Mimicking Human Brain for Language Processing

(Major Project Report)

Submitted by

Rajeev Bopche

(2017CSZ3751)

Under the supervision of

Prof. Shantanu Chaudhary

Department of Computer Science

Indian Institute of Technology Delhi

New Delhi - 110016

Certificate

This is to certified that this project entitled “Mimicking Human Brain for Language Processing” being submitted by Rajeev Bopche, Entry no. 2017CSZ3751, to the Department of Computer Science, Indian Institute of Technology Delhi, New Delhi, India, in partial fulfilment of the requirements for the award of the degree of Masters of Technology, is a bonafide record of research work carried out by him under my guidance and supervision. To the best of my knowledge, the results embodied in this project have not been submitted in part or full to other University or Institute for the award of any degree or diploma.

Prof. Shantanu Chaudhary

Deptt of Computer Science

Indian Institute of Technology, Delhi

Date:

Place:

Acknowledgement

I would like to begin by thanking my advisor, Professor Shantanu Chaudhary , for his supervision and guidance.

Thank you to Dr Sakshi Agarwal for her encouragement, ideas, and invaluable help.

I am also grateful for the input from Shrikantha Bedathur for insights on NLP techniques.

Thank you to Neetu Kumari and Rahul Jasrotia for their discussion and contributions. Thank you to my family for all their love and support.

Above all I am very much thankful to the God for giving me this opportunity to research and participate in the prestigious Institution.

Contents

List of Figures

Chapter 1

The Give-and-Take between Natural and Human Language Processing

Computer scientists, neuroscientists, and linguists have been separately working on discovering the necessary mechanisms for language processing. According to Chomsky's theory of Universal Grammar (Chomsky, 1965), our brains are built with a natural set of structural rules for language, like the ability to distinguish nouns from verbs. Since then, however, others have criticized the existence of Universal Grammar and have hypothesized that in-stead of having an innate mechanism for categorizing words into different parts of speech, our acquisition of language may be explained by probabilistic models (Charter, 2006).

While we're still not sure exactly which mechanisms are responsible for human language processing, there has been great progress in observing the high-level neuronal circuitry that is involved with it. Perhaps closely examining the brain could help explain how it processes language. Developed in the 1950s, electrocorticography (eCoG) monitors brain activity by placing electrodes directly on the surface of the brain. Then in the 1980s, functional magnetic

resonance imaging (fMRI) offered a less obtrusive approach which uses magnets to locate regions in the brain undergoing increased metabolism. These technologies make it possible to observe the dynamic neuronal patterns involved with language processing, albeit in a noisy way.

A parallel problem to understanding human language processing is that of understanding natural language processing (NLP)—computer tasks like translation, auto-completion, and summarization where computers manipulate our human language. Like the progress seen with brain imaging, NLP too has experienced remarkable advancements over the past few years. Some break-through discoveries include the Word2vec embeddings (Mikolov, 2013), which map words into a relatively low dimensional space; the Transformer model (Vaswani, 2017), which uses attention mechanisms to achieve state-of-the-art performance at machine translation; and the Bidirectional Encoder Representations, also known as BERT, (Devlin, 2018) which performs multiple NLP tasks with minimal modifications to its architecture. Yet despite these recent successes, NLP still has not reached human performance. Computers cannot go beyond pattern recognition to truly comprehend a text and formulate it into a narrative.

Cracking natural and human language processing are difficult for the same reason: we haven't discovered the machinery that is necessary for mastering language. At first glance, trying to solve these problems may seem like a Catch-22: if we knew how the brain processed language, then we could design an AI that does it in a similar way. Conversely, if we had a brain-like AI that matched human performance in language, then we could analyze the com-putations in our model, and test whether the brain computes in a similar manner. But what if perhaps the two problems were actually symbiotic, and that solving them simultaneously

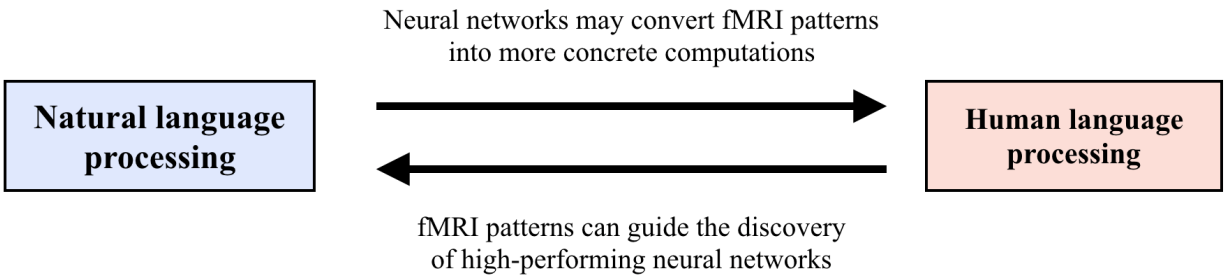

Figure 1.1: NLP and HLP are symbiotic.

may be easier than solving them independently? What if it were possible to bridge the gap between these two pursuits?

It appears like artificial intelligence and brain imaging are maturing to a critical point, and indeed, this maturation has inspired the ultimate scope of this endeavor: to build an AI that matches human performance at language by reverse engineering the brain by way of artificial neural networks; and at the same time, to better understand the brain by converting the opaque patterns from brain imaging data into more concrete computations via artificial neural networks.

