Combining CNNs and Pattern Matching for Question Interpretation in a Virtual Patient Dialogue System

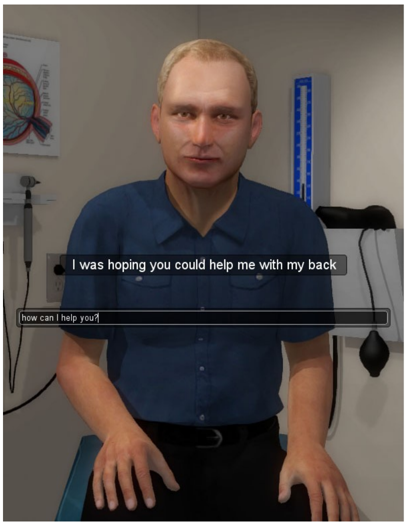
Lifeng Jin[1], **Michael White**[1], **Evan Jaffe**[1], **Laura Zimmerman**[2] and **Douglas Danforth**[2]
[1]Department of Linguistics, [2]Department of Family Medicine
The Ohio State University, Columbus, OH, USA
{jin, white, jaffe}@ling.osu.edu
zimmerman.411@osu.edu, doug.danforth@osumc.edu

Abstract

For medical students, virtual patient dialogue systems can provide useful training opportunities without the cost of employing actors to portray standardized patients. This work utilizes word- and character-based convolutional neural networks (CNNs) for question identification in a virtual patient dialogue system, outperforming a strong word- and character-based logistic regression baseline. While the CNNs perform well given sufficient training data, the best system performance is ultimately achieved by combining CNNs with a hand-crafted pattern matching system that is robust to label sparsity, providing a 10% boost in system accuracy and an error reduction of 47% as compared to the pattern-matching system alone.

1 Introduction

Standardized Patients (SPs) are actors who play the part of a patient with a specific medical history and pathology. Medical students interact with SPs to train skills like taking a patient history and developing a differential diagnosis. However, SPs are expensive and can behave inconsistently from student to student. A virtual patient dialogue system aims to overcome these issues as well as provide a means of automated evaluation of the medical student's interaction with the patient (see Figure 1).



I was hoping you could help me with my back

how can I help you?

Figure 1: Virtual Patient avatar used to train medical students

Previous work with a hand-crafted pattern-matching system called ChatScript (Danforth et al., 2009, 2013) used a 3D avatar and allowed for students to input questions using text or speech. ChatScript matches input text using hand-written patterns and outputs a scripted response for each input question identified by the system. While pattern matching with ChatScript can achieve relatively high accuracy with sufficient pattern-writing skill and effort, it is unable to take advantage of large amounts of training data, somewhat brittle regarding misspellings, and difficult to maintain as new questions and patterns are added.

With an apparent plateau in system performance, this work explores new data-driven methods. In particular, we use convolutional neural networks with both words and characters as input, demonstrating a significant improvement in overall question identification accuracy relative to a strong multiclass logistic regression baseline. Furthermore, inspired by the different error patterns between the ChatScript and CNNs, we develop a simple system combination using a binary classifier that results in the highest overall performance, achieving a remarkable 47% reduction in error in comparison to the ChatScript system alone. Frequency quantile analysis shows that the hybrid system is able to leverage the relatively higher performance of ChatScript on the infrequent label items, while also taking advantage of the CNN system's superior accuracy where more data is available for training.

2 Related Work

Question identification has been formulated as at least two distinct tasks. Multi-class logistic regression is a standard approach that can take advantage of class-specific features but requires a good amount of training data for each class. A pairwise setup involves a more general binary classification decision which is then made for each label, choosing the highest confidence match.

Early work (Ravichandran et al., 2003) found that treating a question answering task as a maximum entropy re-ranking problem outperformed using the same system as a classifier. DeVault et al. (2011) observed maximum entropy systems performed well with simple n-gram features. Jaffe et al. (2015) explored a log-linear pairwise ranking model for question identification and found it to outperform a multiclass baseline along the lines of DeVault et al. However, Jaffe et al. (2015) used a much smaller dataset with only about 915 user turns, less than one-fourth as many as in the current dataset. For this larger dataset, multiclass logistic regression outperforms a pairwise ranking model. With no pairwise comparisons, a multiclass classifier is also much faster, lending itself to real-time use.

It is probable that multiclass vs. pairwise approaches' overall effectiveness depends on the amount of training data; pairwise ranking methods have potential advantages for cross-domain and one-shot learning tasks (Vinyals et al., 2016) where data is sparse or non-existent. In the closely related task of short-answer scoring, Sakaguchi et al. (2015a) found that pairwise methods could be effectively combined with regression-based approaches to improve performance in sparse-data cases.

