Natural Language Processing Internals-1

By Mohit Kumar

Word Represtations

- How do we represent the meaning of a word meaning (dictionary)
  - The idea that is represented by a word ,phrase etc.
  - The idea that a person wants to express by words of signs.
  - The idea that is expressed in a work of writing, art , etc

Commonest linguistic way of thinking of meaning:

- signifier $\Leftrightarrow$ signified (idea or thing) = denotation

Word Represtations:In a Computer?

Common answer: Use a taxonomy like WordNet that has hypernyms (is-a) relationships and synonym sets

```
from nltk.corpus import wordnet as wn
panda = wn.synset('panda.n.01')
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

(here, for *good*):

```
S: (adj) full, good
S: (adj) estimable, good, honorable, respectable
S: (adj) beneficial, good
S: (adj) good, just, upright
S: (adj) adept, expert, good, practiced,
proficient, skillful
S: (adj) dear, good, near
S: (adj) good, right, ripe
...
S: (adv) well, good
S: (adv) thoroughly, soundly, good
S: (n) good, goodness
S: (n) commodity, trade good, good
```

Word Represtations:In a Computer?:Discreet:Problems

- Great as a resource but missing nuances, e.g., **synonyms**:
  - adept, expert, good, practiced, proficient, skillful?
- Missing new words (impossible to keep up to date): wicked, badass, nifty, crack, ace, wizard, genius, ninja
- Subjective
- Requires human labor to create and adapt
- Hard to compute accurate word similarity →

Word Represtations:In a Computer?:Discreet:Problems

The vast majority of rule-based **and** statistical NLP work regards words as atomic symbols: hotel, conference, walk

In vector space terms, this is a vector with one 1 and a lot of zeroes

[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

Dimensionality: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

We call this a "one-hot" representation

It is a **localist** representation

Word Represtations:Discreet to distributed

Its problem, e.g., for web search