It's tricky to understand how the brain processes language because we have such an opaque view of it. fMRI and eCoG data only show the collections of neurons that fire in a given time period. They don't explain what those neurons actually compute—that part is still a mystery. Since deep learning is exceptionally good at pattern recognition, one approach to begin interpreting the neuronal patterns in the brain is to find a neural network that can somehow mimic them. What if the brain's mechanisms for language were actually hiding in plain sight: if they were concealed inside the brain imaging data and the signal is so subtle or complex that we can't easily find it? If a neural network's activity somehow aligned very closely with the patterns in the brain data, then perhaps the network would be performing a

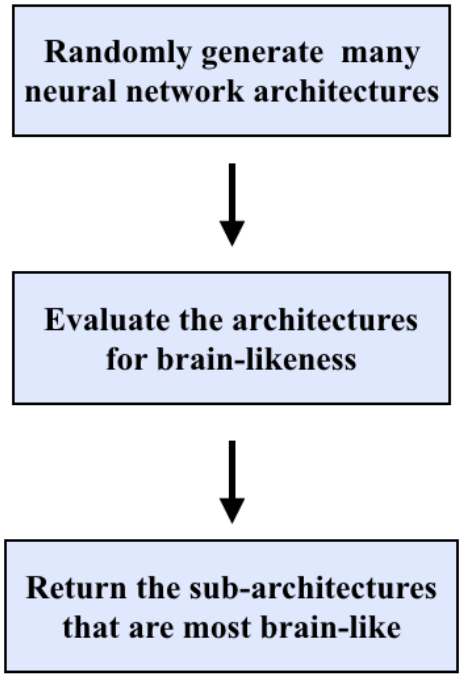

Figure 1.2: Overview for discovering brain-like computations.

computation that resembles that of the brain. We could then look at the computational graph of this "brain-like" neural network, and explicitly see the operations it uses. The network would be like a decoder, converting the brain imaging data into a network architecture whose computations are more concrete and apparent. The network would interpret and give meaning to the otherwise incomprehensible patterns in the brain imaging data. In this way, we can leverage AI to better understand human language processing.

On the other hand, we can leverage brain imaging data to develop a better NLP model. Under normal circumstances, the adult human brain contains all of the mechanisms that are necessary for mastering language. In fact, our brains are so good at parsing, encoding, and interpreting language, that we can often understand a text even when it contains grammatical errors. For instance, we can be fairly certain that, "Ice cream I wants" should actually be

“I want ice cream.” What if we could use fMRI or eCoG data to build an AI model that processes language with the same robustness that our brains have? We could boot-strap off of the brain imaging data to develop an AI that is optimized for language. The process for finding this AI is the same as before: find an artificial neural network that is as brain-like as possible. Data would flow through the network’s architecture in a way that somehow replicates the neuronal patterns of the brain. By optimizing a neural network to be as brain-like as possible, we may consequently uncover the signal which is hidden in the brain images, and create a network that is really good at language.

The rest of this thesis describes a method for discovering brain-like computations in neu-ral networks: Chapter 2 details the process of generating and training a large and diverse collection of neural networks; Chapter 3 discusses the tasks, datasets, and architecture scor-ing function for brain-likeness; and Chapter 4 describes a method for discovering the most brain-like sub-architectures within the collection of models.

Chapter 2

Generating Architectures

2.1 Representing an architecture

An architecture can be defined using a directed graph. Each node is assigned an operation, which is applied to the output of its parent node. For example, the operation could be adding two inputs together. The edges of the graph explain how the data flow through the graph. In particular, if there is an edge from node A to node B, then the output of A is fed as the input to B. The graphical representation is human readable and thus useful for when people wish to understand the structure of the architecture. Additionally, computer libraries for deep learning, like PyTorch and Tensorflow, use the graphical representations for defining the architectures.

We can convert this graphical representation into a string representation, which is helpful for performing analysis later on. Because the string representation doesn't need pointers, it uses less memory than the graphical representation. See Figure 2.1 for an example of how to convert between the graphical and string representations.

(1)

tanh(Add(MM(x_t), MM(x_t)))

(2)

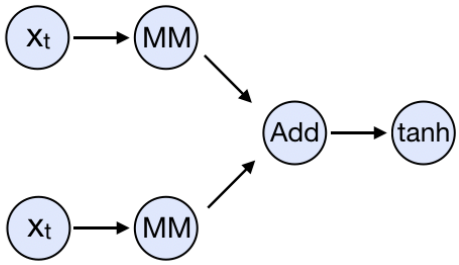

Figure 2.1: An example architecture represented as (1) a string, and (2) a graph. Notice that it is easier to see how data moves through the graphical representation.

2.2 Objective: an abundant and diverse collection of architectures

Our objective is to find networks with high brain scores—but how? One approach may be to look at existing architectures for NLP that perform well on the task they are designed for. For example, we could score the Transformer model for brain-likeness because it has a high BLEU score for translation. Perhaps accurate models for NLP also have high brain scores. Thus, we could collect a lot of existing NLP models that have state-of-the-art performance and measure how brain-like they are. However, this approach is restrictive. One of our goals is to use the brain-likeness to create better AI. But if the analysis is limited to only considering existing models, then we won't discover new computations that improve the current regime. If we instead generated a diverse collection of architectures, we increase the chances of discovering a new, successful computation. Another problem with only considering existing NLP models is that our analysis would not be as statistically rigorous. In order for a computation to be significantly brain-like, it

must appear in numerous models and contexts. We would not be sure whether the computation actually produces higher brain scores, or whether it did in that particular context. Our collection of architectures must be large enough that many of the computations appear in multiple models.

2.3 Randomly sampling architectures

This section presents one approach for generating a large collection of architectures with flexibility in the set of operations that they include. The approach is called architecture sampling, and the first step involves defining a set of operations that the architectures may include. Next, the sampler randomly generates and evaluates architectures from that set of operations. In this way, we may create a large and diverse collection of neural network architectures. To focus on the effect that the architecture has on brain-likeness, we keep all hyper-parameters, like the learning rate and hidden size, constant across all the models. This way, if a sub-architecture appear in more than one model, the comparison is more direct. We know that if the scores are different, then it's because of the architecture.

2.3.1 The architecture space

The architecture space is an abstract space in which each point represents an architecture. For example, perhaps one point in the space is an LSTM, and another point is the Transformer architecture. An architecture space is defined by a set of operations, and a constraint on the size of the architectures.

Recall that our goal for using architecture sampling is to have as much variety and

diversity in the architectures as possible. An obvious way to achieve diversity is for the space to contain many operations. If the architecture space is richer and is defined by more operations, then we increase the likelihood that the space contains an architecture with a high brain score. Some of the operations might work very well and produce very brain-like models, and other operations won't work as well and will produce models that are not very brain-like. On the one hand, the larger the architecture space becomes, the fewer assumptions we make about what a brain-like model should look like. It appears like it's better to have a larger and more diverse set of operations.

