Machine Learning 10-601

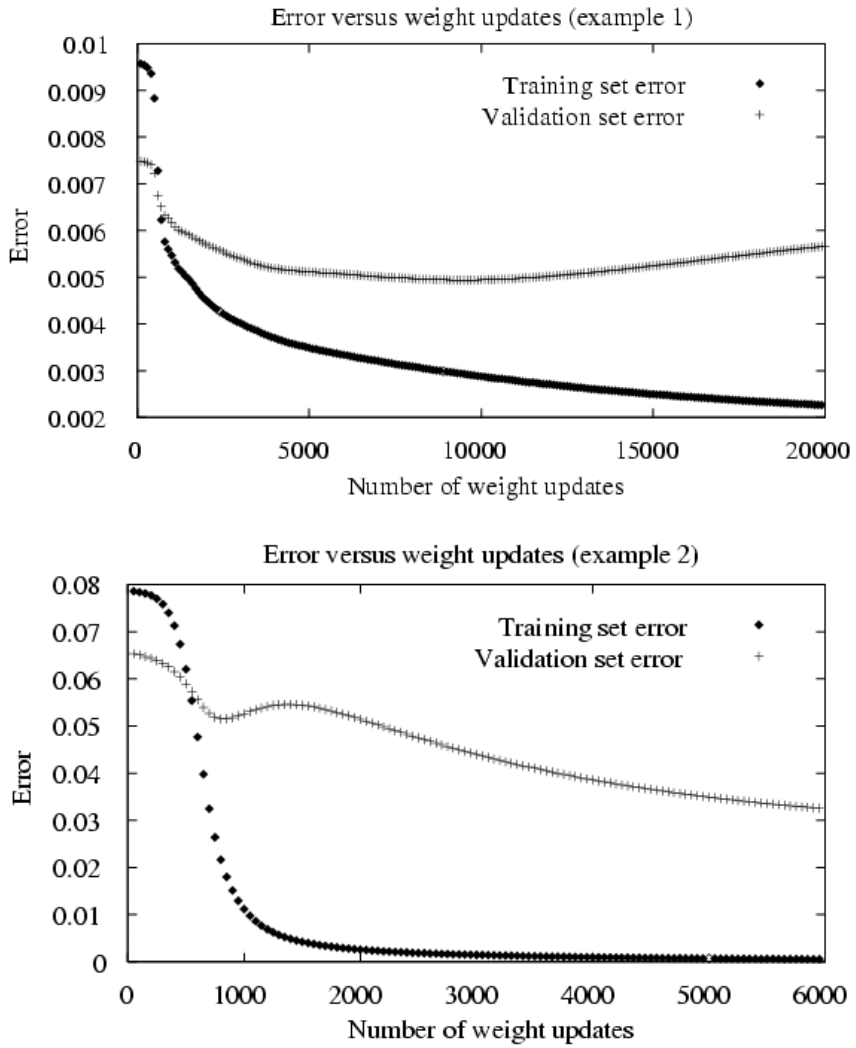
Tom M. Mitchell
Machine Learning Department
Carnegie Mellon University

October 2, 2017

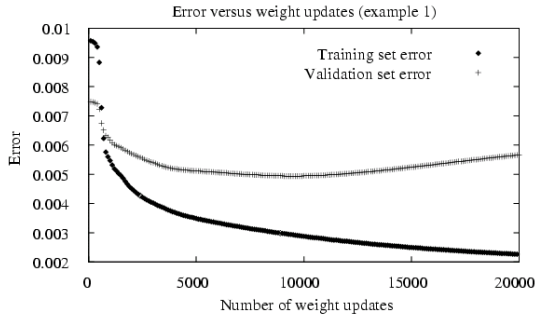
Today:
- Overfitting
- Representation learning
- Sequential models

Reading:
- Goodfellow: Convolutional nets

Overfitting in Deep Networks



Dealing with Overfitting



Our learning algorithm involves a parameter

n=number of gradient descent iterations

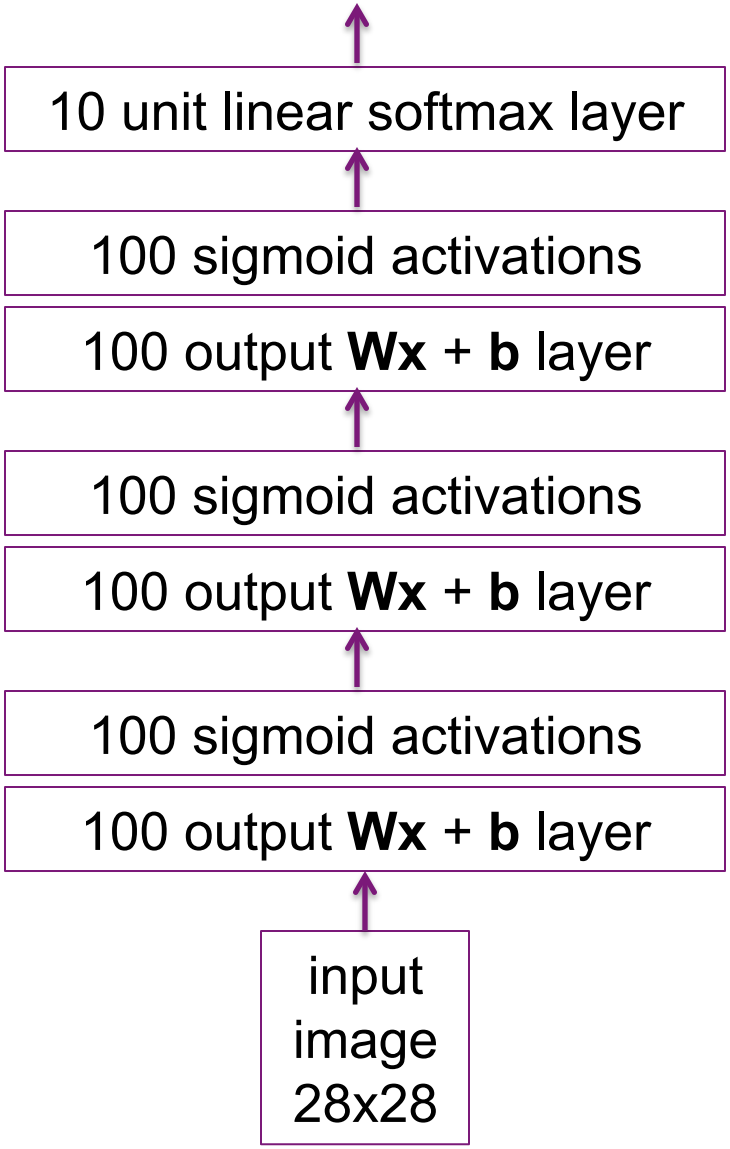
How do we choose n to optimize future error or loss?

- Separate available data into <u>training</u> and <u>validation</u> set
- Use <u>training</u> to perform gradient descent
- n ← number of iterations that optimizes <u>validation</u> set error

→ *gives unbiased estimate of optimal n*

(but still an optimistically <u>biased</u> estimate of true error)

Experiment on MNIST net: Batch Normalization



[Ioffe & Szegedy, 2015]

Batch normalization: managing gradient descent

- Key idea: add batch normalization layers to network.
  For each minibatch, BN layer scales each feature to have mean 0, variance 1. Then adds an offset β and scaling γ parameter to train.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

[Ioffe & Szegedy, 2015]

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

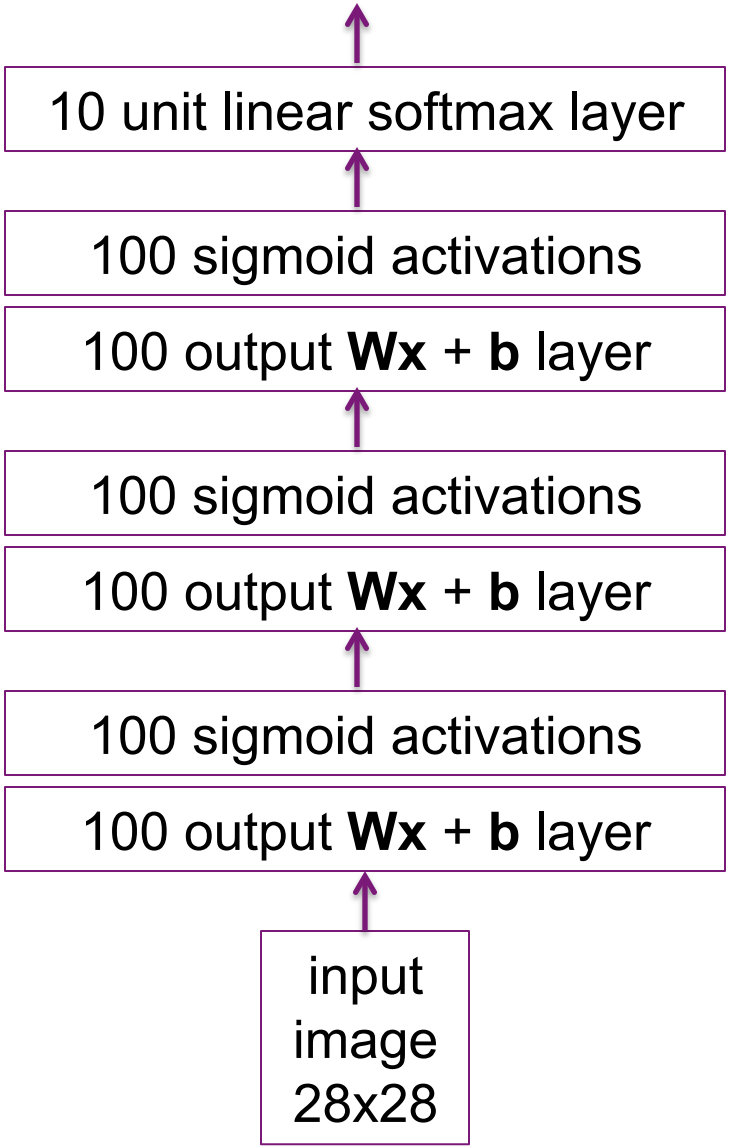
$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

