Example pattern extraction from text with a convolutional neural network

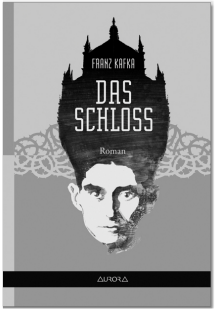
ch

2018

internal

https://github.com/chambrock/bob-andrews

input data



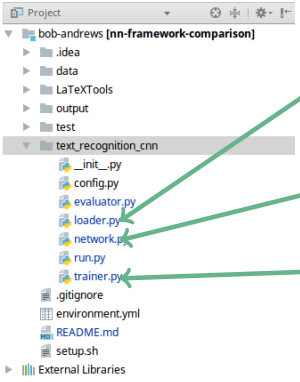
e- eine intrigante natur- scheinbar sinnlos arbeitend wie der wind- nach fernen- fremden aufträgen- in die man nie einsicht bekam---kaum hatte er ein paar schritte auf der landstraSSe gemacht- als er i, truth:0.0

nn man in seinem gedankengange bleibt- nur die einzige wirkliche amtliche verbindung helfen- die er mit klamm hat- also dieses protokoll- nur dieses habe ich gesagt- und wer etwas anderes behauptet- v, truth:0.0

r es still- fritz erkundigte sich drüben- und hier wartete man auf die antwort- k- blieb wie bisher- drehte sich nicht einmal um- schien gar nicht neugierig- sah vor sich hin- die erzählung schwarzers, truth:1.0

