Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

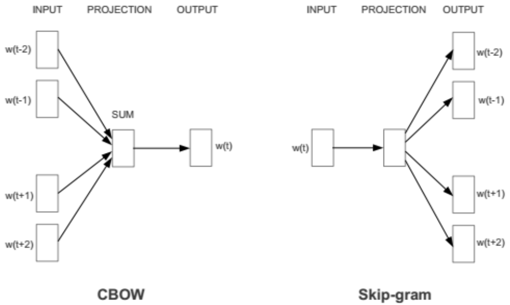
Fabian Barteld,00 Benjamin Milde, Alexander Panchenko

# PART 4: IMPLEMENTING WORD2VEC IN TENSORFLOW

# Introduction

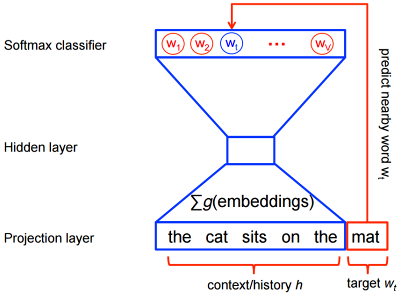
# Training embeddings

- We will now implement Word2Vec in Tensorflow
- (Slides smiliar to https://www.tensorflow.org/tutorials/word2vec)



# Main concepts I

- Neural probabilistic language models are traditionally trained using the maximum likelihood (ML) principle (where $w_t$ is the target word and $h$ is the context):



# Main concepts II

- Neural probabilistic language models are traditionally trained using the maximum likelihood (ML) principle (where $w_t$ is the target word and $h$ is the context):

$$P(w_t|h) = \text{softmax}(\text{score}(w_t, h)) = \frac{exp\{\text{score}(w_t, h)\}}{\sum_{\text{Word w' in Vocab}} exp\{\text{score}(w', h)\}}$$

# Main concepts III

- We train this model by maximizing its log-likelihood on the training set, i.e. by maximizing:

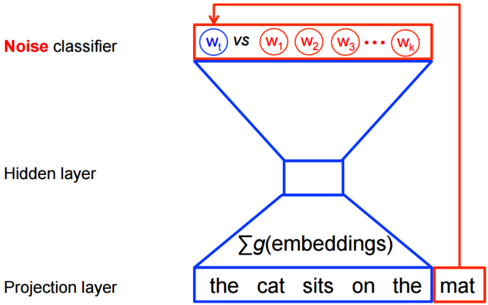
$$J_{\mathsf{ML}} = \log P(w_t|h) \; =$$

$$\mathsf{score}(w_t, h) - \log \left( \sum_{\text{Word w' in Vocab}} \exp \mathsf{score}(w', h) \right).$$

- However this is very expensive, because we need to compute and normalize each probability using the score for all other $V$ words $w'$ in the current context $h$, at every training step.

# Main concepts IV - NCE

- Noise Contrastive Estimation (NCE)
- For feature learning in word2vec we do not need a full probabilistic model. Instead, we train to discriminate the real target words $w_t$ from $k$ imaginary (noise) words w :



# Main concepts V - NCE

- Mathematically, the objective is to maximize:

$$J_{\text{NCE}} = \log Q_\theta(D = 1|w_t, h) + k \mathop{\mathbb{E}}_{\tilde{w} \sim P_{\text{noise}}} [\log Q_\theta(D = 0|\tilde{w}, h)]$$

- discriminate the real target words $w_t$ from $k$ imaginary (noise) words $\tilde{w}$
- where $Q_\theta(D = 1|w, h)$ is the binary logistic regression probability
- under the model of seeing the word $w$ in the context $h$ and assigning the label 1 for datapoint $D$, calculated in terms of the learned embedding vectors $\theta$

# Impl. II - Tensorflow W2V

- In practice we approximate the expectation by drawing k contrastive words from the noise distribution (i.e. we compute a Monte Carlo average)

$$J_{\text{NCE}} \approx \log Q_\theta(D = 1|w_t, h) + \sum_{i=1, w \sim P_{\text{noise}}}^{k} [\log Q_\theta(D = 0|\tilde{w}, h)]$$

- Now we can choose $k \neq |V|$, in practice 5-10 for small datasets, 2-5 for large datasets
- Negative sampling, as in the word2vec paper, is a variant of NCE and uses a specific distribution (uniform raised to the power of 3/4)