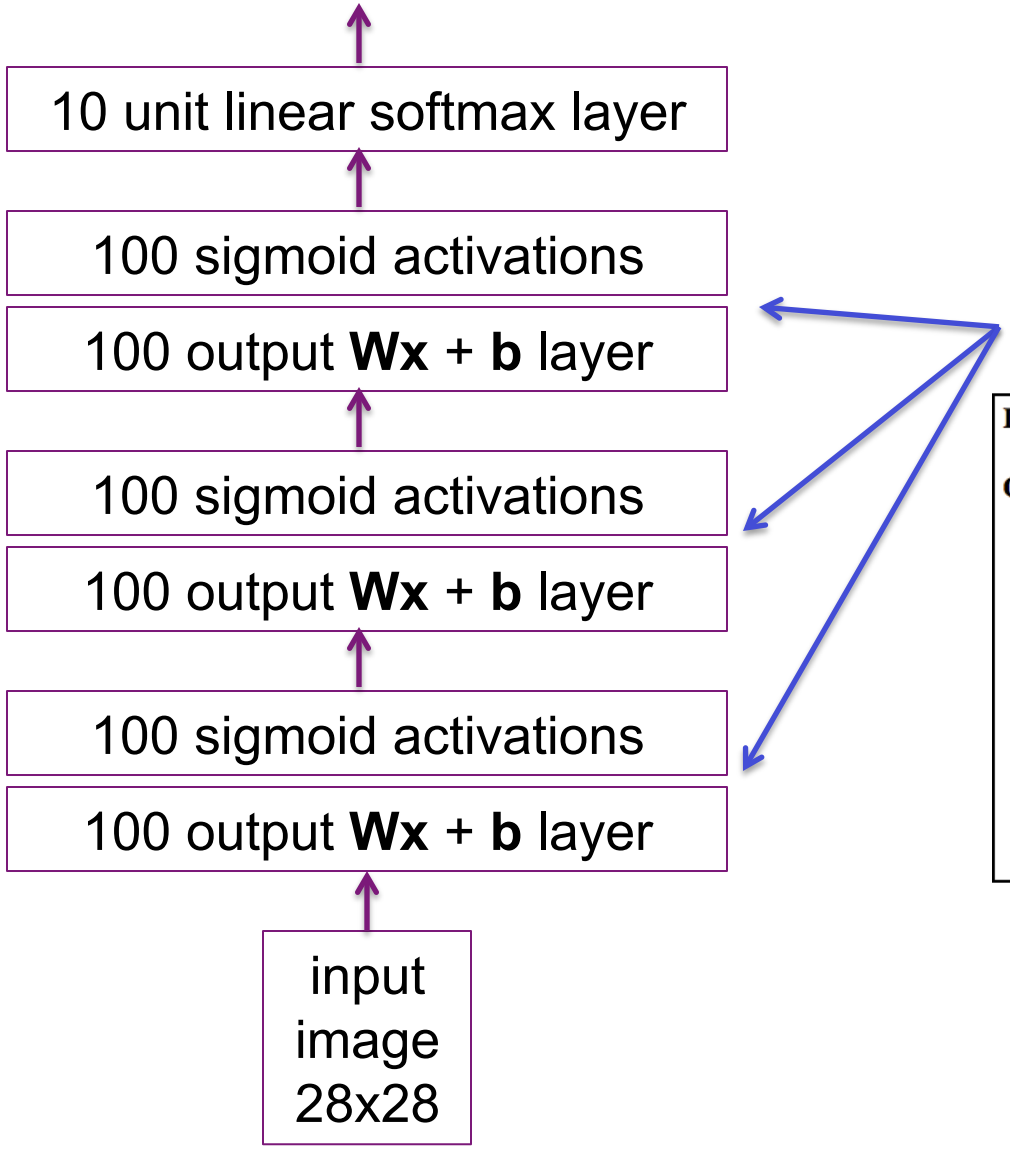
[Ioffe & Szegedy, 2015]

Impact of Batch Normalization on MNIST net



[Ioffe & Szegedy, 2015]

Impact of Batch Normalization on MNIST net



insert Batch Normalization layers here

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

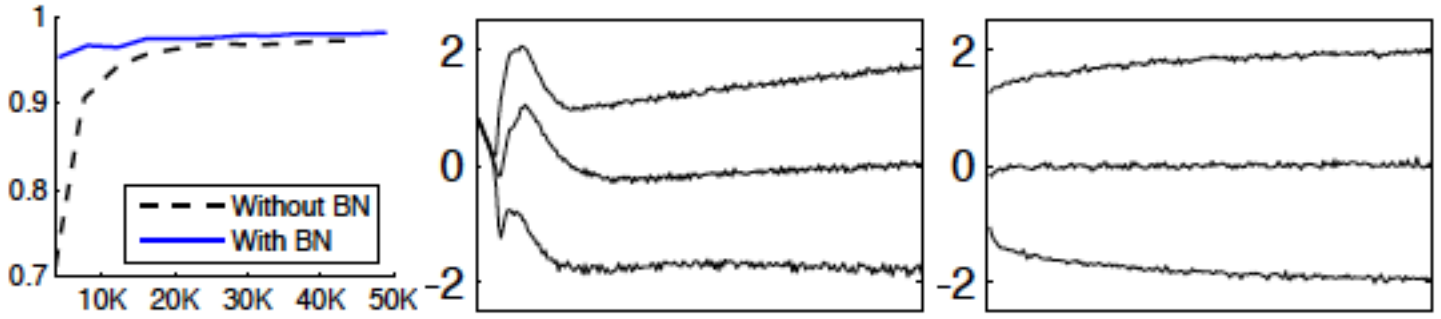
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

[Ioffe & Szegedy, 2015]

Impact of Batch Normalization on MNIST net



(a) (b) Without BN (c) With BN

Figure 1: (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of training, shown as* $\{15, 50, 85\}$*th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.*

[Ioffe & Szegedy, 2015]

Expressive Power of Neural Networks

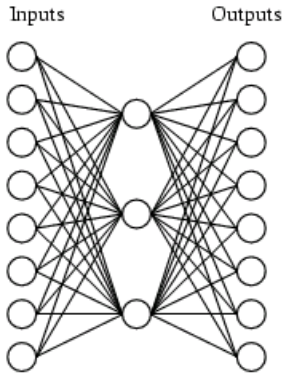
Boolean functions:

- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

Learning Hidden Layer Representations



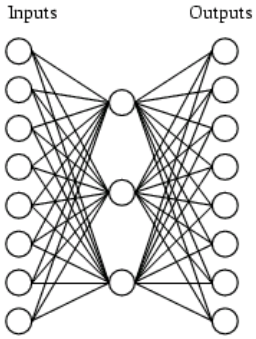
A target function:

Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

Can this be learned??

Learning Hidden Layer Representations

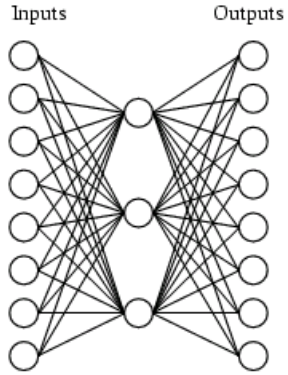
A network:

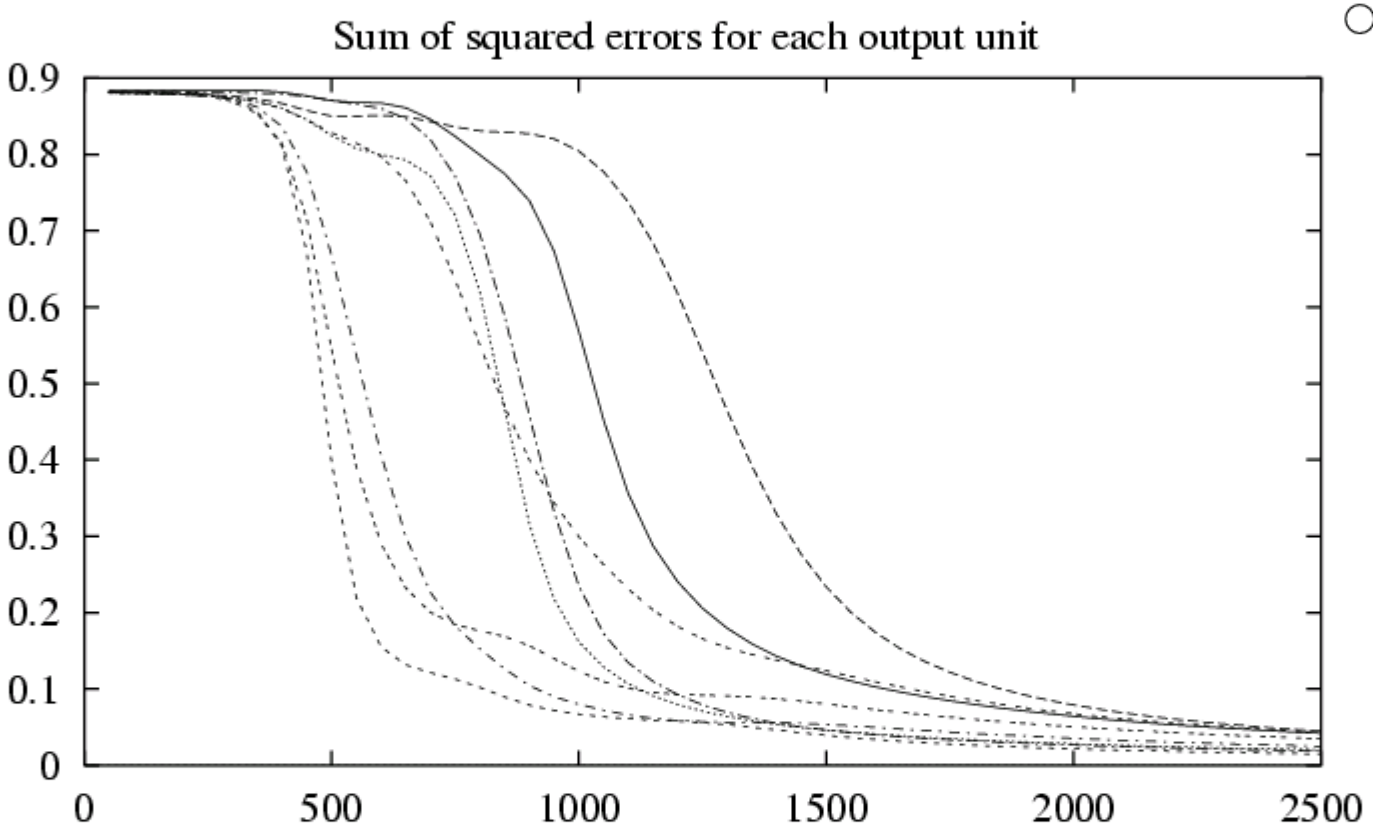


Learned hidden layer representation:

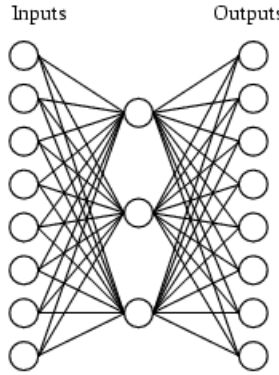
Input		Hidden Values				Output
10000000	$\rightarrow$	.89	.04	.08	$\rightarrow$	10000000
01000000	$\rightarrow$	.01	.11	.88	$\rightarrow$	01000000
00100000	$\rightarrow$	.01	.97	.27	$\rightarrow$	00100000
00010000	$\rightarrow$	.99	.97	.71	$\rightarrow$	00010000
00001000	$\rightarrow$	.03	.05	.02	$\rightarrow$	00001000
00000100	$\rightarrow$	.22	.99	.99	$\rightarrow$	00000100
00000010	$\rightarrow$	.80	.01	.98	$\rightarrow$	00000010
00000001	$\rightarrow$	.60	.94	.01	$\rightarrow$	00000001

