CPSC 340:
Machine Learning and Data Mining

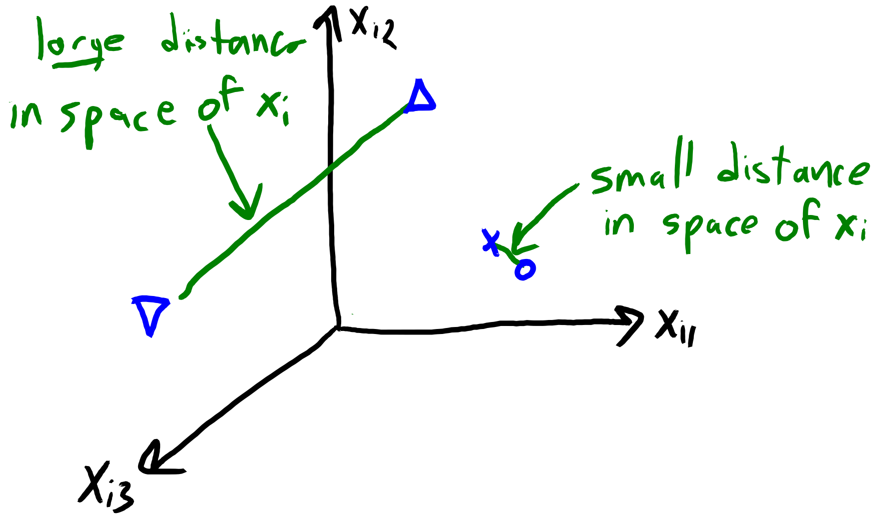
Deep Learning

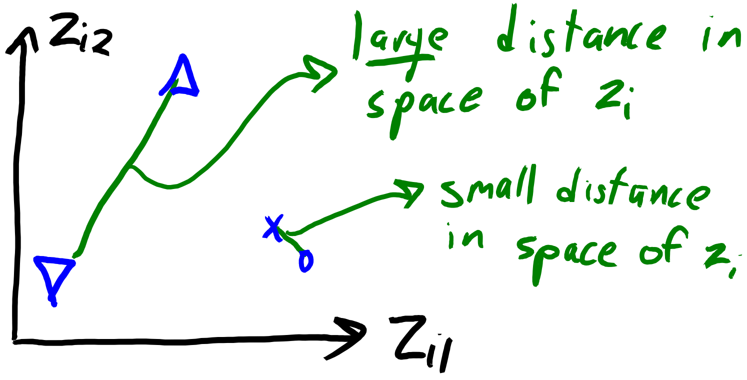
Fall 2018

Last Time: Multi-Dimensional Scaling

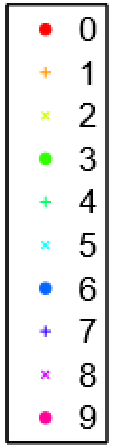
- Multi-dimensional scaling (MDS):
  - Non-parametric visualization: directly optimize the z_i locations.

$$f(Z) = \sum_{i=1}^{n}\sum_{j=i+1}^{n} d_3(d_2(z_i, z_j) - d_1(x_i, x_j))$$

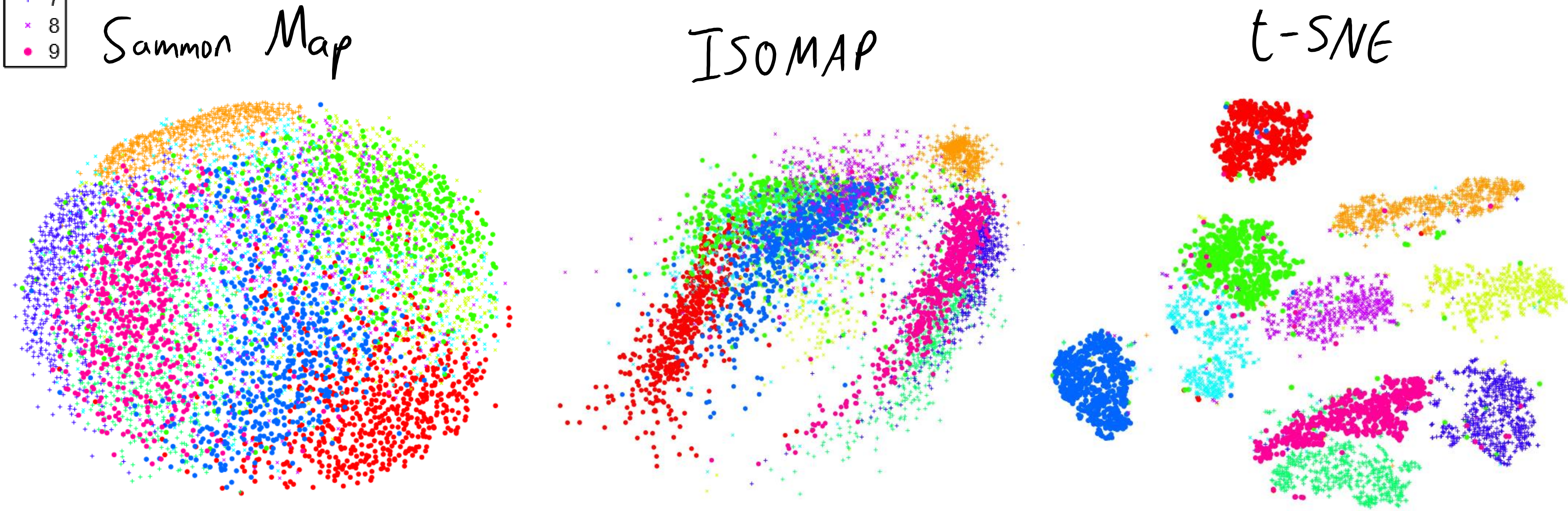




  - Traditional MDS methods lead to a "crowding" effect.

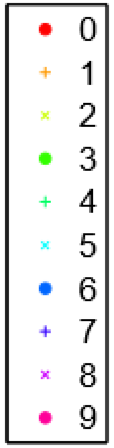


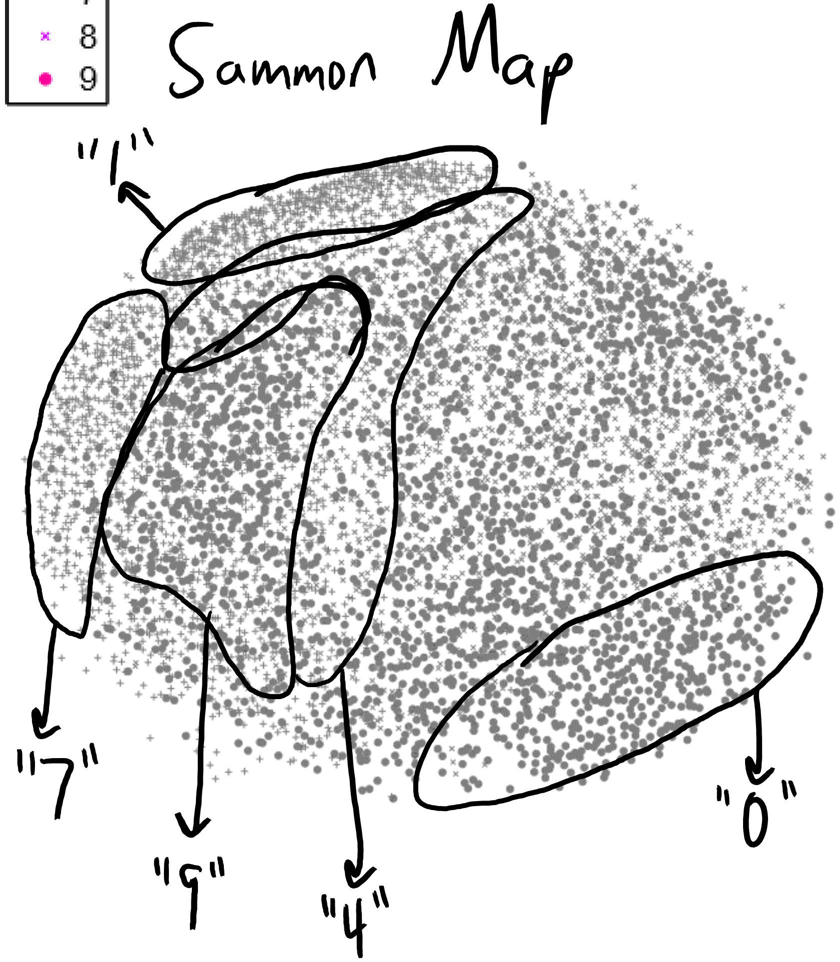
Sammon's Map vs. ISOMAP vs. t-SNE

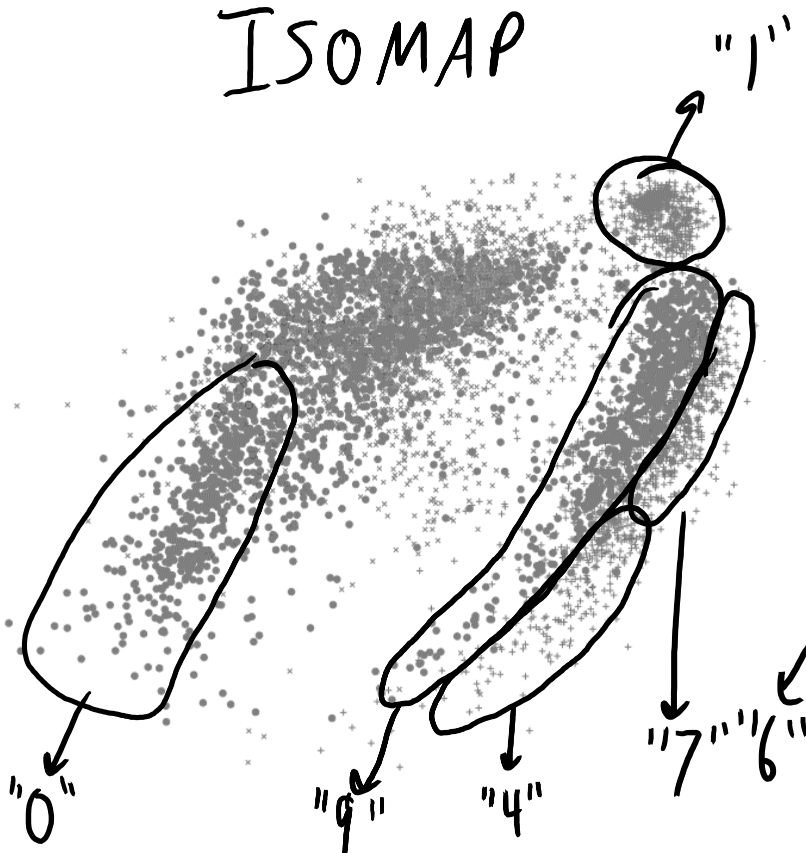


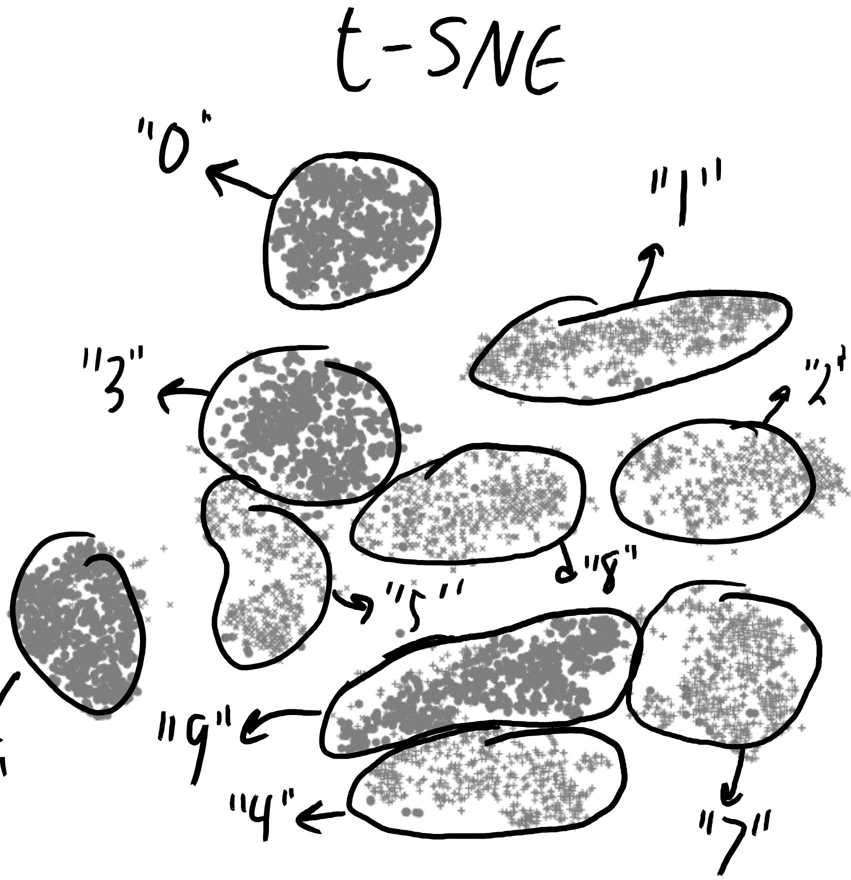
http://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf

Sammon's Map vs. ISOMAP vs. t-SNE







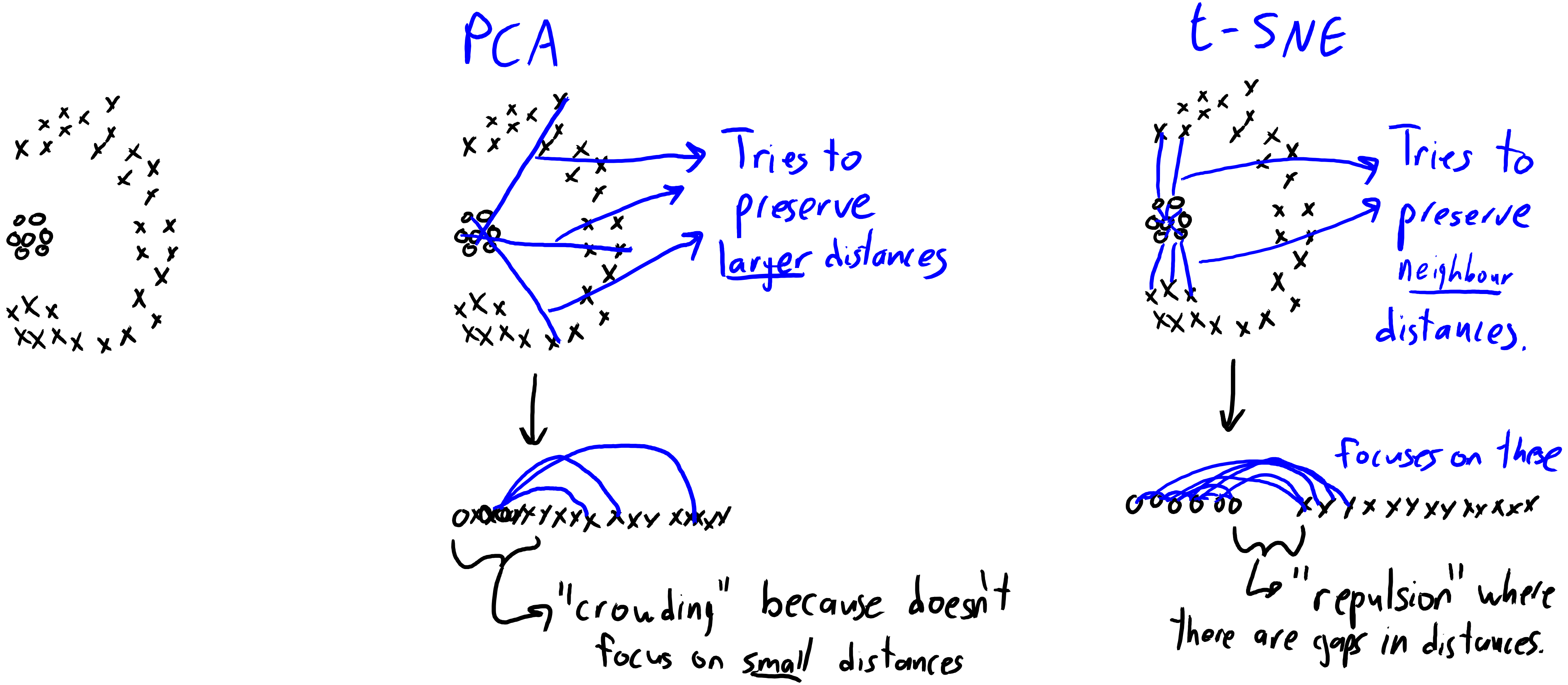


Remember this is unsupervised, algorithms do not know the labels.

http://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf

t-Distributed Stochastic Neighbour Embedding

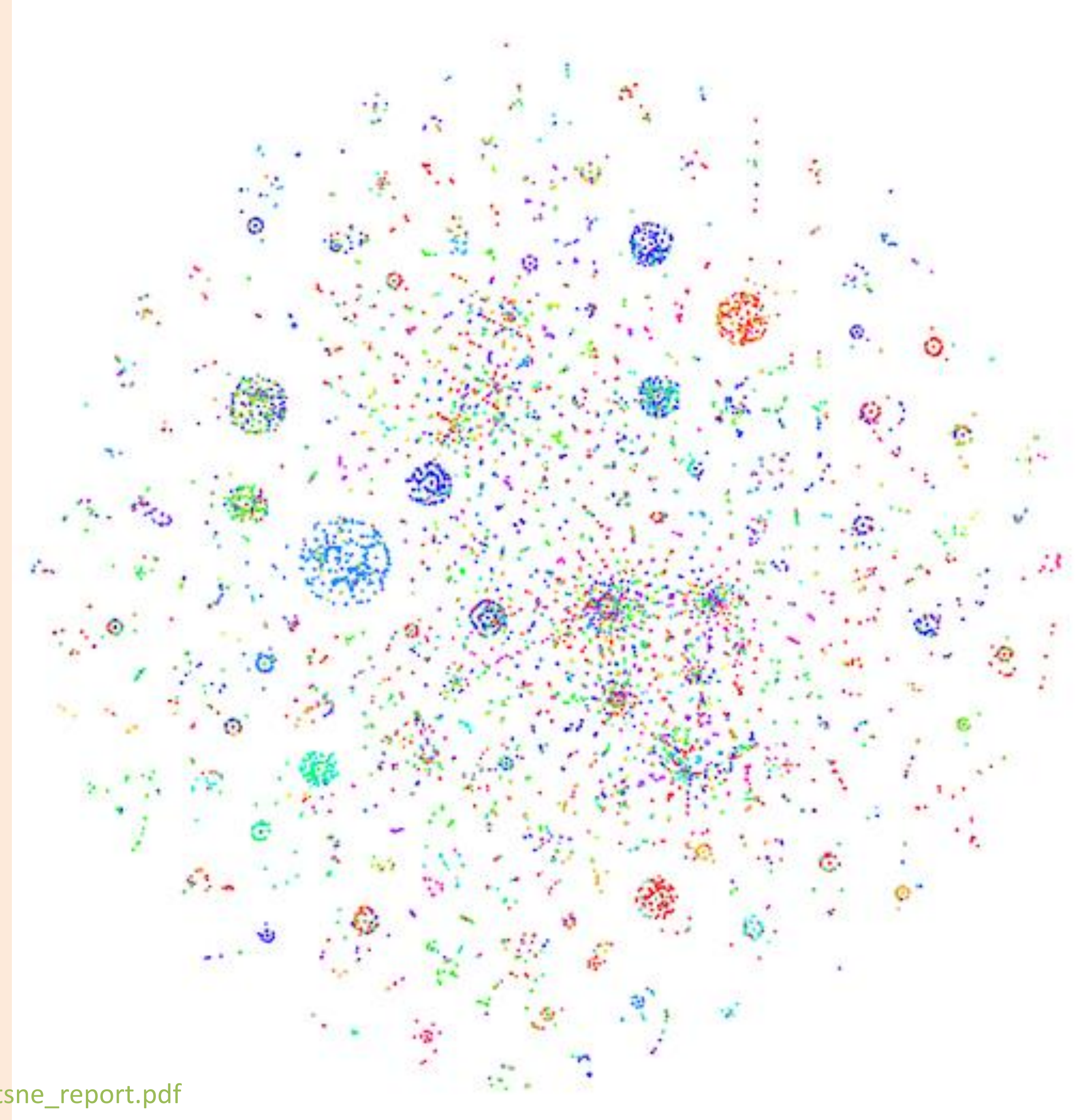
- One key idea in t-SNE:
  - Focus on distance to “neighbours”(allow large variance in other distances)



t-Distributed Stochastic Neighbour Embedding

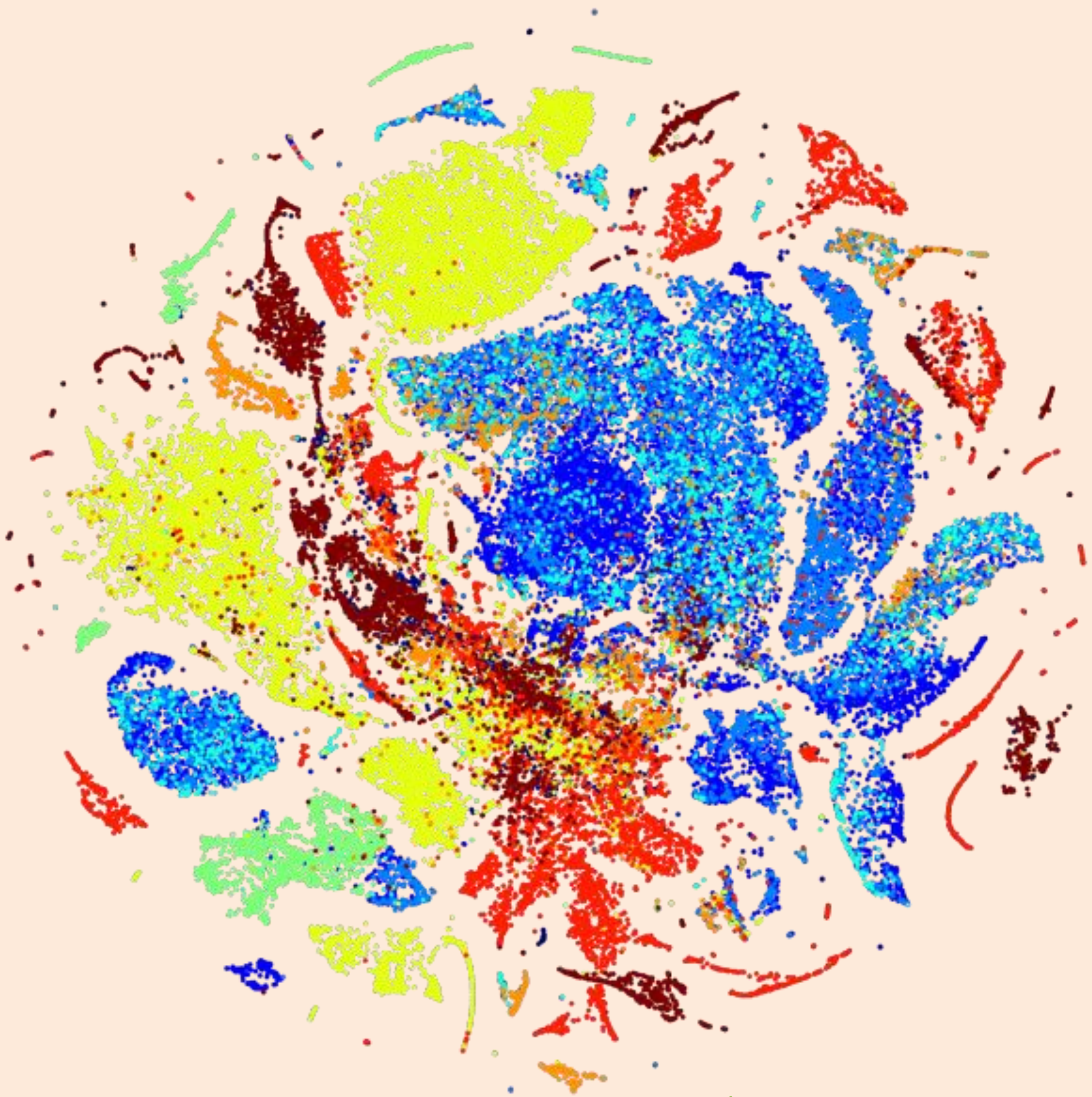
- t-SNE is a special case of MDS (specific d_1, d_2, and d_3 choices):
  - d_1: for each x_i, compute probability that each x_j is a 'neighbour'.
    - Computation is similar to k-means++, but most weight to close points (Gaussian).
    - Doesn't require explicit graph.
  - d_2: for each z_i, compute probability that each z_j is a 'neighbour'.
    - Similar to above, but uses student's t (grows really slowly with distance).
    - Avoids 'crowding', because you have a huge range that large distances can fill.
  - d_3: Compare x_i and z_i using an entropy-like measure:
    - How much 'randomness' is in probabilities of x_i if you know the z_i (and vice versa)?
- Interactive demo: https://distill.pub/2016/misread-tsne

t-SNE on Wikipedia Articles



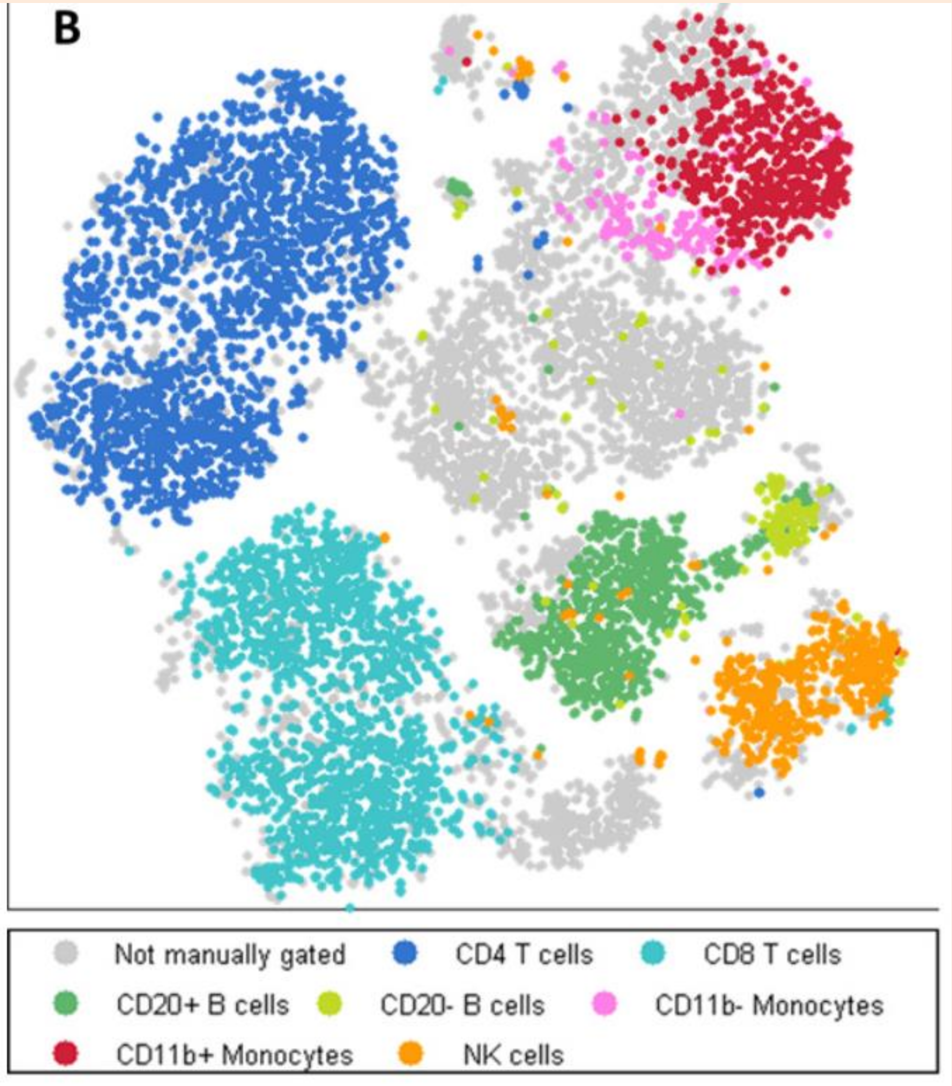
http://jasneetsabharwal.com/assets/files/wiki_tsne_report.pdf

t-SNE on Product Features



http://blog.kaggle.com/2015/06/09/otto-product-classification-winners-interview-2nd-place-alexander-guschin/

t-SNE on Leukemia Heterogeneity



http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4076922/

(pause)

Latent-Factor Representation of Words

- For natural language, we often represent words by an index.
  - E.g., “cat” is word 124056 among a “bag of words”.

- But this may be inefficient:
  - Should “cat” and “kitten” share parameters in some way?

- We want a latent-factor representation of individual words:
  - Closeness in latent space should indicate similarity.
  - Distances could represent meaning?

