Improving Controlled Text Generation via Neuron-level Control Codes

Jay Orten[1], Nancy Fulda[1]
[1]*Brigham Young University, Provo, Utah*
{jo288, nfulda}@cs.byu.edu

Keywords: Language Modeling, Statistical and Machine Learning Methods, Semi-supervised, weakly-supervised and unsupervised learning

Abstract: Task-specific text generation is a highly desired feature for language models, as it allows the production of text completions that are either broadly or subtly aligned with specific objectives. By design, many neural networks switch between multiple behaviors during inference - for example, when selecting a target language in many-to-many translation systems. Such task-specific information is usually presented to the network as an augmentation of its input data. In this work, we explore an alternate approach: transmitting task information directly to each neuron in the network. This removes the need for task information to propagate forward during training, a particularly critical advantage in low-resource settings where maximum benefit must be extracted from each training example. To test this approach, we train over 160 language models from scratch with a large variety of architectures and configurations. Our results show that models with neuron-level augmentation can experience increased learning speed, improved final generation accuracy, and even novel learning capabilities, with larger benefits as network depth increases.

1 INTRODUCTION

Deep neural networks are capable of learning rich feature spaces containing complex information of relevance to many tasks. For this reason, it is often more efficient to train a single model to perform many related tasks than it would be to train a model for each task in isolation. For example, systems such as MultiModel Kaiser et al. (2017) take full advantage of this technique by training one model on a variety of different domains.

Given a broad task range, it is often desirable to have the ability to guide model output according to the objectives of the user. This is typically accomplished by the inclusion of additional input information. The CTRL language model Keskar et al. (2019) accomplishes controlled generation by appending task-specific tokens to the beginning of input prompts. A related method is employed by Meta's M2M-100 model, in which target language data is added as an additional token of the decoder rather than the encoder Fan et al. (2021). Similar to these approaches is the method used implicitly by large-scale text generation models such as GPT-3 and GPT-4 Brown et al. (2020); OpenAI (2023), which rely completely on prompt engineering to perform a wide variety of unique tasks. The goal with all of these models is to enable effective and controllable text generation across various tasks with a single model.

The referenced methods for controlled generation are effective in part because many deep learning systems leverage residual connections to allow more efficient transmission of feature representations between layers He et al. (2015), meaning that task information has the ability propagate throughout the entire network. We theorize that task-specific behaviors in neural networks with generalized internal feature representations could be learned more quickly and more successfully by passing task-specific information directly to the interior layers of the network. To test this theory, we connect a task embedding vector directly to the neurons in the network's hidden layers. Proven effective, this approach would boost efficient training of smaller models while retaining nuanced control over text generation.

The core contributions of this work are: (1) a framework for applying control codes at the neuron level on both small-scale Transformer networks and simple gated recurrent unit (GRU) recurrent neural networks, (2) a structured analysis of this augmentation obtained by training over 160 models from scratch with a variety of data sets and configurations, and (3) the discovery that, within certain constraints on relative depth of the network, the neuron-level control codes can significantly improve learning speed, increase final performance, and even enable new learning capabilities.

2 Background

Language model architectures: Language modeling can be described as a task whereby the next token in a sequence is repeatedly predicted. Well-known architectures for language modeling include Recurrent Neural Networks (RNNs) and Transformer networks.

RNNs operate on sequential data via the combination of the current input vector with a hidden state representing salient information from all previous inputs. Formally,

$$h_t = \mathrm{RU}(x_t, h_{t-1})$$
$$\hat{y}_t = \mathrm{softmax}(W_o \cdot h_t)$$

where h_t represents the hidden state, $\hat{y}_t$ the output of the network, x_t the input of the network, and t the time step. The matrix W_o is a final linear layer. The components of a recurrent unit (RU) function can vary. Popular implementations include Long Short-Term Memory (LSTM) Hochreiter and Schmidhuber (1997) and Gated Recurrent Units (GRU) Cho et al. (2014). RNNs may utilize multiple hidden layers.

Recently, auto-regressive decoder-only Transformer networks have become popular for novel text generation tasks. The Transformer architecture consists of sequential layers, each containing a multi-head attention block followed by a feed-forward network block, as described by Vaswani et al. (2017). Prominent language models based on the transformer architecture include GPT-2 Radford et al. (2019), GPT-3 Brown et al. (2020), PaLM Anil et al. (2023), LLaMA Touvron et al. (2023), and Facebook's M2M-100 machine translation model Fan et al. (2021). These models, although powerful, require extraordinary compute resources to train. Our work seeks to reduce the time and energy consumption required to train such models for multi-task frameworks.