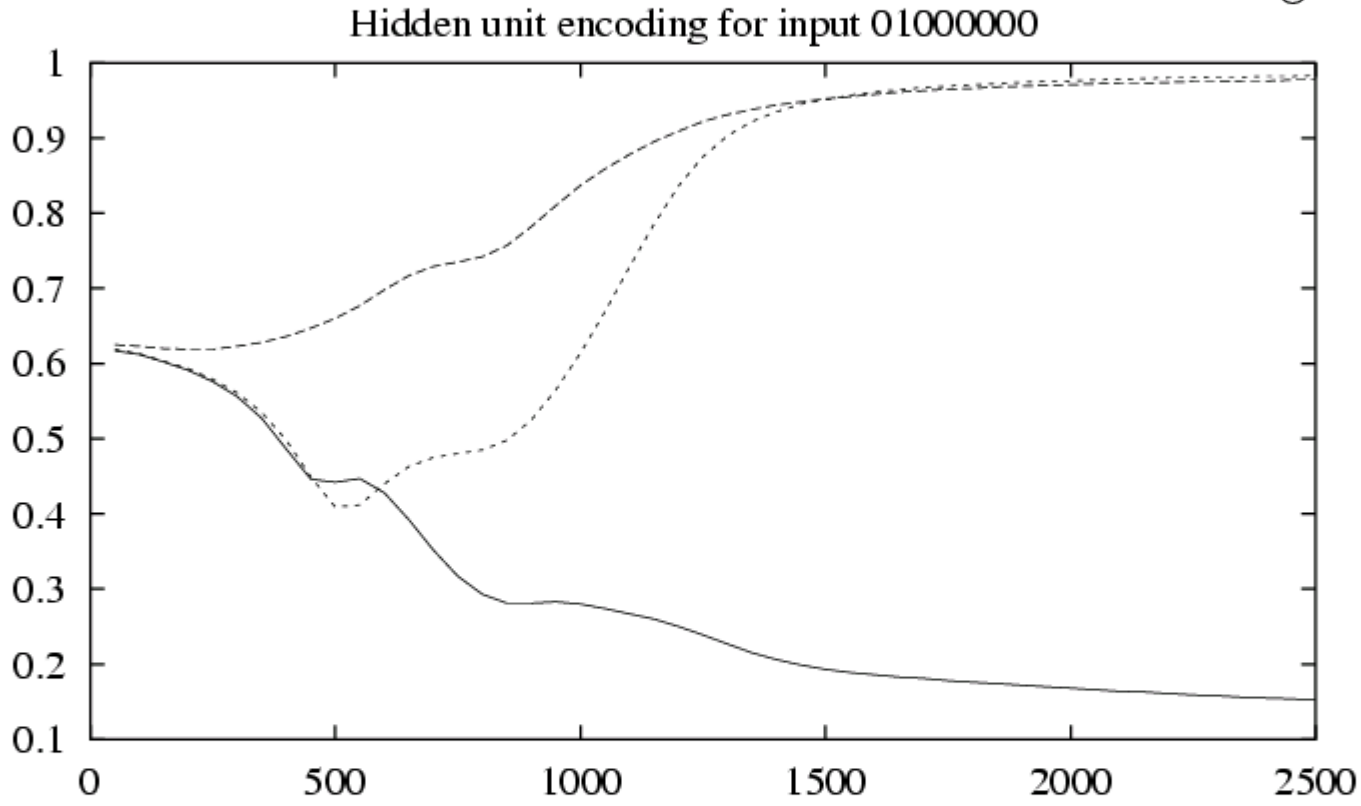
Training


Inputs
Outputs

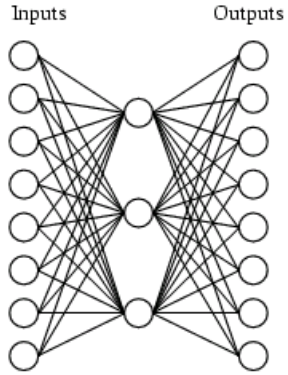

Sum of squared errors for each output unit
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500

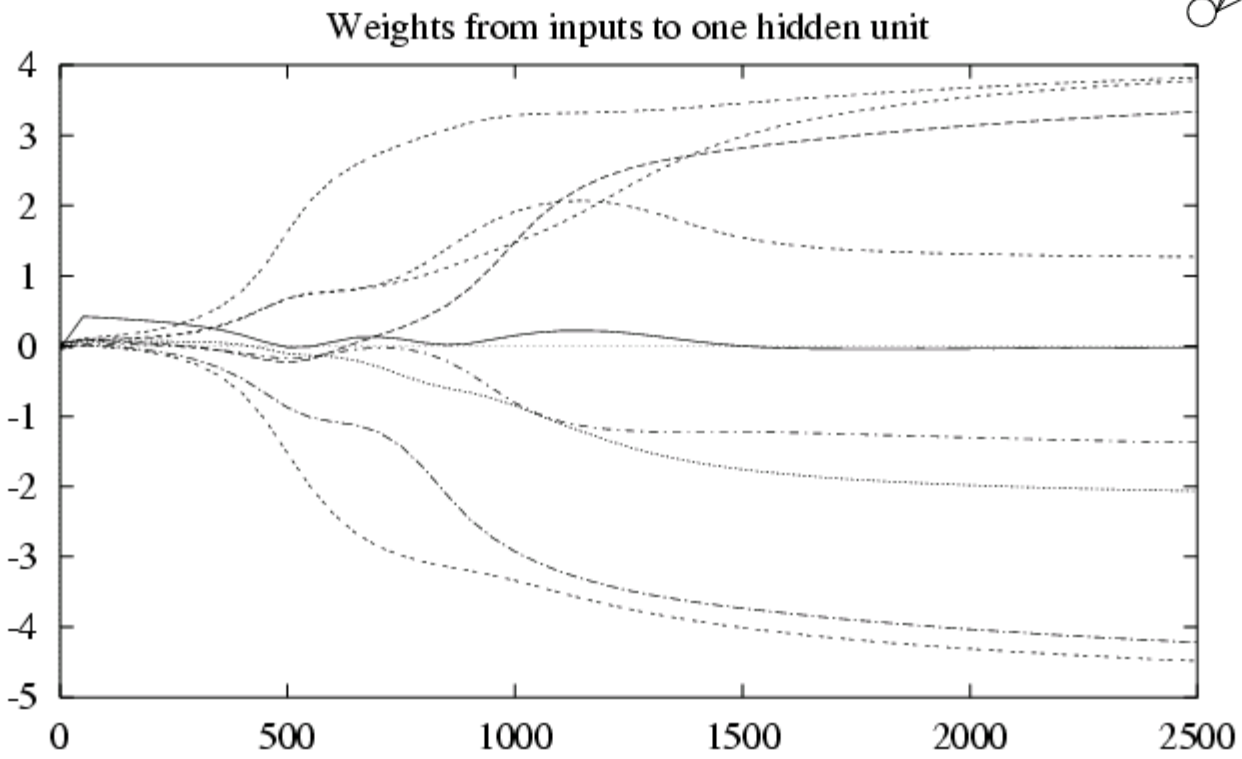
Training


Inputs
Outputs

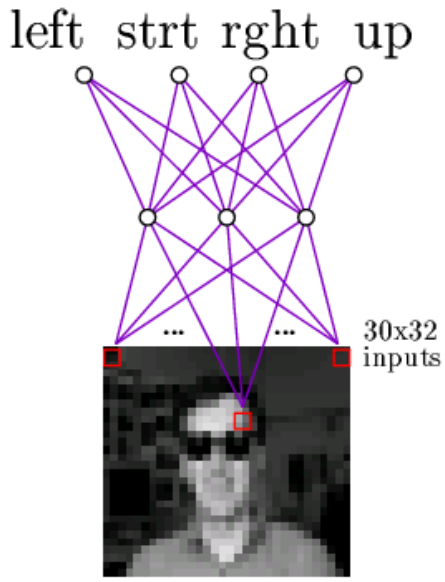

Hidden unit encoding for input 01000000
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
500
1000
1500
2000
2500

Training


Inputs
Outputs


Weights from inputs to one hidden unit
4
3
2
1
0
-1
-2
-3
-4
-5
0
500
1000
1500
2000
2500

