An introduction to Deep Learning for NLP

Konstantinos Skianis, Ph.D. Student

May 05, 2017
Outline

1. Introduction
   - Deep Learning Era
   - Input features

2. CNN
   - Definition & Properties
   - How it works

3. RNN
   - Vanilla RNN
   - LSTM

4. Neural Network Training
   - Optimization Issues
   - Regularization

5. Conclusion
   - Applications & Material
1 Introduction
   - Deep Learning Era
   - Input features

2 CNN
   - Definition & Properties
   - How it works

3 RNN
   - Vanilla RNN
   - LSTM

4 Neural Network Training
   - Optimization Issues
   - Regularization

5 Conclusion
   - Applications & Material
Deep Learning Era

What?

- Many layers of non-linear units for feature extraction and transformation
- Lower level to higher level features form hierarchy of concepts

Why now?

- Large data available
- Computational resources (CPUs and GPUs)

Most used models:

- Convolutional Neural Network (CNNs)
- Long Short Term Memory network-LSTM (variant of RNN)
- Gated Recurrent Unit (GRU)
Sparse vs. dense feature representations

**Figure:** Two encodings of the information: current word is “dog”; previous word is “the”; previous pos-tag is “DET”. (a) Sparse feature vector. (b) Dense, embeddings-based feature vector.
What to use?

**One Hot**: Each feature is its own dimension.
- Dimensionality of one-hot vector is same as number of distinct features.
- Features are completely independent from one another. Example: “word is ‘dog’ ” is as dis-similar to “word is ‘thinking’ ” than it is to “word is ‘cat’ ”.

**Dense**: Each feature is a d-dimensional vector.
- Model training will cause similar features to have similar vectors - information is shared between similar features.

**Benefits of dense and low-dimensional vectors**
- Computational efficient
- Generalization power
- Collobert & Weston, 2008; Collobert et al. 2011; Chen & Manning, 2014 ... advocate the use of dense, trainable embedding vectors for all features.
**Word Embeddings**

**Initialization:**

- word2vec: initialize the word vectors to uniformly sampled random numbers in the range \([-\frac{1}{2d}, \frac{1}{2d}]\) where \(d\) is the number of dimensions.

- xavier initialization: \([-\frac{\sqrt{6}}{\sqrt{d}}, \frac{\sqrt{6}}{\sqrt{d}}]\)

**Problems:**

- Word similarity is hard to define and is usually very task-dependent

**Missing words in pre-trained vectors?**

- Retrain with training data
- Find synonyms?
- Open research problem...
1. Introduction
   - Deep Learning Era
   - Input features

2. CNN
   - Definition & Properties
   - How it works

3. RNN
   - Vanilla RNN
   - LSTM

4. Neural Network Training
   - Optimization Issues
   - Regularization

5. Conclusion
   - Applications & Material
Convolutional Neural Networks

**Definition**

Multiple-layer feedforward neural networks where each neuron in a layer receives input from a neighborhood of the neurons in the previous layer. (Lecun, 1998)

**From Computer Vision to NLP:** 2d grid $\rightarrow$ 1d sequence

**Properties**

- Compositionality: learn complex features starting from small regions $\leftrightarrow$ higher-order features ($n$-grams) can be constructed from basic unigrams
- Local invariance: detect an object regardless the position in image $\leftrightarrow$ ordering is crucial locally and not globally
Convolutional Neural Networks (Convolutional Neural Networks (2))

- A sequence of words \( x = x_1, ..., x_n \), each with their corresponding \( d_{emb} \) dimensional word embedding \( v(x_i) \)

- 1d convolution layer of width \( k \) works by moving a sliding window of size \( k \) over the sentence, and applying the same “filter” to each window in the sequence \( [v(x_i); v(x_{i+1}); ...; v(x_{i+k-1})] \)

- Depending on whether we pad the sentence with \( k - 1 \) words to each side, we may get either \( m = n - k + 1 \) (narrow convolution) or \( m = n + k + 1 \) windows (wide convolution)

