

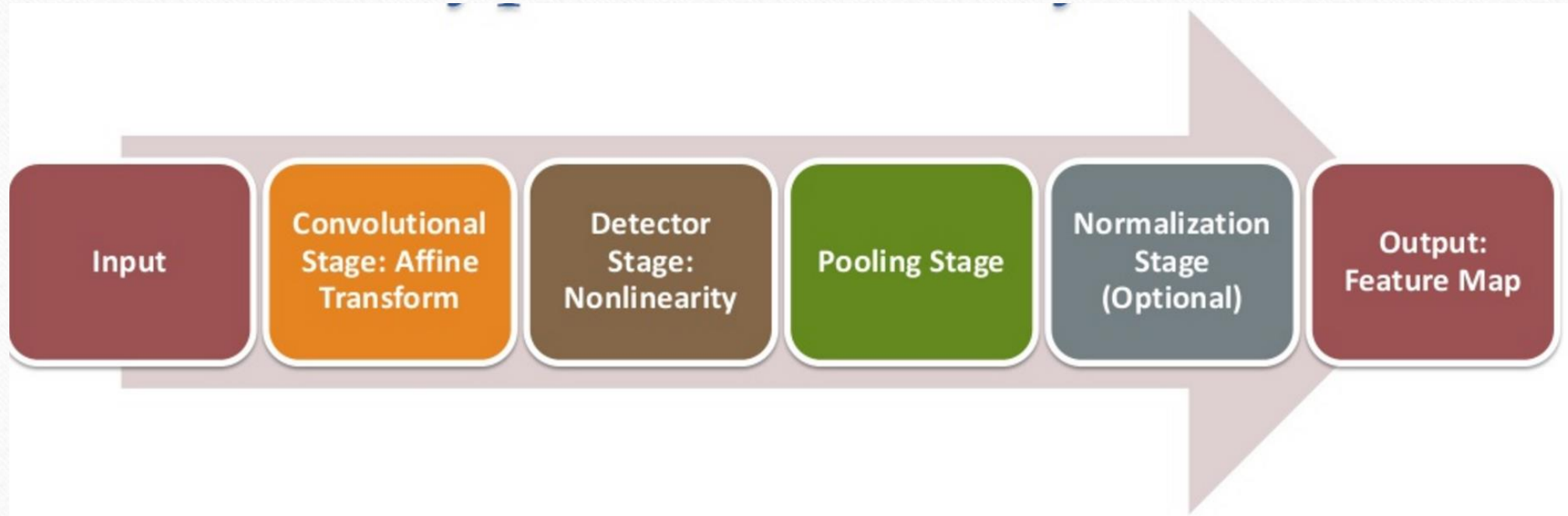
CONVOLUTIONAL NEURAL NETWORK

Subject: Advances in Data Sciences

What are CNNs

Convolutional Neural network is a type of feed forward artificial neural network in which connectivity pattern between neurons is inspired by the organization of animal visual cortex.

Typical CNN layer



4 Principle Terms of CNN

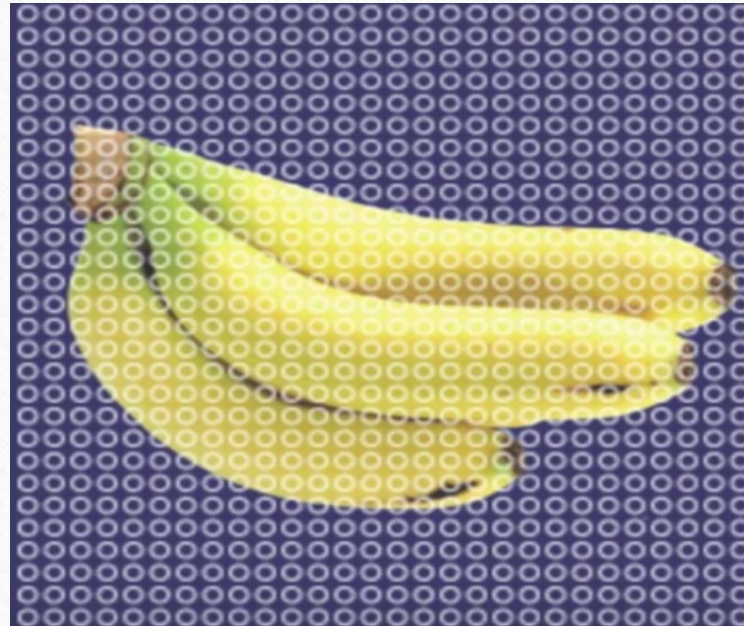
- Local Receptive field: a window on the input of pixels
- Feature Maps: mapping from input to hidden layers
- Shared Weights: positive or negatives
- Pooling: simplifying the information from the feature maps