- If user searches for [Dell notebook battery size], we would like to match documents with "Dell laptop battery capacity"
- If user searches for [Seattle motel], we would like to match documents containing "Seattle hotel"

But

motel $[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T$

hotel $[0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0] = 0$

Our query and document vectors are **orthogonal**

There is no natural notion of similarity in a set of one-hot vectors

Could deal with similarity separately;

instead we explore a direct approach where vectors encode it

Word Represtations:Distributional similarity based

You can get a lot of value by representing a word by means of its neighbors

"You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

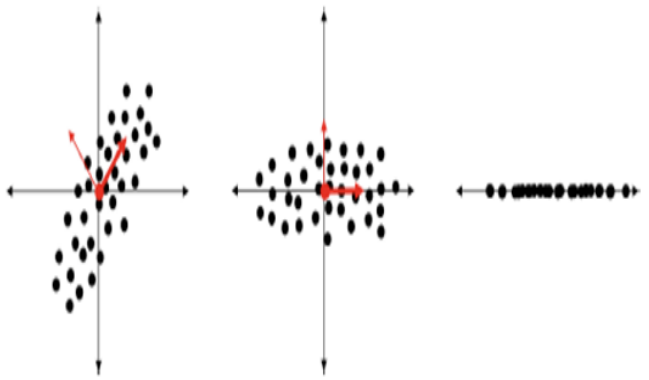
One of the most successful ideas of modern statistical NLP

government debt problems turning into	banking	crises as has happened in
saying that Europe needs unified	banking	regulation to replace the hodgepodge

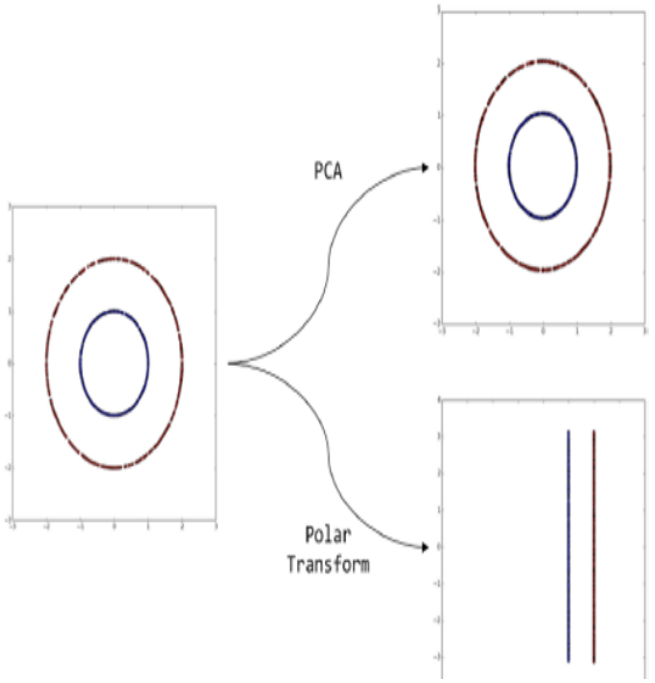
↖ These words will represent *banking* ↗

Word Represtations:Distributional similarity based:Autoencoder

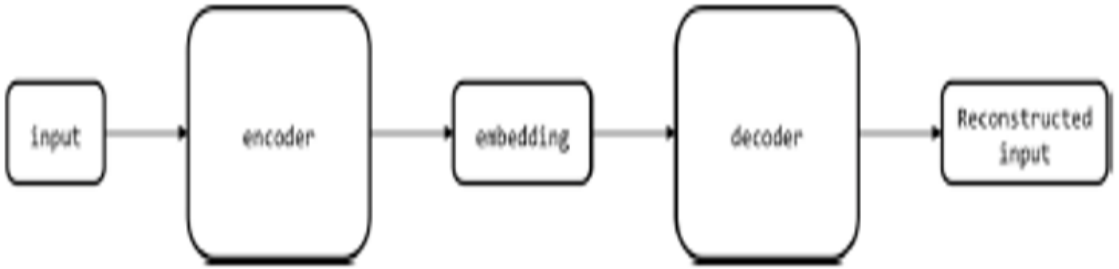
Inspiration

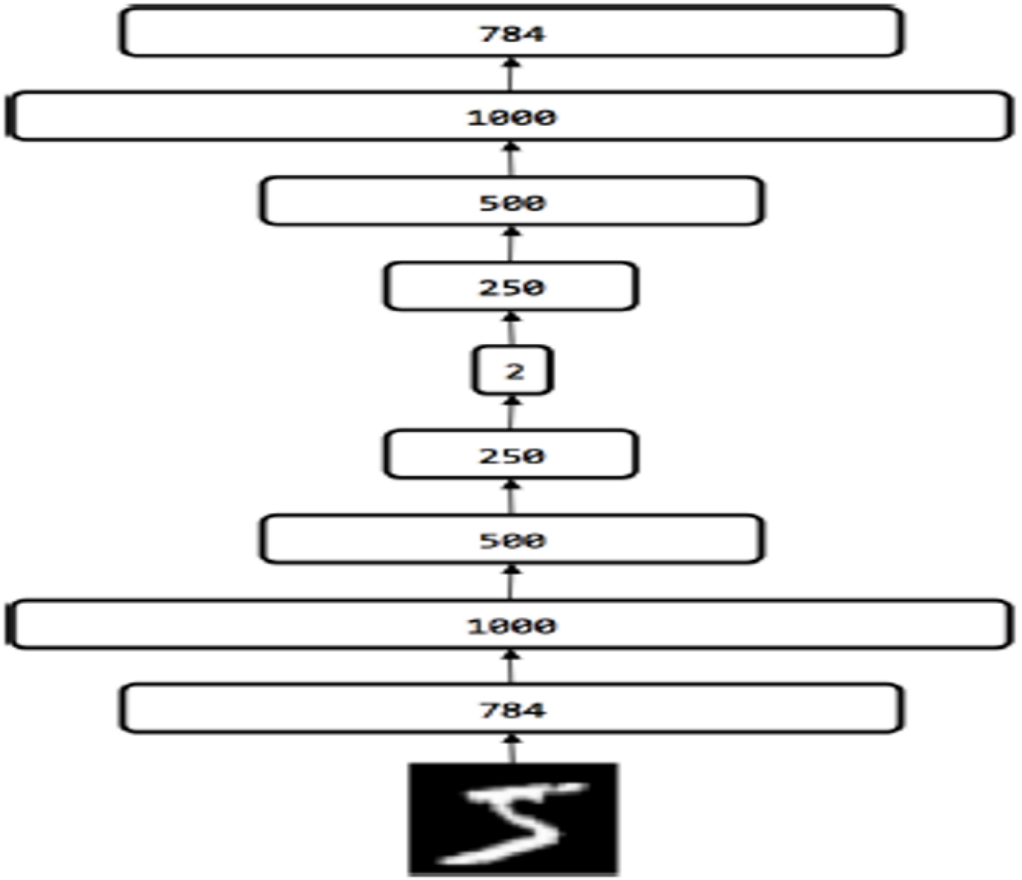


- While PCA is good at finding dominant dimensions, it is not so effective when the dominant-dimension is not linear.

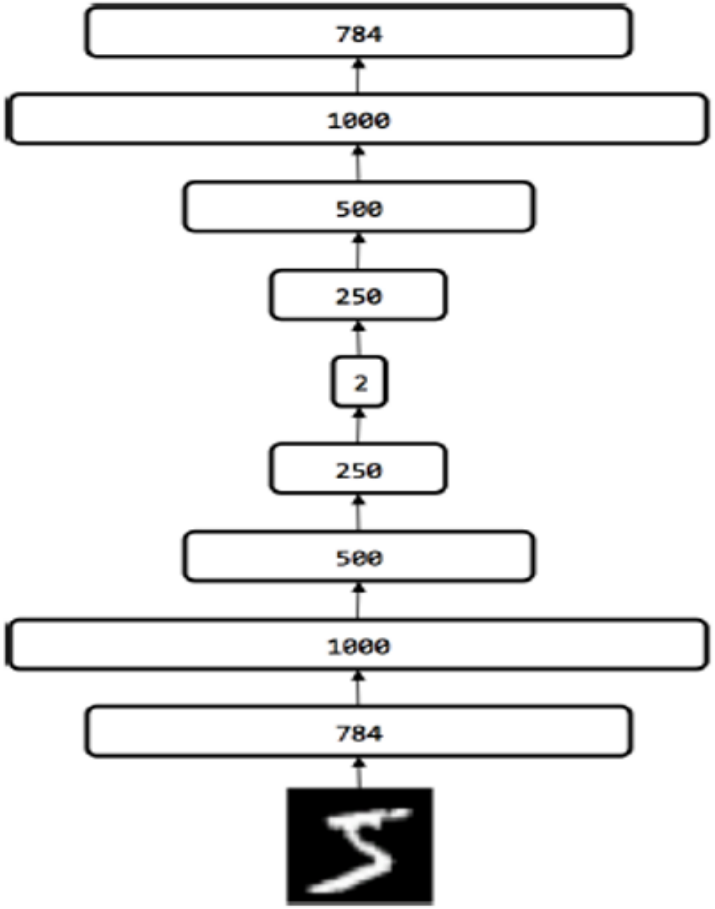


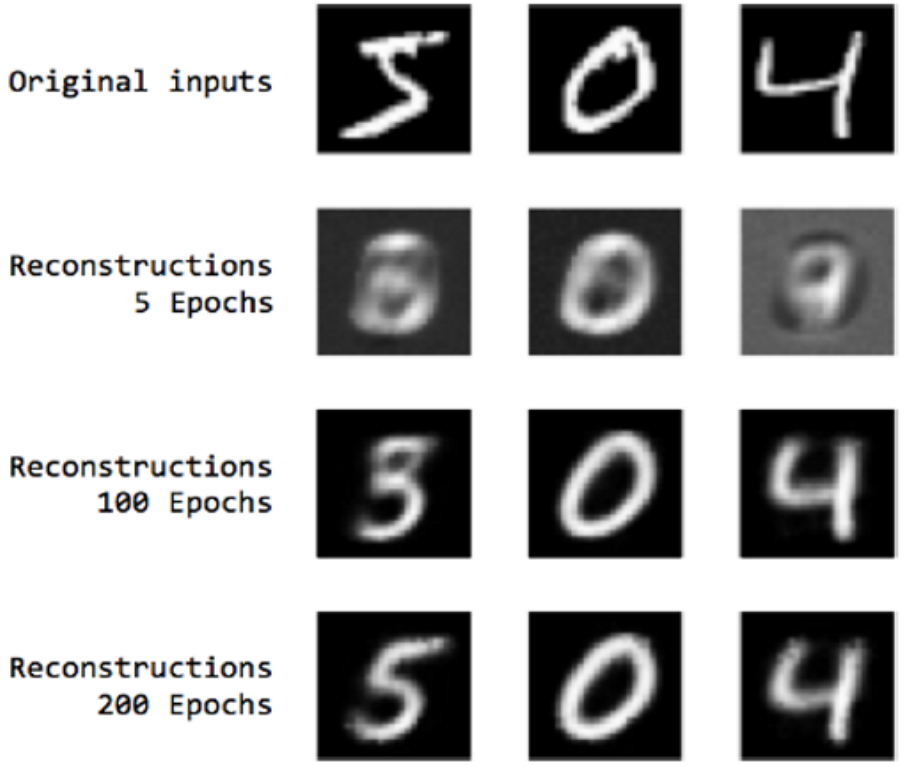
Word Represtations:Distributional similarity based:Autoencoder Inspiration

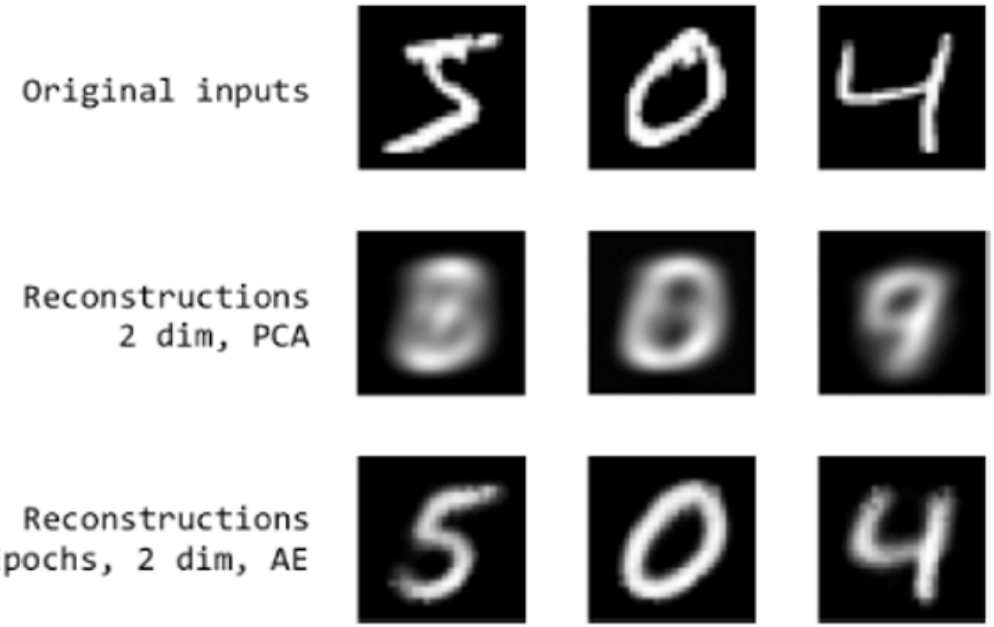




- The inspiration for embedded representation came from auto-encoder

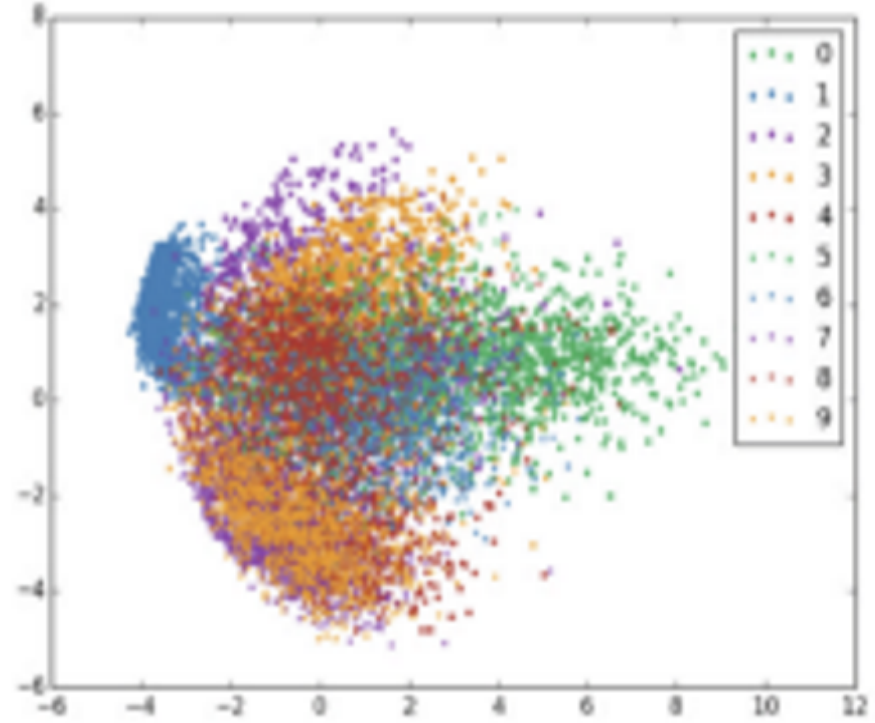




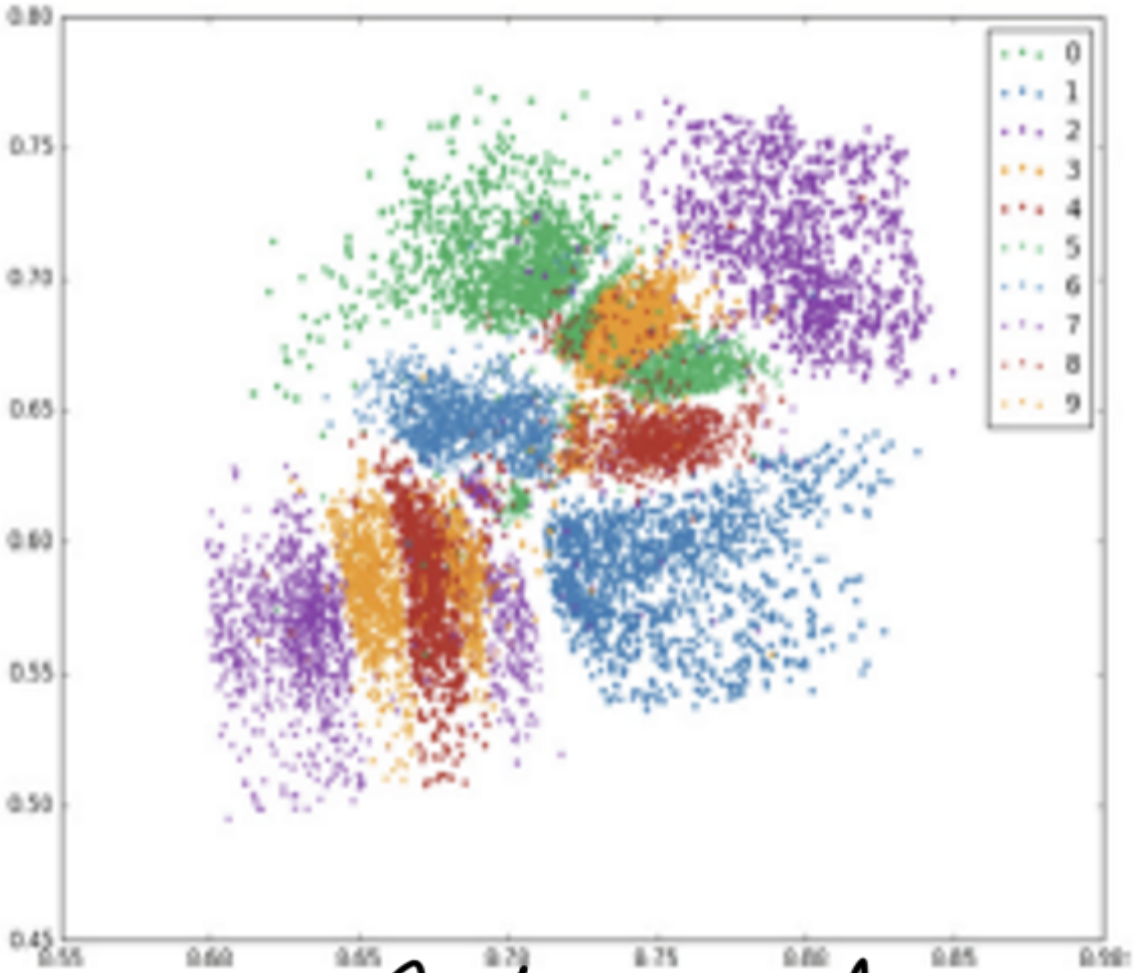


. There is only one winner here.

Word Represtations:Distributional similarity based:Autoencoder Inspiration



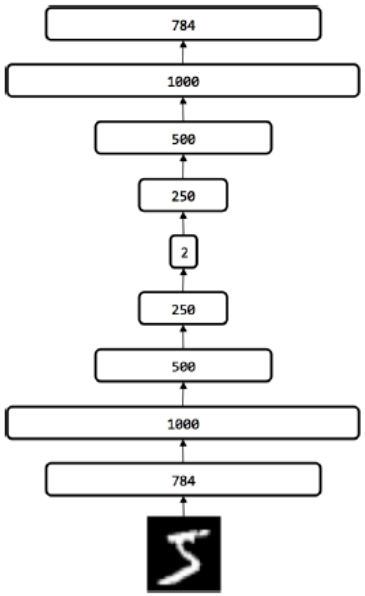
PCA



Autoencoder

- Visualization of 2 dimension encoding done by PCA and auto encoder.

Word Representations:Distributional similarity based:Autoencoder



- Except the name everything is identical about the decodee.

