Implementing Deep Learning for Sentiment Analysis

Johann Mitlöhner

Vienna University of Economics and Business

Mai 4, 2018

Sentiment Analysis

Applying Natural Language Processing and Text Analysis techniques to identify and extract subjective information from a piece of text

- product reviews
- social media

Business Applications:

- Listen to the voice of the customer
- What features do they like and dislike about products
- Determine marketing strategy
- Identify trends
- Respond in a timely manner

Other Applications, e.g. politics: monitor sentiment on election candidates

Types Of Sentiment Analysis

- Manual processing
  - most accurate judge of sentiment
  - still not 100% accurate
  - increasingly infeasible due to prolific growth of social media data
- Keyword processing
  - assign a degree of positivity or negativity to individual words positive: great, like, love or negative: terrible, dislike
  - overall percentage score
  - very fast and cheap to implement
  - problems with multiple meanings and context
- Natural Language Processing
  - aims at detecting meaning and context
  - Many approaches, in the following:
    - word embeddings
    - neural nets
    - Simple architecture i.e. Feed-forward

SemEval-2017 Task 4

Sentiment Analysis in Twitter
Subtask A Message Polarity Classification: positive, negative, or neutral sentiment

- 50.000 tweets
- CrowdFlower used to annotate tweets
  - Each tweet annotated by at least five people
  - hidden tests for quality control
  - Label accepted if three out of five annotators agreed
  - Otherwise average mapped to closest integer value
- Results are available for comparison

Sample Data

100002521464053761 positive
Going to see Rise of the Planet of the Apes tonight...oh yeah.
100002606734245888 negative
Uh drunk and tired and just remembered I'm going to Ireland in the morning and need to pack. Ireland :(
100002627009515520 neutral
@Mariners Guti's catch at the wall in May against the Yankees at Safeco and Trayvon's catch are the leaders in the clubhouse.
100001422170529792 positive
@jordanknight Hope u have a great show tonight. Me & the Chi/IN girls are so excited to see you in London tomorrow!
256869888373714944 negative
Just noticed that the music for Bittersweet Symphony is credited to Mick Jagger and Keith Richards. Find out why and you may dislike them.

Feature Extraction

- text is messy
- machine learning algorithms need well-defined inputs
  - convert text to numbers
  - add more features, e.g. part of speech

Many approaches, e.g.

- Bag of Words
  - One-Hot
  - TF-IDF
- Word Embeddings
  - Word2Vec
  - GloVe
  - pre-trained or train your own (needs very large corpus)

Bag of Words: One-Hot

- define set of words, e.g. most frequent
- 1 if word present in tweet, 0 if not
- fixed-size input vector for each tweet

100001422170529792 positive
@jordanknight Hope u have a great show tonight. Me & the Chi/IN girls are so excited to see you in London tomorrow!
256869888373714944 negative
Just noticed that the music for Bittersweet Symphony is credited to Mick Jagger and Keith Richards. Find out why and you may dislike them.

great	awesome	terrible	dislike
1	0	0	0
0	0	0	1

- Simple method
- word order lost
- still, often acceptable results

Bag of Words: TF-IDF

Problems with One-Hot:

- frequent words carry little information
- stopwords are not flexible to the application domain

TF(t) = (Number of times term t appears in document) / (Total number of terms in document)
IDF(t) = log(Total number of documents / Number of documents with term t in it)
TFIDF(t) = TF(t) * IDF(t)
Still problems common to all Bag of Words methods:

- vocabulary choice is difficult
- sparse encoding makes computation harder
- word order ignored
- context ignored, meaning often lost

Word Embeddings

Words mapped to vectors of real numbers
E.g. Glove Global Vectors for Word Representation

- GloVe model trains on global co-occurrence counts of words
- preserves word similarities with vector distance

Various pre-trained word vectors available, e.g.

- Wikipedia + GigaWord
- Common Crawl
- Twitter 2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB

great 0.10751 0.15958 0.13332 0.16642 -0.032737 0.17592 ...
dislike 0.049184 0.19144 -0.40721 0.58064 -0.33959 0.75671 ...
phrase encoding: element-wise sum of word encodings, optionally divide by number of words

Single-layer Feed-forward Network

- input vector $\mathbf{x}$
- correct response $\mathbf{y}$
- weight matrix $\mathbf{W}$
- bias $\mathbf{b}$
- non-linear activation function, e.g. tanh
- network output $\mathbf{o}$

$$\mathbf{o} = \tanh(\mathbf{W}\mathbf{x} - \mathbf{b})$$

Optimise weights to minimise error $\mathbf{y} - \mathbf{o}$

- various optimisers
- learning rule, e.g. gradient descent

Activation Functions

- Sigmoid $\sigma(x) = 1/(1 + e^{-x})$ fallen out of favour because
  - vanishing gradient at tails, no weight update
  - output not zero-centered, problems with gradient descent
- tanh; note that $\tanh(x) = 2\sigma(2x) - 1$
  - also suffers from vanishing gradient, but
  - output is zero-centered, therefore
  - in practise preferred to sigmoid
- ReLU rectified linear unit $f(x) = \max(0, x)$
  - accelerates convergence of stochastic gradient descent compared to tanh (as much as factor 6)
  - cheap computation
  - can be fragile and never activate again
  - need careful choice of learning rate
- Softmax

Linear Softmax Classifier

y correct class e.g. $(1, 0, 0, ...)$
f $= \mathbf{Wx}$ scores for each class
p vector of probabilities with

$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}}$$

Cross-entropy loss function:

$$L = -\mathbf{y} \cdot \log(\mathbf{p})$$

E.g. $L = -1 \times (1 \times log(0.3) + 0 \times log(0.2) + 0 \times log(0.2) + ...)$
When probability for correct class goes towards 1 loss goes towards 0 since $log(1) = 0$

$$\frac{\delta L_i}{\delta f_k} = p_k - 1(y_i = k)$$

where $1(y_i = k)$ is 1 if $y_i = k$ and 0 otherwise.