Other work involving dialogue utterance classification has traditionally required a large amount of data. For example, Suendermann et al. (2009) acquired 500,000 dialogues with over 2 million utterances, observin that statistical systems outperform rule-based ones as the amount of data increases. Crowdsourcing for collecting additional dialogues (Ramanarayanan et al., 2017) could alleviate data sparsity problems for rare categories by providing additional training examples, but this technique is limited to more general domains that do not require special training/skills. In the current medical domain, workers on common crowdsourcing platforms are unlikely to have the expertise required to take a patient's medical history in a natural way, so any data collected with this method would likely suffer quality issues and fail to generalize to real medical student dialogues. Rossen and Lok (2012) have developed an approach for collecting dialogue data for virtual patient systems, but their approach does not directly address the issue that even as the number of dialogues collected increases, there can remain a long tail of relevant but infrequently asked questions.

CNNs have been used to great effect for image identification (Krizhevsky et al., 2012) and are becoming common for natural language processing. In general, CNNs are used for convolution over input language sequences, where the input is often a matrix representing a sequence word embeddings (Kim, 2014). Intuitively, word embedding kernels are convolving n-grams, ultimately generating features that represent n-grams over word vectors of length equal to the kernel width. CNNs are very popular in systems for tasks like paraphrase detection (Yin and Schütze, 2015; Yin et al., 2016; He et al., 2015), community question answering (Das et al., 2016; Barbosa et al., 2016) and even machine translation (Gehring et al., 2017). Character-based models that embed individual characters as input units are also possible, and have been used for language modeling (Kim et al., 2016) to good effect. It is worth noting that character sequences are more robust to spelling errors and potentially have the same expressive capability as word sequences given long enough character sequences.

3 Dataset

The dataset consists of 94 dialogues of medical students interacting with the ChatScript system. The ChatScript system has been deployed in a medical school to assess student's ability to interact with patients through a text-based interface and the questions typed by the students and the responses given by ChatScript, which then are hand-corrected by annotators, form this dataset. There are 4330 total user turns, with a mean of 46.1 turns per dialogue. Each turn consists of the question

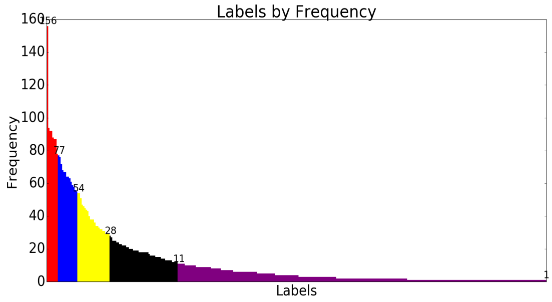


Figure 2: Label frequency distribution is extremely long-tailed, with few frequent labels and many infrequent labels. Values are shown above quintile boundaries.

the student asked, ChatScript's automatic label (with hand-correction) and the scripted response associated with the label. An example turn could be represented with the tuple, *('hello mr. wilkins, how are you doing today?', 'how are you', 'well i would be doing pretty well if my back weren't hurting so badly.')*. The task is to predict the label of the asked question.

There are 359 unique labels, with a mean of 12 instances per label, median of 4, and large standard deviation of 20. Of note, the distribution of labels is extremely long-tailed (Figure 2), with 8 of the most common labels accounting for nearly 20% of the data, while the bottom 20% includes 265 infrequent labels. The most frequent label occurs 156 times.

4 The CNN model

We now turn to the structure of our model. The main model used in this work follows Kim (2014). There are four layers in the model: an embedding layer, a convolution layer, a max-pooling layer and a linear layer. Let $\mathbf{x}_i \in \mathbb{R}^k$ a k-dimensional embedding for the i-th element of the sequence, i.e. the i-th word or character. The representation of a sentence, $\mathbf{S}_j \in \mathbb{R}^{|s_j| \times k}$ is the concatenation of all the embeddings of the elements in the sentence s_j. The multichannel setup, shown in Kim (2014) as marginally effective, is not used in this work. The following equations will all work on sentence $\mathbf{S}_j$, thus j is dropped for clarity.