```python
# Architecture
n_encoder_hidden_1 = 1000
n_encoder_hidden_2 = 500
n_encoder_hidden_3 = 250
n_decoder_hidden_1 = 250
n_decoder_hidden_2 = 500
n_decoder_hidden_3 = 1000

# Parameters
learning_rate = 0.01
training_epochs = 1000
batch_size = 100
display_step = 1

def layer(input, weight_shape, bias_shape, phase_train):
    weight_init = tf.random_normal_initializer(stddev=(1.0/weight_shape[0])**0.5)
    bias_init = tf.constant_initializer(value=0)
    W = tf.get_variable("W", weight_shape,
                        initializer=weight_init)
    b = tf.get_variable("b", bias_shape,
                        initializer=bias_init)
    logits = tf.matmul(input, W) + b
    return tf.nn.sigmoid(layer_batch_norm(logits, weight_shape[1], phase_train))

def encoder(x, n_code, phase_train):
    with tf.variable_scope("encoder"):
        with tf.variable_scope("hidden_1"):
            hidden_1 = layer(x, [784, n_encoder_hidden_1], [n_encoder_hidden_1], phase_train)

        with tf.variable_scope("hidden_2"):
            hidden_2 = layer(hidden_1, [n_encoder_hidden_1, n_encoder_hidden_2], [n_encoder_hidden_2], phase_train)

        with tf.variable_scope("hidden_3"):
            hidden_3 = layer(hidden_2, [n_encoder_hidden_2, n_encoder_hidden_3], [n_encoder_hidden_3], phase_train)

        with tf.variable_scope("code"):
            code = layer(hidden_3, [n_encoder_hidden_3, n_code], [n_code], phase_train)

    return code
```

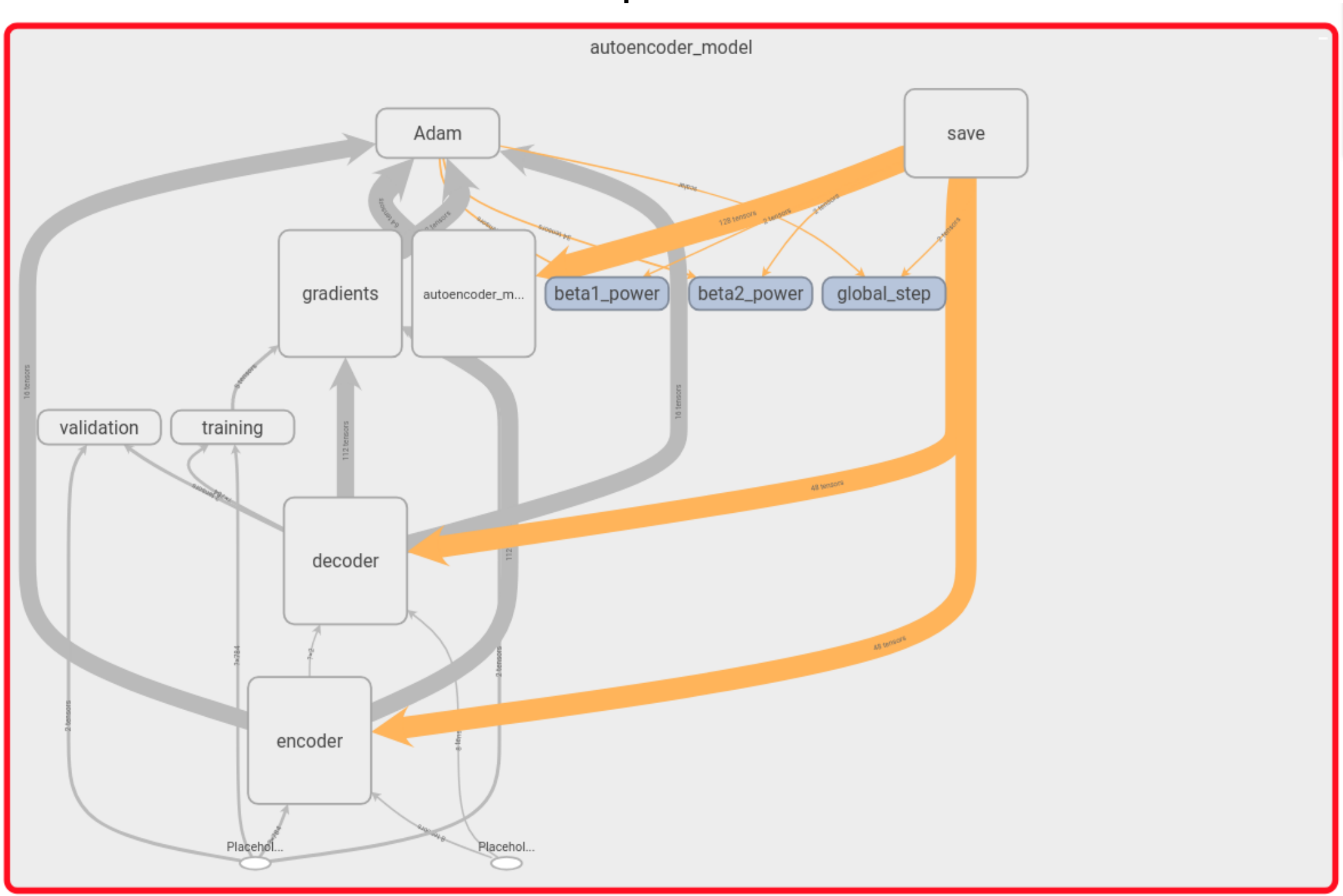
Word Represtations:Distributional similarity based:Autoencoder

```
def loss(output, x):
    with tf.variable_scope("training"):
        l2 = tf.sqrt(tf.reduce_sum(tf.square(tf.subtract(output, x)), 1))
        train_loss = tf.reduce_mean(l2)
        train_summary_op = tf.summary.scalar("train_cost", train_loss)
        return train_loss, train_summary_op

def training(cost, global_step):
    optimizer = tf.train.AdamOptimizer(learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1
        use_locking=False, name='Adam')
    train_op = optimizer.minimize(cost, global_step=global_step)
    return train_op

def image_summary(label, tensor):
    tensor_reshaped = tf.reshape(tensor, [-1, 28, 28, 1])
    return tf.summary.image(label, tensor_reshaped)

def evaluate(output, x):
    with tf.variable_scope("validation"):
        in_im_op = image_summary("input_image", x)
        out_im_op = image_summary("output_image", output)
        l2 = tf.sqrt(tf.reduce_sum(tf.square(tf.subtract(output, x, name="val_diff")), 1))
        val_loss = tf.reduce_mean(l2)
        val_summary_op = tf.summary.scalar("val_cost", val_loss)
        return val_loss, in_im_op, out_im_op, val_summary_op
```

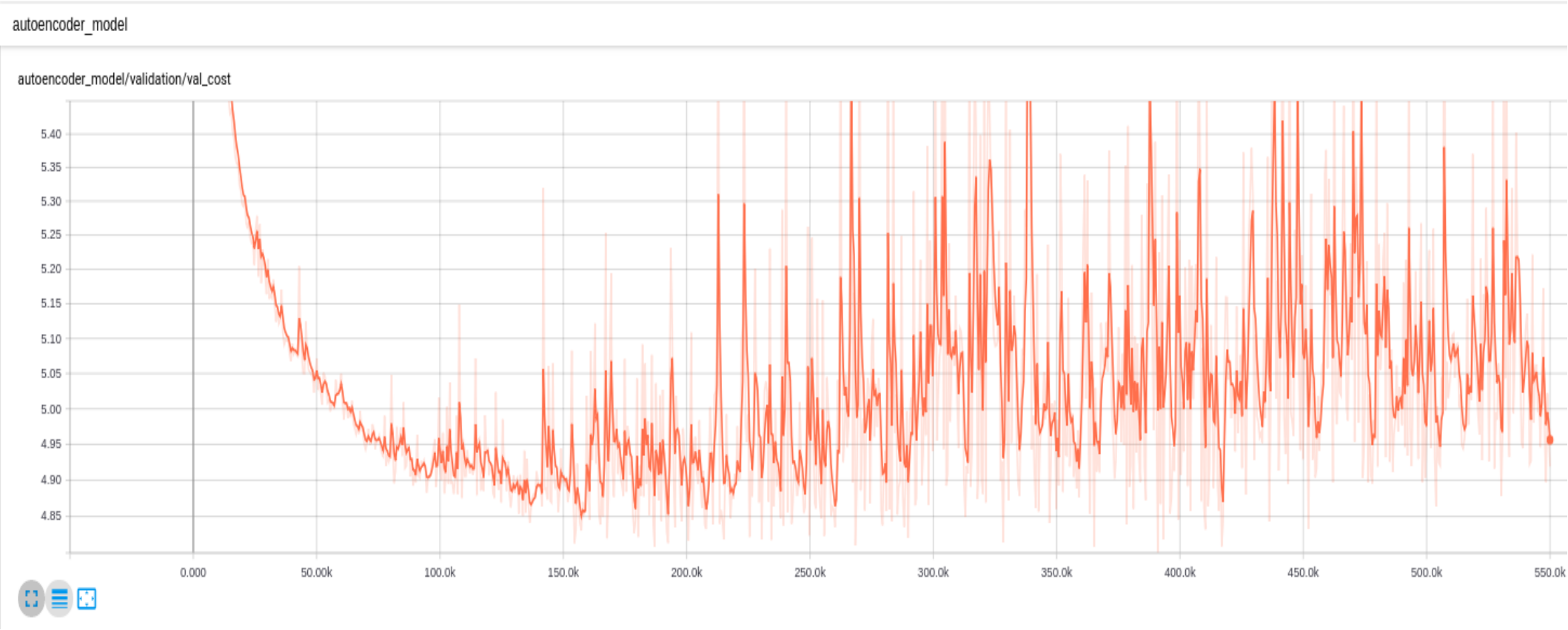
- The main training loop.

```
with tf.variable_scope("autoencoder_model"):

    x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=
    phase_train = tf.placeholder(tf.bool)
    code = encoder(x, int(n_code), phase_train)
    output = decoder(code, int(n_code), phase_train)
    cost, train_summary_op = loss(output, x)
    global_step = tf.Variable(0, name='global_step', trainable=False)
    train_op = training(cost, global_step)
    eval_op, in_im_op, out_im_op, val_summary_op = evaluate(output, x)
    summary_op = tf.summary.merge_all()
    saver = tf.train.Saver(max_to_keep=200)
    sess = tf.Session()
    train_writer = tf.summary.FileWriter(utils.get_project_data_path()+"mnist_a
                                         graph=sess.graph)
    val_writer = tf.summary.FileWriter(utils.get_project_data_path()+"mnist_au
                                         graph=sess.graph)
    init_op = tf.initialize_all_variables()
    sess.run(init_op)
    # Training cycle
    print("mnist.train.num_examples:",mnist.train.num_examples," batch_size:",
    for epoch in range(training_epochs):