- Recent alternative to PCA/NMF is word2vec…

Using Context

- Consider these phrases:
  - “the cat purred”
  - “the kitten purred”

  - “black cat ran”
  - “black kitten ran”

- Words that occur in the same context likely have similar meanings.

- Word2vec uses this insight to design an MDS distance function.

Word2Vec

- Two common word2vec approaches:
  1. Try to predict word from surrounding words (continuous bag of words).
  2. Try to predict surrounding words from word (skip-gram).

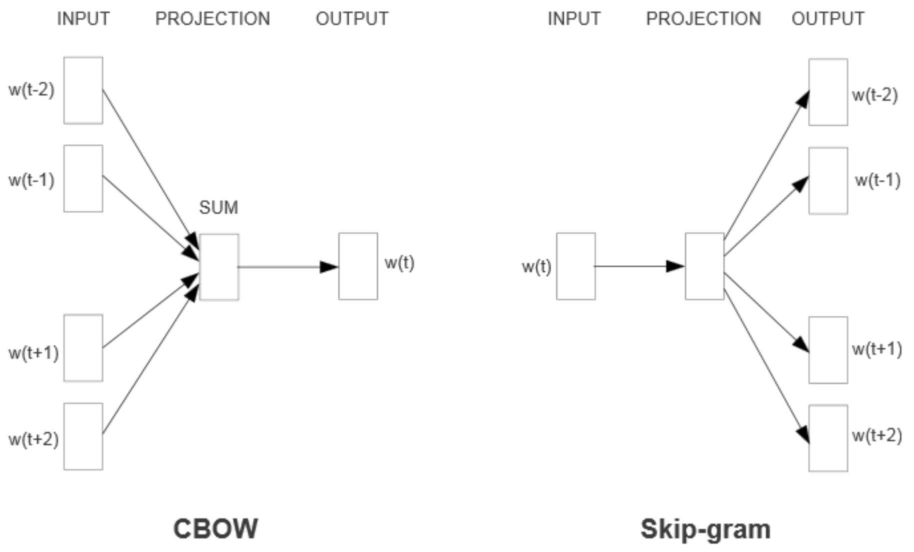


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

- Train latent-factors to solve one of these supervised learning tasks.

https://arxiv.org/pdf/1301.3781.pdf

Word2Vec

- In both cases, each word 'i' is represented by a vector z_i.
- In continuous bag of words (CBOW), we optimize the following likelihood:

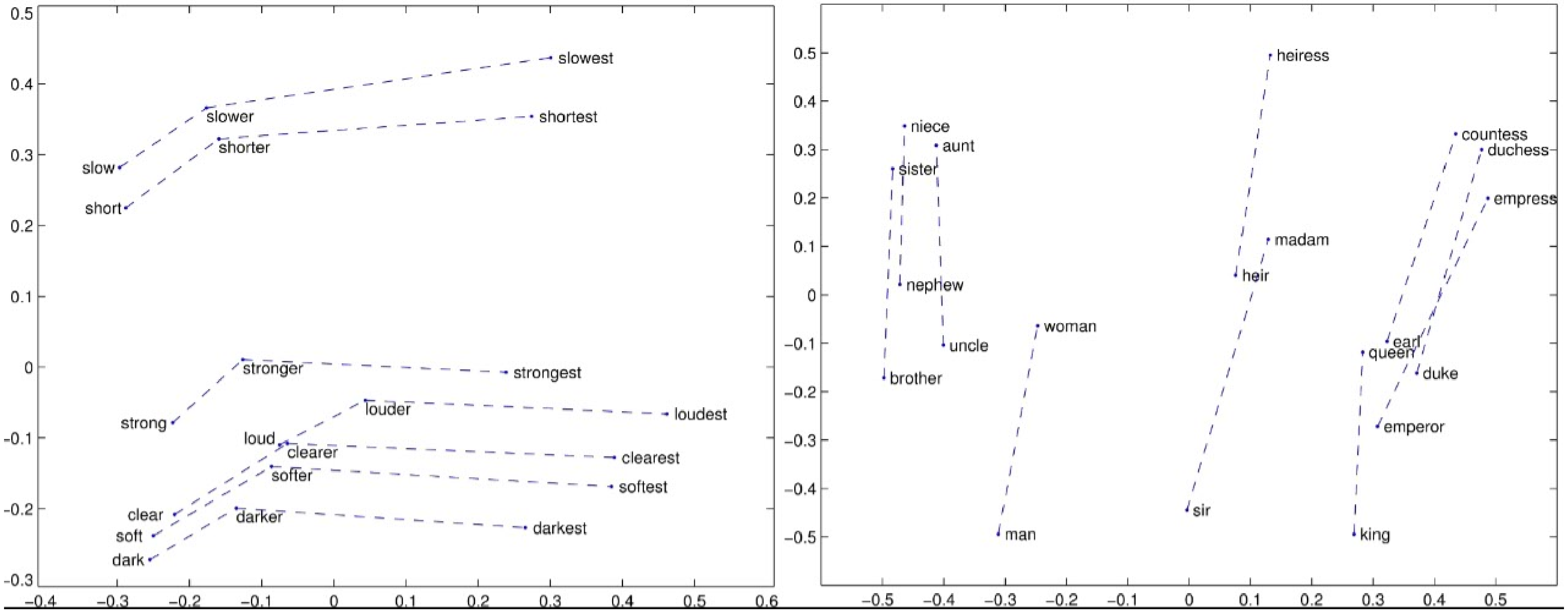
$$p(x_i \mid x_{surround}) = \prod_{j \in surround} p(x_i \mid x_j) \qquad \text{(independence assumption)}$$

$$= \prod_{j \in surround} \frac{\exp(z_i^T z_j)}{\sum_{c=1}^{k} \exp(z_c^T z_j)} \qquad \text{(softmax over all words)}$$

- Apply gradient descent to logarithm:
  - Encourages $z_i^T z_j$ to be big for words in same context (making z_i close to z_1).
  - Encourages $z_i^T z_j$ to be small for words not appearing in same context (makes z_i and z_j far).
- For CBOW, denominator sums over all words.
- For skip-gram it will be over all possible surrounding words.
  - Common trick to speed things up: sample terms in denominator ("negative sampling").

Word2Vec Example

- MDS visualization of a set of related words:



- Distances between vectors might represent semantics.

http://sebastianruder.com/secret-word2vec

Word2Vec

- Subtracting word vectors to find related vectors.

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Table 8 shows words that follow various relationships. We follow the approach described above: the relationship is defined by subtracting two word vectors, and the result is added to another word. Thus for example, *Paris - France + Italy = Rome*. As it can be seen, accuracy is quite good, although

- Word vectors for 157 languages here.

https://arxiv.org/pdf/1301.3781.pdf

End of Part 4: Key Concepts

- We discussed linear latent-factor models:

$$f(W,Z) = \sum_{i=1}^{n} \sum_{j=1}^{d} (\langle w^j, z_i \rangle - x_{ij})^2$$
$$= \sum_{i=1}^{n} \|W^T z_i - x_i\|^2$$
$$= \|ZW - X\|_F^2$$

- Represent 'X' as linear combination of latent factors 'w_c'.
  - Latent features 'z_i' give a lower-dimensional version of each 'x_i'.
  - When k=1, finds direction that minimizes squared orthogonal distance.
- Applications:
  - Outlier detection, dimensionality reduction, data compression, features for linear models, visualization, factor discovery, filling in missing entries.

End of Part 4: Key Concepts

- We discussed linear latent-factor models:

$$f(W,Z) = \sum_{i=1}^{n} \sum_{j=1}^{d} (\langle w^j, z_i \rangle - x_{ij})^2$$

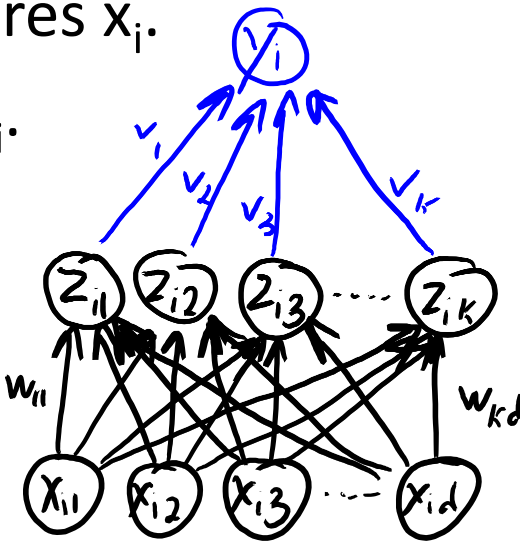
- Principal component analysis (PCA):
  - Often uses orthogonal factors and fits them sequentially (via SVD).
- Non-negative matrix factorization:
  - Uses non-negative factors giving sparsity.
  - Can be minimized with projected gradient.
- Many variations are possible:
  - Different regularizers (sparse coding) or loss functions (robust/binary PCA).
  - Missing values (recommender systems) or change of basis (kernel PCA).

End of Part 4: Key Concepts

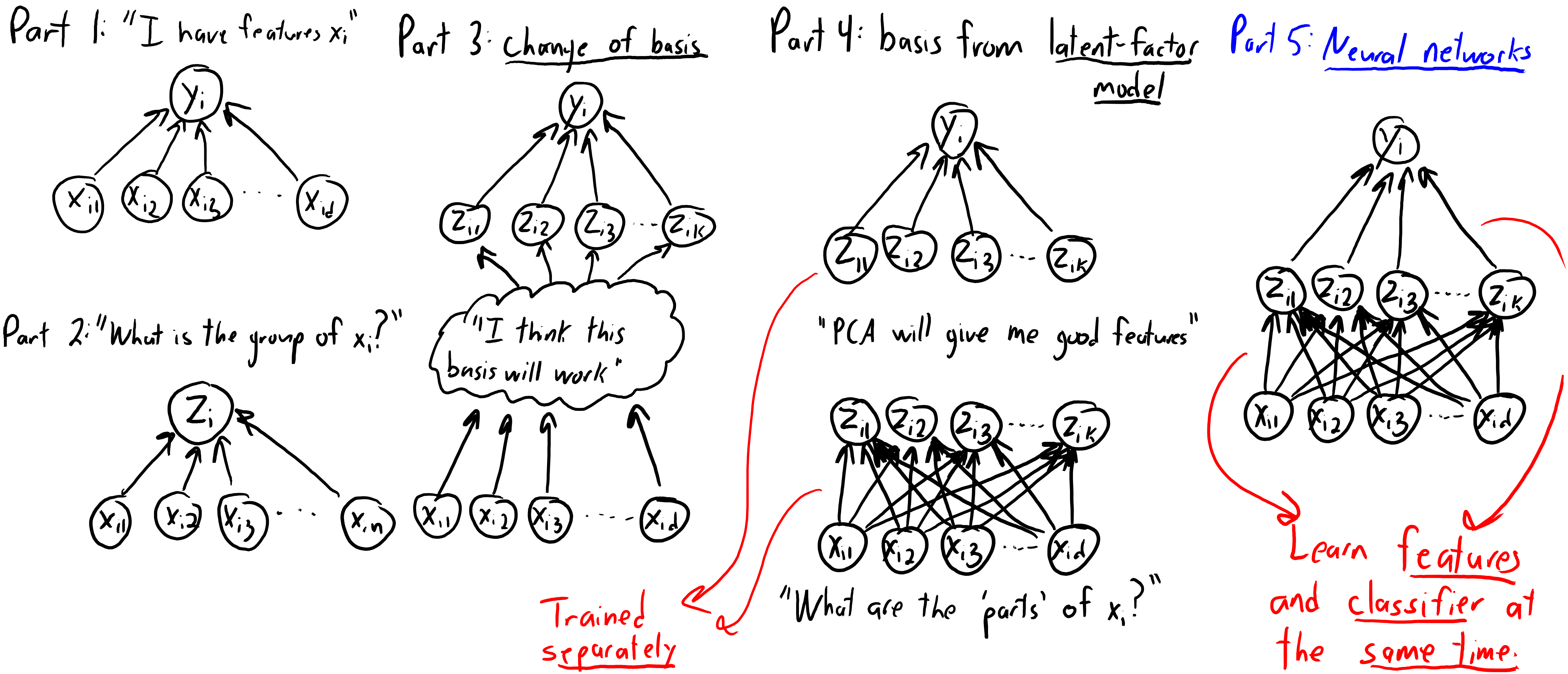
- We discussed multi-dimensional scaling (MDS):
  - Non-parametric method for high-dimensional data visualization.
  - Tries to match distance/similarity in high-/low-dimensions.
    - "Gradient descent on scatterplot points".
- Main challenge in MDS methods is "crowding" effect:
  - Methods focus on large distances and lose local structure.
- Common solutions:
  - Sammon mapping: use weighted cost function.
  - ISOMAP: approximate geodesic distance using via shortest paths in graph.
  - T-SNE: give up on large distances and focus on neighbour distances.
- Word2vec is a recent MDS method giving better "word features".

Supervised Learning Roadmap

- Part 1: “Direct” Supervised Learning.
  - We learned parameters ‘w’ based on the original features x_i and target y_i.
- Part 3: Change of Basis.
  - We learned parameters ‘v’ based on a change of basis z_i and target y_i.
- Part 4: Latent-Factor Models.
  - We learned parameters ‘W’ for basis z_i based on only on features x_i.
  - You can then learn ‘v’ based on change of basis z_i and target y_i.
- Part 5: Neural Networks.
  - Jointly learn ‘W’ and ‘v’ based on x_i and y_i.
  - **Learn basis z_i that is good for supervised learning.**



A Graphical Summary of CPSC 340 Parts 1-5



Notation for Neural Networks

We have our usual supervised learning notation:

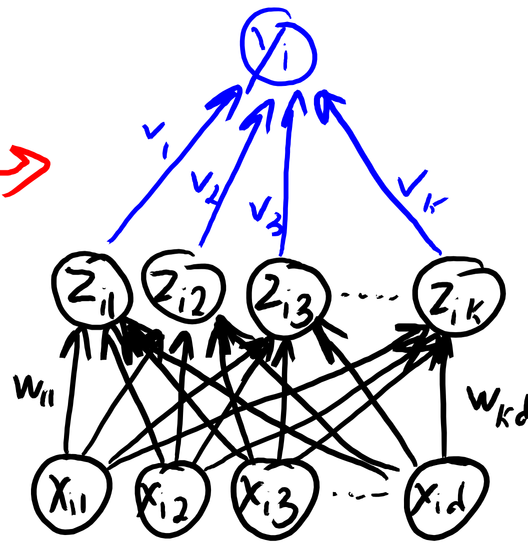
$$X = \underbrace{\begin{bmatrix} \text{—} & x_1^T & \text{—} \\ \text{—} & x_2^T & \text{—} \\ & \vdots & \\ \text{—} & x_n^T & \text{—} \end{bmatrix}}_{n \times d} \qquad y = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{n \times 1}$$

We have our latent features:

$$Z = \underbrace{\begin{bmatrix} \text{—} & z_1^T & \text{—} \\ \text{—} & z_2^T & \text{—} \\ & \vdots & \\ \text{—} & z_n^T & \text{—} \end{bmatrix}}_{n \times k}$$

We have <u>two</u> sets of parameters:

$$v = \underbrace{\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}}_{k \times 1} \qquad W = \underbrace{\begin{bmatrix} \text{—} & w_1 & \text{—} \\ \text{—} & w_2 & \text{—} \\ & \vdots & \\ \text{—} & w_k & \text{—} \end{bmatrix}}_{k \times d}$$



Linear-Linear Model

- Obvious choice: linear latent-factor model with linear regression.

