Lecture 10:
Deep NLP and End-to-End Learning

Ankur – Google
Midterm Grades

- 75% homework (1, 2, 3)
- 25% project proposal

There are no fixed thresholds for assigning letter grades, they are curved based on the distribution of the total scores for all students.

- For the midterm grades, 45-50% of class was given A-range grades.
Course Policies

• Please revisit the collaboration policy on the last page of HW 1.

• For your final project.
  • You must indicate which external code you used, and what work you did.
NLP before Deep Learning

- Usually extract complex linguistically motivated features from the data
- Shove these features into a classifier (typically a linear one) or a structured prediction model (e.g. a CRF)

They solved the problem with statistics
Example 1: Homework 2

- You defined many handcrafted features to predict the class label
  - unigram/bigram/trigram characters etc.
  - special symbols
  - common prefixes
  - suffixes like “Ltd” or “Inc”

- Put all of them into a maximum entropy model.
Advantages

• Easily debuggable.

• Features typically have linguistic motivated meaning

• Much easier to understand how the model works.

• Model will not behave in strangely unexpected ways.
Disadvantages

• May require lots of handcrafted features to work on large amounts of data.
  • Can’t possibly get every pattern

• Linear classifiers

• Restricted model means you don’t benefit enough from large amounts of data (underfitting)
Example: Phrase Based MT

Sentence-aligned corpus

Word alignments

Phrase table (translation model)

cat ||| chat ||| 0.9
the cat ||| le chat ||| 0.8
dog ||| chien ||| 0.8
house ||| maison ||| 0.6
my house ||| ma maison ||| 0.9
language ||| langue ||| 0.9
...

Morgen fliege ich nach Kanada zur Konferenz

Tomorrow I will fly to the conference in Canada
Advantages: Phrase Based MT

- Can easily divide the task into different modules
  - Useful from the engineering point of view
- Each module may have an elegant solution (e.g. word alignment)
- Usually requires less data to train.
- Can’t behave in unexpected ways, since limited to what’s in the phrase table.
Disadvantages: Phrase Based MT

- Error propagation
  - Errors from previous stages propagate to later stages
- Requires intermediate annotation (or unsupervised learning)
  - Gains in intermediate metrics (e.g. AER) don’t always translate to gains in the end task
NLP: Deep Learning View

- The current trend in the NLP community is to be “end-to-end”

- Simply get input / label pairs and train a really complex classifier.

- Why is it possible now when it wasn’t before?
  - More computational power
  - More data
  - New optimization tricks
Outline for this Lecture

- Neural Language Modeling
- Classification with Neural Models
N-grams

\[ P(w_1 \ldots w_n) = \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1}) \]

- Requires Markov assumption
- Requires smoothing i.e. interpolating with lower order n-grams
(Finite Context) Neural LMs
[Bengio et al. 2003, Mnih and Hinton 2007, Botha and Blunsom 2014]

\[ P(w_i | w_{i-n+1}^{i-1}) = \frac{\exp(\nu(w_i))}{\sum_{v \in \mathcal{V}} \exp(\nu(v))} \]

**Markov Assumption**

**Word Embedding** for \( w_j \)

\[ p = \sum_{j=1}^{n-1} \mathbf{e}_{w_j} C_j \]

**Context Vector**

\[ \nu(w) = p^T \mathbf{e}_w + b_w \]

**Score**
Training (Finite Context) Neural LMs
[Bengio et al. 2003, Mnih and Hinton 2007, Botha and Blunsom 2014]

\[
P(w_i | w_{i-n+1}^{i-1}) = \frac{\exp(\nu(w_i))}{\sum_{v \in \mathcal{V}} \exp(\nu(v))}
\]

\[
p = \sum_{j=1}^{n-1} e_{w_j} C_j \quad \nu(w) = p^T e_w + b_w
\]

**Maximum Likelihood**

\[
\max_{C,e,b} \sum_i \log P(w_i | w_{i-n+1}^{i-1})
\]
(Finite Context) Neural LMs
[Bengio et al. 2003, Mnih and Hinton 2007, Botha and Blunsom 2014]

- Initial results not impressive.
- Table below is on a small dataset with heavy vocabulary truncation which disfavors n-gram models. Why?

Table 2. Perplexity scores for the models trained on the 14M word training set. The mixture test score is the perplexity obtained by averaging the model’s predictions with those of the Kneser-Ney 5-gram model. The log-bilinear models use 100-dimensional feature vectors.

