

Deep learning

Representation Learning

Hamid Beigy

Sharif University of Technology

December 11, 2021



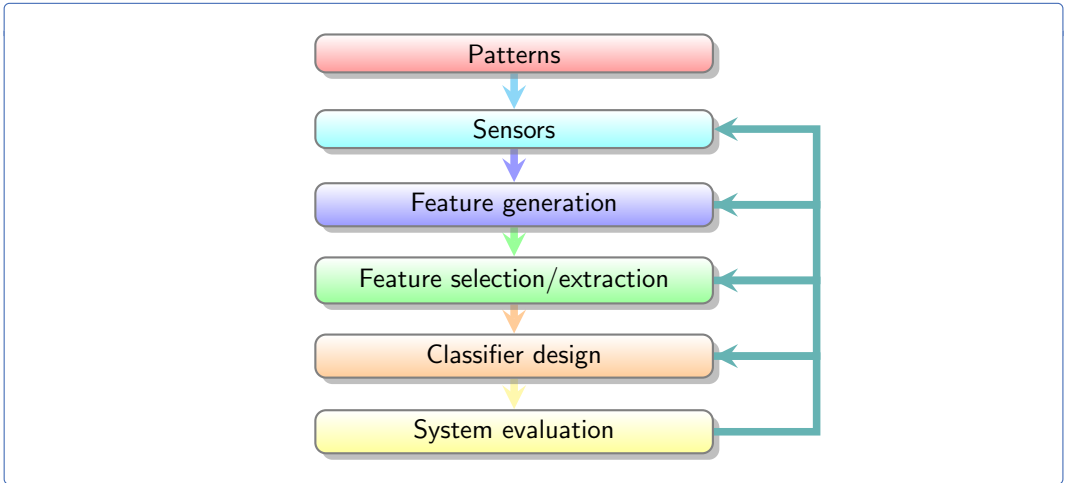


1. Introduction
2. Autoencoders
3. Word embedding
4. Graph embedding
5. Other 2vec embeddings
6. Reading

Introduction



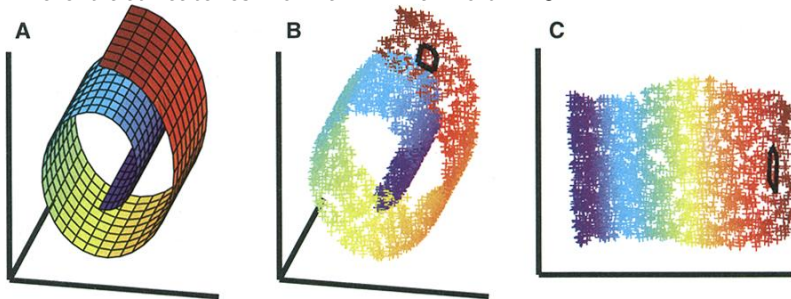
1. The basic stages in design of a classification system.



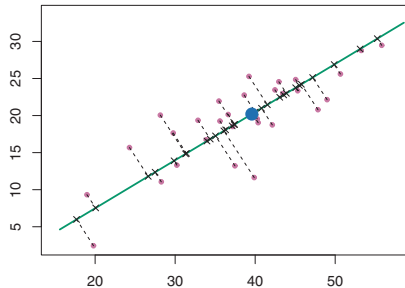


1. Various information processing tasks, such as classification or clustering, can be very easy or very difficult depending on how the information is represented.
2. Good features are essential for successful ML: **90% of effort**.
3. Handcraft features vs feature learning. Which of them results in a good representation?
4. Generally, a representation is said to be good if it makes the subsequent learning tasks easier.
5. The choice of representation depends on what is the subsequent task we want to do.
6. What are good features to represent images, texts, graphs, and etc.?

1. How can we extract features from a 2D manifold in 3D?



2. If data are not on the line, then we have loss in the transformation



3. Can we learn how to find a good representation of data?

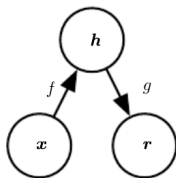


1. Deep learning learns multiple levels of representation of increasing complexity/abstraction.
2. What makes a representation good?
 - ▶ Have distributed representations.
 - ▶ Have multiple levels of representations.
 - ▶ Different explanatory factors of the data tend to change independently of each other in the input distribution.
 - ▶ Have a representation, which is good for different tasks.
 - ▶ Linear relationships in representation space.
 - ▶ King – Queen \approx Man – Woman
 - ▶ Paris – France + Italy \approx Rome
3. Deep networks are appropriate for the above properties.

Autoencoders



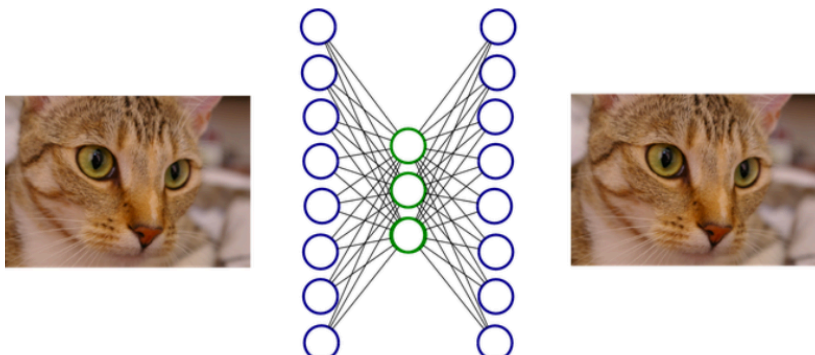
1. An **autoencoder** is a feed-forward neural net whose job it is to take an input x and predict x .
2. In another words, **autoencoders** are neural networks that are trained to copy their inputs to their outputs.
3. It consists of
 - ▶ Encoder $h = f(x)$
 - ▶ Decoder $r = g(h)$



- ▶ Usually constrained in particular ways to make this task more useful.
- ▶ Structure is almost always organized into **encoder network** (f) and **decoder network** (g) and model ($g(f(x))$)
- ▶ Trained by gradient descent with reconstruction loss.
- ▶ This loss measures differences between input and output e.g. MSE :

$$J(\theta) = |g(f(x)) - x|^2$$

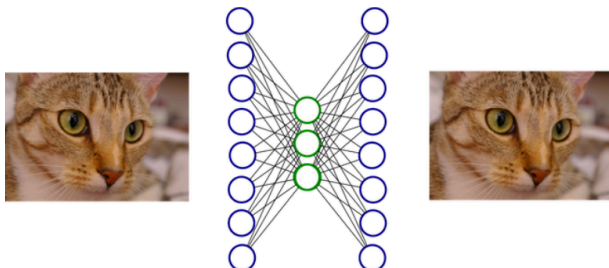
1. Autoencoders consist of an encoder $h = f(x)$ taking an input x to the hidden representation h and a decoder $\hat{x} = g(x)$ mapping the hidden representation h to the input \hat{x} .



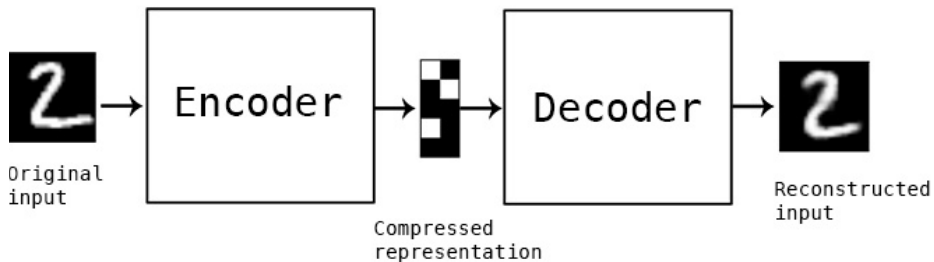
2. The goal is

$$\min_{f,g} \sum \|\hat{x} - x\|^2$$

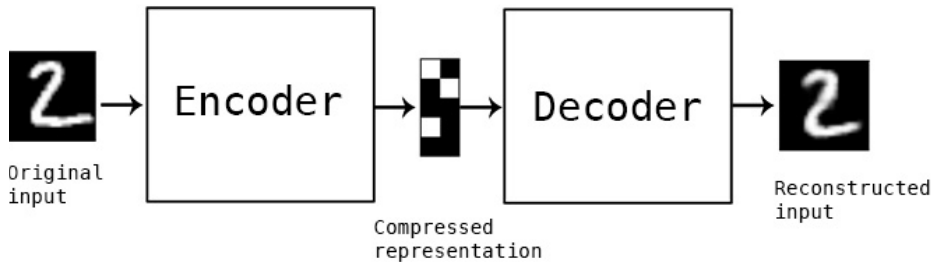
1. An autoencoder is a data compression algorithm.



2. A hidden layer describes the code used to represent the input. It maps input to output through a compressed representation code.

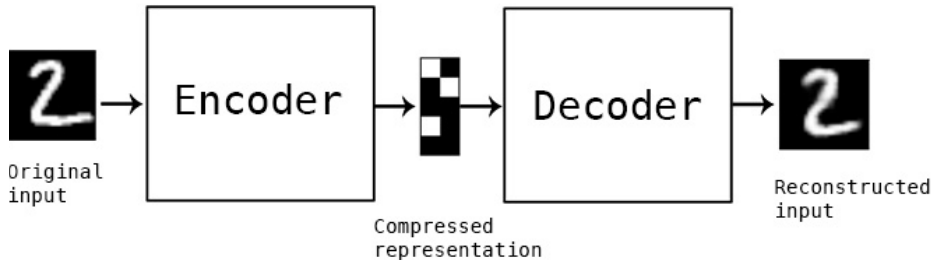


1. PCA can be described as



$$\min_W \sum \|\hat{x} - x\|^2$$
$$W^T W = I$$
$$\min_W \sum \|W^T W x - x\|^2$$

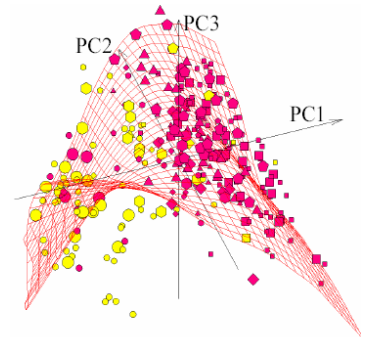
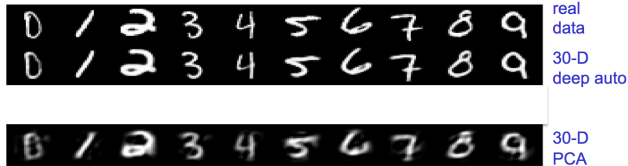
1. Autoencoders can be thought of as a non linear PCA.

