Novel Role Filler Generalization for Recurrent Neural Networks Using Working Memory-Based Indirection

by
Blake Mullinax

A thesis presented to the Honors College of Middle Tennessee State University in partial fulfillment of the requirements for graduation from the University Honors College

Fall 2020

Novel Role Filler Generalization for Recurrent Neural Networks Using Working Memory-Based Indirection

by
Blake Mullinax

APPROVED:

____

Dr. Joshua L. Phillips, Thesis Director
Associate Professor, Computer Science

____

Dr. Mary A. Evins, Thesis Committee Chair
Research Professor, History and
University Honors College

ABSTRACT

Humans encounter and adapt to novel situations every day. However, adaptation is not a trivial task to accomplish. In the field of machine learning, the statistical underpinnings of established deep learning architectures make it difficult for these architectures to handle certain types of novel situations. Previous research demonstrates how computational models could better handle novel situations through indirection, an idea inspired by the interaction between two regions of the human brain: the prefrontal cortex and the basal ganglia. This thesis demonstrates that combining the indirection model with deep learning methods outperforms current architectures.

TABLE OF CONTENTS

LIST OF FIGURES v

I. INTRODUCTION 1

II. BACKGROUND 2

III. METHODOLOGY 9

A. MODELS 9

B. EXPERIMENTS 13

C. ANALYSIS 14

IV. RESULTS 16

V. DISCUSSION 18

WORKS CITED 21

APPENDIX 22

LIST OF FIGURES

Figure 1: A representation of the working memory stripes and gates of the indirection model. The model consists of three stripes of working memory, each with their own input and output gate. Gates can be in closed states, like stripes one and three, or in opened...22

Figure 2: A demonstration of the encoder-decoder working in cooperation with the indirection model, split into three frames. The filler is presented to the encoder, and the encoding is stored in memory in the second frame. In the third frame, the encoding is outputted and given to the decoder to reproduce the filler. 23

Figure 3: The encoder-decoder in both coupled and decoupled form. Layers are represented by each oval. The encoder and decoder each receive one token as input through layer t. The token is wrapped up into a representation by the encoder, and the hidden states of the encoder are used to initialize the states of the decoder, seen as the dotted lines from the encoder to the decoder..24

Figure 4: An abstract overview of the nested indirection model. Each filler in the sentence is fed through the outer encoder, just like the indirection model. Then each encoding is given to the inner encoder, which wraps these into a single encoding for the sentence, seen at the top left. This process is then reversed, feeding the encoding to the inner decoder, which passes its output to the outer decoder. 25

Figure 5: Accuracies of indirection and nested encoder-decoder models for each of the four combination types. Error bars represent ± 1.96 standard errors and approximate 95% confidence...25

Figure 6: Accuracies for all variations of the end-to-end encoder-decoder. Error bars represent ± 1.96 standard errors and approximate 95% confidence.................................26

Figure 7: Accuracies for the nested encoder-decoder with a query time step. Error bars represent ± 1.96 standard errors and approximate 95% confidence.................................26

I. INTRODUCTION

Every day, one encounters new experiences. Contexts may be similar, but one never experiences a day that is truly identical to the one that came before it, and yet, humans are able to adapt. An unexpected car wreck blocks the way to work one morning, and fleets of cars struggling to make it to work on time exchange the normal route for a detour. New slang becomes popular and suddenly a word takes on an entirely new meaning. Life is full of substitutions and improvisation. Humans handle unique scenarios daily. And although artificial neural networks draw inspiration from the human brain, this is no trivial task for machine learning to emulate. Neural networks are heavily dependent on past experiences. If the network has not experienced a particular scenario before, the new scenario will be treated in a manner similar to how the network has treated past experiences. Many times accommodating new scenarios in this way is a sufficient response and the desired outcome, but there are many cases where this does not lead to a correct response. For example, when humans encounter a new word, they need to be able to treat it as such, rather than replacing it with a word they are more familiar. Since humans are able to solve this task, one might then look more closely into neurobiology for inspiration to determine how neural networks might better solve these new experiences.

II. BACKGROUND

Neural networks must undergo a training and testing process. Training is conducted by presenting many different "experiences" as input and feedback data for the neural network. Based on such training data, the network can learn how to respond in the future when given novel, but similar, inputs. If successful, the network will be able to *generalize*: apply useful knowledge gained from experience with the training data to then also respond appropriately to the novel testing data experiences. However, a factor that greatly influences the performance is the architecture of the network itself. Where a certain architecture of a neural network may perform poorly, another architecture may excel. Due to a statistical dependence on what they have been trained on, modern neural network architectures struggle when given situations they have not experienced, or *novel inputs*. Krete and Noelle, drawing inspiration from the brain, researched a method to improve performance when experiencing novel inputs. Krete and Noelle studied the relation between two areas of the brain known as the prefrontal cortex (PFC) and the basal ganglia (BG). In the prefrontal cortex there exist stripes of intensely intraconnected neurons, which are sparsely interconnected with each other (Elston et al., 2011; Pucak et al., 1996). The BG is connected to these stripes (Alexander, DeLong and Strick, 1986). It is believed that the BG working in tandem with the PFC controls the function of working memory (O'Reilly and Frank, 2006; Hazy, Frank and O'Reilly, 2006). When given information that must be used as contextual cues for a task, the stripes in the PFC hold this information in place. The BG controls whether to inhibit or excite this activity. This interaction between the BG and the stripes of neurons in the PFC is correlated with the function of working memory. Separate from short term and episodic memory, working memory is used to actively hold information used in a task at hand. For instance, when dialing a new phone number, one must actively hold the number in the mind. Kriete and Noelle hypothesize that working memory aids in the ability to handle novel combinations of previously experienced elements. An example they give is of one learning to play a new melody from previously learned chords. They believe this capability comes from the brain's ability to map values to "variables" to represent unique, new concepts, similar to the concept of a pointer in computer science. Rather than a stripe in memory storing a representation of a concept itself, the stripe might instead store a reference representation that points to the concept somewhere else in the brain. Kriete and Noelle created a computational model to simulate these processes in a neurobiologically plausible framework, based on the concept of indirection. Indirection refers to the ability of the PFC to serve as an abstract container for concepts already stored in memory. Instead of storing this concept anew, the stripes in memory can reference learned concepts and combine them in new and meaningful ways.