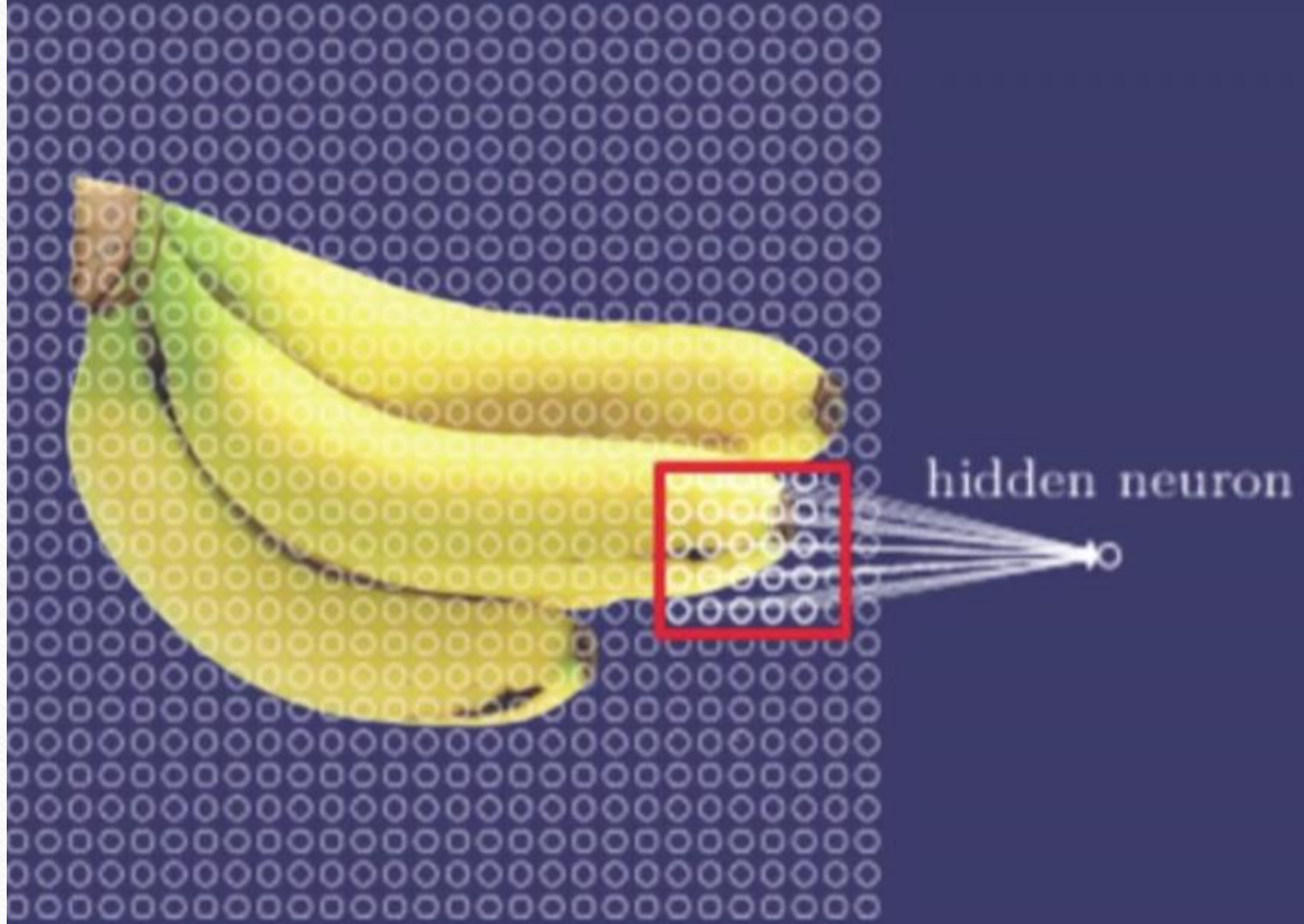
Architecture of CNN

- CNN is a stack of distinct layers that transform the input volume into an output volume through a differential function.
- The layers of CNN are
 1. Convolutional layer
 2. Pooling Layer
 3. Relu Layer
 4. Fully Connected Layer
 5. Loss layer

