SAMSUNG

AI on Her Network Majesty's Service

How to extract knowledge from Social Media

Maciej Pieńkosz
m.pienkosz@samsung.com
NLP Research Group
Samsung R&D Poland

Workshop plan

What we will learn today

- How we can utilize the vast textual amount of data on Social Media for our needs
- How to process and analyze text automatically using Machine Learning/Deep Learning algorithms
- How to implement the Deep Learning algorithms for Text processing using Keras
- Some practical tips for applying Machine Learning/Deep Learning

Agenda:

- NLP on Social Media – why bother?
- Types of NLP tasks: opinion mining, event detection, summarization
- Classic NLP Methods: Machine Learning with hand-engineered features
- Deep Learning for NLP
- Workshop (case study: Disaster events detection on Twitter data)
- Some coding ☺ !

Why Social Media analysis

- **Social media textual data is the collection of openly available texts that can be obtained publicly via blogs and micro-blogs, Internet forums, user-generated FAQs, chat, podcasts, online games, tags, ratings, and comments.**
- **Large volume of user-generated content creates new opportunity for understanding social behavior, building socially intelligent systems and understanding user needs**
- **We can analyze this data to gain insight about:**
  - what people think about certain topics (products, politics, services) – media monitoring (sentiment analysis, opinion mining)
  - what important events they are talking about (disasters, malfunctions/problems with our products, etc.)
  - what is talked about on given topics in general ☺ (information retrieval and summarization)
- **Loads of data for analysis, fairly easy to obtain; true voice of the customers**

Type	Characteristics	Examples
Social Networks	A social networking website allows the user to build a Web page and connect with a friend or other acquaintance in order to share user-generated content.	MySpace, Facebook, LinkedIn, Meetup, Google Plus+
Blogs and Blog Comments	A blog is an online journal where the blogger can create the content and display it in reverse chronological order. Blogs are generally maintained by a person or a community. Blog comments are posts by users attached to blogs or online newspaper posts.	Huffington Post, Business Insider, Engadget, and online journals
Microblogs	A microblog is similar to a blog but has a limited content.	Twitter, Tumblr, Sina Weibo, Plurk
Forums	An online forum is a place for members to discuss a topic by posting messages.	Online Discussion Communities, phpBB Developer Forum, Raising Children Forum
Social Bookmarks	Services that allow users to save, organize, and search links to various websites, and to share their bookmarks of Web pages.	Delicious, Pinterest, Google Bookmarks
Wikis	These websites allow people to collaborate and add content or edit the information on a community-based database.	Wikipedia, Wikitravel, Wikihow
Social News	Social news encourage their community to submit news stories, or to vote on the content and share it.	Digg, Slashdot, Reddit
Media Sharing	A website that enables users to capture videos and pictures or upload and share with others.	YouTube, Flickr, Snapchat, Instagram, Vine

Social Media types

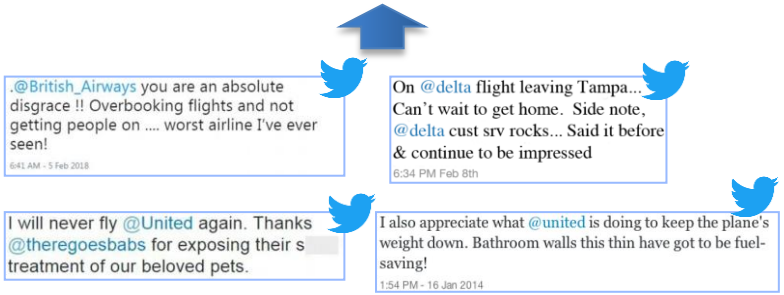
from „Natural Language Processing for Social Media" by A/Farzindar and D.Inkpen, 2015

Sentiment and Opinion Analysis on Social Media

- **Detect attitude (positive, negative, neutral) expressed in the piece of text**

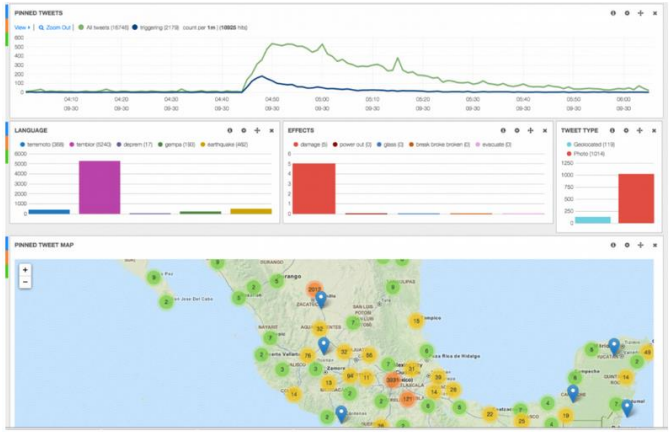
Source: https://www.kaggle.com/crowdflower/twitter-airline-sentiment

- **Applications:**
  - Business: get opinions about products and services, predict market trends
  - Trading: predict the attitude towards company's stock when making investement decisions
  - Politics: Gauge public opinion on candidates
  - „Hate" speech detection



Event detection

- **Social Media (Twitter) may be used to detect and notify about large scale events that particulary influence people's daily life**
  - Social events like large parties, sports events, exhibitions, political campaigns
  - Natural events like storms, heavy rainfall, tornadoes, typhoons/hurricanes/cyclones, earthquakes
- **When big event occurs (such as earthquake) -> people make many tweets related to the earthquake**
- **After Sichuan earthquake (China) in 2008, Twitter was faster at reporting the earthquake than U.S. Geological Survey (USGS), an official US organization in charge of tracking such events**
  - USGS was able to process only ~2,000 realtime earthquake sensors, mostly in US
- **Following this event, USGS developed a system that was able to detect 2014 Napa earthquake (Chile) in 29 seconds!**
  - The system triggers alert within ~2 minutes after the event