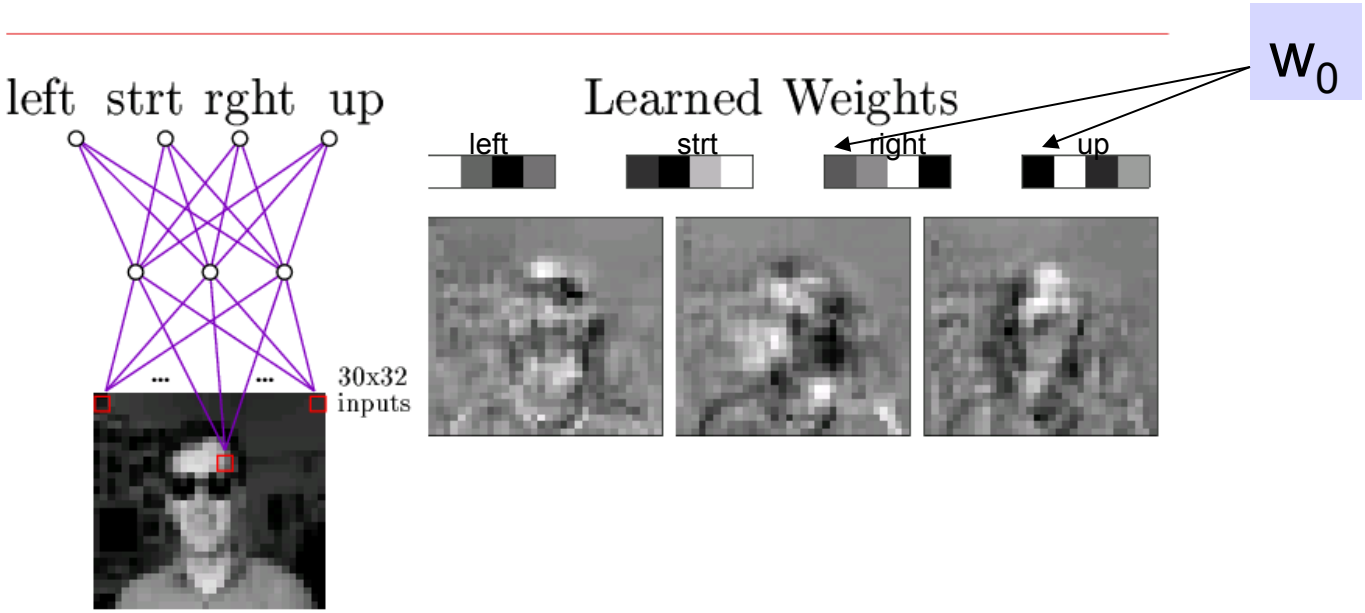
Neural Nets for Face Recognition



Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

Learned Hidden Unit Weights

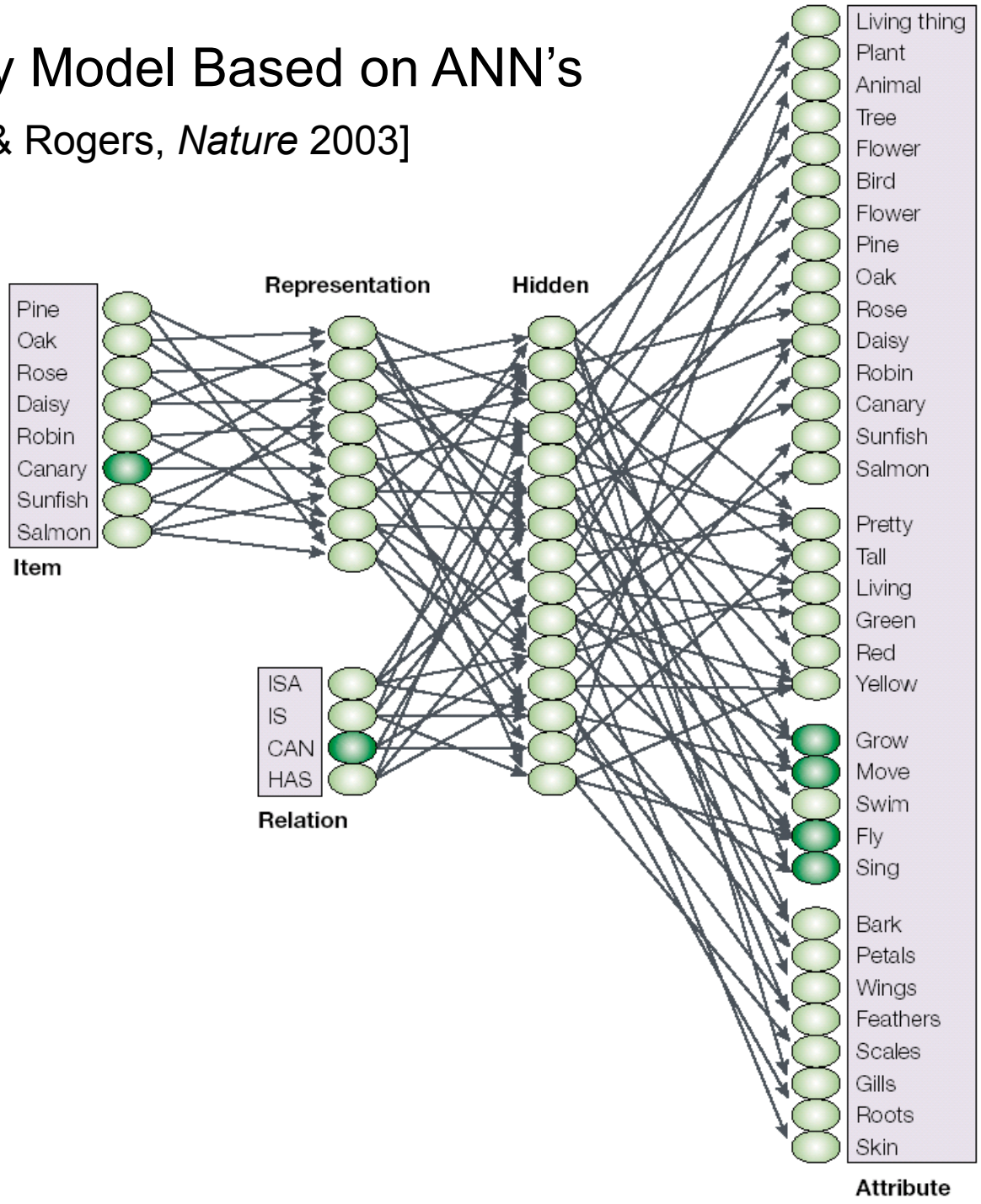


Typical input images

http://www.cs.cmu.edu/~tom/faces.html

Semantic Memory Model Based on ANN's

[McClelland & Rogers, *Nature* 2003]



No hierarchy given.

Train with assertions,
e.g., Can(Canary,Fly)

Learning Distributed Representations for Words

- also called “word embeddings”
- word2vec is one commonly used embedding
- based on skip gram model

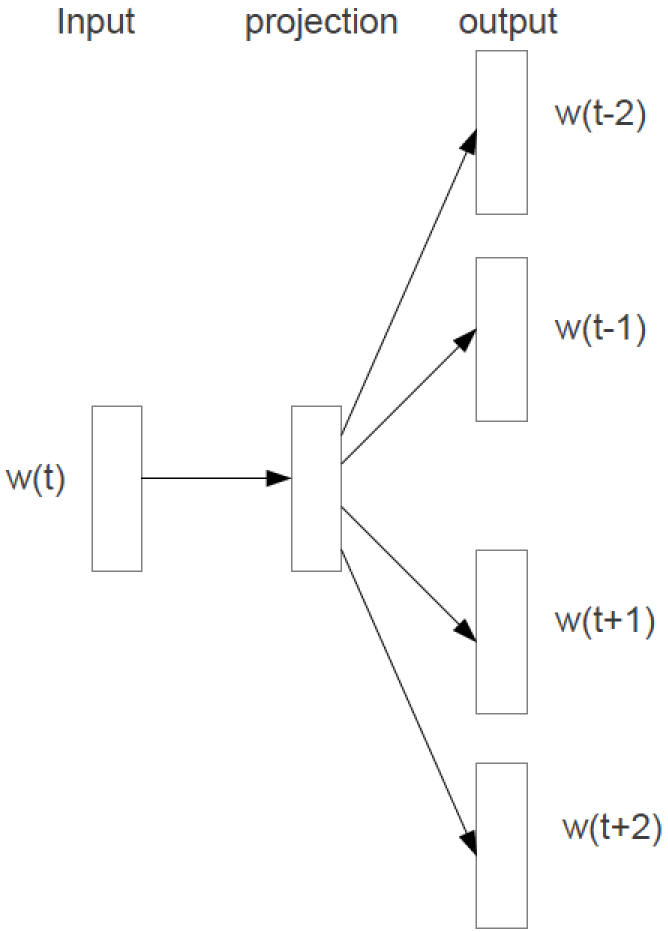
Key idea: given word sequence $w_1\ w_2\ \ldots\ w_T$
train network to predict surrounding words.

for each word w_t predict w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}

e.g., “the dog jumped over the fence in order to get to..”

“the cat jumped off the widow ledge in order to ...”

Word2Vec Word Embeddings



basic skip gram model:

train to maximize:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

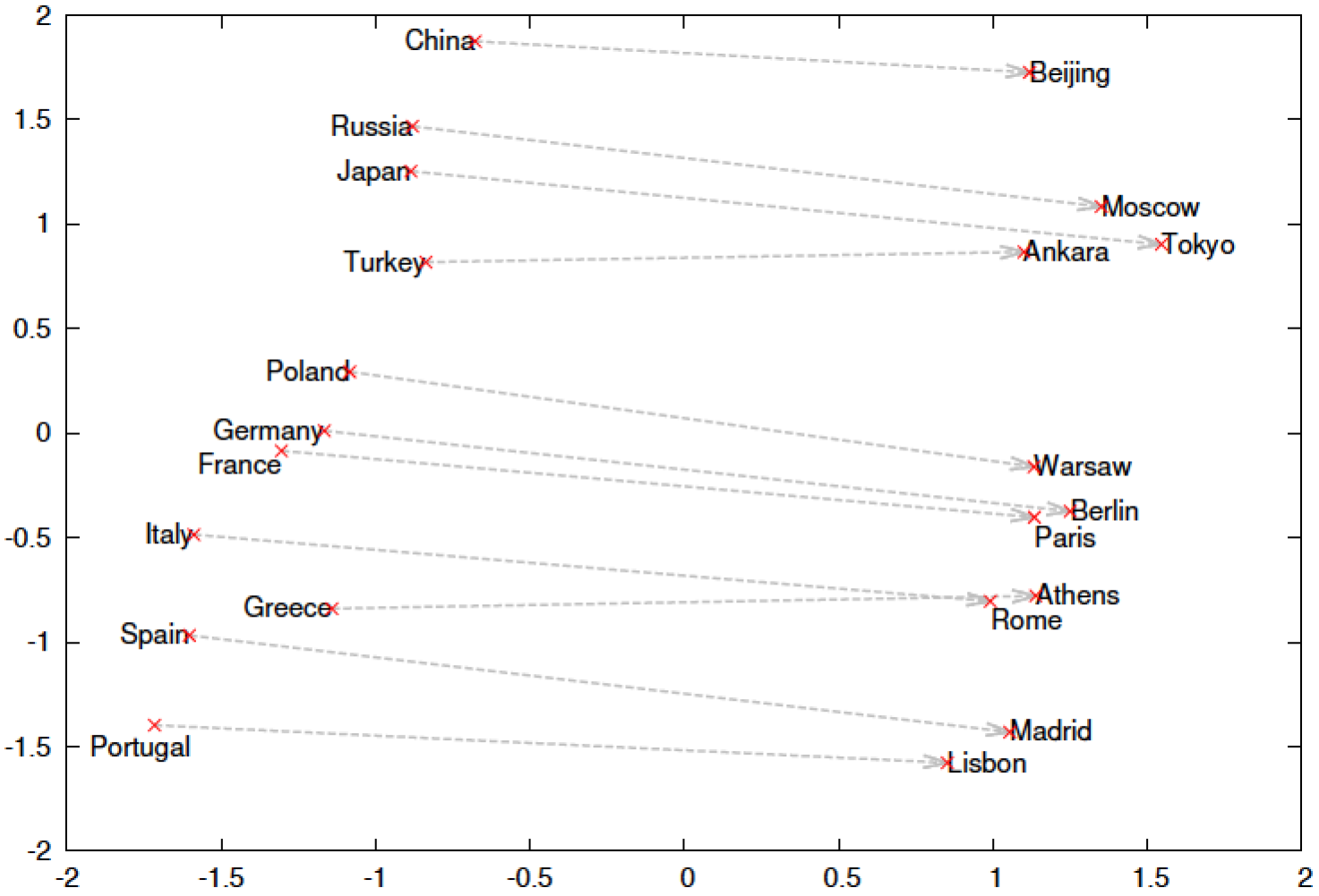
where

$$p(w_O|w_I)=\frac{\exp\left({v'_{w_O}}^{\top}v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^{\top}v_{w_I}\right)}$$

optimized…
+ hierarchical softmax
+ negative sampling
+ subsample frequent w's

[Mikolav et al., 2013]