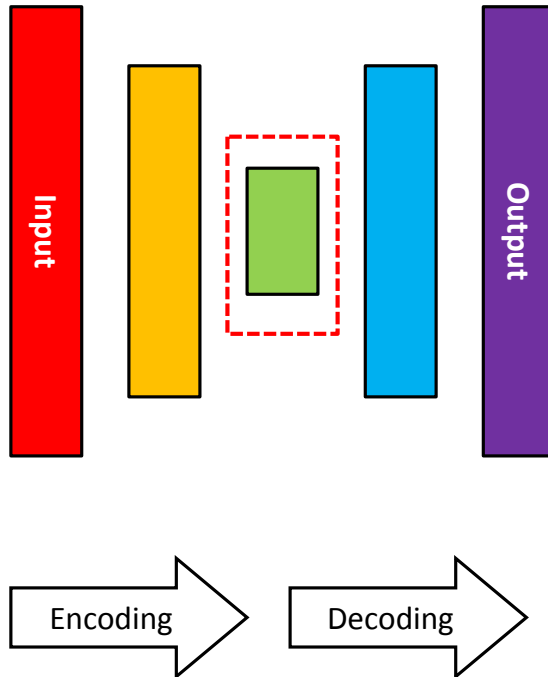


$$\min_{h,g} \sum \|\hat{x} - x\|^2$$

$$\min_{h,g} \sum \|g(f(x)) - x\|^2$$

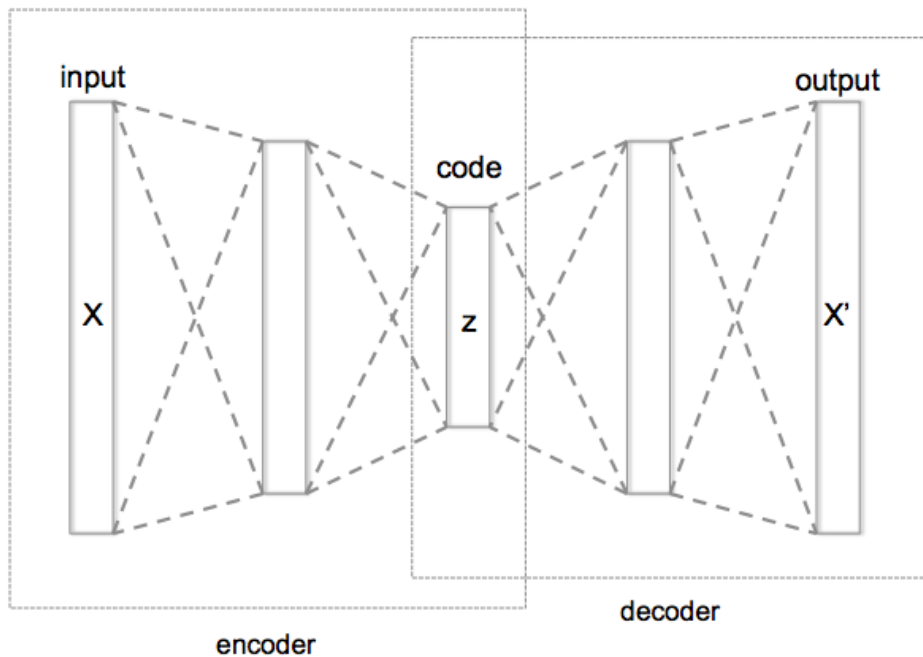
1. Nonlinear autoencoders can learn more powerful codes for a given dimensionality, compared with linear autoencoders (PCA)







1. Encoder + Decoder Structure





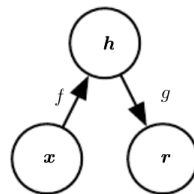
1. Autoencoders are data-specific
 - ▶ They are able to compress data similar to what they have been trained on.
2. This is different from, say, MP3 or JPEG compression algorithm
 - ▶ Which make general assumptions about "sound/images", but not about specific types of sounds/images
 - ▶ Autoencoder for pictures of cats would do poorly in compressing pictures of trees. Because features it would learn would be cat-specific.
3. Autoencoders are lossy
 - ▶ This means that the decompressed outputs will be degraded compared to the original inputs (similar to MP3 or JPEG compression).
 - ▶ This differs from lossless arithmetic compression.



1. Part of neural network landscape for decades.
2. Traditionally used for dimensionality reduction and feature learning.
3. Modern autoencoders also generalized to stochastic mappings

$$p_{\text{encoder}}(h|x) = p_{\text{model}}(h|x)$$

$$p_{\text{decoder}}(x|h) = p_{\text{model}}(x|h)$$



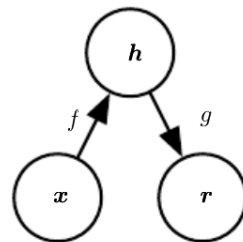
4. These distributions are called stochastic encoders and decoders, respectively.
5. Recent theoretical connection between autoencoders and latent variable models have brought them into forefront of generative models.



1. Consider stochastic decoder $g(h)$ as a **generative model** and its relationship to the joint distribution

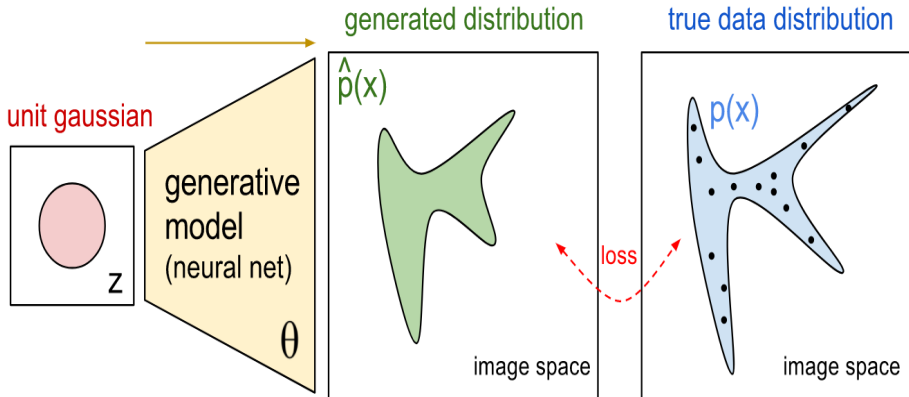
$$p_{model}(x, h) = p_{model}(h) \times p_{model}(x|h)$$

$$\log p_{model}(x, h) = \log p_{model}(h) + \log p_{model}(x|h)$$



2. If h is given from encoding network, then we want most likely x to output.
3. Finding MLE of $x, h \sim$ *maximizing* $p_{model}(x, h)$.
4. $p_{model}(h)$ is prior across latent space values. **This term can be regularizing.**

1. By assuming a prior over latent space, can pick values from underlying probability distribution!





1. Many of the research frontiers in deep learning involve building a probabilistic model of the input, $p_{model}(x)$
2. Many probabilistic models have latent variables, h , with $p_{model}(x) = \mathbb{E}_h [p_{model}(x|h)]$.
3. Latent variables provide another means of [representing the data](#).
4. The more advanced deep models will extend further latent variables: linear factor models.
5. A linear factor model is defined by the use of a stochastic, [linear decoder function](#) that generates x by adding noise to a linear transformation of h .
6. Idea: distributed representations based on latent variables can obtain all of the advantages of learning which we have seen with deep networks



1. Encoder f and decoder g .

$$f : X \mapsto h$$

$$g : h \mapsto X$$

$$\underset{f, g}{\operatorname{argmin}} \| \mathbf{x} - (f \circ g)(\mathbf{x}) \|^2$$

2. One hidden layer

- ▶ Non-linear encoder
- ▶ Takes input $\mathbf{x} \in \mathbb{R}^d$
- ▶ Maps into output $h \in \mathbb{R}^p$

$$h = \sigma_1(W\mathbf{x} + b)$$

$$\hat{\mathbf{x}} = \sigma_2(W'h + b')$$

- ▶ Trained to minimize reconstruction error such as

$$L(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2$$

- ▶ The hidden layer h provides a compressed representation of the input \mathbf{x}



1. An autoencoder is a feed-forward non-recurrent neural network.
 - ▶ With an input layer, an output layer and one or more hidden layers
It can be trained using the following technique
Compute gradients using back-propagation
Followed by mini-batch gradient descent

Autoencoders

Undercomplete Autoencoder



1. An autoencoder whose code dimension is less than the input dimension is called **undercomplete**.
2. Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data.
3. The learning process is described simply as minimizing a loss function

$$L(x, g(f(x)))$$

where L is a loss function penalizing $g(f(x))$ for being dissimilar from x , such as the mean squared error.



1. Assume that the autoencoder has only one hidden layer.
2. What is difference between this network and PCA?
3. When the decoder g is linear and L is the means quared error, an undercomplete autoencoder learns to span the same subspace as PCA.
4. In this case the autoencoder trained to perform the copying task has learned the principal subspace of the training data as a side-effect.
5. If the encoder and decoder functions f and g are nonlinear, a more powerful nonlinear generalization of PCA will be obtained.

Autoencoders

Regularized Autoencoders



1. consider encoder f and decoder g .

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

$$h \in \mathbb{R}^k$$

2. When $k > d$, the autoencoder is called **Overcomplete Autoencoders**.
3. There are other ways we can constraint the reconstruction of an autoencoder than to impose a hidden layer of smaller dimension than the input.
4. Regularized Autoencoders use a loss function that encourages the model to have some properties besides reproducing inputs.
 - ▶ Sparsity representation (Sparse Autoencoders)
 - ▶ Smallness of derivative of representation (Contractive Autoencoders)
 - ▶ Robustness to noise or to missing inputs (Denosing Autoencoders)



1. Sparse Autoencoders try to minimize the following function.

$$L(x, g(f(x))) + \Omega(h)$$

2. The first term is loss for copying inputs
3. The second term is sparsity penalty
4. In general neural network, we are trying to find the **maximum likelihood**: $p(x|\theta)$
5. We often use the log $\log p(x|\theta)$ for simplification, from which we can get the loss function without regularization.
6. What about MAP (Maximum a posterior)?

$$p(\theta|x) \propto p(x|\theta) \times p(\theta)$$

7. Maximizing the log of the above function yields to

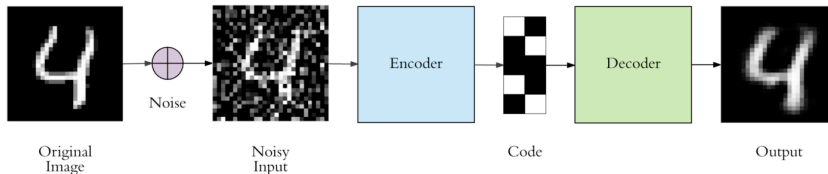
$$\text{maximize } (\log p(x|\theta) + \log p(\theta))$$

8. The first term is loss function and the second term is regularization penalty

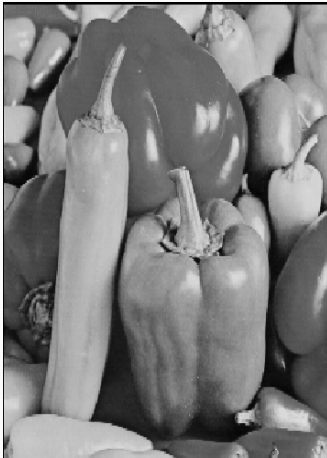
Autoencoders

Denoising Autoencoders

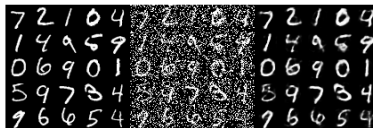
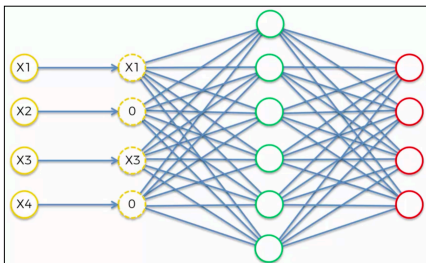
1. The denoising autoencoder (DAE) is an autoencoder that receives a corrupted data point as input and is trained to predict the original, uncorrupted data point as its output.
2. Traditional autoencoders minimize $L(x, g(f(x)))$
 - ▶ L is a loss function penalizing $g(f(x))$ for being dissimilar from x , such as L_2 norm of difference: mean squared error
3. A DAE minimizes $L(x, g(f(\tilde{x})))$
 - ▶ \tilde{x} is a copy of x that is corrupted by some form of noise
 - ▶ The autoencoder must undo this corruption rather than simply copying their input