Source: https://blog.twitter.com/official/en_us/a/2015/usgs-twitter-data-earthquake-detection.html

Summarization

- **Summarization of posts/replies on social media**
- **Reason: Huge number of messages about events, many of which contain irrelevant or redundant information**
- **Find representative posts from a given set of posts**

- “@morrowchris: Christmas card from the Obama’s :) http://pic.twitter.com/F3VU52io" thts special.
- Christmas card from the Obama’s :) http://pic.twitter.com/7xqBQdBV
- RT @liberalease: RT if you like President Obama better than ALL OF GOP COMBINED. / That was easy :)
- Obama did it! The war is OFFICIALLY OVER! #WelcomeHomeTroops! :)
- obama really brought the troops home :)

- Christmas card from the Obama’s :) http://pic.twitter.com/7xqBQdBV
- obama really brought the troops home :)

Source: Liu, et al. 2012: Graph-based Multi-tweet Summarization Using Social Signals

SAMSUNG

How to analyze textual Social data - NLP techniques

Machine learning for Text Analysis – „classical" approach

Recall sentiment analysis task:

Text processing (classification) pipeline

- graet -> great
- 2day -> today
- Hello -> hello
- http://t.co/Ajay0sNPlg -> $URL

„I really like my new phone"

[0, 0.3, 0.7, -0.8, 1, 0, 1]

- Logistic regression
- SVM
- Decision Tree, Random Forest
- Naive Bayes
- MLP (Feedworward Neural Network)
- KNN

Text Normalization

2day i lost my fne -> Today I lost my phone

- **Traditionally, NLP systems are designed to handle input that is:**
  - Gramatically correct
  - No spelling errors
  - Single language
  - The right script
  - Text-like and formal
- **Very often social data is unnormalized text, which is hard for analysis:**
  - Texts are unstructured, presented in many formats and written by different people in many languages and styles
  - Informal language, slang
  - Misspellings, typographic errors
  - Hashtags, @-mentions, URLs
- **Text normalization is usually an important step**
  - Jargon normalization
  - Spelling correction
  - Entiry Recognitions (URLS, @-mentions, etc.)

Basic approach: hand engineered features

What words are in the sentence

- Individual words (Unigrams)
- Pairs of words, phrases (Bigrams)
- Parts of words (ngrams)

Hand engineered features

- Presence of emoticons
- Presence of words from the dictionary
- Part of speech (POS) tags
- Dependency relations
- Presence of negation (‚not', ‚n't', ‚never')
- Presence of Named Entities (NE)
- Presence of exclaimation marks
- WordNet relations
- Custom rules, e.g. presence of certain words/phrases, elongation, etc.

Features for sentence „I like Samsung :]"

- **unigrams:** ‚I', ‚like', ‚samsung', ‚:]'
- **bigrams:** ‚I like', ‚like Samsung', ‚Samsung :]'
- **POS tags:** ‚PRP', ‚VB', ‚NP', ‚SYM'
- **negation:** NO
- **emoticon:** YES
- **dictionary positive:** 2,
- **dictionary negative:** 0
- **Named Entities**: YES
- ….

Representation building

- **Features need to be converted to the numeric form, compliant with the Classifier**
- **Basic representation: one-hot encoding**

Corpus:
- I like Samsung
- New phone is awesome :)
- I do not like new phone

Features:
- Unigrams
- Negation
- Emoticon

Feature size:
- Unigrams: 10 (10 unique words)
- Negation: 2 (yes/no)
- Emoticon: 2 (yes/no)

Unigrams:
- I like Samsung: [1,1,1,0,0,0,0,0,0,0]
- I do not like new phone [1,1,0,1,1,0,0,0,1,1]

Negation:
- I like Samsung: [0]
- I do not like new phone [1]

Emoticon:
- I like Samsung [0]
- I do not like new phone [0]

Resultant corpus:

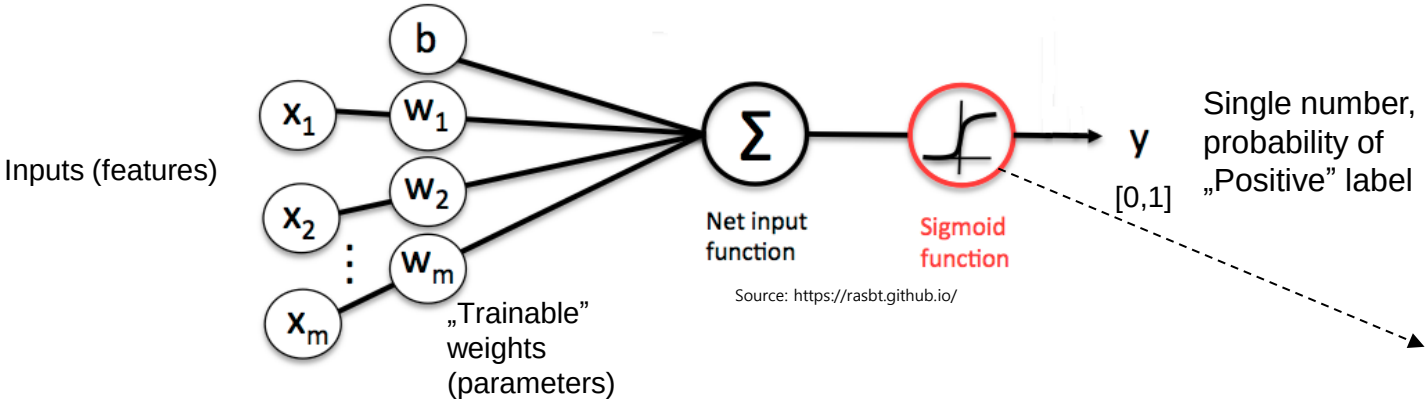
[1,1,1,0,0,0,0,0,0,0,0,0]
[0,0,0,1,1,1,1,1,0,0,0,1]
[1,1,0,1,1,0,0,0,1,1,1,0]