100 Dimensional Skip-gram embeddings, projected to two dimensions by PCA



Skip-gram Word Embeddings

analogy: w1 is to w2, as w3 is to ?w
algorithm: ?w = w2-w1+w3

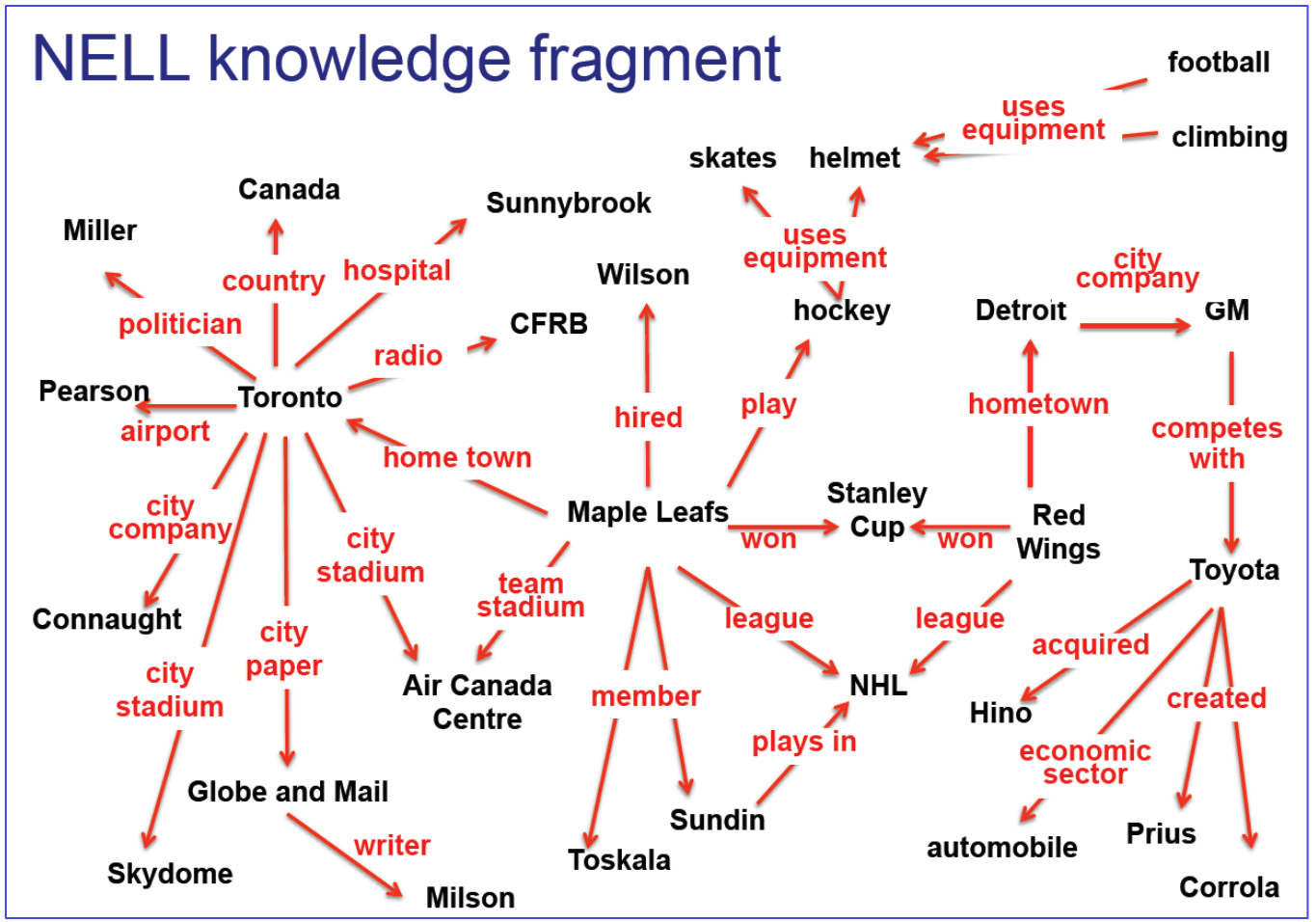
Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

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

[Mikolav et al., 2013]

Learning Representations for Words and Relations

NELL (Never Ending Language Learner) is learning to read the web, building large knowledge graph



[Yang & Mitchell, 2017]

Learning Representations for Words and Relations

Idea: Learn embeddings for each entity, relation <e_1,r,e_2>

e.g., <coffee, IsA, beverage>, <pittsburgh, homeToTeam, steelers>

Each entity e assigned a vector embedding v_e
Each relation r assigned a matrix embedding M_r

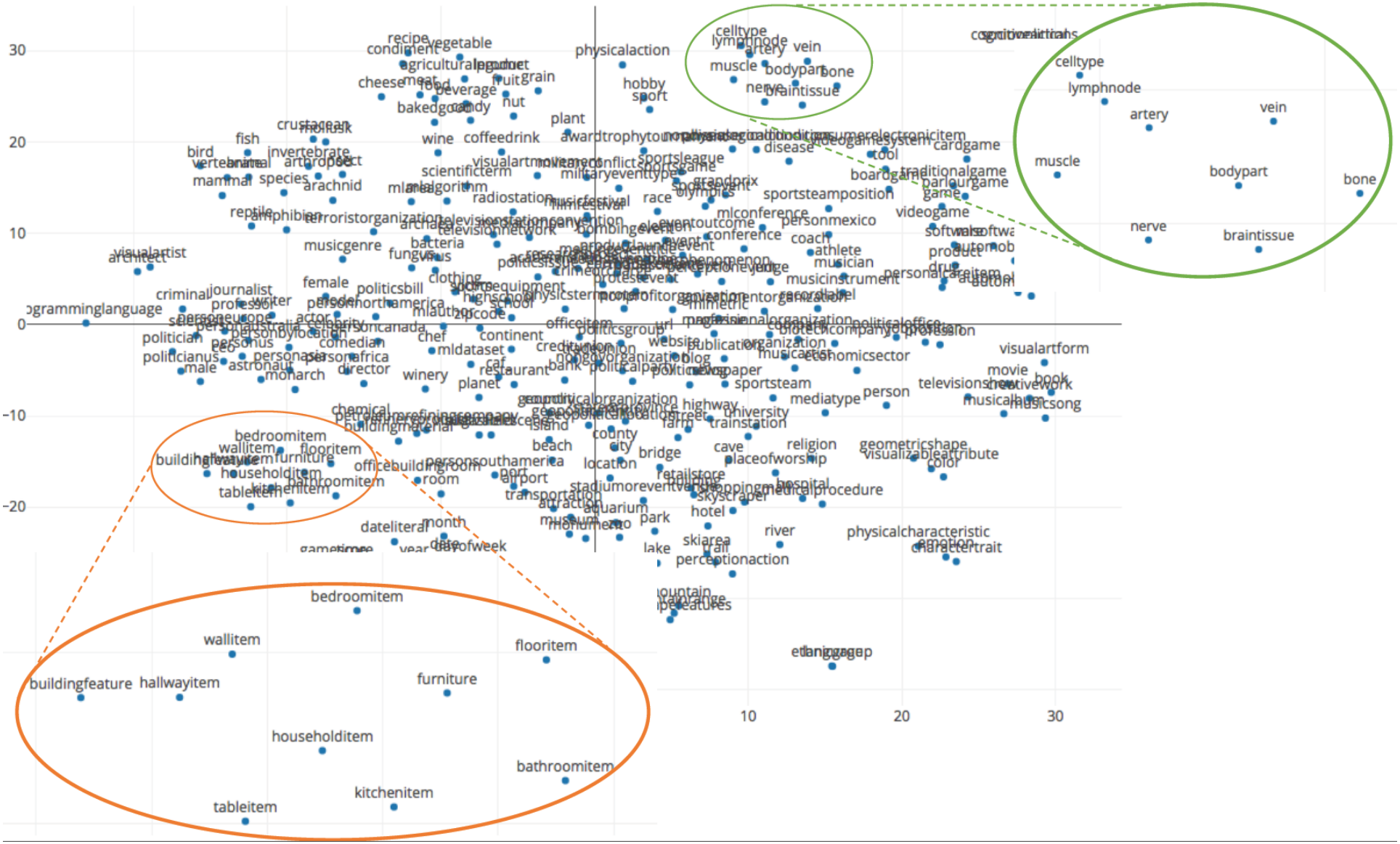
Train v's and M's jointly so that $S_{(e_1,r,e_2)} = \mathbf{v}_{e_1}^T \mathbf{M}_r \mathbf{v}_{e_2}$
is high iff <e_1,r,e_2> is true (i.e., in NELL's knowledge base)

Train on 180,107 NELL entities from 258 semantic categories
to minimize $J(\theta) = \sum_{q=(e_1,r,e_2)\in True} \sum_{q'=(e_1,r,e_2')\in False} max(0, 1-(S_q - S_{q'}))$

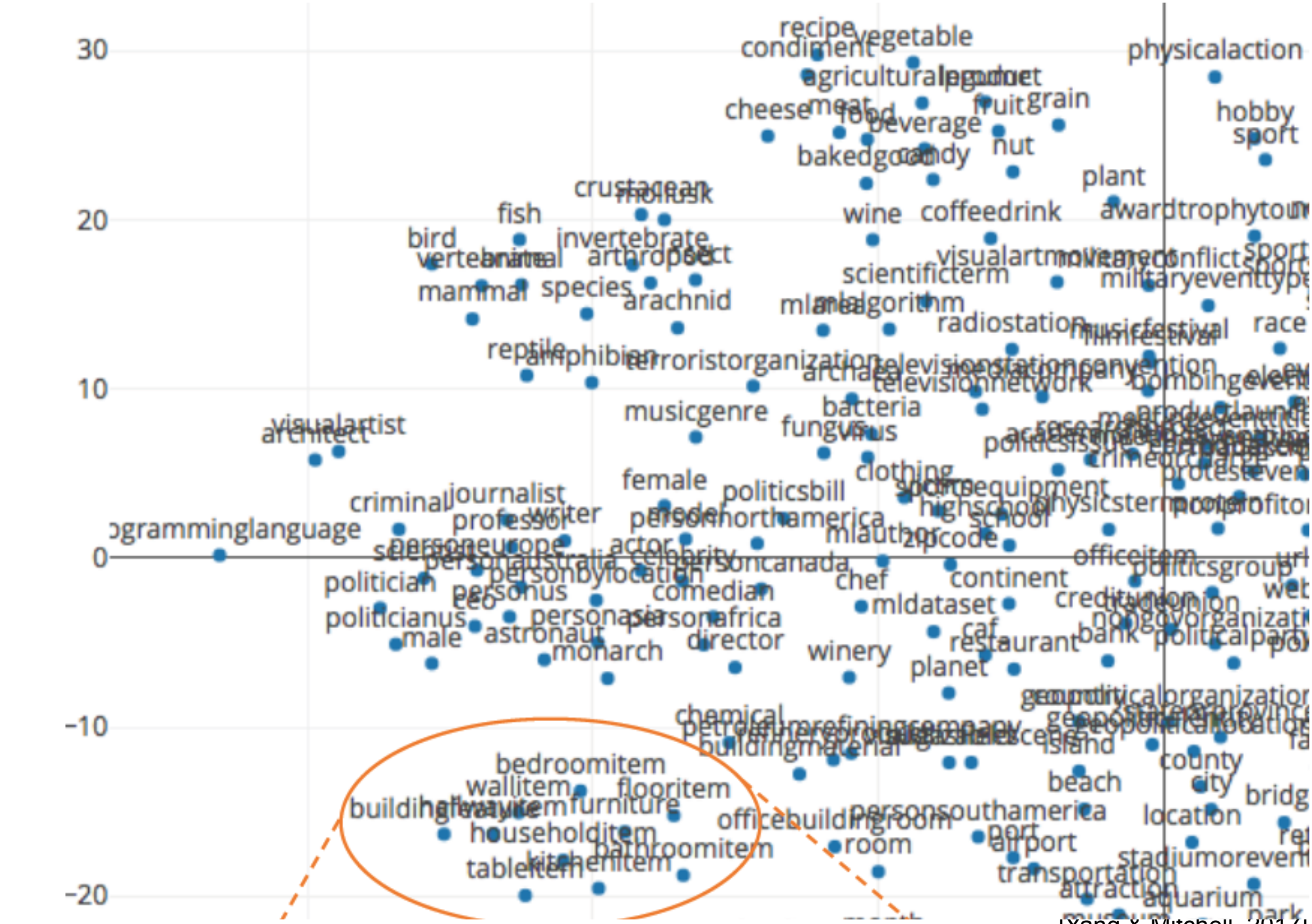
Results: 0.88 accuracy classifying category of noun phrases (1 of 258)

