The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

SkipFlow: Incorporating Neural Coherence Features for End-to-End Automatic Text Scoring

Yi Tay,[1] **Minh C. Phan,**[2] **Luu Anh Tuan,**[3] **Siu Cheung Hui**[4]
[1, 2, 4] Nanyang Technological University
School of Computer Science and Engineering, Singapore
[3] Institute for Infocomm Research, Singapore

Abstract

Deep learning has demonstrated tremendous potential for Automatic Text Scoring (ATS) tasks. In this paper, we describe a new neural architecture that enhances vanilla neural network models with auxiliary neural coherence features. Our new method proposes a new SkipFlow mechanism that models relationships between snapshots of the hidden representations of a long short-term memory (LSTM) network as it reads. Subsequently, the semantic relationships between multiple snapshots are used as auxiliary features for prediction. This has two main benefits. Firstly, essays are typically long sequences and therefore the memorization capability of the LSTM network may be insufficient. Implicit access to multiple snapshots can alleviate this problem by acting as a protection against vanishing gradients. The parameters of the SkipFlow mechanism also acts as an auxiliary memory. Secondly, modeling relationships between multiple positions allows our model to learn features that represent and approximate textual coherence. In our model, we call this *neural coherence* features. Overall, we present a unified deep learning architecture that generates neural coherence features as it reads in an end-to-end fashion. Our approach demonstrates state-of-the-art performance on the benchmark ASAP dataset, outperforming not only feature engineering baselines but also other deep learning models.

Introduction

Automated Text Scoring (ATS) systems are targeted at both alleviating the workload of teachers and improving the feedback cycle in educational systems. ATS systems have also seen adoption for several high-stakes assessment, e.g., the *e-rater system* (Attali and Burstein 2004) which has been used for TOEFL and GRE examinations. A successful ATS system brings about widespread benefits to society and the education industry. This paper presents a novel neural network architecture for this task.

Traditionally, the task of ATS has been regarded as a machine learning problem (Larkey 1998; Attali and Burstein 2004) which learns to approximate the marking process with supervised learning. Decades of ATS research follow the same traditional supervised text regression methods in which handcrafted features are constructed and subsequently passed into a machine learning based classifier. A wide assortment of features may commonly extracted from essays. Simple and intuitive features may include essay length, sentence length. On the other hand, intricate and complex features may also be extracted, e.g.., features such as grammar correctness (Attali and Burstein 2004), readability (Zesch, Wojatzki, and Scholten-Akoun 2015) and textual coherence (Chen and He 2013). However, these handcrafted features are often painstakingly designed, require a lot of human involvement and usually require laborious implementation for every new feature.

Deep learning based ATS systems have recently been proposed (Dong and Zhang 2016; Taghipour and Ng 2016; Alikaniotis, Yannakoudakis, and Rei 2016). A comprehensive study has been done in (Taghipour and Ng 2016) which demonstrated that neural network architectures such as the long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) and convolutional neural network (CNN) are capable of outperforming systems that extensively require handcrafted features. However, all of these neural models do not consider transition of an essay over time, i.e., logical flow and coherence over time. In particular, mainly semantic compositionality is modeled within the recursive operations in the LSTM model which compresses the input text repeatedly within the recurrent cell. In this case, the relationships between multiple points in the essay cannot be captured effectively. Moreover, essays are typically long sequences which pushes the limits of the memorization capability of the LSTM.

Hence, the objective of this work is a unified solution to the above mentioned problems. Our method alleviates two problems. The first is targeted at alleviating the inability of current neural network architectures to model flow, coherence and semantic relatedness over time. The second is aimed at easing the burden of the recurrent model. In order to do so, we model the relationships between multiple snapshots of the LSTM's hidden state over time. More specifically, as our model reads the essay, it models the semantic relationships between two points of an essay using a neural tensor layer. Eventually, *multiple* features of semantic relatedness are aggregated across the essay and used as auxiliary features for prediction.

The intuition behind our idea is as follows. Firstly, semantic relationships across sentences are commonly used as an indicator of writing flow and textual coherence (Wiemer-

Hastings and Graesser 2000; Higgins et al. 2004; Higgins and Burstein 2007; Chen and He 2013; Somasundaran, Burstein, and Chodorow 2014). As such, our auxiliary features (generated end-to-end) aim to capture the logical and semantic flow of an essay. This also provides a measure of *semantic similarity* aside from the flavor of *semantic compositionality* modeled by the base LSTM model.

Secondly, the additional parameters from the external tensor serve as an auxiliary memory for the network. As essays are typically long sequences, modeling the relationship between distant states with additional parameters can enhance memorization and improve performance of the deep architecture by allowing access to intermediate states, albeit implicitly. The semantic relevance scores can then be aggregated by concatenation and passed as an auxiliary feature to a fully-connected dense layer in the final layer of the network. As such, our architecture performs sentence modeling (compositional reading) and semantic matching in a unified end-to-end framework.