A convolutional kernel is defined as a filter $\mathbf{w} \in \mathbb{R}^{hk}$ which slides across the sentence matrix $\mathbf{S}$ to produce a feature map. Because the kernel is as wide as the embeddings, it will only produce one value for each window.

$$c_i = \sigma(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b) \qquad (1)$$

In Eq. 1, b is a scalar and σ is a non-linearity. The feature map $\mathbf{c} \in \mathbb{R}^{|s|-h+1}$ for this kernel is the concatenation of all the feature values from the convolution. In order to maintain fixed dimension for the output, max-over-time pooling (Collobert et al., 2011) is applied to the feature map and the maximum value $\hat{c}$ is extracted from $\mathbf{c}$.

Because there are many kernels for each kernel height h, the output from a group of kernels with the same height is $\mathbf{o}_h = [\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_{n_h}]$, where n_h is the number of kernels for the kernel width h.

We concatenate all the outputs from all the kernels into a single vector $\mathbf{o} \in \mathbb{R}^N$ where $N = \sum_h n_h$, and apply a linear transformation with the softmax non-linearity to it as the final fully-connected neural network layer for the CNN.

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{W}_l \mathbf{o} + \mathbf{b}_l) \qquad (2)$$

where $W_l \in \mathbb{R}^{N \times m}$ is the weight matrix of the final layer, $\mathbf{b}_l \in \mathbb{R}^m$ is the bias term for the final layer, and m is the number of classes that we are trying to predict.

4.1 Regularization

We follow Kim (2014) for regularization strategies. Dropout (Srivastava et al., 2014) prevents the final layer from overfitting to training data by randomly setting some input values to zero according to a Bernoulli distribution with parameter p. We adopt this strategy and put a dropout layer between the max-pooling layer and the final linear layer.

Kim (2014) also applies a max norm constraint to the weight matrix of the final linear layer instead of using $l2$-regularization over all the parameters. In Kim (2014), a row in the weight matrix $\mathbf{W}_l$ is renormalized to the max-norm s if the 2-norm of the row exceeds s after a parameter update. However, in a recent reimplementation of Kim (2014)[1], the renormalization is always applied to the rows of W_l regardless of whether the 2-norm exceeds s or not. This change shows up as a 1% difference in accuracy on the development datasets. Therefore, we use this renormalization strategy instead of max-norm in the original paper

[1] https://github.com/harvardnlp/sent-conv-torch

and refer to it as max-renorm in this work.

5 Ensemble methods

In order to reduce variance of performance when training on different splits of data, models trained with different training datasets and models with different architecture are combined together. Previous research has shown that ensembling models improves performance (Sakaguchi et al., 2015b; Ju et al., 2017; He et al., 2017). We train different models with different splits of training and develop data, and ensemble them together. We use two methods to combine the submodels together: majority voting and stacking. The individual CNNs, or the submodels, first are ensembled together according to their input features into two ensembled models, and then the two ensembles are stacked together to form the final stacked model.

Majority voting

The majority voting strategy is adopted by the submodels to reduce variance and also to provide better generalizability. Each submodel gives one vote to the best class given some input according to their parameters, and whichever class has the most votes wins. Let $\hat{\mathbf{y}}_d$ be the output of d-th submodel in the ensemble, and the final output of the ensemble $\hat{\mathbf{y}}_e$ is

$$\hat{\mathbf{y}}_e = \sum_d \text{hardmax}(\hat{\mathbf{y}}_d) \tag{3}$$

where hardmax is the function that converts the argument of the function into a one-hot vector where the original maximum value of the argument is replaced by 1 and the rest by 0. In the case of ties, we pick the class that appears first in the vector. For the ensemble, the predicted class is $argmax(\hat{\mathbf{y}}_e)$. However, $\hat{\mathbf{y}}_e$ is also an unnormalized distribution used by the stacked model.

Stacking

We use stacking (Wolpert, 1992) to combine results from the ensembles. Stacking is essentially weighted linear interpolation of the ensemble results. Let $\hat{\mathbf{y}}_{e,r}$ be the output of the r-th ensemble, thus the final output of stacking $\hat{\mathbf{y}}_t$:

$$\hat{\mathbf{y}}_t = \text{softmax}(\sum_r \alpha_r \hat{\mathbf{y}}_{e,r}) \tag{4}$$

where α_r is the coefficient of the r-th ensemble. The coefficients need to be trained.

6 Model setup and training

We now explain preprocessing steps, the hyperparameters we used for training, model initialization as well as the training process.

6.1 Preprocessing

We represent a sentence with both a sequence of words and a sequence of characters. Using word sequences as input allows us to take advantage of pre-trained word embeddings so that even if a word never appears in the training set due to data sparsity, its embedding may still provide enough information for the models to classify correctly. Using character sequences allows the models to be robust to spelling variations. This helps the word-based models, which are susceptible to misspelled words. Therefore we train separate word- and character-based CNN models. We then ensemble the word CNN submodels into a word CNN ensemble, and also ensemble the character CNN submodels into a character CNN ensemble, using majority voting in both cases. The two ensembles are then combined together with stacking to form the stacked CNN model. All submodels are trained separately and remain fixed when the stacked model is being trained.

