Deep Learning for Natural Language Processing

Dylan Drover, Borui Ye, Jie Peng

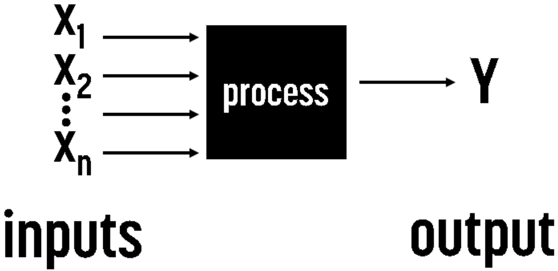
University of Waterloo

djdrover@uwaterloo.ca
borui.ye@uwaterloo.ca

July 8, 2015

Overview

A Black Box with a Billion Dials

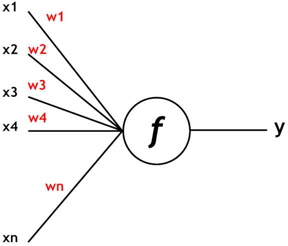


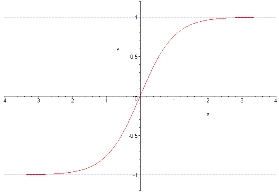
But we can do better...

The Artificial Neuron

- Base unit for (most) neural networks is a simplified version of a biological neuron
- Neuron has a set of inputs which have associated weights, an activation function which then determines whether a neuron will "fire" or be activated
- Together these can form very complex **function modellers/approximators**

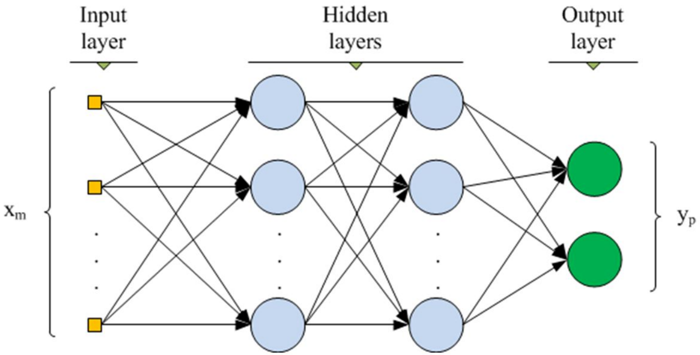
The Artificial Neuron





- Output y is some function of the sum of the weights and the inputs. $y = f\left(\sum_{j=0}^{m} w_{kj} x_j\right)$
- Each neuron has a weight vector and each layer has a weight matrix W

Multilayer Perceptron



- Performs multiple logistic regressions at once for arbitrary function approximation
- The multilayer perceptron which you might consider "deep" but isn't. It suffers from the "**vanishing gradient problem"**

Backpropogation: Learning with Derivatives

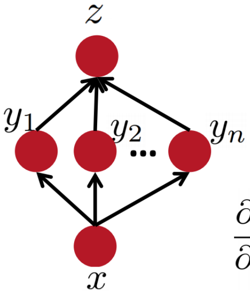
The basic idea behind learning in a neural network is that a network will have:

- **Objective function**: $J(\theta)$, $E(v, h)$ or training vector
  - This applies for supervised and unsupervised learning
  - The network's output is compared with objective to obtain an error
- **Optimization algorithm**: Stochastic Gradient Descent, Contrastive Divergence etc.
  - These algorithms direct learning within the "weight space"

But how do we adjust weights to optimize the objective?

Chain Rule

- Use the chain rule to determine how each output effects the final desired output
- Now have many **gradients** that we can use for optimization



$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Backpropogation

Calculate network error (where, t is the target, y is the network output)

$$E = \frac{1}{2}(t - y)^2$$

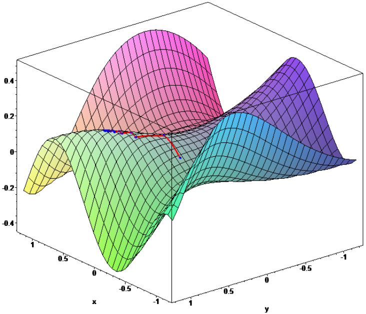
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \mathrm{net_j}} \frac{\partial \mathrm{net_j}}{\partial w_{ij}}$$

$$\frac{\partial \mathrm{net_j}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{k=1}^{n} w_{kj} x_k \right) = x_i$$

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$$

High Dimensional Optimization Problem: Weight Space

- Training a neural network is a **N-dimensional optimization problem**
- Weight in a NN, is a **dimension** and the goal is to mind the minimum error ("hight") with gradient descent
- There are many optimization algorithms for finding weights
  - Hessian-Free Optimization, Stochastic Gradient Descent, RMSProp, AdaGrad, Momentum, etc.