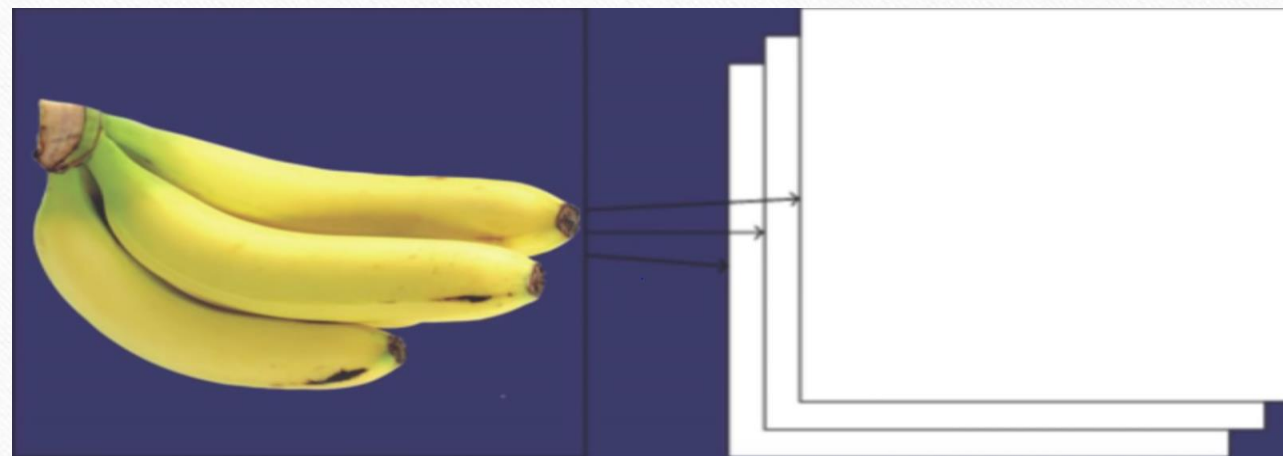
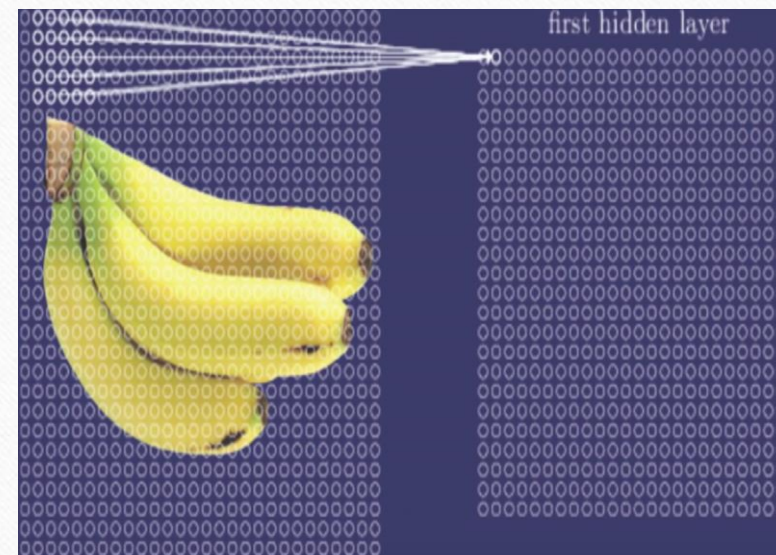
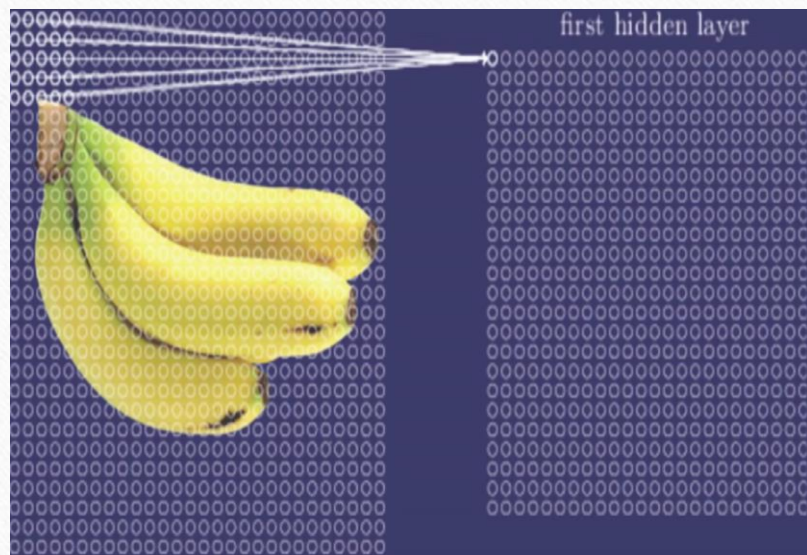
Working of CNN

- Consider picture of Banana divided in terms of neurons rather than pixels.