However, that is not the case. It is critical to choose the correct architecture space to sample from. We must pick the right operations and the right number of operations. The problem is that if we have too many operations, then our architecture space becomes so big that we can't possible explore enough of it. The larger the architecture space becomes, the less likely it is for a given operation or sub-architecture to appear in multiples architectures that we sample. That makes it more difficult to do a statistically significance analysis. Ideally we'd like the sub-architecture to show up in as many models as possible. Therefore, there is a strong tension between having less bias (and potentially more brain-like models), and having less statistical rigor in our final analysis.

2.3.2 Operators

In our case, we choose the fewest operations necessary to sample architectures that are known to have high performance in NLP, like the GRU, LSTM and Transformer models. See Figure 2.2. The set of functions is {layer norm, ReLU, tanh, softmax}. And the set operators is

(1) **GRU**

```
gate3('hm1',
            tanh(
                add(
                    mult(
                        sigmoid(add(MM('x'), MM('hm1'))),
                        MM('hm1')
                    ),
                    MM('x')
                )
            ),
            sigmoid(add(MM('x'), MM('hm1'))))
```

(2) **LSTM**

```
Mult(
     Sigmoid(Add(MM('x'), MM('hm1'))),
     Tanh(Add(
         Mult(
             Tanh(Add(MM('x'), MM('hm1'))),
             Sigmoid(Add(MM('x'), MM('hm1'))),
         ),
         Mult(
            'hm1',
             Sigmoid(Add(MM('x'), MM('hm1'))))))
```

Figure 2.2: GRU and LSTM architectures.

{add, mult, MM, scale, gate3, concat} where,

- Add(x, y) adds vectors x and y.
- Mult(x, y) computes a component-wise multiplication of x and y.
- MM(x) computes $Wx + b$, for some weight matrix W and bias b.
- Scale(x) scales x by the square root of its length.
- Gate3(x, y, f) computes the weighted sum $\sigma(f)x + (1 - \sigma(f))y$, where σ is the sigmoid function.
- Concat(x, y) concatenates vectors x and y.

Chapter 3

Evaluating Architectures

We train our models for language modeling, translation, and summarization. Given some sequence of words, the goal of language modeling is to predict the next word in the sequence. In particular, given the sequence of n words w_0, w_2, , w_{n-1}, we wish to learn a distribution over the words in our vocabulary so that we may predict the following word w_n. The source dataset is some corpus, for example a collection of books or articles. And the target dataset is just the source data shifted over by one word. Then the data are aligned for predicting the following word. Because each input word has a corresponding output, the model only has an encoder.

Unlike language modeling, sequence-to-sequence translation needs both an encoder and decoder. The goal of translation is to translate from a source language to a target language. The dataset consists of a corpus of bilingual sentence pairs, like a collection of English sentences as the source and their corresponding German translations as the target.

Like translation, summarization uses a sequence-to-sequence model with an encoder and decoder. The goal of summarization is to condense a text into fewer words and to distill

it to the point. The dataset is a corpus of full length documents as the source, and their corresponding summaries at the target.

3.1 Datasets

3.1.1 WMT

In order to make a more direct comparison, we use the WMT dataset for both language modeling and translation. In particular, machine translation uses the English sentences as the source and the German sentences as the target. On the other hand, language modeling uses the English sentences as the source and the same sentenced shifted by one as the target. We use the WMT14 training data consisting of 4.5 million sentences pairs. The complete vocabulary size for the English sentences is 116 million words, and 110 million for the German words.

3.1.2 CNN/Daily Mail

We use the CNN and Daily Mail dataset for summarization. It contains news articles from CNN and Daily Mail, and their corresponding summaries which were human generated. The summaries were written as bullet points, but we treat each one as a sentence. There are 286,817 summary pairs in the training set, 13,368 in the validation set, and 11,487 in the test set. The average source document has 766 words and 29 sentences. The average summary contains 53 words and 3.7 sentences. The dataset is released in two versions: one keeps all of the entity names, while the other has replaced infrequently occurring words with integer-ids.

3.2 Measuring a neural net for brain-likeness

Before we can measure a model for "brain-likeness," we first must define what that even means. How do we quantitatively score a neural network for how brain-like its computations are, at tasks like language model, translation and summarization? Certainly existing NLP accuracy scores, like the BLEU or Rouge scores, will not work for brain-likeness. The BLEU score specifically measures the quality of translation from one language to another, and the Rouge score evaluates the accuracy of summarizing larger texts into smaller snippets. These measures are not suitable for capturing the similarity between fMRI patterns and neural network activity. We need to define a new score if we want to be able to quantitatively score neural networks for brain-likeness.

3.2.1 Challenges

At first glance, it's not entirely clear how to do that. On the one hand, neural networks are defined by a collection of operations, and a graphical structure that explains how those operations interact with each other. On the other hand, brain activity data are in the form of fMRI images (or volumes), where each image is composed from a set of 3D pixels (or voxels) which measure the brain activity in that part of the brain. Somehow we must find a similarity score between these two different domains.

What makes the problem even tougher is that our fMRI data are captured while participants were listening to stories. The brain images depict brain activity for listening comprehension. But what if we want to measure the brain-likeness of a neural network trained for a task other than listening comprehension, like translation or summarization? It is very likely

that the neuronal activity changes depending on the task. The brain activity for translation or summarization probably looks different from that of listening comprehension.

Ideally, the task used while collecting the fMRI data should be the same task that the neural network is trained for. That would be a fairer comparison. Unfortunately, for now, the fMRI data are limited to listening comprehension, and it is expensive to collect enough new data for other tasks.

3.2.2 A quantitative approach

We propose applying the Generalized Linear Model (GLM), which is an extension of ordinary linear regression (OLR). The difference between them is that GLM allows the error terms to follow a distribution different from the Gaussian distribution. The GLM model learns a set of weights to map from neural network activations to fMRI voxel intensities. If the predicted voxel intensity is close to the actual intensity, then the model has a high brain score: the neural network activations are capable of explaining the brain patterns found in the fMRI data, and thus there exists an alignment between the patterns in the fMRI data and the neural network activations. The signal that passes through the neural network is predictive of the signal in the brain images.