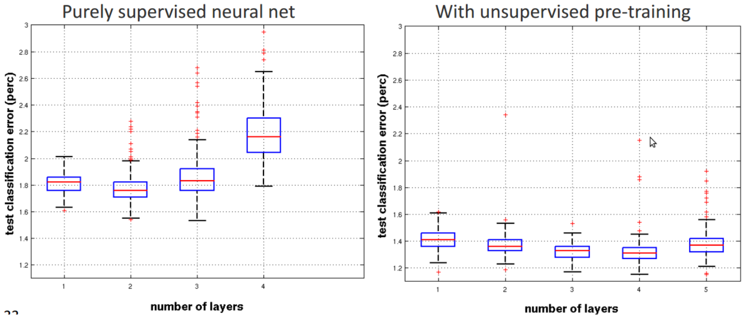


What is "Deep Learning" and why should we care?

- Deep Learning is just a re-branding of artificial neural networks
- Multiple factors led to the new **deeper NN architectures**
  - Pre-training (unsupervised)
  - Faster optimization algorithms
  - Graphic Processing Units (GPU)
- A myriad of techniques existed previously but things began to come together in the mid 2000s

Better Results Through Pre-Training

- Train each layer of network **greedily** using other **layers output as input with unsupervised learning**
- Combine layers and use fine tune training as in MPL
- Network starts with better position in weight space



MNIST error rates [Erhan et al., JMLR 2010]

Promise for NLP

Improved results for Part of Speech tagging and Named Entity Recognition

	POS WSJ (acc.)	NER CoNLL (F1)
State-of-the-art*	97.24	89.31
Supervised NN	**96.37**	**81.47**
Unsupervised pre-training followed by supervised NN**	**97.20**	**88.87**
+ hand-crafted features***	97.29	89.59

* Representative systems: POS: (Toutanova et al. 2003), NER: (Ando & Zhang 2005)

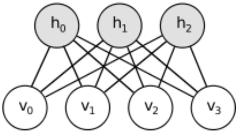
** 130,000-word embedding trained on Wikipedia and Reuters with 11 word window, 100 unit hidden layer – **for 7 weeks!** – then supervised task training

***Features are character suffixes for POS and a gazetteer for NER

Application of Deep Belief Networks for Natural Language Understanding [Sarikaya et al.]

- Sarikaya et al. attempt to solve the problem of **spoken language understanding** (SLU) in the context of call-routing (call-centre data)
- **Data**:
  - 27 000 transcribed utterances serve as unlabelled data
  - Sets of 1K, 2K, 3K, 4K, 5K, 6K, 7K, 8K, 9K, 10K are used as labelled data sets
- Restricted Boltzmann Machines (RBM) were trained with unlabelled data and then stacked to form a Deep Belief Network

Restricted Boltzmann Machine



- Composed of **visble** and **hidden** neurons
- Unlabelled training data is presented as **v**
- Hidden neurons (stochastic binary units) and weights attempt to approximate a joint probability distribution of data (**Generative Model**)

Learning Without Knowing

Joint energy distribution of RBM is defined as:

$$E(v,h) = -\sum_i a_i v_i - \sum b_i h_i - \sum \sum v_i w_{i,j} h_j$$

Which defines the probability distribution:

$$p(\mathbf{v,h}) = \frac{1}{Z} e^{-E(v,h)}$$

Which is marginalized over over hidden vectors:

$$p(\mathbf{v}) = \frac{1}{Z} \sum_h e^{-E(v,h)}$$

Z acts as a normalizing factor:

$$Z = \sum_{v,h} e^{-E(v,h)}$$

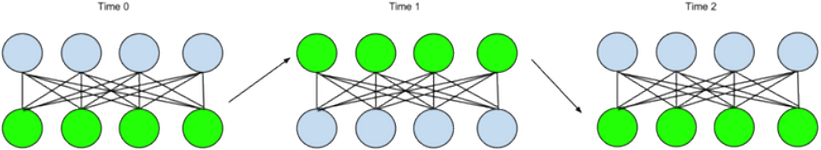
Learning without Knowing

Training uses a "reconstruction" (the math is easier)of the input vector that is achieved in the hidden units with:

$$p(h_j = 1|\mathbf{v}) = \sigma(a_j + \sum_i v_i w_{ij})$$

Which is used in:

$$p(v_i = 1|\mathbf{h}) = \sigma(b_i + \sum_j h_j w_{ij})$$



Learning without Knowing

- The hidden units compare their rough reconstruction to the actual input.
- Based on the reconstruction, the weights are changed to improve

$$\frac{\partial logp(v)}{\partial w_{ij}} = \langle v_i h_j \rangle_v - \langle v_i h_j \rangle_{model}$$

$$\Delta w_{ij} \propto \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$$

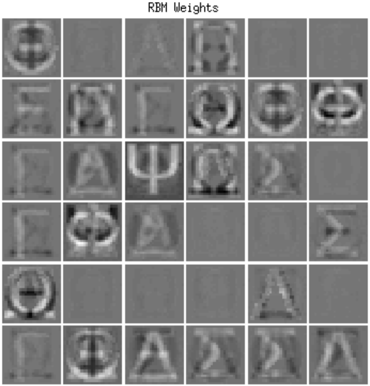
$$\Delta w_{ij} \propto \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}$$

- This is a rough approximation, however it works well in practice.