Kriete and Noelle begin with a simple three-word sentence that contains an agent, a verb, and a patient. The concrete concept being stored in memory, in this case a word in a sentence, is known as a *filler*. However, the filler's abstract function in the sentence, in this case agent, verb, or patient, is known as the *role*. Based on the role-filler combinations included in the training and testing set, the model's performance is assessed on different types of word combination tasks. Three different types of combinations were tested: standard generalization, spurious anticorrelation, and full combinatorial. These different tasks each probe a different generalization capability: the ability to apply past knowledge appropriately to new experiences. For each task type, words are selected from a list of ten, giving one thousand possible sentence combinations. A simple recurrent network (SRN) was used as a control against which to compare their models. During testing for spurious anticorrelation and full combinatorial, Kriete and Noelle found that SRNs did not generalize well. The SRNs achieved a lower accuracy than the indirection model on all generalization tests and achieved below thirty percent accuracy on spurious anticorrelation and full combinatorial, which will be discussed in more detail below. Because SRNs learn from recurring patterns in the environment, lack of generalization in this case is to be expected. However, the indirection model did generalize well on spurious anticorrelation and even for full combinatorial. Through trial-and-error learning, the indirection model can match the appropriate roles (agent, verb, patient) to fillers (the words) and can do so for fillers that have not been seen in specific roles before. However, the model can only represent *fillers* that the model has seen before, so some preliminary word-recognition pretraining was required. Another downside is that the indirection model contains many neurobiologically-motivated details that make such models difficult to construct and slow to train in practice.

Jovanovich builds upon the research from Kriete and Noelle by aiming to create a simplified model using modern neural network techniques that could be more easily reproduced, trained, and tested. In particular, the PFC components were replaced with holographic reduced representations (HRRs), and the BG components were replaced with temporal difference (TD) learning.

Temporal Difference (TD) learning is a type of reinforcement learning. The model learns by receiving a reward signal, but this signal can be very sparse. It may take many actions before the model receives the reward signal. By exploring the environment, the model can learn the *policy*. Learning the policy means the model can predict which actions will lead to the greatest reward in the future. Different types of TD learning algorithms exist. SARSA and Q-learning are two of which to take note of. Jovanovich used the SARSA algorithm in his model. The indirection model discussed in this paper will use Q-learning. Q-learning assigns values to state-action pairs. Given a particular state and an action, the Q-function returns an estimated value of discounted future rewards. The model will choose the state-action pair that returns the greatest value, hopefully leading to a reward signal. Through trial and error this function is updated to more accurately reflect the environment so that the model can more frequently obtain the reward signal.

Plate introduced the idea of Holographic Reduced Representations (HRRs). Provided HRRs of sufficient size, one can create orthogonal encodings to represent any concept one desires. Through circular convolution, these vectors can be combined with one another to represent more complex concepts. For example, given an HRR for a light and an HRR for the color green, through circular convolution of these two vectors one could represent the concept of a green light. Because these encodings are orthogonal to each other, they can be used to represent independent concepts as inputs to a neural network. Orthogonality and circular convolution make HRRs convenient for use as inputs to neural networks. They can represent individual concepts as well as a compositional construction of those concepts. In the case of Jovanovich's model, the HRRs can be used to represent the different roles and actions for the input and output gates which are discussed below.

Jovanovich’s model consists of four discrete time steps: three learning steps and one query step. If the model responded correctly at the end of the query, then it was given a reward signal. If the model responded incorrectly, a reward signal was withheld. Using a single layer network with working memory and input and output gating, Jovanovich's model was able to perform with similar accuracy to Kriete and Noelle’s. Jovanovich tested for the same combinations as Kriete and Noelle in addition to a new combination:

- **Standard Generalization -** During training, each filler has been seen in each role, and each filler has been seen in a sentence with all other fillers. The testing set consists of new sentences of these familiar role-filler pairs. For example, the model saw “a b c,” “e b a,” and “c a b” during training. The model is presented “c b a” during testing. *A, b,* and *c* were all seen together in a sentence during training. *C* was tested in the first role, and it had been trained in the first role. *B* was tested in the second role, and it had been trained in the second role. *A* was tested in the third role, and it had been trained in the third role.
- **Spurious Anticorrelation -** Each filler has been seen in each role. However, pairs of fillers have not been seen in the sentence together with each other. Testing consists of introducing these unseen pairs of fillers into each sentence. For example, the model saw “e b a” and “c b e” during training. The model is presented “c b a” during testing. Because *c* and *a* have been seen in these roles before but not in the same sentence together, this is a spurious anticorrelation, which is anticipated to be a more

challenging generalization to make compared to standard generalization above.

- **Full Combinatorial -** Not all fillers have been seen in each role. During training, a subset of fillers is withheld from the agent role. During testing, the model is presented with these fillers in the agent role. For example, the model was trained on “e b c,” “e a c,” and “e b a.” The model is tested on “a b c.” Because the model never saw *a* as the agent during training, this is an example of full combinatorial generalization.
- **Novel Filler -** A filler has never been seen before in any role. This is the most difficult for models to handle, as it involves a filler the model has no experience with whatsoever. During testing, the model is presented with a sentence containing this unseen, or novel, filler. For example, the model was trained on “a b c,” “b a c,” and “c a b.” The model is presented “z b c” during testing. Since the model never saw *z* during training but was still tested on the filler *z*, this is an example of novel filler.