Controllable text generation: Vanilla language models function as next-word prediction tasks, where the probability of the next token x is determined by all previous tokens as per the chain rule of probability:

$$p(x) = \prod_{t=1}^{n} p(x_t | x_{<t})$$

Conditional language models invoke additional conditioning on some context c:

$$p(x|c) = \prod_{t=1}^{n} p(x_t | x_{<t}, c)$$

The conditioning context c represents additional information upon which generation is conditioned. Commonly, c is implemented as an additional token in the input prompt. Keskar et al. (2019) refers to this token as a control code, and explored a variety of novel approaches to using control codes, such as using web-page links as codes or mixing codes in order to generate cross-over behavior.

The usage of control codes as a conditioning context is common in machine translation, specifically for multilingual translation with a single model Ha et al. (2016). Kobus et al. (2017) accomplished neural machine translation across multiple domains by appending an additional token representing domain to the end of input sentences. Additionally, their method concatenates a special domain embedding to each word embedding. Most notably, Johnson et al. (2017) achieved remarkable multilingual zero-shot translation by introducing a token signifying the target language to the beginning of each input sentence. Caswell et al. (2019) further uses this technique to achieve competitive results on back-translation tasks.

The conditional context c may also be interpreted as a vector of information. For example, Ficler and Goldberg (2017) achieved controllable generation by simultaneously conditioning on multiple parameters involving stylistic properties. This was accomplished by creating a conditional vector consisting of multiple embedding vectors concatenated together. Sennrich et al. (2016) controlled the level of politeness in generations by utilizing what they term 'side constraints' appended to the end of the source text.

A primary contribution of models such as GPT-2 or GPT-3 is that increased model sizes and huge amounts of data allow models to implicitly learn controlled generation Radford et al. (2019); Brown et al. (2020). Because these models are very effective few-shot learners, controlled generation can be accomplished via prompt-engineering alone. However, because an explicit context c is absent, it is difficult to control for specific desired attributes during training and usage.

More recently, Plug and Play Language Models Dathathri et al. (2020) enable controlled text generation for any pre-trained language model via additional attribute classifier models. While this approach requires no modification to the base model, it is intended for use on very large pre-trained language models, and is not suited for low-resource settings where efficient training from scratch on specific tasks is highly desirable.

3 Methodology

This study seeks to understand the impact of neuron-level control codes on model performance. As such, our goal is not to achieve flawless text generation, but rather to compare learning speed and final text quality across many model sizes and architectures. An application of these ideas to state-of-the-art language models is left for future work.

Our core contribution, and the foundation for this research, is the idea that models can learn more quickly if each neuron has direct access to information about the current text-generation task. To enable this, we propose an alternative architecture for conditioned language models that distributes task information throughout the entire network. We compare this alternative method to a default architecture and report results in Section 5 below.

3.1 Definition of Control Codes

We define the conditional context as a control code, c, represented by a special token of a unique form, e.g. '⟨shakespeare⟩'. This token is embedded as an n-dimensional vector, as with all other tokens in a given prompt. Most conditional language models simply include c as an additional token in the prompt; our approach, however, utilizes the embedded vector of c at each linear layer throughout the entire network.

3.2 Applying Neuron-level Control Codes in RNNs

With RNNs, the default method for controlled generation involves concatenating c with each input token x_t in the sequence (See Figure 1). This is a known method for controlled generation Ficler and Goldberg (2017). Where n is the number of layers in the RNN:

$$h_t^n = \mathrm{RU}((x_t \oplus c), h_{t-1}^n) \text{ for } n = 1$$
$$h_t^n = \mathrm{RU}(h_t^{n-1}, h_{t-1}^n) \text{ for } n > 1$$
$$\hat{y}_t = \mathrm{softmax}(W_o \cdot h_t^n)$$

Our alternative architecture, as proposed in this paper, is intended to allow the control information to directly influence every weight in the network, enabling control information to more directly influence gradient changes and improve training. This is accomplished by concatenating c to the input of every RNN cell in each layer, rather than just before the first layer of RNN cells:

$$h_t^n = \mathrm{RU}((h_t^{n-1} \oplus c), h_{t-1}^n) \text{ for } n > 1$$

The control information is also concatenated to the final output from the last layer, before it is passed through a final fully connected feed-forward layer:

$$\hat{y}_t = \mathrm{softmax}(W_o \cdot (h_t^n \oplus c))$$

This is done for each point in the sequence (See Fig. 1).