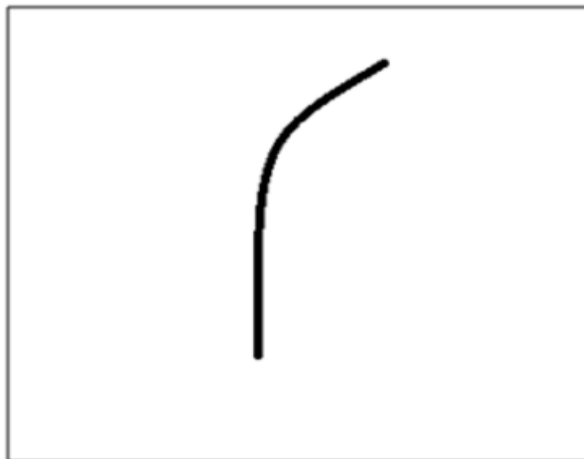


Local Receptive Field



0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter



Visualization of the receptive field

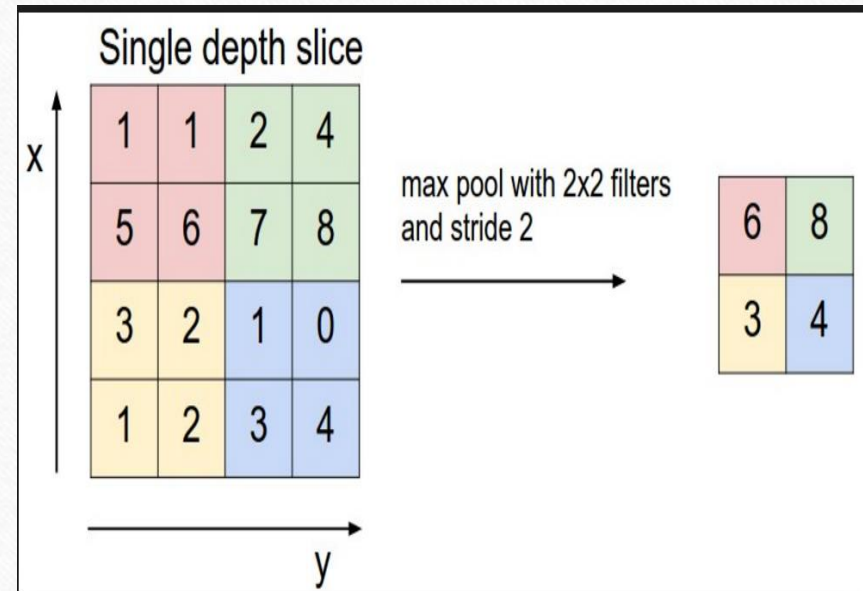
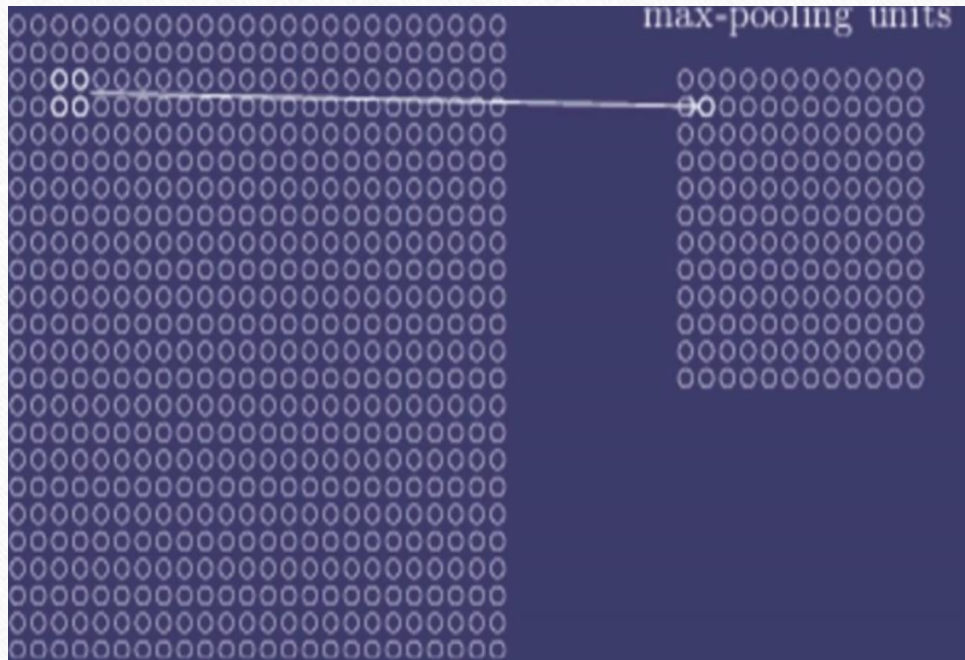
0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



