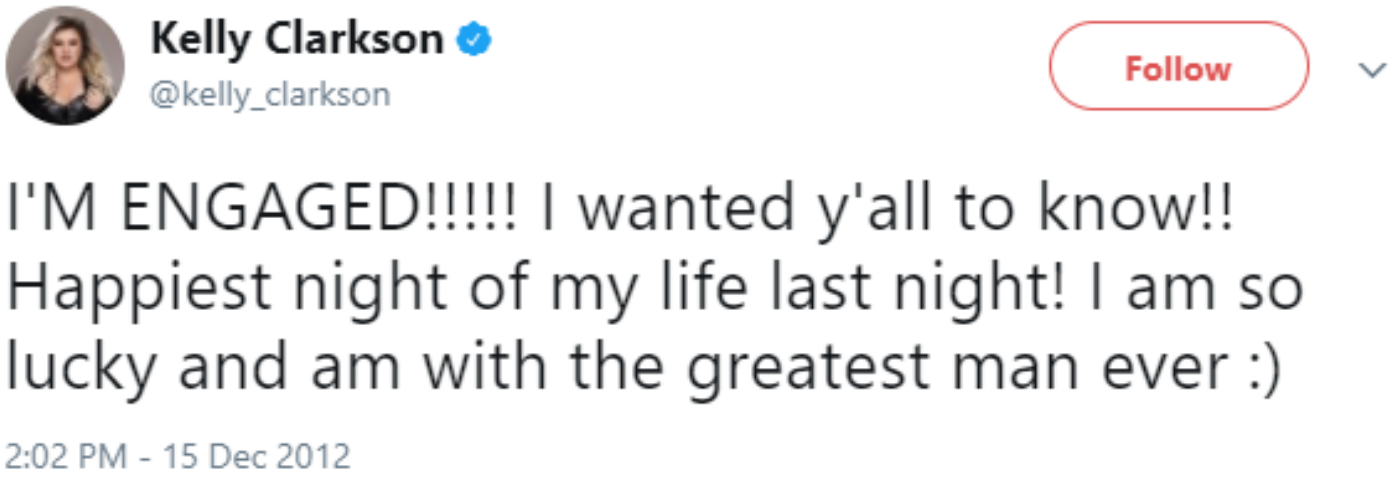
Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts

Paper by Santos and Gatti
Presented by Mitch Kinney

University of Minnesota

November 20, 2017

Plain English: CharSCNN



- The authors call their network the Sentence Convolutional Neural Network (SCNN) and refer to it as CharSCNN.
- Goal of the CharSCNN is to extract sentiment from short sentences such as tweets.
- Uses both word embeddings and character embeddings.

Plain English: CharSCNN

Method

- Use a word embedding matrix where columns are a vector representation of each word in the vocabulary.
- Use/train a character embedding matrix where columns are a vector representation of each character in the vocabulary.
- Combine knowledge of words and characters in the sentence to create a vector representation of the sentence.
- Within the CNN use trained weight matrices and bias vectors to get an estimate of the sentiment of the sentence.