1. By having to remove noise, model must know difference between noise and actual image.



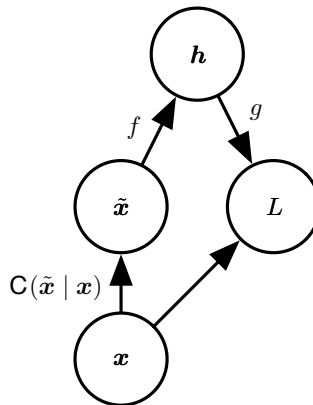
1. An autoencoder with high capacity can end up learning an identity function (also called null function) where input=output
2. A DAE can solve this problem by corrupting the data input
3. How much noise to add?
4. Corrupt input nodes by setting 30 – 50% of random input nodes to zero



Original input, corrupted data, reconstructed data

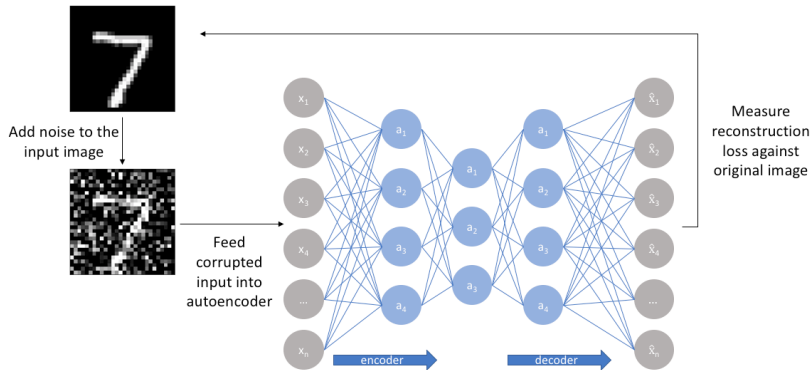


1. The DAE training procedure is



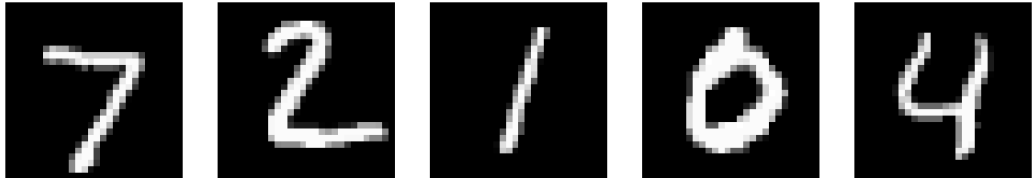
2. We introduce a corruption process $C(\tilde{x}|x)$.

1. We introduce a corruption process $C(\tilde{x}|x)$.
2. The autoencoder then learns a reconstruction distribution $p_{reconstruct}(x|\tilde{x})$ estimated from training pairs (x, \tilde{x}) as follows:
 - ▶ Sample a training example x_i from the training data.
 - ▶ Sample a corrupted version \tilde{x}_i from $C(\tilde{x}_i|x = x_i)$.
 - ▶ Use (x, \tilde{x}) as a training example for estimating the autoencoder reconstruction distribution $p_{reconstruct}(x|\tilde{x}) = p_{decoder}(x|h)$ with h the output of encoder $f(\tilde{x})$ and $p_{decoder}$ typically defined by a decoder $g(h)$.
 - ▶ Typically we can simply perform gradient-based approximate minimization on the negative log-likelihood $-\log p_{decoder}(x|h)$.

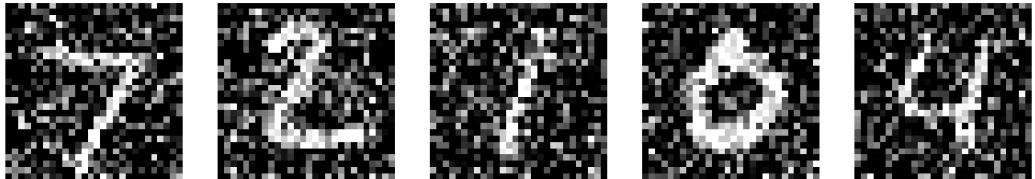




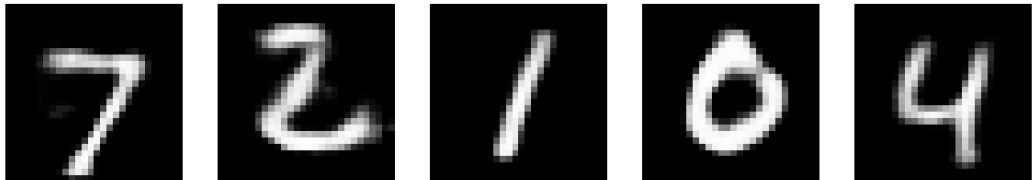
Original Images



Noisy Input



Autoencoder Output

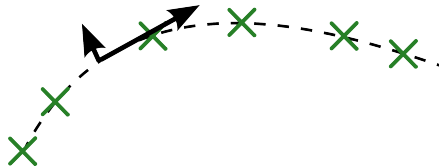


Autoencoders

Contractive Autoencoder

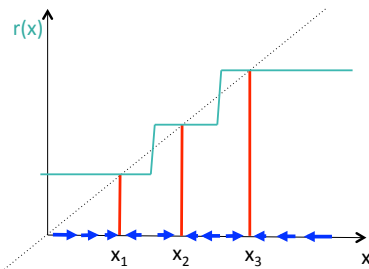


1. Contractive autoencoders are explicitly encouraged to learn a manifold through their loss function (Alain and Bengio 2014).
2. **Desirable property:** Points close to each other in input space maintain that property in the latent space.
3. Method to avoid uninteresting solutions
4. Add an explicit term in the loss that penalizes that solution
5. We wish to extract features that only reflect variations observed in the training set
6. We would like to be invariant to other variations





1. Contractive autoencoder has an explicit regularizer on $h = f(x)$, encouraging the derivatives of f to be as small as possible.
2. This will be true if $f(x) = h$ is continuous, has small derivatives.



3. We can use the **Frobenius Norm** of the **Jacobian Matrix** as a regularization term:

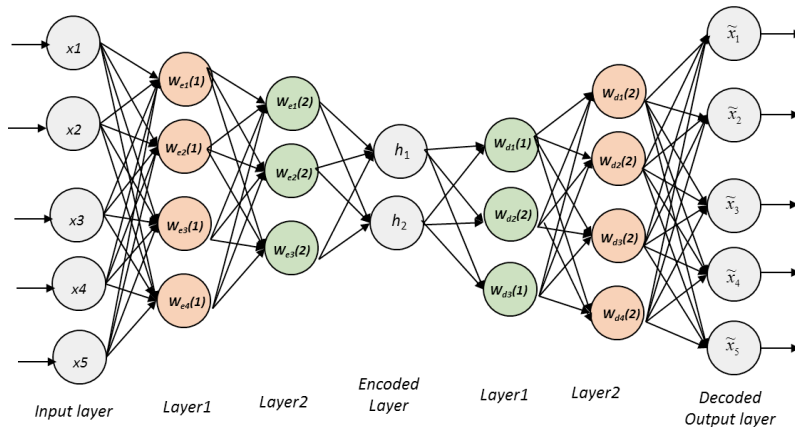
$$\Omega(f, x) = \lambda \left\| \frac{\partial f(x)}{\partial x} \right\|_F^2$$

4. These autoencoders called **contractive** because they contract neighborhood of input space into smaller, localized group in latent space.
5. **Exercise:** What is the difference between DAE and CAE?

Autoencoders

Stacked autoencoder

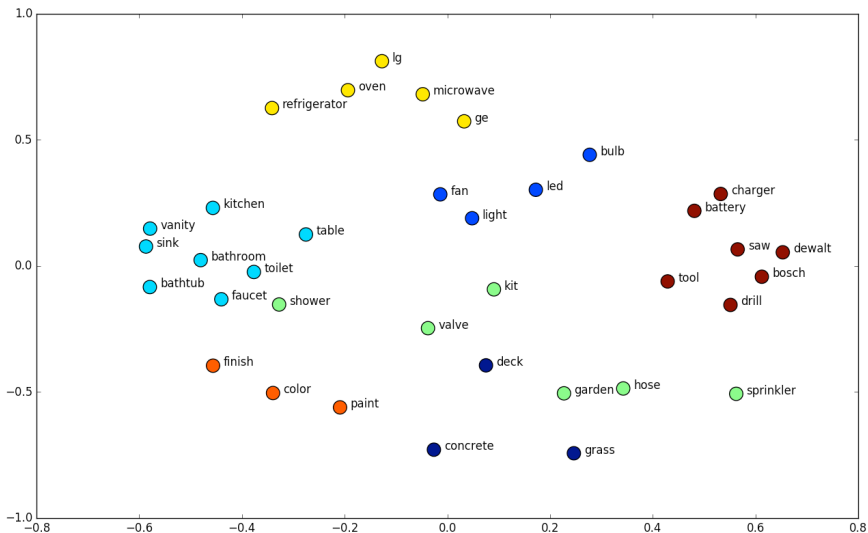
1. A stacked autoencoder is a neural network consist several layers of sparse autoencoders where output of each hidden layer is connected to the input of the successive hidden layer

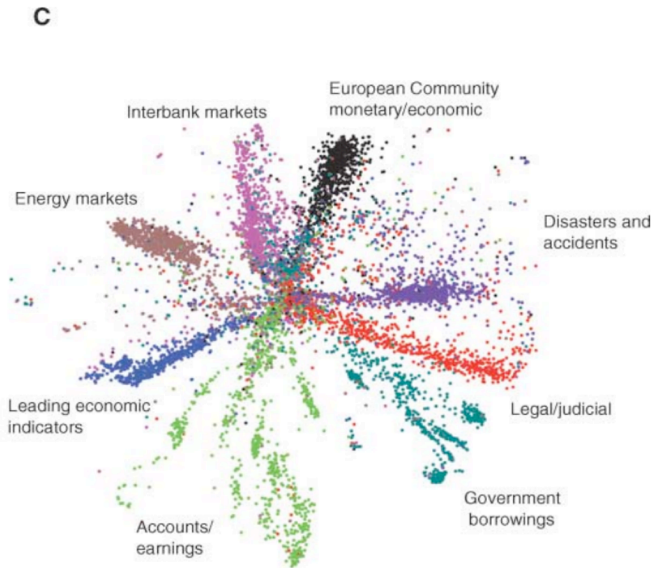
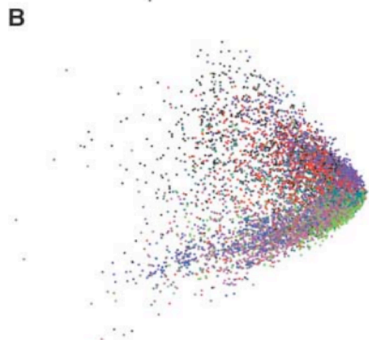
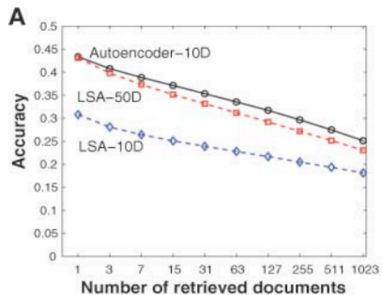


2. Stacked autoencoder improving accuracy in deep learning with noisy autoencoders embedded in the layers (Uma [2018](#)).