        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            minibatch_x, minibatch_y = mnist.train.next_batch(batch_size)
            # Fit training using batch data
            _, new_cost, train_summary = sess.run([train_op, cost, train_summa
            train_writer.add_summary(train_summary, sess.run(global_step))
            # Compute average loss
            avg_cost += new_cost/total_batch
```

Word Represtations:Distributional similarity based:Autoencoder Inspiration



Word Represtations:Distributional similarity based:Autoencoder Inspiration



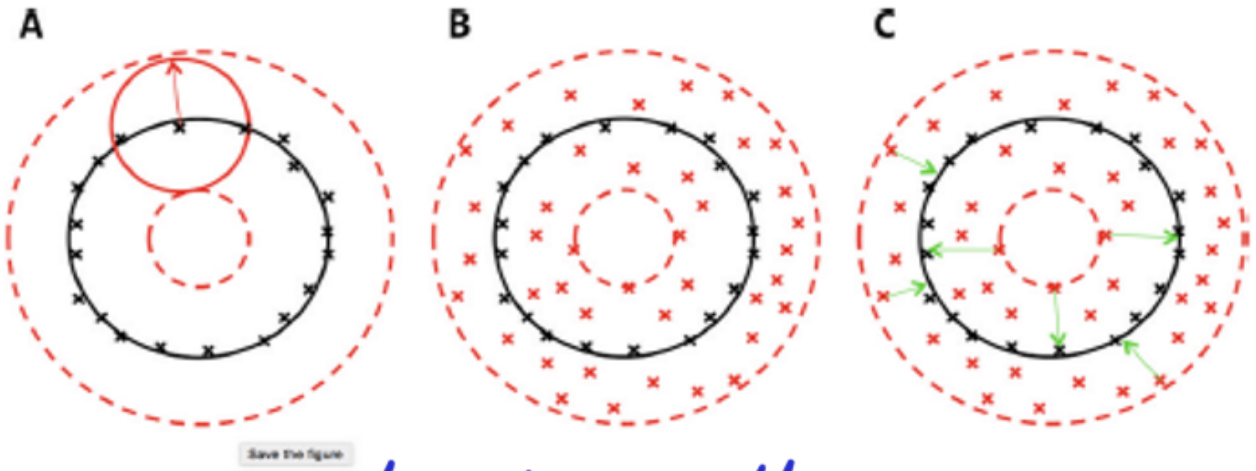
Word Represtations:Distributional similarity based:Autoencoder

Original images

Corrupted images.

- Even if were exposed to a random sampling of pixels from an image, the brain is capable of concluding the ground truth.

Word Representations:Distributional similarity based:Autoencoder Inspiration:Denoising



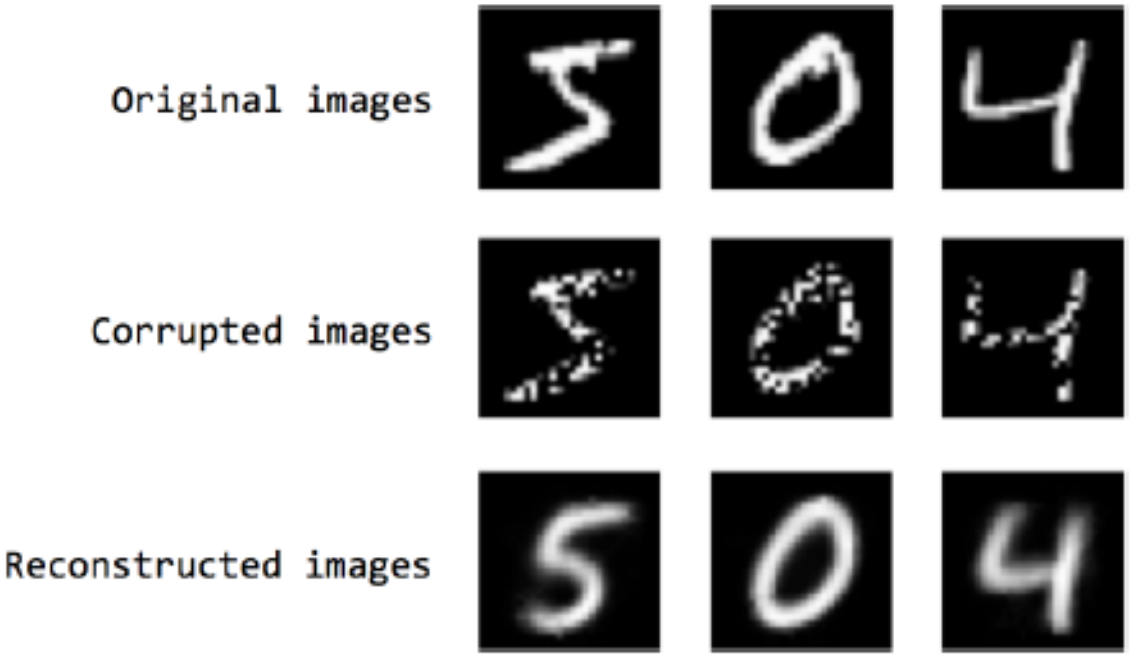
- Manifold is the shape we want to capture when we reduce the dimensions of data.
- The denoising objective enables our model to learn the manifold (black circle) by learning to map corrupted data (black xs) by minimizing the error (green arrows) between their representation.

Word Represtations:Distributional similarity based:Autoencoder Inspiration:Denoising

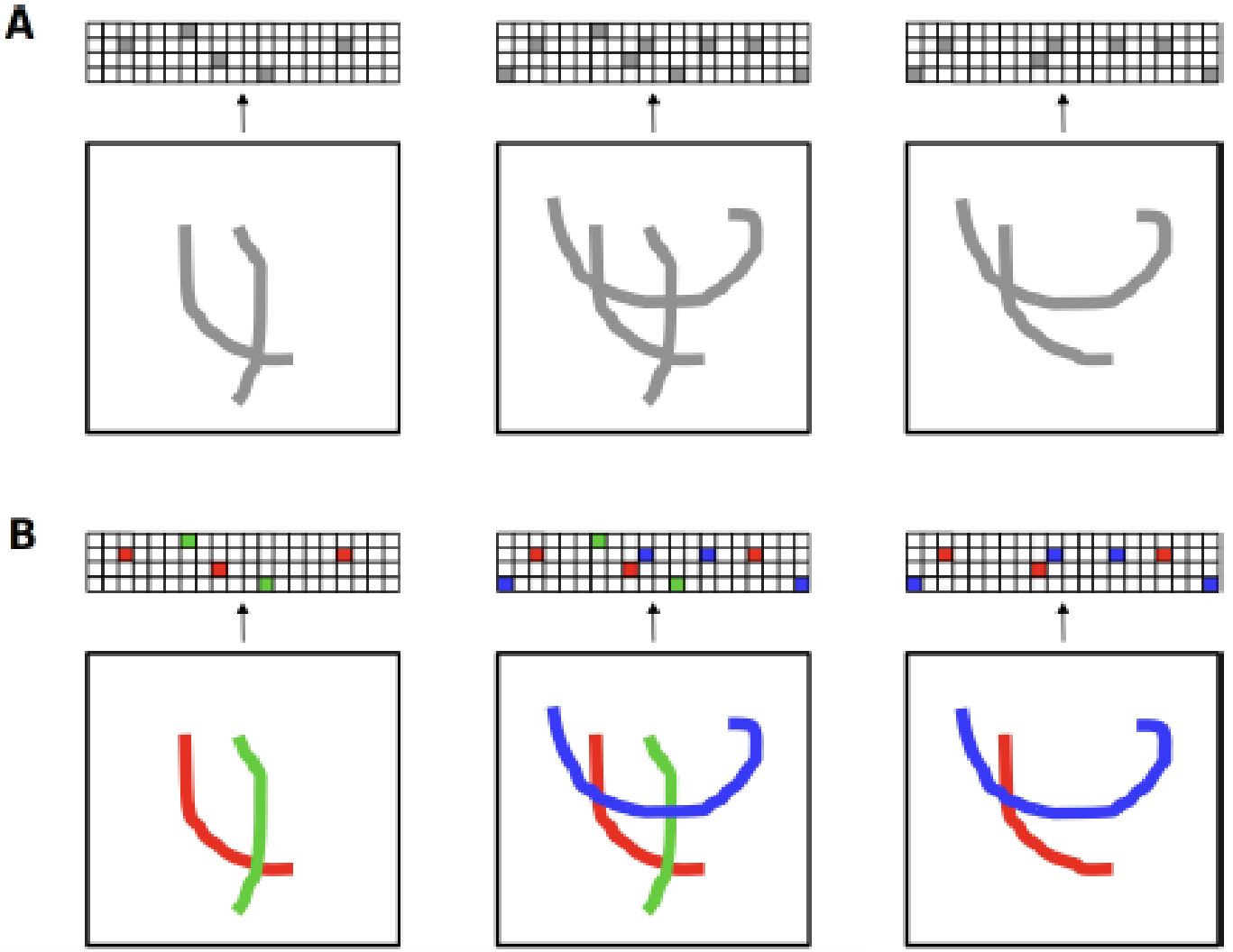
```
def corrupt_input(x):
    corrupting_matrix = tf.random_uniform(shape=tf.shape(x), minval=0,maxval=2,dtype=tf.int32)
    return x * tf.cast(corrupting_matrix, tf.float32)

    with tf.Graph().as_default():

        with tf.variable_scope("autoencoder_model"):

            x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
            corrupt = tf.placeholder(tf.float32)
            phase_train = tf.placeholder(tf.bool)
            c_x = (corrupt_input(x) * corrupt) + (x * (1 - corrupt))
            code = encoder(c_x, int(n_code), phase_train)
            output = decoder(code, int(n_code), phase_train)
            cost, train_summary_op = loss(output, x)
            for epoch in range(training_epochs):