6.2 Hyperparameters

The hyperparameters for both the word CNN and the character CNN submodels are mostly the same. In the following paragraph, if not otherwise mentioned, all hyperparameters are shared. All hyperparameters are tuned on the development dataset of each fold. We set the number of submodels d to 5. We set the number of kernels of the character CNN to be 400, and the word CNN 300. We use kernels of widths 2 to 5 for the character CNN, and 3 to 5 for the word CNN. All nonlinearities in the models are rectified linear units (Nair and Hinton, 2010). We use Adadelta (Zeiler, 2012) as the optimizer for the submodels, and use the recommended values for its hyperparameters ($\rho = 0.9, \epsilon = 1 \times 10^{-6}, learning\_rate = 1.0$). We set the max-renorm to be 3.0 and the dropout rate for the linear layer to be 0.5. We use negative log-likelihood as our training objective to minimize.

6.3 Initialization

For the word CNNs, we follow Kim (2014) to initialize the parameters. We use pre-trained word2vec word embeddings (Mikolov et al., 2013) for words that are in the whole dataset, and initialize embeddings of the other out-of-vocabulary words with $Unif(-0.25, 0.25)$. This keeps the variance of each randomly initialized embedding close to the word2vec embeddings. We also tried the GloVe embeddings Pennington et al. (2014) and found it to be slight worse in performance than word2vec embeddings. We initialize the convolutional kernels with $Unif(-0.01, 0.01)$ and the linear layer $N(0, 1e-4)$. We initialize all bias terms to 0.

For the character CNNs, we initialize all weights to follow $Unif(-1/\sqrt{n_{in}}, 1/\sqrt{n_{in}})$ (Glorot and Bengio, 2010) where n_{in} is the length of the input vector. For the convolutional kernels, the length of the input vector is hk. Additionally, we randomly initialize the embedding matrix with $N(0, 1)$.

6.4 Training

We use 10-fold cross validation for training and evaluation. Shuffling the original dataset reduces performance variance on the development sets, improving generalizability. For each fold, we split the whole dataset into training and testing sets with 90/10 ratio, and further split the training set into training and development sets with 90/10 ratio. For training the submodels, we split the training set into training and development sets at different places to create different training data for each submodel, and add all the labels as training instances to the training set. We use minibatch updates with batch size 50 and train each submodel for 40 epochs, shuffling the training set for each epoch. We evaluate the performance of the submodels after each epoch of training, using early stopping on development data to select the best-performing set of parameters.

Majority voting does not need training, but stacking does. We train the stacked model also for 40 epochs with the training/development split that is done for the first submodel. The optimizer is also Adadelta with recommended hyperparameter values.

	Simple	Ensembled
ChatScript	**79.8**	n/a
Baseline	77.2	n/a
CharCNN	76.16	78.20
WordCNN	76.92	77.67
Stacked	n/a	**79.02**

Table 1: Mean 10-fold Accuracy by System Type. Numbers reported are on the test set.

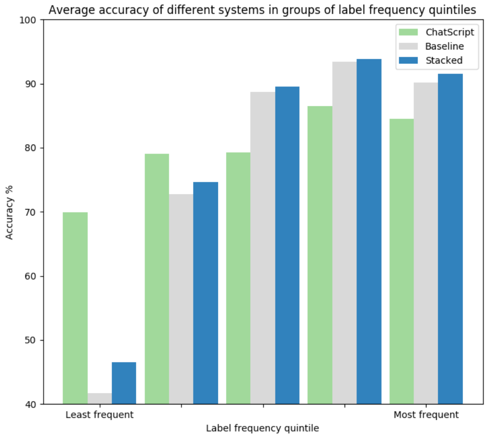


Figure 3: System Accuracy by Label Frequency, in Quintiles. Note the high performance in the least frequent labels for ChatScript, the hand-crafted pattern matching system. With more data, the CNNs perform better.

6.5 Baseline

We also create a simple baseline system for comparison. The baseline system is a logistic regression classifier that takes in one-hot representations of $1, 2, 3$-grams of words, word stems, and $1, 2, 3, 4, 5, 6$-grams of characters from a sentence as features and predict what class this sentence belongs to. The baseline system also follows the 10-fold cross validation training setup.

