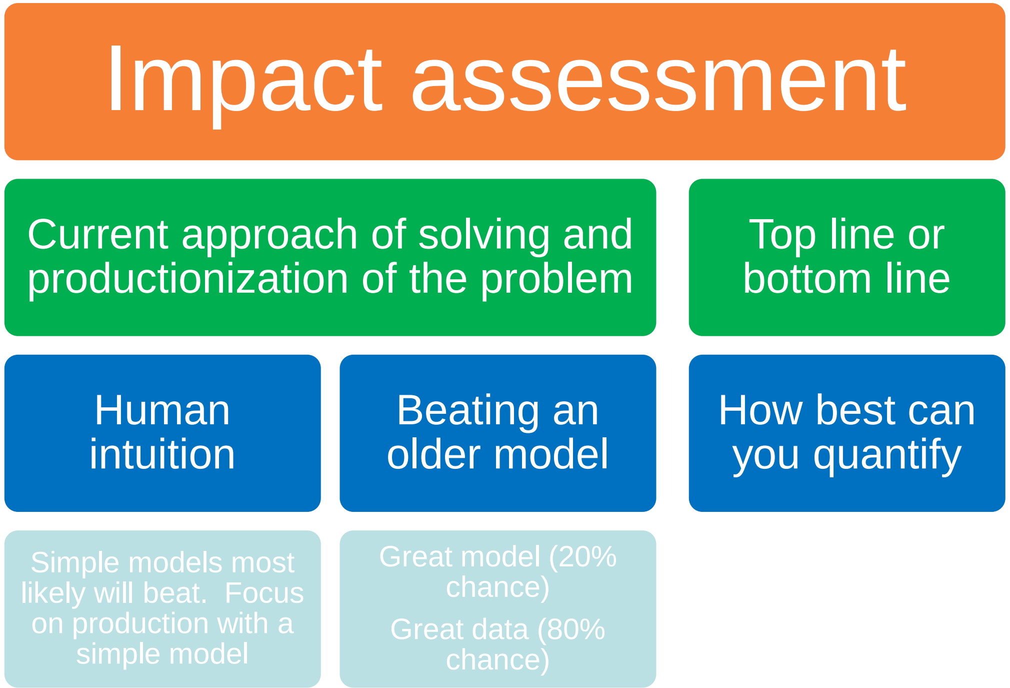
Youtube recommender



Understanding from multiple dimensions

- Business perspective
  - Impact
  - Purpose
- Data perspective
- Design perspective
  - Training
  - Prediction
  - Maintenance
- Algorithm and model perspective