However, now suppose we want to measure the brain score of a neural network that does translation—which is different from the listening comprehension task that the participants did in the fMRI. We begin by normally training the network using some training set for translation. The network optimizes its weights so as to maximize its accuracy on the test set. Once the network completes training, we'd like to evaluate it for brain-likeness. First,

we input the same words that the participants heard in the fMRI into the neural network. For instance, suppose that at one point the participant heard the following sentence: “A man walks to the store.” We group the words into pairs and convert them into embeddings. These embeddings are then fed as input to the network. This way, although the tasks may be different, we still observe the network and the brain activity on the same input, and there is a more direct comparison for applying GLM.

3.3 Utilizing computer clusters

Another challenge with training and evaluating a large collection of architectures is that it takes a lot of computation power to generate and evaluate all of them. At the very least, the models should be trained on a computer with a Graphics Processing Unit (GPU), which speeds up the large matrix multiplications by up to an order of magnitude. Yet even with a GPU, training a neural network on a large dataset is still slow. For WMT, it can takes up to a week to train even a single architecture. Naively generating and training one model after the other on a personal computer could take years to create a collection of models that is large enough for significant exploration and statistical analysis.

To help with this problem, we use remote computer clusters to generate and train the models. A remote computer cluster is comprised of a collection of computers, or nodes, that are managed by a director. A user may communicate with the director using a Secure Shell (SSH) session. If the cluster uses the Simple Linux Utility for Resource Management (SLURM) workload manager, then the user may specify the resources he requires, for exam-ple the memory and time limits. Then the director allocates a compute node which fits the

users specifications, or rejects the request if a node with those specifications do not exist. By training our models on multiple nodes at once, we may parallelize the training, and train a different architecture on each node simultaneously.

Chapter 4

Discovering Influential Sub-Architectures

Recall the progress so far: we have generated and trained a variety of neural networks, and have evaluated them for brain-likeness. The models have a distribution of scores: some are more brain-like than others. At this point, though, we are still left wondering exactly why some models perform better than others. What is it about the high-scorers that encourages better performance? Are there computations that seem responsible?

4.1 Problem statement

To illustrate the idea more concretely, consider the following toy example with three models represented in their string form: Add(x, x) which adds variable x to itself. This model has a brain-likeness score of 28. Add(x, y) which adds variable x to variable y. It has a score 29. And finally Mult(x, x), which computes an element-wise multiplication between x and itself.

Suppose it scored 12. Although each of these networks has a unique architecture overall, there are places where they overlap and have common sub-architectures. For instance, the first two both add x to another variable. By eye, it seems like the sub-architecture of adding x to another variable may encourage the models to score higher. The first two architectures include that sub-architecture, and they scored more than double the third.

But it is difficult to say with confidence. Perhaps the signal is noisy, and we don't have enough data. The architectures include more components than just the "Add" sub-architecture we are considering, so maybe the model's score is determined by some other sub-architecture. It is clear that the signal is noisy because the same sub-architecture appears in two models with different scores.

Our objective is to use the collection of models we've already generated to approximate a sub-architecture's scores despite the noise, and return the sub-architectures with the highest performance. Critically, a model's score is determined solely by its architecture, because all other hyper-parameters are fixed, and the training and test sets are the same. Thus, in essence, our objective is to discover "influentially high" (or "influentially low") computa-tions within the architectures—sub-architectures that are statistically associated with only the highest-scoring (or lowest-scoring) models. Below, we introduce the Detective Method for discovering the influential sub-architectures that appear to persuade a model's score to be among the most extreme. We call these the influential sub-architectures.

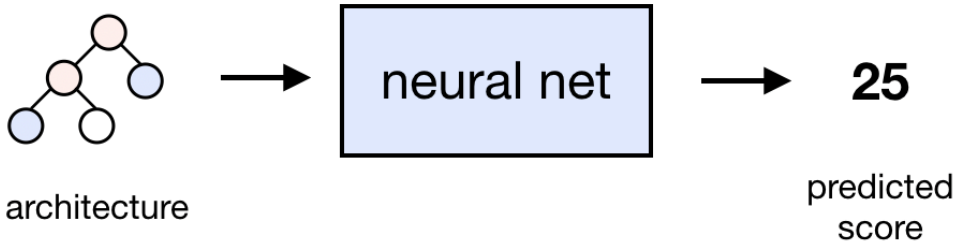

Figure 4.1: Neural network as a black-box scoring function. A neural network can be trained to predict an architecture's score. However, the scoring function it learns is not extractable or reproducible.

4.2 Objective: a method that is extractable, reproducible, and extensive

One approach for learning the influential sub-architectures is to train a neural network that predicts a model's score given its architecture as input, as shown in Figure 4.1. The network learns a scoring function, which maps from an architecture to its predicted score. This method was done by Schrimpf (Schrimpf, 2017). The problem with this approach is that even if the network ends up learning an accurate scoring function, it is difficult to extract the network's knowledge. There is no rigorous way to express what the network has learned into a formal function with variables. Furthermore, even if we could extract the scoring function from the network, we would still be left wondering how the network learned the scoring function. If we trained the network again, it is very likely that the function would change. For those reasons, the neural network approach is too much of a black-box. We would like our results to be easily extractable and reproducible.

Another more explicit approach for discovering influential operations would be to discover edges, or pairs of operations, that consistently appear in high-scoring models. This method

has a storage requirement that is linear in the number of edges, however we don't like this approach because we are interested in finding larger influential sub-architectures. We believe that important computations may happen at a scale larger than a single edge. For instance, the basic cells in the Transformer model contain four or five of our operations. Only considering edges would limit the number of operations to two, and the analysis is limited to very simple computations and would not be very extensive.

The following sub-section introduces a method for finding influential sub-architectures with inspiration from the combinatorial bandits problem and the weighted majority al-gorithm (Littlestone, 1989). The method extends the edge approach and analyzes larger computations by considering sub-architectures with multiple edges and operations, instead of single edges with two operations. Therefore, we tradeoff a larger memory requirement for a more extractable, reproducible, and extensive analysis.

4.3 The Detective Method

The method assumes that each sub-architecture has a ground-truth brain score, which is the value that an architecture scores if it includes that sub-architecture. The number of samples from which to approximate the ground-truth score is the number of models that the sub-architecture appears in. For instance, if a sub-architectures appears in ten models, then there are ten samples. The algorithm's objective is to approximate a sub-architecture's ground-truth brain score from a set of limited and noisy samples, and to return the approximations which are statically significant of being influential.