[Yang & Mitchell, 2017]

Learned Embeddings for NELL's Categories

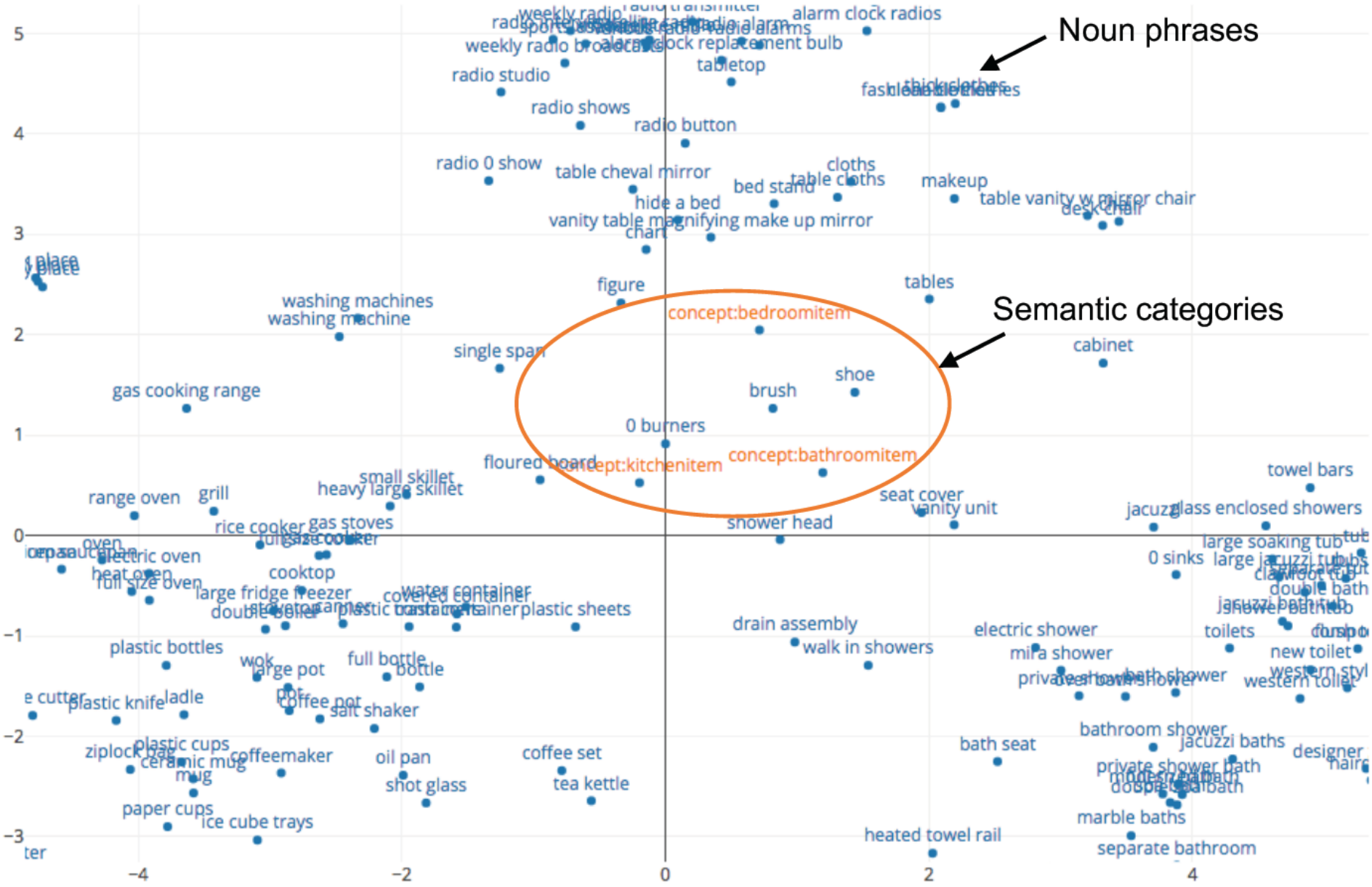


[Yang & Mitchell, 2017]



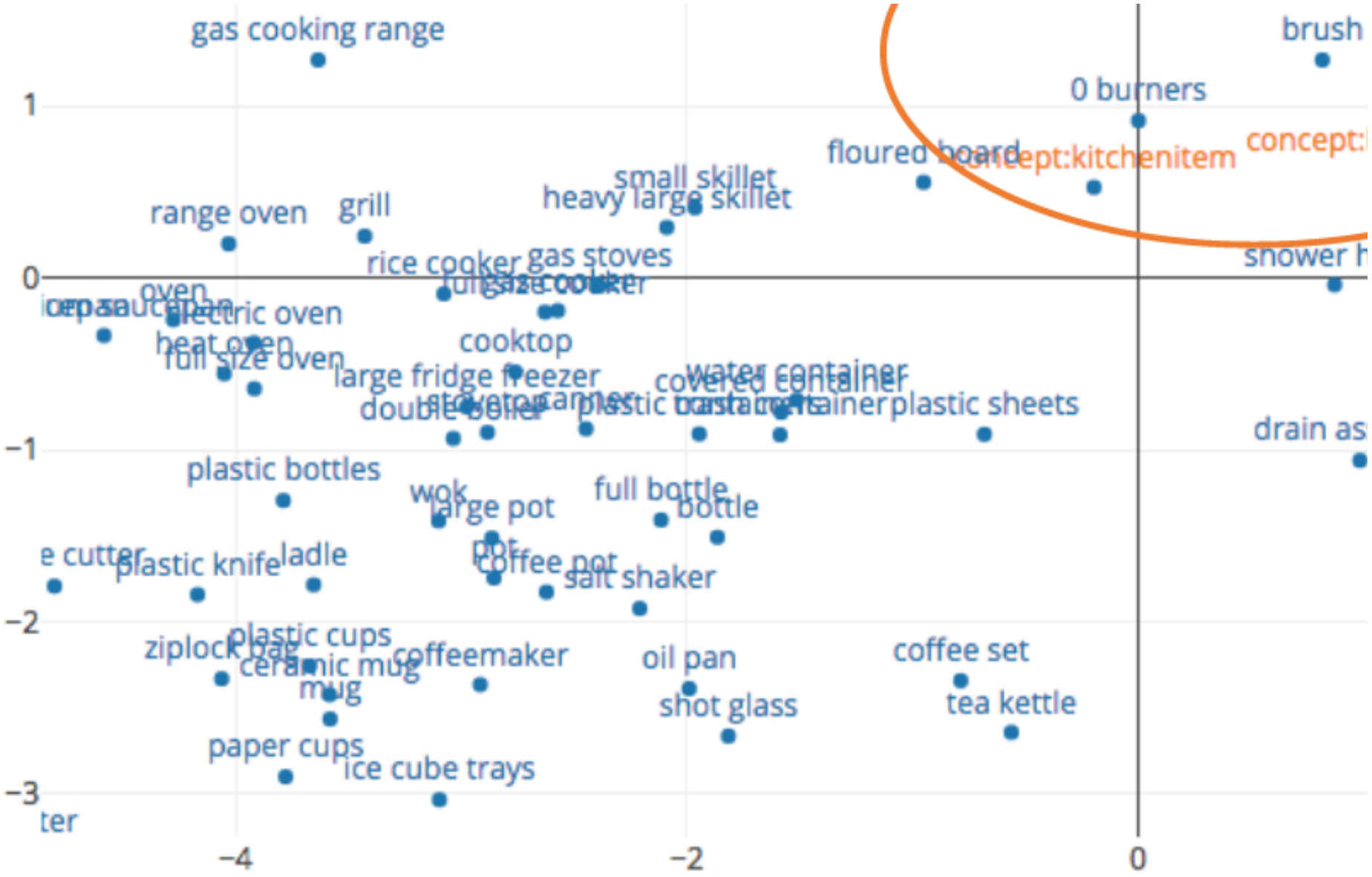
[Yang & Mitchell, 2017]

Learned Embeddings for NELL Categories and Entities



[Yang & Mitchell, 2017]

Learned Embeddings for NELL Categories and Entities



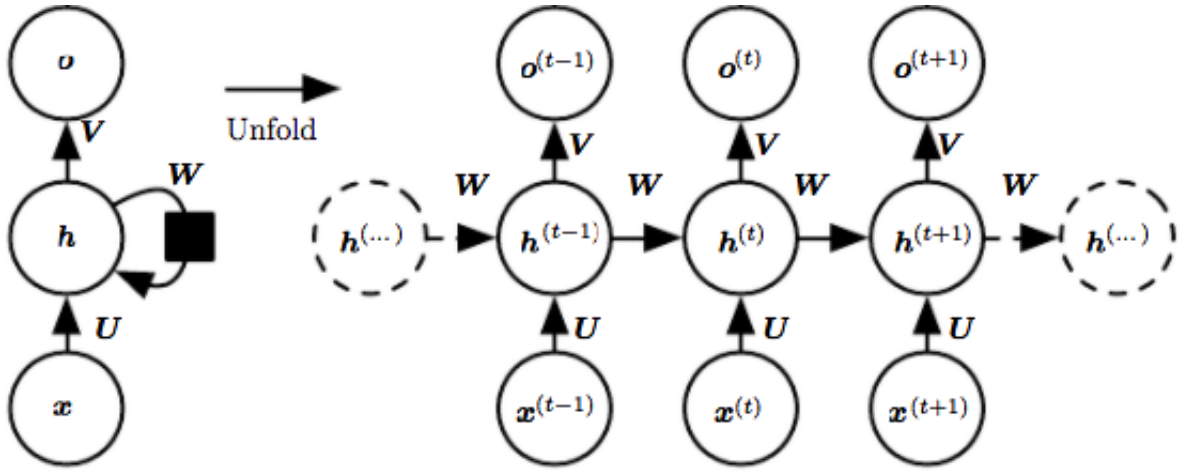
[Yang & Mitchell, 2017]