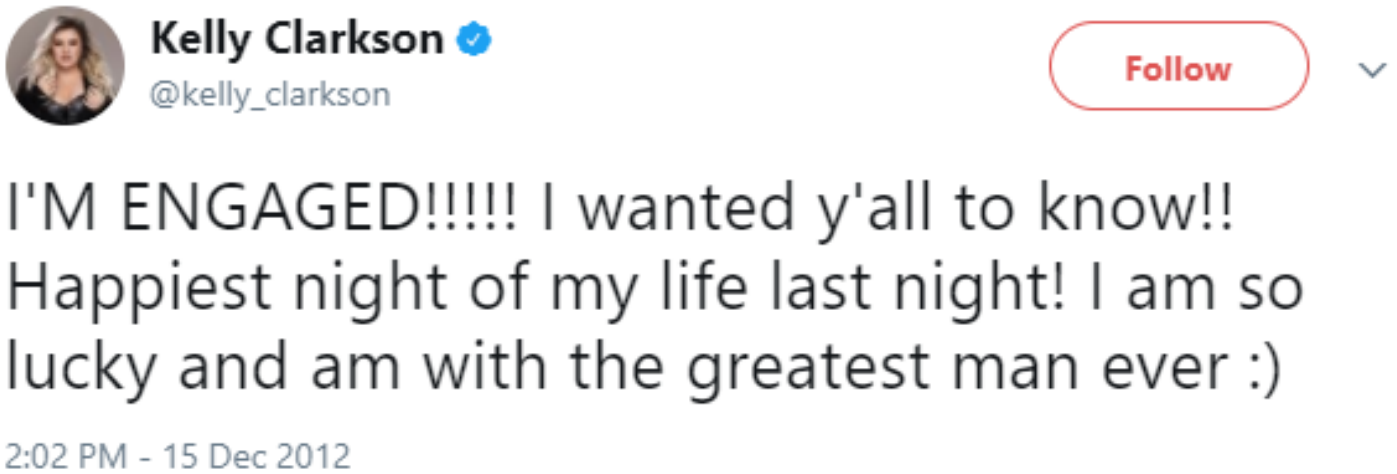
Mathematical Notation: CharSCNN

Word representations

- Define V^{wrd} to be the vocabulary of words and $|V^{wrd}|$ to be its cardinality.
- Define the word embedding matrix $W^{wrd} \in \mathbb{R}^{d^{wrd} \times |V^{wrd}|}$
- Where d^{wrd} is a user supplied hyperparemeter

For simplicity the authors define a single word's embedding vector as $r^{wrd} = W^{wrd} v^{w}$ where v^{w} is a one-hot column vector

Tweet Example: CharSCNN



- This tweet contains N words
- Represented as $[r_1^{wrd}, ..., r_N^{wrd}]$
- Recall each r^{wrd} vector is of length d^{wrd}

Mathematical Notation: CharSCNN

Character representations

- Define V^{chr} and d^{chr} similar to word embedding.
- Also have a character embedding matrix $W^{chr} \in \mathbb{R}^{d^{chr} \times |V^{chr}|}$
- And get $r^{chr} = W^{chr} v^c$ in the same way.

For a word *wrd* of length M characters in our sentence we have the representation

$$wrd = [r_1^{chr}, ..., r_M^{chr}]$$

Mathematical Notation: CharSCNN

Convolutional layer

- Based on a user defined window size of k^{chr} create a subset of the character embedding vector around r_m^{chr} and call it z_m
- $z_m = (r^{chr}_{m-(\frac{k^{chr}-1}{2})}, ..., r_m^{chr}, ..., r^{chr}_{m+(\frac{k^{chr}-1}{2})})^T$, $z_m \in \mathbb{R}^{d^{chr} \cdot k^{chr} \times 1}$