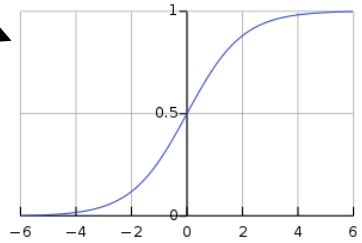
- **Additional possible transformations:**
  - Word frequency
  - Tf/idf
  - Feature selection

Classifier example – Logistic Regression

- **Simple classifier: logistic regression**
- **Full equation(X, W -> vectors):**

$$y = sigmoid(W \cdot X + b)$$



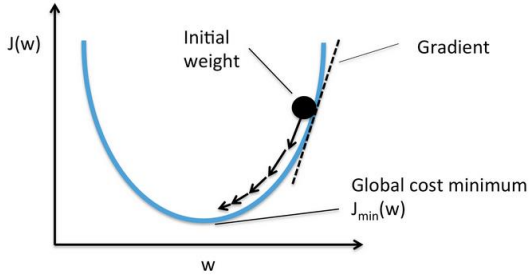


Source: https://en.wikipedia.org/

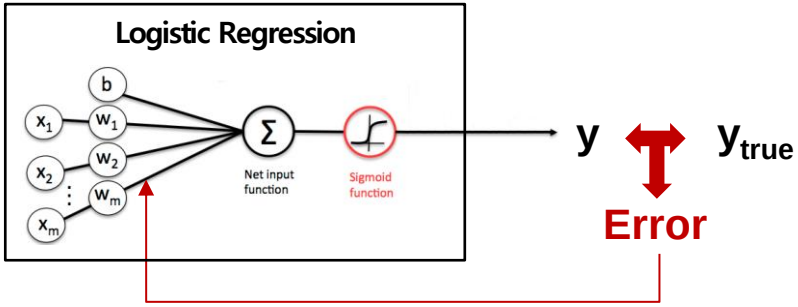
Training Logistic Regression

- **Training -> learn weights W**
- **Simplest algorithm: Stochastic Gradient Descent (SGD)**
  - Recall: $y = sigmoid\ (W \cdot X + b)$
  - Compare model outputs (W, the probability) with true label (0 or 1)
  - Compute error (loss function): (mean squared error, logistic loss)
  - Determine how individual weights contribute to the error – the gradient
  - Update weights in inverse proportion to the gradient

$$W_{new} = W_{old} - \alpha \cdot W_{grad}$$

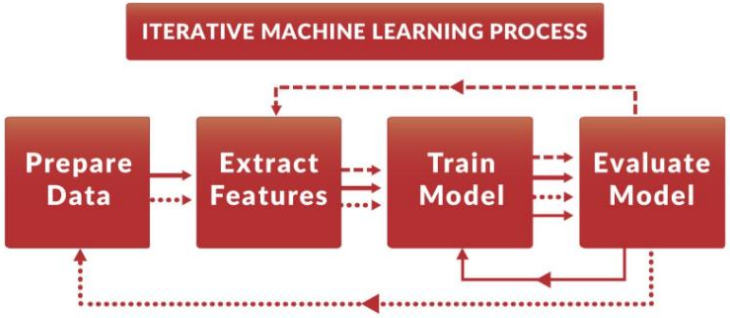


Source: https://stackoverflow.com/



Iterative machine learning process

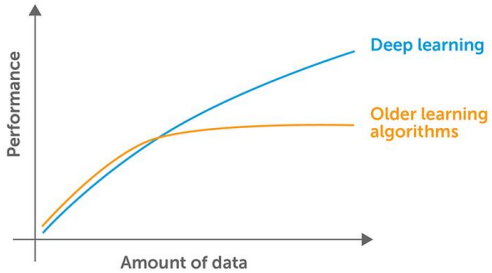
- **Build your dataset: instances (texts) and labels (0 or 1, or even more categories)**
  - Get an opensourced one, or
  - Gather data (e.g. through Twitter API) and label yourself
- **Split your data to training / validation/ test splits**
- **Choose a metric to measure the performance of the algorithm (e.g. accuracy – % of correctly classified examples)**
- **Train on training split**
  - For every example in training data: compute score, compute classification error, backpropagate (compute gradients), update weights
- **Evaluate on validation split**
  - For every example in validation data: classify example, check if prediction was correct (with given metric)
- **Analyze errors, add more features, implement ideas, regularize your model, iterate**
- **Report final performance on test split**



Source: https://www.kdnuggets.com/2017/05/data-version-control-iterative-machine-learning.html

Deep Learning for NLP

- **Problems with hand engineered features:**
  - Have limits: won't capture whole complexity of the language
  - Need to handcraft features (usually separately for every task)
  - Don't capture relations between features (e.g. word „intelligent" and word „smart")
- **Deep Learning – Algorithms based on blocks, multiple, stacked neural network layers that work on basic text features (vision -> pixels, audio -> raw waveform, text -> word/character) and „automagically" learn high level features**
- **With the computing power increase, deep learning takes over the scene**
  - More flexible, distributed representations
  - Fast (although quite slow to train, due to complexity and „deep" structure)
  - Able to learn complex relations present in the data; Able to „let the data speak" in the big data regime

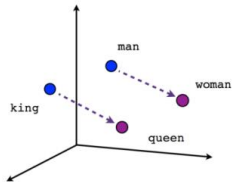


How do data science techniques scale with amount of data?

