



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Fabian Barteld, Benjamin Milde, Prof. Dr.
Chris Biemann

TENSORFLOW – DNNS

Embedding layer



Sparse encoding of one-hot values

["A", "short", "example"] → [3011, 291, 14]

`sklearn.preprocessing.LabelEncoder()` does this



Sparse encoding of one-hot values

["A", "short", "example"] → [3011, 291, 14]

`sklearn.preprocessing.LabelEncoder()` does this
.fit → extract the vocabulary from data and assign Integers
.transform → apply the mapping

Sparse encoding of one-hot values

["A", "short", "example"] → [3011, 291, 14]

`sklearn.preprocessing.LabelEncoder()` does this
.fit → extract the vocabulary from data and assign Integers
.transform → apply the mapping

Does not handle unseen values

Sparse encoding of one-hot values

["A", "short", "example"] → [3011, 291, 14]

`sklearn.preprocessing.LabelEncoder()` does this
.fit → extract the vocabulary from data and assign Integers
.transform → apply the mapping

Does not handle unseen values

Can be used directly to compute the (softmax) cross entropy loss

```
tf.nn.sparse_softmax_cross_entropy_with_logits(  
    labels=y_train, logits=pred)
```

`y_train` is expected to be an array of Integers

Embeddings

Embedding layer:

$$emb : V \rightarrow \mathbb{R}^d$$

Embedding matrix:

$|V| \times d$ matrix

Embedding matrix

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))
```


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:

```
[[embedding0, embedding1], [embedding0, embedding3]]
```

Flatten a sequence of inputs

$$[0 \ 0 \ 0 \ 1 \ 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \ 12 \ 19]$$

- `tf.nn.embedding_lookup` does a sparse matrix multiplication for the lookup

Flatten a sequence of inputs

$$\begin{pmatrix} x_{1,1} & \dots & x_{1,embedding_size} \\ \vdots & & \vdots \\ x_{n,1} & \dots & x_{n,embedding_size} \end{pmatrix}$$



$$(x_{1,1} \quad \dots \quad x_{1,embedding_size} \quad \dots \quad x_{n,1} \quad \dots \quad x_{n,embedding_size})$$

Flatten a sequence of inputs

$$\begin{pmatrix} x_{1,1} & \dots & x_{1,embedding_size} \\ \vdots & & \vdots \\ x_{n,1} & \dots & x_{n,embedding_size} \end{pmatrix}$$

→

$$(x_{1,1} \quad \dots \quad x_{1,embedding_size} \quad \dots \quad x_{n,1} \quad \dots \quad x_{n,embedding_size})$$

Tensorflow has `tf.reshape` to do that

Flatten a sequence of embeddings

Input: tensor of rank 3 ([examples, seq_len, embedding_size])

Output: tensor of rank 2 ([examples, embedding_size*seq_len])

```
# tensor 't' is [[[0.344, 1.12], [-0.12, 0.11]],  
#               [[0.344, 1.12], [3.1, -1.78]]]  
# tensor 't' has shape [2, 2, 2]  
# (2 examples of length 2 with embeddings of dim 2)  
reshape(t, [2, 4]) ==> [[0.344, 1.12, -0.12, 0.11],  
                        [0.344, 1.12, 3.1, -1.78]]
```

Flatten a sequence of embeddings

Input: tensor of rank 3 ([examples, seq_len, embedding_size])

Output: tensor of rank 2 ([examples, embedding_size*seq_len])

```
# tensor 't' is [[[0.344, 1.12], [-0.12, 0.11]],  
#               [[0.344, 1.12], [3.1, -1.78]]]  
# tensor 't' has shape [2, 2, 2]  
# (2 examples of length 2 with embeddings of dim 2)  
reshape(t, [2, 4]) ==> [[0.344, 1.12, -0.12, 0.11],  
                        [0.344, 1.12, 3.1, -1.78]]
```

Special value -1: If one component of shape is the special value -1, the size of that dimension is computed so that the total size remains constant. [...] At most one component of shape can be -1.

https://www.tensorflow.org/api_docs/python/tf/reshape

Multinomial logistic regression

Softmax function

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \text{ for "j" = 1, ..., "K".}$$

- Transforms the vector \mathbf{z} , so that all values add up to one
- Can be used as a probability distribution over outputs (=classes)
- Commonly used in neural networks as output function for classification tasks

Cross entropy

Cross-entropy (for C classes):

$$-\sum_{i=1}^n \sum_{c=1}^C y_{ic} \log p_{ic}$$

p_i is the true label

Note: For two classes this is binary-crossentropy

Softmax and cross entropy in Tensorflow

```
tf.nn.softmax_cross_entropy_with_logits(  
labels=y_train, logits=pred)
```

vs.