Understanding the problem: Business



Impact assessment

Current approach of solving and productionization of the problem

Top line or bottom line

Human intuition

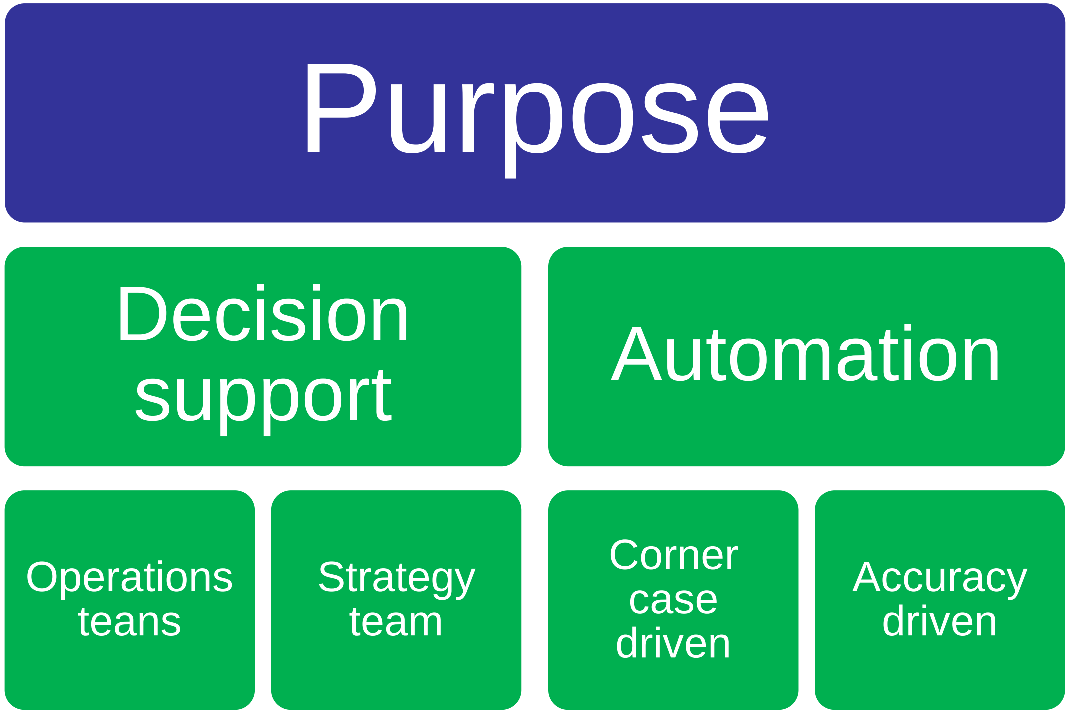
Beating an older model

How best can you quantify

Simple models most likely will beat. Focus on production with a simple model

Great model (20% chance)

Great data (80% chance)