                avg_cost = 0.
                total_batch = int(mnist.train.num_examples/batch_size)
                # Loop over all batches
                for i in range(total_batch):
                    minibatch_x, minibatch_y = mnist.train.next_batch(batch_size)
                    # Fit training using batch data
                    _, new_cost, train_summary = sess.run([train_op, cost, train_summary_op], feed_dict={x: minibatch_x, phase_train: True, corrupt: 1})
                    train_writer.add_summary(train_summary, sess.run(global_step))
```

- Corrupts the input if the place holder is equal to 1.

Word Represtations:Distributional similarity based:Autoencoder Inspiration:Denoising

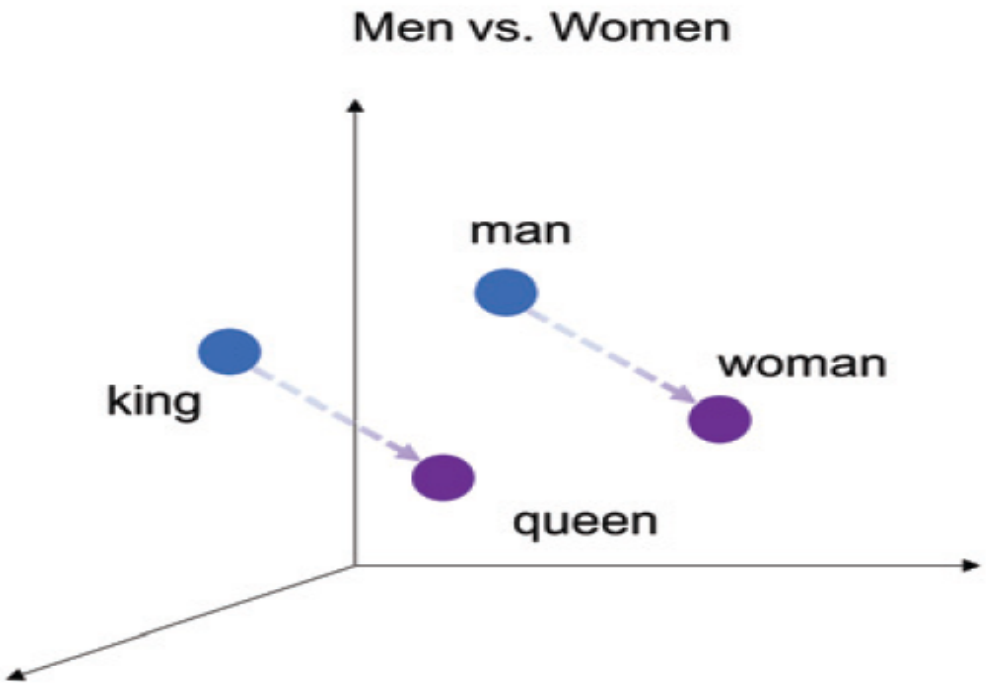


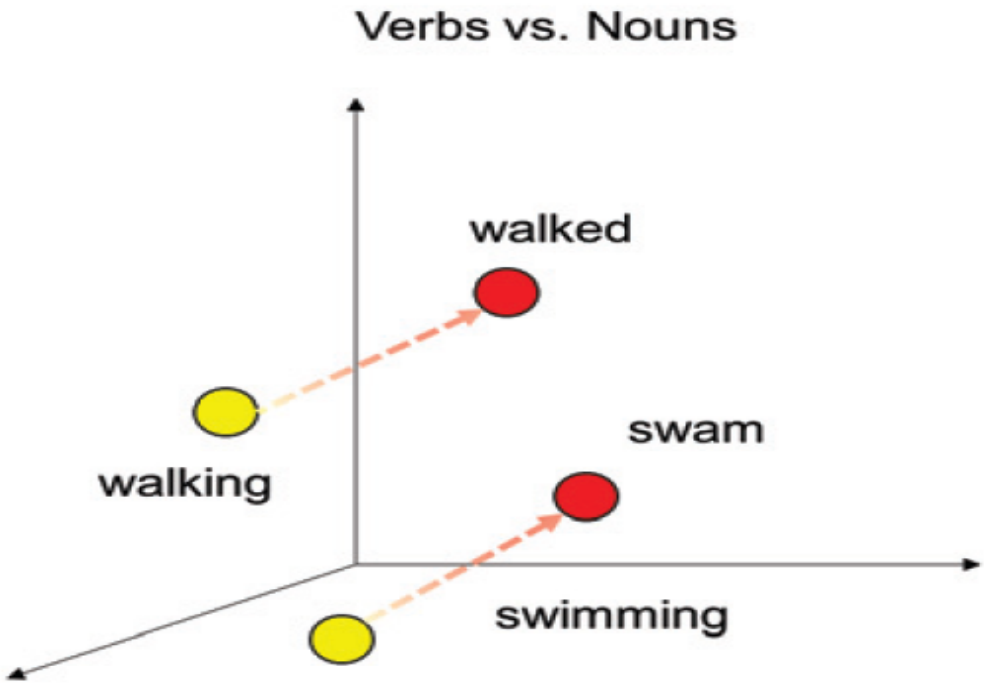
Word Represtations:Distributional similarity based:Autoencoder Inspiration:Sparsity



$$E_{\text{Sparse}} = E + \beta \cdot \text{SparsityPenalty}$$

Word Embedding:Distributed Representation





- Learn word embeddings have got some interesting properties
  - First, the location of word representation in the high dimensional space is determined by the meaning
  - Second, the word vectors have linear relationships. the relationship between words can be thought as the direction in the distance formed by a pair of words. for example, starting from the location of the word king, move the same distance in the direction between man and women, and one will get the word queen

$$[king] - [man] + [woman] \sim = [queen]$$

Word Embeddings:Distributed Representation

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Word Embeddings:Distributed Representation:Advantages

- Capturing local co-occurrence statistics
- Produces state-of-the-art linear semantic relationships
- Efficient use of statistics
- Can train on (comparably) little data and gigantic data
- Fast, only non-zero counts matter
- Good performance with small (100-300) dimension vectors that are important for downstream tasks

Word Embeddings:Distributed Representation:Disadvantages

- **Similarity and relatedness are not the same:**
- **Word ambiguity:**

Word Embeddings:Distributed Representation:Word2vec

Two algorithms

1. **Skip-grams (SG)**

   Predict context words given target (position independent)

2. Continuous Bag of Words (CBOW)

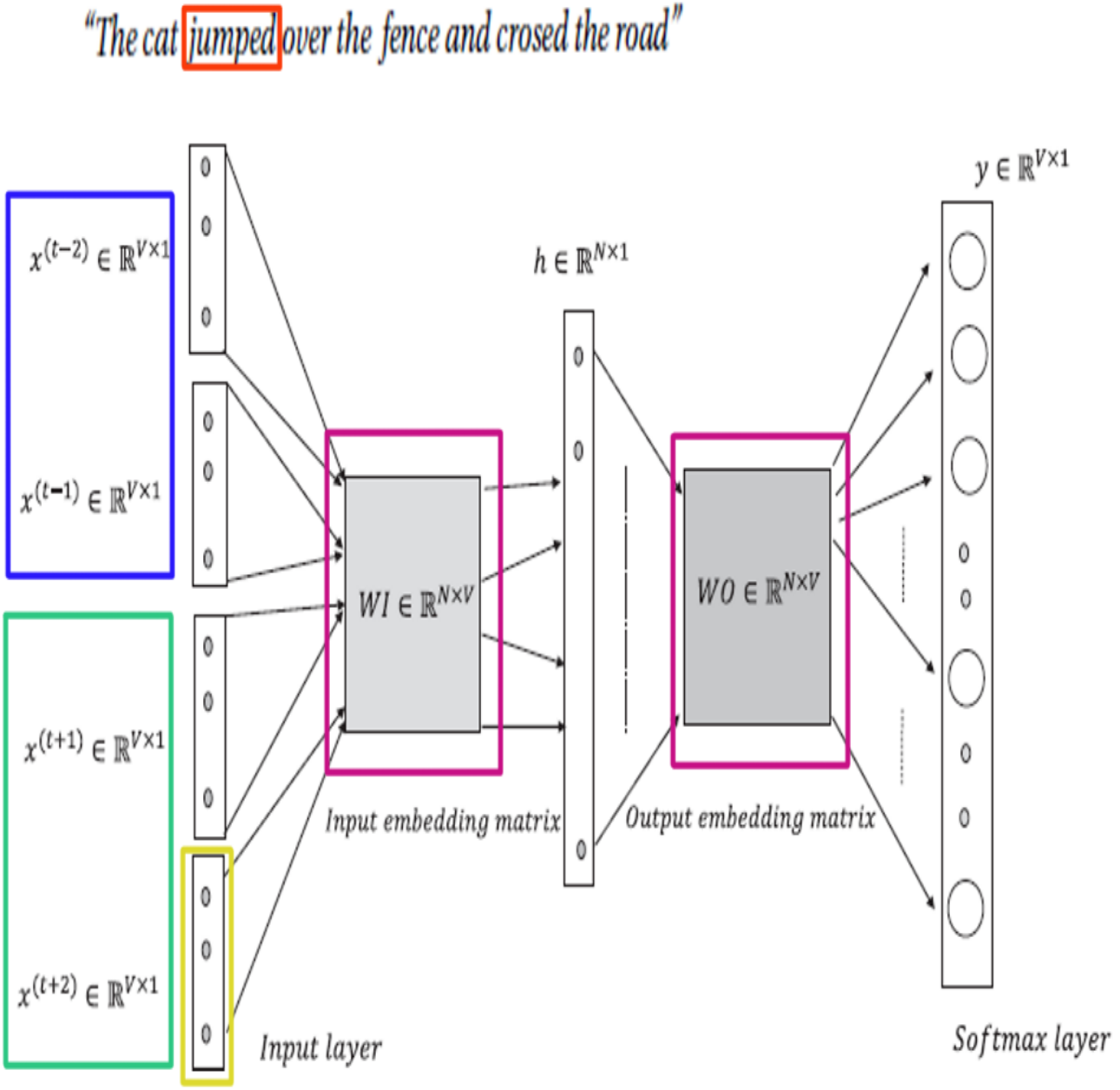
   Predict target word from bag-of-words context

Two (moderately efficient) training methods

1. Hierarchical softmax
2. Negative sampling

Naïve softmax

Word Embeddings:Distributed Representation:Word2vec:CBOW



- Word to predict based on surrounding words
- 2 before and 2 after
- One hot vector
- Shape of embedding matrix is N×V, where N is the number of dimensions, & V is the vocabulary size.

Word Embeddings:Distributed Representation:Word2vec:CBOW

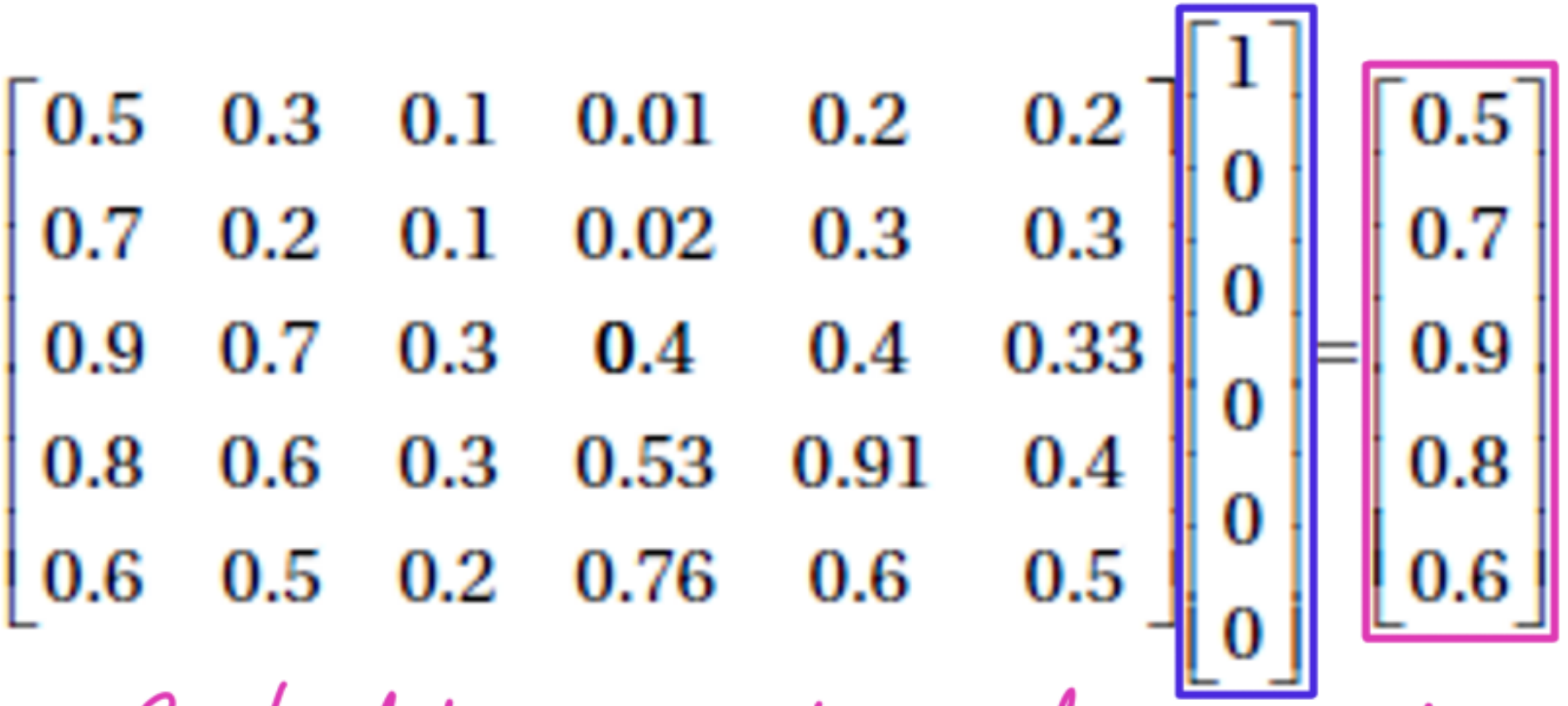
$$x_{cat} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x_{rat} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x_{chased} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x_{garden} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad x_{the} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad x_{was} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$N = 5, \quad V = 6$

$$WI = \begin{array}{c} \begin{matrix} cat & rat & chased & garden & the & was \end{matrix} \\ \begin{bmatrix} 0.5 & 0.3 & 0.1 & 0.01 & 0.2 & 0.2 \\ 0.7 & 0.2 & 0.1 & 0.02 & 0.3 & 0.3 \\ 0.9 & 0.7 & 0.3 & 0.4 & 0.4 & 0.33 \\ 0.8 & 0.6 & 0.3 & 0.53 & 0.91 & 0.4 \\ 0.6 & 0.5 & 0.2 & 0.76 & 0.6 & 0.5 \end{bmatrix} \end{array}$$

- Embedding matrix learnt through training.

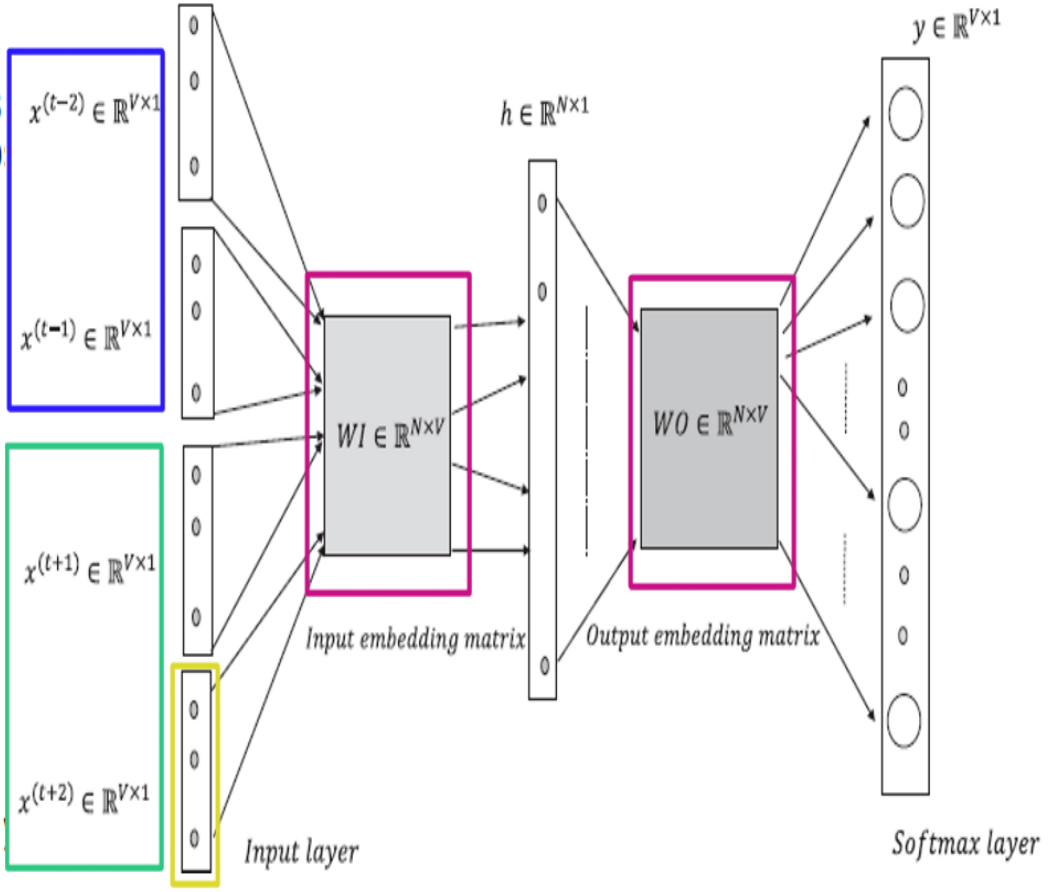
Word Embeddings:Distributed Representation:Word2vec:CBOW



- Embedding matrix for word cat which is the first word in the one hot index.

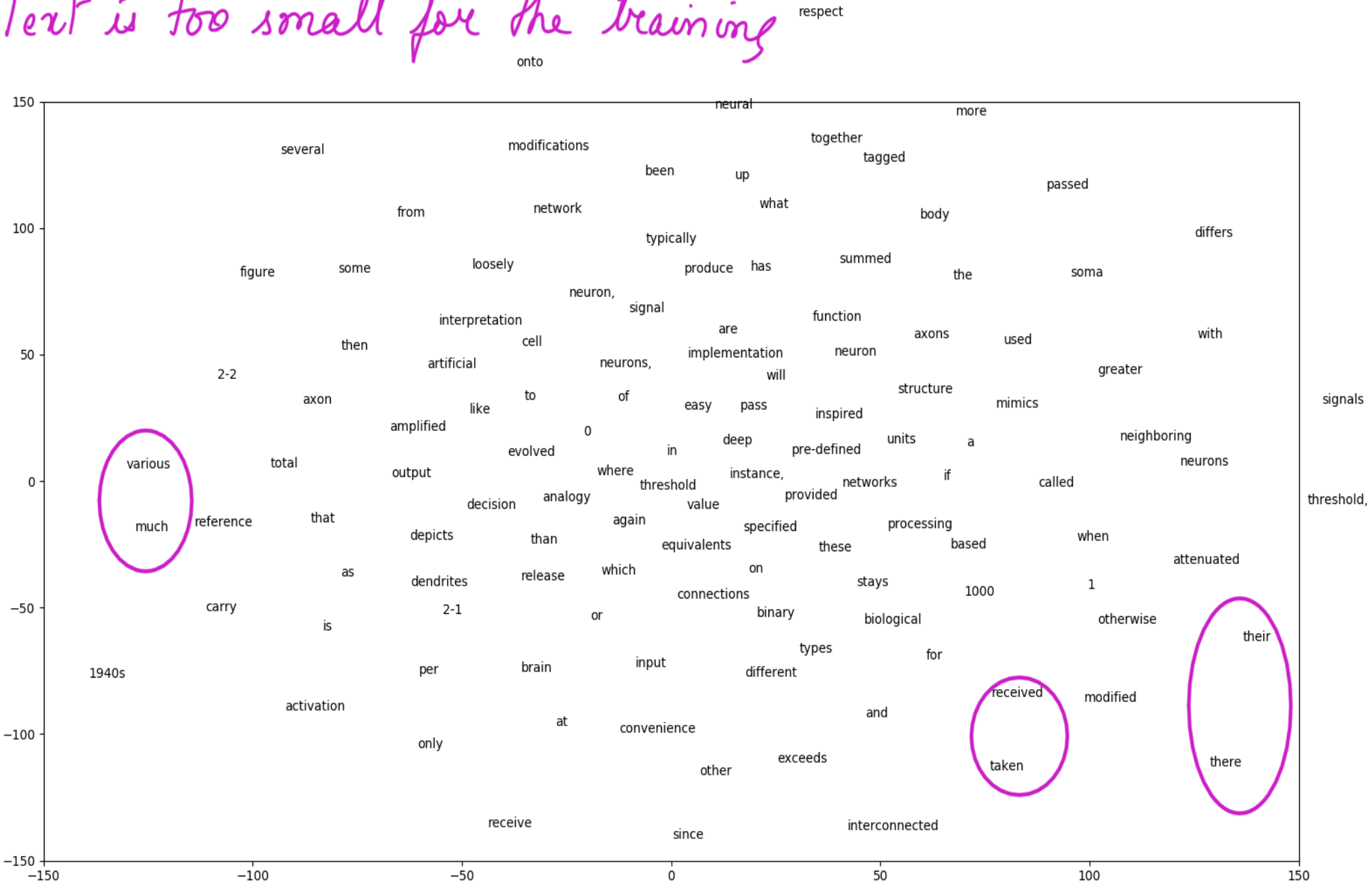
Word Embeddings:Distributed Representation:Word2vec:CBOW