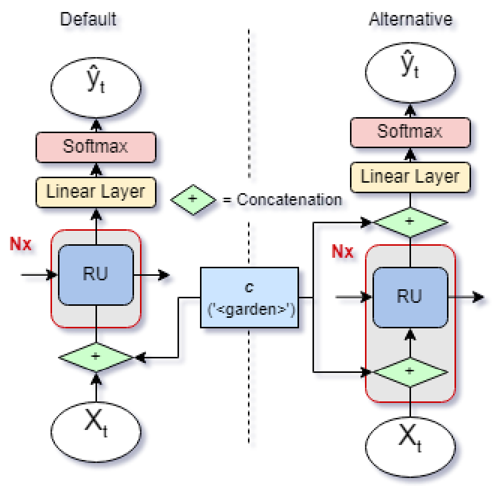


Figure 1: Methods for controlled generation with multi-layer RNNs. The alternative approach involves concatenation of a special vector before every layer in the RNN, not just the first.

3.3 Applying Neuron-level Control Codes in Transformers

The default approach for controlled generation in Transformers is achieved by concatenating the embedded vectors for both the input sequence and the embedded control code c, such that the embedded control token information is represented at the beginning of each sequence in a batch. This combined vector is then passed through positional encoding as described in Vaswani et al. (2017).

To apply the control codes to every neuron, we take advantage of the position-wise fully connected feed-forward network that follows the self-attention block within a Transformer decoder layer:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

This block consists of two linear transformations with a ReLU activation in between.

In this approach, our control token vector, c, is not concatenated with the input sequence. Rather, after the attention block, c is concatenated before each layer in the feed-forward block. Thus, we are directly

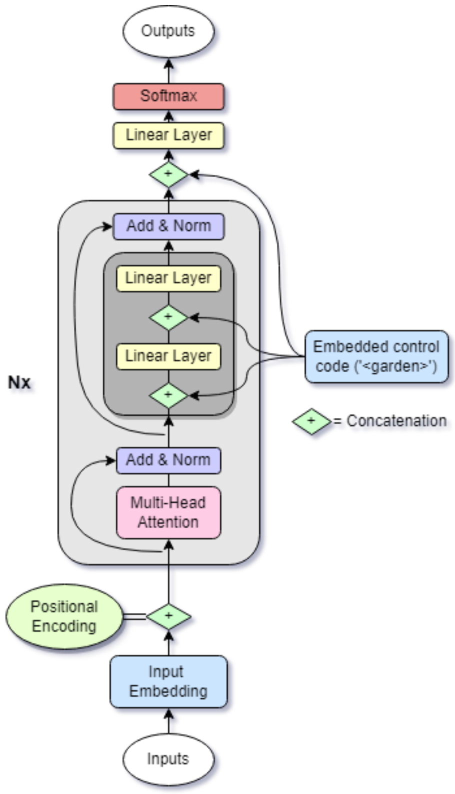


Figure 2: Alternative Transformer architecture. The control information is concatenated at key points within the linear layers of every decoder cell.

augmenting each point in the linear layers with control information:

$$\mathrm{FFN}(x) = (\max(0, (x \oplus c)W_1 + b_1) \oplus c)W_2 + b_2$$

Finally, c is concatenated to the output of the final decoder cell before it is passed through a final linear layer, which decodes to output (see Fig. 2).

In this manner, control information is directly fed to all weights except those in multi-head attention. Efforts to integrate control information in self-attention, such as token-wise concatenation before applying self-attention, led to a significant performance drop.

Our method differs from the default method in that we concatenate the control vector with each sequence element's vector space, injecting control information throughout the entire sequence rather than just at the start. This approach accounts for the Transformer decoder's batch processing of entire sequences. In contrast, the default control architecture prepends the control vector solely at the sequence's outset, like adding a token to a prompt's beginning. We theorize that this difference enables our network to much more directly receive and incorporate control information during training and inference, providing a significant benefit.