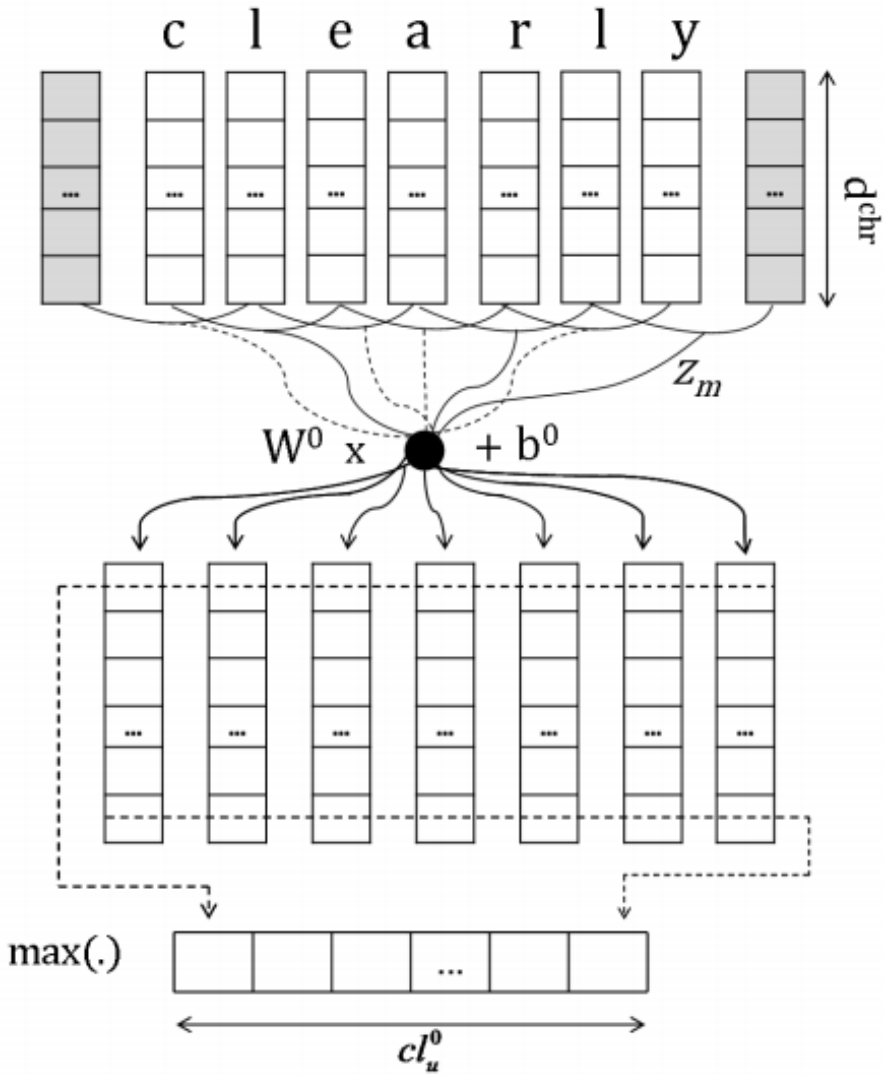
The vector z_m is fed into the convolutional neural network

- Weight matrix $W^0 \in \mathbb{R}^{cl_u^0 \times d^{chr} \cdot k^{chr}}$ and bias vector $b^0 \in \mathbb{R}^{cl_u^0}$
  - cl_u^0 is the user inputted number of convolutional units
- For each m in 1,..,M transform z_m by doing $W^0 z_m + b^0$

The output vector $r^{wch} \in \mathbb{R}^{cl_u^0}$ is defined as

$$r_j^{wch} = \arg\max_m [W^0 z_m + b^0]_j, \quad j = 1, ..., cl_u^0$$

Mathematical Notation: CharSCNN



Mathematical Notation: CharSCNN

Sentence Embedding

- Define a sentence of length N as $[u_1, ..., u_N]$
- Combine word and character embeddings for each word and call it $u_n = (r_n^{wrd}, r_n^{wch})^T$

Use the same convolutional neural network approach as with character embeddings

- With user defined window size k^{wrd}, for n in 1,...,N create
  - $z_n = (u_{n-(\frac{k^{wrd}-1}{2})}, ..., u_n, ..., u_{n+(\frac{k^{wrd}-1}{2})})^T$
  - $z_n \in \mathbb{R}^{(d^{wrd}+cl_u^0)\cdot k^{wrd}\times 1}$

Mathematical Notation: CharSCNN

Sentence Embedding

- Then create sentence embedding vector $r^{sent} \in \mathbb{R}^{cl_u^1}$
  - $r_j^{sent} = \arg\max_n [W^1 z_n + b^1]_j$

Get an estimate s of the sentiment score

- $s = W^3 \cdot h(W^2 r^{sent} + b^2) + b^3$
- where h$(\cdot)$ is the inverse tangent function

Tweet Example: CharSCNN

Kelly Clarkson 

@kelly_clarkson

Follow 

I'M ENGAGED!!!!! I wanted y'all to know!! Happiest night of my life last night! I am so lucky and am with the greatest man ever :)

