Towards Interpretable Multi-Task Learning in Language

Samridhi Shree Choudhary, Shruti Rijhwani
{sschoudh, srijhwan}@andrew.cmu.edu

1 Introduction

The traditional approaches to multitask learning for language specific tasks like parsing, chunking and part-of-speech tagging, focus on using a single, generic neural network architecture that is trained for each of these tasks separately from the labeled data available for the task. For example, Collobert and Weston (2008) develop a single convolutional neural network architecture that is trained for several NLP tasks, by extracting features that are used for joint training [3]. In this work, we would like to explore the idea of learning representations for multitask systems in a method that attempts to explicitly encode features for each task individually, but also allows dependencies between these task-specific encodings. More specifically, we would like to answer the following research questions:

Research Question 1: For a given neural network, can we find the most important neurons that carry the maximum information for making predictions for a specific task and separate the neurons that contain context that is not directly related to this task?

Research Question 2: Can we interpret and use the features represented by these neurons to make predictions for other related tasks?

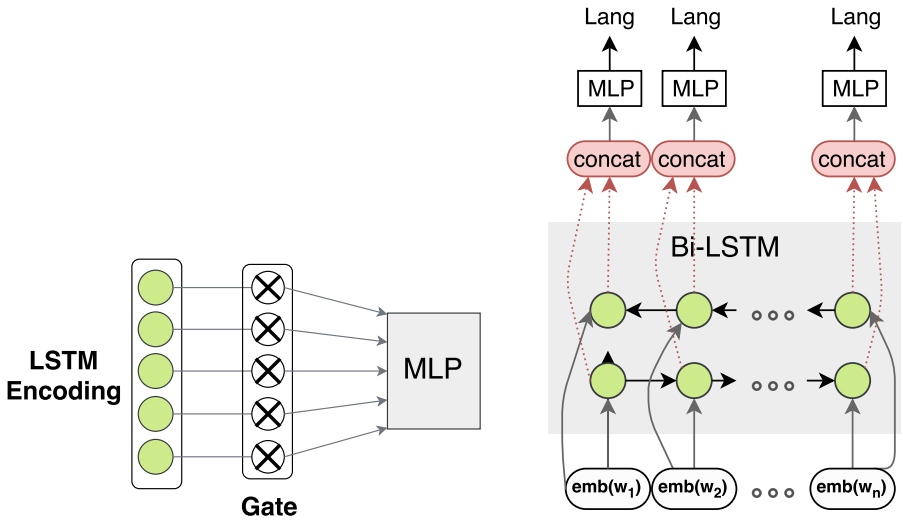
2 Literature Survey

Radford et al. (2017) developed an unsupervised system trained to predict the next character in the text of Amazon reviews [7]. Surprisingly, they discovered that their model learned a very good representation of sentiment in this process. Specifically, a single neuron contained all the information required to predict sentiment with high accuracy.

We could argue that this behaviour is not specific to their particular model and can be extended to other tasks. That is, the representations learned by individual neurons could be exploited for better performing systems. Many modern machine learning systems use representation learning to map raw data to useful features [1]. The task specific representations learned by training supervised high capacity models have been analyzed to reveal fascinating properties [9].

In their recent work, Lin et al. (2017) propose *Deep Neural Inspection* as a way to search for interpretable neurons in RNN models [5]. The domain knowledge learned by RNNs is designed as a library of features. Deep Neural Inspection is then defined as a search problem where each feature is a query and the task is to identify the neurons that are predictive of the feature. We would like to follow on this line of work and explore the possibility of learning a subset of neurons that are most efficient in making the predictions. We aim to do this by *gating* the network's neurons. The gating mechanism selects a subset of the neurons to make predictions for the task that the network is being trained on. This forces the other neurons (*non-gated*) to learn different abstractions from the data. This would throw light on the knowledge accumulated by a network's neurons. This paradigm can easily be extended to a multitask system where different neurons learn features for different tasks (some neurons may not be useful at all or may abstract out useless information). Figure 1a visualizes a single LSTM layer in the architecture of our proposed model, with the example task of

10707 Class Project (2017).



(a) Neuron gating for the output of word-LSTM (b) Model Architecture

Figure 1

part-of-speech (POS) tagging. The red gates and neurons refer to those which are not activated for the task at that state, and only the green neurons are used for tagging.