*(Angle brackets are the expectation with respect to the subscript distribution)

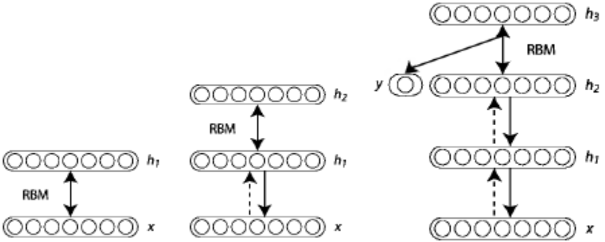
Visualization of Weights from a RBM

- Each square is a set of weights for one neuron
- Features emerge from unsupervised learning



Deep Belief Network

- Use one RBM to train the next layer RBM
  - Each layer learns **features**
  - Each successive layer learns **features of features**
- This continues until a **supervised layer**
- Results in a **Deep Belief Network**



(a) Train RBM for $\mathbf{x}$ (b) Train RBM for $\mathbf{h}^1$ (c) Train RBM for $\mathbf{h}^2$ and y

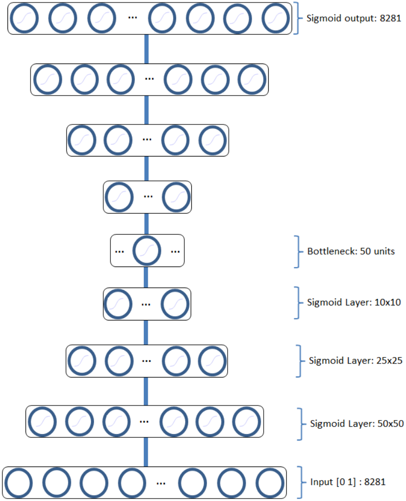
Results

Results show improvements from unsupervised learning in all aspects

Action Classification Accuracy (%)							
Labeled Data	MaxEnt	SVM	Boosting	DBN	DBN-1	DBN-2	DBN-3
1K	76.0	77.8	**79.6**	78.1	78.4	78.3	78.6
2K	80.4	82.2	83.6	82.6	83.7	83.4	**84.1**
3K	82.2	84.3	85.1	84.4	85.3	84.8	**85.6**
4K	83.5	85.3	84.6	85.5	86.4	85.9	**86.6**
5K	84.6	86.2	85.9	86.2	86.9	86.7	**87.4**
6K	85.5	87.0	86.3	87.0	87.4	87.3	**87.8**
7K	86.2	87.7	86.3	87.8	88.0	88.2	**88.4**
8K	86.5	88.0	87.2	88.0	88.0	88.1	**88.1**
9K	87.2	88.5	87.5	88.7	88.8	88.8	**88.9**
10K	87.6	88.5	87.7	88.7	88.7	**88.9**	**88.9**
27K	89.7	90.3	88.1	90.3	90.3	**90.8**	**90.8**

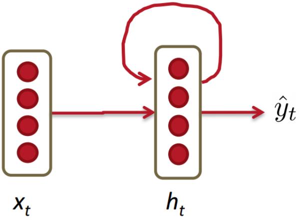
TABLE I
PACKAGE SHIPMENT TASK: ACCURACY FOR TRADITIONAL AND DBN BASED CLASSIFIERS.

Autoencoders

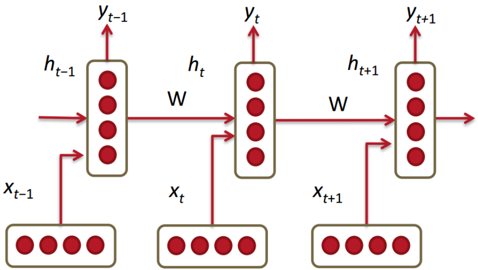


- Creates compressed representation of data (**Dimensionality reduction**)
- Central layer acts as a **non-linear principal component analysis**
- Decoder weights are transposed encoder weight matrices: W_{li} and W_{li}^T
- Each layer trained greedily (similar to DBN layers)

Recurrent Neural Networks

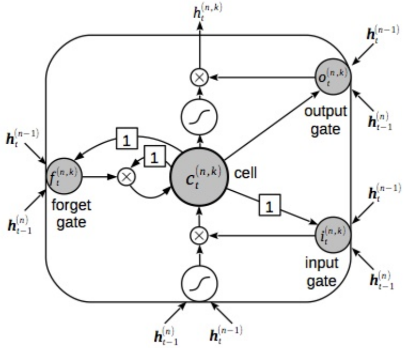


- Input is treated as time series: $\ldots, x_{t-1}, x_t, x_{t+1}, \ldots$
- Retain temporal context or **short term memory**
- Trained with **backpropogation through time (BPTT)**



Long Short Term Memory

- Prevent the "vanishing gradient" problem
- Can retain information for arbitrary amount of time



Learning Phrase Representations using RNN EncoderDecoder for Statistical Machine Translation [Cho et al.]