ReLU layer

- ReLU is the abbreviation of Rectified Linear Units, the **rectifier** is an activation function defined as $f(x) = \max(0, x)$.
- This is a layer of neurons that applies the non-saturating activation function.
- It increases the non linear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Fully connected layer

- Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers.
- Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks.
- Their activations can hence be computed with a matrix multiplication followed by a bias offset

LOSS LAYER

- The loss layer specifies how the network training penalizes the deviation between the predicted and true labels and is normally the last layer in the network.
- Various loss functions appropriate for different tasks may be used there.
 - Softmax loss is used for predicting a single class of K mutually exclusive classes.
 - Sigmoid cross-entropy loss is used for predicting K independent probability values in $[0,1]$.
 - Euclidean loss is used for regressing to real-valued labels

Convolutional Neural Networks for Sentence Classification

Requirements

Code is written in Python (2.7) and requires Theano (0.7).

Using the pre-trained word2vec vectors will also require downloading the binary file from

<https://code.google.com/p/word2vec/>

Architecture

- Pad input sentences so that they are of the same length.
- Map words in the padded sentence using word embeddings (which may be either initialized as zero vectors or initialized as word2vec embeddings) to obtain a matrix corresponding to the sentence.
- Apply convolution layer with multiple filter widths and feature maps.
- Apply max-over-time pooling operation over the feature map.
- Concatenate the pooling results from different layers and feed to a fully-connected layer with softmax activation.
- Softmax outputs probabilistic distribution over the labels.
- Use dropout for regularisation.

Hyperparameters

- RELU activation for convolution layers
- Filter window of 3, 4, 5 with 100 feature maps each.
- Dropout - 0.5
- Gradient clipping at 3
- Batch size - 50
- Adadelata update rule.