  Use features from latent-factor model: $z_i = Wx_i$

  Make predictions using a linear model: $y_i = v^T z_i$

- We want to train 'W' and 'v' jointly, so we could minimize:

$$f(W,v) = \frac{1}{2}\sum_{i=1}^{n}(\underbrace{v^T z_i - y_i}_{\text{linear regression with } z_i \text{ as features}})^2 = \frac{1}{2}\sum_{i=1}^{n}(v^T(\underbrace{Wx_i}_{z_i \text{ come from latent-factor model}}) - y_i)^2$$

- But this is just a linear model: $\hat{y}_i = v^T z_i = v^T(Wx_i) = \underbrace{\overbrace{(v^T W)}^{1 \times d}}_{\text{some vector 'w'}} x_i = w^T x_i$

Introducing Non-Linearity

- To increase flexibility, something needs to be non-linear.
- Typical choice: transform z_i by non-linear function 'h'.

$$z_i = Wx_i \qquad\qquad y_i = v^T h(z_i)$$

  – Here the function 'h' transforms 'k' inputs to 'k' outputs.

- Common choice for 'h': applying sigmoid function element-wise:

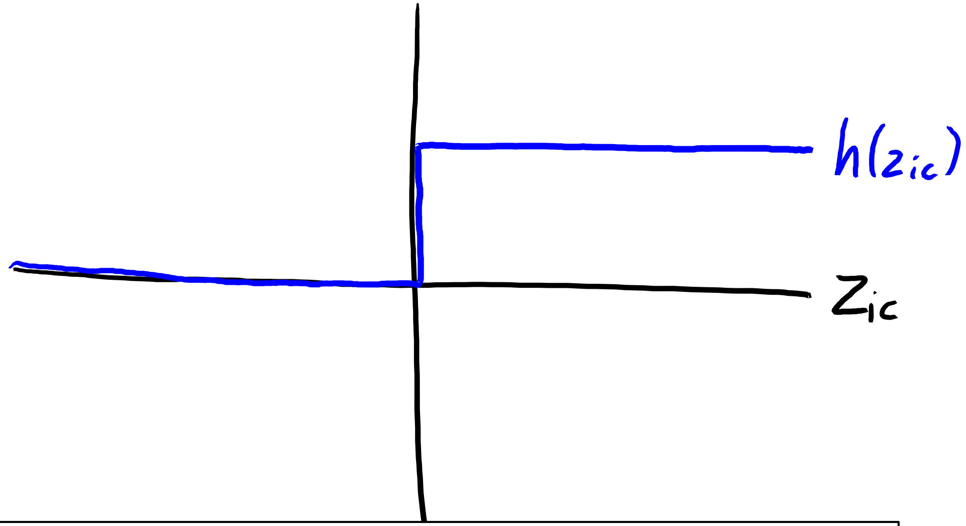
$$h(z_{ic}) = \frac{1}{1 + \exp(-z_{ic})}$$

- So this takes the z_{ic} in $(-\infty,\infty)$ and maps it to $(0,1)$.
- This is called a "multi-layer perceptron" or a "neural network".

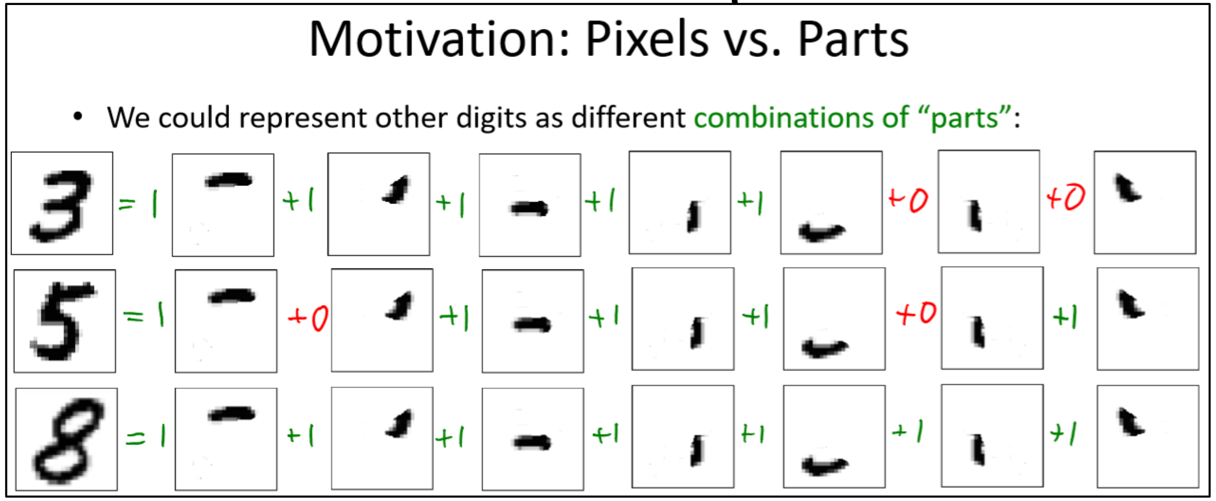
Why Sigmoid?

- Consider setting 'h' to define binary features z_i using:

$$h(z_{ic}) = \begin{cases} 1 & \text{if } z_{ic} \geq 0 \\ 0 & \text{if } z_{ic} < 0 \end{cases}$$



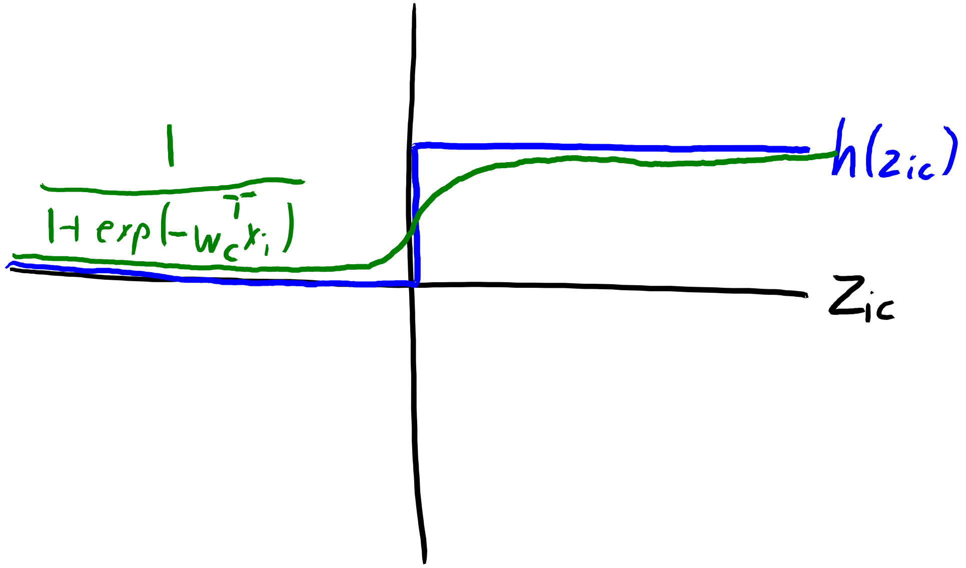
  – Each h(zi) can be viewed as binary feature.
    - "You either have this 'part' or you don't have it."
  – We can make 2^k objects by all the possible "part combinations".



Why Sigmoid?

- Consider setting ‘h’ to define binary features z_i using:

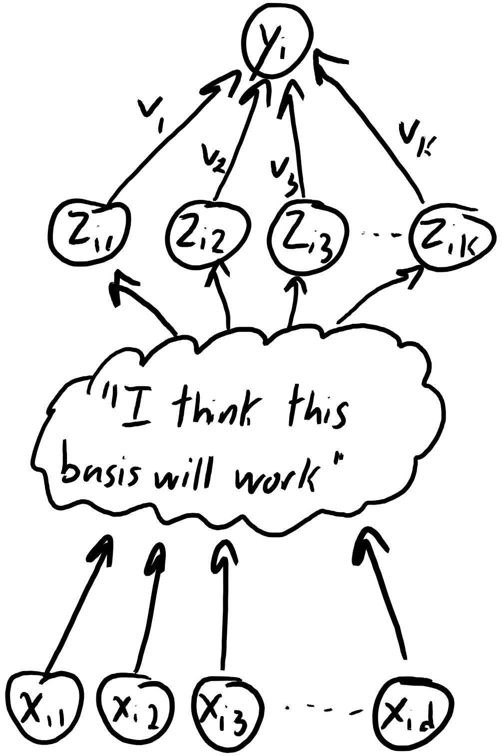
$$h(z_{ic}) = \begin{cases} 1 & \text{if } z_{ic} \geq 0 \\ 0 & \text{if } z_{ic} < 0 \end{cases}$$



  – Each h(zi) can be viewed as binary feature.
    - “You either have this ‘part’ or you don’t have it.”
  – We can make 2^k objects by all the possible “part combinations”.
- But this is hard to optimize (non-differentiable/discontinuous).
- Sigmoid is a smooth approximation to these binary features.

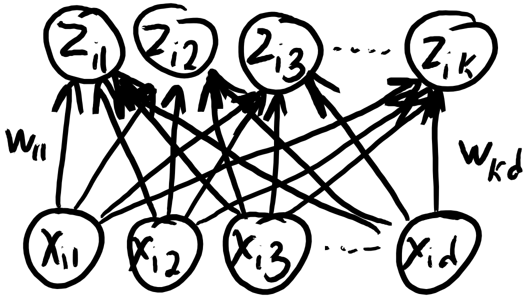
Supervised Learning Roadmap

Hand-engineered features:

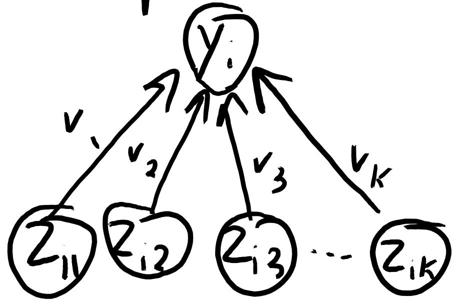


Requires domain knowledge and can be time-consuming

Learn a latent-factor model:

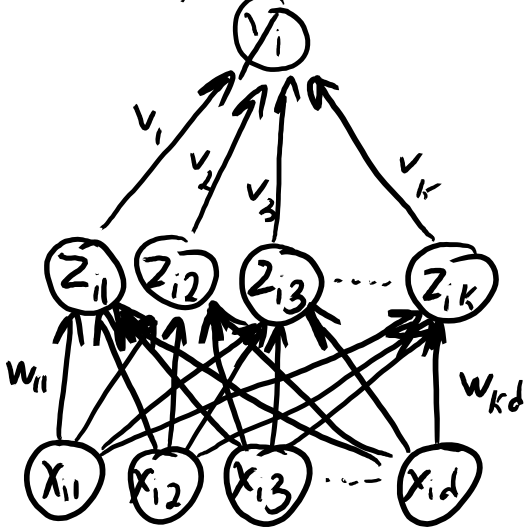


Use latent features in supervized model:



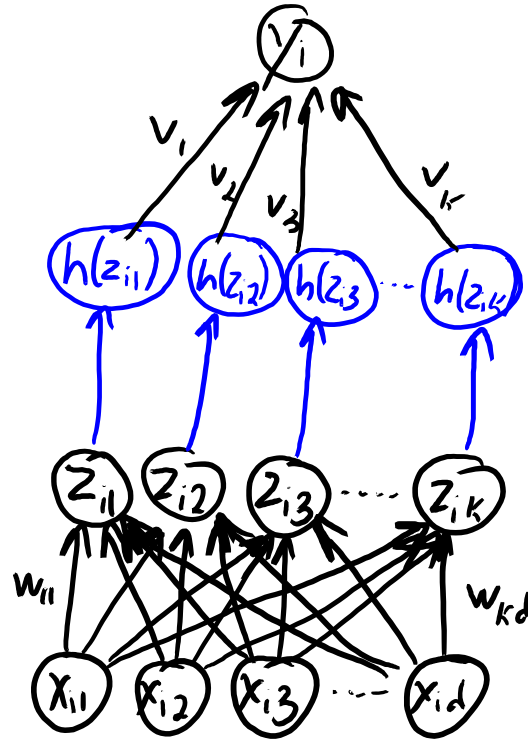
Good representation of x_i might be bad for predicting y_i

Learn 'w' and 'W' together.



But still gives a linear model.

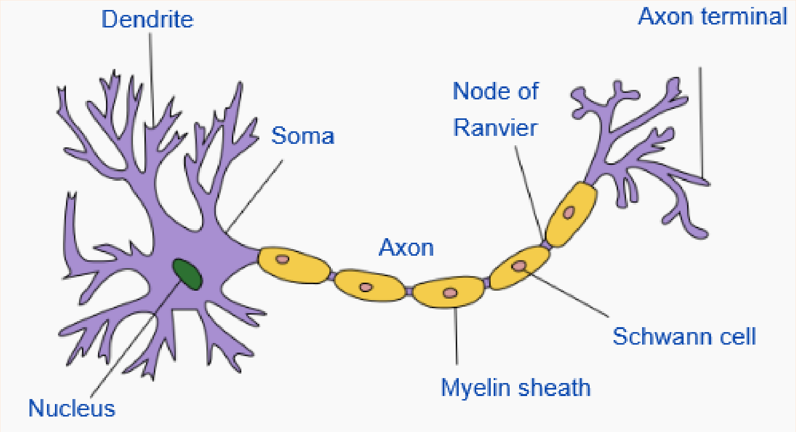
Neural network:



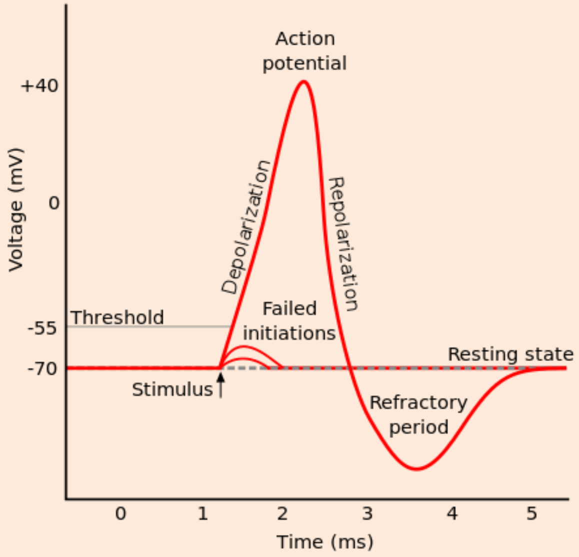
Extra non-linear transformation 'h'.