Two-Layer Network

- Hidden layer $\mathbf{h}$, activation function tanh or ReLU
- Output layer class probabilities $\mathbf{p}$, activation function softmax
- Weight matrices $\mathbf{W}, \mathbf{V}$, bias $\mathbf{b}$

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} - \mathbf{b})$$

$$\mathbf{f} = \mathbf{V}\mathbf{h}$$

$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}}$$

- optimise weights and biases, e.g. gradient descent
- more units in hidden layer: better on *training data*
- not generalising to unseen data, i.e. validation/test data

Training, Validation, Test

Three data sets:

- training data: optimiser works on this data to find weights (and biases)
- validation data:
  - network responses for data not seen during training
  - serve as measure of success for parameter tuning such as number of hidden units
- test data:
  - measure computed only when parameter tuning is finished
  - this result is reported

Reasoning: performance on validation data is also result of optimisation (of parameter and architecture choices by the developer) and therefore specific to validation data
Usual splits: 60/20/20 or 80/10/10

Deep Learning

Connectionism popular in early 90s, but:

- limited computer power
- large training data sets not available
- deep networks with more than two layers not feasible

Deep Learning brings renaissance to neural networks:

- huge data sets available, e.g. social media
- computer power and memory increased dramatically, esp. GPU
- advances in algorithms, e.g. stochastic gradient descent, ReLU
- spectacular successes in image recognition

Stochastic Gradient Descent

Weight update uses gradient, e.g. for softmax

$$\frac{\delta L_i}{\delta f_k} = p_k - 1(y_i = k)$$

where $1(y_i = k)$ is 1 if $y_i = k$ and 0 otherwise.

- Classic gradient descent:
  - whole data set used
  - expensive computation
- Stochastic gradient descrent:
  - choose small number of random data points (minibatch)
  - usually sufficient to calculate gradient and converge
  - **much** faster computation

GPU Computing

- NVIDIA graphics card with CUDA support
- speedup 5x, 10x, .. depending on HW and application
- high-level framework like Keras: no change of code
- data has to fit in card memory

PC Hardware:

- RAM rule of thumb: at least twice the GPU memory
- Power supply
  - high-end graphics cards need several power connectors
  - check if mainboard supplies enough connectors and total power
- space! high-end cards are BIG
  - does the card fit in the case? check dimensions
  - no components on mainboard in the way?

Multiple card feasible with e.g. Tensorflow and Keras

Choices for Graphics Card

NVIDIA or AMD?

- CUDA good support with DL frameworks
- AMD uses OpenCL, currently (2018) much less support

$\Rightarrow$ NVIDIA

Popular models with good value:

Model	VRAM GB	Bandw. GB/sec	Cores	Price
GTX 1080 Ti	11	484	3584	760
High-end, best choice				
GTX 1070 Ti	8	256	2432	500
Can still compete on Kaggle				
GTX 1060	6	216	1280	300
Good enough for many NLP tasks				
GTX 1050 Ti	4	112	768	170
Budget model to get into DL				

GPU Computing: Software

Software for Linux: Python + DL framework

- optimized linear algebra library, e.g. OpenBLAS
- Numpy
  - Python library for fast and convenient array computing
  - if linear algebra library installed:
  - makes use of multi-core with no change of code
  - speed up close to number of cores
- Keras: Python framework for Deep Learning
- Backend for Keras:
  - Theano
  - TensorFlow
- CUDA: Nvidia library for GPU computing

Pure Python vs Framework

- Programming neural networks in pure Python is possible
- Simple two-layer feed-forward net is just a few lines
- More complicated architectures:
  - gradient computation complex
  - pure Python infeasible
- Theano
  - can compute gradients automatically
  - solid choice for experiments in new architectures
  - still too complex for application-oriented projects
- Keras
  - high-level neural networks API
  - written in Python
  - can run on top of Theano or TensorFlow
  - provides standard components for deep learning networks
  - convenient support for GPU computing

Keras Layers

- Dense – fully connected neural net layer
- Dropout – randomly set fraction of input to 0 to avoid overfitting
- Convolution – repeatedly applied same computation to parts of input
- MaxPooling – propagate only maximum of input regions
- Recurrent layers – RNN, GRU, LSTM, ..

```
model = Sequential()
model.add(Dense(400, activation='relu', input_shape=(200,)))
model.add(Dropout(0.2))
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

Keras Optimizers

Adapt weights and biases after each minibatch to decrease loss

- SGD – stochastic gradient descent
- RMSprop – adapt step size to gradient

```
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_val, y_val))
score = model.evaluate(x_val, y_val, verbose=0)
```

GPU Configuration and Timings

- Linux config file $HOME/.keras/keras.json
  "backend": "tensorflow"
  Backend choices: theano, tensorflow
- Hardware:
  - CPU: i7-6700 Quad-Core, 8 threads
  - GPU: GTX 1060 6GB
- Timings: 200 dim embeddings, two hidden ReLU layers with h units each, dropout 0.2, output Softmax, minibatch 128, 30 epochs

h	CPU	CUDA	Speedup
200	15.3	12.4	1.2
400	27.0	12.9	2.0
800	75.4	16.8	4.4
1600	212.8	35.4	6.0

- Switch off GPU on command line e.g.
  CUDA_VISIBLE_DEVICES="" python kera.py tweets.txt 1600

continue in sentiment.html ▸ Link