2:02 PM - 15 Dec 2012

Examples of sentence features extraction

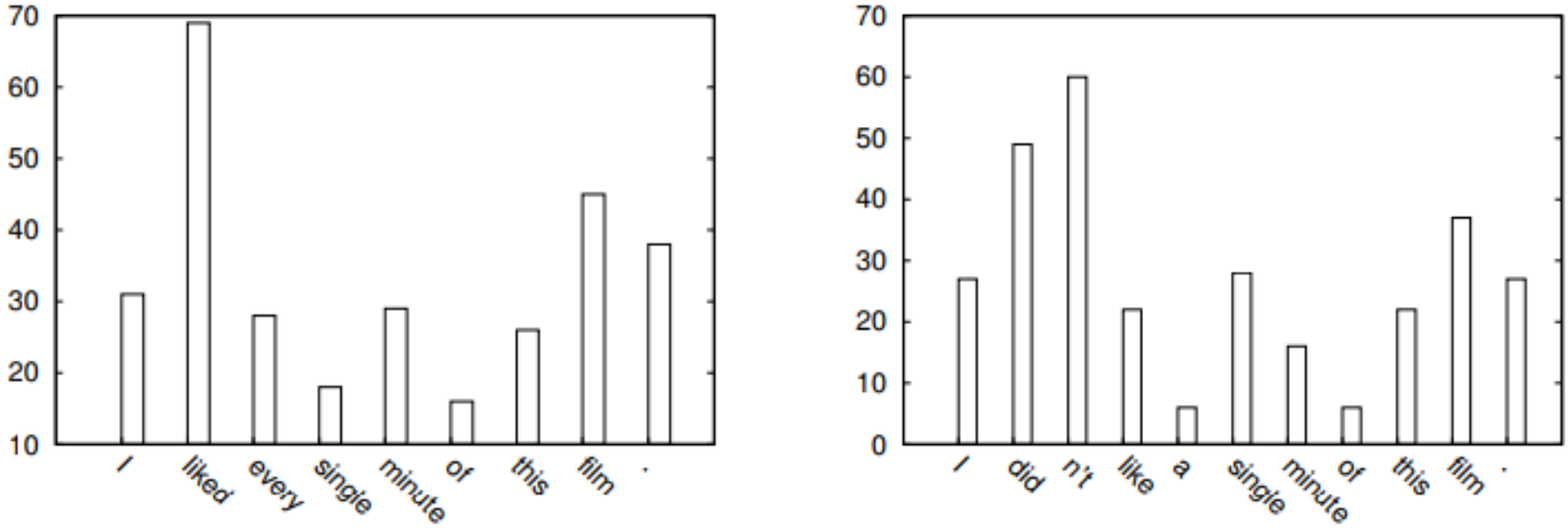


Figure 2: Number of local features selected at each word when forming the sentence-level representation. In this example, we have a positive sentence (left) and its negation (right).

Parameters supplied and trained: CharSCNN

Parameters that need to be user specified or trained by the model

- User specified
  - Vocabularies V^{wrd}, V^{chr}
  - window sizes k^{chr}, k^{wrd}
  - convolutional units cl_u^0, cl_u^1
- Trained
  - Character embedding matrix W^{chr}
  - Weight matrices W^0, W^1, W^2, W^3
  - bias vectors b^0, b^1, b^2, b^3

Training the model: CharSCNN

Let θ be the trainable parameter set, x a short sentence and Y the set of sentiment tags. Using a soft max for a sentiment tag $y \in Y$

$$p(y|x,\theta) = \frac{\exp(s_{\theta,y}(x))}{\sum_{i \in Y} \exp(s_{\theta,i}(x))}$$

$$\Rightarrow \log p(y|x,\theta) = s_{\theta,y}(x) - \log(\sum_{i \in Y} \exp(s_{\theta,i}(x)))$$

Then use the set of training sentences and tags D with stochastic gradient descent to find

$$\hat{\theta} = \arg\min_{\theta} \sum_{(x,y) \in D} -\log p(y|x,\theta)$$

Stochastic gradient descent: Review

- Stochastic gradient descent is an iterative algorithm for minimizing an objective function with a summand.
- Akin to steepest descent where for an objective function Q(w) and step size/ learning rate η we solve

$$\hat{w} = w^0 - \eta \cdot \nabla_w Q(w^0)$$

Stochastic gradient descent: Review

Simple example from Wikipedia: minimizing least squares

$$Q(w) = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2$$

Then for a learning rate η our step would be

$$w_1^{new} = w_1^{old} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} (2(w_1^{old} + w_2^{old} x_i - y_i))$$

$$w_2^{new} = w_2^{old} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} (2x_i(w_1^{new} + w_2^{old} x_i - y_i))$$