Why “Neural Network”?

- Cartoon of “typical” neuron:

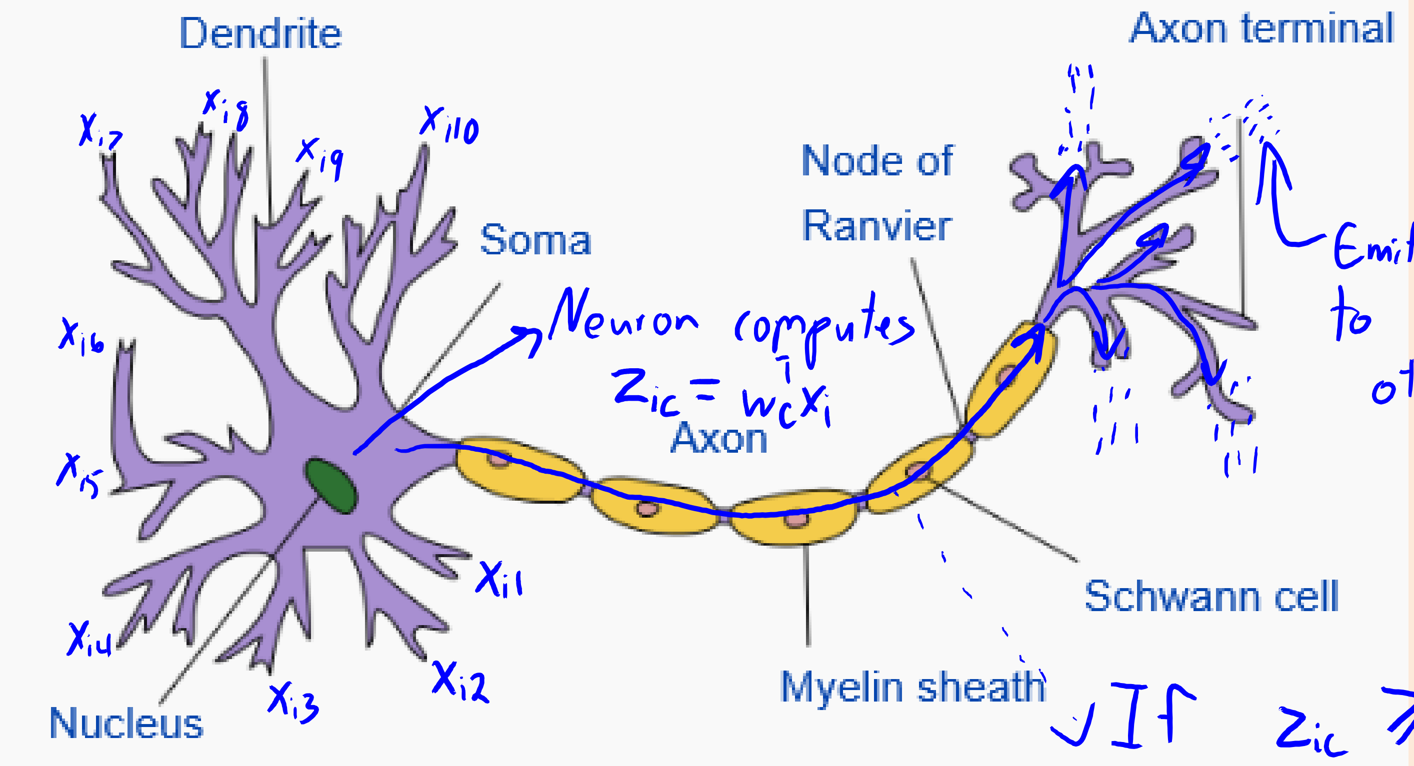


- Neuron has many “dendrites”, which take an input signal.
- Neuron has a single “axon”, which sends an output signal.
- With the right input to dendrites:
  - “Action potential” along axon (like a binary signal):



https://en.wikipedia.org/wiki/Neuron
https://en.wikipedia.org/wiki/Action_potential

Why "Neural Network"?

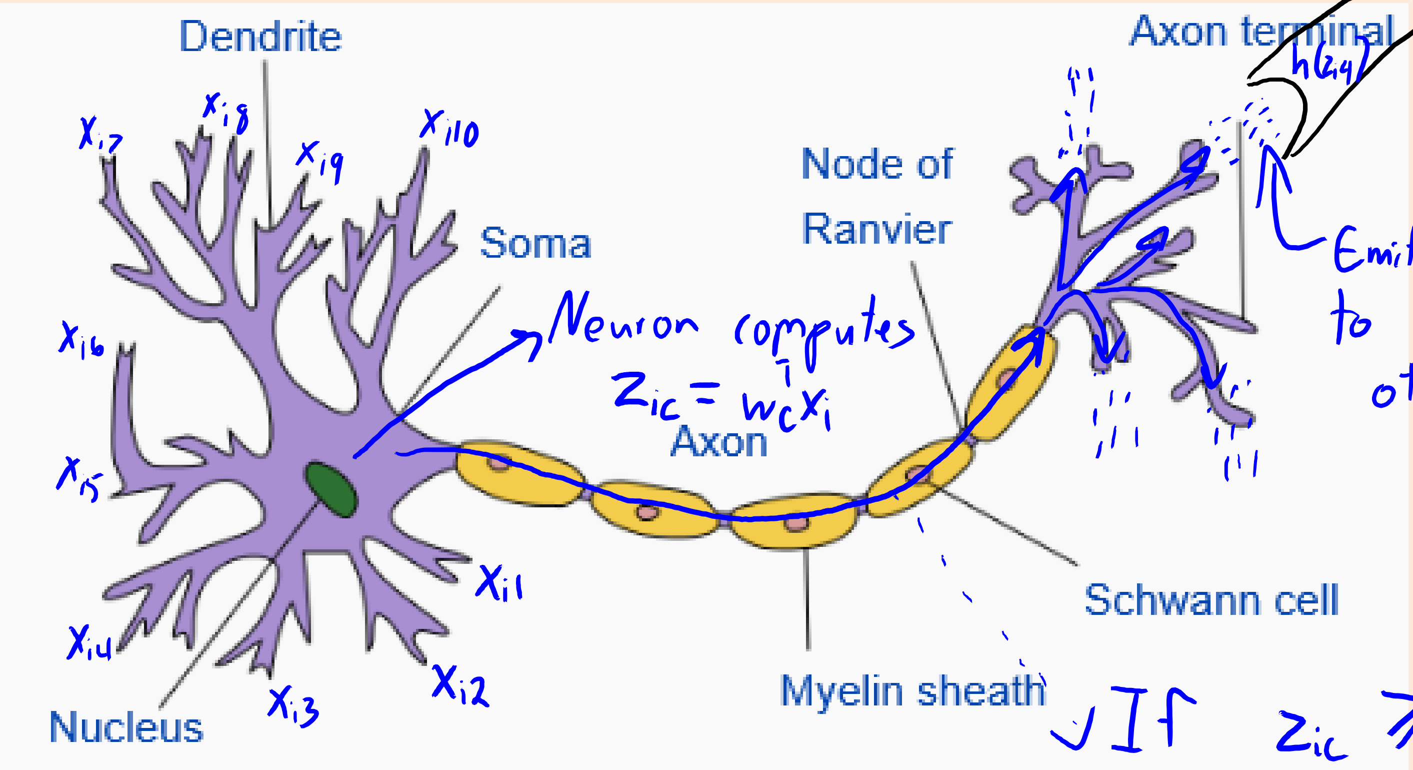


Emits "neurotransmitter" to send signal to other neurons.

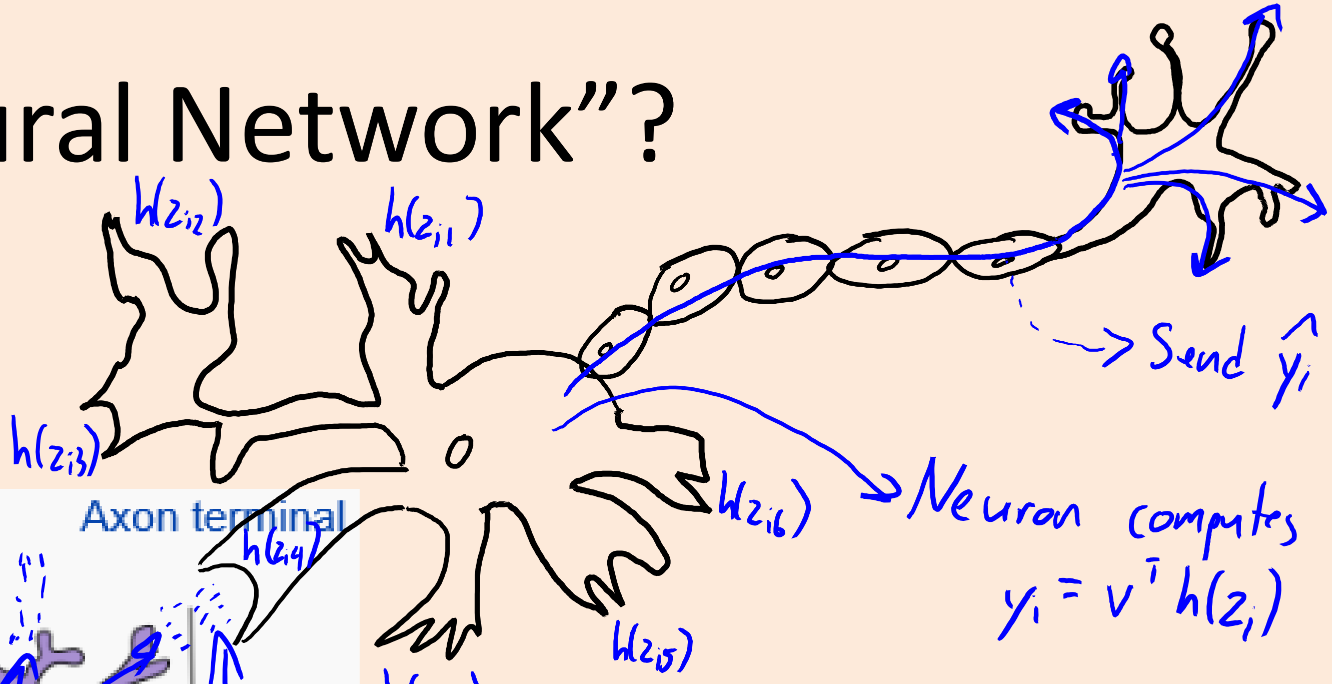
If $z_{ic} \geq 0$ neuron sends signal along axon. } We approximate binary signal with $\frac{1}{1+\exp(-z_{ic})}$

https://en.wikipedia.org/wiki/Neuron

Why “Neural Network”?



Neuron computes $z_{ic} = w_c^T x_i$



Send $\hat{y}_i$

Neuron computes $y_i = v^T h(z_i)$

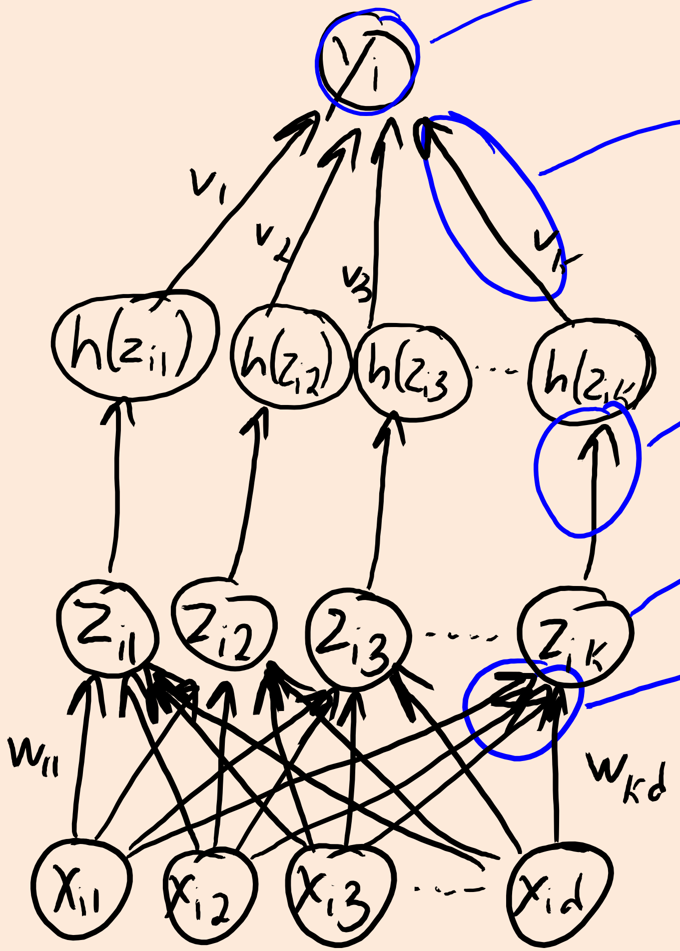
Emits "neurotransmitter" to send signal to other neurons.

If $z_{ic} \geq 0$ neuron sends signal along axon.

We approximate binary signal with $\frac{1}{1+\exp(-z_{ic})}$

https://en.wikipedia.org/wiki/Neuron

Why “Neural Network”?