3 Methods

We use two bi-directional LSTMs for the classification task: at the level of words (**char-LSTM**) and at the level of sentences (**word-LSTM**). Figure 1b shows the architecture of the model used. For a given input sentence, the model first runs a character-level bidirectional LSTM on each word w_i. The final state vector from both the directions are concatenated to produce a word embedding for w_i, called $emb(w_i)$. Next, the model runs a bidirectional LSTM on these word embeddings. The hidden state vectors from both directions are concatenated for each word w_i, to produce an encoding $\boldsymbol{h}_i^{w_i}$, at each word of the sentence. This encoding is the input to a feed-forward network, which classifies the particular word.

We experiment with three different methods of filtering the neurons in word-LSTM encoding before they are used in the feed-forward network. This leads to three flavors of experiments with our model as follows:

Baseline Model: The full encoding i.e., all the neurons in encoding, is passed as an input to the feed-forward network for classification.

Fixed Model: A fixed subset of neurons from the encoding are concatenated and fed as input to the feed-forward network. For multi-task classification, a subset is defined for each task. This is done by a binary gating mechanism, which disables some of the neurons for each task. The sets of enabled neurons are mutually exclusive. The idea here is to allow the neurons used for one task prediction and the neurons used for the other prediction to interact and influence each other within the LSTM across time-steps, but not at the final prediction step.

Gated Model: In the fixed model, we use a manually assigned binary gate which does a mutually exclusive divide of the neurons between the two tasks, but still allows the neurons to influence each other within the LSTM time-steps. In the gated model, we add another dimension to the interaction between the two tasks. The neuron interaction with the LSTM remains, but we also allow each neuron in the encoding to participate in both POS and NER tasks in a weighted fashion. That is, a single neuron can participate in both prediction tasks, unlike the fixed model and the baseline model.

The neurons of the encoding are weighted differently for each task (gating). These gating weights are learned during the training, therefore the neurons may not have the full strength used for each

task. The gates are used with a sigmoid non-linearity, thereby weighting the neurons between 0 and 1. The hope is that the weights are higher for the more predictive neurons of a particular task than the non-predictive neurons.

The following equations explain models more concretely.

For the t^{th} character c_t of w_i:

$$\overrightarrow{\boldsymbol{h}}^{(f)}_{c_t} = \text{LSTM}(c_t, \overrightarrow{\boldsymbol{h}}^{(f)}_{c_{t-1}})$$
$$\overleftarrow{\boldsymbol{h}}^{(b)}_{c_t} = \text{LSTM}(c_t, \overleftarrow{\boldsymbol{h}}^{(b)}_{c_{t+1}})$$
$$\boldsymbol{e}mbed(w_i) = \text{concat}[\overrightarrow{\boldsymbol{h}}^{(f)}_{c_N}; \overleftarrow{\boldsymbol{h}}^{(b)}_{c_0}]$$

For the word w_i, with the representation $embed(w_i)$:

$$\overrightarrow{\boldsymbol{h}}^{(f)}_{i} = \text{LSTM}(embed(w_i), \overrightarrow{\boldsymbol{h}}^{(f)}_{i-1})$$
$$\overleftarrow{\boldsymbol{h}}^{(b)}_{i} = \text{LSTM}(embed(w_i), \overleftarrow{\boldsymbol{h}}^{(b)}_{i+1})$$
$$\boldsymbol{h}^{w_i}_i = \text{concat}[\overrightarrow{\boldsymbol{h}}^{(f)}_{i}; \overleftarrow{\boldsymbol{h}}^{(b)}_{i}]$$