<table>
<thead>
<tr>
<th>Model type</th>
<th>Context size</th>
<th>Model test score</th>
<th>Mixture test score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log-bilinear</td>
<td>5</td>
<td>117.0</td>
<td>97.3</td>
</tr>
<tr>
<td>Log-bilinear</td>
<td>10</td>
<td>107.8</td>
<td></td>
</tr>
<tr>
<td>Back-off KN3</td>
<td>2</td>
<td>129.8</td>
<td></td>
</tr>
<tr>
<td>Back-off KN5</td>
<td>4</td>
<td>123.2</td>
<td></td>
</tr>
<tr>
<td>Back-off KN6</td>
<td>5</td>
<td>123.5</td>
<td></td>
</tr>
<tr>
<td>Back-off KN9</td>
<td>8</td>
<td>124.6</td>
<td></td>
</tr>
</tbody>
</table>

[Mnih and Hinton 2007]
Improving Neural LMs

• As with the rest of deep learning, initial neural LM results weren’t impressive because of lack of computation to train really large models.

• Adding infinite context (Recurrent Nets) also helps
Recurrent Neural Networks

Continuous state vector

Continuous input vector

$h_1 \rightarrow h_2 \rightarrow \ldots \rightarrow h_T$

$x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_T$
Recurrent Neural Networks

\[ h_t = \sigma(A[x_t + h_{t-1}] + b) \]

non-linearity e.g. sigmoid/tanh/relu
Recurrent Neural Networks vs HMMs

\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n \]

\[ w_1 \rightarrow w_2 \rightarrow \ldots \rightarrow w_n \]

\[ h_1 \rightarrow h_2 \rightarrow \ldots \rightarrow h_T \]

\[ x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_T \]
Recurrent Neural Networks vs HMMs/CRFs

**Hidden Markov Models / CRFs**

- Hidden state is a discrete variable
- Model is stochastic i.e. you sample a hidden state

**Recurrent Neural Networks**

- Hidden state is a continuous vector that is a function of nonlinear activations
- Model is deterministic
$h_t = \sigma(A[x_t + h_{t-1}] + b)$  \quad y_t[k] = \frac{\exp(v_k^T h_t)}{\sum_k \exp(v_k^T h_t)}$
RNN LMs [Mikolov et al. 2010]

\[ h_t = \sigma(A[x_t + h_{t-1}] + b) \]

\[ y_t[k] = \frac{\exp(v_k^T h_t)}{\sum_k \exp(v_k^T h_t)} \]

*probability of word k given history*

\[ P(w_t = k|w_1, \ldots, w_{t-1}) \]
RNN LM results [Chelba et al. 2013]

More legitimate dataset / vocabulary size.

<table>
<thead>
<tr>
<th>Model</th>
<th>Num. Params [billions]</th>
<th>Training Time [hours]</th>
<th>[CPUs]</th>
<th>Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpolated KN 5-gram, 1.1B n-grams (KN)</td>
<td>1.76</td>
<td>3</td>
<td>100</td>
<td>67.6</td>
</tr>
<tr>
<td>Katz 5-gram, 1.1B n-grams</td>
<td>1.74</td>
<td>2</td>
<td>100</td>
<td>79.9</td>
</tr>
<tr>
<td>Stupid Backoff 5-gram (SBO)</td>
<td>1.13</td>
<td>0.4</td>
<td>200</td>
<td>87.9</td>
</tr>
<tr>
<td>Interpolated KN 5-gram, 15M n-grams</td>
<td>0.03</td>
<td>3</td>
<td>100</td>
<td>243.2</td>
</tr>
<tr>
<td>Katz 5-gram, 15M n-grams</td>
<td>0.03</td>
<td>2</td>
<td>100</td>
<td>127.5</td>
</tr>
<tr>
<td>Binary MaxEnt 5-gram (n-gram features)</td>
<td>1.13</td>
<td>1</td>
<td>5000</td>
<td>115.4</td>
</tr>
<tr>
<td>Binary MaxEnt 5-gram (n-gram + skip-1 features)</td>
<td>1.8</td>
<td>1.25</td>
<td>5000</td>
<td>107.1</td>
</tr>
<tr>
<td>Hierarchical Softmax MaxEnt 4-gram (HME)</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>101.3</td>
</tr>
<tr>
<td>Recurrent NN-256 + MaxEnt 9-gram</td>
<td>20</td>
<td>60</td>
<td>24</td>
<td>58.3</td>
</tr>
<tr>
<td>Recurrent NN-512 + MaxEnt 9-gram</td>
<td>20</td>
<td>120</td>
<td>24</td>
<td>54.5</td>
</tr>
<tr>
<td>Recurrent NN-1024 + MaxEnt 9-gram</td>
<td>20</td>
<td>240</td>
<td>24</td>
<td>51.3</td>
</tr>
</tbody>
</table>