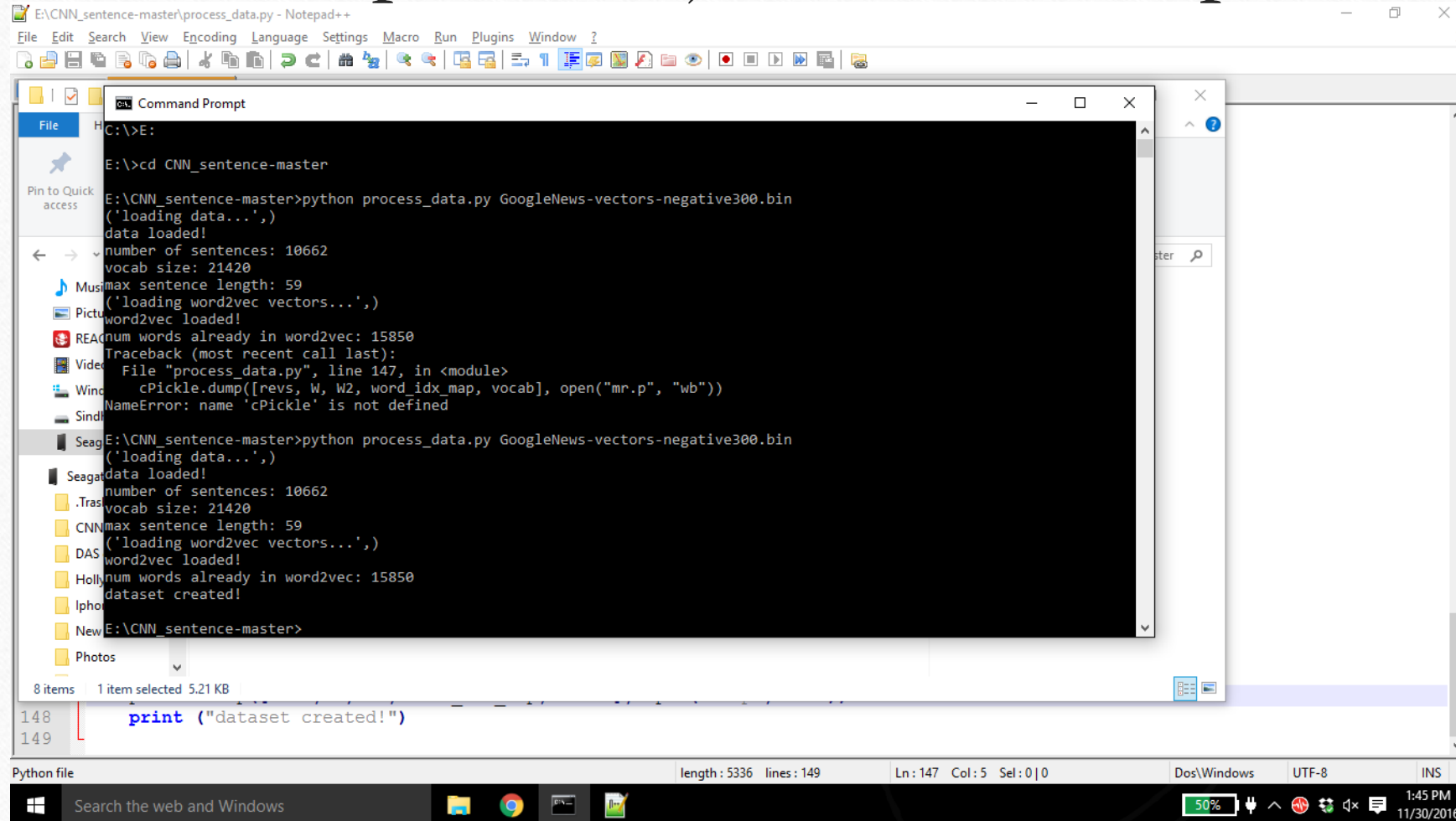
Variants

- CNN-rand
 - Randomly initialized word vectors.
- CNN-static
 - Uses pre-trained vectors from word2vec and does not update the word vectors.
- CNN-non-static
 - Same as CNN-static but updates word vectors during training.

Data Preprocessing

To process the raw data, run
`python process_data.py path`
where `path` points to the word2vec binary file
(i.e. `GoogleNews-vectors-negative300.bin` file).
This will create a pickle object called
`mr.p` in the same folder, which contains the dataset in the
right format.

Here a pickle object named mr.p is created



```
E:\CNN_sentence-master\process_data.py - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

C:\>E:
E:\>cd CNN_sentence-master
E:\CNN_sentence-master>python process_data.py GoogleNews-vectors-negative300.bin
('loading data...')
data loaded!
number of sentences: 10662
vocab size: 21420
max sentence length: 59
('loading word2vec vectors...')
word2vec loaded!
num words already in word2vec: 15850
Traceback (most recent call last):
  File "process_data.py", line 147, in <module>
    cPickle.dump([revs, W, W2, word_idx_map, vocab], open("mr.p", "wb"))
NameError: name 'cPickle' is not defined
E:\CNN_sentence-master>python process_data.py GoogleNews-vectors-negative300.bin
('loading data...')
data loaded!
number of sentences: 10662
vocab size: 21420
max sentence length: 59
('loading word2vec vectors...')
word2vec loaded!
num words already in word2vec: 15850
dataset created!
E:\CNN_sentence-master>
```

148 print ("dataset created!")

149

Python file length: 5336 lines: 149 Ln: 147 Col: 5 Sel: 0 | 0 Dos\Windows UTF-8 INS

Search the web and Windows 50% 1:45 PM 11/30/2016

Command Prompt - python conv_net_sentence.py -static -word2vec

Microsoft Windows [Version 10.0.10586]

(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Sindhuri Yamala>cd/

C:\>E:

E:\>cd CNN

E:\CNN>cd CNN_sentence-master

E:\CNN\CNN_sentence-master>python conv_net_sentence.py -static -word2vec

loading data... data loaded!