- Result of the convolution layer is \( m \) vectors \( p_1, ..., p_m \in \mathbb{R}^{d_{conv}}: p_i = g(w_iW + b) \) where \( g \) is a non-linear activation function that is applied element-wise, \( W \in \mathbb{R}^{kd_{emb} \times d_{conv}} \) and \( b \in \mathbb{R}^{d_{conv}} \) are parameters of the network.
CNN for sentence classification (Zhang and Wallace, 2015)

Figure 1: Illustration of a CNN architecture for sentence classification. We depict three filter region sizes: (2,3,4), each of which has 2 filters. Filters perform convolutions on the sentence matrix and generate (variable-length) feature maps; 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states.

Notably, even naively incorporating word2vec embeddings into feature vectors usually improves results. We also experimented with combining the uni-gram, bi-gram and word vector features with a linear kernel SVM. We kept only the most frequent 30k n-grams for all datasets, and tuned hyperparameters via nested cross-fold validation, optimizing for accuracy (AUC for Irony). For consistency, we used the same pre-processing steps for the data as described in previous work (Kim, 2014). We report means from 10-folds over all datasets in Table 1.

4.1 Baseline Configuration
We first consider the performance of a baseline CNN configuration. Specifically, we start with the architectural decisions and hyperparameters used in previous work (Kim, 2014) and described in Table 2. To contextualize the variance in performance attributable to various architecture decisions and hyperparameter settings, it is critical to assess the variance due strictly to the parameter estimation procedure. Most prior work, unfortunately, has not reported such variance, despite a highly stochastic learning procedure. This variance is attributable to estimation via SGD, random dropout, and random weight parameter initialization. Holding all variables (including the folds)
1. Introduction
   - Deep Learning Era
   - Input features

2. CNN
   - Definition & Properties
   - How it works

3. RNN
   - Vanilla RNN
   - LSTM

4. Neural Network Training
   - Optimization Issues
   - Regularization

5. Conclusion
   - Applications & Material
Recurrent Neural Networks

- CNNs are limited to local patterns
- RNNs were specifically developed to be used with sequences
- The task of language modeling consists in learning the probability of observing the next word in a sentence given the $n - 1$ preceding words, that is $P[w_n|w_1, \ldots, w_{n-1}]$.
- At given time step: $s_t = f(Ux_t + Ws_{t-1})$

**Example**

If the sequence is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word.
• \( x_t \) is the input at time step \( t \). For example, \( x_1 \) could be a one-hot vector corresponding to the second word of a sentence.

• \( s_t \) is the hidden state at time step \( t \) (memory). \( s_t \) is calculated based on the previous hidden state and the input at the current step: \( s_t = f(Ux_t + Ws_{t-1}) \). \( f \) is usually a nonlinearity (tanh or ReLU). \( s_{-1} \), which is required to calculate the first hidden state, is typically initialized to all zeroes.

• \( o_t \) is the output at step \( t \). I.e. if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. \( o_t = \text{softmax}(Vs_t) \).
Long Short Term Memory Networks

Hochreiter & Schmidhuber (1997)
LSTM is explicitly designed to avoid the long-term dependency problem.

Properties

- Chain like structure
- Instead of having a single neural network layer, there are four
- Remove or add information to the cell state, carefully regulated by structures called gates
- Three sigmoid gates, to protect and control the cell state
LSTM architecture

(1) forget gate layer:
\[ f_t = \sigma(U_f x_t + W_f s_{t-1}) \]

(2) input gate layer:
\[ i_t = \sigma(U_i x_t + W_i s_{t-1}) \]

(3) candidate values computation layer:
\[ \tilde{c}_t = \tanh(U_c x_t + W_c s_{t-1}) \]

(4) \[ c_t = f_t \times C_{t-1} + i_t \times \tilde{c}_t \]

(5) output gate layer:
\[ o_t = \sigma(U_o x_t + W_o s_{t-1}) \]

(6) \[ y_t = o_t \times \tanh(C_t) \]
Introduction
- Deep Learning Era
- Input features