Autoencoders

Applications





Word embedding



1. How do you represent a word?

- ▶ Represent words as atomic symbols such as **talk**, **university**, **building**.
- ▶ Represent a word as a **one-hot** vector such as

$$\text{university} = (0, 0, 0, 1, 0, \dots, 0)$$

egg student talk university building ... buy

2. Issues with **one-hot** representation

- ▶ How large is this vector? **dimensionality is large; vector is sparse**
- ▶ Representing **new words** (any idea?).
- ▶ How measure **word similarity**?



1. Linguistic items with similar distributions have similar meanings (**words occur in the same contexts probably have similar meaning**).

$$university = (0.2, 0.1, 0.12, 0.38, 0.2, \dots, 0.12)$$

egg student talk university building buy

2. Word meanings are vector of **basic concept**.
3. What are **basic concepts**?
4. How to assign **weights**?
5. How to define the **similarity/distance**?



1. Distance/similarity

Cosine similarity Word vector are normalized by length

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\|\mathbf{u}\| \|\mathbf{v}\|}$$

Euclidean distance

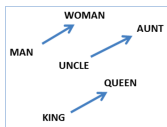
$$d(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|^2$$

Inner product This is same as cosine similarity if vectors are normalized

$$d(\mathbf{u}, \mathbf{v}) = \langle \mathbf{u}, \mathbf{v} \rangle$$

2. Choosing the right similarity metric is important.
3. Word analogy

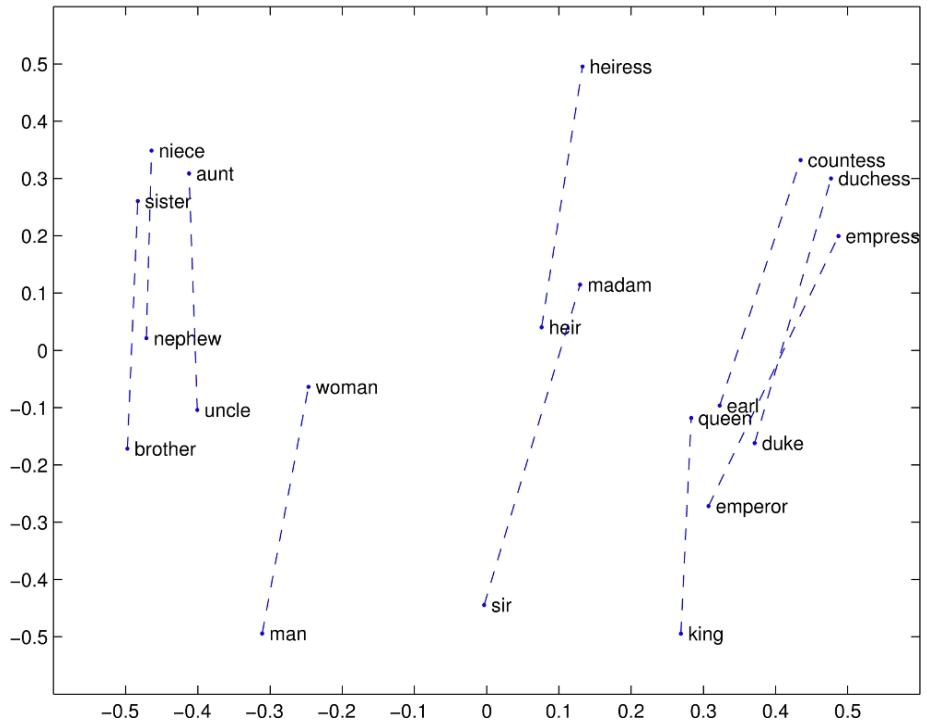
$$v_{man} - v_{woman} + v_{uncle} \approx v_{aunt}$$





1. What are **basic concept**?
 - ▶ We want that the number of basic concepts to be small and
 - ▶ Basis concepts be orthogonal
2. How to assign **weights**?
3. How to define the **similarity/distance** such as cosine similarity?

Distributional representation (example)





Example

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Anthony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Entry is 1 if term occurs. Example: Calpurnia occurs in *Julius Caesar*.

Entry is 0 if term doesn't occur. Example: Calpurnia doesn't occur in *Tempest*.

Each term is represented as a vector of bits.



1. Evaluation of how important a term is with respect to a document.
2. First idea: the more important a term is, the more often it appears: *term frequency*

$$tf_{t,d} = \sum_{x \in d} f_t(x)$$

where

$$f_t(x) = \begin{cases} 1 & \text{if } x = t \\ 0 & \text{otherwise} \end{cases}$$

3. The *order of terms* within a doc is ignored



1. *Inverse document frequency* of a term t :

$$idf_t = \log \frac{N}{df_t} \quad \text{with } N \text{ is the collection size}$$

2. Rare terms have high *idf*, contrary to frequent terms
3. Example (Reuters collection):

Term t	df_t	idf_t
car	18165	1.65
auto	6723	2.08
insurance	19241	1.62
best	25235	1.5

4. In tf-idf weighting, the weight of a term is computed using both *tf* and *idf*:

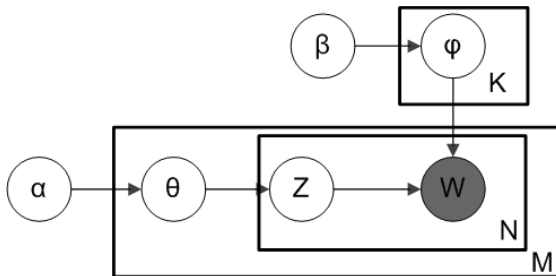
$$w(t, d) = tf_{t,d} \times idf_t \quad \text{called } tf - idf_{t,d}$$



1. we don't need all of the dimensions that represent a word, only the most important ones.
2. There are several techniques such as
 - ▶ **Principle Component Analysis (PCA)**: The most important dimensions contain the most variance
 - ▶ **Latent Semantic Analysis (LSA)**: Project terms and documents into a topic space using SVD on term-document (co-occurrence) matrix.
 - ▶ **Low-rank Approximation**
3. Can we learn the dimensionality reduction from texts?



1. Assumes generative probabilistic model of a corpus (Blei, Ng, and Jordan 2003).
2. Documents are represented as distribution over latent topics, where each topic is characterized by a distribution over words.



Word embedding

Neural probabilistic language model



1. An **language model** is a model for how humans **generate language**.
2. The quality of language models is measured based on their ability to learn a probability distribution over words in vocabulary V .
3. Language models generally try to compute the probability of a word w_t given its $n - 1$ previous words, i.e. $p(w_t | w_{t-1}, \dots, w_{t-n+1})$.
4. Applying the chain rule and Markov assumption, we can approximate the probability of a whole sentence or document by the product of the probabilities of each word given its n previous words:

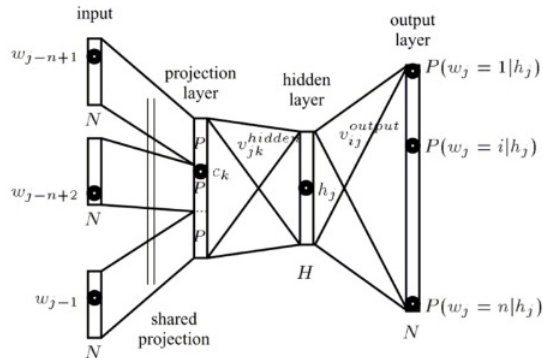
$$p(w_1, \dots, w_T) = \prod_i p(w_i | w_{i-1}, \dots, w_{i-n+1})$$

5. In n-gram based language models, we can calculate a word's probability based on the frequencies of its constituent n-grams:

$$p(w_t | w_{t-1}, \dots, w_{t-n+1}) = \frac{\text{count}(w_{t-n+1}, \dots, w_{t-1}, w_t)}{\text{count}(w_{t-n+1}, \dots, w_{t-1})}$$



1. In NNs, we achieve the same objective using the softmax layer (Bengio et al. 2003).

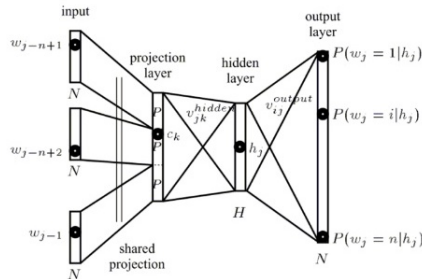


$$p(w_t | w_{t-1}, \dots, w_{t-n+1}) = \frac{\exp(h^\top v'_{w_t})}{\sum_{w_i \in V} \exp(h^\top v'_{w_i})}$$

2. Words are mapped into a $|V| \times m$ matrix \mathbf{C} , where $|V|$ is the size of vocabulary and m is dimension of embedding.



1. In this model, a sequence of words is given to the input of the network and the output is the $p(w|h)$, where h is hidden layer.

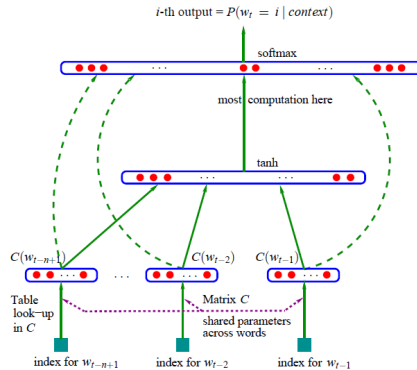


2. The inner product $h^T v'_{w_t}$ computes the (unnormalized) log-probability of word w_t , which we normalize by the sum of the log-probabilities of all words in V .
3. h is the output vector of the penultimate network layer, while v'_w is the output embedding of word w , i.e. its representation in the weight matrix of the softmax layer.
4. The network is trained by maximizing the following objective function

$$J_{\theta} = \frac{1}{T} \sum_{t=1}^T \log p(w_t, w_{t-1}, \dots, w_{t-n+1})$$



1. In NNs, we achieve the same objective using the softmax layer (Bengio et al. 2003).



2. A mapping \mathbf{C} from any element i of V to a real vector $\mathbf{C}(i) \in \mathbb{R}^m$. It represents the **distributed feature vectors** associated with each word in the vocabulary.
3. Learn both the embedding and parameters for probability function jointly.
4. The dotted green line causes the training time to be cut in half (10 epochs to converge instead of 20).