model architecture: CNN-static

using: word2vec vectors

[('image_shape', 64, 300), ('filter_shape', [(100, 1, 3, 300), (100, 1, 4, 300), (100, 1, 5, 300)]), ('hidden_units', [100, 2]), ('dropout', [0.5]), ('batch_size', 50), ('non_static', False), ('learn_decay', 0.95), ('conv_non_linear', 'relu'), ('non_static', False), ('sqr_norm_lim', 9), ('shuffle_batch', True)]

... training

epoch: 1, training time: 396.90 secs, train perf: 80.01 %, val perf: 76.42 %

epoch: 2, training time: 397.74 secs, train perf: 81.64 %, val perf: 75.16 %

epoch: 3, training time: 405.81 secs, train perf: 84.72 %, val perf: 75.89 %

cv: 0, perf: 0.755491881566

[('image_shape', 64, 300), ('filter_shape', [(100, 1, 3, 300), (100, 1, 4, 300), (100, 1, 5, 300)]), ('hidden_units', [100, 2]), ('dropout', [0.5]), ('batch_size', 50), ('non_static', False), ('learn_decay', 0.95), ('conv_non_linear', 'relu'), ('non_static', False), ('sqr_norm_lim', 9), ('shuffle_batch', True)]

... training

epoch: 1, training time: 394.61 secs, train perf: 80.43 %, val perf: 77.79 %

epoch: 2, training time: 400.04 secs, train perf: 83.67 %, val perf: 79.58 %

epoch: 3, training time: 413.10 secs, train perf: 86.65 %, val perf: 79.16 %

cv: 1, perf: 0.779069767442

[('image_shape', 64, 300), ('filter_shape', [(100, 1, 3, 300), (100, 1, 4, 300), (100, 1, 5, 300)]), ('hidden_units', [100, 2]), ('dropout', [0.5]), ('batch_size', 50), ('non_static', False), ('learn_decay', 0.95), ('conv_non_linear', 'relu'), ('non_static', False), ('sqr_norm_lim', 9), ('shuffle_batch', True)]

... training

-



Search the web and Windows



91%



4:20 PM
12/1/2016

Command Prompt - python conv_net_sentence.py -nonstatic -word2vec

C:\>E:

E:\>cd CNN_sentence-master

E:\CNN_sentence-master>python conv_net_sentence.py -nonstatic -word2vec

loading data... data loaded!

architecture: CNN-non-static

using: word2vec vectors

[('image shape', 64, 300), ('filter shape', [(100, 1, 3, 300), (100, 1, 4, 300), (100, 1, 5, 300)]), ('hidden_units', [100, 2]), ('dropout', [0.5]), ('batch_size', 50), ('non_static', True), ('learn_decay', 0.95), ('conv_non_linear', 'relu'), ('non_static', True), ('sqr_norm_lim', 9), ('shuffle_batch', True)]

... training

epoch: 1, training time: 661.18 secs, train perf: 81.25 %, val perf: 76.63 %
epoch: 2, training time: 555.72 secs, train perf: 84.32 %, val perf: 76.11 %
epoch: 3, training time: 550.26 secs, train perf: 89.57 %, val perf: 77.05 %
epoch: 4, training time: 937.48 secs, train perf: 95.24 %, val perf: 80.42 %
epoch: 5, training time: 557.37 secs, train perf: 97.62 %, val perf: 80.21 %
epoch: 6, training time: 556.07 secs, train perf: 99.18 %, val perf: 80.53 %
epoch: 7, training time: 557.54 secs, train perf: 99.34 %, val perf: 80.42 %
epoch: 8, training time: 555.48 secs, train perf: 99.83 %, val perf: 81.47 %
epoch: 9, training time: 556.70 secs, train perf: 99.93 %, val perf: 79.89 %
epoch: 10, training time: 556.90 secs, train perf: 99.92 %, val perf: 80.32 %
epoch: 11, training time: 559.10 secs, train perf: 99.98 %, val perf: 80.74 %
epoch: 12, training time: 558.41 secs, train perf: 99.99 %, val perf: 80.53 %
epoch: 13, training time: 581.92 secs, train perf: 99.99 %, val perf: 80.53 %
epoch: 14, training time: 641.78 secs, train perf: 99.99 %, val perf: 80.53 %
epoch: 15, training time: 734.45 secs, train perf: 99.99 %, val perf: 80.84 %
epoch: 16, training time: 746.65 secs, train perf: 99.99 %, val perf: 80.11 %
epoch: 17, training time: 740.15 secs, train perf: 99.99 %, val perf: 80.21 %
epoch: 18, training time: 741.00 secs, train perf: 99.99 %, val perf: 80.11 %
epoch: 19, training time: 733.91 secs, train perf: 99.99 %, val perf: 79.89 %
epoch: 20, training time: 749.15 secs, train perf: 100.00 %, val perf: 79.58 %



