Assignment 5

Word Class Prediction with Neural Networks

Verena Blaschke

July 18, 2018

Assignment 5

I: Data preprocessing and encoding

II: Simple feed-forward network

III: Effect of padding direction

IV: Back to logistic regression

V: Recurrent networks

I: Data preprocessing and encoding

```
noun  gemeinderat
noun  grenzpolizei
verb  ruinieren
noun  halbtönen
noun  energieexporteuren
...
```

I: Data preprocessing and encoding

noun	gemeinderat
noun	grenzpolizei
verb	ruinieren
noun	halbtönen
noun	energieexporteuren
...	

`train_y`: `[1, 1, 0, 1, 1, ...]` (or with 0s and 1s switched)

I: Data preprocessing and encoding

noun	gemeinderat
noun	grenzpolizei
verb	ruinieren
noun	halbtönen
noun	energieexporteuren
...	

train_y: [1, 1, 0, 1, 1, ...] (or with 0s and 1s switched)

train_x: (before padding)
Alphabet: 30 encodings (+ 1 'unknown')
[('a', 8), ('b', 16), ('c', 22), ('d', 6), ('e', 2), ('f', 19), ('g', 1), ..., ('ö', 17), ('ü', 20)]
unknown: 31

train_x: [[1, 2, 3, 2, 4, 5, 6, 2, 7, 8, 9], ...]

I: Data preprocessing and encoding

Alphabet: 30 encodings (+ 1 'unknown')

```
[('a', 8), ('b', 16), ('c', 22), ('d', 6), ('e', 2),
('f', 19), ('g', 1), ..., ('ö', 17), ('ü', 20)]
```

unknown: 31

padding: 0

Longest word in `train.txt`: 31 characters

Pad shorter words with 0s so all word representations have the same length:

```
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 2, 3, 2, 4, 5, 6, 2, 7, 8, 9], ...]
```

```
train_x = keras.preprocessing.sequence.pad_sequences(
    train_x)
```

I: Data preprocessing and encoding

`test_x`, `test_y`:

- Use the same encoding schemes as for `train_x`, `train_y`.
- Make use of the encoding for 'unknown' characters.
  (Not applicable for the given training & test sets, but relevant for other applications!)

I: Data preprocessing and encoding

`test_x`, `test_y`:

- Use the same encoding schemes as for `train_x`, `train_y`.
- Make use of the encoding for ‘unknown’ characters.
  (Not applicable for the given training & test sets, but relevant for other applications!)
- Pad/truncate the word representations to match the length of the longest word from the **training** set.

```
test_x = keras.preprocessing.sequence.pad_sequences(
    test_x, maxlen=train_x.shape[1])
```

I: Data preprocessing and encoding

One-hot encoding

- `sklearn.preprocessing.OneHotEncoder`
- `keras.utils.to_categorical`
  ⚠flatten the array correctly
- Needs to be able to handle the char-to-int mapping from the training data
  and the encoding for ‘unknown’ characters
  and the padding.