Word embedding

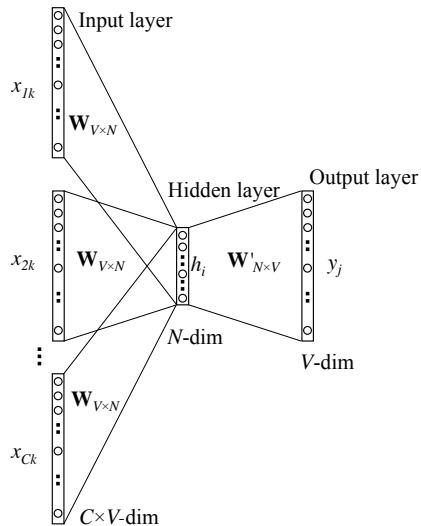
Word2vec algorithm



1. Proposed by Mikolov et. al. and widely used for many NLP applications (Mikolov, Chen, et al. [2013](#); Mikolov, Sutskever, Chen, Gregory S. Corrado, et al. [2013](#)).
2. Key features
 - ▶ Uses neural networks to train word / context classifiers (feed-forward neural net)
 - ▶ Uses local context windows (environment around any word in a corpus) as inputs to the NN
 - ▶ Removed hidden layer.
 - ▶ Use of additional context for training LM's.
 - ▶ Introduced newer training strategies using huge database of words efficiently.
3. In (Mikolov, Chen, et al. [2013](#)), they proposed two architectures for learning word embeddings that are computationally less expensive than previous models.
4. In (Mikolov, Sutskever, Chen, Gregory S. Corrado, et al. [2013](#)), they improved upon these models by employing additional strategies to enhance training speed and accuracy.



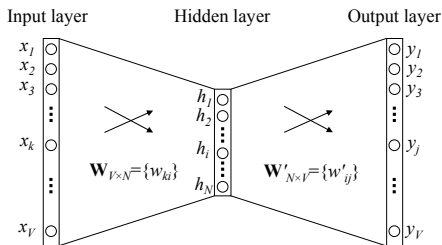
1. Mikolov et al. thus used both the n words before and after the target word w_t to predict it.
2. They called this continuous bag-of-words (CBOW), as it uses continuous representations whose order is of no importance.
3. The objective function of CBOW in turn is



$$l(\theta) = \sum_{t \in \text{Text}} \log P(w_t | w_{t-n}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+n})$$



1. Let vocabulary size be V and hidden layer size be N (Rong 2016).
2. The input is a one-hot encoded vector $\mathbf{x} = (x_1, \dots, x_V)$.



3. Weights between input layer and hidden layer is represented by $V \times N$ matrix \mathbf{W} .
4. Each row of \mathbf{W} is the N -dimension vector representation \mathbf{v}_{w_i} of the input word.
5. Let $x_k = 1$ and $x_j = 0$ for $j \neq k$, then

$$\mathbf{h} = \mathbf{W}^T \mathbf{x} = \mathbf{v}_{w_i}^T$$

6. This implies that activation function of the hidden layer units is simply linear.
7. Weights between hidden layer and output layer is represented by $N \times V$ matrix \mathbf{W}' .



1. Using these weights, we can compute a score \mathbf{u}_j for each word in the vocabulary.

$$\mathbf{u}_j = \mathbf{v}'_{w_j}{}^T \mathbf{h}$$

where \mathbf{v}'_{w_j} is the j -th column of the matrix \mathbf{W}'

2. Then, softmax is used to obtain the posterior distribution of words.

$$p(w_j | w_I) = y_j = \frac{\exp(\mathbf{u}_j)}{\sum_{k=1}^V \exp(\mathbf{u}_k)}$$

where y_j is the output of the j -th unit in the output layer.

3. By replacing the above two equations, we obtain

$$p(w_j | w_I) = \frac{\exp(\mathbf{v}'_{w_j}{}^T \mathbf{v}_{w_I})}{\sum_{k=1}^V \exp(\mathbf{v}'_{w_k}{}^T \mathbf{v}_{w_I})}$$

4. Note that \mathbf{v}_w and \mathbf{v}'_w are two representations of the word w .
5. They are called **input vector**, and **output vector** of the word w .



1. The training objective (for one training sample) is to maximize the following function

$$p(w_j|w_l) = \frac{\exp(\mathbf{v}'_{w_j} \mathbf{v}_{w_l})}{\sum_{k=1}^V \exp(\mathbf{v}'_{w_k} \mathbf{v}_{w_l})}$$

given the input context word w_l with regard to the weights.

$$\begin{aligned} \max p(w_o|w_l) &= \max y_{j^*} \\ &= \max \log y_{j^*} \\ &= u_{j^*} - \log \sum_{k=1}^V \exp(u_k) = -E \end{aligned}$$

2. $E = -\log p(w_o|w_l)$ is the loss function and j^* is the index of the actual output word in the output layer.



1. To derive the update equation of the weights between hidden and output layers, take the derivative of E with regard to j -th unit's net input u_j , we obtain

$$\frac{\partial E}{\partial u_j} = y_j - t_j = e_j$$

where $t_j = \mathbb{I}[j = j^*]$.

2. Next we take the derivative on w'_{ij} to obtain the gradient on weight w'_{ij} .

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i$$

3. By using stochastic gradient descent, we obtain

$$w'_{ij}(t+1) = w'_{ij}(t) - \eta e_j \cdot h_i$$



1. To derive the update equation of the weights between input and hidden layers, take derivative of E with respect to h_i , we obtain

$$\frac{\partial E}{\partial h_i} = \sum_{k=1}^V \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{k=1}^V e_j w'_{ij} = EH_i$$

where

$$h_i = \sum_{k=1}^V x_k w_{ki}$$

2. Now we can take the derivative of E with regard to each element of \mathbf{W} , obtaining

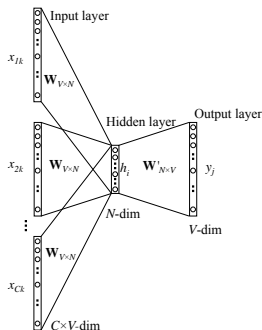
$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = EH_i x_k.$$

3. By using stochastic gradient descent, we obtain

$$w_{ki}(t+1) = w_{ki}(t) - \beta EH_i x_k$$



1. Consider CBOW model with a multi-word context setting.



2. When computing the hidden layer output, the CBOW model takes the average of vectors of input context words

$$\begin{aligned} \mathbf{h} &= \frac{1}{C} \mathbf{W}^T (\mathbf{x}_1 + \mathbf{x}_2 + \dots + \mathbf{x}_C) \\ &= \frac{1}{C} (\mathbf{v}_{w_1} + \mathbf{v}_{w_2} + \dots + \mathbf{v}_{w_C}) \end{aligned}$$

where C is the number of words in the context.



1. The loss function is

$$\begin{aligned} E &= -\log p(w_o | w_{l,1}, \dots, w_{l,c}) \\ &= -u_{j_*} + \log \sum_{k=1}^V \exp(u_k) \\ &= -\mathbf{v}_{w_o}^\top \mathbf{h} + \log \sum_{k=1}^V \exp(\mathbf{v}_{w_k} \mathbf{h}) \end{aligned}$$

which is the same as the objective of the one-word-context model, except that \mathbf{h} is different.

2. The update equation for the hidden-output weights stay the same as that for the one-word-context model

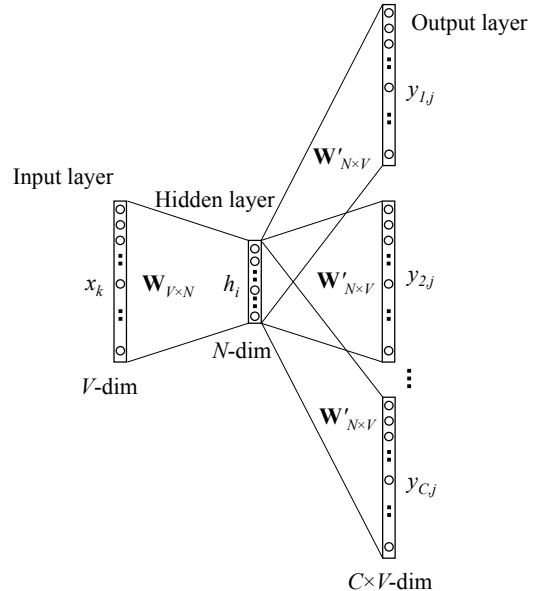
$$\mathbf{v}'_{w_j}(t+1) = \mathbf{v}'_{w_j}(t) - \eta e_j \cdot \mathbf{h} \quad \text{for } j = 1, 2, \dots, V$$

3. The update equation for input-hidden weights is

$$v_{w_l,c}(t+1) = v_{w_l,c}(t) - \frac{1}{C} \eta E \mathbf{H}^\top \quad \text{for } c = 1, 2, \dots, C$$

1. Instead of using the surrounding words to predict the center word as with CBOW, skip-gram uses the center word to predict the surrounding words.
2. The skip-gram objective thus sums the log probabilities of the surrounding n words to the left and to the right of the target word w_t to produce the following objective function.

$$l(\theta) = \sum_{t \in \text{Text}} \sum_{-n \leq j \leq n, j \neq 0} \log P(w_{t+j} | w_t)$$





1. Let vocabulary size be V and hidden layer size be N .
2. The input is a one-hot encoded vector $\mathbf{x} = (x_1, \dots, x_V)$.
3. Weights between input layer and hidden layer is represented by $V \times N$ matrix \mathbf{W} .
4. Each row of \mathbf{W} is the N -dimension vector representation \mathbf{v}_{w_I} of the input word.

$$\mathbf{h} = \mathbf{W}^T \mathbf{x} = \mathbf{v}_{w_I}^T$$

5. On the output layer, instead of outputting one multinomial distribution, C -multinomial distributions are output.
6. Each output is computed using the same hidden-output weight matrix

$$p(w_{c,j} = w_{o,c} | w_I) = y_{c,j} = \frac{\exp(\mathbf{u}_{c,j})}{\sum_{k=1}^V \exp(\mathbf{u}_k)}$$

where $w_{c,j}$ is the j -th word on the c -th panel of output layer and $w_{o,c}$ is the actual c -th word in the output context words.



1. The output layer panels share the same weights, thus

$$u_{c,j} = u_j = \mathbf{v}'_{w_j} \cdot \mathbf{h} \quad \text{for } c = 1, \dots, C.$$

where \mathbf{v}'_{w_j} is the output vector of the j -th word in the vocabulary,

2. The derivation of parameter update equations is

$$\begin{aligned} E &= -\log p(w_{o,1}, w_{o,2}, \dots, w_{o,C} | w_l) \\ &= -\log \prod_{c=1}^C \frac{\exp(u_{c,j_c^*})}{\sum_{k=1}^V \exp(u_{c,k_c})} \\ &= -\sum_{c=1}^C u_{c,j_c^*} + C \cdot \log \sum_{k=1}^V \exp(u_{c,k_c}) \end{aligned}$$

where j_c^* is the index of the actual c -th output context word in the vocabulary.



1. To derive update equation of weights between hidden and output layers, take derivative of E with respect to every unit on every panel of output layer, $u_{c,j}$

$$\frac{\partial E}{\partial u_{c,j}} = y_{c,j} - t_{c,j} = e_{c,j}$$

where $t_{c,j} = \mathbb{I}[j = j^*]$ and $EI_j = \sum_{c=1}^C e_{c,j}$.