Our Contributions

The prime contributions of our paper are as follows:

- For the first time, we consider neural coherence features within the context of an end-to-end neural framework. Semantic similarity and textual coherence have a long standing history in ATS literature (Wiemer-Hastings and Graesser 2000; Higgins and Burstein 2007; Higgins et al. 2004). Our work incorporates this intuition into modern neural architectures.
- Aside from modeling coherence, our method also alleviates and eases the burden of the recurrent model by implicit access to hidden representations over time. This serves as a protection against vanishing gradient. Moreover, a better performance can be achieved with a smaller LSTM parameterization.
- We propose SKIPFLOW LSTM, a new neural architecture that incorporates the intuition of logical and semantic flow into the vanilla LSTM model. SKIPFLOW LSTM obtains state-of-the-art performance on the ASAP benchmark dataset. We also achieve an increase of 6% in performance over a strong feature engineering baseline. In the same experimental configuration, we achieve about 10% increase over a baseline LSTM model, outperforming more advanced extensions such as Multi-Layered LSTMs and attention-based LSTMs.

Related Work

Automated Text Scoring (ATS) systems have been deployed for high-stakes assessment since decades ago. Early high-stakes ATS systems include the Intelligent Essay Assessor (IEA) (Foltz et al. 2013) and Project Essay Grade (Page 1967; Shermis and Burstein 2003). Commercial ATS systems such as the *e-rater* (Attali and Burstein 2004) have been also deployed for GRE and TOEFL examinations.

Across the rich history of ATS research, supervised learning based ATS systems mainly rely on domain-specific feature engineering whereby lexical, syntactic and semantic features are designed by domain experts and subsequently extracted from essays. Then, a simple machine learning classifier trained on these feature vectors can be used to predict the grades of essays. Early work (Larkey 1998) treats ATS as a text categorization problem and uses a Naive Bayes model for grading while the *e-rater* system uses linear regression over handcrafted features. (Phandi, Chai, and Ng 2015) proposed a Bayesian Linear Ridge Regression approach for domain adaptation of essays.

The reliance on handcrafted features is a central theme to decades of ATS research. The complexity and ease of implementation of essay scoring features can be diverse. For example, length-based features are intuitive and simple to extract from essays. On the other hand, there are more complex features such as grammar correctness or lexical complexity. Features such as readability (Zesch, Wojatzki, and Scholten-Akoun 2015), textual and discourse coherence (Chen and He 2013; Somasundaran, Burstein, and Chodorow 2014) are also harder to design in which convoluted pipelines have to be built for feature extraction to be performed. As a whole, feature engineering is generally a laborious process, i.e., apart from designing features, custom code has to be written for each additional feature. For a comprehensive review of feature engineering in the context of ATS, we refer interested readers to (Zesch, Wojatzki, and Scholten-Akoun 2015).

Recently, motivated by the success of deep learning in many domains, several deep learning architectures for ATS have been proposed. (Taghipour and Ng 2016; Dong and Zhang 2016) empirically evaluated the performance of a myriad of deep learning models on the ATS tasks. In their work, models such as the recurrent neural network (RNN) and convolutional neural network (CNN) demonstrated highly competitive results without requiring any feature engineering. On the other hand, an adapted task-specific embedding approach was proposed in (Alikaniotis, Yannakoudakis, and Rei 2016) that learns semantic word embeddings while predicting essay grades. Subsequently, these adapted word embeddings are passed as input to a LSTM network for prediction. The attractiveness of neural text scoring stems from the fact that features are learned end-to-end, diminishing the need for laborious feature engineering to be performed.

Our work extends the vanilla model and enhances with the incorporation of *neural coherence* features. The concept of semantic similarity between sentences has been used to measure coherence in student essays (Higgins and Burstein 2007; Higgins et al. 2004). Textual coherence features have also been adopted in (Chen and He 2013) which measures the semantic similarity between nouns and proper nouns. Lexical chaining (Somasundaran, Burstein, and Chodorow 2014) has also been used for measuring discourse quality in student essays. Our work, however, is the first *neural coherence* model that incorporates these features into an end-to-end fashion. Different from traditional coherence features, our neural features form a part of an overall unified framework.

Our proposed approach is inspired by the field of semantic matching. In semantic matching, a similarity score is produced between two vectors and is often used in many NLP and IR applications. The usage of tensor layers and bilinear similarity is inspired by many of these works. For example, convolutional neural tensor network (CNTN) (Qiu and Huang 2015) and NTN-LSTM (Tay et al. 2017) are recently proposed architectures for question-answer pair matching. However, unlike ours, these works are mainly concerned with matching between two sentences and are often trained with two networks. The tensor layer, also known as the Neural Tensor Network (NTN), was first incepted as a compositional operator in Recursive Neural Networks for sentiment analysis (Socher et al. 2013b). Subsequently, it has also been adopted for rich and expressive knowledge base completion (Socher et al. 2013a). It has also seen adoption in end-to-end memory networks (Tay, Tuan, and Hui 2017). The NTN is parameterized by both a tensor and an ordinary linear layer in which the tensor parameters model multiple instances of second order interactions between two vectors. The adoption of the tensor layer in our framework is motivated by the strong empirical performance of NTN.