- extract 'sentences' (200 letters each) and match pattern with reegex

project structure



x, y_{truth}

$n(w, x) \rightarrow y$

$\text{loss}(n(w, x), y_{\text{truth}}) \rightarrow w|_{@\min}$

standard convolution examples

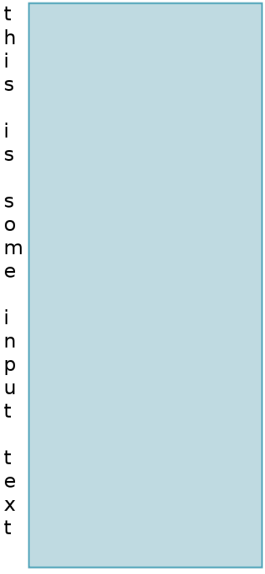
No padding, no strides

No padding but strides

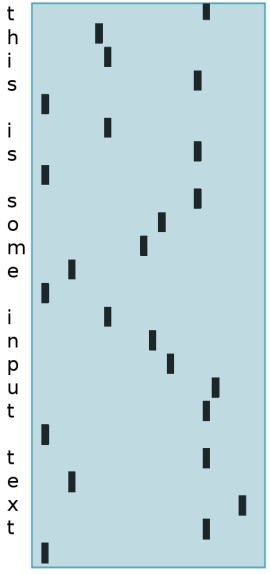
Arbitrary padding, no strides

padding and strides

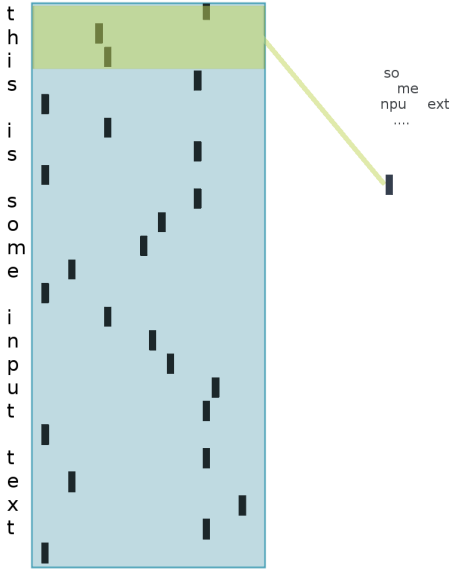
convolutions example



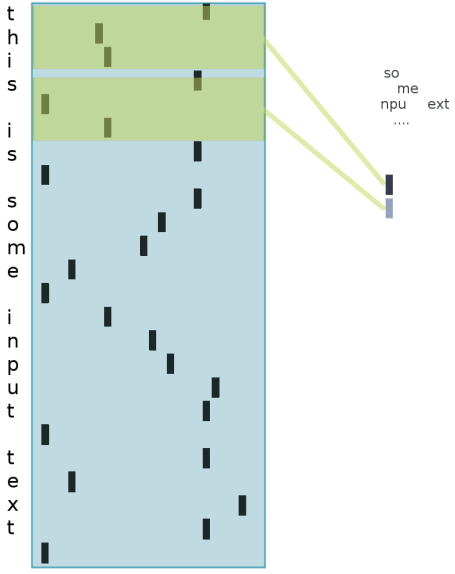
convolutions example



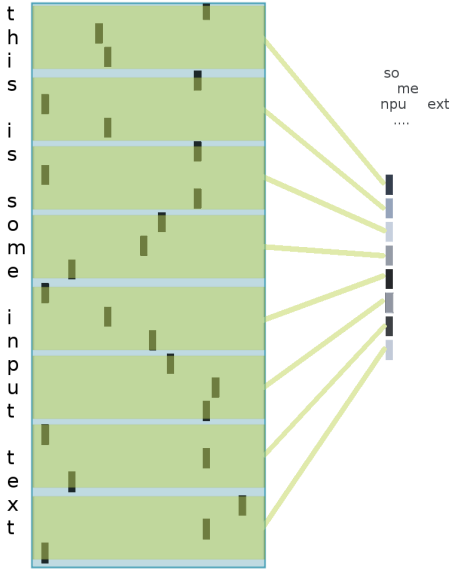
convolutions example



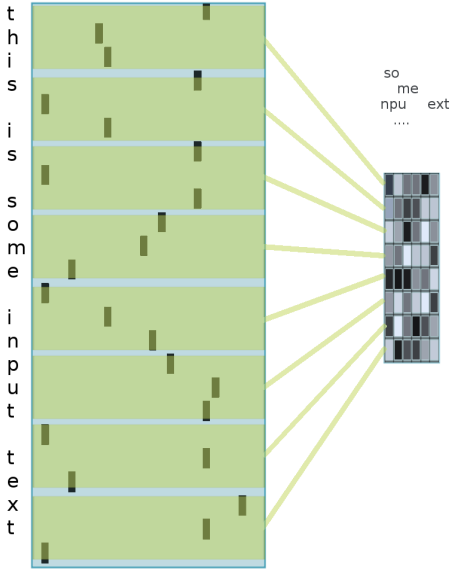
convolutions example



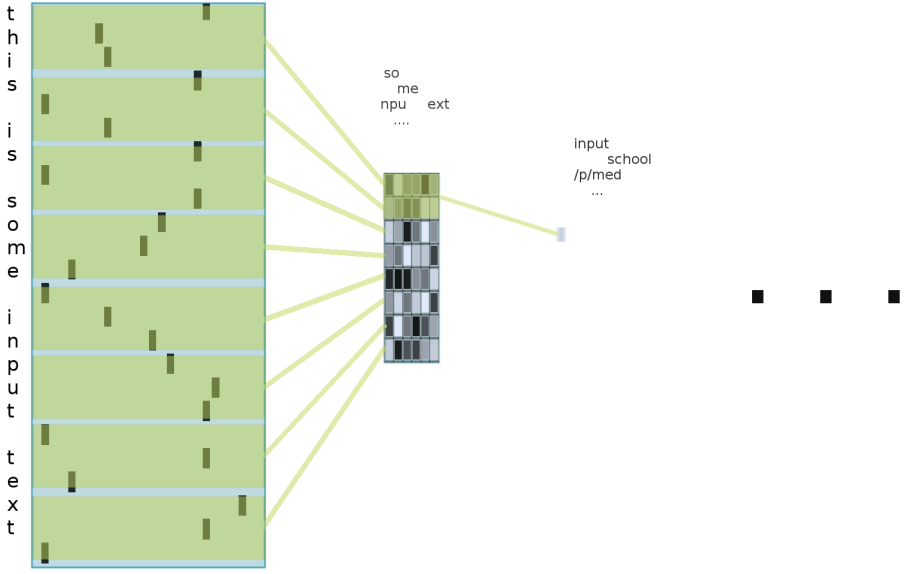
convolutions example



convolutions example



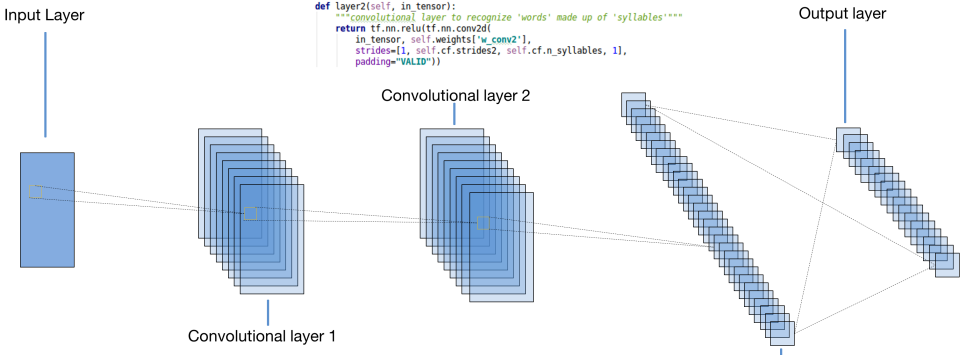
convolutions example



network

```python
def layer0(self, in_tensor):
    """Reshape to make standard input (picture recognition), last is the possible color channel"""
    return tf.reshape(in_tensor, shape=[-1, self.cf.string_length, self.cf.n_chars, 1])
```

```python
def layer4(self, in_tensor):
    """output feed forward layer"""
    return tf.nn.sigmoid(tf.matmul(in_tensor, self.weights['out']) + self.biases['out'])
```




```python
def layer2(self, in_tensor):
    """convolutional layer to recognize 'words' made up of 'syllables'"""
    return tf.nn.relu(tf.nn.conv2d(
        in_tensor, self.weights['w_conv2'],
        strides=[1, self.cf.strides2, self.cf.n_syllables, 1],
        padding="VALID"))
```

```python
def layer1(self, in_tensor):