2. Then, take the derivative of E with respect to matrix \mathbf{W}' ,

$$\frac{\partial E}{\partial w'_{ij}} = \sum_{c=1}^C \frac{\partial E}{\partial u_{c,j}} \cdot \frac{\partial u_{c,j}}{\partial w'_{ij}} = EI_j \cdot h_i$$

3. By using stochastic gradient descent, we obtain

$$w'_{ij}(t+1) = w'_{ij}(t) - \eta EI_j \cdot h_i$$

4. For input weight matrix, we have

$$\mathbf{v}_{w_i}(t+1) = \mathbf{v}_{w_i}(t) - \eta E\mathbf{H}^T$$

where $E\mathbf{H}$ is an N -dimensional vector, each component of which is defined as

$$EH_i = \sum_{j=1}^V EH_j w'_{ij}$$



1. For CBOW and skip-gram, we have two representations of the word w denoted by \mathbf{v}_w and \mathbf{v}'_w .
2. They are called **input vector**, and **output vector** of the word w .
3. Learning the **input vectors** is cheap; but learning the **output vectors** is very expensive, because using the following weight update equation,

$$w'_{ij}(t+1) = w'_{ij}(t) - \eta e_j \cdot h_i$$

for each training example, we iterate through every word w_j in vocabulary, compute their

- ▶ net input u_j ,
 - ▶ probability prediction y_j (or $y_{c,j}$ for skip-gram),
 - ▶ their prediction error e_j (or E_j for skip-gram), and
 - ▶ use their prediction error to update their output vector \mathbf{v}_j .
4. Such computations for all words for every training example is very expensive, and is not scalable.

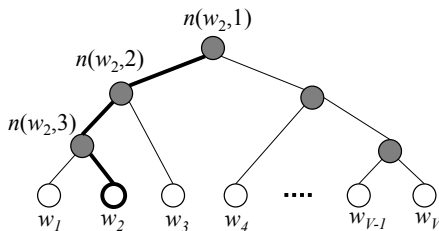


1. To solve the mentioned problem, we can
 - ▶ limit the number of output vectors that must be updated per training instance,
 - ▶ use hierarchical softmax,
 - ▶ use negative sampling.
2. These tricks optimize only the computation of the updates for output vectors.
3. We must compute the following values

E new objective function
 $\frac{\partial}{\partial \mathbf{v}_w}$ new update equation for the output vectors
 $\frac{\partial}{\partial \mathbf{h}}$ weighted sum of errors to be backpropagated for updating input vectors



1. Hierarchical softmax is an efficient way of computing softmax (Morin and Bengio 2005).
2. Hierarchical softmax uses a binary tree to represent all words in the vocabulary.
3. Each leaf of the tree is a word, and there is a unique path from root to leaf.
4. This path is used to estimate the probability of the word represented by leaf.
5. In hierarchical softmax, there is no output representation for words.
6. Each node of the graph (except root and leaves) is associated to a vector that model is going to learn.



7. $n(w, j)$ means the j -th unit on the path from root to the word w .
8. $L(w)$ is the length of path from the root to the leaf representing node w (including the number of nodes in the path including the root and the leaf)



1. In hierarchical softmax, probability of a word being the output word is defined as

$$p(w = w_o) = \prod_{j=1}^{L(w)-1} \sigma \left(\mathbb{I}[n(w, j+1) = ch(n(w, j))] \cdot \mathbf{v}'_{n(w, j)}{}^\top \mathbf{h} \right)$$

where

- ▶ $ch(n)$ is the left child of unit n
- ▶ $\mathbf{v}'_{n(w, j)}$ is the vector representation (**output vector**) of the inner unit $n(w, j)$,
- ▶ \mathbf{h} is the output value of the hidden layer,
- ▶ $\mathbb{I}[x]$ is a special function defined as

$$\mathbb{I}[x] = \begin{cases} +1 & \text{if } x \text{ is true} \\ -1 & \text{if } x \text{ is false} \end{cases}$$

2. The probability of going left at an inner unit n equals

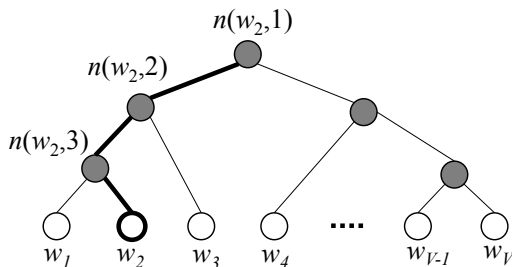
$$p(n, \text{left}) = \sigma \left(\mathbf{v}'_n{}^\top \cdot \mathbf{h} \right)$$

3. The probability of going right at an inner unit n equals

$$p(n, \text{right}) = 1 - \sigma \left(\mathbf{v}'_n{}^\top \cdot \mathbf{h} \right) = \sigma \left(-\mathbf{v}'_n{}^\top \cdot \mathbf{h} \right)$$



1. Considering the following tree



2. We have

$$\begin{aligned}
 p(w_2 = w_o) &= p(n(w_2, 1), \text{left}) \cdot p(n(w_2, 2), \text{left}) \cdot p(n(w_2, 3), \text{right}) \\
 &= \sigma \left(\mathbf{v}'_{n(w_2, 1)} \cdot \mathbf{h} \right) \sigma \left(\mathbf{v}'_{n(w_2, 2)} \cdot \mathbf{h} \right) \sigma \left(-\mathbf{v}'_{n(w_2, 3)} \cdot \mathbf{h} \right)
 \end{aligned}$$



1. For updating rule of \mathbf{v}' of inner units, consider one-word context model and define

$$\mathbb{I}[\cdot] = \mathbb{I}[n(w, j + 1) = ch(n(w, j))]$$

$$\mathbf{v}'_{n_j} = \mathbf{v}'_{n(w, j)}$$

2. For a training instance, the error function is defined as

$$E = -\log p(w = w_o | w_l) = - \sum_{j=1}^{L(w)-1} \log \sigma \left(\mathbb{I}[\cdot] \mathbf{v}'_j{}^\top \cdot \mathbf{h} \right)$$

3. Taking the derivative of E with regard to $\mathbf{v}'_j{}^\top \cdot \mathbf{h}$, obtaining

$$\frac{\partial E}{\partial \mathbf{v}'_j{}^\top \cdot \mathbf{h}} = \sigma \left(\mathbf{v}'_j{}^\top \cdot \mathbf{h} \right) - t_j$$

where $t_j = 1$ if $\mathbb{I}[\cdot] = 1$ and $t_j = 0$ otherwise.

4. Taking derivative of E with respect to vector representation of inner node $n(w, j)$,

$$\frac{\partial E}{\partial \mathbf{v}'_j} = \left(\sigma \left(\mathbf{v}'_j{}^\top \cdot \mathbf{h} - t_j \right) \right) \cdot \mathbf{h}$$

5. Weight update rule becomes

$$\mathbf{v}'_j(t+1) = \mathbf{v}'_j(t) - \eta \left(\sigma \left(\mathbf{v}'_j{}^\top \cdot \mathbf{h} - t_j \right) \right) \cdot \mathbf{h}$$



1. This update equation can be used for both CBOW and the skip-gram model.
2. When used for the skip-gram model, we must repeat this update for all C words in the output context.
3. To backpropagate the error to learn input-hidden weights, we have

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{h}} &= \sum_{j=1}^{L(w)-1} \frac{\partial E}{\partial \mathbf{v}_j'^{\top} \cdot \mathbf{h}} \cdot \frac{\partial \mathbf{v}_j'^{\top} \cdot \mathbf{h}}{\partial \mathbf{h}} \\ &= \sum_{j=1}^{L(w)-1} \left(\sigma \left(\mathbf{v}_j'^{\top} \cdot \mathbf{h} \right) - t_j \right) \\ &= EH\end{aligned}$$

4. This can be directly substituted in update rule of CBOW.
5. For updating rule, the computational complexity per training instance per context word is reduced from $O(V)$ to $O(\log(V))$.



1. The idea of negative sampling is more straightforward than hierarchical softmax.
2. To deal with difficulty of updating too many output vectors per iteration, only update a sample of them.
3. Let the pair of words (apple, juice), the following datasets are built
 D_+ Set {(apple, juice)}. This set contains all pairs (w_l, w_o) that are in the text.
 D_- Set {(apple, word 1), ...}. This set contains all pairs (w_l, w_o) that are not in the text.
4. We can assign a label $z = 1$ to all pairs $(w, w_o) \in D_+$ a label $z = 0$ to all pairs $(w_l, w_o) \in D_-$. Hence,

$$z = \begin{cases} 1 & \text{if } (w_l, w_o) \in D_+ \\ 0 & \text{if } (w_l, w_o) \in D_- \end{cases}$$

5. Hence, the problem becomes as a classification by predicting the label z .



- Hence, the problem becomes as a classification by predicting the label z .
 - ▶ $p(z = 1|(w_I, w_o))$ is the probability that $(w_I, w_o) \in D_+$.
 - ▶ $p(z = 0|(w_I, w_o))$ is the probability that $(w_I, w_o) \in D_-$.
- There are several approaches to solve this, one is to use [the logistic regression algorithm](#).
- In logistic regression , we have

$$\begin{aligned}
 p(z = 1|(w_I, w_o)) &= \frac{1}{1 + \exp(\mathbf{v}_{w_I}^\top \mathbf{v}'_{w_o})} \\
 p(z = 0|(w_I, w_o)) &= \frac{\exp(\mathbf{v}_{w_I}^\top \mathbf{v}'_{w_o})}{1 + \exp(\mathbf{v}_{w_I}^\top \mathbf{v}'_{w_o})} \\
 &= \frac{\exp(\mathbf{v}_{w_I}^\top \mathbf{v}'_{w_o})}{1 + \exp(\mathbf{v}_{w_I}^\top \mathbf{v}'_{w_o})} \times \frac{\exp(-\mathbf{v}_{w_I}^\top \mathbf{v}'_{w_o})}{\exp(-\mathbf{v}_{w_I}^\top \mathbf{v}'_{w_o})} \\
 &= \frac{1}{1 + \exp(-\mathbf{v}_{w_I}^\top \mathbf{v}'_{w_o})}.
 \end{aligned}$$

- Let $\theta = \{\mathbf{v}_{w_I}, \mathbf{v}'_{w_o}\}$ be the parameters to be learned.