Stochastic gradient descent: Review

- For our CNN I'll define the loss function for a certain training sentence and sentiment tag (x, y) as $\ell(\theta) = -\log p(y|x, \theta)$.
- Then I need to find the derivative with respect to one of my parameters, W^2 for instance.
- Use the chain rule to get down to the layer in the network

Stochastic gradient descent: Review

Recall our loss function is...

$$\begin{aligned}\ell(\theta) &= -\log p(y|x,\theta) \\ &= -s_{\theta,y}(x) + \log(\sum_{i \in Y} \exp(s_{\theta,i}(x)))\end{aligned}$$

And the sentiment is computed by...

$$s_{\theta,y}(r_x^{sent}) = W^3 \cdot h(W^2 r_x^{sent} + b^2) + b^3$$

Stochastic gradient descent: Review

Starting with top layer...

$$\frac{d\ \ell(\theta)}{d\ W^2} = \frac{d\ \ell(\theta)}{d\ s_{\theta,y}(x)} \cdot \frac{d\ s_{\theta,y}(x)}{d\ W^2}$$

$$= \frac{-s_{\theta,y}(x) + \log(\sum_{i \in Y} \exp(s_{\theta,i}(x)))}{d\ s_{\theta,y}(x)} \cdot \frac{W^3 h(W^2 r_x^{sent} + b^2) + b^3}{d\ W^2}$$

$$= (-1 + \frac{\exp(s_{\theta,y}(x))}{\sum_{i \in Y} \exp(s_{\theta,i}(x))}) \cdot (\text{diag}(W^3)(1_{hl_u} - h^2(W^2 r_x^{sent} + b^2)$$

$$(r_x^{sent})^T)$$

Note that

- $W^2 \in \mathbb{R}^{hl_u \times cl_u^1}$
- $W^3 \in \mathbb{R}^{1 \times hl_u}$
- $r_x^{sent} \in \mathbb{R}^{cl_u^1}$

What makes CharSCNN novel

The authors claim that the novelty of their model is the

- Inclusion of two convolutional layers which allows for sentences and words of any size
- Feed forward approach rather than recurrent

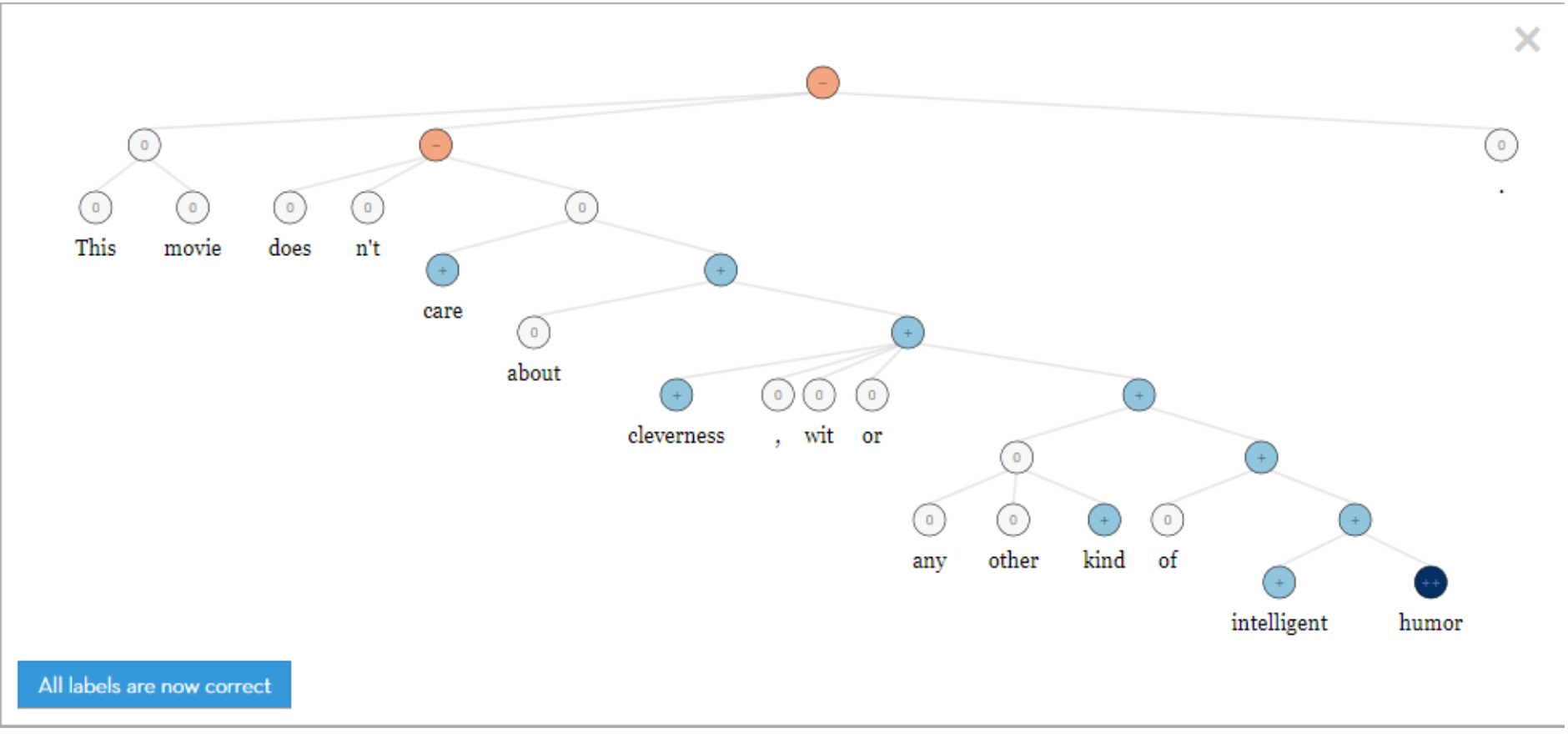
Related Work: Recurrant Neural Tensor Network

