Word2Vec and Doc2Vec

Nagiza F. Samatova, samatova@csc.ncsu.edu
Professor, Department of Computer Science
North Carolina State University

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Word2Vec is a tool to create vector space representations of words

- **“embeddings” is a term often used instead of “vector space representation”:**
  - Embedding means representation of an object embedded (placed) in a vector space.
- **Word2Vec produces vector space representations of words that:**
  - Capture co-occurrence of words in text
  - Captures sematic meaning by accounting for word order
- **Word2Vec is a shallow neural network**
  - Input layer, hidden layer, and output layer
  - The number of neurons in the hidden layer is the number of dimensions in the vector space
  - The neural net in Word2Vec is only used to obtain the weights, it is not used in the normal way – e.g. to perform classification with forward propagation.
- **Word2Vec has two different versions:**
  - continuous bag of words and
  - skip-gram

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Ex: Semantic Meaning of Relationships in Vector Space

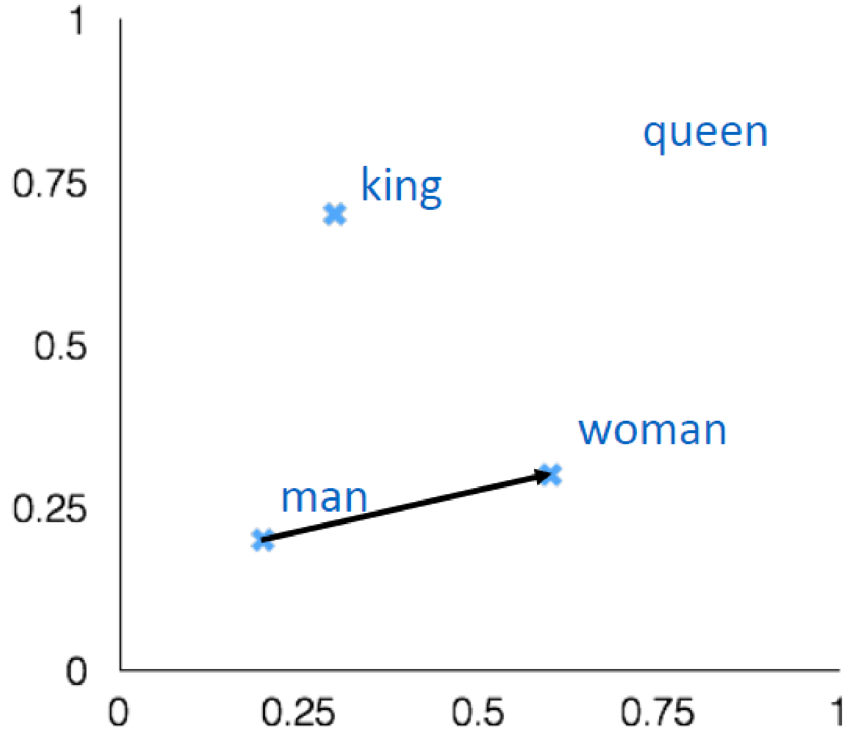
Test for linear relationships, examined by Mikolov et al. (2014)

a:b :: c:? → $$d = \arg\max_{x} \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

Word Analogies

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



http://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf (Author of GloVe)

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

word2vec: Input, output, parameters, training…

- **Input**
  - Corpus – the text that you train word2vec model on (e.g., the entire Wikipedia, your Inbox, all Harry Potter books, etc.)
  - Vocabulary (it is extracted by word2vec but could be an input)
- **Parameters**
  - Number of dimensions of the target vector space – N:
    - Usually small: 64, 128, …, under 1,000
    - Unlike bag of words: ~100,000 of words, with each word as a dimension
  - context window size:
    - Usually: 5-11 words
  - others
- **Output**
  - Vectors in the N-dimensional space (Euclidean space)
    - Each word is represented by an array of N numbers

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Continuous Bag of Words (CBOW): One-word context

We assume that there is only one word considered per context.

Model will predict one target word given one context word

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Neural network: One-word context

The input vector $\boldsymbol{x} = \{x_1, \dots, x_V\}$ is a one-hot encoded vector

E.g. “eats” -> [0 0 0 1 0 0 0]

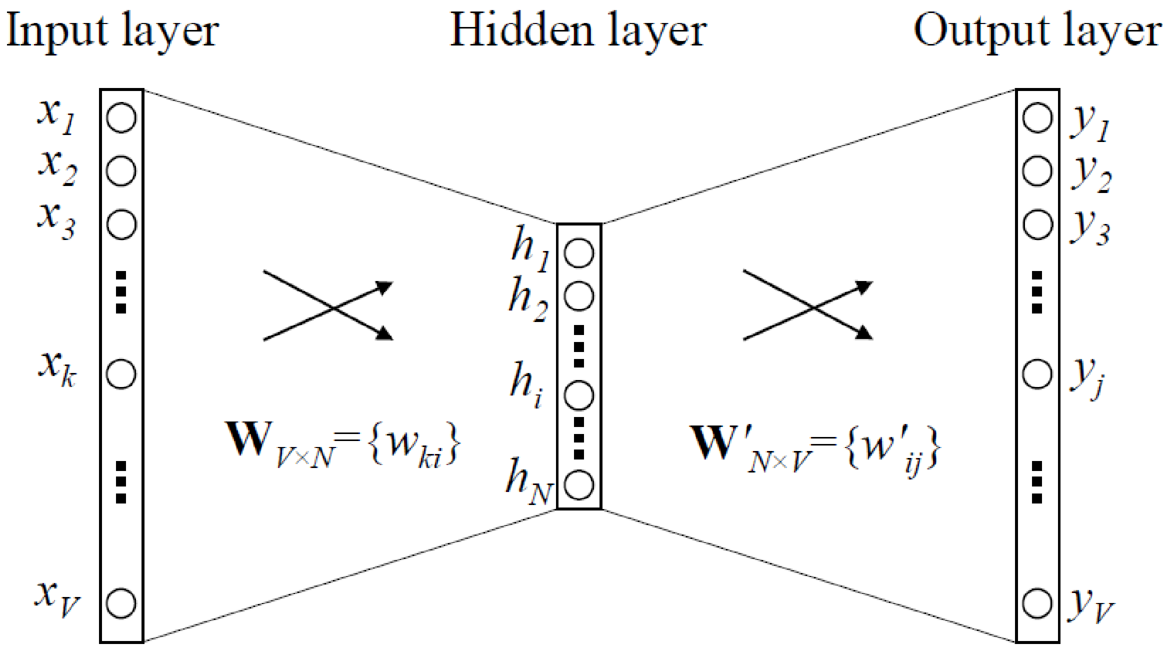
V is the vocabulary size

N is size of the hidden layer

The weights between the input and the hidden layer can be represented by a $V \times N$ matrix $\boldsymbol{W}$

Each row of $\boldsymbol{W}$ is the N-dimensional vector $\boldsymbol{v}_w$

$\boldsymbol{v}_w$ is the vector representation of associated word of the input layer.



The weights between the hidden and the output layer can be represented by a $V \times N$ matrix $\boldsymbol{W}'$

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

CBOW: Bag of words context

We assume that there several words considered per context.