Purpose

Decision support

Automation

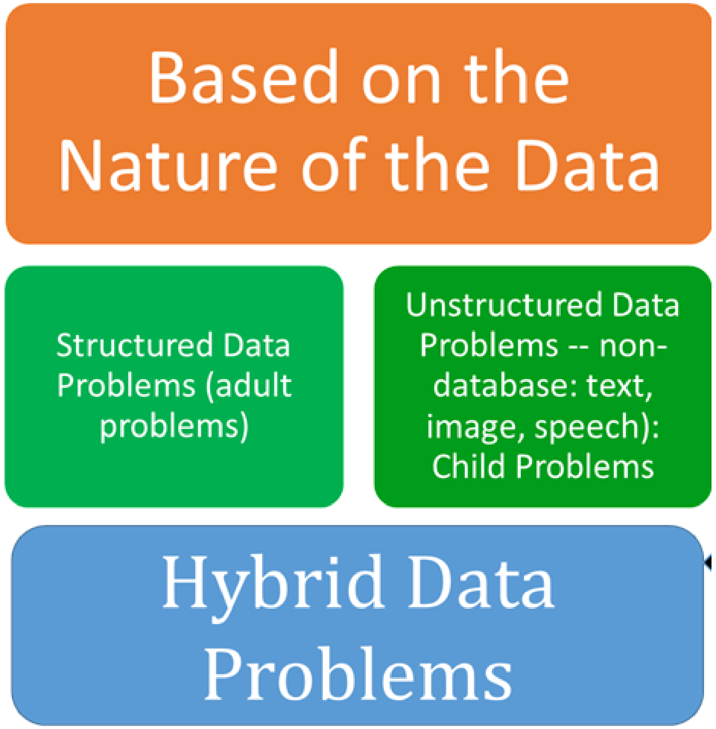
Operations teans

Strategy team

Corner case driven

Accuracy driven

Understanding the Problem: Data perspective

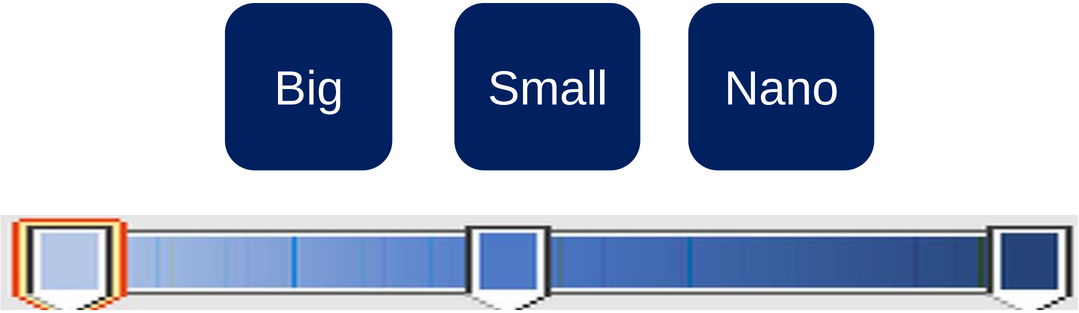


Based on the Nature of the Data