Baseline Model (Single Task):

$$\boldsymbol{o_i} = \texttt{MLP}(\boldsymbol{h}^{w_i}_i)$$
$$\boldsymbol{p}_i = \texttt{softmax}(\boldsymbol{o_i})$$

Gated Model (Multi Task):

$$\boldsymbol{g_a} = \sigma(W_a h^{w_{i-1}}_i + U_a x_t + b_a)$$
$$\boldsymbol{g_b} = \sigma(W_b h^{w_{i-1}}_i + U_b x_t + b_b)$$
$$o_a = \texttt{MLP}(h^{w_i}_i \odot g_a)$$
$$o_b = \texttt{MLP}(h^{w_i}_i \odot g_b)$$
$$\boldsymbol{p}_a = \texttt{softmax}(\boldsymbol{o_a})$$
$$\boldsymbol{p}_b = \texttt{softmax}(\boldsymbol{o_b})$$

Fixed Model (Multi Task):

$$\boldsymbol{g_a} = [11111 \cdots 00000 \cdots 11111]$$
$$\boldsymbol{g_b} = \mathbf{1} - g_a$$
$$o_a = \texttt{MLP}(h^{w_i}_i \odot g_a)$$
$$o_b = \texttt{MLP}(h^{w_i}_i \odot g_b)$$
$$\boldsymbol{p}_a = \texttt{softmax}(\boldsymbol{o_a})$$
$$\boldsymbol{p}_b = \texttt{softmax}(\boldsymbol{o_b})$$

The gates in the fixed model, $\boldsymbol{g_a}$ and $\boldsymbol{g_b}$ are binary gates, which we fix (they are not learnt). The 1 represents using the neuron fully and 0 represents not using the neuron at all. The neurons used for o_a are not used at all in o_b. This controls the learning such that a specific set of neurons is used for prediction task a and the remainder of the neurons are used for prediction task b. Notably, the neurons from one task can influence the other task in future time-steps, since we are not controlling the LSTM context learning.

no/es me/es lebante/es ahorita/es cuz/en I/en felt/en como/es si/es me/es kemara/es por/es dentro/es !/other :o/other

Figure 2: Language Identification Example. The languages are *es* (Spanish), *en* (English), *Other*

Salomon/NNP/B-ORG Brothers/NNPS/I-ORG in/IN/O California/NNP/B-GPE

Figure 3: POS and NER Example. The POS tag for each word is in blue and the NER tag in red

.

4 Dataset

- **Language Identification**: The English-Spanish dataset from Solorio et al. (2014) [8] contains 13,000 word-level language annotated tweets, both monolingual and mixed language. We split the data in the ratio 80-10-10 for train, validation and test sets respectively. An example annotation from the dataset is shown in Figure 2.
- **Part-of-Speech Tagging**: The Penn Treebank dataset is a large corpus of Wall Street Journal articles annotated with a set of 36 part-of-speech tags [6].
- **Named Entity Recognition**: For consistency with our POS tagging data, we use the named entity tags for the Penn Treebank corpus as annotated in the OntoNotes dataset [4]. We restrict our tag set to the most common named entities – geopolitical entity (GPE), location (LOC), facility (FAC), organization (ORG), person (PERSON) and non-entity/other (O).

In order to ensure our sentences have both POS and NER tags, we take an intersection of the data present in the original Penn Treebank [6] and the OntoNotes database [4]. The total number of annotated sentences is 27891. We use the data splits described in described in Collins (2002) [2] for train (sections 0-18), validation (sections 19-21) and test (sections 22-24) sets. Figure 3 shows an example from our dataset with POS and NER tags for each word in the sentence.

5 Experiments

We performed two categories of experiments - **Single Task** *(proof-of-concept)* and **Multi Task**.