The influential sub-architectures returned by the method are useful for two main reasons.

First, they help explain which computations have higher score. The method returns the simplest sub-architectures that are statistically associated with only the highest (or lowest) scoring models. The models are trimmed to their most brain-like components, bringing the essential computations front and center. Also, because the sub-architectures are composed of fewer operations than the model as a whole, it is easier to interpret their computations for a deeper meaning. This potential is particularly interesting in the context of models with high brain scores. The second reason returned sub-architectures are useful is for defining the architecture space in the future. In particular, we may update the architecture space to include the influential sub-architectures as unit operations. That way, we bias the space to include operations that have been shown to work well, and generate higher-scoring models in future architecture samplings.

4.3.1 Method description

Like a talent detective, the method considers the performance of every candidate sub-architecture, and selects the set that it has determined are remarkable. The process begins by iterating through the models, and for each one, enumerating all of its sub-architectures. Recall the assumption that the sub-architecture's ground-truth score is the score of the model it ap-pears in. Thus, we map each sub-architecture to a list of all the model scores that the sub-architecture appears in. The list collects all of the observed brain scores for that sub-architecture. See Figure 4.2.

Now that the data are organized into a map from sub-architectures to lists of scores, we must approximate a brain score from each list and test it for statistical significance. We

approximate sub-architecture's i ground-truth score by the mean of its list, μ_i. We require that lists contain at least n samples, and use a t-test to test whether the sub-architecture is influential. In particular, we test two hypotheses for each i: first, whether μ_i is in the top p% of scorers (whether it's influentially high). And second, whether μ_i is in the bottom p%of scorers (whether it's influentially low). The value of p sets a threshold score for a model to be considered influential.

Because we must test two hypotheses for each sub-architecture, it is dangerous to use an unadjusted p-value. The results would likely include many false positives. In order to reduce the number of false positives, we use the Holm-Bonferroni correction. If the adjusted p-value is smaller than $\alpha = 0.05$, the corresponding sub-architecture is influential, and we keep it for further evaluation.

Finally, because we enumerate all possible sub-architectures, it is likely that many of the influential sub-architectures are themselves sub-architectures of each other. For example, suppose Add(x, x) and Add(x,) are both influential sub-architectures, where " " indicates that the position may be filled by any variable. Notice that Add(x, x) is just a special case of Add(x,). If Add(x,) is influential, then Add(x, x) is also influential. Notice that it is stronger to know that Add(x,) is influential than it is to know Add(x, x) is influential. To return the influential sub-architectures in their simplest and strongest form, we organize them into two trees: one for the influentially high and the other for the influentially low sub-architectures. Each node in the tree is an influential sub-architecture, and there is an edge from sub-architecture$_1$ to sub-architecture$_2$ if sub-architecture$_2$ is a sub-architecture of sub-architecture$_1$. The leaves of these tree are the set of sub-architectures we return.

Let's see how the Detective Method works on the very simple example in Figure 4.2. In the