7 CNN Results

System performance is measured by correct question identification for each of the 4330 user turns. Accuracies reported are the average 10-fold cross-validation accuracies. Apart from performance results from the baseline logistic classifier and the stacked model, we also include results from the rule-based ChatScript system.

Table 1 shows the test accuracies of different systems averaged over the 10 folds. For machine learning systems, the stacked CNN model performs the best overall. For single models, the baseline system works the best. It is widely believed that deep learning models are generally data-hungry, and the training sets are small compared to popular training sets for deep learning models. In terms of single model performance, the simple logistic regression is better than the deep learning models and it is reasonable to believe that data sparsity is at issue. However, through ensembling and stacking, the final stacked model performs the best, and the performance gap between a machine learning system and a carefully created and actively maintained rule-based system on this task becomes very small. The 2-point difference between the baseline system and the stacked CNN model is highly significant ($p = 8.19 \times 10^{-6}$, McNemar's test).

System accuracies by label frequency show a striking difference between ChatScript and all other systems for the most infrequent labels. Fig. 3 shows a clear advantage for ChatScript in the quintile with the least frequent labels. ChatScript is not trained, so data sparsity does not affect performance of this system as much as the machine learning systems. Also, most of the time, the training instances for a rare label are very close to the label itself. Therefore by pattern matching, ChatScript performs best among all models for items with rare labels. The stacked CNN model performs slightly better than the baseline model in this quintile, but still is very low compared to ChatScript.

However, ChatScript does relatively worse to the other systems as label frequency increases. This is expected because when training instances increase in a dataset, it means that probabilistically variants of the label that differ substantially will also increase. Therefore more non-conflicting patterns need to be added and existing patterns need to be updated, which may be difficult or even impossible to do. Machine learning based systems are good at frequent labels. There are more training instances, and constraint of non-conflicting rules does not apply to such models. We can see in Figure 3 that the stacked CNN model outperforms the baseline in all quintiles, and outperforms ChatScript in the last three quintiles where training data is ample. The clear difference of model behavior of ChatScript and the stacked CNN shows that they may be combined together and perform even better in all quintiles.

The effectiveness of ensembling

Figure 4 shows accuracy numbers on the test set of each fold for the best individual submodels and ensembles. The best individual submodels are chosen based on performance on the development set. For the character CNN submodels and the ensemble, it is clear that the ensembled model always performs better than the best individual model. For all 10 folds, the ensembled character CNN model always outperforms the best model in the ensemble and the average performance gain is about 1.04%. For the word CNN submodels and the ensemble, the relation is less obvious. Although in 9 out of 10 folds, the ensemble outperforms the best individual model, the difference between performances of the two systems is smaller compared to the difference between best character CNN submodel and the character CNN ensemble, and the average performance gain is about 0.75%. A Student's t test on the accuracy numbers confirms this observation. The improvement gained from ensembling the character CNN submodels is significant ($p = 0.0045$), but the improvement gained from ensembling the word submodels is not ($p = 0.43$).

The result for the ensembles and the stacked model can also be seen in Figure 4. Except for fold 1, where the stacked model is outperformed by the character CNN ensemble, the stacked model outperforms all the ensembles in all the other folds. The performance gained from stacking the character CNN ensemble on top of the word CNN ensemble is significant ($p = 0.049$), but insignificant the other way around ($p = 0.11$). This could mean that the character CNN ensemble has all the information it can extract from text for prediction that the word CNN ensemble is not providing new information for it to do better.

Comparing the stacked model with the individual best model, stacking always provides significant performance gain ($p = 0.033$ for the best word CNN submodel and $p = 9 \times 10^{-4}$ for the best character CNN submodel).

Error analysis

One of the hypotheses of why the stacked CNN model works better is because it has access to word embeddings, and word embeddings are good at modeling words that are superficially different but synonymous. Table 2 shows a few examples where the baseline classifier makes the wrong pre-

No.	Question	Baseline predicted label	Stacked CNN predicted label
1	constipation	do you use any contraception	do you have any bowel problems
2	does anything aggravate your back pain	what makes the pain better	what makes the pain worse
3	are you employed	are you happy	what do you do for a living
4	have you taken any tylenol or done anything to help your back pain this time	what makes the pain better	are you taking any medication for the pain
5	have you ever had any psychotherapy treatment	have you tried any treatment	do you have a history of depression
6	have you injured your back previously	have you had back injury	when was your last period
7	can you stand up	are you able to stand	what do you do for a living

Table 2: Prediction examples. The stacked CNN model predicts the correct label for the first 4 cases and the wrong label for the last 3 cases. The baseline predicts the last 3 cases correctly.