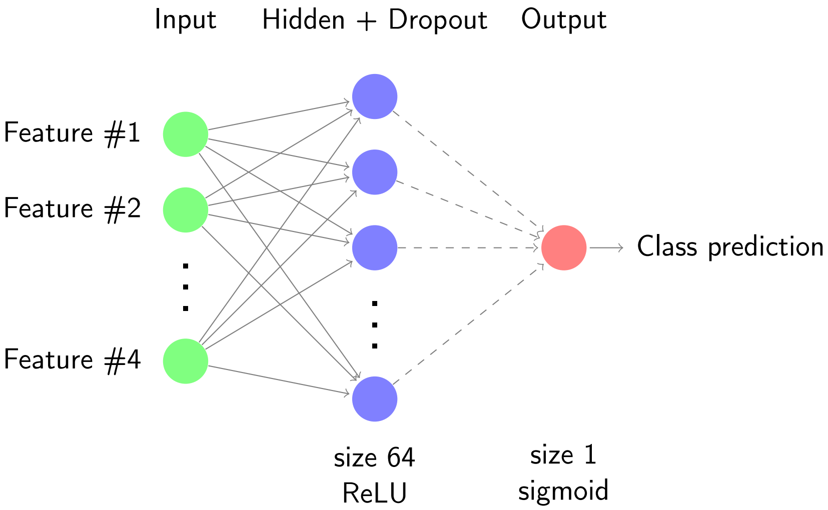
I: Data preprocessing and encoding

One-hot encoding

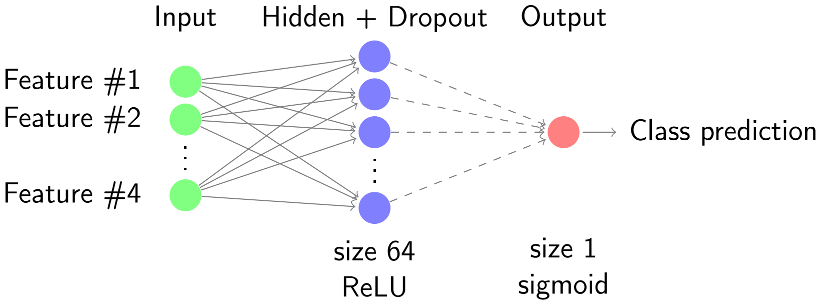
- `sklearn.preprocessing.OneHotEncoder`
- `keras.utils.to_categorical`
  ⚠flatten the array correctly
- Needs to be able to handle the char-to-int mapping from the training data
  and the encoding for 'unknown' characters
  and the padding.

`train_x`	`(20000, 31)`	max word len
`test_x`	`(6561, 31)`	max word len
`train_onehot`	`(20000, 992)`	max word len x unique chars
`test_onehot`	`(6561, 992)`	max word len x unique chars

II: Simple feed-forward network



II: Simple feed-forward network



```
model = Sequential()
model.add(Dense(units=64,
                input_dim=train_onehot.shape[1],
                activation='relu'))
model.add(Dropout(rate=0.7))
model.add(Dense(units=1, activation='sigmoid'))
```

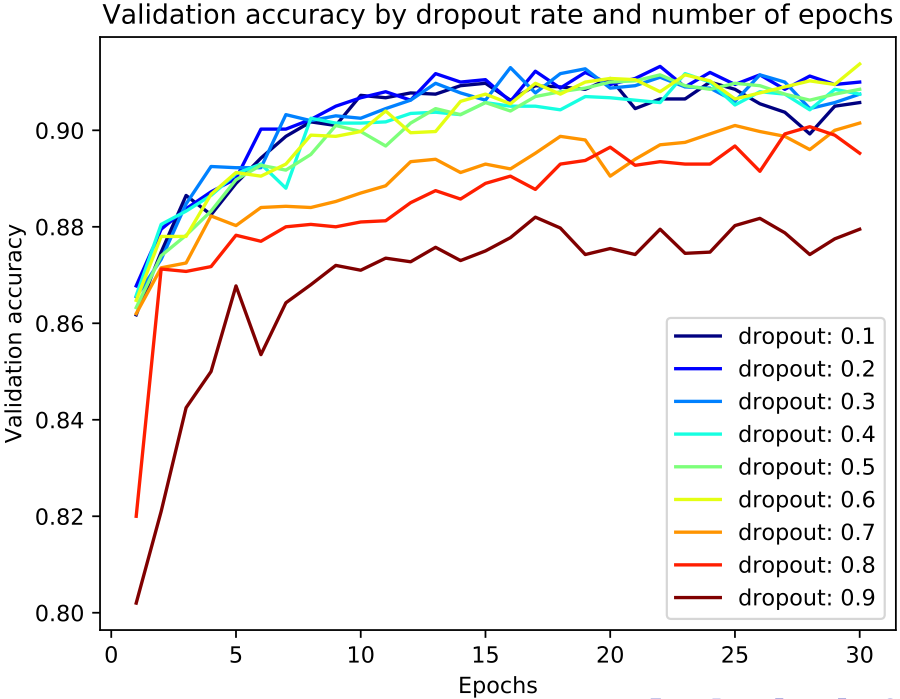
II: Simple feed-forward network

```
model = Sequential()
model.add(Dense(units=64,
                input_dim=train_onehot.shape[1],
                activation='relu'))
model.add(Dropout(rate=0.7))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
hist = model.fit(train_onehot, train_y,
                 batch_size=32,
                 epochs=30,
                 validation_split=0.2)
```