In our approach, we generate neural coherence features by performing semantic matching k times while reading. This can be interpreted as jointly matching and reading. These additional parameters can also be interpreted as an auxiliary memory which can also help and ease the burden of the LSTM memory. LSTMs are known to have difficulty in modeling long term dependencies[1] and due to their compositional nature, measuring relatedness and coherence between two points becomes almost impossible. Moreover, our SkipFlow mechanism serves as an additional protection against the vanishing gradient problem by exposing hidden states to deeper layers. In a similar spirit, attention mechanisms (Bahdanau, Cho, and Bengio 2014) learn a weighted combination of hidden states across all time steps and produces a global feature vector. However, our approach learns auxiliary features that are used for prediction.

Our SkipFlow LSTM Model

In this section, we introduce the overall model architecture of SkipFlow. Figure 1 depicts the proposed architecture of our model.

Embedding Layer

Our model accepts an essay and the target score as a training instance. Each essay is represented as a fixed-length sequence in which we pad all sequences to the maximum length. Let L be the maximum essay length. Subsequently, each sequence is converted into a sequence of low-dimensional vectors via the embedding layer. The parameters of the embedding layer are defined as $W_e \in \mathbb{R}^{|V| \times N}$ where $|V|$ is the size of the vocabulary and N is the dimensionality of the word embeddings.

Long Short-Term Memory (LSTM)

The sequence of word embeddings obtained from the embedding layer is then passed into a long short-term memory (LSTM) network (Hochreiter and Schmidhuber 1997).

$$h_t = LSTM(h_{t-1}, x_i) \qquad (1)$$

where x_t and h_{t-1} are the input vectors at time t. The LSTM model is parameterized by output, input and forget gates, controlling the information flow within the recursive operation. For the sake of brevity, we omit the technical details of LSTM which can be found in many related works. At every time step t, LSTM outputs a hidden vector h_t that reflects the semantic representation of the essay at position t. To select the final representation of the essay, a temporal mean pool is applied to all LSTM outputs.

SkipFlow Mechanism for Generating Neural Coherence Features

In this section, we describe the process of generating neural coherence features within our end-to-end framework.

Skipping and Relevance Width In our proposed approach, the relationships between two positional outputs of LSTM across time steps are modeled via a parameterized compositional technique that generates a *coherence feature*. Let δ be a hyperparameter that controls the relevance width of the model. For each LSTM output, we select pairs of sequential outputs of width δ, i.e., $\{(h_i, h_{i+\delta}), (h_{i+\delta}, h_{i+2\delta}), (h_{i+2X}, h_{i+3\delta}), ..\}$ are the tuples from the outputs that are being composed, h_t denotes the output of LSTM at time step t. In our experiments, the starting position[2] is fixed at $i = 3$. For the sake of simplicity, if the width δ exceeds the max length, we loop back to the beginning of the essay in a circular fashion. The rationale for fixed length matching is as follows. Firstly, we want to limit the amount of preprocessing required as determining important key points such as nouns and pronouns require preprocessing of some sort. Secondly, maintaining specific indices for each essay can be cumbersome in the context of batch-wise training of deep learning models using libraries restricted by static computational graphs. Finally, LSTMs are memory-enabled models and therefore, intuitively, a slight degree of positional impreciseness should be tolerable.

Neural Tensor Layer We adopt a tensor layer to model the relationship between two LSTM outputs. The tensor layer is a parameterized composition defined as follows:

$$s_i(a, b) = \sigma(u^T f(v_a^T M^{[1:k]} v_b + V[v_a, v_b] + b)) \qquad (2)$$

where f is a non-linear function such as $tanh$. $M^{[1:k]} \in \mathbb{R}^{n \times n \times k}$ is a tensor (3d matrix). $a, b \in \mathbb{R}^d$ are the vector outputs of LSTM at two time steps of δ-width apart where d is the dimensionality of LSTM parameters. For each slice of the tensor M, each bilinear tensor product $v_a^T M_k v_b$ returns a scalar to form a k dimensional vector. σ is the sigmoid function which constraints the output to $[0, 1]$. The other parameters are the standard form of a neural network. In our model, two vectors (outputs of LSTM) are passed through the tensor layer and returns a similarity score $s_i(h_i, h_{i+X}) \in [0, 1]$ that

[1] Essays are typically long documents spanning 300-800 words on average.

[2] We set the starting position $i > 0$ to avoid matching against the initial state.