Structured Data Problems (adult problems)

Unstructured Data Problems -- non-database: text, image, speech): Child Problems

Hybrid Data Problems



Big

Small

Nano

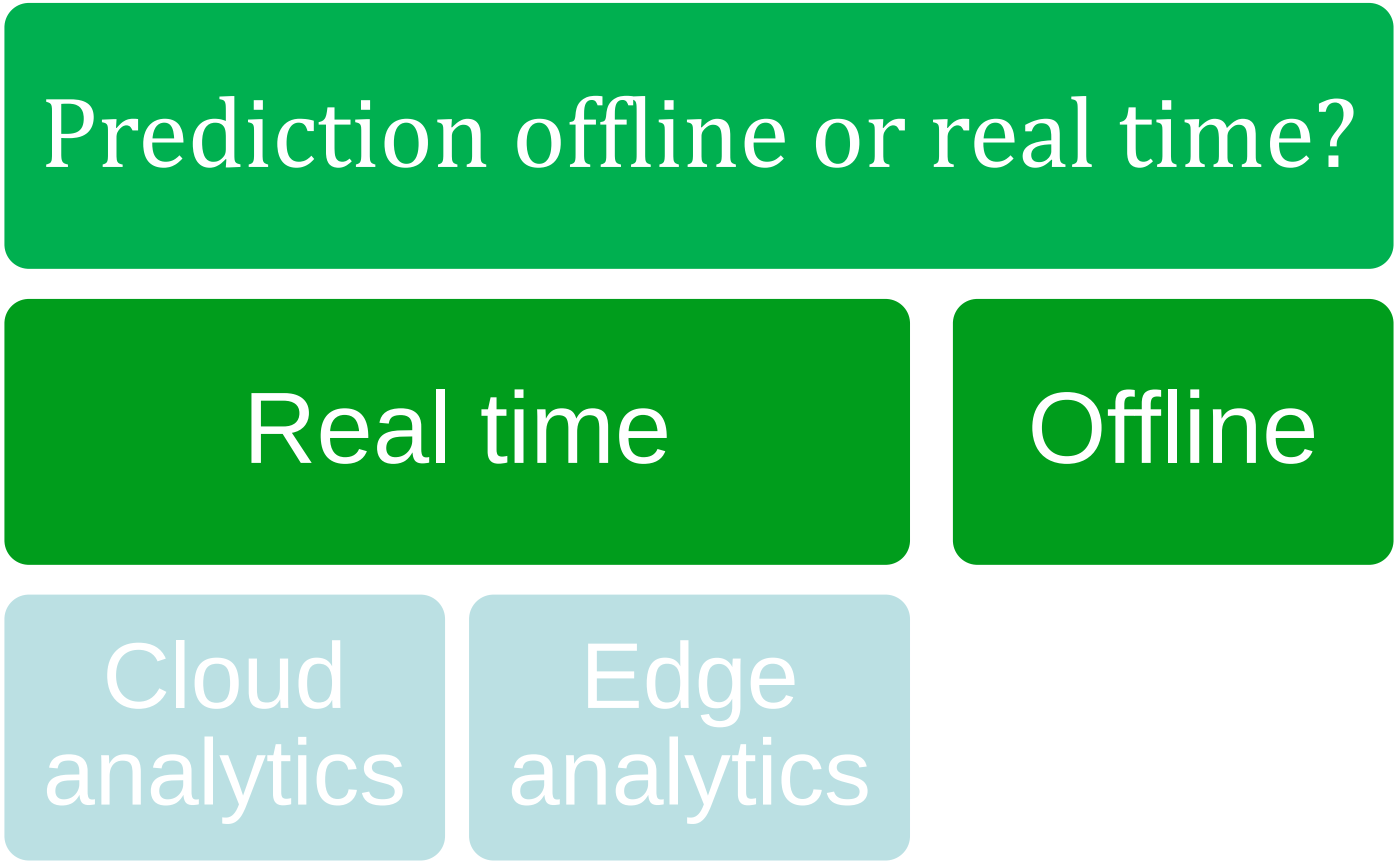
Small or nano data
Not enough for
statistics