```
W = tf.Variable(tf.random_normal([vocab_size, emb_dims], mean=0.0, stddev=0.02, dtype=tf.float32))
b = tf.Variable(tf.random_normal([emb_dims], mean=0.0, stddev=0.02, dtype=tf.float32))
W_outer = tf.Variable(tf.random_normal([emb_dims, vocab_size], mean=0.0, stddev=0.02, dtype=tf.float32))
b_outer = tf.Variable(tf.random_normal([vocab_size], mean=0.0, stddev=0.02, dtype=tf.float32))

hidden = tf.add(tf.matmul(x, W), b)
logits = tf.add(tf.matmul(hidden, W_outer), b_outer)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(co

epochs, batch_size = 100, 10
batch = len(x_train) // batch_size

# train for n_iter iterations
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for epoch in range(epochs):
        batch_index = 0
        for batch_num in range(batch):
            x_batch = x_train[batch_index: batch_index + batch_size]
            y_batch = y_train[batch_index: batch_index + batch_size]
            sess.run(optimizer, feed_dict={x: x_batch, y: y_batch})
    print('epoch:', epoch, 'loss :', sess.run(cost, feed_dict={x: x_batch,
    W_embed_trained = sess.run(W)
```

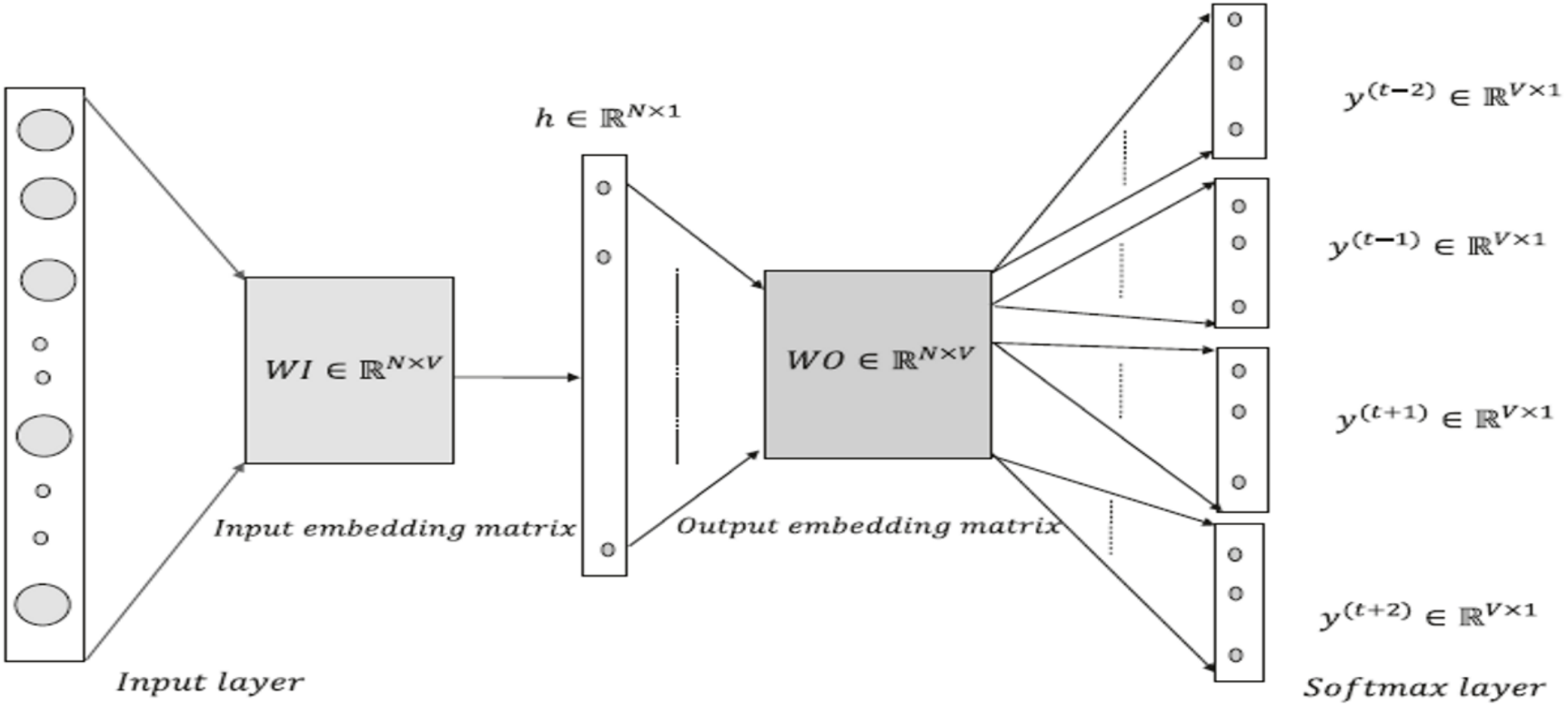


Word Embeddings:Distributed Representation:Word2vec:CBOW

- Text is too small for the training



Word Embeddings:Distributed Representation:Word2vec:Skipgram



- Skip-gram models work the other way around. Instead of trying to predict the current word from the context words, as in Continuous Bag of Words, in Skip-gram models the context words are predicted based on the current word.
- Generally, given a current word, context words are taken in its neighborhood in each window.
- For a given window of five words there would be four context words that one needs to predict based on the current word.

Word Embeddings:Distributed Representation:Word2vec:Skipgram:tensorflow

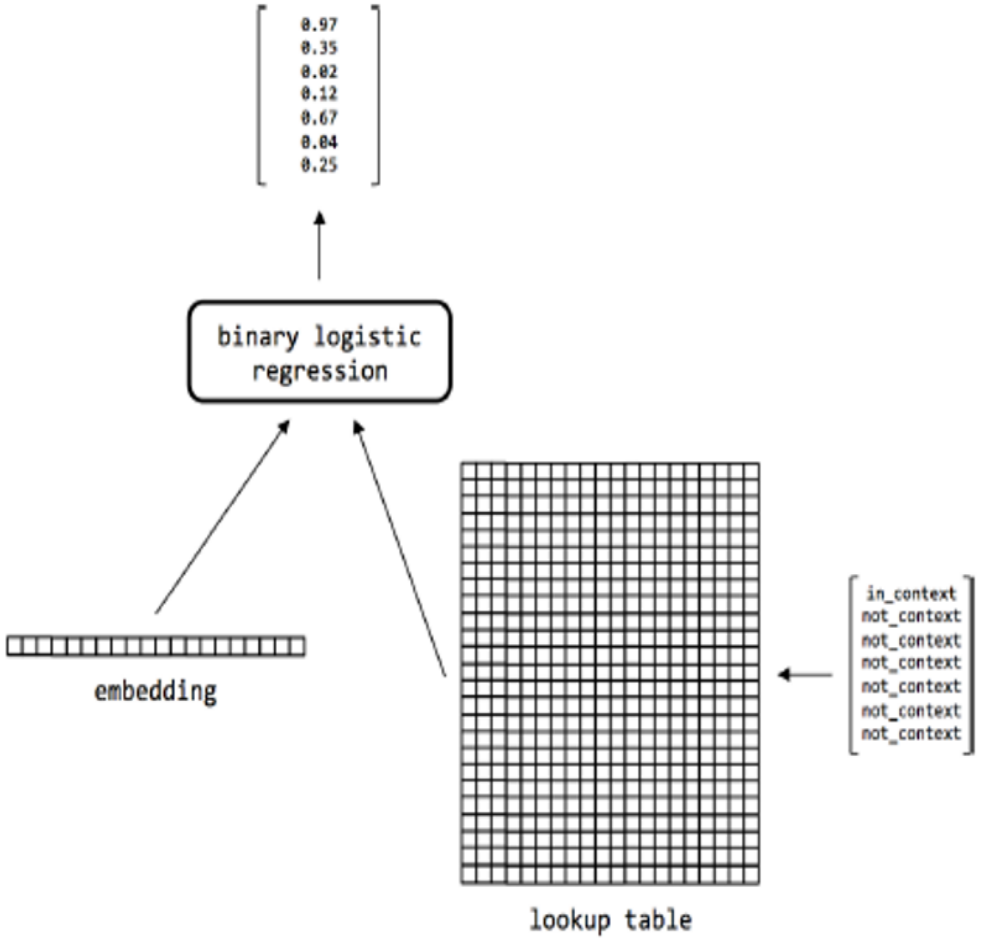
```
# Ops and variables pinned to the CPU because of missing GPU implementation
with tf.device('/device:CPU:0'):
    # Look up embeddings for inputs.
    embeddings = tf.Variable(
        tf.random_uniform([vocabulary_size, embedding_size], -1.0, 1.0),name ="W")
    embed = tf.nn.embedding_lookup(embeddings, train_inputs)