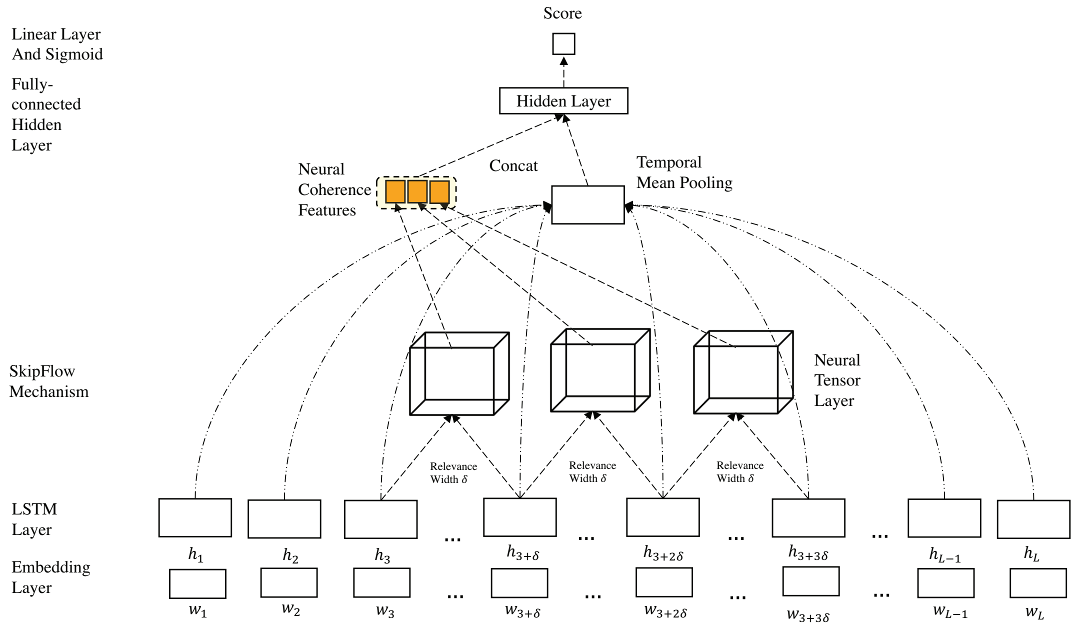
Score
Linear Layer And Sigmoid
Hidden Layer
Fully-connected Hidden Layer
Neural Coherence Features
Concat
Temporal Mean Pooling
SkipFlow Mechanism
Neural Tensor Layer
Relevance Width δ
Relevance Width δ
Relevance Width δ
LSTM Layer
Embedding Layer
h_1 h_2 h_3 ... $h_{3+\delta}$ ... $h_{3+2\delta}$ ... $h_{3+3\delta}$ ... h_{L-1} h_L
w_1 w_2 w_3 ... $w_{3+\delta}$ ... $w_{3+2\delta}$ ... $w_{3+3\delta}$ ... w_{L-1} w_L

Figure 1: Illustration of our proposed SKIPFLOW LSTM model with width δ. Note that tensors depicted are **shared** parameters and there is only one tensor parameter in the entire architecture.

determines the coherence feature between the two vectors. The parameters of the tensor layer are shared throughout all output pairs. The usage of bilinear product enables dyadic interaction between vectors through a similarity matrix. This enables a rich interaction between hidden representations. Moreover, the usage of multiple slices encourages different aspects of this relation to be modeled.

Fully-connected Hidden Layer

Subsequently, all the scalar values $s_1, s_2, \cdots, s_n$ that are obtained from the tensor layer are concatenated together to form the neural coherence feature vector. n is the number of times that coherence is being measured, depending on the relevance width δ and maximum sequence length L. Recall that the essay representation is obtained from a mean pooling over all hidden states. This essay vector is then concatenated with the coherence feature vector. This vector is then passed through a fully connected hidden layer defined as follows:

$$h_{out} = f(\mathbf{W}_h([e, s_1, s_2, ..., s_n])) + b_h \tag{3}$$

where $f(.)$ is a non-linear activation such as *tanh* or *relu*, $\mathbf{W}_h$ and b_h are the parameters of the hidden layer. e is the final essay representation obtained from temporal mean pooling and $s_1, s_2, ..., s_n$ are the scalar values obtained from the neural tensor layer, i.e., each scalar value is the matching score from $\{(h_i, h_{i+\delta}), (h_{i+\delta}, h_{i+2\delta}), (h_{i+2\delta}, h_{i+3\delta}), ..\}$.

Linear Layer with Sigmoid

Finally, we pass h_{out} into a final linear regression layer. The final layer is defined as follows:

$$y_{out} = \sigma\left(\mathbf{W}_f([h_{out}])\right) + b_f \tag{4}$$

where $\mathbf{W}_f, b_f$ are parameters of the final linear layer, σ is the sigmoid function and $y_{out} \in [0, 1]$. The output at this final layer is the normalized score of the essay. Following (Taghipour and Ng 2016), the bias is set to the mean expected score.

Learning and Optimization

Our network optimizes the mean-square error which is defined as:

$$MSE(z, z^*) = \frac{1}{N}\sum_{i=1}^{N}(z_i - z_i^*)^2 \tag{5}$$

where z_i^* is the gold standard score and z_i is the model output. The parameters of the network are then optimized using gradient descent.