- Combination of a RNN with an autoencoder
- Encoder maps variable length source phrase $X = (x_1, x_2, ..., x_N)$ (English sentence) into a fixed length internal representation vector **c**
- The decoder then maps this back into another variable length sequence, the target sentence $Y = (y_1, y_2, ..., y_N)$ (French Sentence)
- Analysis showed that the internal representation (in the 1000 hidden units) preserved syntactic and **semantic information**
- Learns probability distribution:

$$p(y_1, ..., y_{T'} | x_1, ..., x_T)$$

Learning Phrase Representations using RNN EncoderDecoder for Statistical Machine Translation

- The decoder uses the compressed semantic meaning vector **c** as well as the previous word in its translation

$$\mathbf{h}_{\langle t \rangle} = f(\mathbf{h}_{\langle t-1 \rangle}, y_{t-1}, \mathbf{c})$$

- Next element in translated sequence is conditioned on:

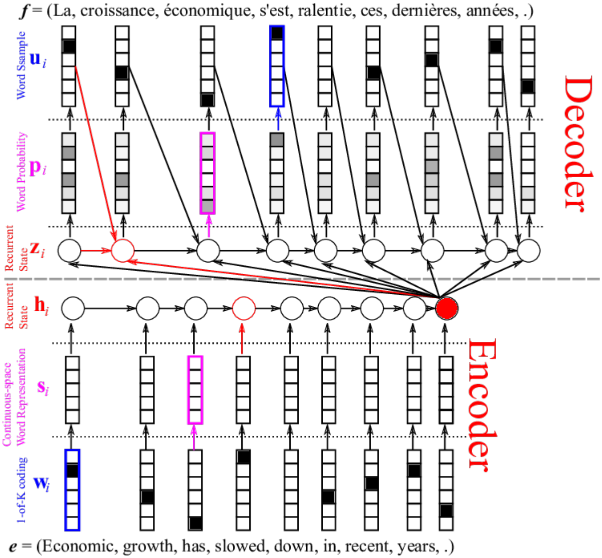
$$P(y_t | y_{t-1}, ..., y_1, \mathbf{c}) = g(\mathbf{h}_{\langle t-1 \rangle}, y_{t-1}, \mathbf{c})$$

- Training of the network attempts to maximize the conditional log likelihood of:

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log p_{\theta}(\mathbf{y}_n, \mathbf{x}_n)$$

where θ are the model parameters (weights) and $(\mathbf{y}_n, \mathbf{x}_n)$ are input and output pairs.

Encoder - Decoder



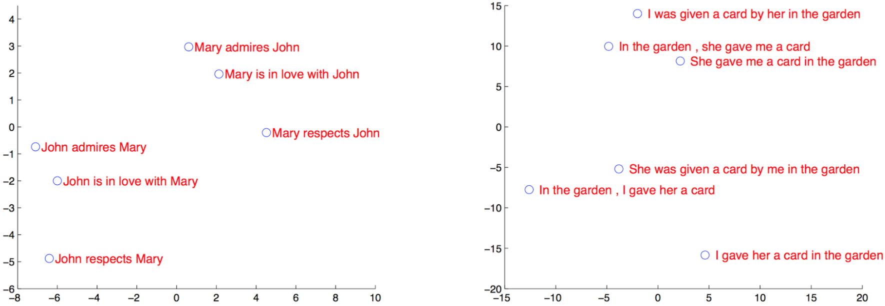
Big Data

- The training of the model used:
  - Bilingual corpora Europarl (61M words)
  - News commentary (5.5M words)
  - UN transcriptions (421M words)
  - 870M words from crawled corpora
- However to optimize results only a subset of 348M words for training translation
- Final BLEU scores for the model were 34.54

Learning Phrase Representations using RNN EncoderDecoder for Statistical Machine Translation

- This model is not restricted to just translation (which is why ANN are so useful and exciting)
- This model also created **semantic relations** between similar words and sentences from their continuous vector space representations (more on that later)

Results


Mary admires John
Mary is in love with John
Mary respects John
John admires Mary
John is in love with Mary
John respects Mary
I was given a card by her in the garden
In the garden , she gave me a card
She gave me a card in the garden
She was given a card by me in the garden
In the garden , I gave her a card
I gave her a card in the garden

Results

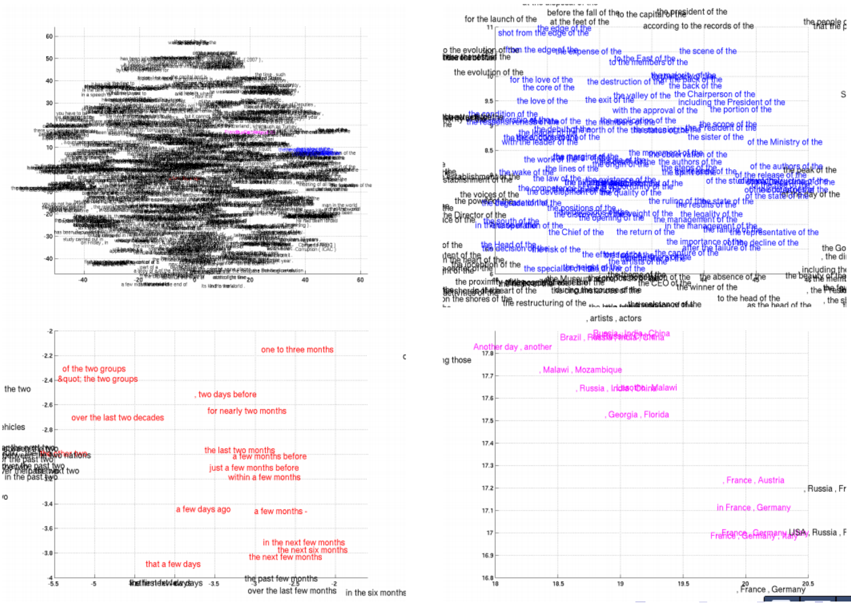
Source	Translation Model	RNN Encoder–Decoder
at the end of the	[a la fin de la] [ŕ la fin des années] [être supprimés à la fin de la]	[à la fin du] [à la fin des] [à la fin de la]
for the first time	[r © pour la premirëre fois] [été donnés pour la première fois] [été commémorée pour la première fois]	[pour la première fois] [pour la première fois ,] [pour la première fois que]
in the United States and	[? aux ?tats-Unis et] [été ouvertes aux États-Unis et] [été constatées aux États-Unis et]	[aux Etats-Unis et] [des Etats-Unis et] [des États-Unis et]
, as well as	[?s , qu'] [?s , ainsi que] [?re aussi bien que]	[, ainsi qu'] [, ainsi que] [, ainsi que les]
one of the most	[?t ?l' un des plus] [?l' un des plus] [être retenue comme un de ses plus]	[l' un des] [le] [un des]