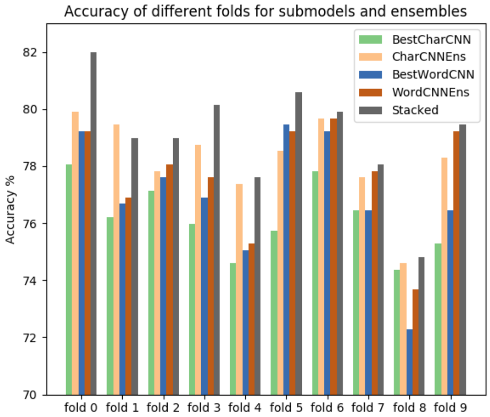


Figure 4: System Accuracy of the Submodels and Ensembles by Fold

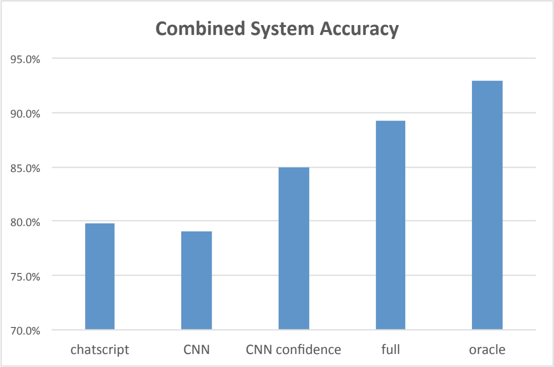


Figure 5: Accuracy of model trained to combine ChatScript and CNN predictions

diction but the stacked CNN model makes the correct prediction. These examples show how the stacked CNN model is able to use semantic information provided by the word embeddings to make the correct prediction whereas the baseline classifier can not. Example 1 requires the models to know 'constipation' is related to 'bowel'. The baseline classifier is confused by the spelling similarities between 'constipation' and 'contraception', but the stacked model is able to make the right prediction. Example 2 requires the models to know that 'aggravate' means 'get worse', not 'get better', and the stacked model makes the correct decision. Similarly, 'employed' and 'tylenol' in examples 3 and 4 all show that the stacked CNN can tap into the semantic information provided by the word embeddings and use them in prediction.

However, it appears that the semantic information in word vectors can sometimes backfire as well. In examples 5 and 6, the words that are similar in meaning are making unhelpful connections. The stacked CNN links 'psychotherapy treatment' to 'depression' in example 5, but the question as a whole is about treatment, so the prediction from the stacked model is wrong. Similarly, the stacked model predicts example 6 to be in the class 'when was your last period' maybe because 'last' and 'previously' are similar, apparently missing the 'injury' part of the question. The stacked CNN missed example 7 because of data sparsity. There are only two instances of the class 'are you able to stand' in the whole dataset, therefore the stacked CNN has low confidence in the gold class and instead chooses a class which has much more training examples.

8 Combining ChatScript with the CNN

While the fully ensembled and stacked CNN performs at 79% accuracy, which is slightly below that of ChatScript, its error pattern is distinct from ChatScript, as seen in Fig. 3. ChatScript, because it only uses pattern matching to do classification, is less affected by the imbalance of training instances belonging to target classes in the training data. The CNN, however, is affected by such imbalance and generally performs worse when training instances for one class is scarce. Meanwhile, despite the use of automatic spelling correction in ChatScript, a substantial portion (11.1%) of the

Chatscript errors were on questions with typos or other spelling errors in them; on these items, the CNN managed to make the correct prediction 74.1% of the time. This indicates that the character CNNs are more robust to spelling errors, as there is no need to make a possibly erroneous guess as to the correctly spelled word. Additionally, whereas the CNN always makes its best guess on test items, the ChatScript patterns failed to match (yielding no answer) on 7.4% of the questions, representing 36.5% of the ChatScript errors. On these questions, the CNN achieved 59.6% accuracy, indicating that they are considerably more difficult to recognize than the average question.

Given that our two methods make rather different errors, we investigated whether it would make sense to combine them, and found that an oracle that always chose the correct system if either was right could achieve 92.9% accuracy, much higher than the ChatScript systems 79.8% accuracy by itself. As such, we experimented with training a logistic regression binary classifier for automatically choosing between the two systems, again using 10-fold cross-validation. The binary classifier was trained to choose the CNN prediction when it was correct and ChatScript was wrong, otherwise to choose the ChatScript output—including in cases where the ChatScript patterns yielded no match, on the assumption that a no-match response would be preferable to an incorrect response. To make its choices, the binary classifier used the following features:[2]

Log Prob The log probability of the predicted class in the final output of the stacked model.