Jovanovich’s model with no assumed action lacks the mechanisms necessary to achieve a high accuracy. Due to architectural constraints and lack of pretraining, the model could only output as many representations as it had experience with. Jovanovich notes that, given the proper mechanisms, the model could reach higher performance, presumably as good as the pretrained model of Kriete and Noelle.

An encoder-decoder is an architecture of neural networks that allows a set of input tokens to be mapped to a set of output tokens. Encoder-decoder networks are an established deep learning architecture that are often found in fields such as machine translation to translate between languages. In this way, the encoder-decoder architecture may be used to create a representation from a string of characters, i.e., a filler, that could be stored in memory to later be decoded. To our knowledge, besides basic LSTM models, encoder-decoder models have never been trained or tested on the generalization tasks described above. It is possible that the additional representational power provided by the encoder-decoder architectures may be successful on the tasks. While this would be unexpected, encoder-decoder performance would still provide a good baseline for comparison for indirection models. Before encoder-decoder models can be applied to a problem, they must be trained. Therefore, an end-to-end encoder-decoder that could handle both the encodings of fillers and the sentence task itself would be of interest.

The indirection model alone was unable to solve novel fillers in Jovanovich's research without an assumed action interpretation. The model relied on HRRs to store representations in memory. However, because these were created during training, the model did not have a method to output the filler when queried later during testing. To work around this limitation, the assumed action (AA) model was created which simply assumes that the stored representation released to downstream processes was the correct one for that filler. Therefore, the indirection model's ability to represent a filler is dependent on the method used to encode fillers. However, its ability to store or retrieve a filler is independent of the representation of fillers. So, for the indirection model, how fillers are *represented* is abstracted away from the function of how fillers are *stored and queried*.

III. METHODOLOGY

A. MODELS

Indirection

The indirection model and all subsequent models were programmed in Python using Keras Tensorflow. The indirection model consists of three working memory stripes, shown in Figure 1. Each working memory stripe has its own input and output gate. Each memory stripe's input and output gate exist to control whether fillers will be stored in the stripe or forwarded downstream, respectively. As seen in Figure 1, each gate can be in an opened position, as the input gate for stripe two, or in a closed position, as shown in stripes one and three. The input and output gates are controlled by single layer neural networks trained using Q-learning. The input gates are given an HRR as input to determine whether the word will be stored in the gate's respective stripe of memory. A role HRR is convolved with the store HRR to represent the current experience of the agent. This HRR is then convolved with HRRs for the actions open and close, respectively, to create the input HRRs. These two HRRs represent the current state-action pairs being considered by the neural network, and the network's job is to report Q-values for the two pairs. The HRR that produces the greater Q-value from the input gate determines whether the gate will open or close. For example, if the agent of the sentence is presented to the model, the input gate will receive an HRR representing **agent**, **store**, and **open**. The input gate then receives an HRR representing **agent**, **store**, and **close**. The action, **open** or **close**, that produces a higher output from the gate will be chosen. If opened, the *filler* will be allowed to pass through the gate and be stored in the working memory slot analogous to how the PFC stripes in the brain purportedly hold onto active task representations.

In order to overcome the AA limitation, an encoder-decoder neural network is used which can perform the minimal generalization skills needed to represent novel fillers. Figure 2 illustrates how the encoder-decoder works in cooperation with the indirection model in three frames. In the first frame, the filler “John” is presented first to the encoder, which produces a corresponding representation. If an input gate decides to open for this role, the encoded filler will pass through the gate into that stripe of working memory, as in frame two. In the third frame, the output gate decides to open, and the encoding is passed through the gate to the decoder, which translates the encoding into the appropriate filler.

Figure 3 demonstrates the architecture of the encoder-decoder model. Each oval represents a layer of the neural network. The encoder and decoder each contain their own Long Short Term Memory (LSTM) layer, seen in black. The encoder and decoder receive one token each as input through layer *t*. In the case of encoding and decoding fillers, each token represents one letter of the filler. Start and stop tokens were used for the decoder to delineate the beginning and ending of words. This delineation allows for variable lengths of tokens should one want to train on different lengths of words. The letters and start/stop tokens were represented using one-hot encodings. One-hot encodings are a standard method for encoding discrete data. A vector of zeros the size of the number of discrete items of data is created. The vector represents one of the discrete concepts depending on which element of the vector is “hot,” i.e., set to one. One-hot encodings are convenient for being used as neural network inputs and targets. The encoder-decoder is trained coupled together. The tokens are presented one by one during each time step. The token is wrapped up into a representation by the encoder, and the hidden states of the encoder are used to initialize the states of the decoder, seen as the dotted lines from the encoder to the decoder. After training, the encoder and decoder are split apart to be used in the indirection model. Instead of the initial state of the decoder being copied directly from the encoder, it is received through input layers. The separated encoder and decoder can then be placed into their respective roles in the indirection model.