(a) Long, frequent source phrases

Source	Translation Model	RNN Encoder–Decoder
, Minister of Communications and Transport	[Secrétaire aux communications et aux transports :] [Secrétaire aux communications et aux transports]	[Secrétaire aux communications et aux transports] [Secrétaire aux communications et aux transports :]
did not comply with the	[vestimentaire , ne correspondaient pas à des] [susmentionnée n' était pas conforme aux] [présentées n' étaient pas conformes à la]	[n' ont pas respecté les] [n' était pas conforme aux] [n' ont pas respecté la]
parts of the world .	[© gions du monde .] [régions du monde considérées .] [région du monde considérée .]	[parties du monde .] [les parties du monde .] [des parties du monde .]
the past few days .	[le petit texte .] [cours des tout derniers jours .] [les tout derniers jours .]	[ces derniers jours .] [les derniers jours .] [cours des derniers jours .]
on Friday and Saturday	[vendredi et samedi à la] [vendredi et samedi à] [se déroulera vendredi et samedi ,]	[le vendredi et le samedi] [le vendredi et samedi] [vendredi et samedi]

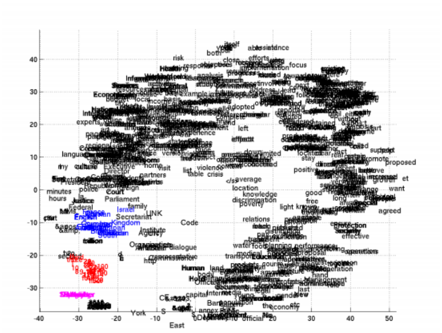
(b) Long, rare source phrases

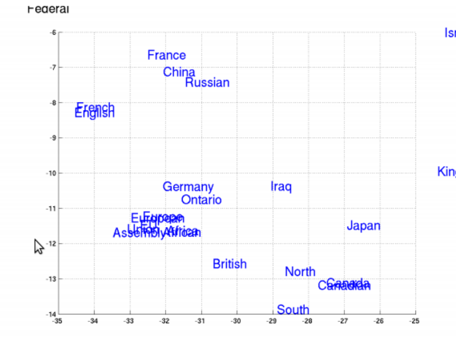
Results



one to three months
of the two groups
" the two groups
, two days before
for nearly two months
over the last two decades
the last two months
a few months before
just a few months before
within a few months
a few days ago
a few months -
in the next few months
the next six months
the next few months
that a few days
the past few months
over the last few months
in the six months
, artists , actors
Brazil ,
Another day , another
, Malawi , Mozambique
, Georgia , Florida
, France , Austria
in France , Germany
, France , Germany
according to the records of the
the scene of the
the evolution of the
the destruction of the
the love of the
the sister of the
of the Ministry of the
the Director of the
the Chief of the
the return of the
the absence of the
the winner of the
to the head of the
the restructuring of the

Results





Federal
France
China
Russian
French
English
Germany
Ontario
Iraq
Japan
British
North
Canada
Canadian
South

Neural Probabilistic Language Model

Presenter: Borui Ye

Papers:

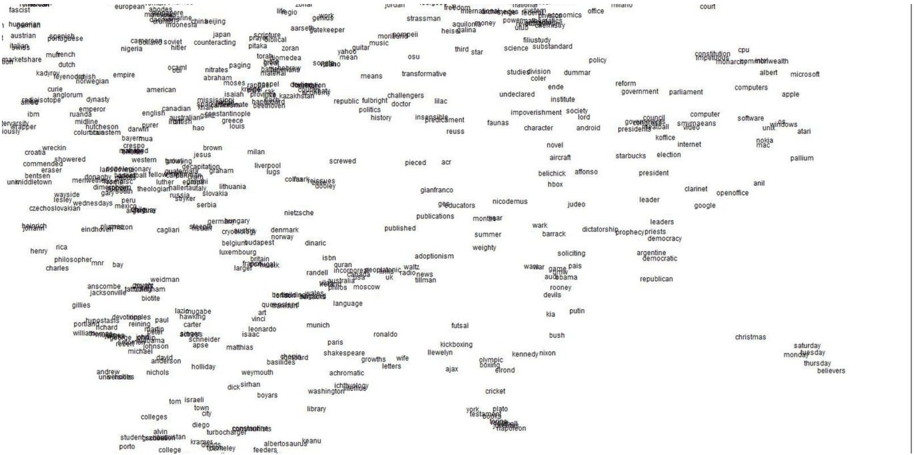
1. Efficient Estimation of Word Representations in Vector Space
2. A Neural Probabilistic Language Model