    """convolutional layer to recognize 'syllables'"""
    return tf.nn.relu(tf.nn.conv2d(
        in_tensor,
        self.weights['w_conv1'],
        strides=[1, self.cf.strides1, self.cf.n_chars, 1],
        padding="VALID"))
```

```python
def layer3(self, in_tensor):
    """flatten everything and make a standard feed forward layer"""
    fc_input = tf.reshape(in_tensor, [-1, self.len_layer2_out * self.cf.n_words])
    return tf.matmul(fc_input, self.weights['w_fc']) + self.biases['b_fc']
```

step through code

1 datamart

run.py

```
import ...

output_map_batch = {"batch_count": [], "accuracy_train": [], "cost_train": []}
output_map_epoch = {"epoch": [], "batch_count": [], "cost_test": [], "accuracy_test": []}

cf = cf.Config()
tex_writer = TeXwriter(".././output", "doc")
tex_writer.addSection("Parameters")
tex_writer.addText(cf.to_tex())
loader = ld.Loader(cf)
char_trf = loader.char_trf
network = nw.Network(cf)
trainer = tn.Trainer(cf, network)
evaluator = ev.Evaluator(cf, network)

with tf.Session() as sess:

    sess.run(tf.global_variables_initializer())

    while loader.epochs < cf.epochs:
        batch_x, batch_y = loader.get_next_train_batch(cf.batch_size, shuffle=cf.shuffle)
        current_output = trainer.train(sess, batch_x, batch_y)

        output_map_batch["batch_count"].append(loader.batches)
        output_map_batch["accuracy_train"].append(current_output[3])
        output_map_batch["cost_train"].append(current_output[1])

        if loader.new_epoch:
            x, y = loader.get_test_data()
            current_test_output = trainer.test(sess, x, y)
```