Model will predict one target word given many context word

context

target

Cookie monster eats → cookies

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Continuous bag-of-words neural network architecture

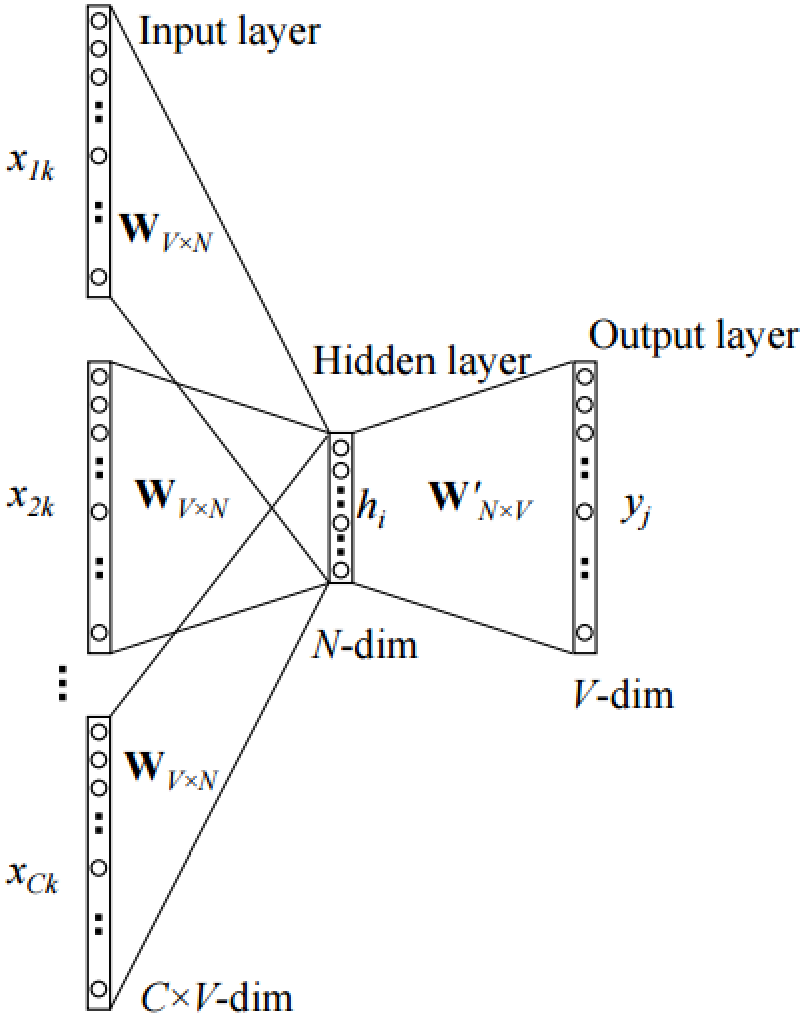


Figure 2: Continuous bag-of-word model

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Skip-Gram Model: One-word target -> context

We assume that there is only one word considered in target and many (window-size) in context.

Model will predict many context words given one target word

target

context

eats → Cookie monster ____ cookies

- Skip-gram example:
  - 3-grams:
    - Cookie, monster, cookies
    - Cookie,eats, cookies
    - monster, eats, cookies

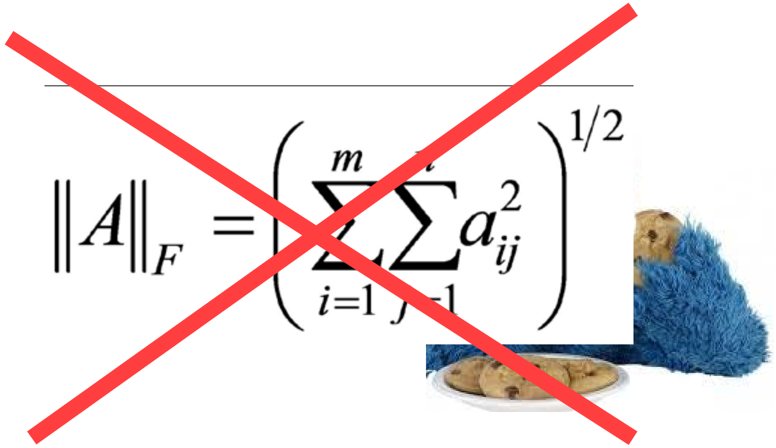
© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Skip-Gram with **Negative Sampling**

Idea: don't just use 1's for the words you see in the context also use 0 for the words you don't see in the context

target → context

eats → **Frobenius norm eats cookies**



© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

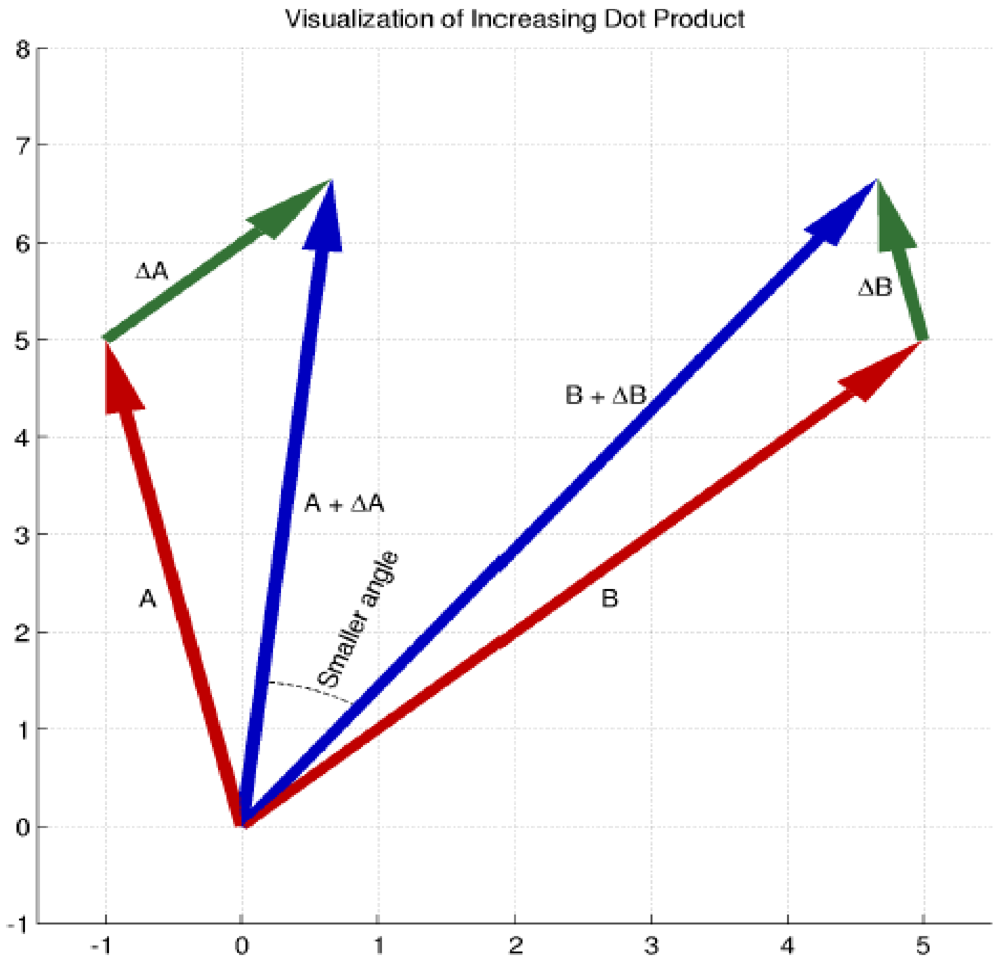
Skip-gram overview

- Each word corresponds to two vectors, one is the target word and the other as a context word.
- The algorithm starts with random vectors for the context and zero vectors for the target words.
- Each time one word occurs in the context of the other, the context vector and target vector are modified slightly so that their product is slightly larger.