5.1 Single Task - Proof of Concept

In order to have a proof-of-concept for our hypothesis (*Is the most important information for prediction contained in a subset of neurons in the network?*), we trained a model to perform single-task classification - predict the language of the words in a multilingual (code-switched) corpus. This was extended further to use only a few neurons for the language prediction.

The fundamental architecture of all the models is the same (section 3) with differences in hyper-parameter values and the the way predictions are made. The common hyper-parameter values for all the models are: *Character Embedding Size* (256), *Char-LSTM Hidden Layer Size* (256) and the *MLP Layer Size* (256). The output layer size is 3, since we are predicting among 3 tags.

Full: This refers to the model where we feed all the neurons of the word-LSTM output for each word to the MLP in order to predict its language. This represents the regular method of using a Bi-LSTM network and provides a baseline for comparing the results of the prediction when using only a subset of neurons. The word-LSTM hidden size used for this model was 512.

Fixed: This refers to the model where we feed only fixed set of neurons of the word-LSTM output for each word to the MLP in order to predict its language. We use this model to test our hypothesis. The word-LSTM hidden size used for this model was 512, out of which we used two fixed neurons (one from the forward and one from the backward) for prediction.

Small: In this model, we restrict the hidden dimension of the word-LSTM to be the same as the number of fixed neurons used for prediction in the *Fixed* model. Therefore, the word-LSTM hidden size used for this model was 2. The purpose behind estimating this model was to validate that the task at hand could not be learned trivially by using just the same number of neurons as the prediction

Model	Validation Accuracy	Test Accuracy	Validation Loss
Full	95.5%	94.5%	9096.23
Fixed	94.8%	94.04%	9236.81
Small	94.2%	93.9%	9312.32

Table 1: Single Task - Validation and Test Accuracy Values

neurons in the *Fixed* model with the view that the abstract information learned by other neurons in the *Fixed* model is also useful for the prediction task.

5.2 Multi-Task

In this series of experiments, we attempt to perform multi-task learning by jointly training models for two tasks – part-of-speech (POS) tagging and named entity recognition (NER). We designed three experiments for this task as mentioned in section 3 – baseline, fixed and gated. The common hyperparameters for these models are: *Character Embedding Size* (n), *Char-LSTM Hidden Layer Size* (n) and the *MLP Layer Size* ($2n$). The primary difference between these models is in the word-LSTM hidden layer and the gating mechanism applied between the word-LSTM and the MLP (Figure 1a).

Baseline: We train separate models for both POS and NER tagging, therefore the POS and the NER neurons do not interact with each other at all. This model serves to be the comparison we use for the fixed and gated versions of our experiments, in order to check the effects that adding interaction between the tasks and using a gating mechanism has on the performance of the system. We use the *Word-LSTM Hidden Layer Size* as n.

Fixed: The fixed model is a model that is jointly trained for both our prediction tasks. Each task has its own MLP for prediction, as described by the equations in Section 3. We use the *Word-LSTM Hidden Layer Size* as $2n$. However, during prediction, we equally divide the neurons between the two prediction tasks. Therefore, each MLP is only given n neurons.

Gated: We use the *Word-LSTM Hidden Layer Size* as n and use a *Gate* vector of size n for each task to weight the word-LSTM encoding before using it in the MLP. The weight matrices used to compute the gate are different for each task.

6 Results

6.1 Single Task

Table 1 shows the validation and test accuracy values for the three models as described in section 5.1. We use early stopping with the validation set accuracy to choose our best model for reporting the test accuracy. Notably, we see that the test accuracy in the **Full** model (with 512 hidden units) is higher than the other settings. However, the **Fixed** model gives better results than the **Small** model.

These results are in line with our hypothesis described in section 5.1, where we suggest that we can separate information useful for a certain task with a small number of neurons, without losing much in terms of performance. Even though we are using only 2 neurons for prediction in the **Fixed** model, the other neurons in the hidden layer influence the predictions in the future time-steps.