Nested Encoder-Decoder

To compare the performance of modern architectures against the indirection model, a nested encoder-decoder was built. This model contrasts the indirection model by having a similar architecture but uses only established deep learning techniques. An *outer* encoder-decoder handles the representations for the fillers, just as it did for the indirection model. However, an *inner* encoder-decoder is used to encode the three word sentence instead of the indirection framework. Figure 4 illustrates this process, beginning in the bottom left corner. Each filler in the sentence is fed through the outer encoder, just like the indirection model. Then each encoding is given to the inner encoder, which wraps these into a single encoding for the sentence, seen at the top left of Figure 4. The sentence encoding is then given to the inner decoder, which unwraps the sentence encoding into the respective encodings for each filler in the sentence that was given to the model. The construction of the inner encoder-decoder is similar to that of the outer encoder-decoder with additional input and output layers. The inner encoder receives three tokens as input: two input layers for the encoding from the outer encoder and one input layer for a start and stop token. The start and stop token is represented through one-hot encoding and used to delineate the beginning and end of the sentence. The output of the decoder outputs these three tokens: two state vectors and a token for start and stop. When decoupled, the inner decoder receives two more input layers to initialize its internal state from the encoder.

End-to-End Encoder-Decoder

To eliminate the need for pretraining the encoder-decoder, an end-to-end encoder-decoder was built. The end-to-end encoder-decoder resembles the structure of the outer encoder-decoder. However, the encoding of each filler as well as the encoding of the sentence is accomplished through one model. Rather than having the nested structure of the previous model, the end-to-end architecture attempts to solve the problem with only one encoder-decoder. For example, "John", "Ate", and "Fish" would all be presented to the end-to-end encoder-decoder. Rather than producing intermediary encodings for the fillers, the end-to-end model directly produces the sentence encoding. This is a more difficult task to accomplish. Two levels of start and stop tokens were created. Sentence-level start and stop tokens were used to delineate the beginning and ending of the sentence. Filler-level start and stop tokens were used to delineate between the fillers in the sentence.

End-to-End Encoder-Decoder with Query

In addition to the end-to-end encoder-decoder model, another end-to-end encoder-decoder was created that could receive information about the roles of fillers and be queried about a given role. The architecture was the same as the previous model, but with one additional input layer to receive information about the roles. With this extra input layer, the model could be given roles to be associated with the fillers and could be queried.

Nested Encoder-Decoder with Query

A variation of the nested encoder-decoder was also built. This model can also accept information about the roles of the fillers and be queried, so that it more similarly resembles the sentence task for the indirection model. The outer encoder-decoder remained the same, but the inner encoder-decoder received an extra layer of input. This layer, like the end-to-end model, received information about the roles to be associated with the fillers, as well as the query for the filler to be outputted.

B. EXPERIMENTS

Each model was presented with the sentence task similar to that of Krete and Noelle and Jovanovich. A training and testing set was generated for each of the four types of combinations containing ten fillers. The size of the training set and testing set for each type of generalization was 200 sentences and 100 sentences, respectively. Each sentence contained three fillers, corresponding to the actor, verb, and patient. Fillers of a length of five letters were used. Each model was then presented with the three fillers of the sentence.

The sentence task for the indirection model involved four time steps: three store time steps plus one query time step. During each store time step, a filler from the sentence is presented to the model on each of the three store time steps. The filler is encoded and stored in a stripe of memory if an input gate opens. On the fourth time step, the model is queried for a randomly chosen role. If only one output gate opens containing the appropriate filler, then the model receives a reward signal of 1. A Huber Loss Function was used along with a stochastic gradient descent optimizer with a learning rate of 0.1. HRRs of size 1024 were used to encode the inputs to the input gate and output gate. An epsilon-greedy action selection policy with an epsilon value of .025 was used so that the model would occasionally choose a random action to explore more possibilities while learning. The encoder-decoder was constructed using the functional Application Programming Interface (API) in Keras. The encoder-decoder was pretrained on a corpus of ten thousand words for four hundred epochs. It achieved approximately 95% accuracy and a loss of approximately .16.

The sentence task for the nested encoder-decoder also involved four time steps. During the first three time steps the fillers of the sentence are presented to the model. During the fourth time step, the model reproduces the entire sentence as it was presented to it. The sentence task is identical for the end-to-end encoder-decoder model.

C. ANALYSIS

Each model was run ten times for each type of combination. The indirection model was run until it achieved an accuracy of ninety-five percent or greater or until it reached 100000 epochs. The nested encoder-decoder and nested encoder-decoder with a query time step were run for 1600 epochs. The end-to-end models were run for 600 epochs. Accuracy was calculated for both the words and letters. Word-level accuracy was calculated as the fraction of completely correct words the model outputted out of the total amount of words. Letter-level accuracy was calculated as the fraction of how many letters the model outputted correctly compared to the total number of letters in all of the words. Mean accuracies and the standard error of the means were calculated from the ten samples. Barplots and graphs for all accuracy data were generated using Python and Matplotlib.

IV. RESULTS

Figure 5 presents the accuracies of the indirection model and nested encoder-decoder for all four types of combinations. The indirection model achieved one-hundred percent accuracy across all four types of combinations for both word-level and letter-level accuracy. The nested encoder-decoder model achieved word-level and letter-level accuracies above ninety percent for two of the four combination types: standard generalization and spurious anticorrelation. For full combinatorial, the nested encoder-decoder achieved a letter-level accuracy below forty percent and a word-level accuracy below twenty percent. Accuracy for novel filler is higher, with a letter-level accuracy of approximately sixty-one percent and a word level accuracy of approximately forty-nine percent. That the model would perform better on novel filler than full combinatorial is an unexpected result. Novel filler would appear to be the more difficult task, yet the nested encoder-decoder performs less optimally on full combinatorial. Letter-level accuracy is much higher than word-level accuracy. This indicates the model rarely gets an entire word correct but more frequently gets a majority of the letters in a word correct.