Dissertation by Eric D. Moyer
“What machines understand
about personality words after
reading the news”

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Intuition: minimization of context-target dot-product



- Each time one word occurs in the context of the other, the context vector and target vector are modified slightly so that their dot product is slightly larger.
- the dot product between a word and a target will be proportional to the probability of that word appearing in the context of that target when compared to the dot products of other words also appearing in that context.
- This means that words will cluster around their most common contexts.

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Training skip-gram model on a tiny example

Corpus: Mary had a little lamb, little lamb little, little lamb Mary had a little lamb, little lamb little, little lamb Mary had a little lamb, little lamb little, little lamb Mary had a little lamb, little lamb little, little lamb

Vocabulary (V=5): Mary, had, a, little, lamb

Dimensionality: N = 2

Context window size: w = 1

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Training skip-gram model

Mary had a little lamb, little lamb, little lamb, little lamb
Mary had a little lamb, little lamb, little lamb, little lamb
Mary had a little lamb, little lamb, little lamb, little lamb
Mary had a little lamb, little lamb, little lamb, little lamb

Mary had a little lamb, little lamb, little lamb, little lamb
Mary had a little lamb, little lamb, little lamb, little lamb
Mary had a little lamb, little lamb, little lamb, little lamb
Mary had a little lamb, little lamb, little lamb, little lamb

Mary had a little lamb, little lamb, little lamb, little lamb
Mary had a little lamb, little lamb, little lamb, little lamb
Mary had a little lamb, little lamb, little lamb, little lamb
Mary had a little lamb, little lamb, little lamb, little lamb

context

target

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

One-hot encoding

Index	Word	One-hot Encoding vector
0	Mary	[1,0,0,0,0]
1	had	[0,1,0,0,0]
2	a	[0,0,1,0,0]
3	little	[0,0,0,1,0]
4	lamb	[0,0,0,0,1]

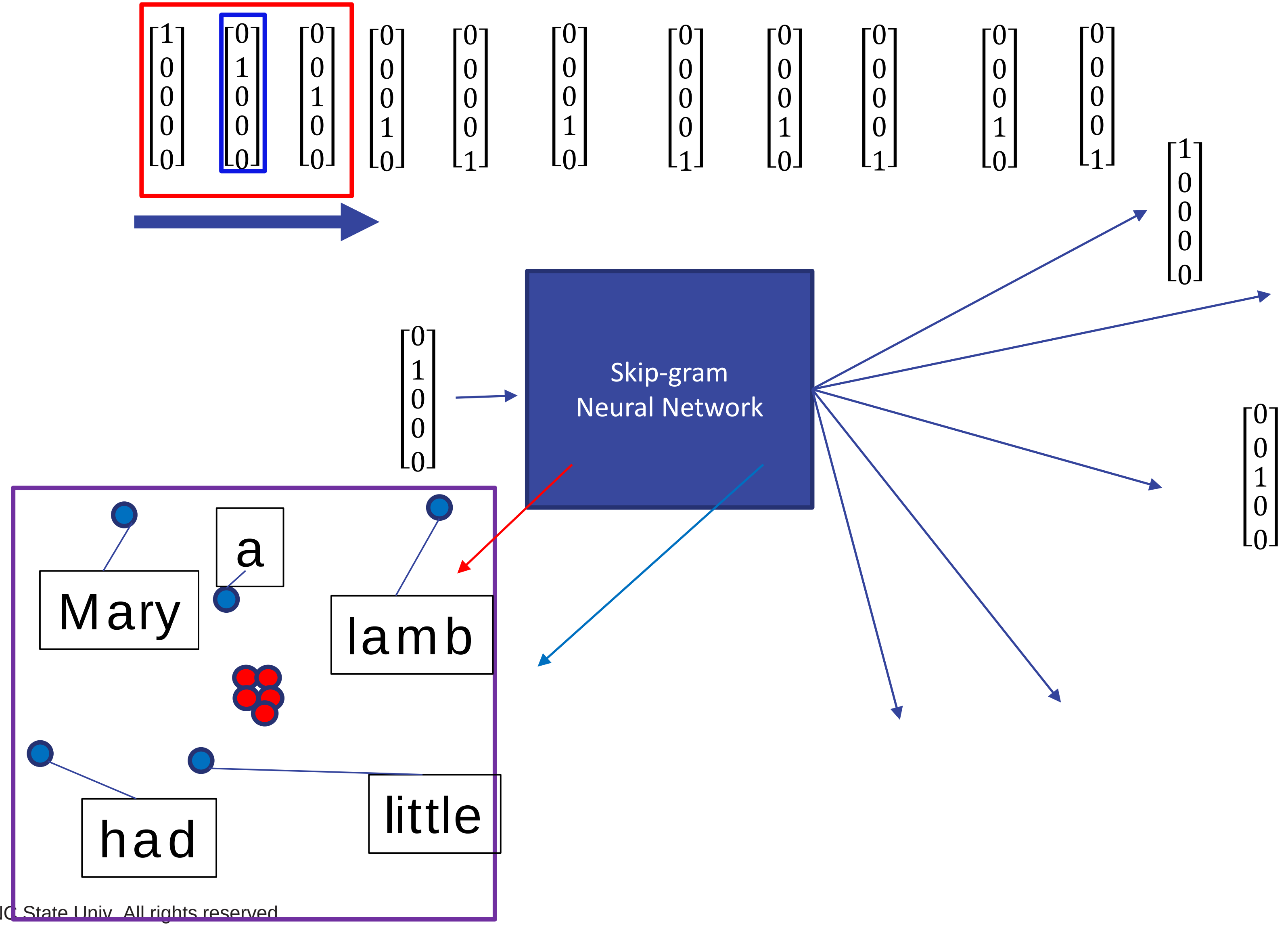
Mary had a little lamb, little lamb, little lamb, little lamb

$$\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix}$$

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Initialized weights in neural net and start