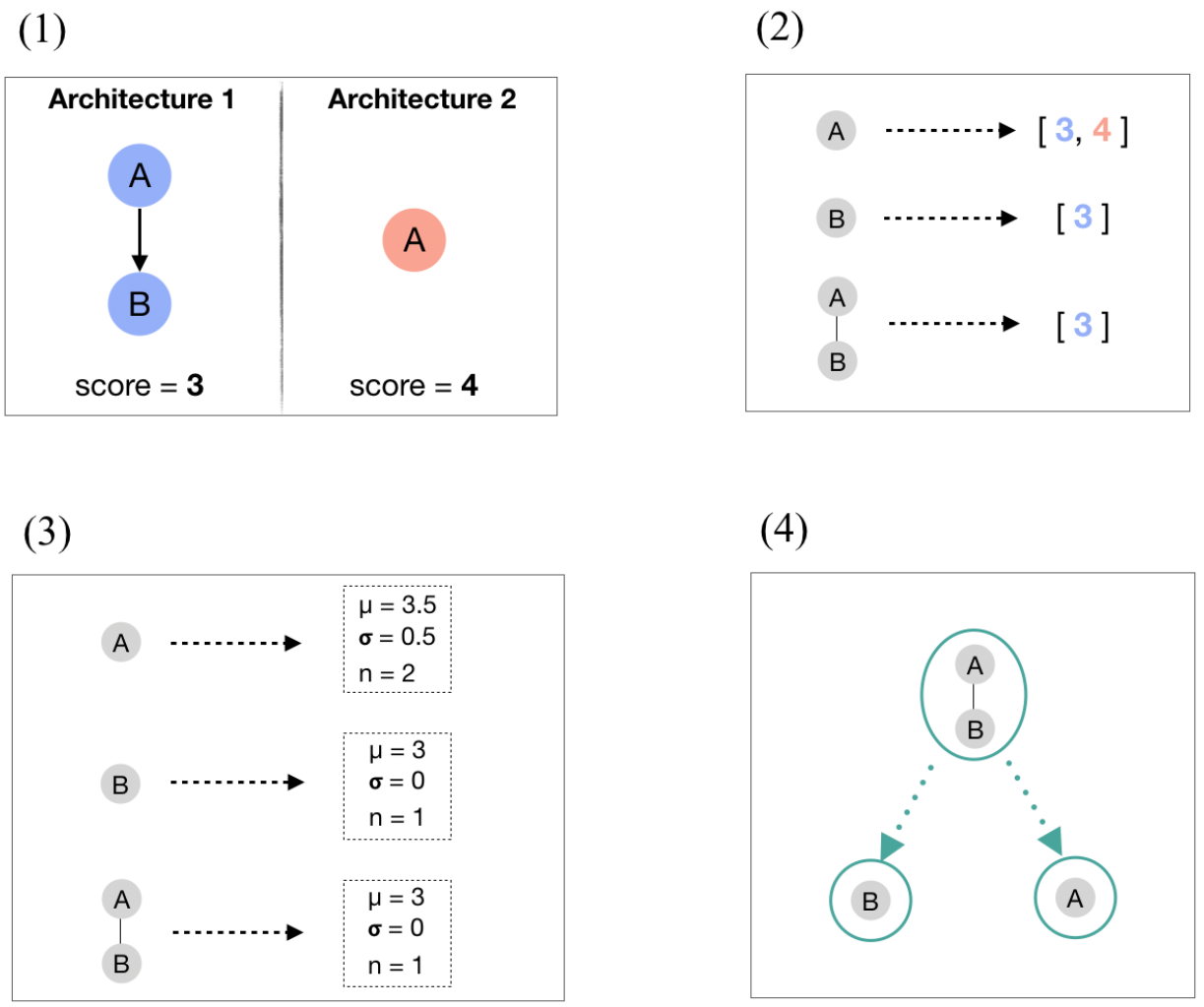

Figure 4.2: The Detective Method on a toy example. (1) Depicts the set of scored architectures from a random sampling. (2) Shows the map from enumerated sub-architectures to lists of the model scores they appear in. (3) Calculates the lists' means μ_i, standard deviations σ_i and lengths n_i, and tests for influential sub-architectures. (4) Organizes influential sub-architectures into a tree. Child nodes are sub-architectures of their parent.

top left corner, diagram 1 shows a collection of architectures and their corresponding scores on a given task. Architecture 1 scored 3, and has two operations: A and B. Architecture 2 scored 4, and only has operation A. Next, in diagram 2, we enumerate every sub-architecture in the collection. Architecture 1 has three sub-architectures: A, B, and AB, while Architecture 2 only has A. Then, each sub-architecture is mapped to a list of every architecture score that the sub-architecture appears in. In particular, A appears in Architectures 1 and 2, and therefore is mapped to scores 3 and 4. Similarly, sub-architecture B only appears in Architecture 1, and so is mapped just to Architecture 1's score. Next, in diagram 3 we calculate the means, standard deviations, and lengths of the lists, and perform a t-test using the Holm-Bonferroni correction to check which sub-architectures are high-performing with statistical significance. Suppose all three are statistically significant, and therefore they are all influential. Finally, the influential sub-architectures are arranged into a tree, where child nodes are sub-architectures of their parents.

4.3.2 Space requirements

As previously argued, we trade off a more thorough search of influential sub-architectures for a larger storage requirement: it is linear in the number of architectures in the collection, but exponential in the size of each architecture. In order to approximate the expected number of sub-architectures T that the method generates, we may think of the architecture as a tree, and then count the total number of contiguous subtrees it has. Let the leaf be on level 0 of the tree, the leaves' parents are on level 1, and so on (see Figure 4.3).

Suppose the architectures in the collection have an average of n nodes, where each node

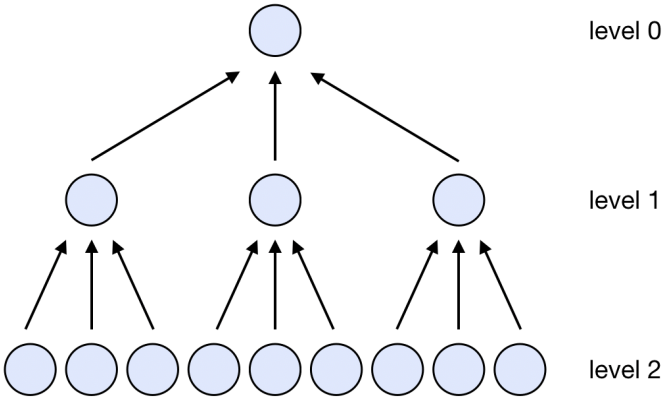

Figure 4.3: An average architecture for memory calculations, with b = 3 and n = 13. The leaf is on level 0.

has an average branching factor of b. Let t_d be the number of subtrees that are rooted at a node on level d of the graph. In expectation, there are b^d nodes on each level. We may choose to either include or exclude each node, but we may not exclude all of them because the subtrees must be contiguous. Thus, we may solve for t_d recursively as,

$$t_d = (2^b - 1)t_{d+1}.$$

The base case is for leaf nodes: $t_{leaf} = t_{\log_b(n)} = 1$, since a leaf consists of only a single node. Notice that t_d solves explicitly as $t_d = (2^b - 1)^{\log_b(n)-d}$. Since there are b^d nodes on level d, the expected number of contiguous subtrees from one architecture is

$$T = \sum_{d=0}^{\log_b(n)} b^d t_d = \sum_{d=0}^{\log_b(n)} b^d (2^b - 1)^{\log_b(n)-d}.$$

Multiplying T by the number of architectures in the collection gives an upper bound for the expected number of sub-architectures that the method considers. Because sub-architectures repeat across different models, the estimate here is an upper bound.

4.3.3 Case study

The purpose of this case study is to show the type of results the Detective Method returns. Because the brain scoring function is still being implemented at the time of writing this, the following example is trained for translation on the Multi30k dataset and scored with the BLEU score. Multki30k contains 31,014 English sentences and their corresponding German translations and is an extension of the Flickr30k dataset.

The architecture space is defined by the same set of operations defined in Chapter 3:{layer norm, reLU, tanh, softmax, add, mult, MM, scale, gate3}. We generated a collection of 4,344 models trained on Multi30k, and evaluated their performance using the BLEU score. The average model was composed of 20.1 operations, with the smallest having six operations and the largest having 115 operations. The average BLEU score was 17.8, where the worst score was 0 and the best score was 36.73.

Next, we performed the Detective Method on these architectures. The threshold percentile p for a sub-architecture to be influential was set to 80: a sub-architecture's approximated score must be in the top 20% for it to be influential. We may increase the score threshold percentile p to only include higher-scoring sub-architectures, or decrease it to relax the cutoff and consider more of them.