Table 1: Results on the 1B Word Benchmark test set with various types of language models.
Vanishing/Exploding gradients

A key weakness of vanilla RNNs is the vanishing/exploding gradients phenomenon.

Derivative of RNN is computed by applying chain rule which can lead to some updates become very small or very large.
LSTM (Long-Short Term Memory) [Hochreiter and J. Schmidhuber 1997]

Proposed to capture long-term dependencies without vanishing gradients problem.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
LSTM (Long-Short Term Memory) [Hochreiter and J. Schmidhuber 1997]

RNN:

LSTM:
LSTM (Long-Short Term Memory) [Hochreiter and J. Schmidhuber 1997]

**LSTM cell:**

\[
\begin{align*}
    f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
    i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
    o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
    c_t &= f_t \odot c_{t-1} + i_t \odot o_c(W_c x_t + U_c h_{t-1} + b_c) \\
    h_t &= \sigma_t \odot \sigma_h(c_t)
\end{align*}
\]
## LSTM LM Results [Jozefowicz et al. 2017]

### Table 1. Best results of single models on the 1B word benchmark. Our results are shown below previous work.

<table>
<thead>
<tr>
<th>Model</th>
<th>Test Perplexity</th>
<th>Number of Params [billions]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sigmoid-RNN-2048 (Ji et al., 2015a)</td>
<td>68.3</td>
<td>4.1</td>
</tr>
<tr>
<td>Interpolated KN 5-gram, 1.1B n-grams (Chelba et al., 2013)</td>
<td>67.6</td>
<td>1.76</td>
</tr>
<tr>
<td>Sparse Non-Negative Matrix LM (Shazeer et al., 2015)</td>
<td>52.9</td>
<td>33</td>
</tr>
<tr>
<td>RNN-1024 + MaxEnt 9-gram features (Chelba et al., 2013)</td>
<td>51.3</td>
<td>20</td>
</tr>
<tr>
<td>LSTM-512-512</td>
<td>54.1</td>
<td>0.82</td>
</tr>
<tr>
<td>LSTM-1024-512</td>
<td>48.2</td>
<td>0.82</td>
</tr>
<tr>
<td>LSTM-2048-512</td>
<td>43.7</td>
<td>0.83</td>
</tr>
<tr>
<td>LSTM-8192-2048 (No Dropout)</td>
<td>37.9</td>
<td>3.3</td>
</tr>
<tr>
<td>LSTM-8192-2048 (50% Dropout)</td>
<td>32.2</td>
<td>3.3</td>
</tr>
<tr>
<td>2-Layer LSTM-8192-1024 (BIG LSTM)</td>
<td>30.6</td>
<td>1.8</td>
</tr>
<tr>
<td>BIG LSTM+CNN Inputs</td>
<td><strong>30.0</strong></td>
<td><strong>1.04</strong></td>
</tr>
<tr>
<td>BIG LSTM+CNN Inputs + CNN Softmax</td>
<td>39.8</td>
<td>0.29</td>
</tr>
<tr>
<td>BIG LSTM+CNN Inputs + CNN Softmax + 128-dim correction</td>
<td>35.8</td>
<td>0.39</td>
</tr>
<tr>
<td>BIG LSTM+CNN Inputs + Char LSTM predictions</td>
<td>47.9</td>
<td>0.23</td>
</tr>
</tbody>
</table>