```
tf.nn.sparse_softmax_cross_entropy_with_logits(  
labels=y_train, logits=pred)
```

- Difference is how y_{train} is given:
- e.g. $[[00010], [00001]]$ vs. $[3, 4]$ or $[0, 0, 1, 0]$ vs. $[2]$

Hands on: A simple POS tagger

A POS tagger (Brown corpus)

- use an embedding layer
- predict the POS tag given the embeddings of the target word and n context words

Getting the predicted class

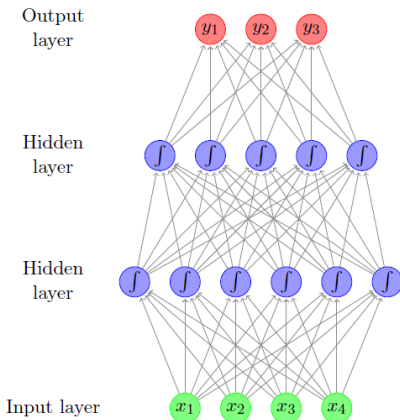
- Getting the class index:
`tf.argmax` returns the index of the largest value in a tensor

Getting the predicted class

- Getting the class index:
tf.argmax returns the index of the largest value in a tensor
- Getting the class names:
Use .inverse_transform of a LabelEncoder to transform sparse one-hot encodings back to non-numeric labels

Deep Feed-Forward Networks

A feed-forward DNN



Multiple layers

First approach: just add layers

```
last_dim = input_dim
for dim in [100, 100]:
    hidden = tf.Variable(tf.random_uniform(
        [last_dim, dim], -0.1, 0.1, dtype=tf.float32))
    b = tf.Variable(tf.random_uniform(
        [dim], -0.1, 0.1, dtype=tf.float32))
    x = tf.add(tf.matmul(x, hidden), b)
    last_dim = dim
```


Multiple layers

First approach: just add layers

```
last_dim = input_dim
for dim in [100, 100]:
    hidden = tf.Variable(tf.random_uniform(
        [last_dim, dim], -0.1, 0.1, dtype=tf.float32))
    b = tf.Variable(tf.random_uniform(
        [dim], -0.1, 0.1, dtype=tf.float32))
    x = tf.add(tf.matmul(x, hidden), b)
    last_dim = dim
```

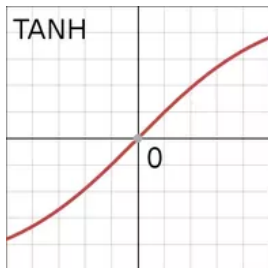
Won't add anything to the models capacity:
as each layer is just a linear function this is just composing linear functions – the result is a linear function

Adding non-linearities

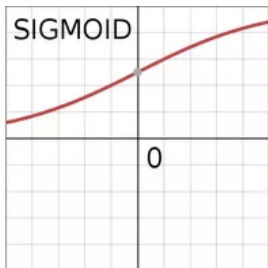
Different choices: Sigmoid, tanh, ReLU (Rectified Linear Unit)

```
last_dim = input_dim
for dim in [100, 100]:
    hidden = tf.Variable(tf.random_uniform(
        [last_dim, dim], -0.1, 0.1, dtype=tf.float32))
    b = tf.Variable(tf.random_uniform(
        [dim], -0.1, 0.1, dtype=tf.float32))
    x = tf.add(tf.matmul(x, hidden), b)
    x = tf.nn.relu(x)
    last_dim = dim
```

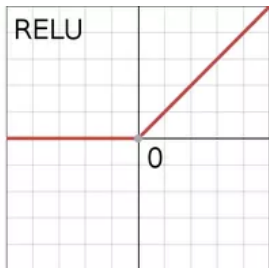
Non-linearities



`tf.tanh`



`tf.sigmoid`



`tf.nn.relu`

MLP with one hidden layer

One hidden layer is enough to create a universal approximator.

Example

```
last_dim = input_dim
for dim in [100, 100]:
    ### add hidden layers as before

## define the output layer
out_w = tf.Variable(tf.random_uniform(
    [last_dim, out_dim], -0.1, 0.1,
    dtype=tf.float32))
b = tf.Variable(tf.random_uniform(
    [out_dim], -0.1, 0.1, dtype=tf.float32))
out = tf.add(tf.matmul(x, out_w), b)
```

Hands on 1

Add hidden layers to the POS tagger.

Dropout

- Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of machine learning research 15.1 (2014): 1929-1958.

In Tensorflow:

```
tf.nn.dropout( x, keep_prob, noise_shape=None,  
seed=None, name=None)
```


Hands on 2

Extend POS tagger with a train a test set, measure accuracy.
(Hint: `sklearn.metrics.accuracy_score`) Add dropout to the
POS tagger and compare train and validation loss+accuracy.

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

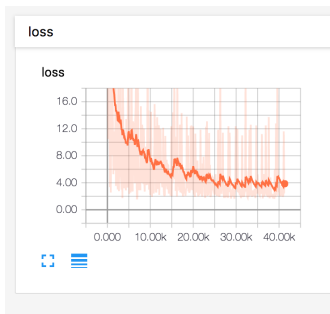
```
if current_step % 100==0 and current_step != 0:  
    summary_str = sess.run(train_summary_op, feed_dict=feed_dict)  
    summary_writer.add_summary(summary_str, current_step)  
    summary_writer.flush()
```

- Some useful statistics on tensors (e.g. neural network weights):

```
with tf.name_scope('x1'):  
    tf.summary.scalar('mean', mean)  
    with tf.name_scope('stddev'):  
        stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))  
    tf.summary.scalar('stddev', stddev)  
    tf.summary.scalar('max', tf.reduce_max(var))  
    tf.summary.scalar('min', tf.reduce_min(var))  
    tf.summary.histogram('histogram', var)
```

Tensorboard - running it

```
tensorboard --logdir=w2v_summaries_1499773534  
--host=127.0.0.1  
or  
python3 -m tensorboard.main --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