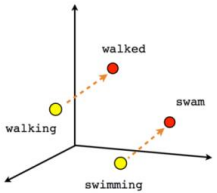
Source: http://blog.operasolutions.com/ai-machine-learning-and-deep-learning-defined

Foundation of Deep NLP models: Word Vectors (Embeddings)

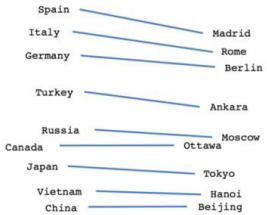
- **Distributed representations of words – basic feature for Deep NLP models**
- **Every word is represented by a fix-sized vector (50d, 100d, 300d, 1024d…)**
- **These vectors learn in an unsupervised fashion on a large corpus of text (Wiki, common crawl, Twitter..)**
- **Learn relations between words:**
  - Man – woman + queen ~ king
  - France – Paris + Warsaw ~ Poland
- **Good people made pretrained embeddings available opensource**
  - Examples: Glove,Word2vec, FastText



Male-Female

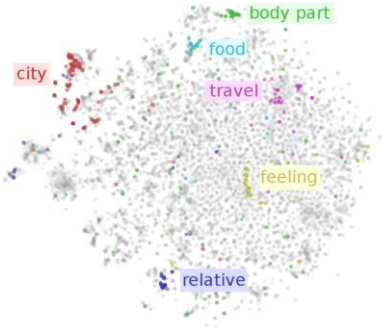


Verb tense



Country-Capital

Source: www.tensorflow.org



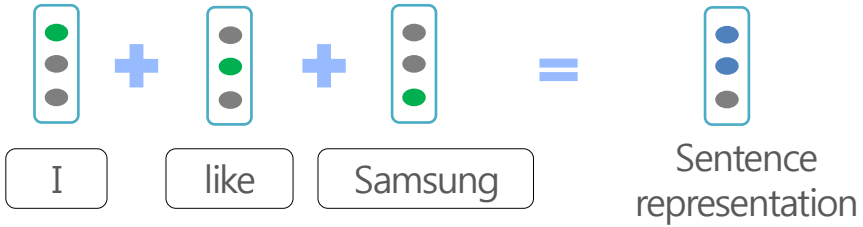
Source: http://ruder.io/word-embeddings-1/

Foundation of Deep NLP models: Word Vectors (Embeddings)

```
>>> from gensim.models.word2vec import Word2Vec

>>> embeddings = Word2Vec.load_word2vec_format('word_vectors.txt', binary=False)

>>> embeddings['vehicle'][:10] # 10 values of a vector of dimensionality 50
array([-0.756091. -1.01268494,  2.04105091,  2.43842196,  2.95695996,
       -0.33063, -1.34891498, -0.251019,  2.78287601,  0.55933303], dtype=float32)

>>> embeddings.similarity('vehicle', 'car') # cosine similarity between two vectors
0.787731
```

Simple baseline: Word Vector sum

- **How to build a representation for the whole text?**
- **Simplest solution: just sum up the embeddings**

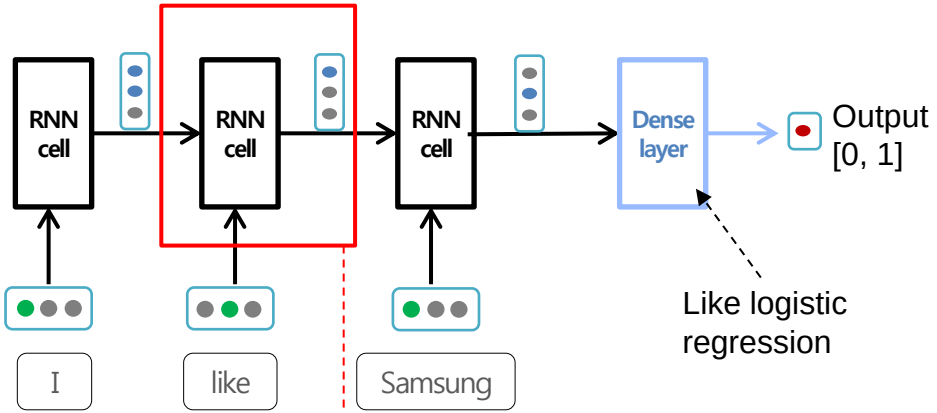


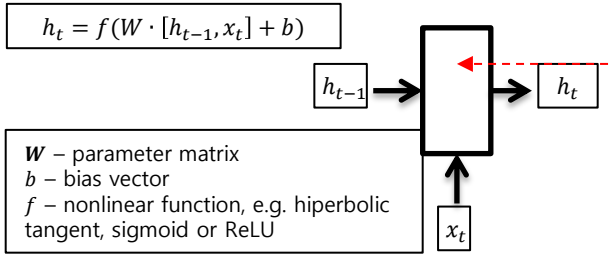
- **And add a classifier on top**
- **Solid baseline**
- **Problems:**
  - Doesn't take word order into account – sometimes it matters:

    The acting was good, but I do not like this movie anyway 😡
    The acting was not good, but I do like this movie anyway 😄
  - Every word is treated with equal importance
  - Composition method is rather poor

Recurrent Neural Networks

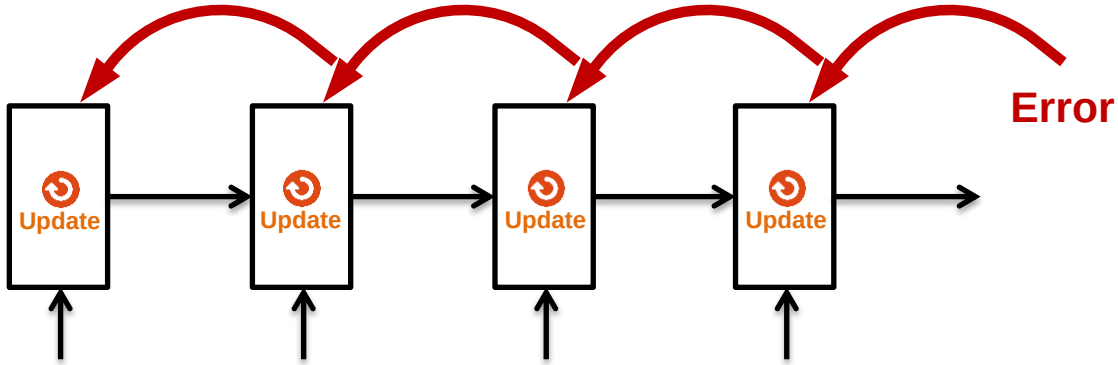
- Enables richer composition of word vectors
- Follows sequential nature of the language





RNN training

- During training, we have to learn both recurrent weights and output layer weights (and maybe word vectors as well), so we can do this jointly
- Similar to logistic regression presented earlier
- Training-> weights update
- Reminder: Predictions from the network are compared with true labels, and the error is used to update weights
- Chain rule is used to compute gradients for the deeper layers
- We start from the „back” (closer to ouput) and propagate error „backwards” -> Backpropagation



SAMSUNG

Case Study

Case study: Disaster Tweets detection from Twitter

- Recall the USGS algorithm for detecting earthquakes – we will try to play with similar problem
- Detect Tweets related to disaster events (earthquake, crash, bomb, fire, attack) as opposed to casual posts
- Binary classification task, will experiment on ~10,000 instance dataset from https://www.figure-eight.com/data-for-everyone/

Disasters (label 1)

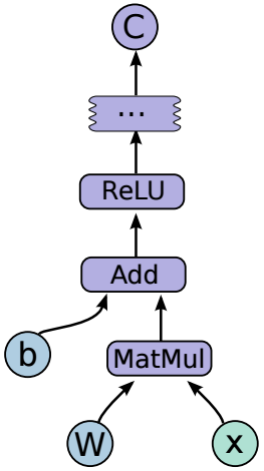
- Just happened a terrible car crash
- #AFRICANBAZE: Breaking news:Nigeria flag set ablaze in Aba. $URL
- Suspect in latest theater attack had psychological issues $URL
- Small casualty on the way to Colorado $URL
- 1 injured in Naples boat explosion $URL
- Rescuers find survivors of Nepal earthquake buried ... $URL

Casual (label 0)

- Hey! How are you?
- @sunkxssedharry will you wear shorts for race ablaze ?
- I'm not gonna lie I'm kinda ready to attack my Senior year ??????????
- Casualty Roleplay somebody please am so bore
- My head exploded i swear
- @lizXy_ IMAGINE IF AN EARTHQUAKE HAPPENED

High level Deep Learning python frameworks

- **High level deep learning frameworks make it easy to build complex architectures from simple building blocks**
- **Pre-defined layers (LSTM, Dense, Embeddings) and other abstractions (Model, Optimizer, Metric)**
- **Automatic differentiation implemented**
  - No need to implement backpropagation… ufff ☺
- **Based on the idea of computational graph**
  - Graph declaration and graph execution phases separated
- **Frameworks:**
  - Tensorflow
  - PyTorch
  - **Keras**



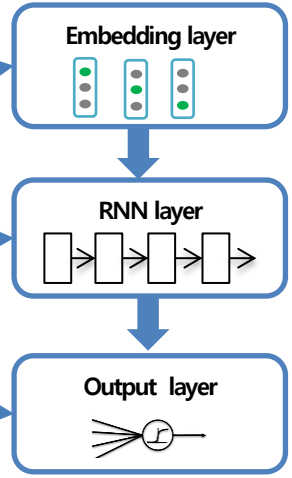
Source: https://medium.com/@asjad/

TensorFlow
PYTORCH
Keras
GLUON
mxnet
Microsoft CNTK
Caffe2
DL4J

Disasters prediction model in Keras

The code

```
model = Sequential()

model.add(Embedding(input_length=sequence_len,
                    input_dim=vocab_size,
                    output_dim=100))

model.add(SimpleRNN(128))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=SGD(),
              metrics=['accuracy'])

model.fit(x=x_train,
          y=y_train,
          validation_data=(x_valid, y_valid),
          batch_size=32,
          epochs=10)
```



Defining loss, metric and optimization algorithm

Actual model training

Let’s see it in action!

- **Jupiter Notebook**
- **Keras docs: https://keras.io/**
- **Troubleshooting:**
  - Progres bar issue: https://github.com/bstriner/keras-tqdm

SAMSUNG

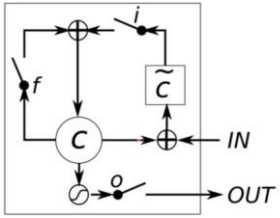
Improvement time!