Since the performance on **Small** is lower than that of **Fixed**, the task is not trivially easy to model with a hidden size of 2 and the information from the other neurons (that are not used directly for prediction) help improve accuracy indirectly. It is also worth noting that the total validation loss for the **Small** model is reasonably higher than that of the **Fixed** model. This shows that the predictions made by **Fixed** are closer in terms of target probability of the true values.

6.2 Multi-Task

We experimented with $\boldsymbol{n = 32}$ and $\boldsymbol{n = 64}$, leading to two different sets of parameters as explained in section 5.2. Table 2 shows the validation and test accuracy values for the set of models with hyper-parameters corresponding to $\boldsymbol{n = 32}$ (Model - 32) while table 3 shows the same for the model with $\boldsymbol{n = 64}$ (Model - 64). Similar to the single task experiments, we use early stopping with validation accuracy to select the optimum model for reporting the test accuracy.

Model	Validation Accuracy	Test Accuracy
POS		
Baseline	85.2%	85.8%
Fixed	86.5%	85.9%
Gated	89.4%	89.6%
NER		
Baseline	96.6%	96.7%
Fixed	95.1%	95.2%
Gated	95.3%	95.6%

Table 2: Multi-Task (Model - 32) - Validation and Test Accuracy Values

Model	Validation Accuracy	Test Accuracy
POS		
Baseline	93.3%	93.1%
Fixed	93.6%	93.5%
Gated	94.5%	94.6%
NER		
Baseline	97.8%	97.9%
Fixed	97.2%	97.4%
Gated	97.1%	97.3%

Table 3: Multi-Task (Model - 64) - Validation and Test Accuracy Values

As we can see from the tables 2 and 3, the prediction of the POS tags follow the same performance pattern in terms of the validation and the test accuracy: Gated > Fixed > Baseline. However, NER prediction pattern is different. The fixed and the gated models have almost equal performance in terms of accuracy, however they are worse than the baseline model for both Model-32 and Model-64. The NER accuracy across models is very high (~95 - 98 %). This is primarily because the NER class ***'Other'*** is a staggering majority in our dataset comprising ~90% of all the tags. The baseline performance on NER prediction for both Model-32 and Model-64 are very high are tough to beat as can be seen with a corresponding lower performance by both fixed and the gated flavours of models. Figure 4, shows the plot of the total validation loss vs the number of epochs for 25 epochs for *Model-32*. We can see that the 'Gated' model has the lowest validation loss indicating that this model is closer to the true distribution of the tags. We discuss the implications of these results in the following section.

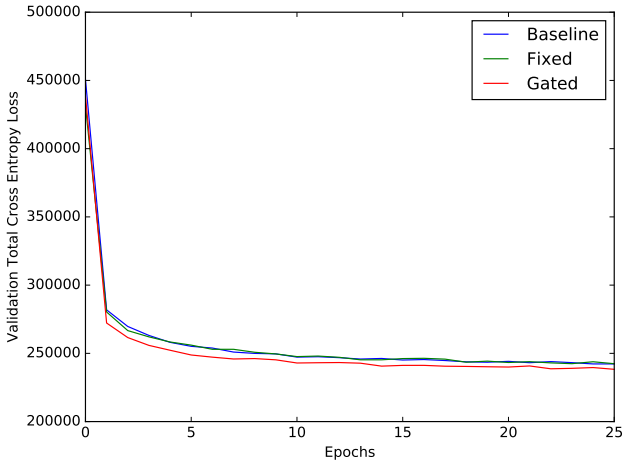
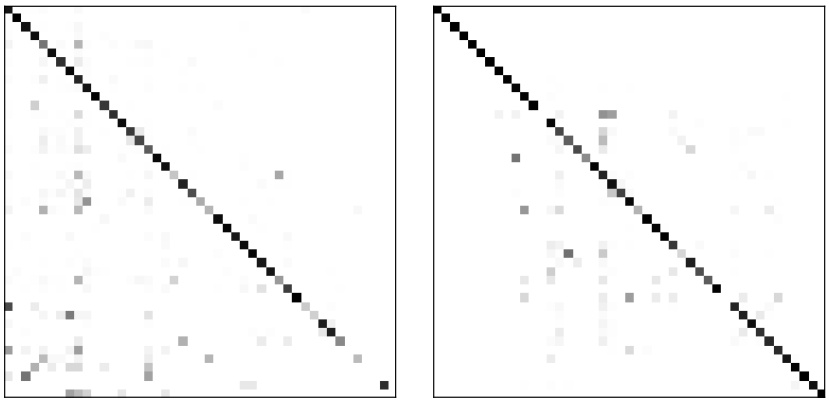