What is Word Vector

One-hot representation: represents a word using a long vector. For example:
"microphone" is represented as : [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 ...]
"phone" is represented as : [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ...]

- **PROS**: this method can coordinate well with max entropy, SVM, CRF algorithm.
- **CONS**: word gap

What is Word Vector (Cont.)



How To Train Word Vector

Language Model In practice, we usually need to calculate the probability of a sentence:

$$\begin{aligned} P(S) &= p(w_1, w_2, w_3, w_4, w_5, ..., w_n) \\ &= p(w_1)p(w_2|w_1)p(w_3|w_1, w_2)...p(w_n|w_1, w_2, ..., w_{n-1}) \end{aligned}$$

Markov's Assumption Each word depends only on the last $n-1$ words.

$$\begin{aligned} P(S) &= p(w_1, w_2, w_3, w_4, w_5, ..., w_n) \\ &= p(w_1)p(w_2|w_1)p(w_3|w_1, w_2)...p(w_n|w_1, w_2, ..., w_{n-1}) \\ &\approx p(w_1)p(w_2|w_1)p(w_3|w_2)...p(w_n|w_{n-1}) \quad (bigram) \end{aligned}$$

How To Train Word Vector (Cont.)

Problems With N-gram Model:

- It is not taking into account contexts farther than 1 or 2 words
- Cannot capture the similarities among words.

Example:

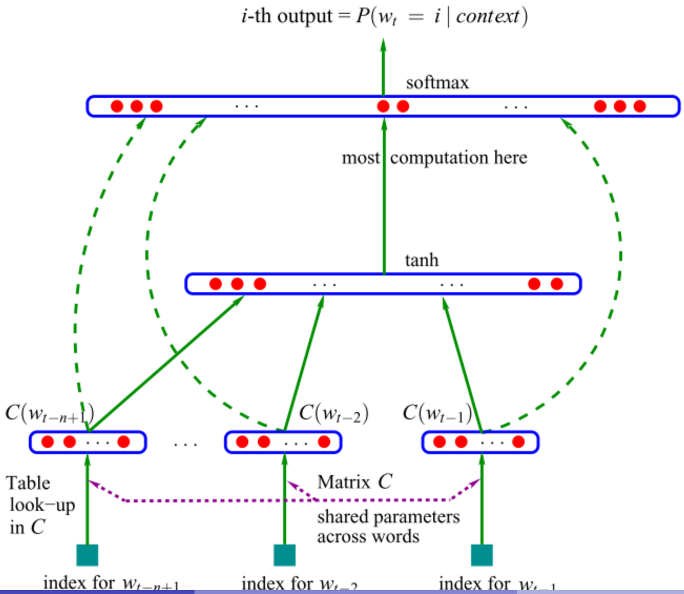
The cat is walking in the bedroom

A dog was running in a room

Bengio's Neural Network

Training Set a sequence $w_1 ... w_T$ of words $w_t \in V$, where the vocabulary V is a large but finite set.
Objective learn a model : $f(w_t, ..., w_{t-n+1}) = \hat{P}(w_t | w_1^{t-1})$
Constraint $\sum_{i=1}^{|V|} f(i, w_{t-1}, ..., w_{t-n+1}) = 1$, with $f > 0$

Bengio's Neural Network (Cont.)



Parameters

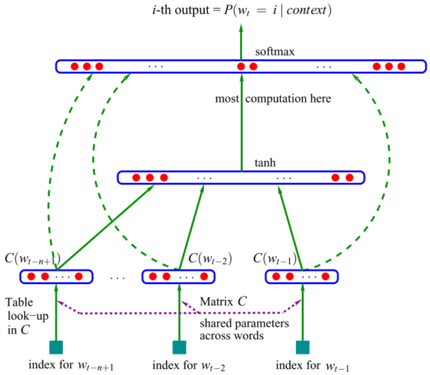
Definitions

C : a shared word vector matrix, $C \in \mathbb{R}^{|V|*m}$

x : vector of hidden layer, $x = (C(w_{t-1}), C(w_{t-2}), ..., C(w_{t-n+1}))$

y : vector of output layer, $y = b + Wx + Utanh(d + Hx)$

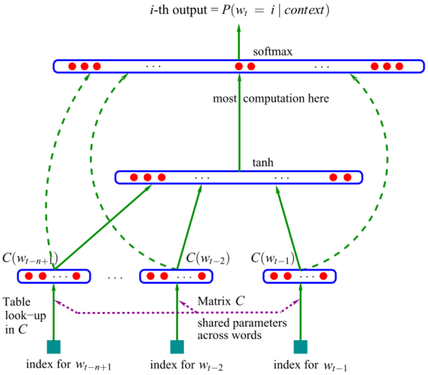
$P(w_t = i|context) = \hat{P}(w_t|w_{t-1}, ..., w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$