Figure 6 presents the accuracies for the end-to-end encoder-decoder model with a query time step. The prescient model achieved a word-level accuracy of approximately fifty percent with letter-level accuracy of approximately eighty-four percent on both standard generalization and spurious anticorrelation. For full combinatorial and novel filler, the end-to-end encoder-decoder achieved zero percent accuracy on the word level. For letter-level, the model achieved approximately fifty-seven percent and sixty-eight percent respectively. For the non-prescient model not given the roles, the model achieved a relatively consistent accuracy across all four combination tasks. The highest of which was for standard generalization, with an average word-level accuracy of thirty-seven percent and an average letter-level accuracy of thirty-eight percent. The end-to-end encoder-decoder model given the corresponding roles with one query time step achieved a mean word-level accuracy greater than that of the other end-to-end encoder-decoder models on all four combination types. For full combinatorial, it achieved a word and letter-level accuracy of approximately fifty-nine percent. For novel filler, it achieved approximately fifty-five and fifty-six percent accuracy for word-level and letter-level, respectively.

Figure 7 shows the accuracy of the nested encoder-decoder with a query time step. The model achieved less than thirty percent accuracy across all types of combinations. The word-level accuracy was 5.1 percent or less across all types of combinations.

V. DISCUSSION

The indirection model equipped with an encoder-decoder was able to match performance with Jovanovich's assumed action model and was able to outperform the model for novel fillers. The indirection model also outperforms the established architectures for full combinatorial and novel filler. Full combinatorial and novel filler are much more difficult than standard generalization and spurious anticorrelation. To our knowledge, modern architectures have not been directly tested for novel fillers. When tested for full combinatorial and novel filler, established architectures suffer. These two types of combinations present the model with inputs that greatly differ from what was seen during training, and therefore performance greatly drops. The indirection model's architecture allows it to not be focused on what the contents of the filler actually are. This allows it to focus on the roles of fillers. Through indirection, the model can achieve these higher accuracies. Statistical dependencies undergird previous models, and as a result the performance suffered. A higher order relational mechanism, i.e., indirection, was needed to improve performance. By abstracting away the details of the filler, the indirection model could overcome the statistical hurdles of full combinatorial and novel filler but lacked the means to output a novel word stored in memory. Coupling the indirection model with an encoder-decoder surmounts this problem. This novel indirection model does not suffer in performance when facing full combinatorial or novel filler tasks, as demonstrated by the results in Figure 5, and it outperformed established deep learning techniques across all four types of combinations.

The end-to-end models were designed to further test these architectures. As seen in Figure 6, the prescient model achieved a higher accuracy than the model given no roles during store time steps. The model without role information does not know what will be queried in the coming steps, while the prescient model knows from the beginning which role will be queried. This accounts for the prescient model's higher performance. The end-to-end encoder-decoder with the best performance was given the corresponding role with each filler during the store time steps. This provided more information with which the model could form associations. The prescient model receives which role will be queried at the end but no direct input about which roles correspond to which fillers. The word-level accuracy dropped to zero for full combinatorial and novel filler. The drop in accuracy points to a problem of overfitting. The model sacrifices the word-level accuracy to achieve a higher letter-level accuracy. The model may be caught in a local minimum. Achieving a higher accuracy for both word-level and letter-level may require the model to first have a lower letter-level accuracy, thus causing the model to be trapped in a local minimum. One cause getting caught in a local minimum may be that loss is calculated at the letter level. If loss were calculated as a combination of word-level and letter-level, it may provide more of an incentive for the model to escape this kind of local minimum. Until these problems are solved, an end-to-end indirection model cannot be constructed. Solving these training issues is a nontrivial task that will require further exploration.

Integration of the indirection model into the Keras library would make an end-to-end indirection model more feasible. However, the nested structure of the model may interfere with performance. For instance, if the outer encoder-decoder is incorrect, but the indirection model opened the correct gate, should it be punished as well? If the indirection model opens the incorrect gate, but the decoder decodes the filler correctly, should the decoder be punished? How to resolve these issues is not yet clear and will require further research.

The nested encoder-decoder with a query time step achieved a poor accuracy and encountered an overfitting problem similar to that of the end-to-end encoder-decoder. In Figure 7, one can see the disparity between word-level and letter-level accuracies. Again, the drop in accuracy points to a problem of overfitting. The model is sacrificing word-level accuracy to achieve higher letter-level accuracy. Further exploration will be required to determine how to make these models perform with better accuracy if possible.

Overall, the use of indirection appears necessary to solving the full combinatorial and novel filler tasks. Combining the indirection framework with deep-learning methods resulted in overcoming the assumed-action limitation observed by Jovanovich. While pretraining was still required due to unresolved technical limitations, the end-to-end deep learning models also clearly demonstrated an inability to solve these tasks. This inability suggests that future work integrating the indirection framework fully into Keras/Tensorflow to allow full end-to-end training is a promising direction for future research.

WORKS CITED

Alexander, G E, M R DeLong, and P L Strick. "Parallel Organization of Functionally Segregated Circuits Linking Basal Ganglia and Cortex." *Annual Review of Neuroscience* (1986): 357-81.

Elston, G N, et al. "Pyramidal Cells in Prefrontal Cortex of Primates." *Frontiers in Neuroanatomy* (2011).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Hazy, Thomas, Michael Frank, and Randall O'Reilly. "Banishing the Homunculus: Making Working Memory Work." *Neuroscience* (2006): 105-18.

Jovanovich, Mike. "Biologically Inspired Task Abstraction and Generalization Models of Working Memory." Master's thesis, Middle Tennessee State University, 2017. http://jewlscholar.mtsu.edu/xmlui/handle/mtsu/5561.

Kriete, Trenton, et al. "Indirection and Symbol-like Processing in the Prefrontal Cortex and Basal Ganglia." *Proceedings of the National Academy of Sciences* (2013): 16390-95.

O'Reilly, Randall C and Michael J Frank. "Making Working Memory Work: A Computational Model of Learning in the Prefrontal Cortex and Basal Ganglia." *Neural Comput.* (2006): 283-328.