Entropy The entropy of the distribution over classes output by the stacked model.

Confidence For each submodel, the confidence score is the unnormalized score for the predicted class. For the stacked model, the confidence score is the average of all confidence scores from the submodels.

CNN Label The label predicted by the CNN.

CS Label The label matched by ChatScript.

CS Log Prob The log probability according to the CNN of the ChatScript prediction.

Note that an automatic method for choosing between the two systems could in principle do worse than simply always choosing the ChatScript system. Nevertheless, as shown in Fig. 5, a classifier trained to make the choice based on the log prob, entropy and confidence of the CNN's prediction achieves 85.0%, a large gain. This binary classifier can be improved further by taking into account how likely the logistic regression model considers the ChatScript choice, including the special case of no match from the ChatScript system, along with the specific label of both system predictions. The full model, making use of these additional features, achieves 89.3% accuracy, a huge gain that represents more than two-thirds of the potential gains revealed by the oracle analysis, and a **47% reduction in error** over the ChatScript system by itself. This shows that the stacked CNN effectively compliments the rule-based ChatScript system on the mid-to-high frequency labels, making the final system much stronger than either of the component systems.

We also investigated whether it would make sense to always choose the CNN prediction when the ChatScript system yielded no match. By allowing the binary combination classifier to choose the ChatScript system even when it yielded no match, the combined system reduced the number of no match outputs from 7.4% to 2.4%, close to the 3.0% oracle rate for cases where neither system is correct. Always choosing the CNN prediction in these cases increases the combined system accuracy by 0.6%, but at the cost of a 1.8% increase in erroneous responses rather than no-match responses. As such, it appears preferable to allow the binary combination classifier to make the choice even in the no-match cases.

9 Discussion and Future Work

CNNs are very sensitive to their hyperparameters and initializations. Differences in normal vs. uniform weight matrix initializations were observed to impact word- and character-based CNN models differently. He et al. (2017) use orthonormal initializations following Saxe et al. (2013), while Kim (2014) suggests initializing unknown word embeddings using parameters (e.g., variance) sampled from pre-trained word embeddings, etc.; further exploration of hyperparameter tuning and initialization strategies are left as future work.

Models with more complicated architecture,

[2] Note that ChatScript does not output scores for its matched patterns, so we did not pursue a stacking-based approach.

such as Memory Networks (Weston et al., 2015), Highway Networks (Srivastava et al., 2015) and Convolutional sequence models (Gehring et al., 2017) can also be explored and integrated as well, although more data is needed to successfully train these models. Other ensemble methods like Super Learner (Ju et al., 2017) should be tried as well.

Since label sparsity is at the heart of the performance difference between ChatScript and the CNN models, a more direct way to deal with lack of training examples (possibly obviating the need for a hand-crafted system like ChatScript) could be to automatically generate paraphrases to augment available data, potentially with a content author in the loop; we are currently exploring strategies for doing so.

10 Conclusion

This work shows the value of combining a hand-authored pattern matching system with CNN models to overcome label sparsity in training. The stacked CNN model with ensembled word and character CNN submodels significantly outperforms the logistic regression baseline. Within the CNN models, ensembling is found to significantly improve performance for the character model, while stacking always provides significant improvement over the best word- or character-based submodels. The final system uses a binary classifier over ChatScript and a stacked CNN, improving overall accuracy by 10% and achieving an impressive 47% error reduction on a question identification task in a virtual patient dialogue system.

Acknowledgments

Thanks to Kellen Maicher for creating the virtual environment, Bruce Wilcox for authoring ChatScript, and Eric Fosler-Lussier and William Schuler for feedback and discussion. This project was supported by funding from the Department of Health and Human Services Health Resources and Services Administration (HRSA D56HP020687), the National Board of Medical Examiners Edward J. Stemmler Education Research Fund (NBME 1112-064), and the National Science Foundation (NSF IIS 1618336). This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under grant no. DGE-1343012. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The project does not necessarily reflect NBME policy, and NBME support provides no official endorsement.

References

Luciano Barbosa, Dasha Bogdanova, Bianca Zadrozny, and Rio De Janeiro. 2016. Learning Hybrid Representations to Retrieve Semantically Equivalent Questions. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*. 1, pages 694–699.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research* 12:2493–2537.

Douglas Danforth, A. Price, K. Maicher, D. Post, B. Liston, D. Clinchot, C. Ledford, D. Way, and H. Cronau. 2013. Can virtual standardized patients be used to assess communication skills in medical students. In *Proceedings of the 17th Annual IAMSE Meeting, St. Andrews, Scotland*.