The method considered a total of 13,169,097 unique candidate sub-architectures. Of those, 892 are influentially high-scoring, and 47 are influentially low-scoring. Recall that many of these influential sub-architectures are themselves sub-architectures of each other. Thus, after organizing them into two trees, the method returns 21 influentially high-scoring, and 10 influentially low-scoring sub-architectures. This is a substantial pruning from the

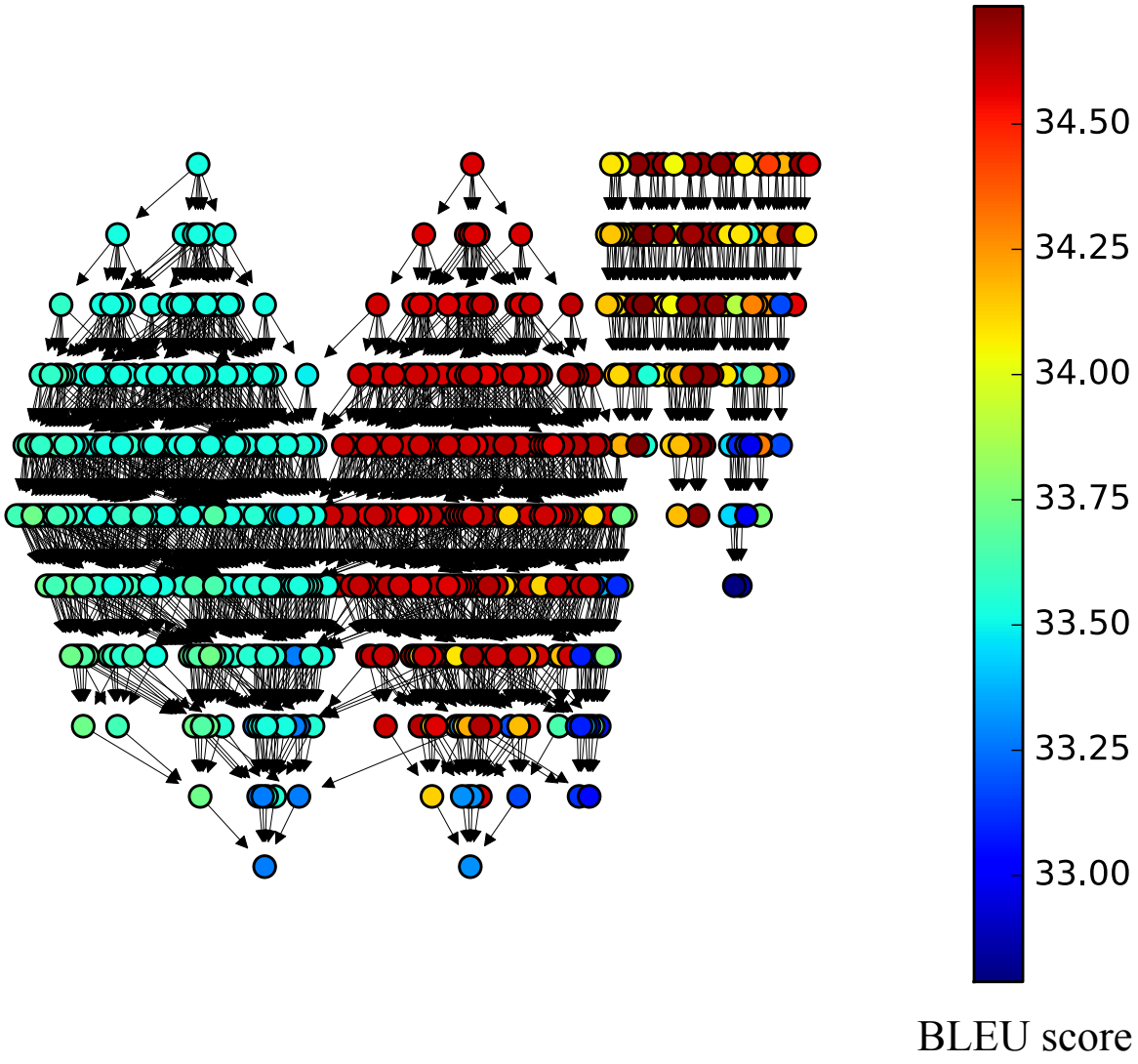

Figure 4.4: Influentially good sub-architecture tree. Each node is an influentially good sub-architecture from the case study. The node's color is the sub-architecture's approximated BLEU score, and the leaves are the influential sub-architectures in their simplest form.

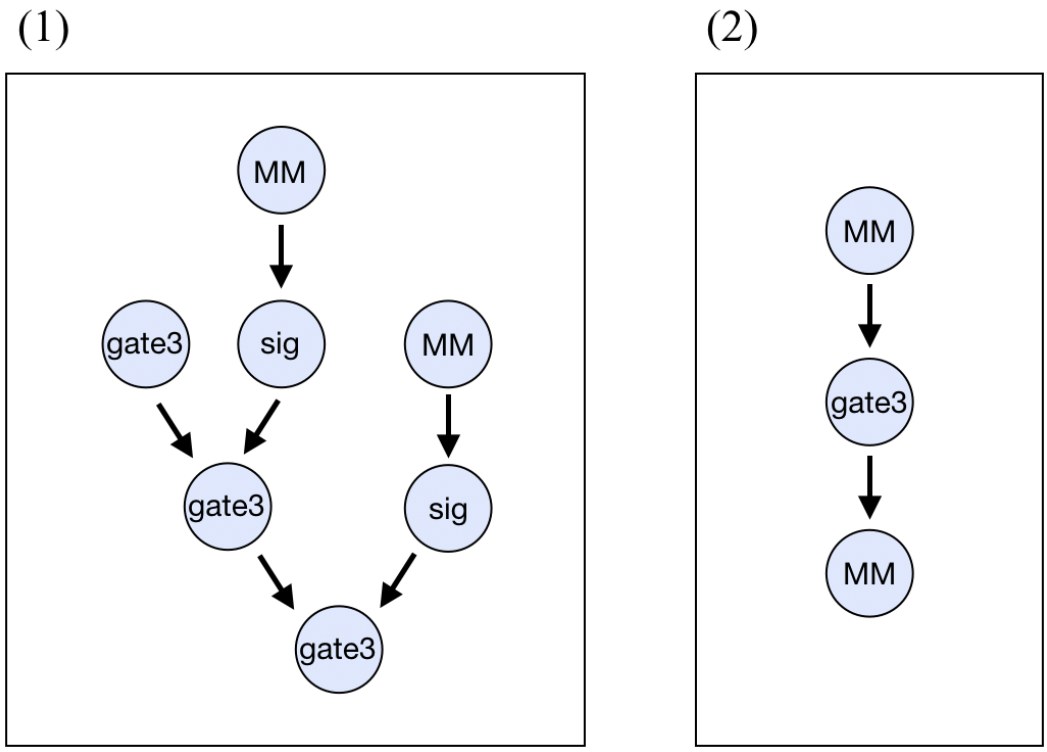

Figure 4.5: Examples of influential sub-architectures from the case study. (1) An influentially good and (2) an influentially bad sub-architecture returned on the Multi30k case study.

more than 13.1 million original candidates.