Plate, Tony. *Holographic Reduced Representations: Convolution Algebra for Compositional Distributed Representations*. Sydney: Morgan Kaufmann Pubishers Inc., 1991.

Pucak, Michele L, et al. "Patterns of Intrinsic and Associational Circuitry in Monkey Prefrontal Cortex." *Journal of Comparative Neurology* (1996): 614-30.

APPENDIX

Models and code pertaining to this thesis can be located at https://github.com/ChaningBlake/thesis

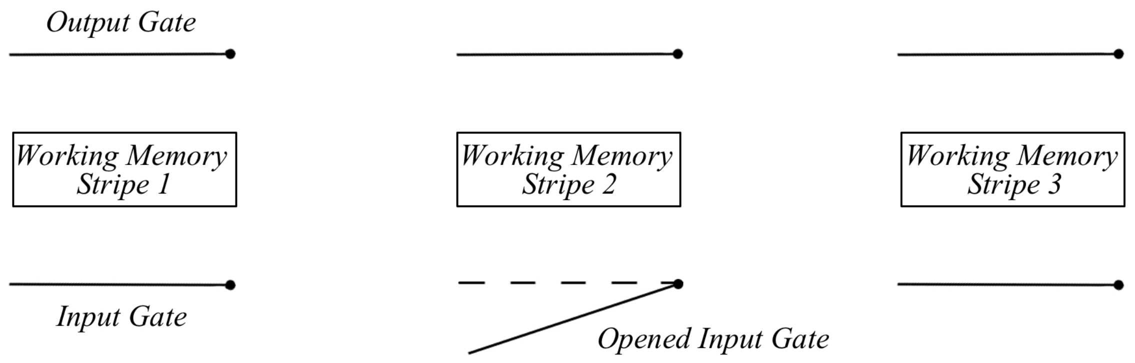

Figure 1. A representation of the working memory stripes and gates of the indirection model. The model consists of three stripes of working memory, each with its own input and output gate. Gates can be in closed states, like stripes one and three, or in opened states.

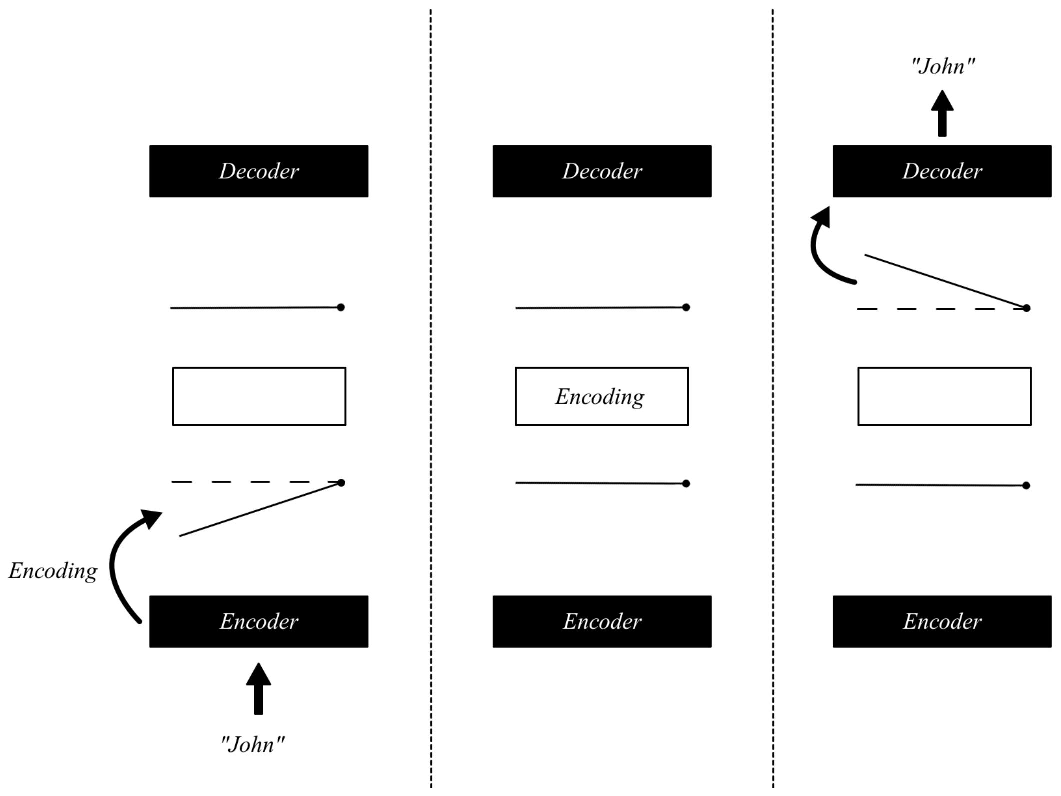

Figure 2. A demonstration of the encoder-decoder working in cooperation with the indirection model, split into three frames. The filler is presented to the encoder, and the encoding is stored in memory in the second frame. In the third frame, the encoding is outputted and given to the decoder to reproduce the filler.