Advanced architectures: LSTM & GRU

- We can improve our basic RNN by adding a fancy gating mechanism to ignore unimportant noisy words (the, a, #$$#WHJ) and „remember” previous words

LSTM cell

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

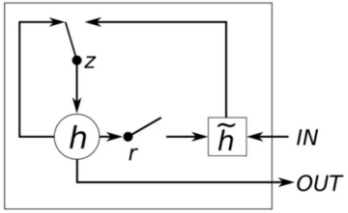


Source: https://deeplearning4j.org/lstm.html

```
from tensorflow.python.keras.layers import LSTM
model.add(LSTM(units=128))
```

GRU cell

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)
\end{aligned}
$$



Source: https://deeplearning4j.org/lstm.html

```
from tensorflow.python.keras.layers import GRU
model.add(GRU(units=128))
```

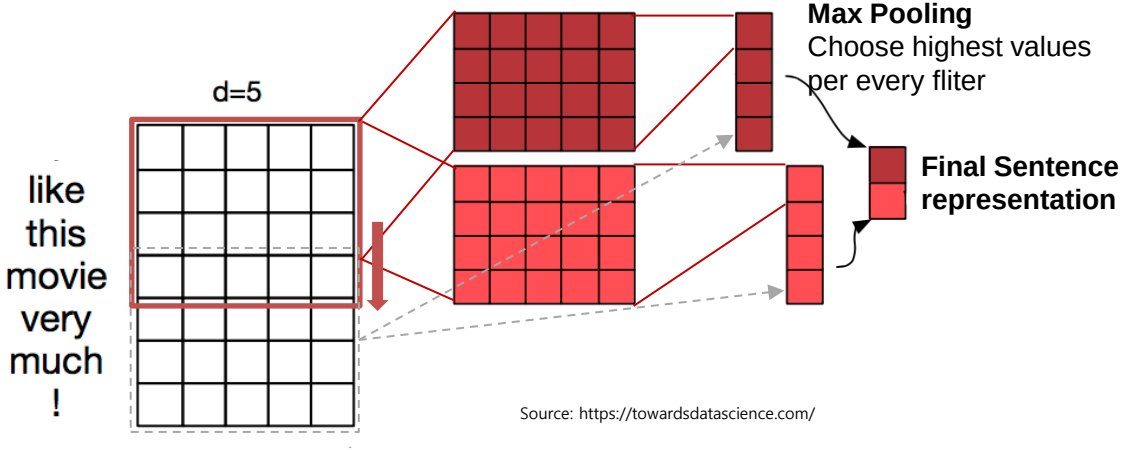
Convolutional Neural Networks (CNN)

- Alternative to Recurrent Networks for computing sentence representation
- Originally used for vision, but successful in NLP as well
- Keras implementation:

```
from tensorflow.python.keras.layers import LSTM
model.add(LSTM(units=128))

from tensorflow.python.keras.layers import Conv1D, \
    GlobalMaxPooling1D
model.add(Conv1D(filters=128,
                 kernel_size=3,
                 padding='valid',
                 activation='relu', strides=1))
model.add(GlobalMaxPooling1D())
```

1D Convolution
2 Filters of size 4

$$c_i = relu(W_{1D} \cdot [x_i, x_{i+1}, x_{i+2}, x_{i+3}])$$



Stacking layers

- Combine multiple layers of RNNs/CNN together
- 2 layer LSTM

```
model.add(LSTM(units=128, return_sequences=True))
model.add(LSTM(units=64))
```

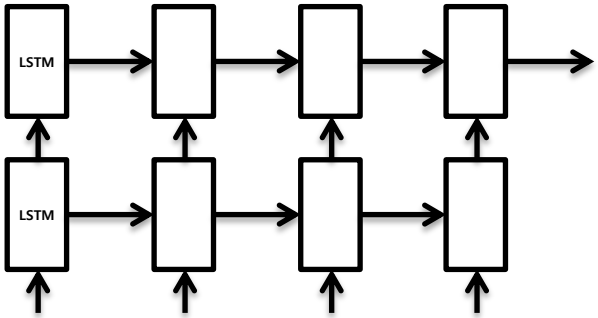
- 2 layer CNN

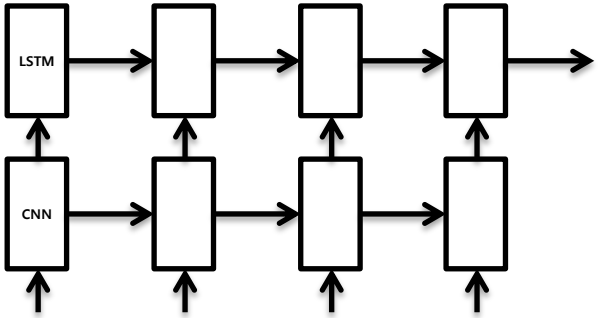
```
model.add(Conv1D(filters=128, kernel_size=3))
model.add(Conv1D(filters=64, kernel_size=5))
model.add(GlobalMaxPooling1D())
```

- LSTM+CNN

```
model.add(LSTM(units=128, return_sequences=True))
model.add(Conv1D(filters=128, kernel_size=3))
model.add(GlobalMaxPooling1D())
```

- And so on ....