Experimental Evaluation

In this section, we describe our experimental procedure, dataset and empirical results.

Dataset

We use the ASAP (Automated Student Assessment Prize) dataset for experimental evaluation. This comes from the competition which was organized and sponsored by the William and Flora Hewlett Foundation (Hewlett) and ran on Kaggle from 10/2/12 to 30/4/12. This dataset contains 8 essay prompts as described in Table 1. Each prompt can be interpreted as a different essay topic along with a different genre such as argumentative or narrative.

Prompt	#Essays	Avg Length	Scores
1	1783	350	2-12
2	1800	350	1-6
3	1726	150	0-3
4	1772	150	0-3
5	1805	150	0-4
6	1800	150	0-4
7	1569	250	0-30
8	723	650	0-60

Table 1: Statistics of ASAP dataset. Scores denote the range of possible marks in the dataset.

Experimental Setup

We use 5-fold cross validation to evaluate all systems with a 60/20/20 split for train, development and test sets. The splits are provided by (Taghipour and Ng 2016) and the experimental procedure is followed closely. We train all models for 50 epochs and select the best model based on the performance on the development set. The vocabulary is restricted to the 4000 most frequent words. We tokenize and lowercase text using NLTK[3], and normalize all score range to within [0,1]. The scores are rescaled back to the original prompt-specific scale for calculating Quadratic Weighted Kappa (QWK) scores. Following (Taghipour and Ng 2016), the evaluation is conducted in prompt-specific fashion. Even though training prompts together might seem ideal, it is good to note that each prompt can contain genres of essays that are very contrastive such as narrative or argumentative essays. Additionally, prompts can have different marking schemes and level of students. As such, it would be extremely difficult to train prompts together.

Evaluation Metric

The evaluation metric used is the Quadratic Weighted Kappa (QWK) which measures agreement between raters and is a commonly used metric for ATS systems. The QWK score ranges from 0 to 1 but becomes negative if there is less agreement than expected by chance. The QWK score is calculated as follows. First, an $N \times N$ histogram matrix $\mathbf{O}$ is constructed. Next, a weight matrix $\mathbf{W}_{i,j} = \frac{(i-j)^2}{(N-1)^2}$ is calculated that corresponds to the difference between rater's scores where i and j are reference ratings by the annotator and the ATS system. Finally, another $N \times N$ histogram matrix $\mathbf{E}$ is constructed assuming no correlation between rating scores. This is done using an outer product between each rater's histogram vector and normalized such that $sum(\mathbf{E}) = sum(\mathbf{O})$. Finally, the QWK score is calculated as $\kappa = 1 - \frac{\sum_{i,j} w_{i,j} O_{i,j}}{\sum_{i,j} w_{i,j} E_{i,j}}$.

Baselines and Implementation Details

In this section, we discuss the competitor algorithms that are used as baselines for our model.

- **EASE** - The major non deep learning system that we compare against is the *Enhanced AI Scoring Engine* (EASE). This system is publicly available, open-source[4] and also took part in the ASAP competition and ranked third amongst 154 participants. EASE uses manual feature engineering and applies different regression techniques over the handcrafted features. Examples of the features of EASE include length-based features, POS tags and word overlap. We report the results of EASE with the settings of Support Vector Regression (SVR) and Bayesian Linear Ridge Regression (BLRR).
- **CNN** - We implemented a CNN model using $1D$ convolutions similar to (Taghipour and Ng 2016). We use a filter width of 3 and a final embedding dimension of 50. The outputs from the CNN model are passed through a mean pooling layer and finally through the final linear layer.
- **RNN / GRU / LSTM** - Similar to (Taghipour and Ng 2016), we implemented and tested all RNN variants, namely the vanilla RNN, GRU (Gated Recurrent Unit) and LSTM. We compare mainly on two settings of *mean pooling* and *last*. In the former, the average vector of all outputs from the model is used. In the latter, only the last vector is used for prediction. A fully connected linear layer connects this feature vector to the final sigmoid activation function. We use a dimension of 50 for all RNN/GRU/LSTM models.
- **LSTM Variants** - Additionally, we also compare with multiple LSTM variants such as the Attention Mechanism (ATT-LSTM), Bidirectional LSTM (BI-LSTM) and the Multi-Layer LSTM (ML-LSTM). We use the AttentionCellWrapper implementation in TensorFlow with an attention width of 10.

Our Models We compare two settings of our model, namely the bilinear and tensor composition. They are denoted as SkipFlow LSTM (Bilinear) and SkipFlow LSTM (Tensor) respectively. The bilinear setting is formally described as $s(a, b) = a^T \, \mathbf{M} \, b$, where a, b are vectors of two distant LSTM outputs and $\mathbf{M}$ is a similarity matrix. The bilinear setting produces a scalar value, similar to the output of the tensor layer. The tensor layer, aside from the combination of multiple bilinear products, also includes a separate linear layer along with a non-linear activation function. For the tensor setting, the number of slices of the tensor is tuned amongst $\{2, 4, 6, 8\}$. For both models, the hidden layer is set to 50. There is **no** dropout for this layer and the bias vector is set to 0. The relevance width of our model δ is set amongst $\{20, 50, 100\}$. In addition, to demonstrate the effectiveness and suitability of the LSTM model for joint modeling of semantic relevance, we conduct further experiments with the SkipFlow extension of the CNN model which we call the SkipFlow CNN. Similarly, we apply the same procedure on the convolved representations. Aside from swapping the LSTM for a CNN, the entire architecture remains identical.