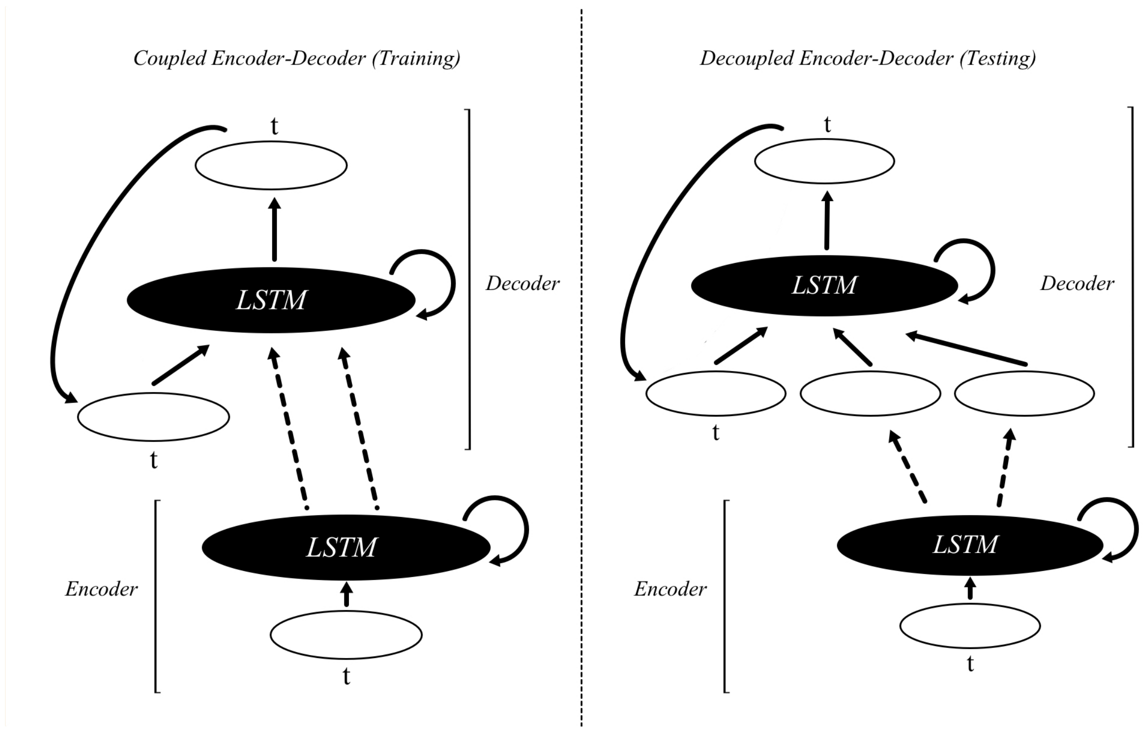
Coupled Encoder-Decoder (Training)
Decoupled Encoder-Decoder (Testing)
t
LSTM
Decoder
t
LSTM
Encoder
t
t
LSTM
Decoder
t
LSTM
Encoder
t

Figure 3. The encoder-decoder in both coupled and decoupled form. Layers are represented by each oval. The encoder and decoder receive one token as input through layer t. The token is wrapped up into a representation by the encoder, and the hidden states of the encoder are used to initialize the states of the decoder, seen as the dotted lines from the encoder to the decoder.

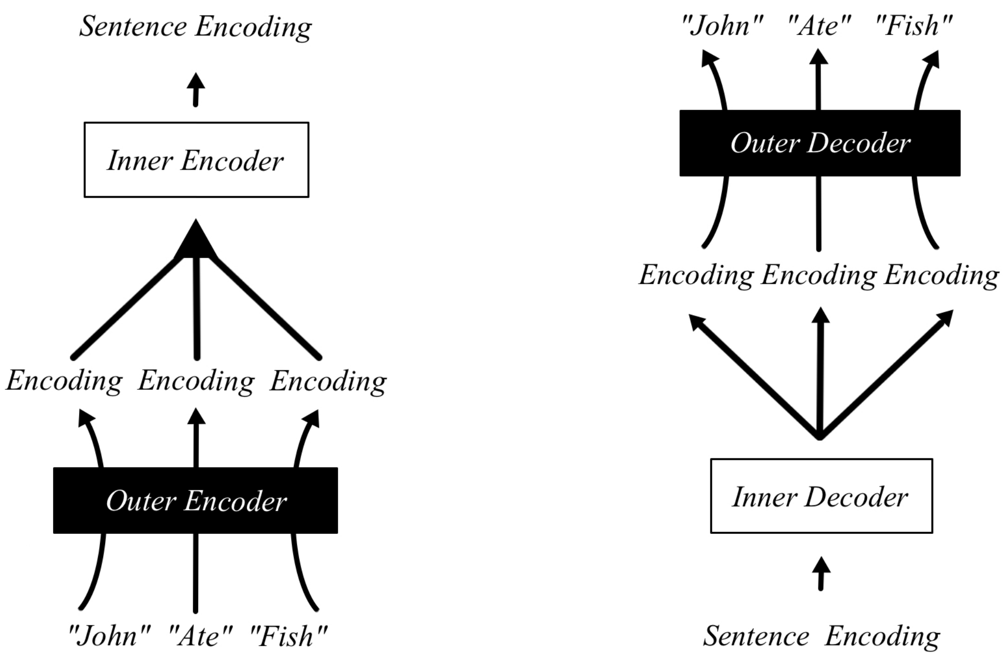

Figure 4. An abstract overview of the nested indirection model. Each filler in the sentence is fed through the outer encoder, just like the indirection model. Then each encoding is given to the inner encoder, which wraps these into a single encoding for the sentence, seen at the top left. This process is then reversed, feeding the encoding to the inner decoder, which passes its output to the outer decoder.