Figure 4: Multi-Task (Model - 32) - Total Validation Loss Vs Epochs for Baseline, Fixed and Gated



(a) Baseline Model (b) Gated Model

Figure 5: Confusion Matrix for Part-of-Speech Prediction. Darker element indicates higher value.

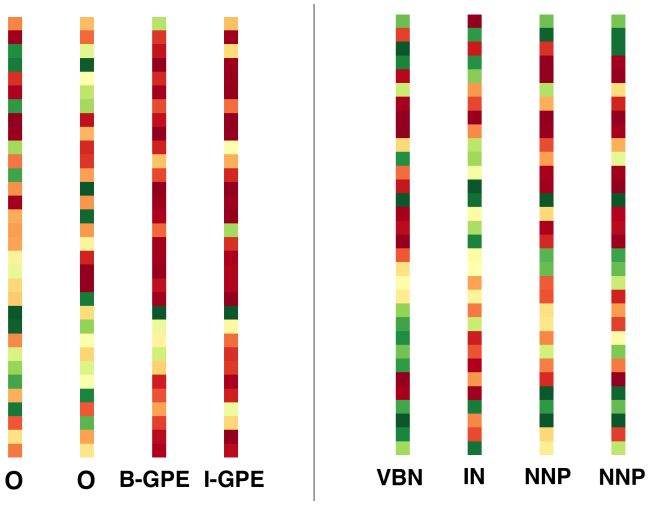


Figure 6: Gate Vectors for phrase – *outlawed in South Africa*. Left is NER, right is POS.

7 Discussion and Analysis

In the Tables 2 and 3, we see that, for POS prediction, having the influence of the NER task gives a boost in accuracy. There is a small boost with the *Fixed* model, but a significant boost in the case of the *Gated* model (4% accuracy increase in Model-32). This is an interesting observation. The fact that the neurons interact only within the LSTM (Fixed) but not at the final prediction does not give as much as a boost as when the model also learns to decide which neurons are needed for prediction (Gated). Note that, in the gated model, we are sharing LSTM parameters between the two tasks, unlike the baselines. The influence of NER neurons for POS tagging is clear from these results. The influence of POS neurons on NER is slightly less clear. As discussed in the previous section, the class distribution of NER is rather skewed, so it is possible that adding POS disrupts this homogeneity.

We now analyze the accuracy, as well as the influence of the neurons on each other. We use Model-32 for these visualizations, chosen randomly, as patterns are similar across both hyper-parameter settings.

Confusion Matrix: In Figures 5a and 5b we see the confusion matrix visualizations for the baseline and gated POS predictions respectively. These do not indicate the actual POS tags for clarity, but serve to clarify the patterns. We see that the diagonal elements are much darker (hence, higher value) for the gated model than the baseline model.

This corroborates our observations on the accuracy, as the diagonal elements represent the *correct predictions* – that the gated model, in general, gets more predictions correct. The predictions for the baseline model are spread out (bottom left corner seems to have a bias!). The diagonal of the baseline matrix has more *white spots* than that of the fixed matrix, indicating that the baseline always makes wrong predictions for these tags.