The influentially good sub-architecture tree is shown in Figure 4.4. Each node is an influential sub-architecture, where the node's color is the estimated ground-truth score com-puted by the Detective Method. While the method only returns the simplest influential sub-architectures, which are the leaves of the tree, the tree shows all of the influentially good sub-architectures. This representation depicts how adding an operation to a sub-architecture affects its performance.

Working our way up the tree, we see the affect that different operations have on a sub-architecture's performance. Notice that in general, the sub-architectures toward the top of the tree have higher scores than those toward the bottom. This makes sense because the

larger sub-architectures can compute more complicated and nuanced operations that are more specifically optimized for the domain. A few of the influential sub-architectures are shown in Figure 4.5. It appears useful to consecutively stack gate3s, and have MM as an input to the sigmoid. Conversely, having an MM after a gate3 results in weak performance.

Chapter 5

Contributions

Mimicking human language processing via neural networks can help explain how our brains process language, as well as help create more accurate and natural AI for NLP. Although it would be wildly exciting in its own right to discover a neural network with a high brain score, it would be a disservice to merely make the discovery. In order to continue advancing AI and our understanding of how the brain processes language, we must recognize why the model is brain-like—what are the computations that make it so special? This deeper understanding would be an indispensable source of inspiration, both for designing a better AI and for hypothesizing about the brain.

The method presented here describes a way to generate and evaluate NLP models for brain-likeness, and then pinpoint with statistical significance the sub-architectures that ap-pear in high-scoring models. Now, instead of just evaluating existing NLP models, the method discovers new brain-like architectures by performing a random architecture sam-pling. By using the Detective Method, we no longer rely on a black box scoring function de-fined through a neural network to find the influential sub-architectures, which represent the

brain-like computations in their simplest form. Not only are these smaller sub-architectures distilled down to their most brain-like components, but their simpler computations are also easier to interpret than the architecture as a whole. Being able to extract the most brain-like computations with statistical rigor can help inspire the computations that the brain performs.

Additionally, the method's influential sub-architectures can help refine the architecture space from which to sample in the future. For instance, we can define an influential sub-architecture to be a single operation in order to bias more models to include it. Along similar lines, the method may test the effectiveness of a new operation. In particular, if the operation only appears in low-scoring influential sub-architectures, then we may not want to include it in the future. Refining the search space with better operations will generate more accurate models.

Bibliography

Asano E, Juhasz C, Shah A, Muzik O, Chugani DC, Shah J, Sood S, Chugani HT (2005) Origin and propagation of epileptic spasms delineated on electrocorticography. Epilepsia 46:10861097.

Braitenberg V (1992) Corticonics: neural circuits of the cerebral cortex. Trends Neurosci 15(4):156157.

Bello I, Zoph B, Vasudevan V, Le QV. (2017). Neural optimizer search with reinforcement learning.CoRR, abs/1709.07417.

Chater N, Manning C.D. (2006). Probabilistic models of language processing and acquisition. Trends in Cognitive Sciences.

Chomsky, N (1965). Aspects of the Theory of Syntax. MIT Press, Massachusetts.

Devlin J, Chang M, Lee K, Toutanova K.BERT: pre-training of deep bidirectional transformers for language under-standing.CoRR, abs/1810.04805, 2018.

Google (2019) cloud.google.com/compute/docs/regions-zones/. Accessed: 2019-02-25.

Hale, J. (2001). A probabilistic Earley parser as a psycholinguistic model. Pittsburgh, Pennsylvania.

Jain S, Huth A. (2018). Incorporating context into language encoding models for fmri. S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems Vol. 31, pages 66296638. Curran Associates, Inc.

Levy, R. (2008). Expectation-based syntactic comprehension. Cognition, 106(3), 11261177.

Licata G (2012). Fuzzy Logic, Knowledge and Natural Language, Fuzzy Inference System, Mohammad Fazle Azeem, IntechOpen.

Littlestone, N., Warmuth, M. (1989). Weighted Majority Algorithm. IEEE Symposium on Foundations of Computer Science.

Izhikevich EM. (2006). Polychronization: computation with spikes, Neural Comput , Vol. 18 (pg. 245-282).

Kell AJE, Yamins DLK, Shook EN, Norman-Haignere SV, McDermott JH. (2018). A Task-Optimized Neural Network Replicates Human Auditory Behavior, Predicts Brain Responses, and Reveals a Cortical Processing Hierarchy. Neuron 2018; 98: 630644. pmid:29681533.

Schrimpf M, Merity S, Bradbury J, Socher R. (2017). A flexible approach to automated RNN architecture generation. CoRR, abs/1712.07316.

Tenenbaum J, Kemp C, Griffiths T, Goodman N (2011). How to Grow a Mind: Statistics, Structure, and Abstraction. Science, Vol. 331, 2011

Toulasm D. (2009) The importance of functional mri (fmri) in the neurosurgical strategy in brain tumors.

Mikolov T, Sutskever I, Chen K, Corrado G, Dean J. (2013). Distributed representations of words and phrases and their compositionality. Advances in Neural Information Processing Systems, 26, 10 2013.

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A. N., Kaiser, L, Polosukhin I (2017). Attention is all you need. In the Annual Conference on Neural Information Processing Systems (NIPS).

Yamins D, Hong H, Cadieu C, Solomon E, Seibert D, DiCarlo J. (2014). Performance-optimized hierarchical models predict neural responses in higher visual cortex. Proceedings of the National Academy of Sciences, 111(23):86198624.

Zhuang C, Kubilius J, Hartmann M, Yamins D. (2017). Toward goal-driven neural network models for the rodent whisker-trigeminal system. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 25552565. Curran Associates, Inc.