To facilitate fair comparison, we implemented and evaluated **all** deep learning models ourselves in TensorFlow. We also implemented the architectures of (Taghipour and Ng 2016) which we denote with †. For training, the ADAM optimizer (Kingma and Ba 2014) was adopted with a learn-

[3] http://www.nltk.org

[4] http://github.com/edx/ease

		Dataset / Prompts								
ID	Model	1	2	3	4	5	6	7	8	Average
1	RNN (Last)	0.524	0.025	0.004	0.001	0.001	0.004	0.165	0.094	0.102
2	GRU (Last)	0.521	0.265	0.274	0.678	0.441	0.563	0.420	0.182	0.418
3	LSTM (Last)	0.319	0.200	0.317	0.690	0.389	0.522	0.423	0.189	0.467
4	RNN (Mean)	0.597	0.488	0.603	0.745	0.740	0.759	0.741	0.489	0.645
5	GRU (Mean)	0.608	0.515	0.593	0.737	0.725	0.738	0.733	0.515	0.646
6	LSTM (Mean)	0.583	0.523	0.591	0.757	0.737	0.756	0.706	0.514	0.646
7	BI-LSTM (Mean)	0.794	0.625	0.665	0.674	0.776	0.613	0.679	0.506	0.667
8	SkipFlow CNN (Bilinear)	0.780	0.620	0.628	0.719	0.775	0.721	0.729	0.409	0.673
9	ML-LSTM (L=2)	0.800	0.630	0.667	0.687	0.774	0.612	0.728	0.545	0.676
10	ML-LSTM (L=3)	0.700	0.554	0.641	0.753	0.780	0.764	0.752	0.558	0.688
11	SkipFlow CNN (Tensor)	0.782	0.657	0.666	0.727	0.781	0.756	0.757	0.440	0.696
12	EASE* (SVR)	0.781	0.630	0.621	0.749	0.782	0.771	0.727	0.534	0.699
13	EASE* (BLRR)	0.761	0.621	0.606	0.742	0.784	0.775	0.730	0.617	0.705
14	RNN† (d=300)	0.793	0.667	0.591	0.752	0.713	0.770	0.784	0.576	0.706
15	CNN	0.789	0.674	0.590	0.742	0.726	0.757	0.771	0.614	0.708
16	GRU† (d=300)	0.792	0.666	0.592	0.757	0.727	0.753	0.779	0.649	0.714
17	LSTM$^{\dagger,\phi}$ (d=300)	0.792	0.676	0.583	0.745	0.725	0.765	0.780	0.651	0.715
18	ATT-LSTM	0.793	0.660	0.664	0.768	0.800	0.789	0.788	0.637	0.737
19	ML-LSTM (L=4)	0.793	0.682	0.643	0.769	0.777	0.772	0.789	0.683	0.739
20	SkipFlow LSTM* (Bilinear)	0.830	0.678	0.677	0.778	0.795	0.807	0.790	0.670	0.753
21	SkipFlow LSTM* (Tensor)	**0.832**	**0.684**	**0.695**	**0.788**	**0.815**	**0.810**	**0.800**	**0.697**	**0.764**

Table 2: Experimental results of all compared models on the ASAP dataset. Best result is in bold and 2nd best is underlined. Results are sorted by average performance. † denotes our implementation of a model from (Taghipour and Ng 2016), ϕ denotes the baseline for statistical significance testing, $*$ denotes statistically significant improvement. $\star$ denotes non deep learning baselines.

ing rate amongst $\{0.01, 0.001, 0.0001\}$ and mini-batch size amongst $\{64, 128, 256\}$. The gradient of the norm is clipped to 1.0. The sequences are all padded with zero vectors up till the total maximum length[5]. We use the same embeddings from (Taghipour and Ng 2016) and set them to trainable parameters. All experiments are conducted on a Linux machine running two GTX1060 GPUs.

Experimental Results

Table 2 reports the empirical results of all deep learning models. First, it is clear that the mean pooling is significantly more effective as compared to the last LSTM output. In the *last* setting, the performance of RNN is significantly worse compared to LSTM and GRU possibly due to the weaker memorization ability. However, the performance of LSTM, GRU and RNN are similar using the *mean pooling* setting. This is indeed reasonable because the adoption of a mean pooling layer reduces the dependency of the model's memorization ability due to implicit access to all intermediate states. Overall, we observe that the performance of LSTM and GRU is quite similar with either *mean pooling* or *last* setting. Finally, we note that the performance of CNN is considerably better than RNN-based models. We also observe that a multi-layered LSTM performs considerably better than a single-layered LSTM. We also observe that adding layers also increases the performance. On the other hand, the bidirectional LSTM did not yield any significant improvements in performance. The performance of ATT-LSTM is notably much higher than the base LSTM. The best performing LSTM model is a multi-layered LSTM with 4 layers.