### Table 2. Best results of ensembles on the 1B Word Benchmark.

<table>
<thead>
<tr>
<th>Model</th>
<th>Test Perplexity</th>
<th>Number of Params [billions]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large Ensemble (Chelba et al., 2013)</td>
<td>43.8</td>
<td></td>
</tr>
<tr>
<td>RNN+KN-5 (Williams et al., 2015)</td>
<td>42.4</td>
<td></td>
</tr>
<tr>
<td>RNN+KN-5 (Jie Tal, 2015)</td>
<td>42.0</td>
<td></td>
</tr>
<tr>
<td>RNN+SNM10-Skip (Shazeer et al., 2015)</td>
<td>41.3</td>
<td></td>
</tr>
<tr>
<td>Large Ensemble (Shazeer et al., 2015)</td>
<td>41.0</td>
<td></td>
</tr>
<tr>
<td>Our 10 Best LSTM Models (Equal Weights)</td>
<td>26.3</td>
<td></td>
</tr>
<tr>
<td>Our 10 Best LSTM Models (Optimal Weights)</td>
<td>26.1</td>
<td></td>
</tr>
<tr>
<td>10 LSTM + KN-5 (Equal Weights)</td>
<td>25.3</td>
<td></td>
</tr>
<tr>
<td>10 LSTM + KN-5 (Optimal Weights)</td>
<td>25.1</td>
<td></td>
</tr>
<tr>
<td>10 LSTM + SNM10-Skip (Shazeer et al., 2015)</td>
<td>23.7</td>
<td></td>
</tr>
</tbody>
</table>

### 4.4 Training Procedure

The models were trained until convergence with an AdaGrad optimizer using a learning rate of 0.2. In all the experiments the RNNs were unrolled for 20 steps without ever resetting the LSTM states. We used a batch size of 128. We clip the gradients of the LSTM weights such that their norm is bounded by 1.0 (Pascanu et al., 2012).

Using these hyper-parameters we found large LSTMs to be relatively easy to train. The same learning rate was used in almost all of the experiments. In a few cases we had to reduce it by an order of magnitude. Unless otherwise stated, the experiments were performed with 32 GPU workers and asynchronous gradient updates. Further details will be fully specified with the code upon publication.

Training a model for such large target vocabulary (793471 words) required to be careful with some details about the approximation to full Softmax using importance sampling. We used a large number of negative (or noise) samples: 8192 such samples were drawn per step, but were shared across all the target words in the batch (2560 total, i.e. 128 times 20 unrolled steps). This results in multiplying (2560 x 1024) times (1024 x (8192+1)) (instead of (2560 x 1024) times (1024 x 793471)), i.e. about 100-fold less computation.
Outline for this Lecture

• Neural Language Modeling

• Classification with Neural Models
Premise: Bob is in his room, but because of the thunder and lightning outside, he cannot sleep.

Hypothesis 1: Bob is awake. - entailment

Hypothesis 2: It is sunny outside. - contradiction

Hypothesis 3: Bob has a big house. - neutral
Recent Work (Sentence Encoding)

\[(a_1, \ldots, a_T)\]  \quad \text{and} \quad \[(b_1, \ldots, b_T)\]

word embeddings
Recent Work (Sentence Encoding)

\[(a_1, ..., a_T)\]

\[(b_1, ..., b_T)\]
Recent Work (Sentence Encoding)

\[
x_h x_t = h^{T} (a_1, \ldots, a_T)
\]

\[
x_h x_t = h^{T} (b_1, \ldots, b_T)
\]

similarity layer
Recent Work (Sentence Encoding)

\[
\begin{align*}
    h_x = T_x = 1 \\
    h_x = h_T \ (a_1, \ldots, a_T) \\
    \text{(output)} \\
    (b_1, \ldots, b_T)
\end{align*}
\]
Recent Work (Sentence Encoding)

\[
\begin{align*}
    h_x &= T_X t = 1 h_t = h_T h_x = h_T (a_1, \ldots, a_T) \\
    h_x &= T_X t = 1 h_t = h_T h_x = h_T (b_1, \ldots, b_T)
\end{align*}
\]

output
Recent Work (Sentence Encoding)

\[
\begin{align*}
    h &= X_t h_t h_x = h_T h_x (a_1, \ldots, a_T) \\
    b &= b_1, \ldots, b_T
\end{align*}
\]
Recent Work (Sentence Encoding)

$h(x) = T X t = 1 h x = T X t = 1 h x = T (a_1, \ldots, a_T) (b_1, \ldots, b_T)$
Bag-of-Embeddings Encoding

\[ \bar{a} = \sum_{t=1}^{T} a_t \quad \bar{b} = \sum_{t=1}^{T} b_t \]
Can use RNNs/LSTMs to “encode” sentences

![Diagram of RNN/LSTM states and inputs]