Sequential Neural Nets

Recurrent Networks

- Many tasks involve sequential data
  - predict stock price at time t+1 based on prices at t, t-1, t-2, …
  - translate sentences (word sequences) from Spanish to English
  - transcribe speech (sound sequences) to text (word sequences)

- Key idea: recurrent network uses (part of) its state at t as input for t+1

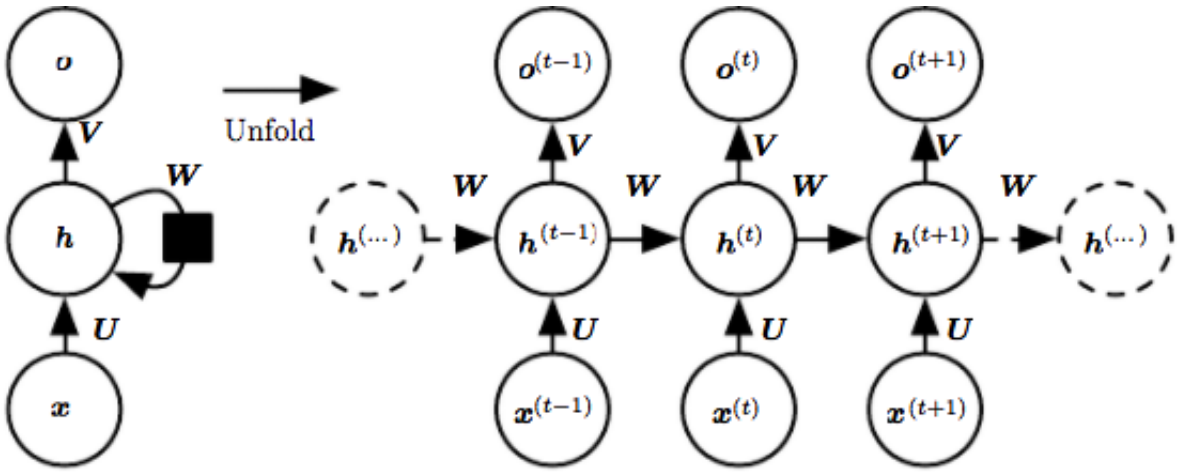


[Goodfellow et al., 2016]

Training Recurrent Networks

Key principle for training:

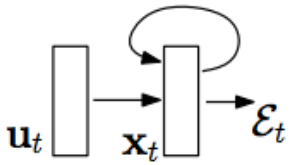
1. Treat as if unfolded in time, resulting in directed acyclic graph
2. Note shared parameters in unfolded net → sum the gradients



* problem: vanishing and/or exploding gradients

[Goodfellow et al., 2016]

Gradient Clipping: Managing exploding gradients



$$\mathbf{x}_t = \mathbf{W}_{rec}\sigma(\mathbf{x}_{t-1}) + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b}$$

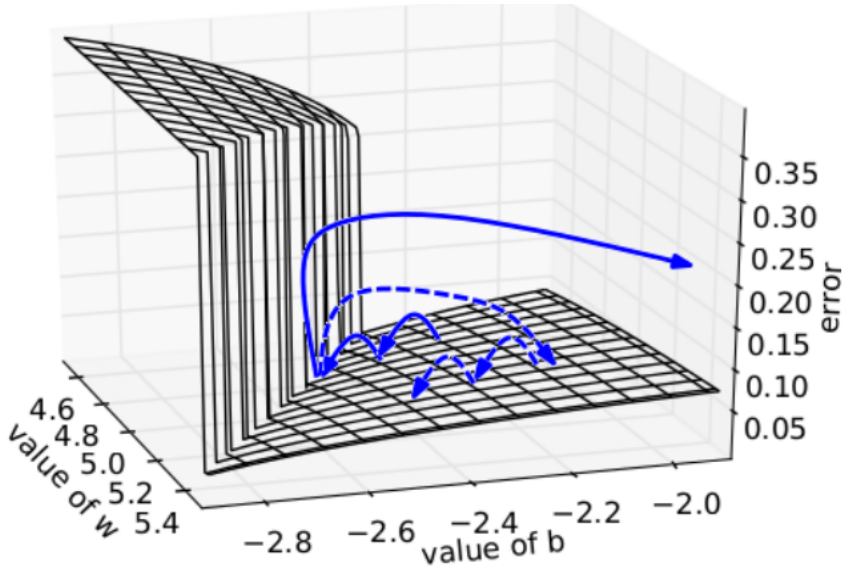


Figure 6. We plot the error surface of a single hidden unit recurrent network, highlighting the existence of high curvature walls. The solid lines depicts standard trajectories that gradient descent might follow. Using dashed arrow the diagram shows what would happen if the gradients is rescaled to a fixed size when its norm is above a threshold.

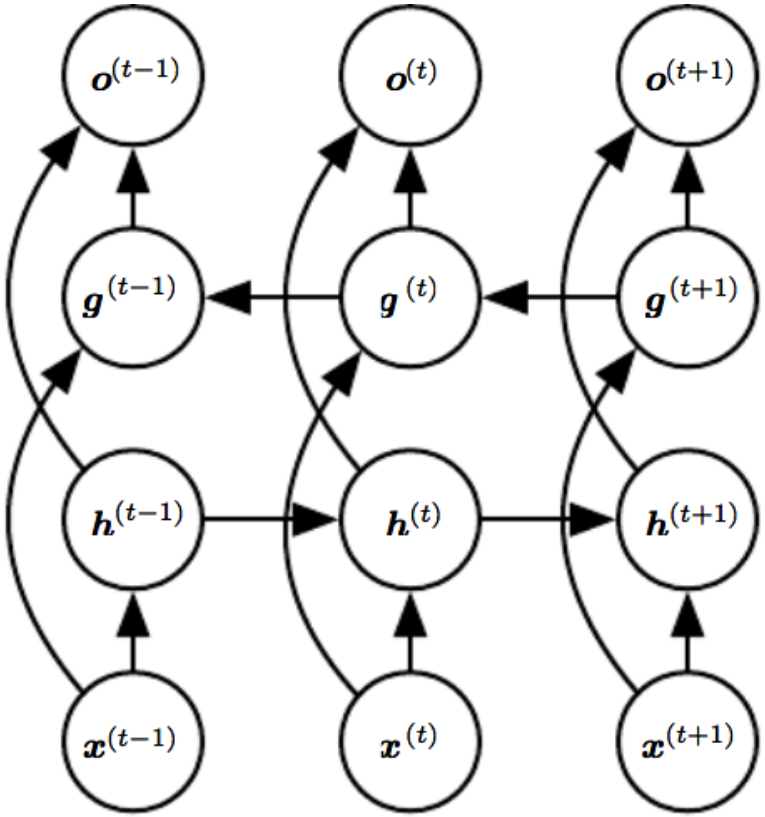
Algorithm 1 Pseudo-code for norm clipping dients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

[Pascanu et al., 2013]

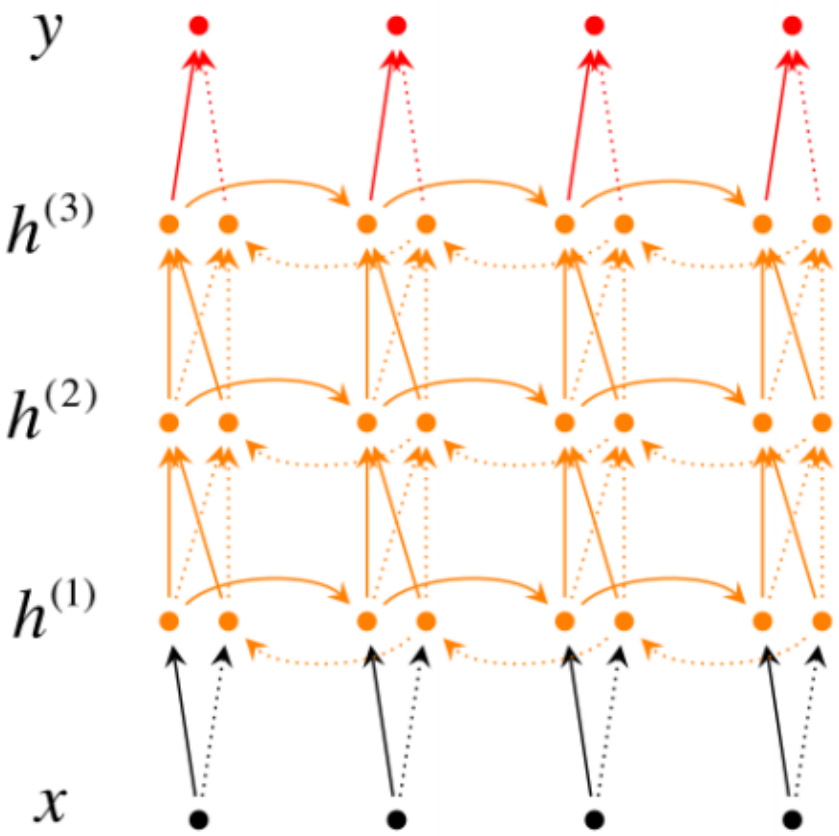
Bi-directional Recurrent Neural Networks

- Key idea: processing of word at position t can depend on following words too, not just preceding words



[Goodfellow et al., 2016]

Deep Bidirectional Recurrent Network



$$\vec{h}_t^{(i)} = f(\vec{W}^{(i)} h_t^{(i-1)} + \vec{V}^{(i)} \vec{h}_{t-1}^{(i)} + \vec{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\vec{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Each bidirectional layer builds on the one below

[Irsoy & Cardie, 2014]

Deep Bidirectional Recurrent Network: Opinion Mining

(4)

[In any case] , [it is high time] that a social debate be organized ...

DEEPRNN [In any case] , it is HIGH TIME that a social debate be organized ...

SHALLOW In ANY case , it is high TIME that a social debate be organized ...

(5)

Mr. Stoiber [has come a long way] from his refusal to [sacrifice himself] for the CDU in an election that [once looked impossible to win] , through his statement that he would [under no circumstances] run against the wishes...