Large data
Does not fit
in a machine

Design consideration during training

		Algorithm complexity		
		Linear	Non-Linear	Deeply Non-Linear
Data complexity	Small (less than 100,000)	Desktop/Single server		
	Medium (100K-1M)	Desktop/Single server		GPUs
	Big (>1M)	Spark	GPU/SPARK	

Understanding the problem: Design perspectives during prediction



Prediction offline or real time?

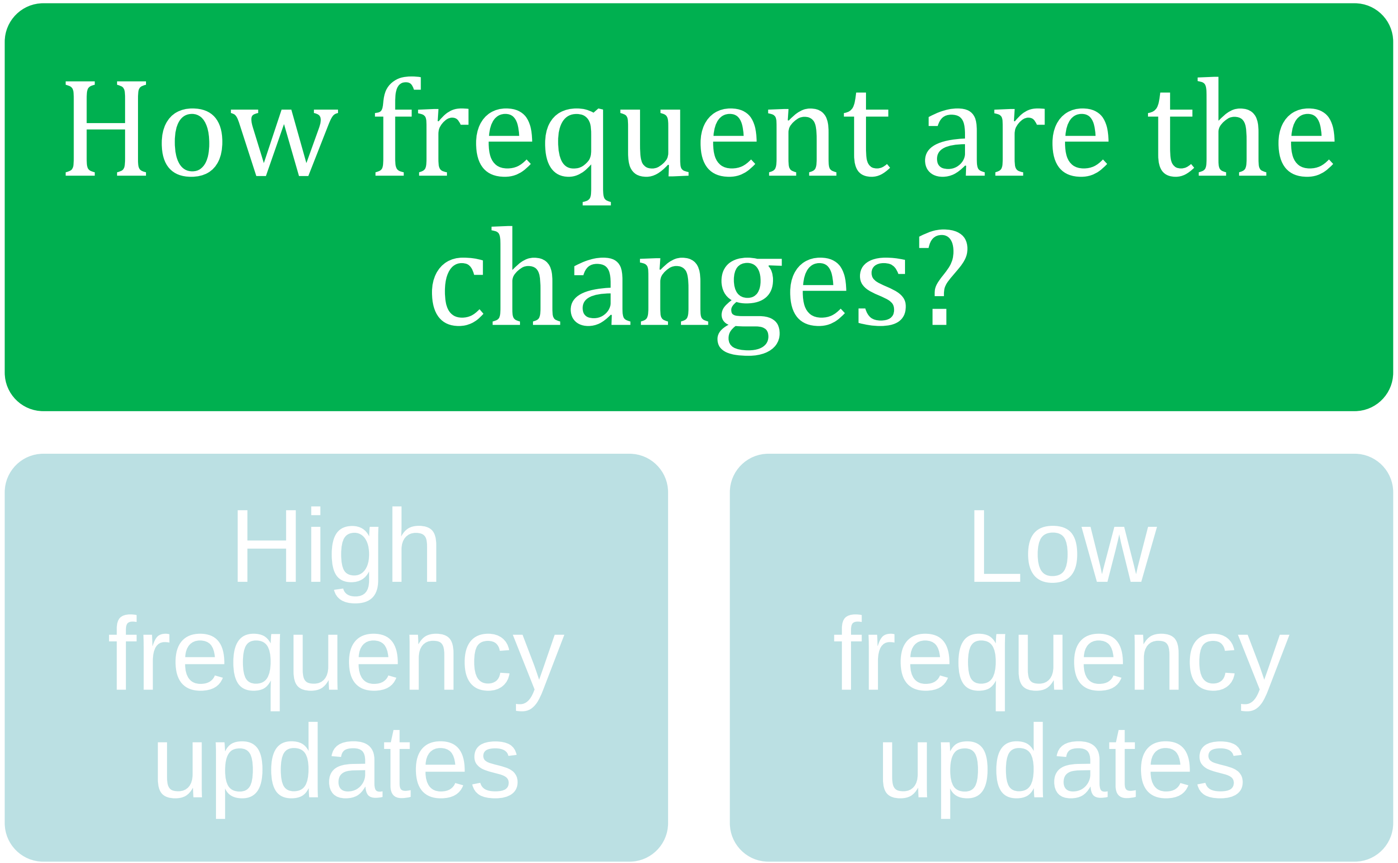
Real time

Offline

Cloud analytics

Edge analytics

Understanding the problem: Design perspectives (from maintenance)