Additionally, we observe that SkipFlow LSTM (Tensor) outperforms the baseline LSTM (Mean) by almost 10% in QWK score. Evidently, we see the effectiveness of our proposed approach. The tensor setting of SkipFlow LSTM is considerably better than the bilinear setting which could be due to the richer modeling capability of the tensor layer. On the other hand, we also note that the SkipFlow extension of CNN model did not increase the performance of CNN. As such, we see that the SkipFlow mechanism seems to only apply to the compositional representations of recurrent-based models. Moreover, the width of the CNN is 3 which might be insufficient to offset the impreciseness of our fixed width matching.

Finally, we compare SkipFlow LSTM with deep learning models[6] of (Taghipour and Ng 2016). The key difference is that these models (denoted with † in Table 2) have a higher dimensionality of $d = 300$. First, we observe that a higher dimensionality improves performance over $d = 50$. Our SkipFlow LSTM (Tensor) outperforms LSTM$^{\dagger}(d = 300)$ significantly by 5%. The performance of LSTM$^{\dagger}(d = 300)$ and GRU$^{\dagger}(d = 300)$ are in fact identical and are only slightly better than feature engineering baselines such as EASE (BLRR). We also observe that ATT-LSTM and ML-LSTM (L=4) with both $d = 50$ also consistently outperform LSTM† and GRU†. Conversely, our SkipFlow LSTM (Tensor) model outperforms the best feature engineering

[5] We used the dynamic RNN in TensorFlow in our implementation.

[6] For fair comparison, we only compare against single neural models and not against ensemble approaches.

baseline (EASE) by about 6%.

Comparison against Published Results

Finally we compare with published state-of-the-art results from (Taghipour and Ng 2016). While our reproduction of the vanilla LSTM ($d = 300$)could not achieve similar results, our SkipFlow model still outperforms the reported results with a much smaller parameterization ($d = 50$ instead of $d = 300$). SkipFlow also remains competitive to an ensemble of 20 models (CNN + LSTM) with just a single model.

System	QWK
LSTM w/o MOT	0.540
LSTM + Attention	0.731
CNN + LSTM	0.708
BiLSTM	0.699
LSTM (d=300)	0.746
10 x CNN ensemble	0.726
10 x LSTM ensemble	0.756
20 x LSTM + CNN Ensemble	0.761
SkipFlow LSTM (Tensor)	**0.764**

Table 3: Comparision against published works. Single model of SkipFlow outperforms model ensembles.

Runtime and Memory

Table 4 reports the runtime and parameters of several LSTM variants. We observe that the runtime of our models only incur a small cost of 1-2 seconds over the baseline LSTM model. Our model also not only outperforms LSTM† and ML-LSTM (L=4) in terms of QWK score but also in terms of memory footprint and runtime. SkipFlow is also faster then the attention mechanism (ATT-LSTM).

Model	Epoch/s	# Param
LSTM	8s	13K
LSTM† (d=300)	12s	450K
BI-LSTM	18s	13K
ML-LSTM (L=4)	27s	50K
ATT-LSTM	20s	15K
SkipFlow LSTM (Bilinear)	9s	18K
SkipFlow LSTM (Tensor)	10s	25K

Table 4: Comparisons of runtime and parameter size on prompt 1. All models are $d = 50$ unless stated otherwise.

Effect of Hyperparameters

In this section, we investigate the effect of hyperparameters, namely the number of tensor slices k and the relevance width δ. While we report the results on the test set, it is good to note that the curves on the development set follow exactly the same rank and pattern.

Effect of Tensor Slices on Performance

Figure 2 shows the effect of the number of tensor slices (k) on performance. The prompts[7] are separated into two graphs due to the different ranges of results. The optimal k value is around 4 to 6 across all prompts. Intuitively, a small k (2) and an overly large k (8) often result in bad performance. The exception lies in prompts 5 and 6 where increasing the number of slices to $k = 8$ either improved or maintained the QWK score.

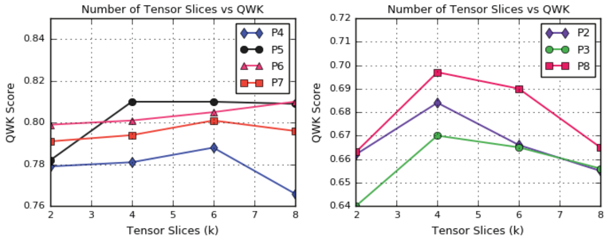

Figure 2: Effect of tensor slices on performance with $\delta = 100$.