Gate Visualization: We visualized the gate vectors for POS and NER for a randomly selected phrase from our dataset, *outlawed in South Africa*. These are seen in Figure 6, where the four bars on the right are for NER and four on the left are POS. The colors are a gradual representation from red (for 0.0 value – neuron is completely discarded) to green (for 1.0 value – neuron is completely used).

We see several interesting observations in this visualization. First, the gate vectors for the NER tokens 'GPE' are extremely similar – a large percentage of the neurons are disable (seen with the red color) with a strongly greeen single neuron *in the same position for both words*! This suggests that the model needs just a single neuron to strongly indicate the presence of 'GPE'. This observation is in line with [7], where a single neuron contained all the sentiment information.

We observe that the POS gate vectors for the IN tag has more active (green) neurons than the NNP and VBN tags. We hypothesize that since IN is a very common word seen in several contexts, the model needs more neurons to store all that information. In contrast, the VBN and NNP are comparatively rarer in the corpus and hence need fewer neurons for prediction. The two NNP word gates seem to have similar activations, particularly with regards to the active (green) neurons.

Correlation between POS and NER gates: We study the correlation between the gate vectors of POS and NER to see whether certain neurons are helpful for both tasks (positively correlated) or helpful for one but not the other (negatively correlated). We use the Pearson's co-efficient for this purpose. We calculate the correlation between each set of gate neurons (32 in total) across all the words in our test set.

We notice that 2 neurons are highly negatively correlated (< -0.5 co-efficient), but 7 neurons are highly positively correlated ($> +0.5$ co-efficient). This, to some extent, shows that there is a dependency between the neurons used for POS and NER. The *Fixed* model does not allow information to be shared between the tasks for. The *Gated* model, however, learns how to share this information, as well as to separate information as seen by the negatively correlated gate neurons indicating a reason towards its better performance. Therefore, sharing information is not only beneficial to accuracy, but also allows for fewer parameters to train, since the *Gated* model uses only a single LSTM of size n (baselines have individual LSTMs each of size n). Our experiments show that allowing the model to decide when to share and separate parameters can achieve better scores even with a considerably smaller capacity.

8 Conclusion and Future Work

We started out with the aim to answer two central questions. The proof-of-concept experiment on language identification helped us provide a solid ground to validate our belief that the most important information for prediction of a task in a neural network might be contained in a subset of neurons of the LSTM encoding. We extended our model to include more than one task to predict POS and NER for a sentence.

While the baseline model trained separately on each of these tasks perform well, the jointly-trained model using either a fixed number of neurons or a weighted concatenation of output neurons, performs extremely in terms of the accuracy. This is a strong validation to our first research question indicating that in a jointly trained network for multiple related tasks, we can find the most important neurons for predicting the individual tasks by learning task specific weights on the output encoding. We also performed a correlation analysis to find similarity in the POS and the NER gates, in order to address our second research question, noting the interesting observations in the previous section.

In future, we would like to foray deeper into our second research question and examine the effects of the interaction of the features of the related tasks on their prediction. It would also be interesting to interpret the most predictive features of each task and see if they are similar or dissimilar. Lastly, we believe that such interpretation studies are important for understanding the properties of neural networks, and we would like to extend our analysis to other tasks, including those outside the language processing domain.

References

[1] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[2] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002.

[3] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[4] Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. Ontonotes: the 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*, pages 57–60. Association for Computational Linguistics, 2006.

[5] Kevin Lin and Eugene Wu. Searching for meaning in rnns using deep neural inspection. 2017.

[6] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[7] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

[8] Thamar Solorio, Elizabeth Blair, Suraj Maharjan, Steven Bethard, Mona Diab, Abdelati Hawwari, Fahad AlGhamdi, Julia Hirschberg, Alison Chang, et al. Overview for the first shared task on language identification in code-switched data. 2014.

[9] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.