Predictions based on aggregation $v^T h(Wx_i)$ at y_i "neuron"

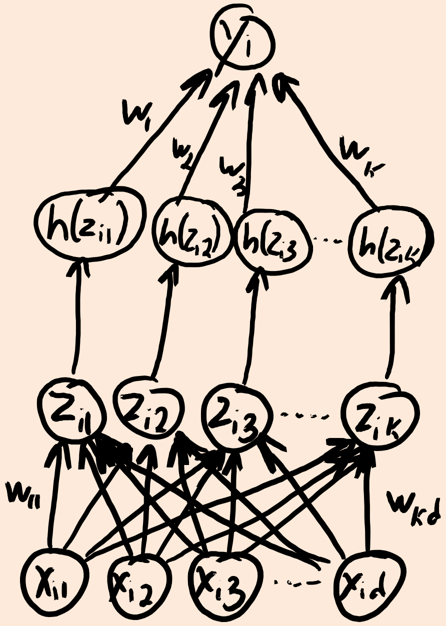
Synapse between z_{ik} and y_i "neuron"

binary signal $h(w_c^T x_i)$ sent along "axon"

Neuron aggregates signals: $w_c^T x_i$

"dendrites" for z_{ik} "neuron" are reciving x_{ij} values

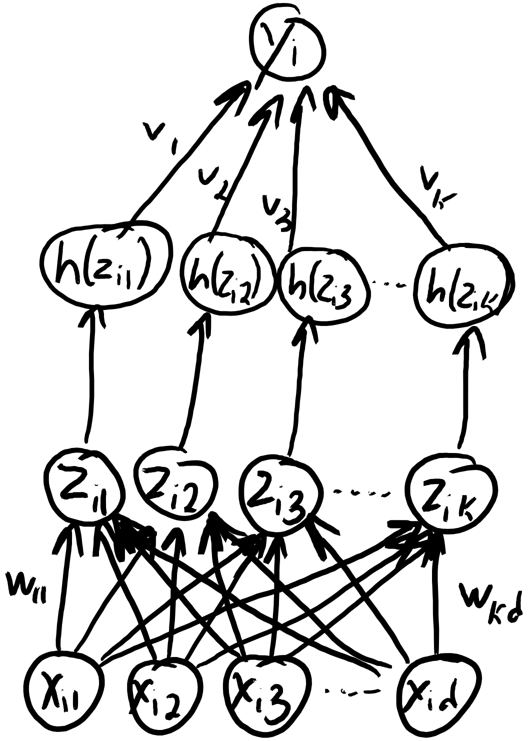
“Artificial” Neural Nets vs. “Real” Networks Nets



- Artificial neural network:
  - x_i is measurement of the world.
  - z_i is internal representation of world.
  - y_i is output of neuron for classification/regression.
- Real neural networks are more complicated:
  - Timing of action potentials seems to be important.
    - “Rate coding”: frequency of action potentials simulates continuous output.
  - Neural networks don’t reflect sparsity of action potentials.
  - How much computation is done inside neuron?
  - Brain is highly organized (e.g., substructures and cortical columns).
  - Connection structure changes.
  - Different types of neurotransmitters.

Deep Learning

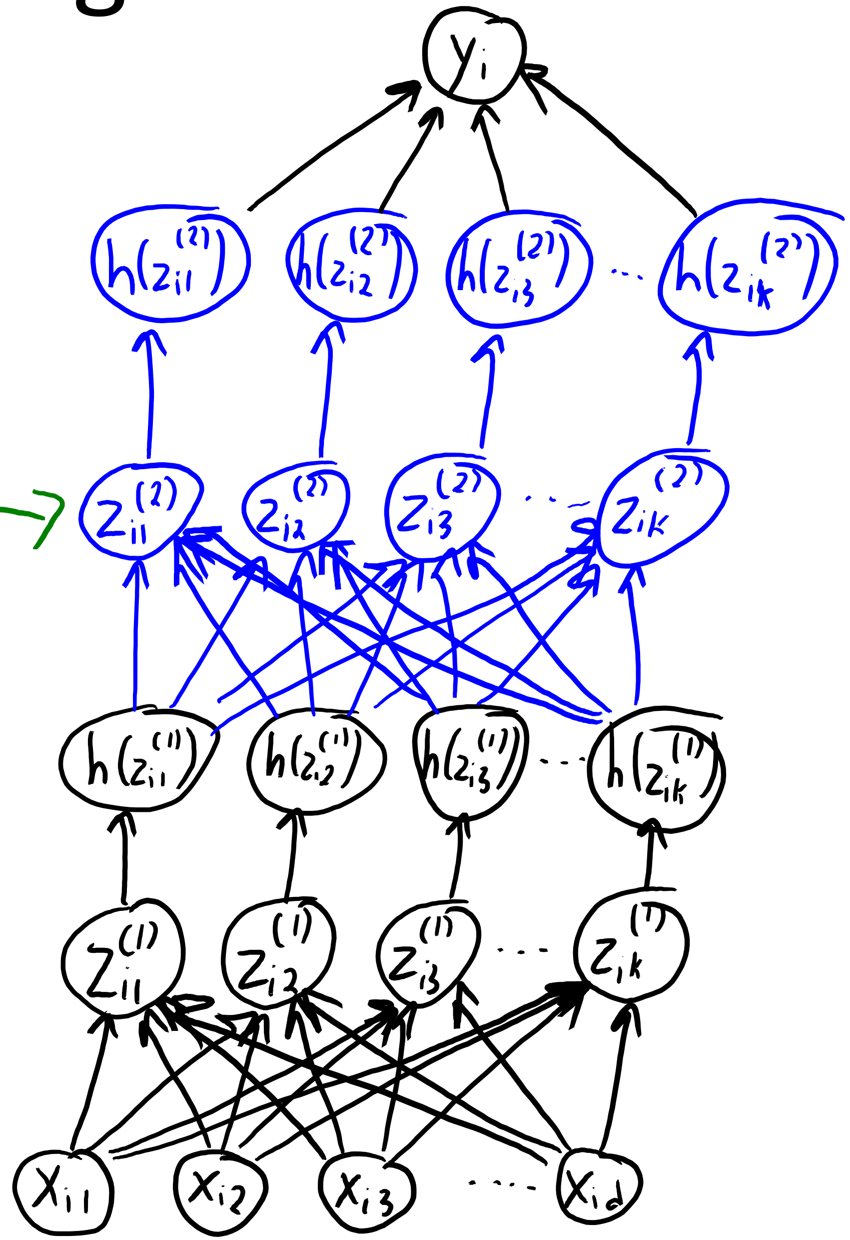
Neural network:



Deep learning:

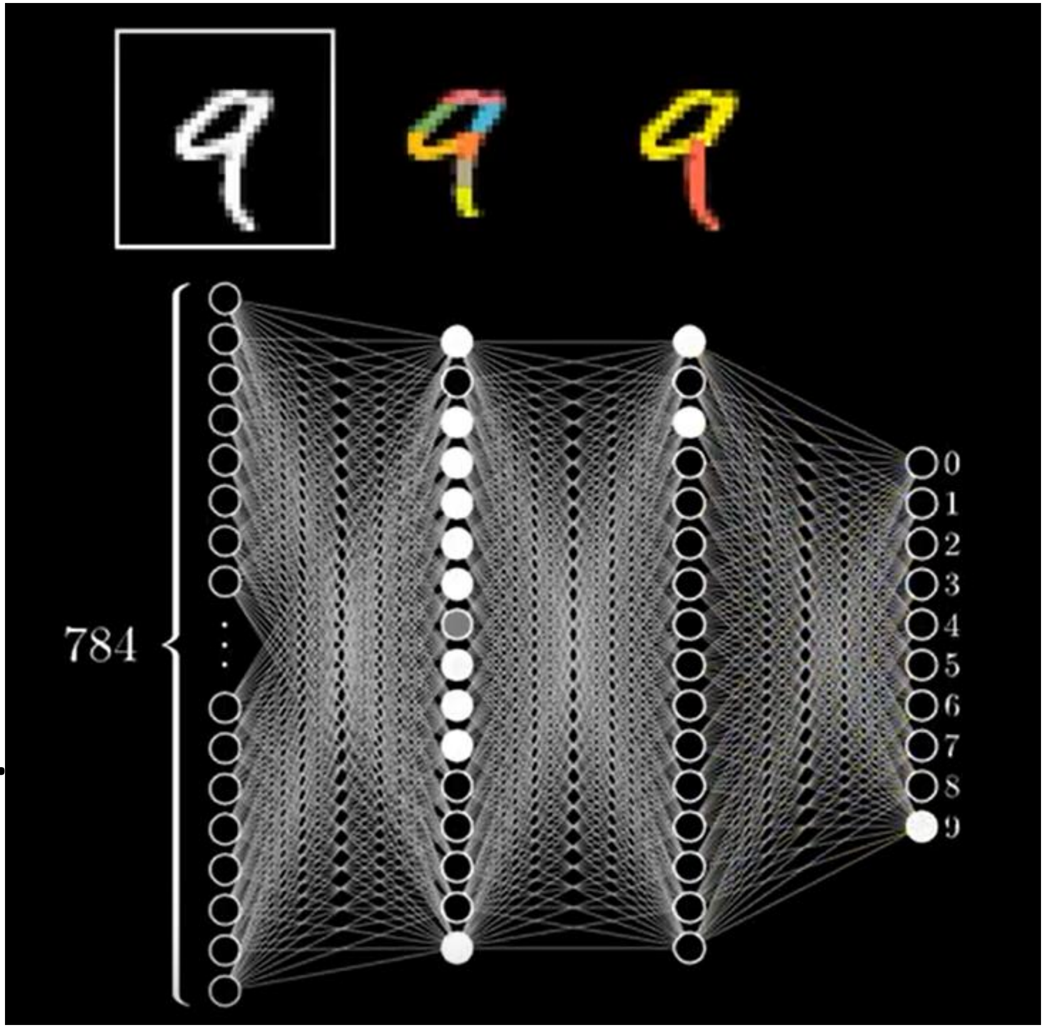
Second "layer" of latent features

You can add more "layers" to go "deeper"



"Hierarchies of Parts" Motivation for Deep Learning

- Each "neuron" might recognize a "part" of a digit.
  - "Deeper" neurons might recognize combinations of parts.
  - Represent complex objects as hierarchical combinations of re-useable parts (a simple "grammar").



- Watch the full video here:
  - https://www.youtube.com/watch?v=aircAruvnKk

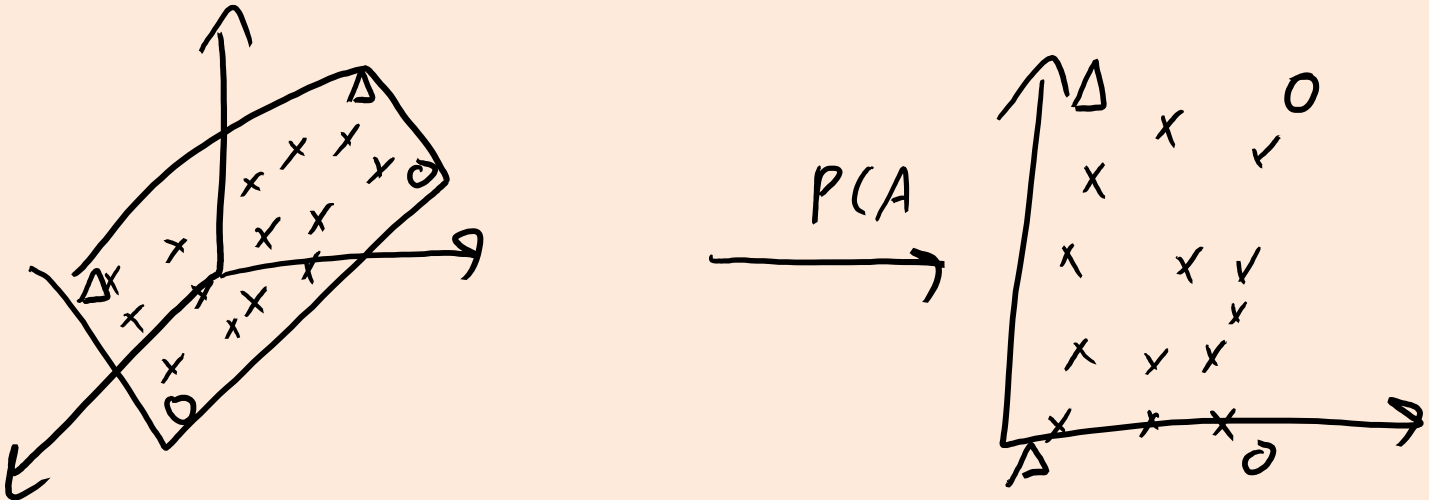
Summary

- Word2vec:
  - Latent-factor (continuous) representation of words.
  - Based on predicting word from its context.
- Neural networks learn features z_i for supervised learning.
- Sigmoid function avoids degeneracy by introducing non-linearity.
- Biological motivation for (deep) neural networks.
- Deep learning considers neural networks with many hidden layers.

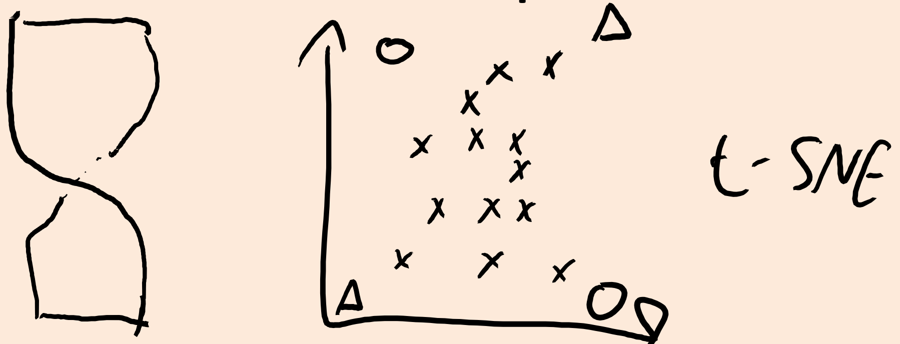
- Next time:
  - Training deep networks.

Does t-SNE always outperform PCA?

- Consider 3D data living on a 2D hyper-plane:



- PCA can perfectly capture the low-dimensional structure.
- T-SNE can capture the local structure, but can “twist” the plane.
  - It doesn’t try to get long distances correct.