SGD and advanced optimization algorithms

Recall: SDG update rule:

$$W_{new} = W_{old} - \alpha \cdot W_{grad}$$

Momentum

- Add running average (~mean) of the gradients to the gradient update

$$m_{new} = \gamma \cdot m_{old} + \alpha \cdot W_{grad}$$
$$W_{new} = W_{old} - m_{new}$$

RMSprop

- Before update, divide gradients by decaying average of squared gradients. (~variance)

$$v_{new} = \gamma \cdot v_{old} + (1 - \gamma) \cdot (W_{grad})^2$$
$$W_{new} = W_{old} - \frac{\alpha}{\sqrt{v_{new} + \epsilon}} \cdot W_{grad}$$

ADAM

- Combine RMSprop with momentum

$$m_{new} = \beta_1 \cdot m_{old} + (1 - \beta_1) \cdot W_{grad}$$
$$v_{new} = \beta_2 \cdot v_{old} + (1 - \beta_2) \cdot (W_{grad})^2$$
$$W_{new} = W_{old} - \frac{\alpha}{\sqrt{v_{new} + \epsilon}} \cdot m_{new}$$

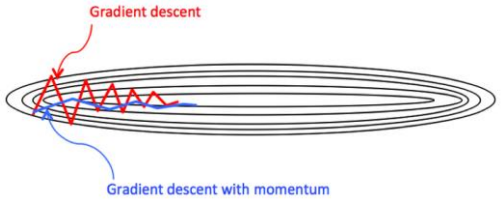
Algorithm	Keras implementation

SGD	`from tensorflow.python.keras.optimizers import SGD`
	`model.compile(optimizer=SGD(lr=0.01),`
	`loss=...)`

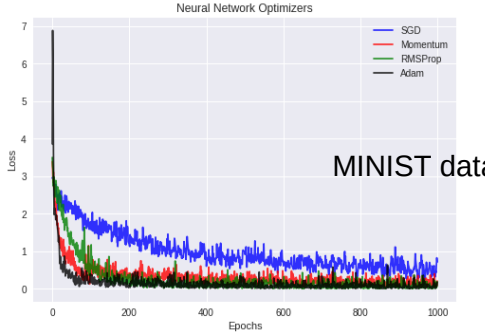
Momentum	`from tensorflow.python.keras.optimizers import SGD`
	`model.compile(optimizer=SGD(lr=0.01, momentum=0.9),`
	`loss=...)`

RMSprop	`from tensorflow.python.keras.optimizers import RMSprop`
	`model.compile(optimizer=RMSprop(lr=0.01),`
	`loss=...)`

Adam	`from tensorflow.python.keras.optimizers import Adam`
	`model.compile(optimizer=Adam(lr=0.01),`
	`loss=...)`



Source: http://mc.ai/



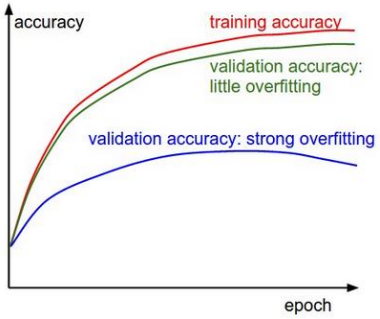
MINIST dataset

Source: https://towardsdatascience.com/

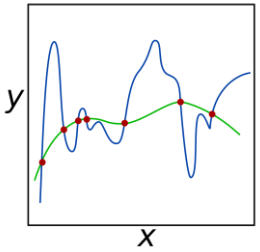
Regularization

Overfitting problem

Means that the model is just memorizing training examples and is not able to generalize well to unseen examples



Source: http://cs231n.github.io/neural-networks-3/

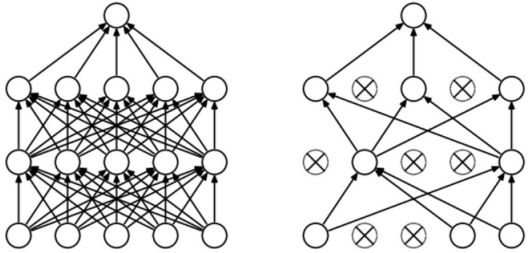


Source: https://en.wikipedia.org/

Dropout

- During training, randomly ignore random fraction of nodes (neurons) in the layer
- Forces network to learn more robust features, not depend on single activations

```
from tensorflow.python.keras.layers import Dropout
model.add(...)
model.add(Dropout(rate=0.01))
model.add(...)
```

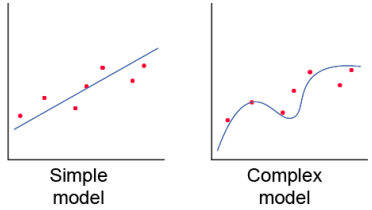


Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

L2 regularization

- Minimize loss as: $L(W) = Error(W) + \lambda \cdot L2(W)$

```
from tensorflow.python.keras.regularizers import l2
LSTM(units=128, activity_regularizer=l2(1e-6))
Dense(units=128, activity_regularizer=l2(1e-6))
Conv1D(filters=128, kernel_size=3, activity_regularizer=l2(1e-7))
```



Source: http://laid.delanover.com/

Batch Normalization

- Normalize outputs of the layer (mean=0, stdev=1)

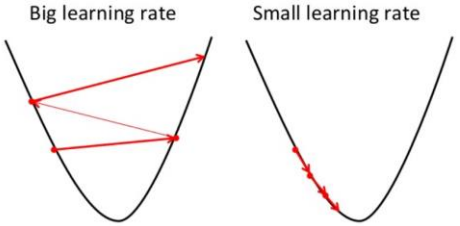
```
from tensorflow.python.keras.layers import BatchNormalization
model.add(...)
model.add(BatchNormalization())
model.add(...)
```

Hyperparameter search

Learning rate

- How big update do we make in SGD

```
model.compile(optimizer=SGD(lr=0.01), loss=...)
```