II: Simple feed-forward network

Tuning the number of epochs:

```
hist = model.fit(train_onehot, train_y,
                 batch_size=32,
                 epochs=30,
                 validation_split=0.2)
best_epoch = np.argmax(hist.history['val_acc'])
best_score = hist.history['val_acc'][best_epoch]
best_epoch += 1
```


Validation accuracy by dropout rate and number of epochs
Validation accuracy
0.80
0.82
0.84
0.86
0.88
0.90
0
5
10
15
20
25
30
Epochs
dropout: 0.1
dropout: 0.2
dropout: 0.3
dropout: 0.4
dropout: 0.5
dropout: 0.6
dropout: 0.7
dropout: 0.8
dropout: 0.9

II: Simple feed-forward network

Evaluation:
with `sklearn.metrics`:

```
pred = np.around(model.predict(test_onehot))
print(accuracy_score(test_y, pred))
print(precision_recall_fscore_support(test_y, pred,
                                      average='macro'))
```

III: Effect of padding direction

```
train_x = keras.preprocessing.sequence.pad_sequences(
    train_x, padding='post')
```

padding direction	accuracy	precision	recall	F1
pre	0.9131	0.8743	0.8359	0.8531
post	0.9059	0.8612	0.8248	0.8412

pre	post
00000zahl**ung**	zahl**ung**00000
00ausbild**ung**	ausbild**ung**00

IV: Back to logistic regression

```
model = Sequential()
model.add(Dense(units=1,
                input_dim=train_onehot.shape[1],
                activation='sigmoid',
                kernel_regularizer=keras.regularizers.l2(
                    0.5)))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

IV: Back to logistic regression

```
model = Sequential()
model.add(Dense(units=1,
                input_dim=train_onehot.shape[1],
                activation='sigmoid',
                kernel_regularizer=keras.regularizers.l2(
                    0.5)))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

model	accuracy	precision	recall	F1	
FNN	0.9131	0.8743	0.8359	0.8531	
Logit (Keras)	0.8070	0.4035	0.5	0.4466	(epochs 1-50)
Logit (sklearn)	0.8879	0.83255	0.7891	0.8079	

V: Recurrent networks

```
model = Sequential()
model.add(Embedding(input_dim=len(alphabet) + 2,
                    input_length=train_x.shape[1],
                    output_dim=32,
                    mask_zero=True))
model.add(Dropout(rate=0.6))
model.add(GRU(units=64))
model.add(Dropout(rate=0.7))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
hist = model.fit(train_x, train_y, batch_size=32,
                 epochs=20, validation_split=0.2)
```

V: Recurrent networks

model	accuracy	precision	recall	F1
FNN (tuned*)	0.9131	0.8743	0.8359	0.8531
RNN (untuned**)	0.9250	0.8730	0.8940	0.8829

* dropout=0.6, epochs=29
** dropout (embeddings)=0.1, dropout (GRU)=0.1, epochs=20, embedding depth=100, GRU size=64

V: Recurrent networks

Why `train_x` instead of `train_onehot`?

```
train_x   ...2, 3, 1, 28, ..., 2, 1, 17, 3, 23, ...
train_y   ...0, 0, 0, 1, 0, ..., 0, 0, 0...
```

V: Recurrent networks

Why `train_x` instead of `train_onehot`?

```
train_x   ...2, 3, 1, 28, ..., 2, 1, 17, 3, 23, ...
train_y   ...0, 0, 0, 1, 0, ..., 0, 0, 0...
```

Pre-padding or post-padding?
(`mask_zero`)

V: Recurrent networks

Why `train_x` instead of `train_onehot`?

```
train_x   ...2, 3, 1, 28, ..., 2, 1, 17, 3, 23, ...
train_y   ...0, 0, 0, 1, 0, ..., 0, 0, 0...
```

Pre-padding or post-padding?
(`mask_zero`)

Good embedding depth?