Common choices:

\[ h_x = h_T \]  \quad \text{the last state}

\[ h_x = \sum_{t=1}^{T} h_t \]  \quad \text{sum/mean of the states}
Convolutional Nets

A Full Convolutional Neural Network (LeNet)

Sentence Encoding on SNLI

- Stanford Natural Language Inference [Bowman et al. 2015]

<table>
<thead>
<tr>
<th>Sentence encoding-based models</th>
<th>Params</th>
<th>Acc</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bowman et al. '15 100D LSTM encoders</td>
<td>220k</td>
<td>84.8</td>
<td>77.6</td>
</tr>
<tr>
<td>Bowman et al. '16 300D LSTM encoders</td>
<td>3.0m</td>
<td>83.9</td>
<td>80.6</td>
</tr>
<tr>
<td>Vendrov et al. '15 1024D GRU encoders w/ unsupervised 'skip-thoughts' pre-training</td>
<td>15m</td>
<td>98.8</td>
<td>81.4</td>
</tr>
<tr>
<td>Mou et al. '15 300D Tree-based CNN encoders</td>
<td>3.5m</td>
<td>83.3</td>
<td>82.1</td>
</tr>
<tr>
<td>Bowman et al. '16 300D SPINN-PI encoders</td>
<td>3.7m</td>
<td>89.2</td>
<td>83.2</td>
</tr>
<tr>
<td>Yang Liu et al. '16 600D (300+300) BiLSTM encoders</td>
<td>2.0m</td>
<td>86.4</td>
<td>83.3</td>
</tr>
<tr>
<td>Munkhdalai &amp; Yu '16b 300D NTI-SLSTM-LSTM encoders</td>
<td>4.0m</td>
<td>82.5</td>
<td>83.4</td>
</tr>
<tr>
<td>Yang Liu et al. '16 600D (300+300) BiLSTM encoders with intra-attention</td>
<td>2.8m</td>
<td>84.5</td>
<td>84.2</td>
</tr>
<tr>
<td>Munkhdalai &amp; Yu '16a 300D NSE encoders</td>
<td>3.0m</td>
<td>86.2</td>
<td>84.6</td>
</tr>
<tr>
<td>Qian Chen et al. '17 600D (300+300) Deep Gated Attn. BiLSTM encoders (code)</td>
<td>11.6m</td>
<td>90.5</td>
<td>85.5</td>
</tr>
<tr>
<td>Tao Shen et al. '17 300D Directional self-attention network encoders</td>
<td>2.4m</td>
<td>91.1</td>
<td>85.6</td>
</tr>
</tbody>
</table>
Learning and Optimization

• Typical strategy is a stochastic optimization technique:
  • Instead of computing gradient over whole data, compute only gradient over a small “minibatch”
  • Speeds up training, and the added randomness helps get out of local optima’

• Other common optimizers are:
  • Adagrad [Duchi et al. 2011]
  • ADAM [Kingma and Ba, 2014]
  and many more!
Dropout Regularization [Srivastava et al. 2014]

- Neural nets overfit very easily.

- Empirically, often better to train a massive neural network with lots of regularization than a smaller neural network

- Dropout [Srivastava et al. 2014] is a very popular regularization technique
  - Dropout random neurons in training
  - (Use all the neurons in test)
  - Intuitively some sort of model averaging
Dropout Regularization [Srivastava et al. 2014]

Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Figure 2: **Left:** A unit at training time that is present with probability $p$ and is connected to units in the next layer with weights $w$. **Right:** At test time, the unit is always present and the weights are multiplied by $p$. The output at test time is same as the expected output at training time.
Dropout Regularization [Srivastava et al. 2014]

Dropout setting that do not use dropout or unsupervised pretraining achieve an error of about 1.60% (Simard et al., 2003). With dropout the error reduces to 1.35%. Replacing logistic units with rectified linear units (ReLUs) (Jarrett et al., 2009) further reduces the error to 1.25%. Adding max-norm regularization again reduces it to 1.06%. Increasing the size of the network leads to better results. A neural net with 2 layers and 8192 units per layer gets down to 0.95% error. Note that this network has more than 65 million parameters and is being trained on a data set of size 60,000. Training a network of this size to give good generalization error is very hard with standard regularization methods and early stopping. Dropout, on the other hand, prevents overfitting, even in this case. It does not even need early stopping. Goodfellow et al. (2013) showed that results can be further improved to 0.94% by replacing ReLU units with maxout units. All dropout nets use $p = 0.5$ for hidden units and $p = 0.8$ for input units. More experimental details can be found in Appendix B.1.