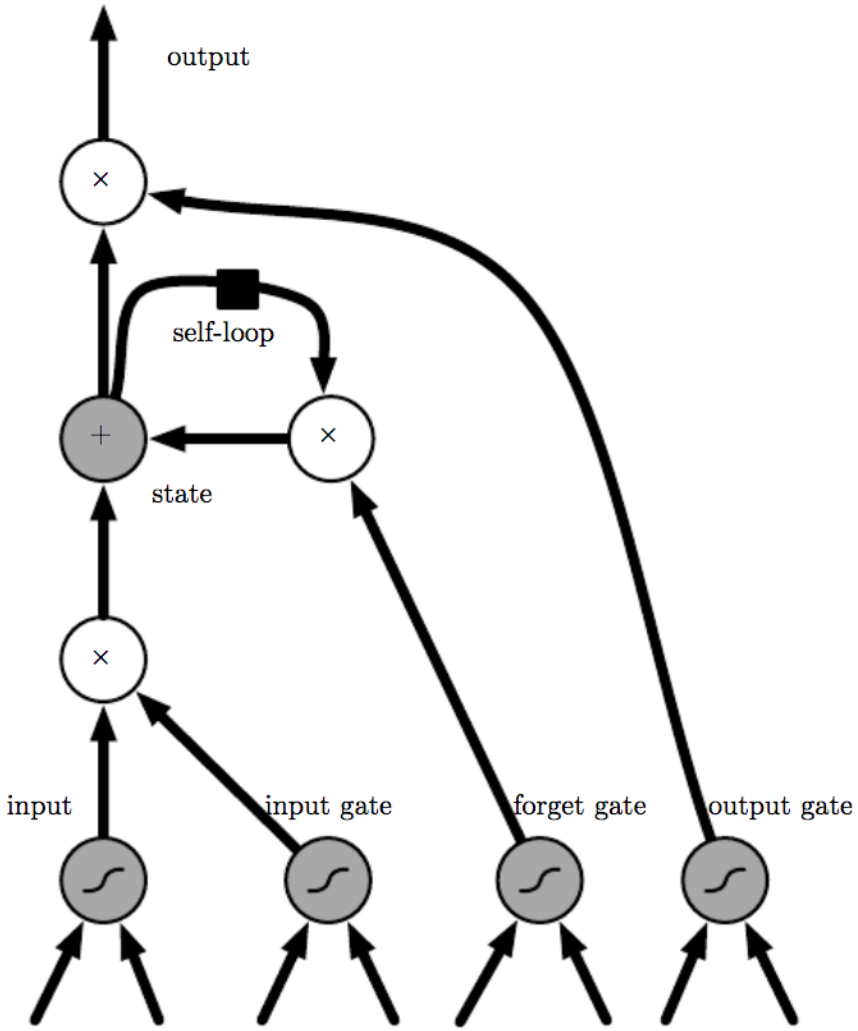
DEEPRNN Mr. Stoiber [has come a long way from] his [refusal to sacrifice himself] for the CDU in an election that [once looked impossible to win] , through his statement that he would [under no circumstances run against] the wishes...

SHALLOW Mr. Stoiber has come A LONG WAY FROM his refusal to sacrifice himself for the CDU in an election that [once looked impossible] to win , through his statement that he would under NO CIRCUMSTANCES run against the wishes...

Figure 3: DEEPRNN Output vs. SHALLOWRNN Output. In each set of examples, the gold-standard annotations are shown in the first line. Tokens assigned a label of Inside with no preceding Begin tag are shown in ALL CAPS.

[Irsoy & Cardie, 2014]

Long Short Term Memory (LSTM) unit



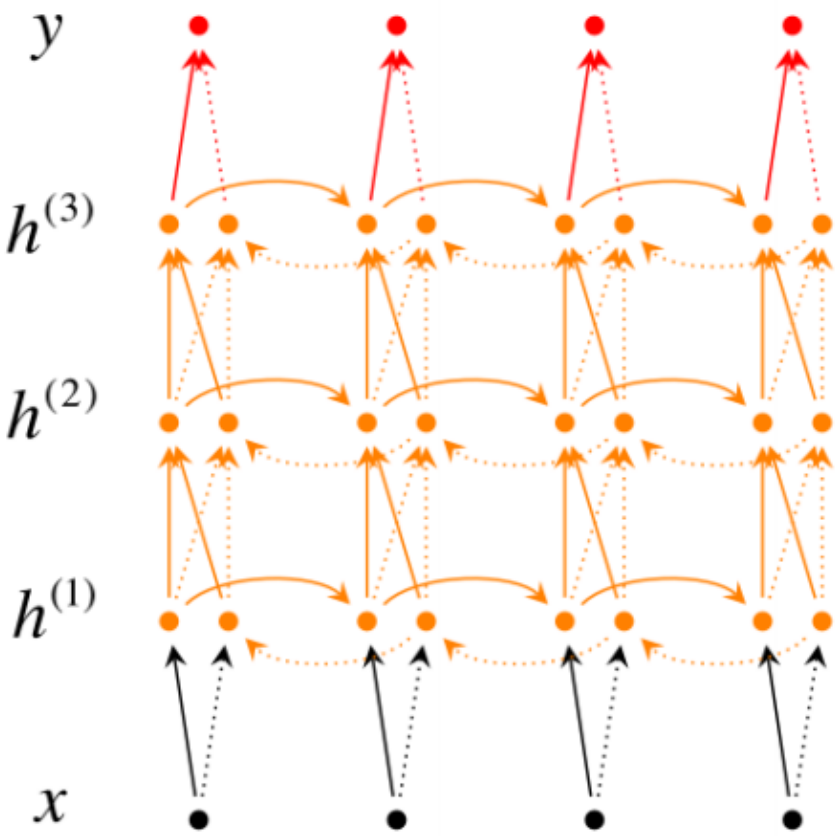
input gate: if 1, add input to memory state

forget gate: if 0, zero out memory state, else retain (partially)

output gate: if 1, read memory to output

[Goodfellow et al., 2016]

Deep Bidirectional Recurrent Network



$$\vec{h}_t^{(i)} = f(\vec{W}^{(i)} h_t^{(i-1)} + \vec{V}^{(i)} \vec{h}_{t-1}^{(i)} + \vec{b}^{(i)})$$

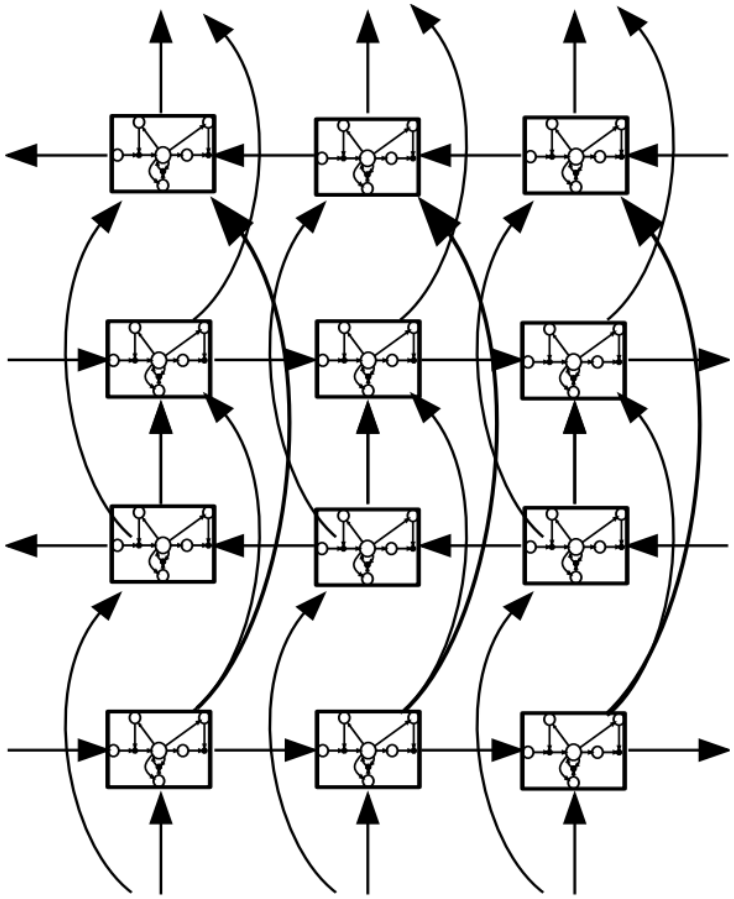
$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\vec{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Each bidirectional layer builds on the one below

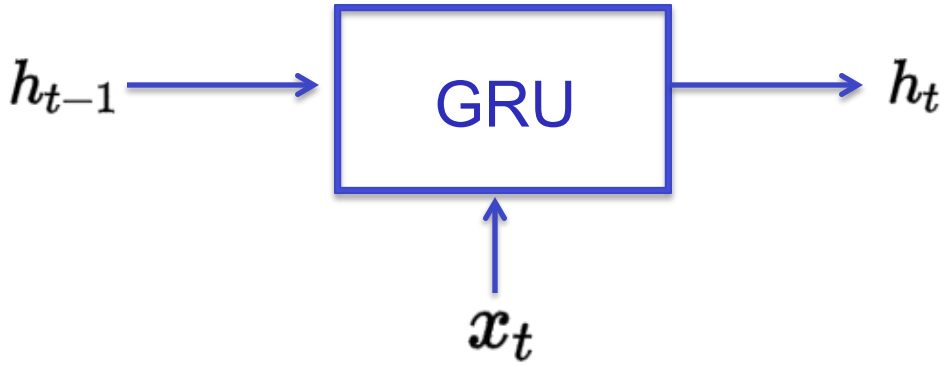
[Irsoy & Cardie, 2014]

Deep Bidirectional LSTM Network



["Hybrid Speech Recognition with Deep Bidirectional LSTM," Graves et al., 2013]

Gated Recurrent Units (GRUs)



∘ denotes the Hadamard product. $h_0 = 0$.

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

Variables

- x_t: input vector
- h_t: output vector
- z_t: update gate vector
- r_t: reset gate vector
- W, U and b: parameter matrices and vector

Activation functions

- σ_g: The original is a sigmoid function.
- σ_h: The original is a hyperbolic tangent.

fewer parameters than LSTM
found equally effective for
- speech recognition
- music analysis

see [Chung et al., 2014]

Programming Frameworks for Deep Nets

- TensorFlow (Google)
- TFLearn (runs on top of TensorFlow, but simpler to use)
- Theano (University of Montreal)
- Pytorch (Facebook)
- CNTK (Microsoft)
- Keras (can run on top of Theano, CNTK, TensorFlow)

Many support use of Graphics Processing Units (GPU's)

Major factor in dissemination of Deep Network technology

```
# Specify that all features have real-value data
feature_columns = [tf.feature_column.numeric_column("x", shape=[4])]

# Build 3 layer DNN with 10, 20, 10 units respectively.
classifier = tf.estimator.DNNClassifier(feature_columns=feature_columns,
                                        hidden_units=[10, 20, 10],
                                        n_classes=3,
                                        model_dir="/tmp/iris_model")
# Define the training inputs
train_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": np.array(training_set.data)},
    y=np.array(training_set.target),
    num_epochs=None,
    shuffle=True)

# Train model.
classifier.train(input_fn=train_input_fn, steps=2000)

# Define the test inputs
test_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": np.array(test_set.data)},
    y=np.array(test_set.target),
    num_epochs=1,
    shuffle=False)

# Evaluate accuracy.
accuracy_score = classifier.evaluate(input_fn=test_input_fn)["accuracy"]

print("\nTest Accuracy: {0:f}\n".format(accuracy_score))
```

TensorFlow example

What you should know:

- Representation learning in neural networks
  - hidden layers re-represent inputs to predict outputs
  - auto-encoders
  - word embeddings
- Sequential models
  - recurrent networks, and unfolding them
  - memory units: LSTM, etc.
  - deep sequential neural networks
- Pragmatics of training deep nets
  - vanishing gradients and exploding gradients
  - gradient clipping, batch normalization, …
  - frameworks such as TensorFlow, Pytorch, ...