1. Hence, the logistic regression becomes

$$p(z|(w_l, w_o)) = \left(\frac{1}{1 + \exp(\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o})} \right)^z \left(\frac{1}{1 + \exp(-\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o})} \right)^{1-z}$$

2. We use the **maximum likelihood** as the objective function. Hence, the likelihood is

$$L(\theta) = \prod_{(w_l, w_o) \in (D_+ \cup D_-)} \left(\frac{1}{1 + \exp(\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o})} \right)^z \left(\frac{1}{1 + \exp(-\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o})} \right)^{1-z}.$$

3. and the **log likelihood** as

$$\ell(\theta) = \sum_{(w_l, w_o) \in (D_+ \cup D_-)} \log \left(\frac{1}{1 + \exp(\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o})} \right)^z + \log \left(\frac{1}{1 + \exp(-\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o})} \right)^{1-z}$$



1. The **log likelihood** as

$$\ell(\theta) = \sum_{(w_l, w_o) \in (D_+ \cup D_-)} \log \left(\frac{1}{1 + \exp(\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o})} \right)^z + \log \left(\frac{1}{1 + \exp(-\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o})} \right)^{1-z}$$

2. By separating two cases for $z = 1$ and $z = 0$, we have

$$\begin{aligned} \ell(\theta) &= \sum_{(w_l, w_o) \in D_+} \log \left(\frac{1}{1 + \exp(\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o})} \right) + \sum_{(w_l, w_o) \in D_-} \log \left(\frac{1}{1 + \exp(-\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o})} \right) \\ &= \sum_{(w_l, w_o) \in D_+} \log \left(\sigma(\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o}) \right) + \sum_{(w_l, w_o) \in D_-} \log \left(\sigma(-\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o}) \right) \end{aligned}$$

3. If we use only one pair (w_l, w_o) instead of all pairs $(w_l, w_o) \in (D_+ \cup D_-)$, we have

$$\ell(\theta) = \log \left(\sigma(\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o}) \right) + \log \left(\sigma(-\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o}) \right)$$



1. The dataset $D_+ \cup D_-$ is highly unbalanced, hence K pairs $(w_l, w_o) \in D_-$ are used.

$$\ell(\theta) = \log \left(\sigma(\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_o}) \right) + \sum_{k=1}^K \log \left(\sigma(-\mathbf{v}_{w_l}^\top \mathbf{v}'_{w_{o_k}}) \right)$$

2. We sample K words **word 1**, ..., **word K** using uni-gram language model distribution $P_n(w)$.
3. The objective function becomes,

$$E = -\ell(\theta) = -\log \sigma \left(\mathbf{v}'_{w_o}^\top \mathbf{v}'_{w_o} \right) - \sum_{w_j \in D_-} \mathbb{E}_{w'_o \sim P_n(w)} \left[\log \sigma \left(-\mathbf{v}'_{w_j}^\top \mathbf{v}'_{w'_o} \right) \right]$$

where

- ▶ where w_o is the output word (i.e., the positive sample),
- ▶ and \mathbf{v}'_{w_o} is its output vector.



1. To obtain update equations, first take derivative of E with respect to net input of output unit w_j .

$$\frac{\partial E}{\partial \mathbf{v}'_{w_j}{}^\top \cdot \mathbf{h}} = \sigma \left(\mathbf{v}'_{w_j}{}^\top \cdot \mathbf{h} \right) - t_j$$

where t_j is label of word w_j . $t_j = 1$ if $w_j \in D_+$ and $t_j = 0$ if $w_j \in D_-$.

2. Taking derivative of E with respect to output vector of the word w_j ,

$$\frac{\partial E}{\partial \mathbf{v}'_{w_j}} = \left(\sigma \left(\mathbf{v}'_{w_j}{}^\top \cdot \mathbf{h} - t_j \right) \right) \cdot \mathbf{h}$$

3. This results in the following update equation for its output vector:

$$\mathbf{v}'_{w_j}(t+1) = \mathbf{v}'_{w_j}(t) - \eta \left(\sigma \left(\mathbf{v}'_{w_j}{}^\top \cdot \mathbf{h} - t_j \right) \right) \cdot \mathbf{h}$$

4. To backpropagate the error to hidden layer

$$\frac{\partial E}{\partial \mathbf{h}} = \sum_{w_j \in (D_+ \cup D_-)} \left[\sigma \left(\mathbf{v}'_{w_j}{}^\top \cdot \mathbf{h} \right) - t_j \right] \cdot \mathbf{v}'_{w_j} = EH$$

5. This can be plugged into update equation of CBOW.
6. For skip-gram, we must calculate EH for each word in the skip-gram context and then plug into update equation.

Word embedding

Global vectors for word representation



1. Skip-gram doesn't utilize the statistics of corpus since they train on separate local context windows instead of on global co-occurrence counts.
2. The statistics of word occurrences in a corpus is the primary source of information available to all unsupervised methods for learning word representations.
3. Glove model aims to combine the count-based matrix factorization and the context-based skip-gram model together (Pennington, Socher, and Manning 2014).
4. Let X_{ij} be the number of times word j occurs in the context of word i .
5. Let $X_i = \sum_k X_{ik}$ be the number of times any word appears in context of word i .
6. Let $P_{ij} = P(j|i) = \frac{X_{ij}}{X_i}$ be the probability that word j appear in context of word i .



1. Co-occurrence probabilities for target words **ice** and **steam** with selected context words from a 6 billion token corpus.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k ice)/P(k steam)$	8.9	8.5×10^{-2}	1.36	0.96

2. Considering two words $i = ice$ and $j = steam$ and study their relationship using various probe words, k .
3. For words k related to **ice** but **not steam**, say $k = solid$, we expect the ratio $\frac{P_{ik}}{P_{jk}}$ will be large.
4. For words k related to **steam** but **not ice**, say $k = gas$, we expect the ratio $\frac{P_{ik}}{P_{jk}}$ will be small.
5. For words k like **water** or **fashion**, that are either related to both **ice** and **steam**, or **to neither**, the **ratio should be close to one**.



1. This argument suggests that the appropriate starting point for word vector learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves.
2. Noting that the ratio $\frac{P_{ik}}{P_{jk}}$ depends on three words i, j, k , the most general model takes the form of

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

where $w \in \mathbb{R}^d$ are word vectors and $\tilde{w} \in \mathbb{R}^d$ is separate context word vector.

3. We would like F to encode $\frac{P_{ik}}{P_{jk}}$ in the word vector space.
4. Since vector spaces are inherently linear structures, the most natural way to do this is with vector differences. Hence

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$



1. Parameters of F are vectors while the right-hand side is a scalar.
2. F can be a complicated function such as a neural network, but a simplified function can be used also.

$$F((w_i - w_j)^\top \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

3. For word-word co-occurrence matrices, the distinction between a word and a context word is arbitrary. Hence, we are free to exchange the two roles, i.e. $w \leftrightarrow \tilde{w}$ and $X \leftrightarrow X^\top$.
4. Hence, model should be invariant under this relabeling. Thus

$$F((w_i - w_j)^\top \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} = \frac{F(w_i^\top \tilde{w}_k)}{F(w_j^\top \tilde{w}_k)} = \frac{X_{ik}}{X_{ij}}$$

5. $F = \exp$ is the solution of the above equation. Hence,

$$w_i^\top \tilde{w}_k = \log P_{ik} = \log X_{ik} - \log X_{ij}$$



1. We have,

$$w_i^T \tilde{w}_k = \log P_{ik} = \log X_{ik} - \log X_i$$

2. The above equation would exhibit the exchange symmetry if not for $\log X_i$ on the right-hand side.
3. Term $\log X_i$ is independent of k so it can be absorbed into a bias b_i for w_i .
4. Adding an additional bias \tilde{b}_k for \tilde{w}_k restores the symmetry. Hence

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log X_{ik}$$



1. In an ideal setting, where you have perfect word vectors, the following expression will be zero.

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k - \log X_{ik} = 0$$

2. Hence, we can define the objective function as

$$J(w_i, \tilde{w}_k) = \left(w_i^T \tilde{w}_k + b_i + \tilde{b}_k - \log X_{ik} \right)^2$$

3. Now, the final cost function is

$$J(w_i, \tilde{w}_k) = \sum_{i,k=1}^{|V|} f(X_{ik}) \left(w_i^T \tilde{w}_k + b_i + \tilde{b}_k - \log X_{ik} \right)^2$$

where f is a weighting function, which is defined manually.



1. GloVe becomes a global model for unsupervised learning of word representations that outperforms other models on word analogy, word similarity, and named entity recognition tasks.
2. Advantages
 - ▶ Fast training
 - ▶ Scalable to huge corpora
 - ▶ Good performance even with small corpus, and small vectors
 - ▶ Early stopping. We can stop training when improvements become small.
3. Drawbacks
 - ▶ Uses a lot of memory: the fastest way to construct a term-co-occurrence matrix is to keep it in RAM as a hash map and perform co-occurrence increments in a global manner
 - ▶ Sometimes quite sensitive to initial learning rate

Word embedding

Character-level embedding



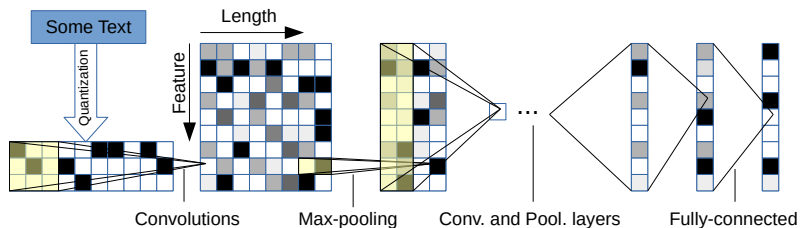
1. One drawback of Word2vec and Glove is the fact that they don't handle [out-of-vocabulary](#).
2. The other drawback of word2vec is [morphology](#). For example, for words with same radicals such as [eat](#) and [eaten](#), Word2Vec doesn't do any parameter sharing.
3. FastText introduced the concept of subword-level embeddings, based on the skip-gram model, but where each word is represented as a bag of character-grams (Bojanowski et al. [2017](#)).
4. A vector representation is associated to each character-gram, and words are represented as the sum of these representations.
5. This allows the model to compute word representations for words that did not appear in the training data.



1. The modification to the skip-gram/CBOW model are
2. Sub-word generation
 - ▶ Adding angular brackets to denote the beginning and end of a word. For example, `where` \mapsto `<where>`.
 - ▶ Taking a list of character n -grams a word. For example, for $n = 3$, we have `< wh, whe, her, ere, re >`.
 - ▶ Applying hashing to bound the memory requirements. This paper used `FNV-1a` variant of the `Fowler-Noll-Vo hashing`.
3. Using Skip-gram/CBOW with negative sampling
 - ▶ The input to Fasttext are `< wh, whe, her, ere, re >` and `<where>`.
 - ▶ The embedding for the center word is calculated by taking a sum of vectors for the character n -grams and the whole word itself.
4. In practice, they extracted all the `n-grams` for n greater or equal to `3` and smaller or equal to `6`.



1. A list of character are defined 70 characters including 26 English letters, 10 digits, 33 special characters and new line character.
2. The network architectures are: 9 layers deep with 6 convolutional layers and 3 fully-connected layers.

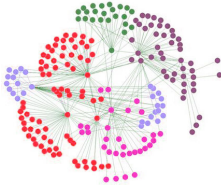


3. Please also read the paper (Józefowicz et al. 2016).

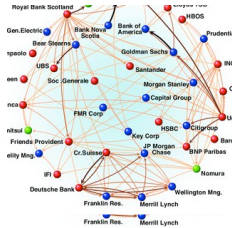
Graph embedding

1. Networks are a general language for describing and modeling complex systems.
2. Many data are networks such as

Social networks



Economic networks



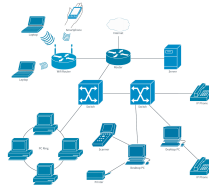
Biological networks



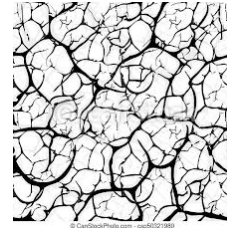
Citation networks



Internet



Networks of neurons





1. Why networks?

- ▶ Networks are a uniform language for describing complex data.
- ▶ Networks are used as a common language in many different fields such as computer, biology, social sciences.
- ▶ There are too many data sets available in the form of networks.

2. Machine learning tasks in networks

Node classification Predict a type of a given node.

Link prediction Predict whether two nodes are linked.