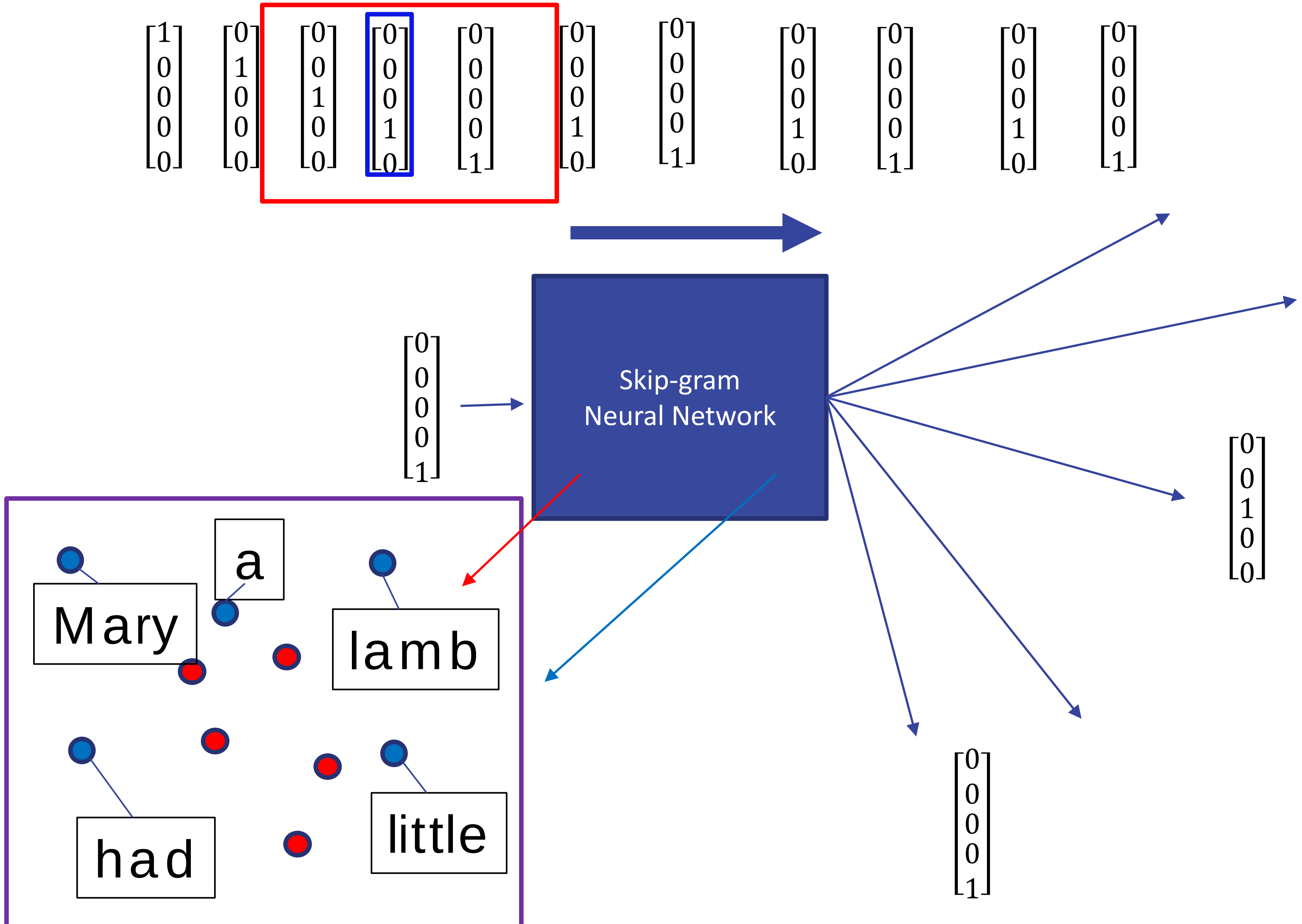
Mary had a little lamb, little lamb, little lamb, little lamb



© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Training neural network

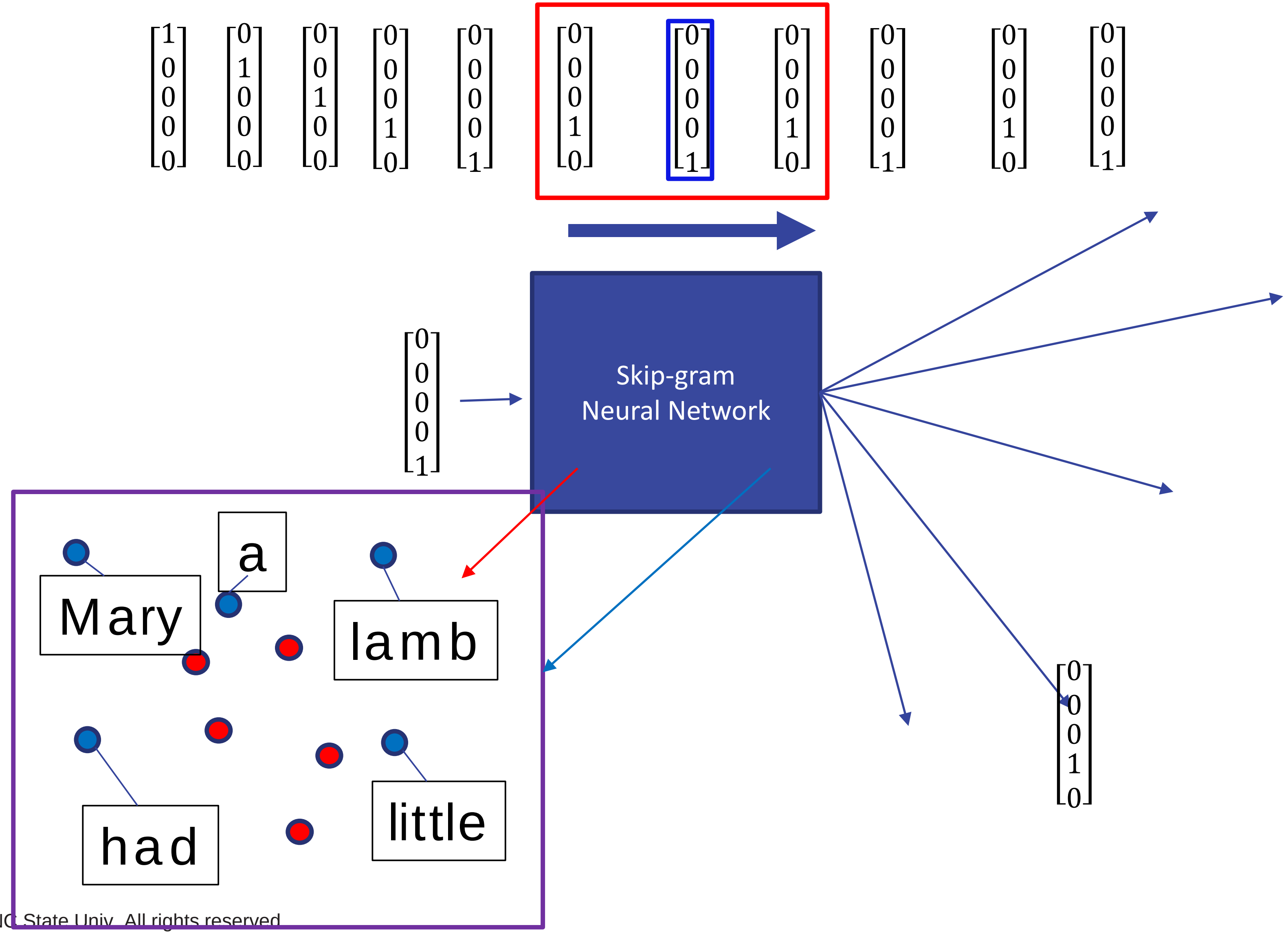
Mary had a little lamb, little lamb, little lamb, little lamb



© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Training neural network

Mary had a little lamb, little lamb, little lamb, little lamb

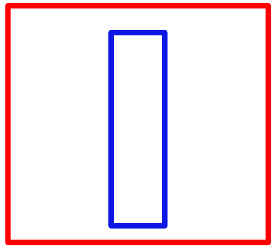


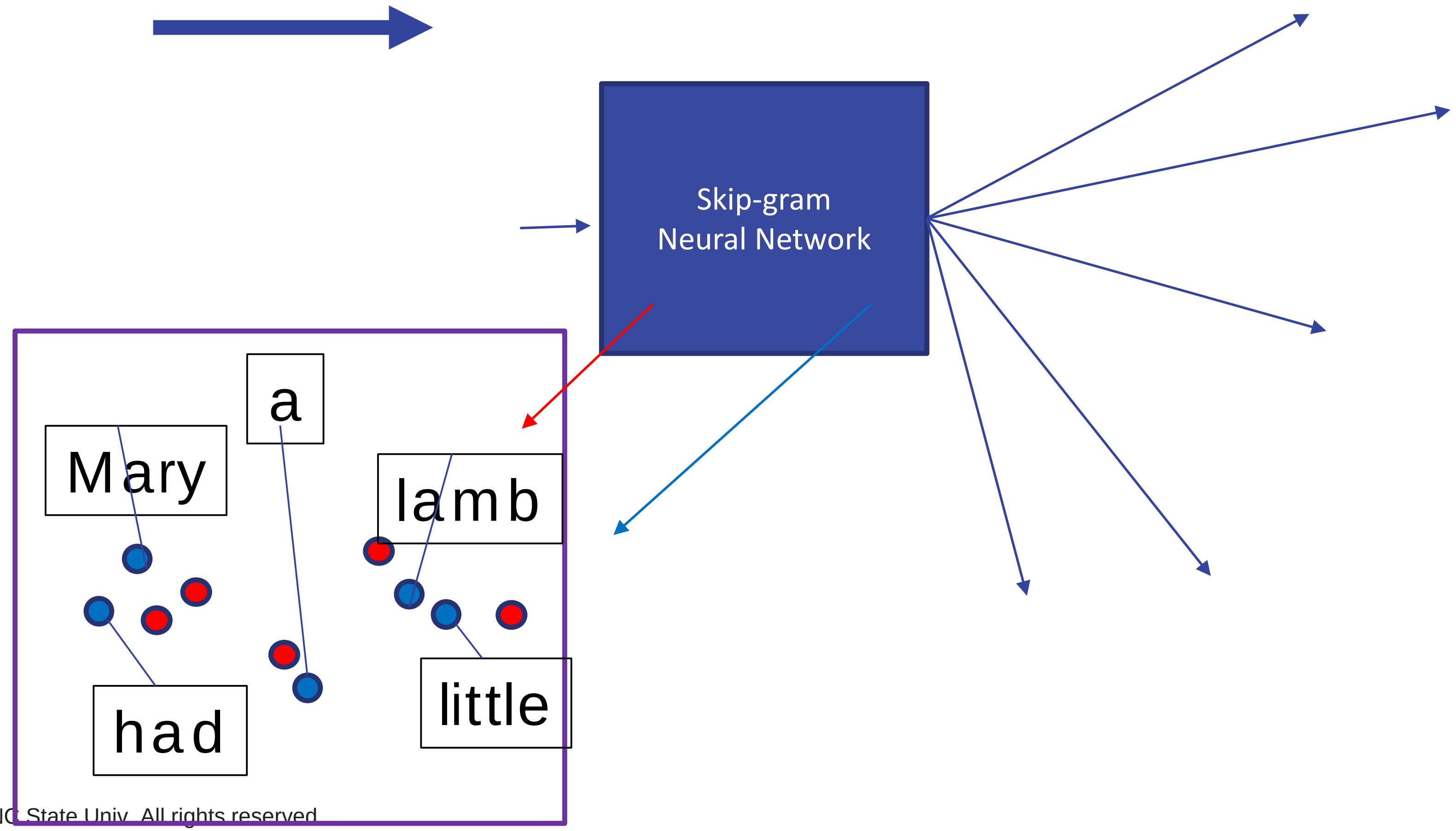
© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Finished training neural network

Mary had a little lamb, little lamb, little lamb, little lamb

$$\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix}\begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\\0\\1\end{bmatrix}$$





© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Skip-gram model neural net

- The target word is in the input layer.
- The context words are in the output layer

v_{w_I} is the input vector of the only word on the input layer

For a given input word, and assuming its index in the vocabulary k, one-hot input vector representation is x.

$$x_i = \begin{cases} 1 \ for \ i = k \\ 0 \ for \ i \neq k \end{cases}$$

$$v_{w_I} \coloneqq \boldsymbol{h} = \boldsymbol{x}^T \boldsymbol{W} = \boldsymbol{W}_{(k,\cdot)}$$

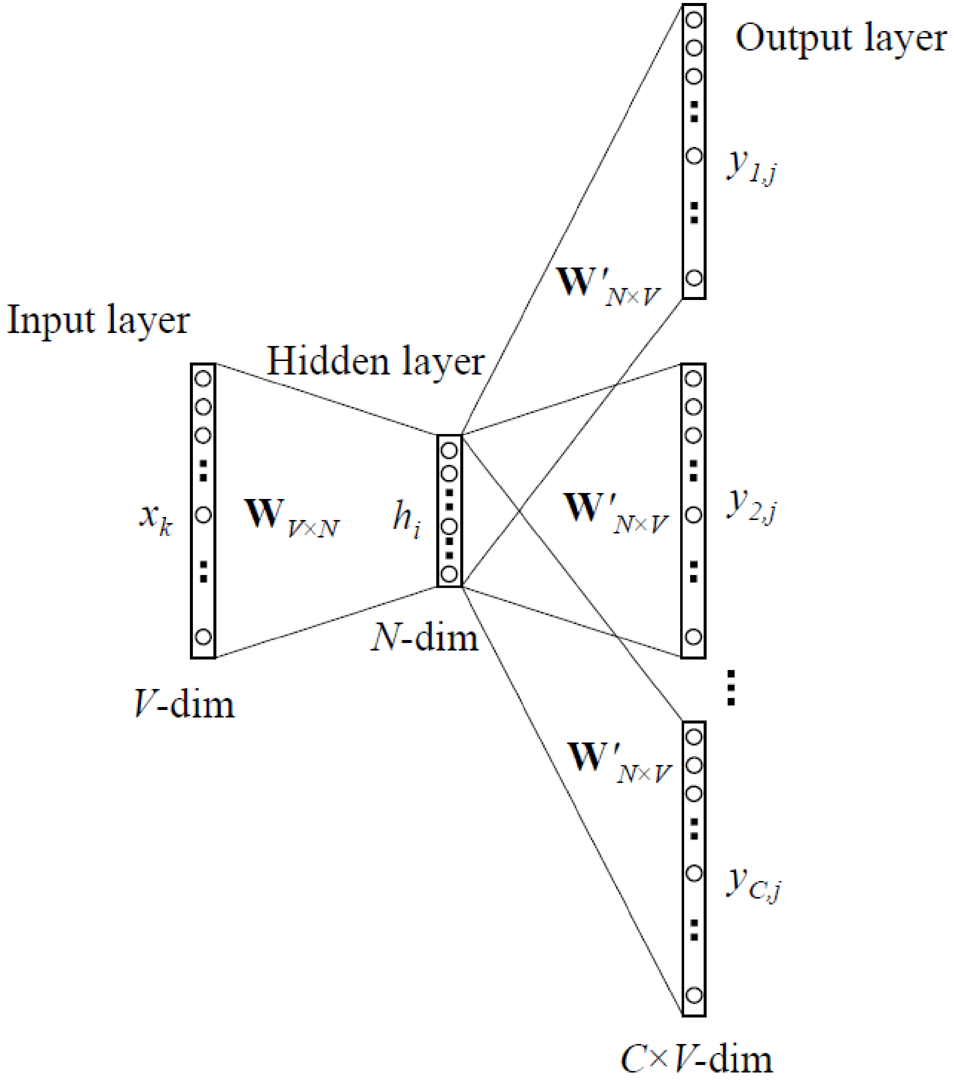


Figure 3: The skip-gram model.