The authors use a tree based method to do a bottom up sentiment analysis of short sentences

- First build an-almost binary tree with words as leaves
- Then assign word embeddings and filter up the tree to classify the entire sentence
- Must train the weight matrix connecting children to parents and the word embedding matrix

Related Work: Recurrant Neural Tensor Network

"This movie doesn't care about cleverness, wit or any other kind of intelligent humor"



Data Overview

SSTb (Stanford Sentiment Treebank corpus)

- 11,855 movie reviews.
- Manually annotated by three judges on true sentiment score.

STS (Stanford Twitter Sentiment corpus)

- 1.6 million tweets gathered from searching ":)" and ":(".
- Automatically labeled positive or negative based on whether a happy or sad face is attached to the tweet.
- Authors used about 7% of the data in their experiment to keep run-time low.

Results: Pretrained W^{wrd}

To get the word embeddings matrix W^{wrd} the authors used *word2vec*

- *word2vec* is a neural network approach to create vector representations of words.
- These vector representations are able to capture relationships between words.
- Method proposed by a team at Google Deepmind.

Results: STb corpus

Authors used two models: SCNN and CharSCNN

- SCNN only uses word embeddings. $u_n = r_n^{wrd}$
- CharSCNN uses both word embeddings and character embeddings. $u_n = (r_n^{wrd}, r_{wch})$

Authors also trained the CNN with and without phrases

- STb contains full sentences and parts of the full sentences called phrases
- Authors wanted to know if including the phrases helped training

In total comparing four methods to previous approaches

Results: STb corpus

Movie reviews are either very negative, negative, neutral, positive and very positive

Model	Phrases	Fine-Grained	Positive/Negative
CharSCNN	yes	48.3	85.7
SCNN	yes	48.3	85.5
CharSCNN	no	43.5	82.3
SCNN	no	43.5	82.0
RNTN	yes	45.7	85.4

Results: STS corpus

The authors then compared their model for sentiment analysis using the twitter data to other approaches. I've shown the best one attributed to Speriosu et al. 2011.

Model	Accuracy
CharSCNN	86.4
SCNN	85.2
LProp	84.7

Conclusion

In this presentation...

- Overview of CharSCNN network
- Review of stochastic gradient descent
- Results showing (at the time) state of the art performance

Thank you!