Parameter Estimation

The goal is to find the parameter that maximized the training corpus penalized log-likelihood: $L = \frac{1}{T}\sum_t \log f(w_t, w_{t-1}, \ldots, w_{t-n+1}; \theta) + R(\theta)$ where $\theta = (b, d, W, U, H, C)$

SGD: $\theta \leftarrow \theta + \epsilon \frac{\partial \hat{P}(w_t | w_{t-1}, \ldots, w_{t-n+1})}{\partial \theta}$



Google's Word2Vec

Project url : `http://code.google.com/p/word2vec/`

Feature : Additive Compositionality :

vector('Paris') - vector('France') + vector('Italy') $\approx$ vector('Rome')

vector('king') - vector('man') + vector('woman') $\approx$ vector('queen')

Google's Word2Vec (Cont.)

./distance

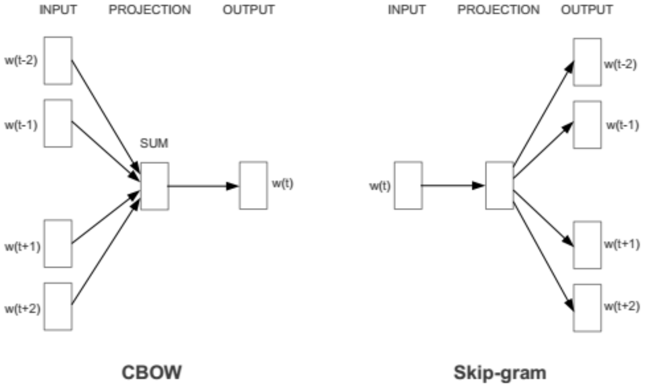
```
Enter word or sentence (EXIT to break): china

Word: china  Position in vocabulary: 486

                                              Word       Cosine distance
------------------------------------------------------------------------
                                            taiwan              0.768188
                                             japan              0.652825
                                             macau              0.614888
                                             korea              0.614887
                                               prc              0.613579
                                           beijing              0.605946
                                            taipei              0.592367
                                          thailand              0.577905
                                          cambodia              0.575681
                                         singapore              0.569950
                                          republic              0.567597
                                          mongolia              0.554642
                                           chinese              0.551576
```

Two Models and Two Algorithms

Models	Continuous Bag of Words		Skip-gram	
Alg.	Hierarchical Softmax	Negative Sampling	Hierarchical Softmax	Negative Sampling



CBOW+Hierarchical Softmax

Predict the probability of a word given its context:

$$P(w|Context(w))$$

Learning objective : maximize log-likelihood:

$$\zeta = \sum_{w \in C} \log p(w|Context(w))$$

CBOW+Hierarchical Softmax (Cont.)

- **Input Layer** $2c$ word vectors in $Context(w)$:
  $v(Context(w)_1), v(Context(w)_2), ...v(Context(w)_{2c}) \in \mathbb{R}^m$
- **Projection Layer** Adding all the vectors in input layer:

$$x_w = \sum_{i=1}^{2c} v(Context(w)_i) \in \mathbb{R}^m$$

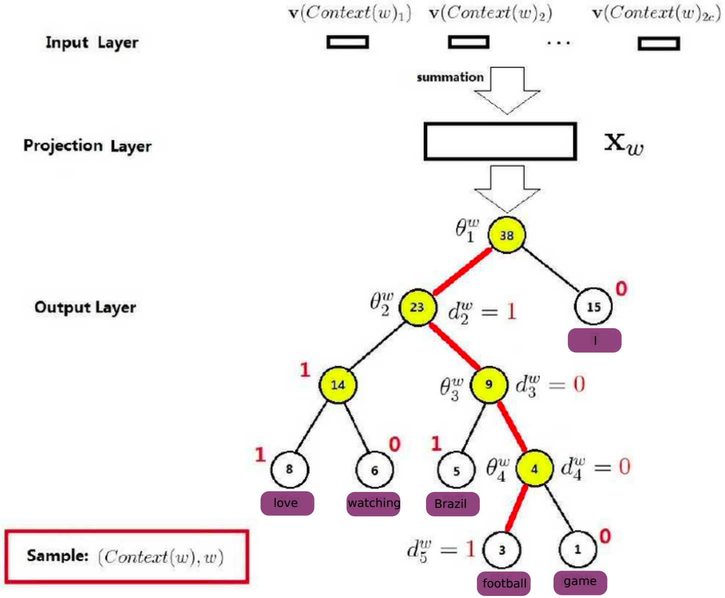
- **Output Layer** A Huffman tree using words in vocabulary as leaves.

CBOW+Hierarchical Softmax (Cont.)