It is the k-th row of W, which is the input->hidden weight matrix

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

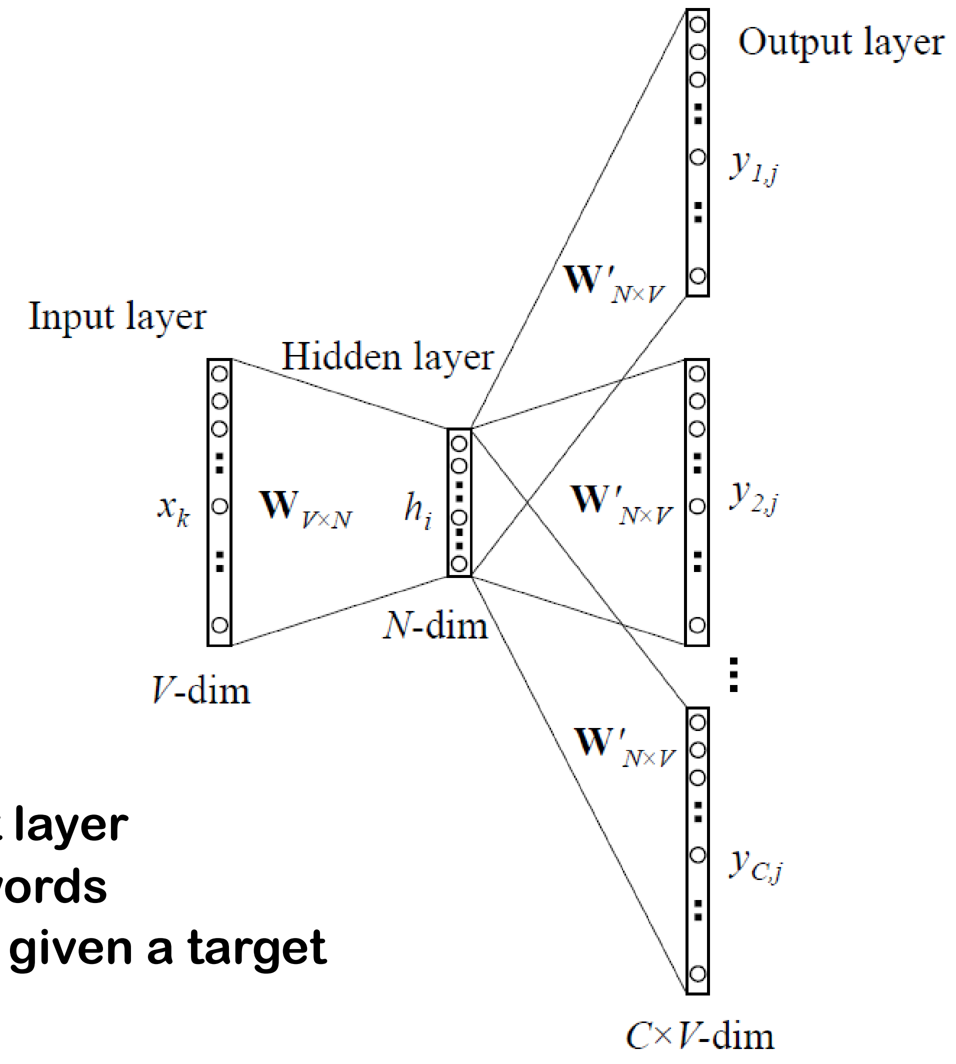
Skip-gram model: details

One the output layer we output C multinomial distributions

Each output computed using the same hidden->output weight matrix W' and softmax link function:

$$y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u'_j)}$$

- Also note that we hope that:
- $y_{c,j} = p(w_{c,j} = w_{O,c} | w_I)$
- $w_{c,j}$ is the j-th word on the c-th panel of the output layer
- $w_{O,c}$ is the actual c-th word in the output context words
- conditional probability of observing context word given a target word



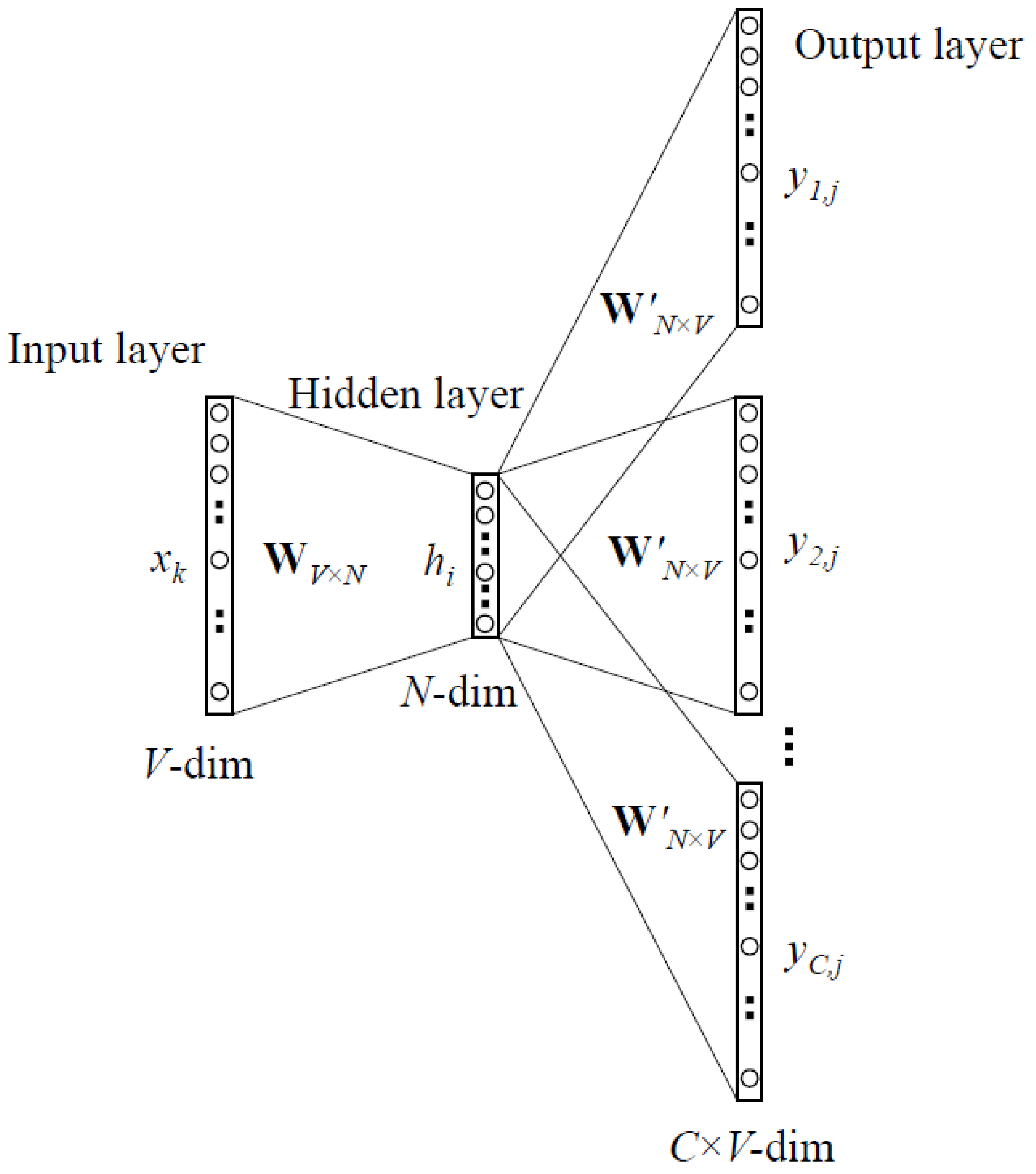
© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Skip-gram model: output

$$y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{j'=1}^{V} \exp(u'_j)}$$

$$u_j = \boldsymbol{v}'^{T}_{w_j} \cdot \boldsymbol{h} \text{ for } c = 1,2,\dots,C$$