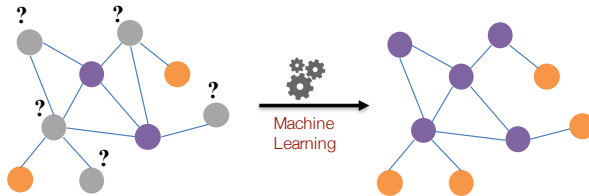
Community detection Identify densely linked clusters of nodes.

Network similarity How similar are two (sub)networks

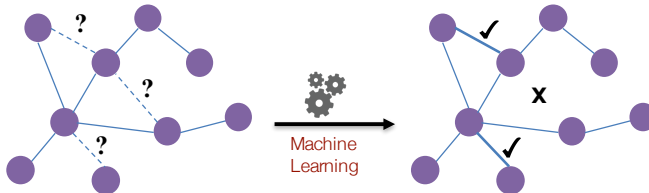
¹Most slides of graph embedding are taken from Leskovec et al. slides

1. Some examples of machine learning tasks in networks

Node classification



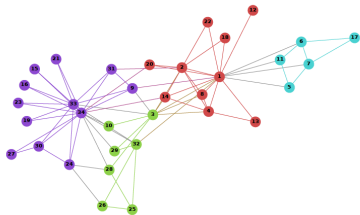
Link prediction



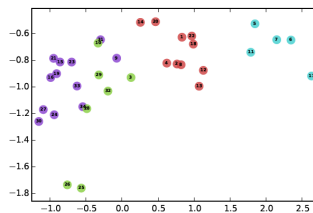


1. A graph $G = (V, E)$ is represented as a set of nodes V and a set of edges E represented as a binary matrix A .
2. The goal is to learn task-independent features from graphs.
3. For example, node embedding is to learn function $f : V \mapsto \mathbb{R}^d$, where similar nodes in the graph have embeddings that are close together.

Input graph



Learned features



4. This is a hard task, because of
 - ▶ Complex topographical structure and there is no spatial locality like grids
 - ▶ No fixed node ordering or reference point (i.e., the isomorphism problem)
 - ▶ Often dynamic and have multi-modal features.



1. Goal is to encode nodes so that *similarity in the embedding space* (e.g., dot product) approximates *similarity in the original network*.
2. Let \mathbf{z}_u be the embedding of node u .
3. Goal is to find the *encoder function* f such that $\text{similarity}(u, v) \approx \mathbf{z}_u^\top \mathbf{z}_v$.
4. Learning node embedding
 - ▶ Define an encoder
 - ▶ Define a node similarity function
 - ▶ Optimize the parameters of the encoder so that $\text{similarity}(u, v) \approx \mathbf{z}_u^\top \mathbf{z}_v$.
5. Two key components
 - ▶ Encoder function $f(u) = \mathbf{z}_u$.
 - ▶ Similarity measure $\text{similarity}(u, v) \approx \mathbf{z}_u^\top \mathbf{z}_v$.

Graph embedding

Similarity measures

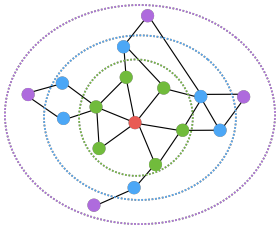


1. Similarity function is the weight of edge (u, v) in the original network.
2. The intuition is dot products $\mathbf{z}_u^\top \mathbf{z}_v$ approximate edge existence.
3. The loss function can be defined as

$$\ell = \sum_{(u,v) \in V \times V} \left\| \mathbf{z}_u^\top \mathbf{z}_v - \mathbf{A}_{uv} \right\|^2$$

4. Find embedding matrix $\mathbf{Z} \in \mathbb{R}^{d \times |V|}$ such that minimizes ℓ .
5. We can use the highly scalable and general method [stochastic gradient descent](#) or [matrix decomposition solvers](#) such as SVD, which works in limited cases.

1. Consider k -hop node neighbors.



- ▶ The red node is the target node.
- ▶ The green nodes are the 1-hop neighbors (A).
- ▶ The blue nodes are the 2-hop neighbors (A^2).
- ▶ The purple nodes are the 3-hop neighbors (A^3).

2. The idea is to find embeddings to predict k -hop neighbors.
3. The loss function can be defined as

$$\ell = \sum_{(u,v) \in V \times V} \left\| \mathbf{z}_u^T \mathbf{z}_v - \mathbf{A}_{uv}^k \right\|^2$$

4. Another option is to measure the overlap between node neighborhoods using overlap functions such as [Jaccard similarity](#).

Graph embedding

Random-walk approaches



1. The idea is to consider $\mathbf{z}_u^\top \mathbf{z}_v$ as probability that nodes u and v co-occur on a random walk over the network.
2. In random-walk embeddings, we
 - ▶ Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R .
 - ▶ Optimize embeddings to encode these random walk statistics.
3. Random walk is good because
 - ▶ It is flexible to define different node similarity and
 - ▶ It doesn't need to consider all node pairs in training.
4. In random-walk optimization, we
 - ▶ Run short random-walks starting from all nodes $u \in V$ using R , denoted by $N_R(v)$.
 - ▶ Find embedding maximizing likelihood of random-walk co-occurrences of each node

$$\ell = - \sum_{u \in V} \sum_{v \in N_R(u)} \log P(v | \mathbf{z}_u)$$
$$P(v | \mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}$$

5. Use negative sampling to improve the efficiency of optimization algorithm



1. The idea of **DeepWalk** is to use random walks to produce embeddings (Perozzi, Al-Rfou, and Skiena 2014).
2. The random walk starts in a selected node then we move to the random neighbor from a current node for a defined number of steps.
3. The method basically consists of three steps:

Sampling A graph is sampled with random walks. It is sufficient to perform from 32 to 64 random walks with length of about 40 steps from each node.

Training skip-gram Random walks are comparable to sentences in word2vec approach. The skip-gram network accepts a node from the random walk as a one-hot vector as an input and maximizes the probability for predicting neighbor nodes. It is typically trained to predict around 20 neighbor nodes (10 nodes left and 10 nodes right).

Computing embeddings Embedding is the output of a hidden layer of the network. The DeepWalk computes embedding for each node in the graph.

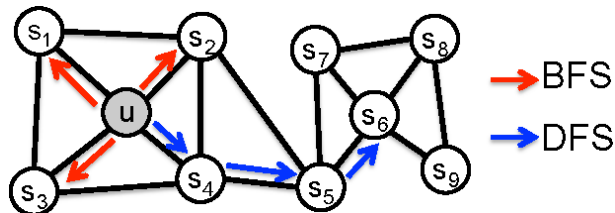
1. The DeepWalk process can be visualized as²



2. The model can take a target node to predict its **context**, which in the case of a graph, means its connectivity, structural role, and node features.
3. DeepWalk method performs random walks randomly what means that embeddings do not preserve the local neighborhood of the node well.

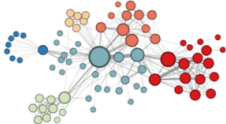
²Credit: Primož Godec

1. Node2vec is a modification of DeepWalk with a small difference in random walks.
2. The idea of `node2vec` is to use flexible, biased random walks that can trade off between `local` and `global` views of the network (Grover and Leskovec 2016).
3. Node2vec has two parameters p and q .
 - ▶ Parameter q defines how probable is that the random walk would discover the undiscovered part of the graph. This parameter prioritizes a breadth-first-search (BFS) procedure.
 - ▶ Parameter p defines how probable is that the random walk would return to the previous node. This parameter prioritizes a depth-first-search (DFS) procedure.
4. The decision of where to walk next is therefore influenced by probabilities $\frac{1}{p}$ or $\frac{1}{q}$.

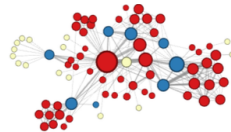


1. BFS is ideal for learning local neighbors, while DFS is better for learning global variables.
2. Node2vec can switch to and from the two priorities depending on the task.
3. This means that given a single graph, Node2vec can return different results depending on the values of the parameters.

DFS-based



BFS-based



4. Experiments demonstrated that BFS is better at classifying according to structural roles (hubs, bridges, outliers, etc.) while DFS returns a more community driven classification scheme.

Other 2vec embeddings



item2vec The embedding of items used in the collaborative filtering.

tweet2vec Finds vector-space representations of whole tweets.

entity2vec Learning embeddings of entities from text that describes them.

author2vec Learning author representations by combining content and link information.

emoji2vec Learning Emoji representations from their description.

image2vec Vector representation of images.

Speech2Vec Learning fixed-length vector representations of audio segments.

Time2Vec Learning a vector representation of time.

Wikipedia2Vec It is a tool used for obtaining embeddings of words and entities from Wikipedia.

Gene2vec Distributed representation of genes based on co-expression.

Skill2vec Learning the relevant skills from job description.

Reading







1. Chapter 14 of [Deep Learning Book](#)³

³Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.







-  Alain, Guillaume and Yoshua Bengio (2014). “What Regularized Auto-Encoders Learn from the Data-Generating Distribution”. In: *Journal of Machine Learning Research* 15.110, pp. 3743–3773. URL: <http://jmlr.org/papers/v15/alain14a.html>.
-  Bengio, Yoshua et al. (2003). “A Neural Probabilistic Language Model”. In: *Journal of Machine Learning Research* 3, pp. 1137–1155.
-  Blei, David M., Andrew Y. Ng, and Michael I. Jordan (2003). “Latent Dirichlet Allocation”. In: *Journal of Machine Learning Research* 3, pp. 993–1022.
-  Bojanowski, Piotr et al. (2017). “Enriching Word Vectors with Subword Information”. In: *Trans. Assoc. Comput. Linguistics* 5, pp. 135–146.
-  Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.
-  Grover, Aditya and Jure Leskovec (2016). “node2vec: Scalable Feature Learning for Networks”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 855–864.
-  Hinton, G. E. and R. R. Salakhutdinov (2006). “Reducing the Dimensionality of Data with Neural Networks”. In: *SCIENCE* 313, pp. 504–507.



-  Józefowicz, Rafal et al. (2016). “Exploring the Limits of Language Modeling”. In: *CoRR* abs/1602.02410.
-  Mikolov, Tomas, Kai Chen, et al. (2013). “Efficient Estimation of Word Representations in Vector Space”. In: *International Conference on Learning Representations*.
-  Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, et al. (2013). “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems 26*, pp. 3111–3119.
-  Mikolov, Tomas, Ilya Sutskever, Kai Chen, Gregory S. Corrado, et al. (2013). “Distributed Representations of Words and Phrases and their Compositionality”. In: *Proc. of Advances in Neural Information Processing Systems*, pp. 3111–3119.
-  Morin, Frederic and Yoshua Bengio (2005). “Hierarchical Probabilistic Neural Network Language Model”. In: *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS)*.
-  Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). “Glove: Global Vectors for Word Representation”. In: *Proc. of Advances in Neural Information Processing Systems*, pp. 1532–1543.



-  Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena (2014). “DeepWalk: online learning of social representations”. In: *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 701–710.
-  Rong, Xin (2016). “word2vec Parameter Learning Explained”. In: *CoRR* abs/1411.2738.
-  Uma, K. V. (2018). “Improving the Classification accuracy of Noisy Dataset by Effective Data Preprocessing”. In: *International Journal of Computer Applications* 180.36, pp. 37–46.
-  Zhang, Xiang, Junbo Zhao, and Yann LeCun (2015). “Character-level Convolutional Networks for Text Classification”. In: *NIPS*.

Questions?