step through code

1. datamart
2. network ini

run.py

```
import ...

output_map_batch = {"batch_count": [], "accuracy_train": [], "cost_train": []}
output_map_epoch = {"epoch": [], "batch_count": [], "cost_test": [], "accuracy_test": []}

cf = cf.Config()
tex_writer = TeXwriter(".././output", "doc")
tex_writer.addSection("Parameters")
tex_writer.addText(cf.to_tex())
loader = ld.Loader(cf)
char_trf = loader.char_trf
network = nw.Network(cf)
trainer = tn.Trainer(cf, network)
evaluator = ev.Evaluator(cf, network)

with tf.Session() as sess:

    sess.run(tf.global_variables_initializer())

    while loader.epochs < cf.epochs:
        batch_x, batch_y = loader.get_next_train_batch(cf.batch_size, shuffle=cf.shuffle)
        current_output = trainer.train(sess, batch_x, batch_y)

        output_map_batch["batch_count"].append(loader.batches)
        output_map_batch["accuracy_train"].append(current_output[3])
        output_map_batch["cost_train"].append(current_output[1])

        if loader.new_epoch:
            x, y = loader.get_test_data()
            current_test_output = trainer.test(sess, x, y)
```

step through code

1. datamart
2. network ini

network.py

```python
import tensorflow as tf

class Network:
    """dims: [batch, height, width, channels]"""
    weights = None
    biases = None

    def __init__(self, config):
        self.cf = config

        self.len_layer1_out = self._how_many_strides_fit(self.cf.string_length, self.cf.syllable_length, self.cf.str
        self.len_layer2_out = self._how_many_strides_fit(self.len_layer1_out, self.cf.word_length, self.cf.strides2)

        self._initialize_params_()

    def _how_many_strides_fit(self, tot_length, window_size, stride_width):
        """calculate how many non-padded strides fit"""
        return int((tot_length - window_size) / stride_width) + 1

    def _initialize_params_(self):
        """initialize all parameters which are later optimized by tensorflow"""
        rand_std = 0.01

        self.weights = {
                'w_conv1': tf.Variable(tf.random_normal(
                    [self.cf.syllable_length, self.cf.n_chars, 1, self.cf.n_syllables],
                    stddev=rand_std)),
                'w_conv2': tf.Variable(tf.random_normal(
                    [self.cf.word_length, 1, self.cf.n_syllables, self.cf.n_words],
                    stddev=rand_std)),
                'w_fc': tf.Variable(tf.random_normal(
                    [self.len_layer2_out * self.cf.n_words, self.cf.output_number],
                    stddev=rand_std)),
                'out': tf.Variable(tf.random_normal(
                    [self.cf.output_number, self.cf.n_classes],
                    stddev=rand_std))
            }

        self.biases = {
                'b_fc': tf.Variable(tf.random_normal([self.cf.output_number])),
                'out': tf.Variable(tf.random_normal([self.cf.n_classes]))
            }
```

step through code

1. datamart
2. network ini
3. graph ini

run.py

```
import ...

output_map_batch = {"batch_count": [], "accuracy_train": [], "cost_train": []}
output_map_epoch = {"epoch": [], "batch_count": [], "cost_test": [], "accuracy_test": []}

cf = cf.Config()
tex_writer = TeXwriter(".././output", "doc")
tex_writer.addSection("Parameters")
tex_writer.addText(cf.to_tex())
loader = ld.Loader(cf)
char_trf = loader.char_trf
network = nw.Network(cf)
trainer = tn.Trainer(cf, network)
evaluator = ev.Evaluator(cf, network)

with tf.Session() as sess:

    sess.run(tf.global_variables_initializer())

    while loader.epochs < cf.epochs:
        batch_x, batch_y = loader.get_next_train_batch(cf.batch_size, shuffle=cf.shuffle)
        current_output = trainer.train(sess, batch_x, batch_y)

        output_map_batch["batch_count"].append(loader.batches)
        output_map_batch["accuracy_train"].append(current_output[3])
        output_map_batch["cost_train"].append(current_output[1])

        if loader.new_epoch:
            x, y = loader.get_test_data()
            current_test_output = trainer.test(sess, x, y)
```