- $\boldsymbol{v}'^{T}_{w_j}$ is the output vector of the j-th word in the vocabulary, w_j
- $\boldsymbol{v}'^{T}_{w_j}$ is a column from the hidden->output matrix, $\boldsymbol{W}'$



© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Python implementation of word2vec

- https://radimrehurek.com/**gensim**/models/word2vec.html
  https://rare-technologies.com/word2vec-tutorial/

```
>>> model = Word2Vec(sentences, size=100, window=5, min_count=5, workers=4)
```

```
>>> model.most_similar(positive=['woman', 'king'], negative=['man'])
[('queen', 0.50882536), ...]

>>> model.doesnt_match("breakfast cereal dinner lunch".split())
'cereal'

>>> model.similarity('woman', 'man')
0.73723527

>>> model['computer']  # raw numpy vector of a word
array([-0.00449447, -0.00310097,  0.02421786, ...], dtype=float32)
```

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Doc2Vec: Distributed Representations of Sentences and Documents

- Doc2Vec has 2 models:
  - Paragraph Vector Distributed Bag of Words (PV-DBOW)
  - Paragraph Vector Distributed Memory (PV-DM)

https://cs.stanford.edu/~quocle/paragraph_vector.pdf

Distributed Representations of Sentences and Documents

Quoc Le QVL@GOOGLE.COM
Tomas Mikolov TMIKOLOV@GOOGLE.COM
Google Inc, 1600 Amphitheatre Parkway, Mountain View, CA 94043

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

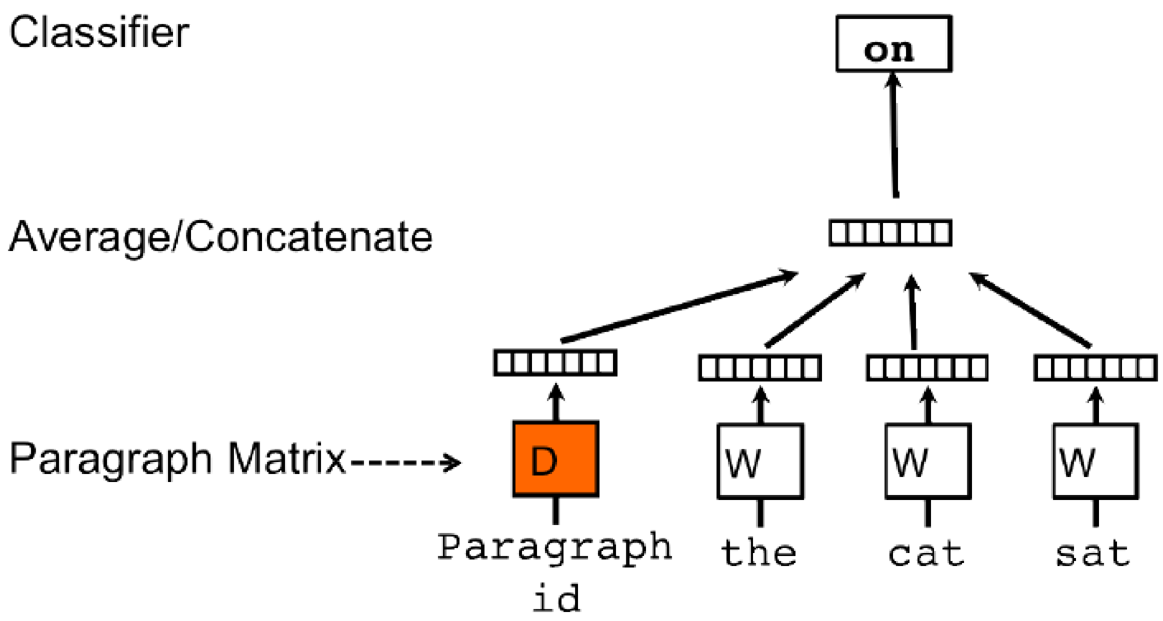
Doc2Vec

- “In our model, the vector representation is trained to be useful for predicting words in a paragraph.
- ...we concatenate the paragraph vector with several word vectors from a paragraph and predict the following word in the given context.
- Both word vectors and paragraph vectors are trained by the stochastic gradient descent and backpropagation.
- While paragraph vectors are unique among paragraphs, the word vectors are shared.
- At prediction time, the paragraph vectors are inferred by fixing the word vectors and training the new paragraph vector until convergence.”

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Paragraph vector: distributed memory

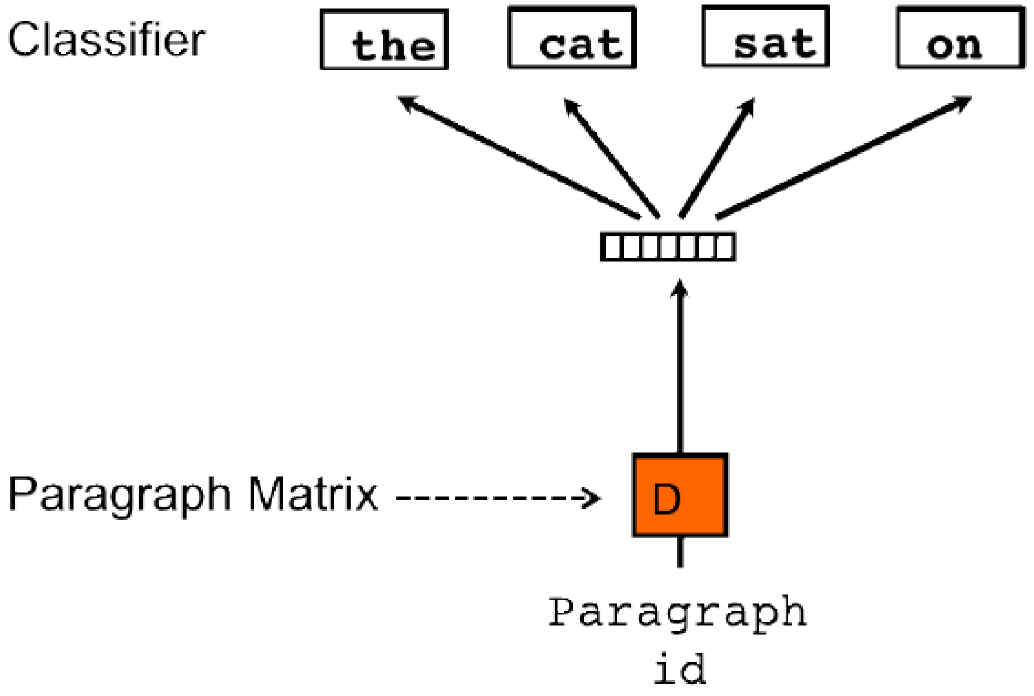
- every **paragraph** is mapped to a unique **vector**, represented by a column in matrix D
- and every **word** is also mapped to a unique **vector**, represented by a column in matrix W.
- The paragraph vector and word vectors are averaged or **concatenated** to predict the next word in a context.



© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Paragraph vector: distributed bag of words

- ...ignore the context words in the input, but force the model to predict words randomly sampled from the paragraph in the output.
- In reality, what this means is that at each iteration of stochastic gradient descent, we sample a text window, then sample a random word from the text window and form a classification task given the Paragraph Vector.



© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Doc2Vec usage with gensim package in Python

https://radimrehurek.com/gensim/models/doc2v

```
>>> model = Doc2Vec(documents, size=100, window=8, min_count=5, workers=4)
```

```
>>> trained_model.n_similarity(['sushi', 'shop'], ['japanese', 'restaurant'])
0.6154046656104969

>>> trained_model.n_similarity(['restaurant', 'japanese'], ['japanese', 'restaurant'])
1.0000000000000004
```

- class gensim.models.doc2vec.Doc2Vec(documents=None, size=300, alpha=0.025, window=8, min_count=5, max_vocab_size=None, sample=0, seed=1, workers=1, min_alpha=0.0001, dm=1, hs=1, negative=0, dbow_words=0, dm_mean=0, dm_concat=0, dm_tag_count=1, docvecs=None, docvecs_mapfile=None, comment=None, trim_rule=None, **kwargs)

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Project: Sentiment analysis with Doc2Vec

- perform sentiment analysis over IMDB movie reviews and Twitter data.
- build BOW, Doc2Vec, Word2Vec models with labeled training and testing data to evaluate the model.
- classify tweets or movie reviews as either positive or negative given labeled training For classification, you will experiment with logistic regression as well as a Naive Bayes classifier from python's well-regarded machine learning package scikit-learn.

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Project steps

1. Load datasets

```
def main():
    train_pos, train_neg, test_pos, test_neg = load_data(path_to_data)
```

2. Train Doc2Vec model

```
    if method == "d2v":
        train_pos_vec, train_neg_vec, test_pos_vec, test_neg_vec =
        feature_vecs_DOC(train_pos, train_neg, test_pos, test_neg)
        #filename = './'+path_to_data+'train_pos_vec_d2v.txt'
```

3. Train logistic regression model with Doc2Vec vector-features

```
        nb_model, lr_model = build_models_DOC(train_pos_vec,
        train_neg_vec)
```

4. Evaluate the model on test data

```
    evaluate_model(lr_model, test_pos_vec, test_neg_vec, True)
```

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Training Doc2Vec steps:

Step 1. Create TaggedDocument-s from lists of words

```
labeled_train_pos = [TaggedDocument(words, ["TRAIN_POS_" + str(i)])
for i, words in enumerate(train_pos)]
labeled_train_neg = [TaggedDocument(words, ["TRAIN_NEG_" + str(i)])
for i, words in enumerate(train_neg)]
labeled_test_pos = [TaggedDocument(words, ["TEST_POS_" + str(i)])
for i, words in enumerate(test_pos)]
labeled_test_neg = [TaggedDocument(words, ["TEST_NEG_" + str(i)])
for i, words in enumerate(test_neg)]
```

Step 2. Initialize the model

```
model = Doc2Vec(min_count=1, window=10, size=100, sample=1e-4,
negative=5, workers=4)
```

```
sentences = labeled_train_pos + labeled_train_neg + labeled_test_pos
 + labeled_test_neg
model.build_vocab(sentences)
```

Step 3. Train the model

```
for i in range(5):
    print("Training iteration %d" % (i))
    random.shuffle(sentences)
    model.train(sentences,total_examples=model.corpus_count, epochs=
    model.iter)
```

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Training Doc2Vec steps:

Step 4. Extract vectors

```
    # Use the docvecs function to extract the feature vectors for the training and test data
    train_pos_vec = [model.docvecs["TRAIN_POS_" + str(i)] for i in range(len(labeled_train_pos))]
    train_neg_vec = [model.docvecs["TRAIN_NEG_" + str(i)] for i in range(len(labeled_train_neg))]
    test_pos_vec = [model.docvecs["TEST_POS_" + str(i)] for i in range(len(labeled_test_pos))]
    test_neg_vec = [model.docvecs["TEST_NEG_" + str(i)] for i in range(len(labeled_test_neg))]

    # Return the four feature vectors
    return train_pos_vec, train_neg_vec, test_pos_vec, test_neg_vec
```

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Build logistic regression model with Doc2Vec vectors

```
def build_models_DOC(train_pos_vec, train_neg_vec):
    """
    Returns a GaussianNB and LosticRegression Model that are fit to the
    training data.
    """
    Y = ["pos"]*len(train_pos_vec) + ["neg"]*len(train_neg_vec)

    # Use sklearn's GaussianNB and LogisticRegression functions to fit
    two models to the training data.
    # For LogisticRegression, pass no parameters
    X = train_pos_vec + train_neg_vec
    nb_model = sklearn.naive_bayes.GaussianNB()
    nb_model.fit(X, Y)
    lr_model = sklearn.linear_model.LogisticRegression()
    lr_model.fit(X, Y)
    return nb_model, lr_model
```

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Evaluating the model

```
Doc2Vec
Training iteration 0
Training iteration 1
Training iteration 2
Training iteration 3
Training iteration 4
end of training
Naive Bayes
-----------
predicted:      pos     neg
actual:
pos             5482    7018
neg             2262    10238
accuracy: 0.628800

Logistic Regression
-------------------
predicted:      pos     neg
actual:
pos             10745   1755
neg             1700    10800
accuracy: 0.861800
```

```python
def evaluate_model(model, test_pos_vec, test_neg_vec, print_confusion=False):
    """
    Prints the confusion matrix and accuracy of the model.
    """
    # Use the predict function and calculate the true/false positives and true/false negative.
    pos_results = list(model.predict(test_pos_vec))
    neg_results = list(model.predict(test_neg_vec))
    tp = pos_results.count("pos")
    tn = neg_results.count("neg")
    fn = pos_results.count("neg")
    fp = neg_results.count("pos")
    accuracy = float(tp + tn) / (tp + tn + fp + fn)

    if print_confusion:
        print("predicted:\tpos\tneg")
        print("actual:")
        print("pos\t\t%d\t%d" % (tp, fn))
        print("neg\t\t%d\t%d" % (fp, tn))
    print("accuracy: %f" % (accuracy))
```

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

If you would like to write your own implementation of word2vec's neural net …

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Error function for Skip-gram model

$$E = -\log p(w_{O,1}, w_{O,2}, \ldots, w_{O,C} | w_I) =$$

$$-\log \prod_{c=1}^{C} \frac{\exp(u_{c,j_c^*})}{\sum_{j'=1}^{V} \exp(u_j')} =$$

$$-\sum_{c=1}^{C} u_j^* + C \cdot log \sum_{j'=1}^{V} \exp(u_{j'})$$

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

Now we need the gradient for the stochastic gradient descent

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Prediction errors and gradients

Taking the derivative of E with regard to the net input of every node on every panel of the output layer, $u_{c,j}$ and setting it as prediction error:

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

Defining EI_i as a sum of prediction errors over all context words:

$$EI_j = \sum_{c=1}^{C} e_{c,j}$$

Taking the derivative of E with regard to the hidden->output matrix $\boldsymbol{W}'$ with elements w'_{ij}

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

Thus we obtain the update equation for the hidden->output matrix $\boldsymbol{W}'$

$$w'^{\,new}_{ij} = w'^{\,old}_{ij} - \eta \cdot EI_j \cdot h_i$$

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.

Update Equations for Skip-gram Model

$$\boldsymbol{v}'_{w_j}{}^{new} = \boldsymbol{v}'_{w_j}{}^{old} - \eta \cdot EI_j \cdot \boldsymbol{h}$$ update of "context" vectors

$$\boldsymbol{v}_{w_I}^{new} = \boldsymbol{v}_{w_I}^{old} - \eta \cdot \boldsymbol{EH}$$ update of "target" vectors

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

$e_{c,j} := y_{c,j} - t_{c,j}$ is the prediction error on the node

$\boldsymbol{EI} = \{EI_1, \dots, EI_V\}$ is a V –dimensional vector, the sum of all prediction errors over all context words

$$EI_j = \sum_{c=1}^{C} e_{c,j}$$

© 2018 Nagiza F. Samatova, NC State Univ. All rights reserved.