Douglas Danforth, Mike Procter, Richard Chen, Mary Johnson, and Robert Heller. 2009. Development of virtual patient simulations for medical education. *Journal For Virtual Worlds Research* 2(2).

Arpita Das, Harish Yenala, Manoj Chinnakotla, and Manish Shrivastava. 2016. Together We Stand : Siamese Networks for Similar Question Retrieval. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*. pages 378–387.

David DeVault, Anton Leuski, and Kenji Sagae. 2011. An evaluation of alternative strategies for implementing dialogue policies using statistical classification and rules. In *Proceedings of the 5th International Joint Conference on Natural Language Processing (IJCNLP)*. pages 1341–1345.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional Sequence to Sequence Learning. In *Proceedings of 34th International Conference on Machine Learning (ICML 2017)*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*. volume 9, pages 249–256.

Hua He, Kevin Gimpel, and Jimmy Lin. 2015. Multi-Perspective Sentence Similarity Modeling with Convolutional Neural Networks. *Proceedings of the*

2015 Conference on Empirical Methods in Natural Language Processing (EMNLP 2015) (1):1576–1586.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. Deep Semantic Role Labeling : What Works and What's Next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL 2017)*.

Evan Jaffe, Michael White, William Schuler, Eric Fosler-Lussier, Alex Rosenfeld, and Douglas Danforth. 2015. Interpreting questions with a log-linear ranking model in a virtual patient dialogue system. In *Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications*. pages 86–96.

Cheng Ju, Aurélien Bibaut, and Mark J. van der Laan. 2017. The Relative Performance of Ensemble Methods with Deep Convolutional Neural Networks for Image Classification. *arXiv* page 1704.01664vi [stat.ML].

Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)* pages 1746–1751.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-Aware Neural Language Models. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 16)*. pages 2741–2749.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., pages 1097–1105.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*. pages 3111–3119.

Vinod Nair and Geoffrey E Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*. 3, pages 807–814.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*. pages 1532–1543.

Vikram Ramanarayanan, David Suendermann-Oeft, Hillary Molloy, Eugene Tsuprun, Patrick Lange, and Keelan Evanini. 2017. Crowdsourcing multimodal dialog interactions: Lessons learned from the HALEF case. In *Proceedings of the AAAI-17 Workshop on Crowdsourcing, Deep Learning, and Artificial Intelligence Agents*. pages 423–431.

Deepak Ravichandran, Eduard Hovy, and Franz Josef Och. 2003. Statistical qa-classifier vs. re-ranker: What's the difference? In *Proceedings of the ACL 2003 workshop on Multilingual summarization and question answering-Volume 12*. Association for Computational Linguistics, pages 69–75.

Brent Rossen and Benjamin Lok. 2012. A crowdsourcing method to develop virtual human conversational agents. *International Journal of HCS* 70(4):301–319.

Keisuke Sakaguchi, Michael Heilman, and Nitin Madnani. 2015a. Effective feature integration for automated short answer scoring. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, pages 1049–1054.

Keisuke Sakaguchi, Michael Heilman, and Nitin Madnani. 2015b. Effective Feature Integration for Automated Short Answer Scoring. In *Proceedings of the Conference of the North American Chapter of the Association of Computational Linguistics (NAACL 2015)*. pages 1049–1054.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *Advances in Neural Information Processing Systems* pages 1–9.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15:1929–1958.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway Networks. In *Proceedings of the Deep Learning workshop (ICML 2015)*.

D. Suendermann, K. Evanini, J. Liscombe, P. Hunter, K. Dayanidhi, and R. Pieraccini. 2009. From rule-based to statistical grammars: Continuous improvement of large-scale spoken dialog systems. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 2009*. pages 4713–4716.

Oriol Vinyals, Charles Blundell, Timothy Lillcrap, Koray Kavukcuoglu, and Daan Wierstra. 2016. Matching Networks for One Shot Learning Oriol. In *Proceedings of Neural Information Processing Systems (NIPS 2016)*. pages 817–825.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory Networks. In *ICLR*. pages 1–15.

D.H. Wolpert. 1992. Stacked generalization. *Neural Networks* 5(2):241–259.

Wenpeng Yin and Hinrich Schütze. 2015. Convolutional Neural Network for Paraphrase Identification. *Proceedings of 2015 Conference of the North American Chapter of the Association for Computational Linguistics Human Language Technologies (NAACL HLT 2015)* pages 901–911.

Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. 2016. ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs. *Transactions of the Association for Computational Linguistics* 4:259–272.

Matthew D. Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *CoRR* .