step through code

1. datamart
2. network ini
3. graph ini

trainer.py

```python
class Trainer(object):

    def __init__(self, config, network):
        self.cf = config
        self.network = network

        self.x = None
        self.y = None
        self.train_output = None
        self.test_output = None
        self.initialize_graph()

    def initialize_graph(self):
        """build graph"""
        self.x = tf.placeholder(tf.float32, [None, self.cf.string_length, self.cf.n_chars])
        self.y = tf.placeholder(tf.float32, [None, self.cf.n_classes])

        self.pred = self.network.predict(self.x)
        self.cost = self._loss_(self.pred, self.y)
        self.optimizer = tf.train.AdamOptimizer(learning_rate=self.cf.learning_rate).minimize(self.cost)

        correct = tf.equal(tf.argmax(self.pred, 1), tf.argmax(self.y, 1))
        self.accuracy = tf.reduce_mean(tf.cast(correct, 'float'))

    def _loss_(self, prediction, y):
        """loss function"""
        #single_loss = tf.nn.softmax_cross_entropy_with_logits(logits=prediction, labels=y)
        single_loss = tf.pow(prediction - y, 2)
        return tf.reduce_mean(single_loss)
```

step through code

1. datamart
2. network ini
3. graph ini

network.py

```python
def predict(self, input_features):
    """glue together all layers to transform features to predictions"""
    out_layer0 = self.layer0(input_features)
    out_layer1 = self.layer1(out_layer0)
    out_layer2 = self.layer2(out_layer1)
    out_layer3 = self.layer3(out_layer2)
    out_layer4 = self.layer4(out_layer3)

    return out_layer4

def layer0(self, in_tensor):
    """Reshape to make standard input (picture recognition), last is the possible color channel"""
    return tf.reshape(in_tensor, shape=[-1, self.cf.string_length, self.cf.n_chars, 1])

def layer1(self, in_tensor):
    """convolutional layer to recognize 'syllables'"""
    return tf.nn.relu(tf.nn.conv2d(
        in_tensor,
        self.weights['w_conv1'],
        strides=[1, self.cf.strides1, self.cf.n_chars, 1],
        padding="VALID"))

def layer2(self, in_tensor):
    """convolutional layer to recognize 'words' made up of 'syllables'"""
    return tf.nn.relu(tf.nn.conv2d(
        in_tensor, self.weights['w_conv2'],
        strides=[1, self.cf.strides2, self.cf.n_syllables, 1],
        padding="VALID"))

def layer3(self, in_tensor):
    """flatten everything and make a standard feed forward layer"""
    fc_input = tf.reshape(in_tensor, [-1, self.len_layer2_out * self.cf.n_words])
    return tf.matmul(fc_input, self.weights['w_fc']) + self.biases['b_fc']

def layer4(self, in_tensor):
    """output feed forward layer"""
    return tf.nn.sigmoid(tf.matmul(in_tensor, self.weights['out']) + self.biases['out'])
```

step through code

1. datamart
2. network ini
3. graph ini
4. fitting 

$$w_{\mathrm{I}}$$

$$w_{\mathrm{I}} - \gamma \sum_{\mathrm{batch}} \frac{\partial \mathrm{loss}}{\partial w_{\mathrm{I}}}$$

run.py

```
import ...

output_map_batch = {"batch_count": [], "accuracy_train": [], "cost_train": []}
output_map_epoch = {"epoch": [], "batch_count": [], "cost_test": [], "accuracy_test": []}

cf = cf.Config()
tex_writer = TeXwriter(".././output", "doc")
tex_writer.addSection("Parameters")
tex_writer.addText(cf.to_tex())
loader = ld.Loader(cf)
char_trf = loader.char_trf
network = nw.Network(cf)
trainer = tn.Trainer(cf, network)
evaluator = ev.Evaluator(cf, network)

with tf.Session() as sess:

    sess.run(tf.global_variables_initializer())

    while loader.epochs < cf.epochs:
        batch_x, batch_y = loader.get_next_train_batch(cf.batch_size, shuffle=cf.shuffle)
        current_output = trainer.train(sess, batch_x, batch_y)

        output_map_batch["batch_count"].append(loader.batches)
        output_map_batch["accuracy_train"].append(current_output[3])
        output_map_batch["cost_train"].append(current_output[1])

        if loader.new_epoch:
            x, y = loader.get_test_data()
            current_test_output = trainer.test(sess, x, y)
```