Search the web and Windows



4:20 PM
12/1/2016

Select Command Prompt - python conv_net_sentence.py -nonstatic -rand

epoch: 25, training time: 465.77 secs, train perf: 100.00 %, val perf: 75.58 %

cv: 2, perf: 0.757326007326

[('image shape', 64, 300), ('filter shape', [(100, 1, 3, 300), (100, 1, 4, 300), (100, 1, 5, 300)]), ('hidden_units', [100, 2]), ('dropout', [0.5]), ('batch_size', 50), ('non_static', True), ('learn_decay', 0.95), ('conv_non_linear', 'relu'), ('non_static', True), ('sqr_norm_lim', 9), ('shuffle_batch', True)]

... training

epoch: 1, training time: 467.74 secs, train perf: 62.74 %, val perf: 61.05 %

epoch: 2, training time: 468.99 secs, train perf: 66.79 %, val perf: 59.89 %

epoch: 3, training time: 467.68 secs, train perf: 84.94 %, val perf: 68.74 %

epoch: 4, training time: 467.64 secs, train perf: 93.05 %, val perf: 71.58 %

epoch: 5, training time: 467.38 secs, train perf: 97.37 %, val perf: 72.95 %

epoch: 6, training time: 467.46 secs, train perf: 98.30 %, val perf: 75.47 %

epoch: 7, training time: 467.28 secs, train perf: 99.68 %, val perf: 73.47 %

epoch: 8, training time: 467.62 secs, train perf: 99.91 %, val perf: 74.21 %

epoch: 9, training time: 467.45 secs, train perf: 99.97 %, val perf: 73.16 %

epoch: 10, training time: 467.56 secs, train perf: 99.97 %, val perf: 73.79 %

epoch: 11, training time: 467.50 secs, train perf: 100.00 %, val perf: 72.00 %

epoch: 12, training time: 467.38 secs, train perf: 100.00 %, val perf: 74.53 %

epoch: 13, training time: 467.79 secs, train perf: 100.00 %, val perf: 74.95 %

epoch: 14, training time: 467.02 secs, train perf: 100.00 %, val perf: 74.21 %

epoch: 15, training time: 467.34 secs, train perf: 100.00 %, val perf: 74.63 %

epoch: 16, training time: 468.07 secs, train perf: 100.00 %, val perf: 75.89 %

epoch: 17, training time: 467.81 secs, train perf: 100.00 %, val perf: 74.32 %

epoch: 18, training time: 467.51 secs, train perf: 100.00 %, val perf: 75.26 %

epoch: 19, training time: 467.58 secs, train perf: 100.00 %, val perf: 75.37 %

epoch: 20, training time: 466.66 secs, train perf: 100.00 %, val perf: 75.89 %

epoch: 21, training time: 467.43 secs, train perf: 100.00 %, val perf: 75.89 %

epoch: 22, training time: 466.95 secs, train perf: 100.00 %, val perf: 75.58 %

epoch: 23, training time: 467.42 secs, train perf: 100.00 %, val perf: 75.58 %

epoch: 24, training time: 467.43 secs, train perf: 100.00 %, val perf: 74.74 %

epoch: 25, training time: 467.44 secs, train perf: 100.00 %, val perf: 75.05 %

cv: 3, perf: 0.741362290227

[('image shape', 64, 300), ('filter shape', [(100, 1, 3, 300), (100, 1, 4, 300), (100, 1, 5, 300)]), ('hidden_units', [100, 2]), ('dropout', [0.5]), ('batch_size', 50), ('non_static', True), ('learn_decay', 0.95), ('conv_non_linear', 'relu'), ('non_static', True), ('sqr_norm_lim', 9), ('shuffle_batch', True)]

... training

epoch: 1, training time: 467.75 secs, train perf: 60.63 %, val perf: 57.26 %

epoch: 2, training time: 468.05 secs, train perf: 72.21 %, val perf: 64.63 %

epoch: 3, training time: 1954.90 secs, train perf: 79.16 %, val perf: 67.26 %



Search the web and Windows



90%



4:21 PM
12/1/2016

References

https://github.com/yonkim/CNN_sentence

<https://arxiv.org/abs/1408.5882>

<https://arxiv.org/pdf/1408.5882v2.pdf>

Thank You.