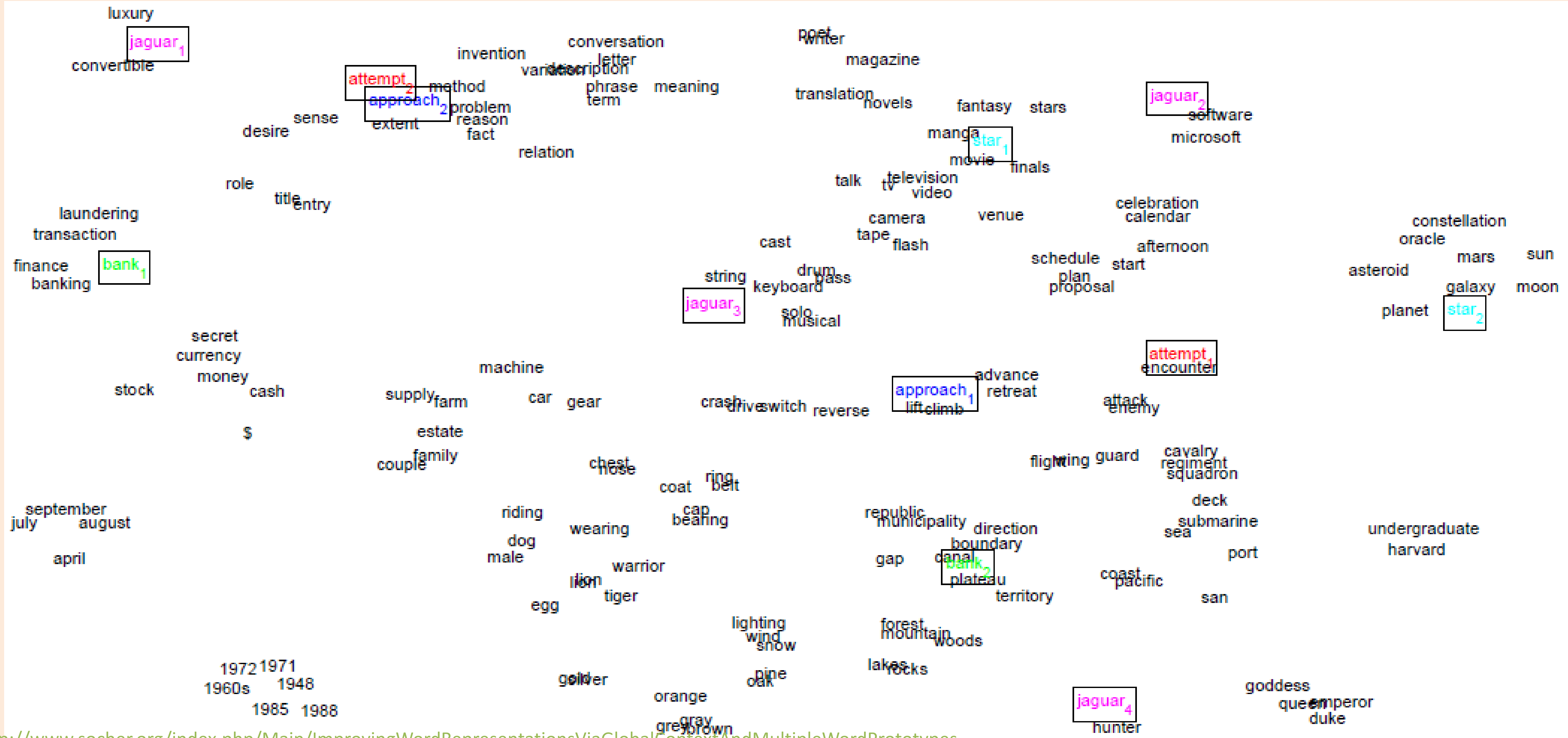
Multiple Word Prototypes

- What about homonyms and polysemy?
  - The word vectors would need to account for all meanings.

- More recent approaches:
  - Try to cluster the different contexts where words appear.
  - Use different vectors for different contexts.

$$X_{jaguar} = \begin{bmatrix} \cdots \\ \cdots \\ \; \end{bmatrix} \begin{array}{l} z_{j1} \\ z_{j2} \\ z_{j3} \end{array}$$

Multiple Word Prototypes



http://www.socher.org/index.php/Main/ImprovingWordRepresentationsViaGlobalContextAndMultipleWordPrototypes

Why $z_i = Wx_i$?

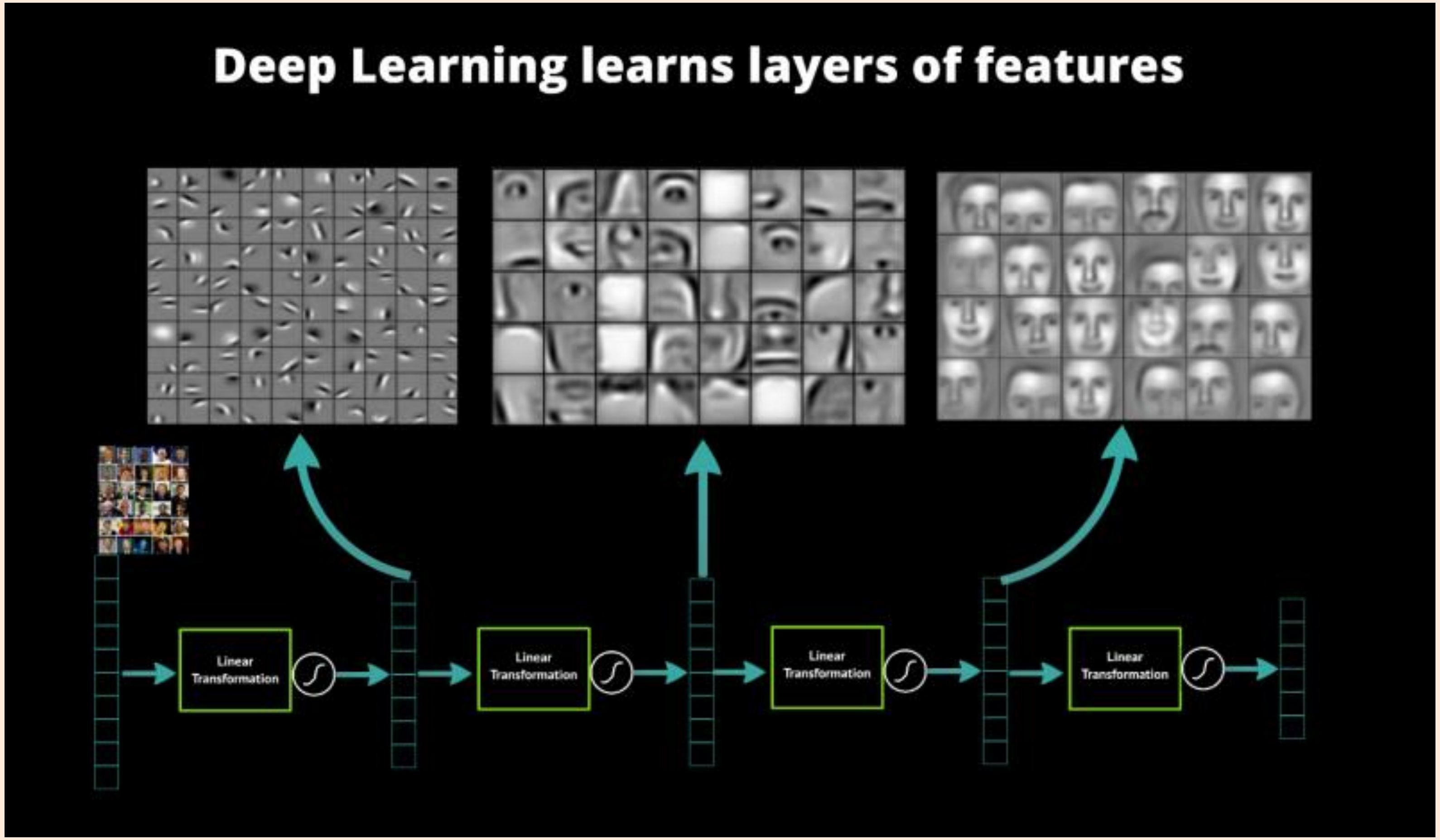
- In PCA we had that the optimal $Z = XW^T(WW^T)^{-1}$.
- If W had normalized+orthogonal rows, $Z = XW^T$ (since $WW^T = I$).
  - So $z_i = Wx_i$ in this normalized+orthogonal case.

- Why we would use $z_i = Wx_i$ in neural networks?
  - We didn't enforce normalization or orthogonality.

- Well, the value $W^T(WW^T)^{-1}$ is just "some matrix".
  - You can think of neural networks as just directly learning this matrix.

Cool Picture Motivation for Deep Learning

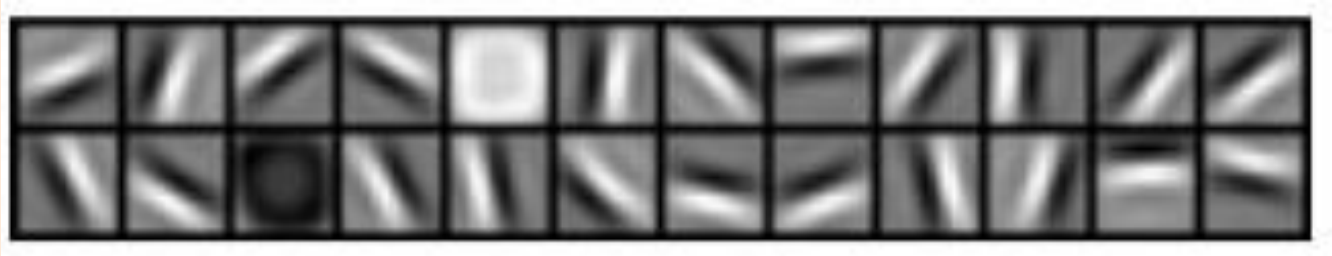
- Faces might be composed of different “parts”:



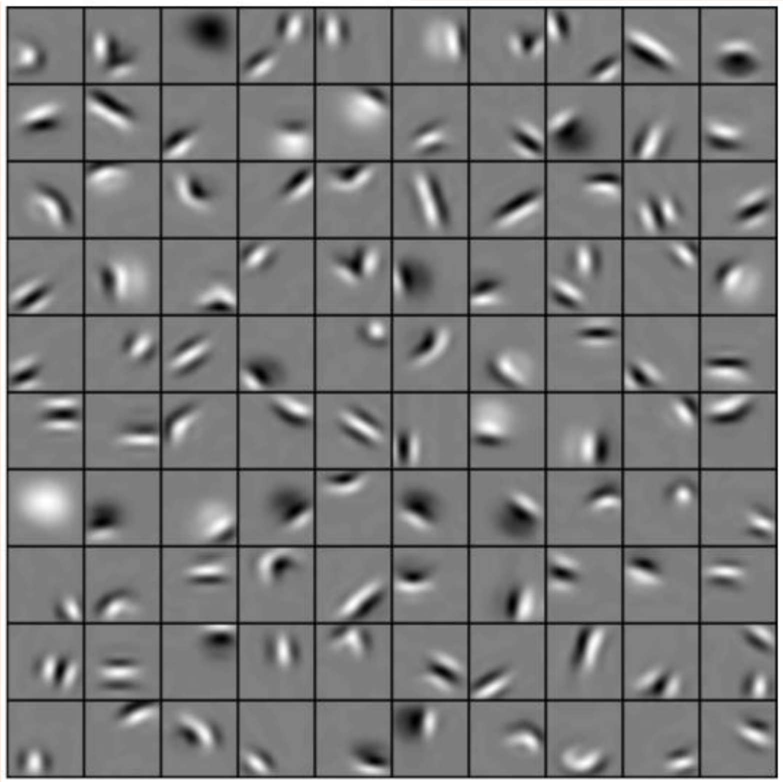
http://www.datarobot.com/blog/a-primer-on-deep-learning/

Cool Picture Motivation for Deep Learning

- First layer of z_i trained on 10 by 10 image patches:



"Gabor filters"



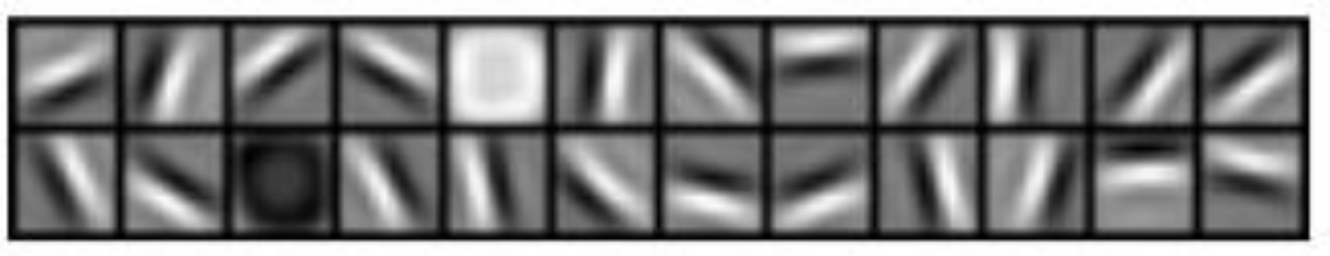
- Attempt to visualize second layer:
  – Corners, angles, surface boundaries?

- Models require many tricks to work.
  – We'll discuss these next time.

http://www.cs.toronto.edu/~rgrosse/icml09-cdbn.pdf

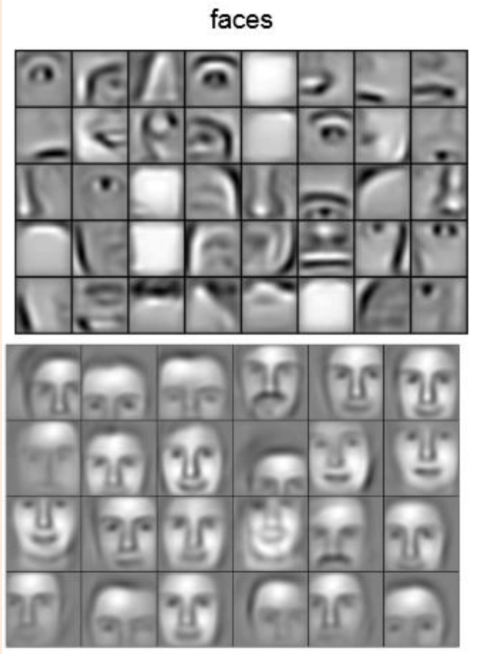
Cool Picture Motivation for Deep Learning

- First layer of z_i trained on 10 by 10 image patches:



"Gabor filters"