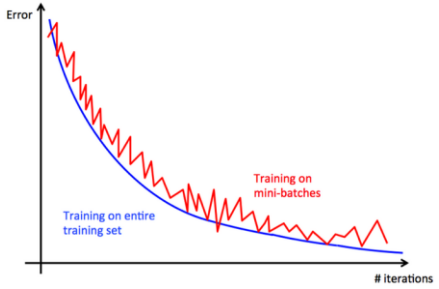
Source: https://www.slideshare.net/simaokasonse/learning-deep-learning

Minibatch size

- How many training samples do we backpropagate before updating the weights

$$W_{new} = W_{old} - \alpha \cdot W_{grad}$$

```
model.fit(x=..., y=..., batch_size=32)
```



Source: http://mc.ai/

Number of layers in RNN/CNN, embedding sizes

```
SimpleRNN(units=128)
Conv1D(units=128, kernel_size=3)
```

Number of training epochs

- How long do we train our model

```
model.fit(x=..., y=..., epochs=20)
```

Other model hyperparameters (https://keras.io/)

SAMSUNG

Further topics

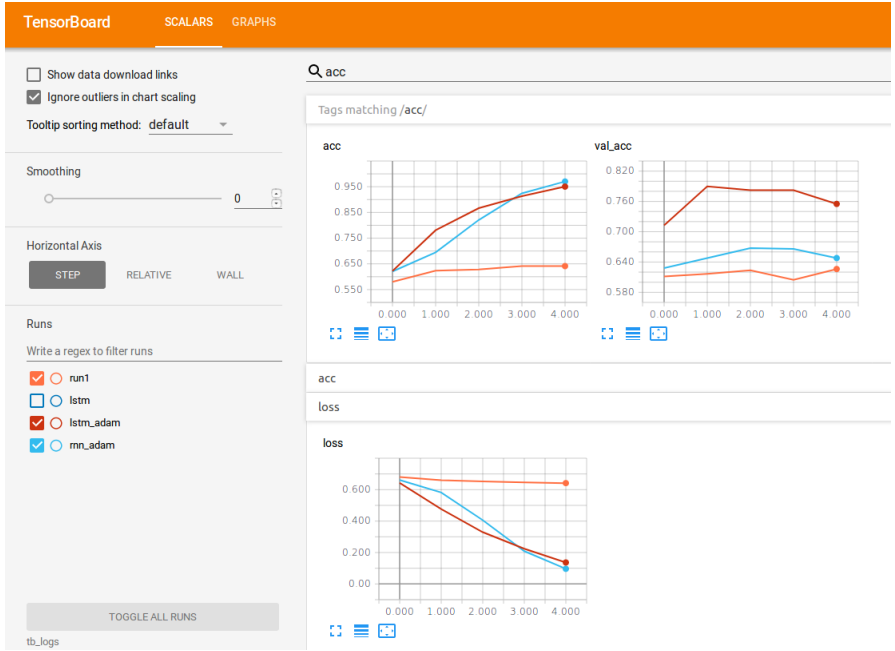
Visualize training with Tensorboard

- Graphical report of the training process
- Save training logs for Tensorboard

```
from tensorflow.python.keras.callbacks import TensorBoard
tb_callback=TensorBoard(log_dir='tb_logs/run1', )
model.fit(x=x_train, y=y_train, validation_data=(x_valid, y_valid),
          batch_size=32, epochs=5,
          callbacks=[tb_callback])
```

- Run Tensorboard

```
13:13 $ tensorboard --logdir=tb_logs
TensorBoard 1.5.1 at http://AMDC2534:6006 (Press CTRL+C to quit)
```



Use pre-trained word vectors

Use pre-trained word vectors file to initialize embeddings in our model

```
an 0.36143 0.58615 -0.23718 0.079656 0.80192 0.49919 -0.33172 -0.197
be 0.91102 -0.22872 0.2077 -0.20237 0.50697 -0.057893 -0.41729 -0.07
has 0.54822 0.038847 0.10127 0.31319 0.095487 0.41814 -0.79493 -0.58
are 0.96193 0.012516 0.21733 -0.06539 0.26843 0.33586 -0.45112 -0.60
have 0.94911 -0.34968 0.48125 -0.19306 -0.0088384 0.28182 -0.9613 -0
but 0.35934 -0.2657 -0.046477 -0.2496 0.54676 0.25924 -0.64458 0.173
were 0.73363 -0.74815 0.45913 -0.56041 0.091855 0.33015 -1.2034 -0.1
not 0.55025 -0.24942 -0.0009386 -0.264 0.5932 0.2795 -0.25666 0.0930
this 0.53074 0.40117 -0.40785 0.15444 0.47782 0.20754 -0.26951 -0.34
```

Unzip word vectors file, load word vectors

```
def load_word_vectors(tokenizer, vocab_size):
    vectors_path = os.path.join('data', 'twitter.glove.100d.txt')
    embedding_matrix = np.zeros(shape=(vocab_size, 100))
    with open(vectors_path, encoding='utf8') as wv_file:
        for line in tqdm(list(wv_file)):
            ...
    return embedding_matrix
```

Intialize embedding layer

```
embeddings = load_word_vectors(tokenizer, vocab_size)

model.add(Embedding(input_length=sequence_len, input_dim=vocab_size, output_dim=100,
                    weights=[embeddings], trainable=True))
```

SAMSUNG

Thank you!

Maciej Pieńkosz
m.pienkosz@samsung.com
NLP Research Group
Samsung R&D Poland