step through code

1. datamart
2. network ini
3. graph ini
4. fitting 

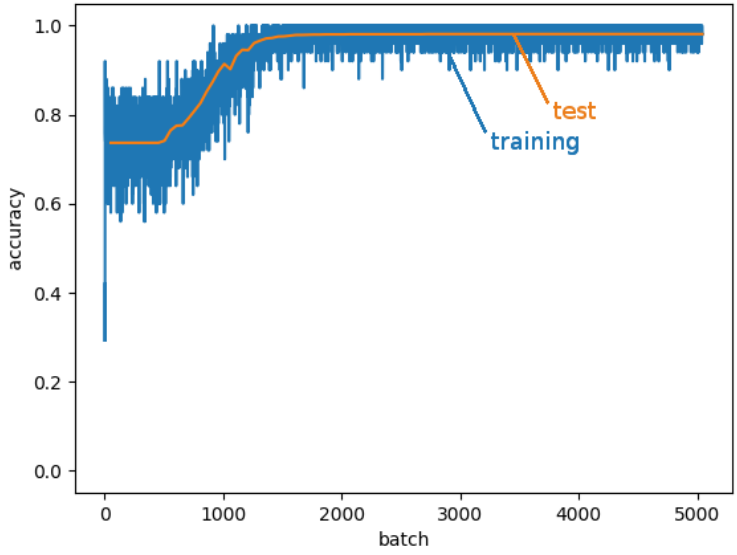
trainer.py

```
def _loss_(self, prediction, y):
    """loss function """
    #single_loss = tf.nn.softmax_cross_entropy_with_logits(logits=prediction, labels=y)
    single_loss = tf.pow(prediction - y, 2)
    return tf.reduce_mean(single_loss)

def train(self, sess, batch_x, batch_y):
    """run the graph build in the constructor"""
    self.train_output = sess.run(
          [self.optimizer, self.cost, self.network.weights, self.accuracy],
          feed_dict={self.x: batch_x, self.y: batch_y})
    return self.train_output

def test(self, sess, x, y):
    """same as train, only do not include the optimizer to not move the weights when generating output"""
    self.test_output = sess.run([self.cost, self.accuracy], feed_dict={self.x: x, self.y: y})
    return self.test_output
```

w_{I}

$$w_{\mathrm{I}} - \gamma \sum_{\mathrm{batch}} \frac{\partial \mathrm{loss}}{\partial w_{\mathrm{I}}}$$

training results - convergence



training results - character importance

cht- -sie verfolgen dich-- - -hast du es denn nicht bemerkt-- fragte frieda- -nein-- sagte k- und suchte sich vergeblich an einzelheiten zu erinnern- -zudringliche und lüsterne jungen sind es wohl- ab, truth:1.0, pred: 1.0

sie dir diese waffe anvertraut in der hoffnung- daSS du sie in einer für mich besonders schlimmen oder entscheidungsreichen stunde anwenden würdest- miSSbrauche ich dich- so miSSbraucht sie dich ähnlich-, truth:0.0, pred: 0.01

elfen wolle- und er danke ihm für die gute absicht- es sei ja möglich- daSS er später einmal etwas brauchen werde- dann werde er sich an ihn wenden- die adresse habe er ja- dagegen könne vielleicht er-, truth:1.0, pred: 1.0

er unterredung ab- es kann manches zur sprache kommen- aber das wichtigste ist doch für mich- daSS ich ihm gegenüberstehe- ich habe nämlich noch mit keinem wirklichen beamten unmittelbar gesprochen- es, truth:0.0, pred: 0.0

te wie ein offenbarungswort untersuchen und von der deutung das eigene lebensglück abhängig machen- nichts kann verfehlter sein- freilich habe auch ich- nicht anders als du- mich von ihm beirren lasse, truth:0.0, pred: 0.01

urfte- nicht hoch ein- fern war ihm bewunderung oder gar neid- denn nicht klamms nähe an sich war ihm das erstrebenswerte- sondern daSS er- k-- nur er- kein anderer mit seinen- mit keines anderen wünsc, truth:1.0, pred: 0.99