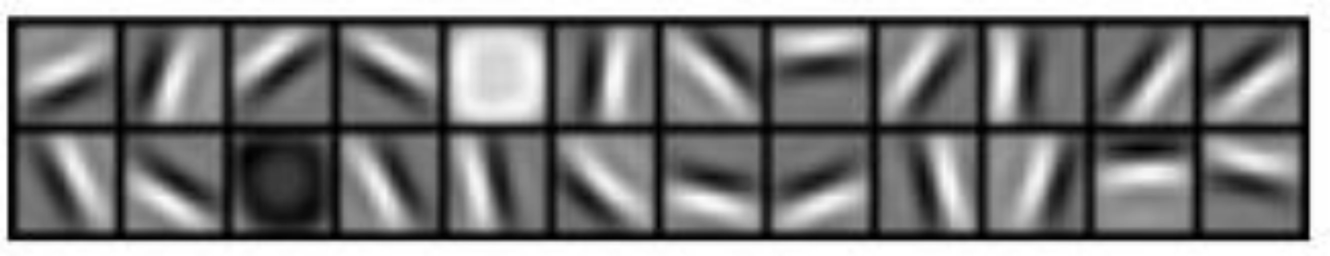
- Visualization of second and third layers trained on specific objects:



http://www.cs.toronto.edu/~rgrosse

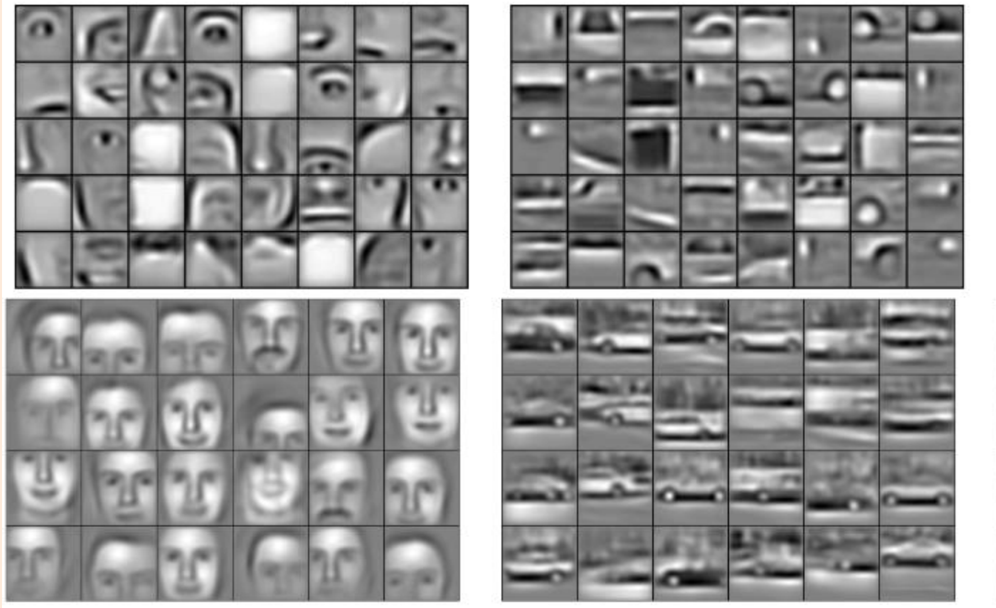
Cool Picture Motivation for Deep Learning

- First layer of z_i trained on 10 by 10 image patches:



"Gabor filters"

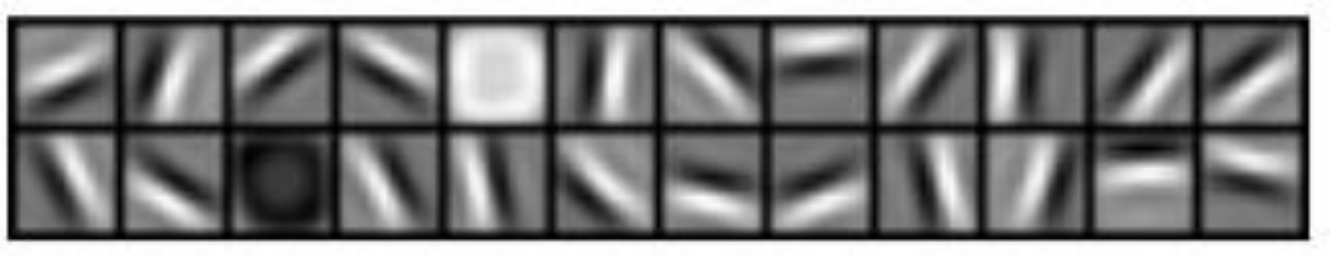
- Visualization of second and third layers trained on specific objects:



http://www.cs.toronto.edu/~rgrosse/icml09-cdbn.pdf

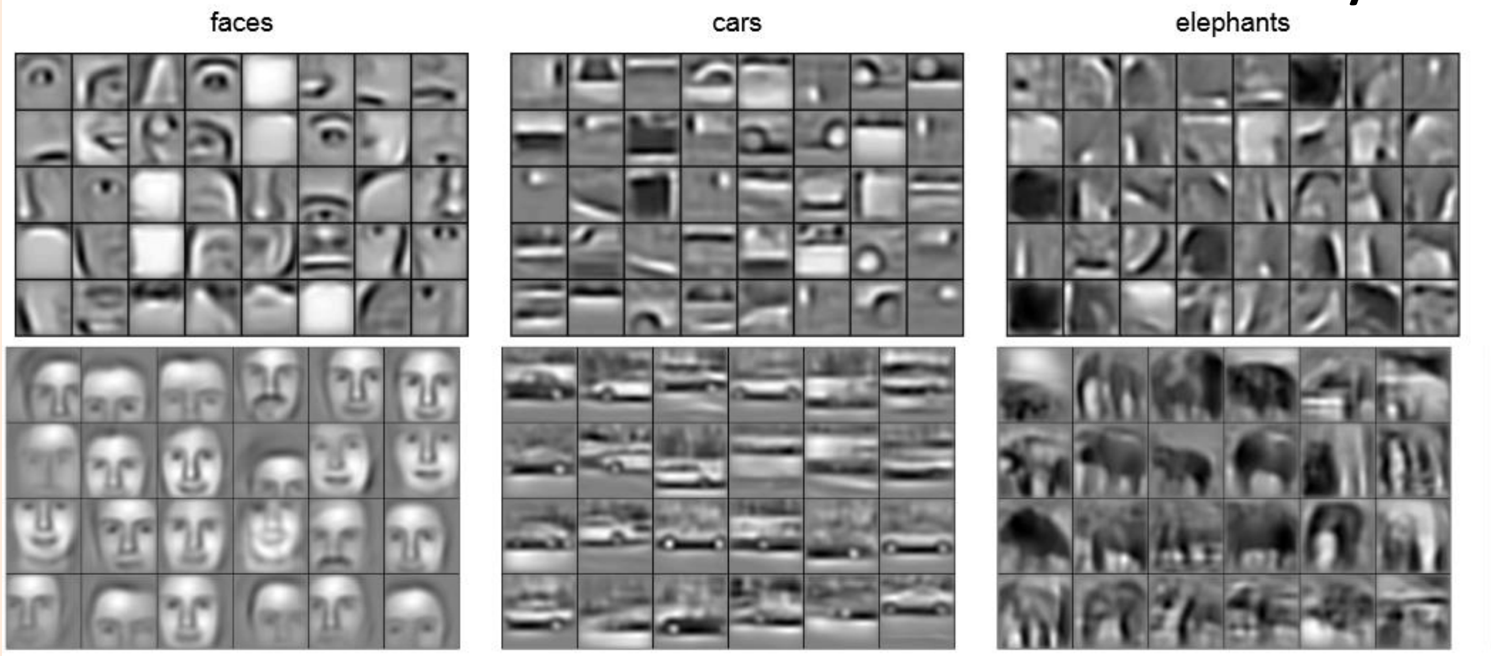
Cool Picture Motivation for Deep Learning

- First layer of z_i trained on 10 by 10 image patches:



"Gabor filters"

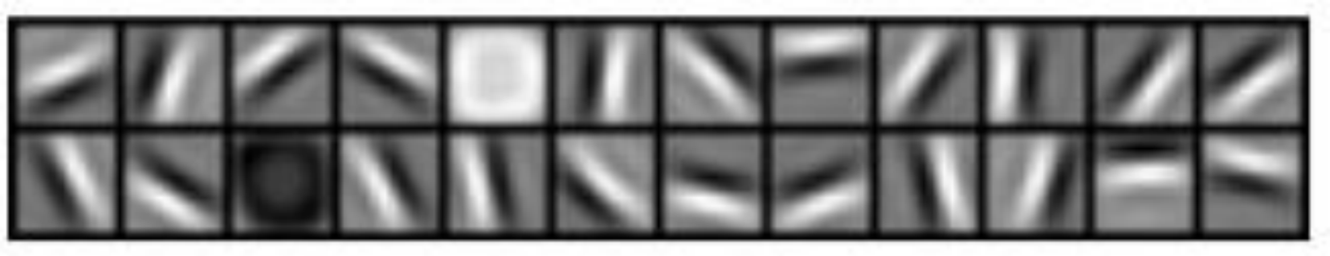
- Visualization of second and third layers trained on specific objects:



http://www.cs.toronto.edu/~rgrosse/icml09-cdbn.pdf

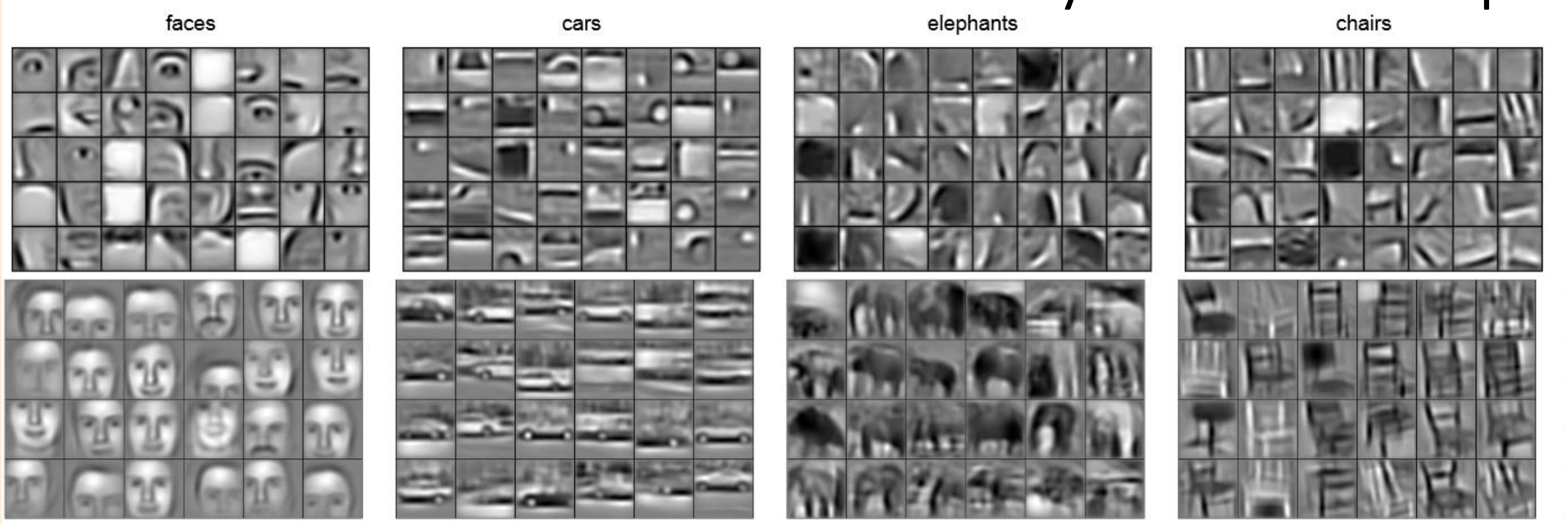
Cool Picture Motivation for Deep Learning

- First layer of z_i trained on 10 by 10 image patches:



"Gabor filters"

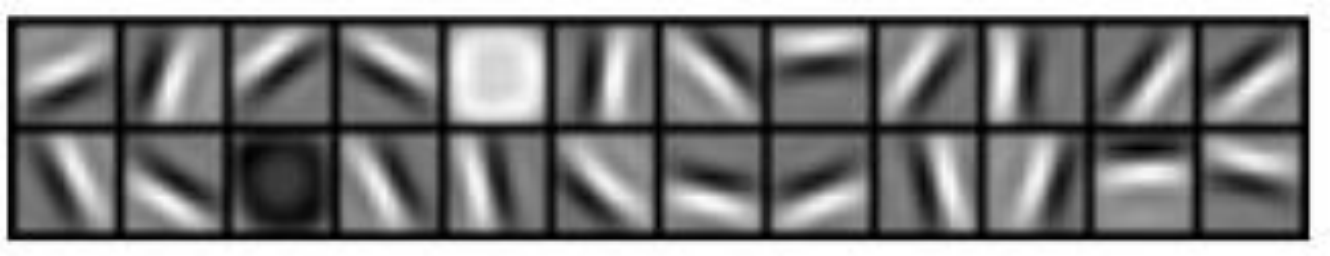
- Visualization of second and third layers trained on specific objects:



http://www.cs.toronto.edu/~rgrosse/icml09-cdbn.pdf

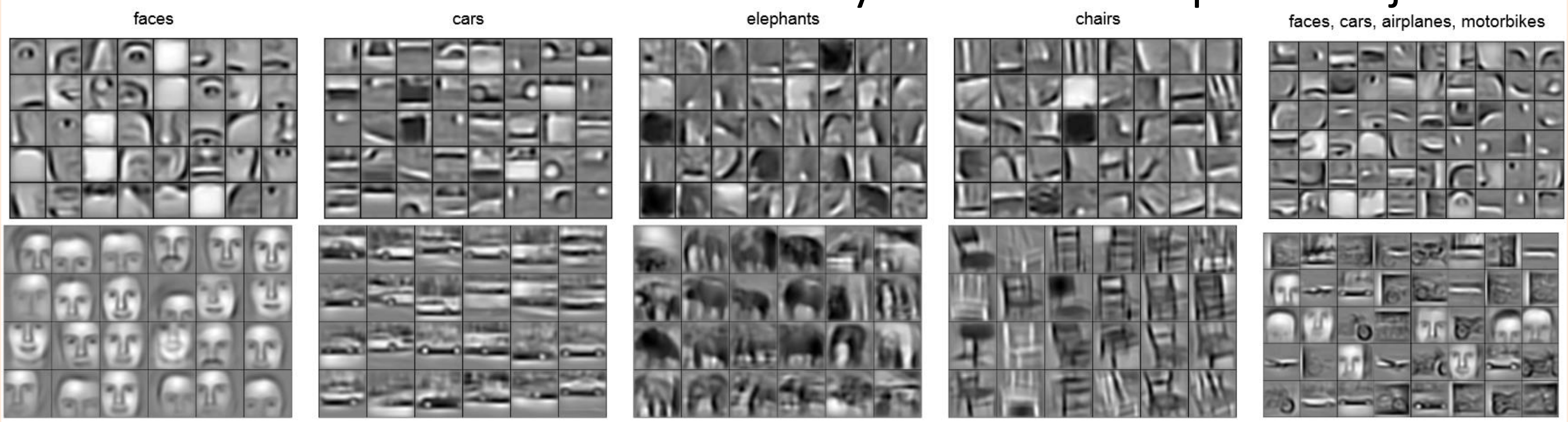
Cool Picture Motivation for Deep Learning

- First layer of z_i trained on 10 by 10 image patches:



"Gabor filters"

- Visualization of second and third layers trained on specific objects:



http://www.cs.toronto.edu/~rgrosse/icml09-cdbn.pdf