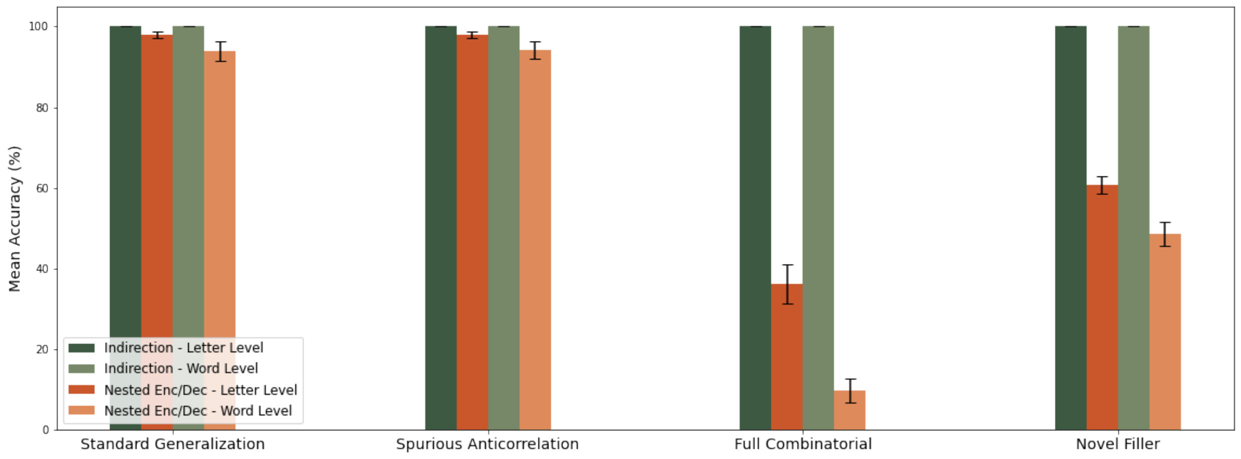

Figure 5. Accuracies of indirection and nested encoder-decoder models for each of the four combination types. Error bars represent ± 1.96 standard errors and approximate 95% confidence. The indirection model outperforms the nested encoder-decoder across all four types of combinations.

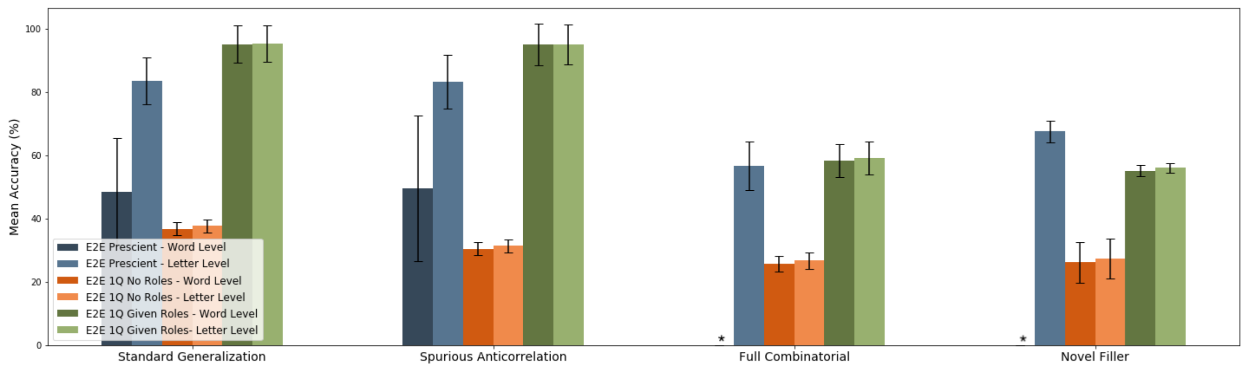

Figure 6. Accuracies for all variations of the end-to-end encoder-decoder. Error bars represent ± 1.96 standard errors and approximate 95% confidence. The end-to-end suffer greatly for full combinatorial and novel filler. Large gaps between word-level and letter-level accuracy can be seen for the prescient model, which signifies an overfitting problem.

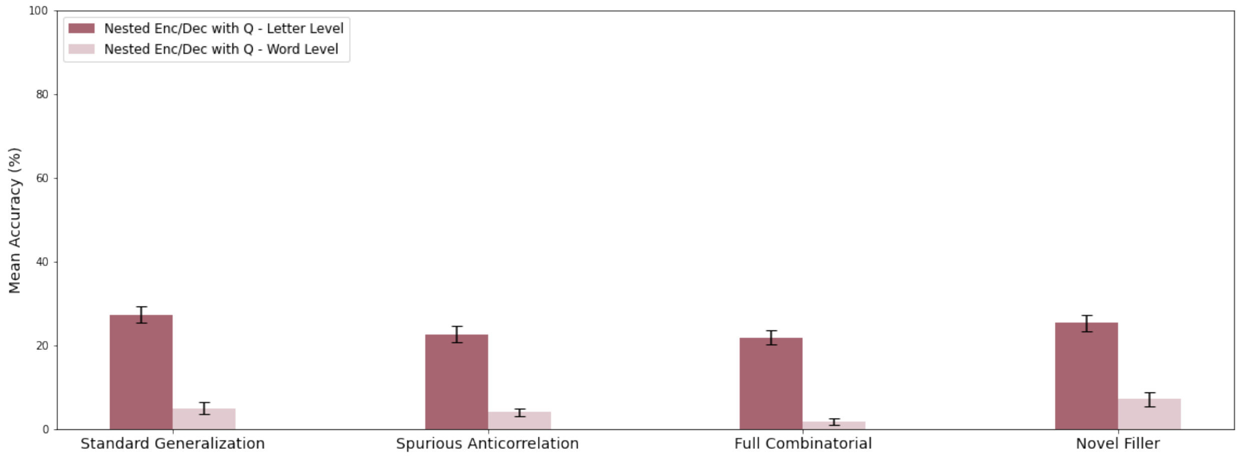

Figure 7. Accuracies for the nested encoder-decoder with a query time step. Error bars represent ± 1.96 standard errors and approximate 95% confidence. The nested encoder-decoder achieves a low accuracy across all combination types. The wide disparity between word-level and letter-level accuracy signifies an overfitting problem.