Notations

- p^w: Path from root to corresponding leaf w
- l^w: Number of nodes included in p^w
- $p_1^w, p_2^w, ..., p_{l^w}^w$: l^w nodes of path p^w
- $d_2^w, d_3^w, ..., d_{l^w}^w \in \{0, 1\}$: Huffman code of each node on path p^w, root does not have code
- $\theta_1^w, \theta_2^w, ..., \theta_{l^w}^w$: vector of each node on path p^w,

Huffman Tree



Learning Objective

We assign every node a label:

$$Lable(p_i^w) = 1 - d_i^w, i = 2, 3, ..., l^w$$

So the probability of a node being classified as positive label is :

$$\delta(x_w^T \sigma) = \frac{1}{1 + e^{-x_w^T \theta}}$$

Then:

$$p(w|Context(w)) = \prod_{j=2}^{l_w} p(d_j^w | x_w, \theta_{j-1}^w)$$

where

$$p(d_j^w | x_w, \theta_{j-1}^w) = \begin{cases} \sigma(x_w^T \theta_{j-1}^w), & d_j^w = 0; \\ 1 - \sigma(x_w^T \theta_{j-1}^w), & d_j^w = 1; \end{cases}$$

Learning Objective

Full learning objective is to maximize:

$$\zeta = \sum_{w \in C} \log \prod_{j=2}^{l^w} \{[\sigma(x_w^T \theta_{j-1}^w)]^{1-d_j^w} [1 - \sigma(x_w^T \theta_{j-1}^w)]^{d_j^w}\}$$

$$= \sum_{w \in C} \sum_{j=2}^{l^w} \{(1 - d_j^w) \log[\sigma(x_w^T \theta_{j-1}^w)] + d_j^w \log[1 - \sigma(x_w^T \theta_{j-1}^w)]\}$$

CBOW+Negative Sampling

In a nutshell, it doesn't have Huffman tree in the output layer, but a set of negative samples instead (Given $Context(w)$, word w is positive, while others are negative). Negative samples are randomly selected.
Assume that we have had a negative sample set $NEG(w) \neq \Phi$, $\forall \tilde{w} \in D$, we denote the label of w as follows:

$$L^w(\tilde{w}) = \begin{cases} 1, & \tilde{w} = w; \\ 0, & \tilde{w} \neq w; \end{cases}$$

CBOW+Negative Sampling (Cont.)

Given $(Context(w), w)$ our goal is to maximize:

$$g(w) = \prod_{u \in w \bigcup NEG(w)} p(u|Context(w))$$

where

$$p(u|Context(w)) = \begin{cases} \sigma(x_w^T \theta^u), & L^w(u) = 1; \\ 1 - \sigma(x_w^T \theta^u), & L^w(u) = 0; \end{cases}$$

CBOW+Negative Sampling (Cont.)

To increase the probability of positive sample and decrease negative ones:

$$g(w) = \prod_{u \in w \bigcup NEG(w)} p(u|Context(w))$$
$$= \sigma(x_w^T \theta^w) \prod_{u \in NEG(w)} (1 - \sigma(x_w^T \theta^w))$$

Then:

$$G = \prod_{w \in C} = g(w)$$

where C is the corpus.

Learning Objective

Full learning objective: maximize the following:

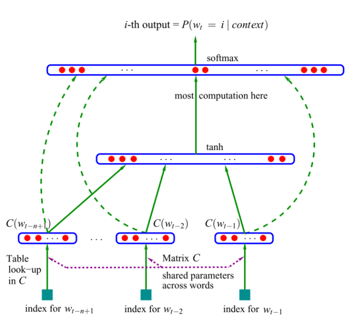
$$\zeta = \log G = log \prod_{w \in C} g(w) = \sum_{w \in C} \log g(w)$$

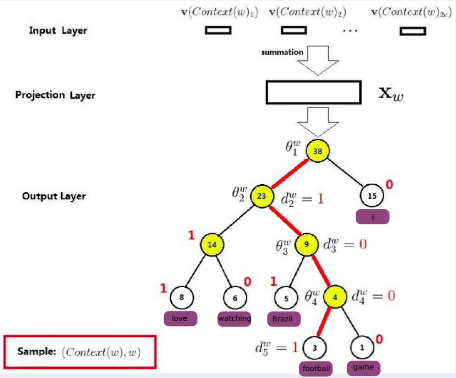
$$= \sum_{w \in C} \log \prod_{u \in \{w\} \bigcup NEG(w)} \{[\sigma(x_w^T \theta^u)]^{L^w(u)}[1 - \sigma(x_w^T \theta^u)]^{(1 - L^w(u))}\}$$

Difference Between CBOW and Skip-gram

- Skip-gram is more accurate.
- Skip-gram is slower given larger context.

Why Use Negative Sampling & Hierachical Softmax





References

Sarikaya, Ruhi, Geoffrey E. Hinton, and Anoop Deoras. "Application of deep belief networks for natural language understanding." Audio, Speech, and Language Processing, IEEE/ACM Transactions on 22.4 (2014): 778-784.

Cho, Kyunghyun, et al. "Learning phrase representations using rnn encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078 (2014).

http://nlp.stanford.edu/courses/NAACL2013/NAACL2013-Socher-Manning-DeepLearning.pdf

http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/

http://blog.csdn.net/itplus/article/details/37969979

http://licstar.net/archives/328

Thank you!