    # Construct the variables for the NCE loss
    nce_weights = tf.Variable(
        tf.truncated_normal([vocabulary_size, embedding_size],
                            stddev=1.0 / math.sqrt(embedding_size)))
    nce_biases = tf.Variable(tf.zeros([vocabulary_size]))
```

Training embedding layer is quite simple

- Initialize look up table.

Word Embeddings:Distributed Representation:Word2vec:Skipgram:tensorflow

```
# Compute the average NCE loss for the batch.
# tf.nce_loss automatically draws a new sample of the negative labels each
# time we evaluate the loss.
# Explanation of the meaning of NCE loss:
#   http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
loss = tf.reduce_mean(
    tf.nn.nce_loss(weights=nce_weights,
                   biases=nce_biases,
                   labels=train_labels,
                   inputs=embed,
                   num_sampled=num_sampled,
                   num_classes=vocabulary_size))

train_summary_op = tf.summary.scalar("train_cost", loss)
# Construct the SGD optimizer using a learning rate of 1.0.
optimizer = tf.train.GradientDescentOptimizer(1.0).minimize(loss)

# Compute the cosine similarity between minibatch examples and all embeddings.
norm = tf.sqrt(tf.reduce_sum(tf.square(embeddings), 1, keep_dims=True))
normalized_embeddings = embeddings / norm
valid_embeddings = tf.nn.embedding_lookup(
    normalized_embeddings, valid_dataset)
similarity = tf.matmul(
    valid_embeddings, normalized_embeddings, transpose_b=True)
```



- The naive way to construct the decoder would be to attend to reconstruct the one hot encoding
- this is highly inefficient
- Mikolov used a strategy for implementing the decoder known as noise contrasting estimation(NCE)

Word Embeddings:Distributed Representation:Word2vec:Skipgram:tensorflow

```
  # Note that this is expensive (~20% slowdown if computed every 5
  if step % 10000 == 0:
      sim = similarity.eval()
      for i in range(valid_size):
          valid_word = reverse_dictionary[valid_examples[i]]
          top_k = 8  # number of nearest neighbors
          nearest = (-sim[i, :]).argsort()[1:top_k + 1]
          log_str = 'Nearest to %s:' % valid_word
          for k in range(top_k):
              close_word = reverse_dictionary[nearest[k]]
              log_str = '%s %s,' % (log_str, close_word)
          print(log_str)
final_embeddings = normalized_embeddings.eval()
```

```
Average loss at step  0 :  249.41293334960938
Nearest to UNK: debbie, romagna, immortality, sawdust, cookie, kalmar, fargo,
Nearest to four: serapeum, strings, welling, limnic, empower, subtitle, wane,
Nearest to eight: engel, balancing, linguistic, phlogiston, interpreter, adolf
Nearest to into: depot, invention, minimise, bilge, affliction, hiroshima, gro
Nearest to for: mutable, finland, beeb, situ, shareholders, insertions, akron,
Nearest to people: decrypted, roaring, frud, ewen, suspension, cooper, portill
Nearest to an: hula, zuid, sales, xj, lew, serra, rana, cva,
Nearest to were: faithfully, rawlinson, revolutionize, dwelt, explainable, com
Nearest to called: complement, metered, eat, adjectival, developed, anybody, c
Nearest to world: percy, boitano, bump, gratis, revelation, argo, stele, servi
Nearest to no: audiobook, obsoletes, sugarcane, behaved, hair, clutching, inte
Nearest to with: cooperstown, mechanics, babel, sarcophagi, scissors, allowed,
Nearest to five: gendered, luis, watcher, trill, higgins, yury, detonation, ab
Nearest to may: legalised, faroe, horrifying, jacobson, seam, layers, lamarcki
Nearest to six: comprises, inch, candles, cats, scrolls, bestow, rationalist,
Nearest to on: derives, pulley, chuck, convenience, assist, mcluhan, morbidity
```

- Better representation as training progresses.

```
Average loss at step  100000 :  4.707459555625915
Nearest to UNK: ursus, agouti, dasyprocta, cebus, circ, michelob, kapoor, four,
Nearest to four: five, seven, six, three, eight, two, zero, nine,
Nearest to eight: seven, nine, six, five, four, three, zero, ursus,
Nearest to into: from, during, through, out, depot, projecting, on, invention,
Nearest to for: ursus, operatorname, in, of, with, kapoor, microcebus, or,
Nearest to people: dasyprocta, suspension, roaring, economies, busan, louvre, archie,
Nearest to an: dddddd, operators, ss, northumbrian, inducing, zuid, shaking, clashed,
Nearest to were: are, was, have, had, is, be, being, include,
Nearest to called: and, metered, gardening, ursus, anybody, clo, bouvet, canaan,
Nearest to world: agnosticism, post, bringing, speedup, boitano, revelation, primige
Nearest to no: any, christianum, rationing, dmd, daphne, innings, dinar, a,
Nearest to with: between, in, by, operatorname, including, when, cebus, for,
Nearest to five: four, seven, six, three, eight, two, zero, nine,
Nearest to may: can, would, will, could, might, should, must, sarcoma,
```

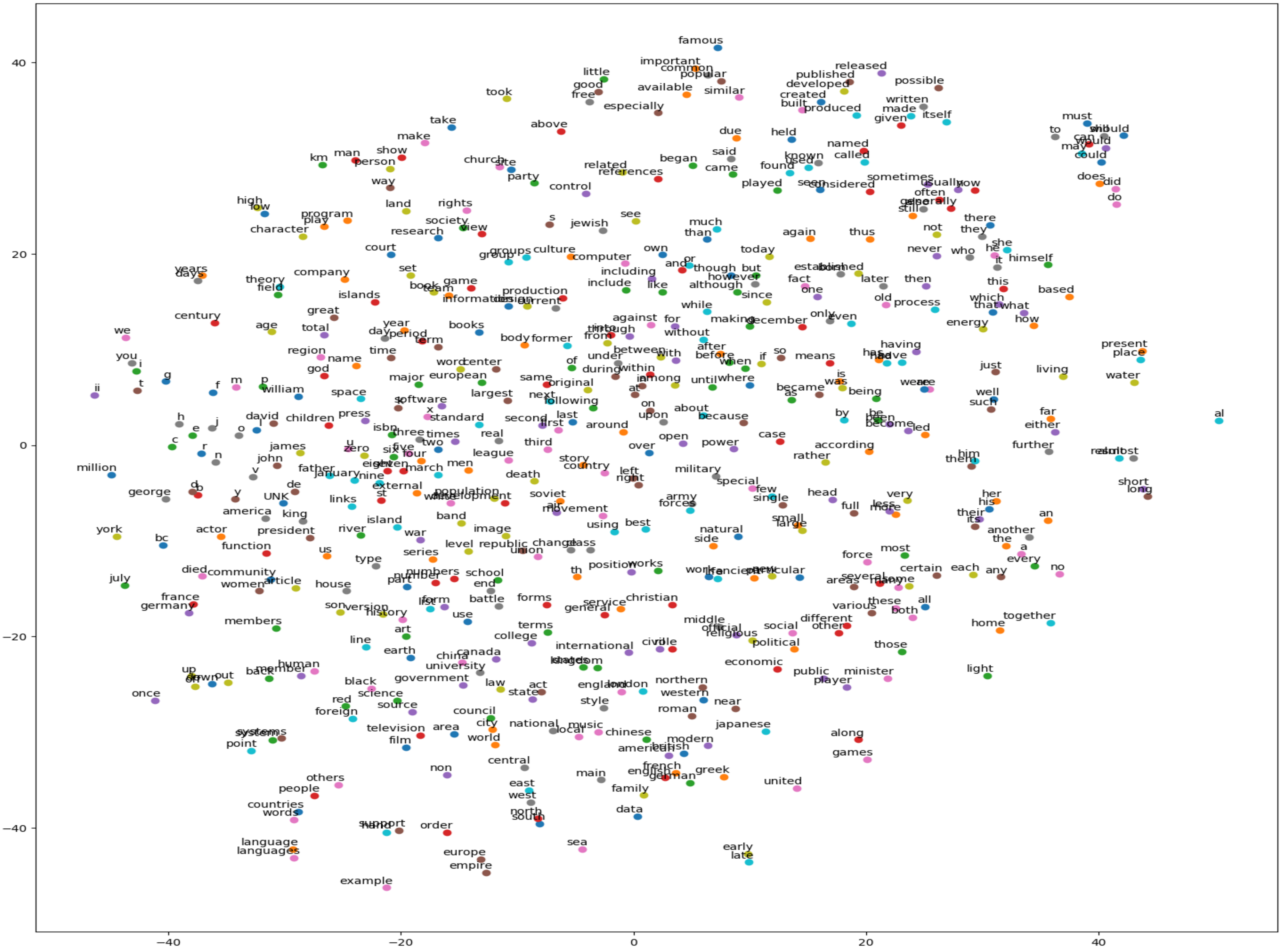
Word Embeddings:Distributed Representation:Word2vec:Skipgram:tensorflow

```
def write_embeddings(filename="word2vec_tf_embedding.txt"):
    with open(utils.get_embeddings_path()+filename, 'w') as file_:
        file_.write('%s %s\n' % (vocabulary_size, embedding_size))
        for i in range(vocabulary_size):
            embed = final_embeddings[i, :]
            word = reverse_dictionary[i]
            file_.write('%s %s\n' % (word, ' '.join(map(str, embed))))
```