Effect of Relevance Width δ on Performance

Figure 3 shows the influence of the hyperparameter relevance width δ on the performance. We observe that a small width produces worse results as compared to a large width. This is possibly due to insufficient tensor parameters or underfitting in lieu of a large number of matches is required with a small width. For example, consider prompt 8 that has the longest essays. Adopting $\delta = 20$ for prompt 8 requires about $\approx$ 300 to 400 comparisons that have to be modeled by a fixed number of tensor parameters. A quick solution is to increase the size of the tensor. However, raising both δ and k would severely increase computational costs. Hence, a trade-off has to be made between δ and k. Empirical results show that a value from 50 to 100 for δ works best with $k = 4$.

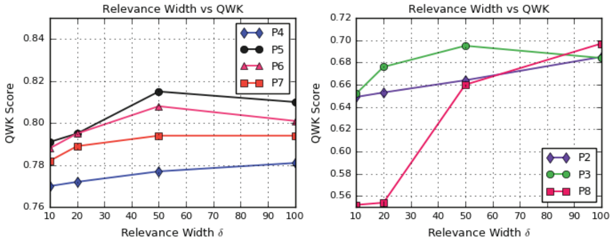

Figure 3: Effect of relevance width δ on performance with tensor slices $k = 4$.

Conclusion

In this paper, we proposed a new deep learning model for Automatic Text Scoring (ATS). We incorporated the intuition of textual coherence in neural ATS systems. Our model,

[7] We omit prompt 1 because it has a much higher score which distorts the visualization of our graphs.

SKIPFLOW LSTM, adopts parameterized tensor compositions to model the relationships between different points within an essay, generating neural coherence features that can support predictions. Our approach outperforms a baseline LSTM on the same setting by approximately 10% and also produces significantly better results as compared to multi-layered and attentional LSTMs. In addition, we also achieve a significant 6% improvement over feature engineering baselines.

Acknowledgements

The authors would like to thank anonymous reviewers of AAAI 2018, EMNLP 2017 and ACL 2017 whom have helped improve this work.

References

Alikaniotis, D.; Yannakoudakis, H.; and Rei, M. 2016. Automatic text scoring using neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

Attali, Y., and Burstein, J. 2004. Automated essay scoring with e-rater® v. 2.0. *ETS Research Report Series* 2004(2).

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Chen, H., and He, B. 2013. Automated essay scoring by maximizing human-machine agreement. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, 1741–1752.

Dong, F., and Zhang, Y. 2016. Automatic features for essay scoring - an empirical study. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, 1072–1077.

Foltz, P. W.; Streeter, L. A.; Lochbaum, K. E.; and Landauer, T. K. 2013. Implementation and applications of the intelligent essay assessor. *Handbook of automated essay evaluation* 68–88.

Higgins, D., and Burstein, J. 2007. Sentence similarity measures for essay coherence.

Higgins, D.; Burstein, J.; Marcu, D.; and Gentile, C. 2004. Evaluating multiple aspects of coherence in student essays.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

Larkey, L. S. 1998. Automatic essay grading using text categorization techniques. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, 90–95. ACM.

Page, E. B. 1967. Grading essays by computer: Progress report. In *Proceedings of the invitational Conference on Testing Problems*.

Phandi, P.; Chai, K. M. A.; and Ng, H. T. 2015. Flexible domain adaptation for automated essay scoring using correlated linear regression. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, 431–439.

Qiu, X., and Huang, X. 2015. Convolutional neural tensor network architecture for community-based question answering. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 1305–1311.

Shermis, M. D., and Burstein, J. C. 2003. *Automated essay scoring: A cross-disciplinary perspective*. Routledge.

Socher, R.; Chen, D.; Manning, C. D.; and Ng, A. Y. 2013a. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, 926–934.

Socher, R.; Perelygin, A.; Wu, J. Y.; Chuang, J.; Manning, C. D.; Ng, A. Y.; and Potts, C. 2013b. Recursive deep models for semantic compositionality over a sentiment treebank. Citeseer.

Somasundaran, S.; Burstein, J.; and Chodorow, M. 2014. Lexical chaining for measuring discourse coherence quality in test-taker essays.

Taghipour, K., and Ng, H. T. 2016. A neural approach to automated essay scoring. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, 1882–1891.

Tay, Y.; Phan, M. C.; Luu, A. T.; and Hui, S. C. 2017. Learning to rank question answer pairs with holographic dual LSTM architecture. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, 695–704.

Tay, Y.; Tuan, L. A.; and Hui, S. C. 2017. Dyadic memory networks for aspect-based sentiment analysis. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, 107–116.

Wiemer-Hastings, P., and Graesser, A. C. 2000. Select-a-kibitzer: A computer tool that gives meaningful feedback on student compositions. *Interactive learning environments* 8(2):149–169.

Zesch, T.; Wojatzki, M.; and Scholten-Akoun, D. 2015. Task-independent features for automated essay grading. In *Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications, BEA@NAACL-HLT 2015, June 4, 2015, Denver, Colorado, USA*, 224–232.