# Impl. I - Tensorflow W2V

- In practice we approximate the expectation by drawing k contrastive words from the noise distribution (i.e. we compute a Monte Carlo average)

$$J_{NCE} \approx \sum_{(v_i,v_j)\in B_p} \log \sigma(\mathbf{v}_i \cdot \mathbf{v}'_\mathbf{j}) + \sum_{(v_i,v_j)\in B_n} \log \sigma(-\mathbf{v}_i \cdot \mathbf{v'}_j)\Big)$$

# Impl. II - Tensorflow W2V

```
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))
```

- We can use the NCE loss op of Tensorflow to construct a variant of word2vec. Internally, nce_weights also uses embedding_lookup and does a form of negative sampling directly in Tensorflow.

# Impl. III - Tensorflow W2V

The embeddings matrix is a variable that we want to optimize:

```
embeddings = tf.Variable(
tf.random_uniform([vocabulary_size,
embedding_size], -1.0, 1.0))
```

# Impl. IIII - Tensorflow W2V

We also need 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]))
```

# Impl. IV - embedding_lookup:

```
embed = tf.nn.embedding_lookup(embeddings, train_inputs)
```

e.g. If your list of sentences is: $[[0, 1], [0, 3]]$ (sentence 1 is $[0, 1]$, sentence 2 is $[0, 3]$, the function will compute a tensor of embeddings, which will be of shape $(2, 2, \text{embedding\_size})$ and will look like:

$$[[\text{embedding0, embedding1}], [\text{embedding0, embedding3}]]$$

# Exercise 1 - simple version

- Lets put it together: We can use tf.nn.embedding_lookup for the input projection and tf.nn.nce_loss for the loss (no other layers needed!).
- For simplicity, lets also implement CBOW and Skipgram with a window size of 1.
- E.g. for ”the quick brown fox jumped over the lazy dog”
- (context, target) pairs: ([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox)
- We can simplify to: (the, quick), (brown, quick), (quick, brown), (fox, brown), ... **CBOW**
- or (quick, the), (quick, brown), (brown, quick), (brown, fox), ... **Skip-gram**

# Exercise 2 - advanced version

- Lets try to make a version that does not use tf.nn.nce_loss, as easy as that makes our lives!
- We can also do the negative sampling on the host and code up a linear regression as in the previous tutorials
- Host will assign labels (1 for true context pairs, 0 for noise pairs)
- You have to change the code in the get_batch function and the inputs to your model and adapt your model accordingly

# Hints

- Hint1: The negative samples need a second embedding matrix
- Hint2: For the loss, to get the logits, use the dot product between embedding pairs.
- Hint3: There is no tf.dot(), but you can combine tf.reduce_sum(x,1) and tf.multiply(a,b).
- Hint4: Readable pure Python code with comments: , or if you're feeling masochistic the original uncommented word2vec C impl at:

# Tensorboard

- Visualize loss, embeddings and much more in your browser
- You need to add a few lines of code to tell Tensorboard what to log
- Make sure train_summary_dir is a new directory for every new experiment!

```
loss_summary = tf.summary.scalar('loss', loss)
train_summary_op = tf.summary.merge_all()
summary_writer = tf.summary.FileWriter(train_summary_dir, sess.graph)
```

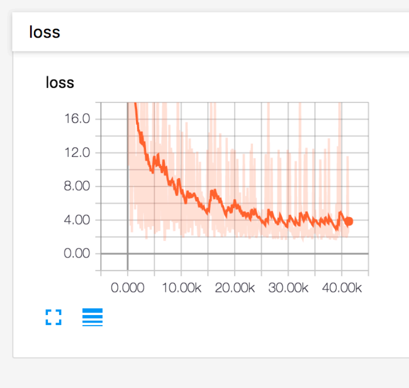
# Tensorboard

- You need to regularly call the train_summary_op in training
- Not as often as the training step, because it will otherwise slowdown your training if you have more complex summaries

```
if current_step % 100==0 and current_step != 0:
^^Isummary_str = sess.run(train_summary_op, feed_dict=feed_dict)
^^Isummary_writer.add_summary(summary_str, current_step)
```

# Tensorboard - running it

```
python3 -m tensorflow.tensorboard --logdir=w2v_summaries_1499773534
--host=127.0.0.1
```



# Tensorboard - embeddings

- Possible to nicely visualize embeddigs, see `https://www.tensorflow.org/get_started/embedding_viz`
- Also checkout `http://projector.tensorflow.org/`, live demo of pretrained embeddings