CNN
- Definition & Properties
- How it works

RNN
- Vanilla RNN
- LSTM

Neural Network Training
- Optimization Issues
- Regularization

Conclusion
- Applications & Material
Optimization Issues

\[ \mathbb{E}_{x,y \sim \hat{p}_{\text{data}}(x,y)} [L(f(x; \theta), y)] = \frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)}) \]

**Learning ≠ Pure optimization**

- Performance measure \( P \), that is defined with respect to the test set
- May also be intractable
- Reduce a different cost function \( J(\theta) \) hoping it will improve \( P \)

**Properties**

- Usually **non-convex**
- Any deep model is essentially guaranteed to have an extremely large number of local minima
- Model identifiability: a sufficiently large training set can rule out all but one setting of parameters \( \rightarrow \) **weight space symmetry**
- Local minima is a good approximation to global minima
More Optimization

Issues

• All of these local minima arising from non-identifiability are equivalent to each other in cost value → not a problematic form of non-convexity
• Local minima can be problematic if they have high cost in comparison to the global minimum
• Saddle point as being a local minimum along one cross-section of the cost function and a local maximum along another cross-section
Initialization of weights

- May get stuck in a local minimum or a saddle point
- Starting from different initial points (e.g. parameters) may result in different results
- Random values has an important effect on the success of training
- Xavier initialization, Glorot and Bengio (2010):

\[
W \sim U \left[ - \frac{\sqrt{6}}{\sqrt{d_{in} + d_{out}}}, + \frac{\sqrt{6}}{\sqrt{d_{in} + d_{out}}} \right]
\]

- When using ReLU non-linearities → sampling from a zero-mean Gaussian distribution whose standard deviation is \( \sqrt{\frac{2}{d_{in}}} \), He et al. (2015)

Vanishing and Exploding Gradients

- Error gradients to either vanish (become exceedingly close to 0) or explode (become exceedingly high) in backpropagation
Regularization

Overfitting

- Many parameters
- Prune to overfitting

Example: LSTM has a set of 2 matrices: $U$ and $W$ for each of the 3 gates. $n$ is the hidden layer size and $m$ is the vocabulary size. (ie $n = 100$, $m = 8000$)

- $U$ has dimensions $n \times m$
- $W$ has dimensions $n \times n$
- there is a different set of these matrices for each of the three gates (like $U_{\text{forget}}$ for the forget gate)
- there is another set of these matrices for updating the cell state $S$

$\leftarrow$ total number of parameters $= 4(nm + n^2) = 3,240,000$

Solution

- Dropout: randomly dropping (setting to 0) half of the neurons in the network (or in a specific layer) in each training example. (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012)
1 Introduction
   - Deep Learning Era
   - Input features

2 CNN
   - Definition & Properties
   - How it works

3 RNN
   - Vanilla RNN
   - LSTM

4 Neural Network Training
   - Optimization Issues
   - Regularization

5 Conclusion
   - Applications & Material
Deep Learning models for numerous tasks

- **CNNs**: document classification, short-text categorization, sentiment classification, relation type classification between entities, event detection, paraphrase identification, semantic role labelling, qa

- **Recurrent**: language modeling, sequence tagging, machine translation, dependency parsing, sentiment analysis, noisy text normalization, dialog state tracking, response generation

- **Recursive** (generalization of RNN that can handle trees): constituency-dependency parse re-ranking, discourse parsing, semantic relation classification, political ideology detection based on parse trees, sentiment classification, target-dependent sentiment classification, qa
Understanding Neural Networks

Deep “dark” networks

If the network fails, it is hard to understand what went wrong!

- Hard to provide concrete interpretation
- Visualization to the rescue!
- http://colah.github.io/
- Visualizing and understanding convolutional networks, M. Zeiler and R. Fergus (2014)
Future: Deep Generative Models

- Probability distributions over multiple variables
- Boltzmann Machines, RBM, Deep Belief Networks

Resources

- Natural language processing (almost) from scratch, R. Collobert et al., 2011
- A Primer on Neural Network Models for Natural Language Processing, Goldberd, 2015
- Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016

Conference

- International Conference on Learning Representations (ICLR)

Thank you!