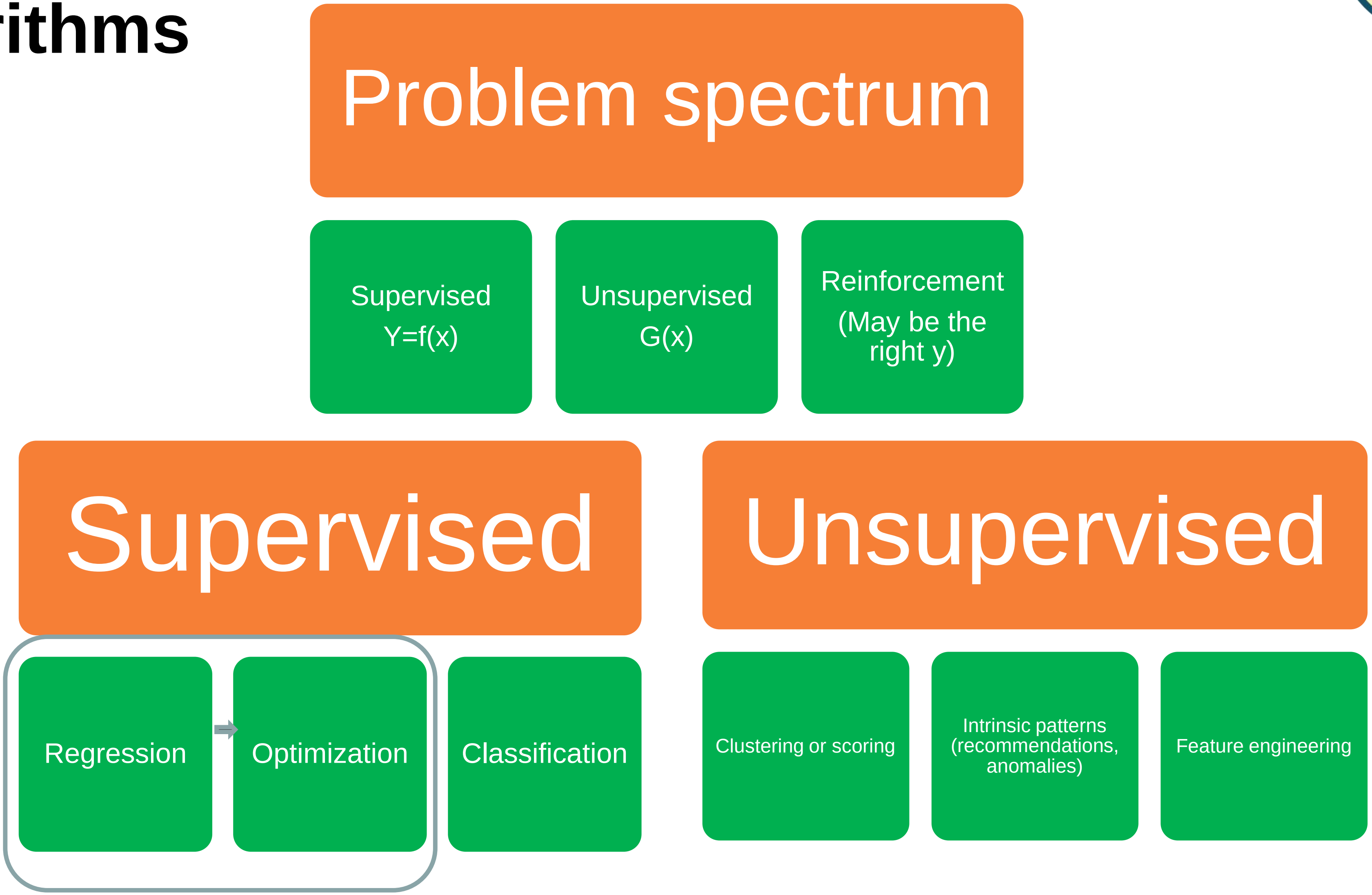


How frequent are the changes?

High frequency updates

Low frequency updates

Algorithms



Problem spectrum

Supervised
Y=f(x)

Unsupervised
G(x)

Reinforcement
(May be the right y)