```
1 50000 128
2 UNK 0.14411964 0.038926728 0.0468082
  0.11361387 0.029596144 -0.006695945
  0.14622778 0.054516587 0.071768954 0
  -0.10669074 0.10041072 0.05558411 -0
  -0.07785477 -0.037994776 0.030469757
  -0.018863345 -0.08392478 -0.07927676
  0.09733617 0.15191945 0.20354484 0.1
  -0.010660334 0.020415287 0.035158467
3 the -0.0931185 2.0003445e-05 -0.0223
  0.055386644 -0.06750202 0.002192009
  0.008588252 0.093157694 0.005710450
  -0.12631446 -0.0025746326 0.13330321
  -0.18590038 0.016289629 -0.046267584
  -0.0039573973 0.043337774 -0.023271
  0.14434543 0.022243379 0.08256889 0.
  0.065142825 0.069952704 -0.06197627
4 of -0.1756537 -0.09205165 -0.153466
  -0.09841012 0.0056424583 -0.0673455
  0.096074775 0.12766993 0.04939218 0.
  -0.054601394 -0.07466 0.20192826 -0.
  -0.023379844 -0.0786018 0.06471708 -
  -0.098518655 -0.031796243 -0.100585
  0.07133745 0.09354205 0.19651748 -0.
  0.13939632 -0.087714225 0.19580626 0
5 and -0.095537096 -0.042545207 0.0094
```

$$[0\,0\,0\,\ldots\,0\,1\,0\,0] \times \begin{bmatrix} 14 & 22 & 3 & \ldots & 13 & 22 & 27 & 22 \\ 14 & 11 & 6 & \ldots & 28 & 12 & 25 & 25 \\ 4 & 8 & 22 & \ldots & 11 & 24 & 24 & 12 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 18 & 26 & 1 & \ldots & 11 & 8 & 25 & 19 \\ 25 & 24 & 17 & \ldots & 9 & 8 & 14 & 3 \\ 26 & 0 & 18 & \ldots & 17 & 12 & 16 & 18 \\ 16 & 22 & 3 & \ldots & 4 & 18 & 17 & 8 \end{bmatrix} = [25 \quad 24 \quad 17 \quad \ldots \quad 9 \quad 8 \quad 14 \quad 3]$$

- Text form of embeddings. written out to a file
- Words
- : learnt embeddings.


40
20
0
−20
−40
−40
−20
0
20
40
famous
important
common
popular
little
good
free
available
similar
especially
released
published
developed
created
built
produced
possible
written
made
given
itself
took
above
take
make
due
held
named
said
called
came
found
known
used
man
show
person
km
church
site
party
related
references
began
played
control
seen
considered
sometimes
usually
often
generally
still
must
to
should
can
would
may
could
does
did
do
way
high
low
program
play
character
land
rights
society
view
research
s
jewish
see
much
than
again
thus
not
there
they
court
groups
group
culture
computer
own
or
and
today
never
who
he
she
it
himself
years
days
theory
field
company
set
book
game
team
production
information
design
current
including
include
like
though
but
however
although
since
established
fact
one
later
then
this
based
islands
great
century
age
total
year
day
period
term
books
body
former
against
for
into
through
from
without
making
december
only
even
old
process
which
that
what
energy
how
we
region
name
time
god
word
center
between
with
after
before
under
of
during
within
when
if
so
means
has
had
have
having
just
present
place
living
water
you
i
g
t
m
p
f
william
major
european
same
original
among
until
where
became
as
is
was
being
were
are
well
such
ii
space
largest
next
following
at
software
x
standard
second
first
last
on
upon
about
because
by
be
been
become
led
far
either
al
h
e
j
o
l
david
children
press
isbn
three
times
real
around
case
open
c
r
n
u
zero
james
six
five
two
league
third
over
power
according
further
almost
million
john
father
january
nine
eight
seven
four
march
men
story
country
left
right
military
rather
him
them
d
b
de
external
st
population
development
death
special
few
single
head
very
short
long
george
y
UNK
links
soviet
air
movement
army
forces
less
more
full
her
his
their
its
an
america
king
island
band
using
best
small
large
another
the
a
york
bc
actor
president
river
war
image
natural
side
function
level
republic
change
class
union
most
force
us
series
type
died
community
numbers
number
school
works
work
position
th
life
france
financial
part
particular
new
several
certain
each
any
every
no
areas
many
same
july
women
article
house
end
battle
forms
christian
service
general
these
various
all
both
germany
son
version
history
list
form
use
members
middle
official
religious
different
other
social
together
home
terms
college
line
art
earth
international
civil
role
political
those
china
canada
university
united
kingdom
economic
up
human
member
back
out
down
off
once
black
government
law
act
state
england
london
northern
western
public
minister
player
light
science
red
foreign
source
style
near
council
roman
systems
system
point
television
area
city
national
local
music
chinese
japanese
along
film
world
american
british
modern
games
central
non
others
people
main
english
french
german
greek
united
east
west
family
countries
words
north
south
data
support
hand
order
language
languages
sea
early
late
europe
empire
example

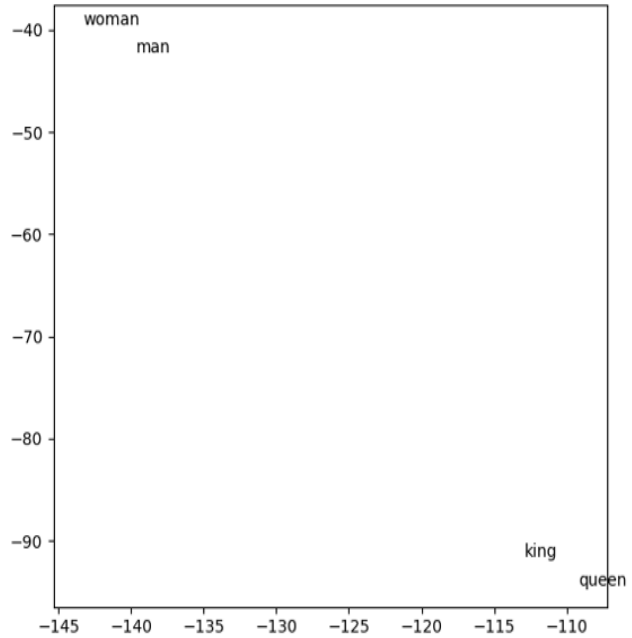
Word Embeddings:Distributed Representation:Word2vec:Skipgram:tensorflow

```
fasttextfile=utils.get_embeddings_path()+"wiki-news-300d-1M.vec"
glovefile=utils.get_embeddings_path()+"glove.6B.300d.txt"
word2vecfile=utils.get_embeddings_path()+"GoogleNews-vectors-negative300.bin"
customfile=utils.get_embeddings_path()+"word2vec_tf_embedding.txt"
if (len(sys.argv)==2):
    print(sys.argv)
    embedding=sys.argv[1]

model =None
print("embedding:",embedding)
if(embedding == 'fasttext'):
    print("fasttext")
    model = KeyedVectors.load_word2vec_format(
        fasttextfile, binary=False)
elif(embedding == 'glove'):
    print("glove")
    word2vec_output_file = 'glove.6B.300d.txt.word2vec'
    glove2word2vec(glovefile, word2vec_output_file)
    filename = 'glove.6B.300d.txt.word2vec'
    model = KeyedVectors.load_word2vec_format(filename, binary=False)
elif(embedding == 'word2vec'):
    print("word2vec")
    model = KeyedVectors.load_word2vec_format(
        word2vecfile, binary=True)
elif(embedding == 'custom'):
    """restoration works except the word sequence is no longer there.
    vocab_size=50000
    embedding_dim = 128
    embeddings = tf.Variable(tf.constant(0.0, shape=[vocab_size, embedding_dim]),
                     trainable=False)
    embedding_saver = tf.train.Saver({"W": embeddings})
    """
    print("custom")
    sess = tf.Session()
    model = KeyedVectors.load_word2vec_format(customfile, binary=False)
render_word_embedding(model)
```

- Pretrained word embedding.
- Custom one we trained a few slides back.
- Simple demonstration of various embeddings.

Word Embeddings:Distributed Representation:Word2vec:Skipgram:tensorflow

```
def render_word_embedding(model):
    print("Similar to run:", model.wv.most_similar(positive="run"))
    print("Similar to dirty:", model.wv.most_similar(positive="dirty"))
    print("Similar to polite:", model.wv.most_similar(positive="polite"))

    similar_words = model.wv.most_similar(
        positive=['woman', 'king'], negative=['man'], topn=5)
    print("'woman', 'king'-'man':", similar_words)
    print("dirty-dirty:", model.wv.similarity(w1="dirty", w2="dirty"))
    print("dirty-clean:", model.wv.similarity(w1="dirty", w2="clean"))

    king_wordvec = model['king']
    queen_wordvec = model['queen']
    man_wordvec = model['man']
    woman_wordvec = model['woman']

    pseudo_king = queen_wordvec - woman_wordvec + man_wordvec
    cosine_simi = np.dot(pseudo_king / np.linalg.norm(pseudo_king), king_wordvec / np.linalg.norm(king_wordvec))
    print("cosine similarity:", cosine_simi)
```




```
Similar to run: [('runs', 0.656993567943573), ('running', 0.6062964797019958), ('drive', 0.4834049046039513), ('ran', 0.4764978885650635), ('scamper', 0.46932119131088257), ('tworun_double',
0.46402233839035034), ('go', 0.4631645381450653), ('twoout', 0.4574934244155884), ('walk', 0.45697975158691406), ('Mark_Grudzielanek_singled', 0.456517994403839)]

Similar to dirty: [('dirtiest', 0.6239495873451233), ('Kaim_denounced', 0.6171257495880127), ('filthy', 0.6117339730262756), ('lowdown_nasty', 0.6110668182373047), ('dirtier', 0.6008209586143494),
('grubby', 0.5749551653862), ('stinky', 0.5713394284248352), ('smelly', 0.5709947347640991), ('Dirty', 0.567045509815216), ('grimy', 0.5337402820587158)]

Similar to polite: [('courteous', 0.752097487449646), ('everybody_Pendergrast', 0.718908429145813), ('respectful', 0.6748367547988892), ('mannerly', 0.6553859114646912), ('gracious',
0.6316326260566711), ('considerate', 0.6307364106178284), ('mannered', 0.6278921961784363), ('exceedingly_polite', 0.6189175248146057), ('rude', 0.6183223128318787), ('unfailingly_polite',
0.6108803749084473)]

'woman', 'king'-'man': [('queen', 0.7118192315101624), ('monarch', 0.6189672946929932), ('princess', 0.5902429819107056), ('crown_prince', 0.5499460697174072), ('prince', 0.5377322435379028)]

dirty-dirty: 1.0
dirty-clean: 0.5180181
cosine similarity: 0.7046408
```

Word Embeddings:Distributed Representation:Word2vec:Skipgram:tensorflow

Name	Year	URL	Comments
Word2Vec	2013	https://code.google.com/archive/p/word2vec/	A multilingual pre-trained vector is available at https://github.com/Kyubyong/wordvectors.
GloVe	2014	http://nlp.stanford.edu/projects/glove/	Developed by Stanford, it is claimed to be better than Word2Vec. GloVe is essentially a count-based model that combines global matrix decomposition and local context window.
FastText	2016	https://github.com/icoxfog417/fastTextJapaneseTutorial	In FastText, the atomic unit is *n*-gram characters, and a word vector is represented by the aggregation of the *n*-gram characters. Learning is quite fast.
LexVec	2016	https://github.com/alexandres/lexvec	LexVec performs factorization of the **positive pointwise mutual information (PPMI)** matrix using **window sampling and negative sampling (WSNS)**. Salle and others, in their work of *Enhancing the LexVec Distributed Word Representation Model Using Positional Contexts and External Memory*, suggest that LexVec matches and often outperforms competing models in word similarity and semantic analogy tasks.
Meta-Embeddings	2016	http://cistern.cis.lmu.de/meta-emb/	By Yin and others, *Learning Word Meta-Embeddings*, 2016. It combines different public embedding sets to generate better vectors (meta-embeddings).