Dropout nets pretrained with stacks of RBMs and Deep Boltzmann Machines also give improvements as shown in Table 2. DBM—pretrained dropout nets achieve a test error of 0.79% which is the best performance ever reported for the permutation invariant setting. We note that it possible to obtain better results by using 2-D spatial information and augmenting the training set with distorted versions of images from the standard training set. We demonstrate the effectiveness of dropout in that setting on more interesting data sets.

Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.
Challenges: End-to-end Learning

- Neural models tend to generalize poorly to examples that are different than the training distribution.

Example: Reading comprehension on SQUAD dataset [Rajpurkar et al., 2016]

Figure 1: An example from the SQuAD dataset. The BiDAF Ensemble model originally gets the answer correct, but is fooled by the addition of an adversarial distracting sentence (in blue).
Challenges: End-to-end Learning

- Neural models can often produce bizarre outputs that are not easily fixed or explainable.

https://medium.com/@felixhill/deep-consequences-fa823a588e97
Challenges: End-to-end Learning

Wiseman et al. 2017

<table>
<thead>
<tr>
<th>TEAM</th>
<th>WIN</th>
<th>LOSS</th>
<th>PTS</th>
<th>FG_PCT</th>
<th>RB</th>
<th>AS</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heat</td>
<td>11</td>
<td>12</td>
<td>103</td>
<td>49</td>
<td>47</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>Hawks</td>
<td>7</td>
<td>15</td>
<td>95</td>
<td>43</td>
<td>33</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PLAYER</th>
<th>AS</th>
<th>RB</th>
<th>PT</th>
<th>FG</th>
<th>FGA</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tyler Johnson</td>
<td>5</td>
<td>2</td>
<td>27</td>
<td>8</td>
<td>16</td>
<td>Miami</td>
</tr>
<tr>
<td>Dwight Howard</td>
<td>4</td>
<td>17</td>
<td>23</td>
<td>9</td>
<td>11</td>
<td>Atlanta</td>
</tr>
<tr>
<td>Paul Millsap</td>
<td>2</td>
<td>9</td>
<td>21</td>
<td>8</td>
<td>12</td>
<td>Atlanta</td>
</tr>
<tr>
<td>Goran Dragic</td>
<td>4</td>
<td>2</td>
<td>21</td>
<td>8</td>
<td>17</td>
<td>Miami</td>
</tr>
<tr>
<td>Wayne Ellington</td>
<td>2</td>
<td>3</td>
<td>19</td>
<td>7</td>
<td>15</td>
<td>Miami</td>
</tr>
<tr>
<td>Dennis Schroder</td>
<td>7</td>
<td>4</td>
<td>17</td>
<td>8</td>
<td>15</td>
<td>Atlanta</td>
</tr>
<tr>
<td>Rodney McGruder</td>
<td>5</td>
<td>5</td>
<td>11</td>
<td>3</td>
<td>8</td>
<td>Miami</td>
</tr>
<tr>
<td>Thabo Sefolosha</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>11</td>
<td>Atlanta</td>
</tr>
<tr>
<td>Kyle Korver</td>
<td>5</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>Atlanta</td>
</tr>
</tbody>
</table>

The Atlanta Hawks defeated the Miami Heat, 103 - 95, at Philips Arena on Wednesday. Atlanta was in desperate need of a win and they were able to take care of a shorthanded Miami team here. Defense was key for the Hawks, as they held the Heat to 42 percent shooting and forced them to commit 16 turnovers. Atlanta also dominated in the paint, winning the rebounding battle, 47 - 34, and outscoring them in the paint 58 - 26. The Hawks shot 49 percent from the field and assisted on 27 of their 43 made baskets. This was a near wire - to - wire win for the Hawks, as Miami held just one lead in the first five minutes. Miami ( 7 - 15 ) are as beat - up as anyone right now and it’s taking a toll on the heavily used starters. Hassan Whiteside really struggled in this game, as he amassed eight points, 12 rebounds and one blocks on 4 - of - 12 shooting ...
Conclusion

• Deep learning and end-to-end learning has changed NLP in recent years.

• However it does not come without drawbacks
  • No free lunch :)