Supervised

Regression

Optimization

Classification

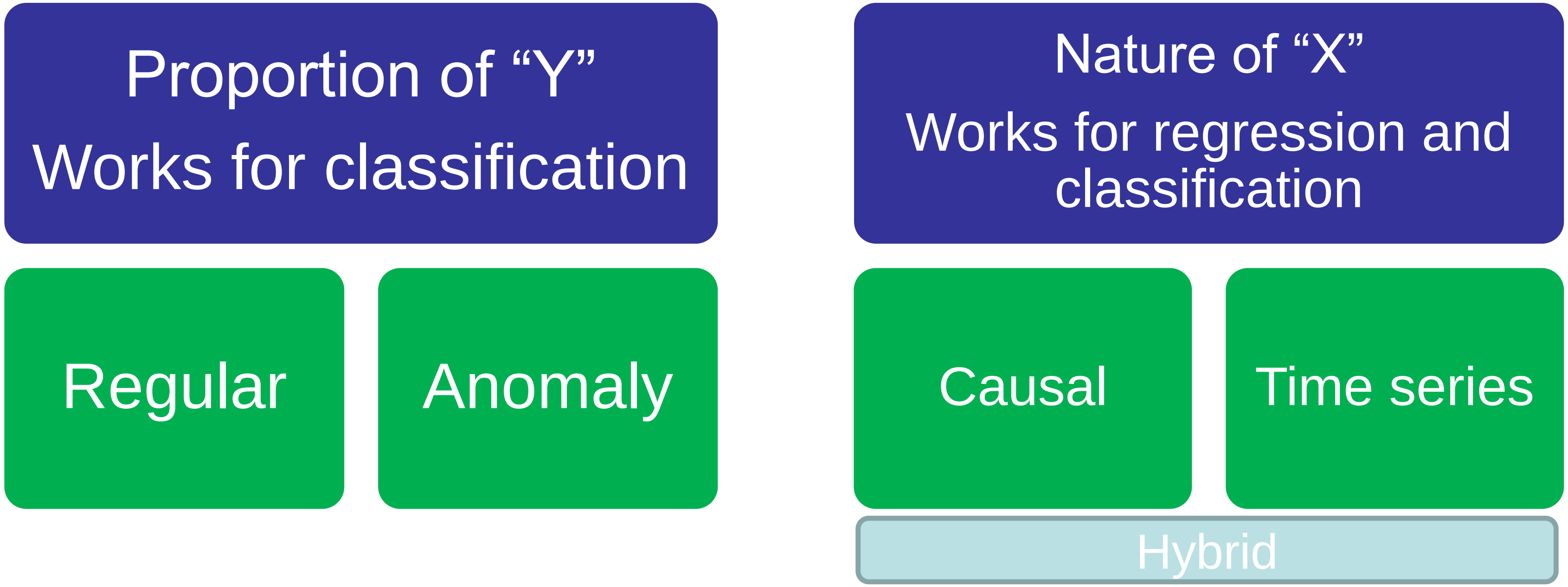
Unsupervised

Clustering or scoring

Intrinsic patterns (recommendations, anomalies)

Feature engineering

Understanding the problem: Algorithm continued



Proportion of “Y”

Works for classification

Regular

Anomaly

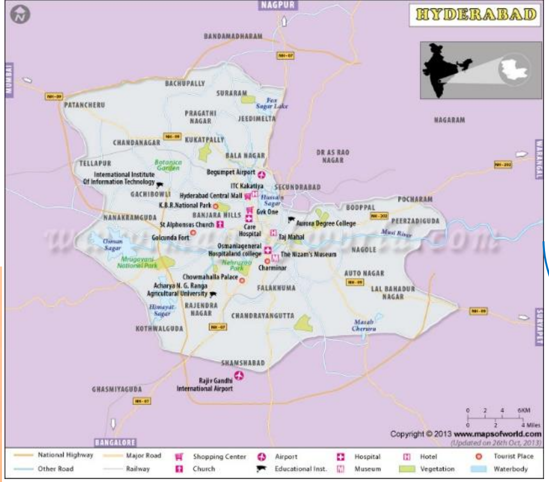
Nature of “X”

Works for regression and classification

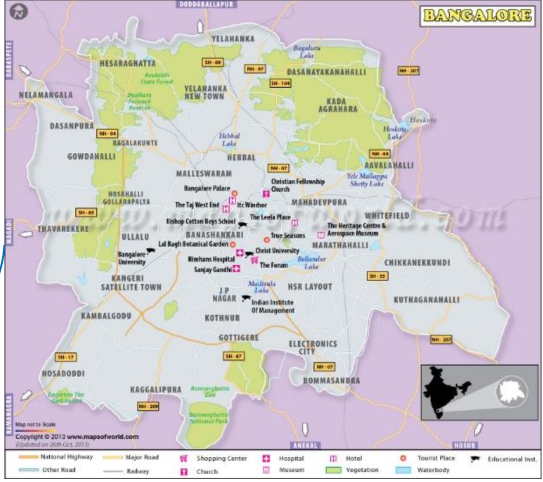
Causal

Time series

Hybrid



Inspire...Educate...Transform.



HYDERABAD

Office and Classrooms

Plot 63/A, Floors 1&2, Road # 13, Film Nagar,
Jubilee Hills, Hyderabad - 500 033
+91-9701685511 (Individuals)
+91-9618483483 (Corporates)

BENGALURU

Office

Incubex, #728, Grace Platina, 4th Floor, CMH Road,
Indira Nagar, 1st Stage, Bengaluru – 560038
+91-9502334561 (Individuals)
+91-9502799088 (Corporates)

Classroom

KnowledgeHut Solutions Pvt. Ltd., Reliable Plaza,
Jakkasandra Main Road, Teacher's Colony, 14th Main
Road, Sector – 5, HSR Layout, Bengaluru - 560102

Social Media

Web:	http://www.insofe.edu.in
Facebook:	https://www.facebook.com/insofe
Twitter:	https://twitter.com/Insofeedu
YouTube:	http://www.youtube.com/InsofeVideos
SlideShare:	http://www.slideshare.net/INSOFE
LinkedIn:	http://www.linkedin.com/company/international-school-of-engineering

This presentation may contain references to findings of various reports available in the public domain. INSOFE makes no representation as to their accuracy or that the organization subscribes to those findings.