4 Experimental Setup

All models were trained on a single 11 GB NVIDIA GeForce GTX 1080 Ti, running on a Linux operating system. Implementation of models was accomplished using PyTorch 2.0 and respective libraries. All models were trained with a batch size of 16 and a sequence length of 256.

4.1 Data Sets

In order to establish more robust conclusions, various data sets were used to introduce variety to the experimental trials. Each data set contains unique text styles. It is intended that this will test the capabilities of the models to generate text in different domains. Data sets chosen are relatively small, but contain controlled content for testing conditional generation.

All experiments were done on data sets consisting of at least two topics, or sources. Every epoch, a model would be trained on all data from each source in the specific data set. Special tokens such as '⟨music⟩' and '⟨garden⟩' were used to differentiate between the different sources. At evaluation time, controlled generation for the desired generation type was accomplished via the corresponding control token.

RNN architectures were trained on two small data sets extracted from Project Gutenberg Project Gutenberg (2023), one monolingual and one bilingual. Transformer-based architectures were trained and tested using five data sets with varying sizes and content styles. See Tables 1 and 2. These data sets are relatively small compared to benchmark data sets for large language models. Using these smaller data sets allowed us faster training time and more flexibility in testing.

4.2 RNN Experimental Setup

Both default and alternative RNN architectures were trained on identical setups for direct comparison against each other. RNNs consisted of three layers of GRU cells. It is assumed that a long short-term memory (LSTM) architecture would behave similarly. Hidden sizes of 256, 512, and 1024 were tested. All networks were trained on the 'Books' data set for 50 epochs. Models were trained on the 'Languages' data set for 150 epochs. Cross-entropy loss was used as the loss function, and Adam as the optimizer. A learning rate of 0.001 was used.

Data set	Description	Tokens
Languages	Combined full text of two literary sources, one in English and one in Tagalog. *The Great Gatsby* by F. Scott Fitzgerald and *Bulalakaw ng Pág-Asa* by Ismael A. Amado Project Gutenberg (2023).	89,564
Books	Combined full text of two literary sources, both in English. *The Great Gatsby* by F. Scott Fitzgerald and a collection of works from Shakespeare Project Gutenberg (2023).	116,232

Table 1: Data sets used to train RNNs

Data set	Description	Tokens
Books-2	Combined full text of two literary sources, both in English. *The Great Gatsby* by F. Scott Fitzgerald and a collection of works from Shakespeare Project Gutenberg (2023).	116,232
Books-3	Combined full text of three literary sources, all in English. All sources used in Books-2, with the addition of *A Tale of Two Cities* by Charles Dickens Project Gutenberg (2023).	276,985
Books-6	Combined full text of six literary sources, all in English. All sources used in Books-3, with the addition of *Alice in Wonderland* by Lewis Carroll, *The Iliad* by Homer, and *Moby Dick* by Herman Melville Project Gutenberg (2023).	780,658
Reviews	English Amazon reviews for two different kinds of topics: outdoor equipment and music Ni et al. (2019).	1,183,235
Scripts	English text from two differently styled sources: news articles and blog posts. News articles from BBC Business Greene and Cunningham (2006), and blog posts from Blog Authorship Corpus J. Schler and Pennebaker (2006).	115,286

Table 2: Data sets used to train Transformer-based models

4.3 Transformer Experimental Setup

All Transformer models were trained with 2 attention heads and an embedding dimension of 200, both arbitrary decisions. Hidden sizes of 128, 256, 512, and 1024 were tested. Layer sizes of 2, 4, 6, and 8 were tested. The purpose of this was to explore how model size may effect results for both architectures. Consequently, 160 models were trained, each with a different architecture, data set, hidden size, and number of layers.

A learning rate initializing at 5 was used for all training runs. A learning rate scheduler was used with a gamma of 0.95 every epoch. Cross-entropy loss was used as the loss function, and stochastic gradient descent as the optimizer.

Models were trained for 50 epochs, regardless of configuration. While 50 epochs is a relatively short training time, it allowed us to explore a large variety of models.

4.4 Evaluation

A variety of methods were used for evaluation. Loss was the only metric used to evaluate RNNs, as RNNs were simply used for preliminary tests. In contrast, the Transformer models were evaluated using loss, perplexity, BERT score Zhang et al. (2020), a simple Variance metric, and a Degeneracy metric. The BLEU score Papineni et al. (2002) was also calculated, using SacreBLEU Post (2018). In practice, BLEU scores did not display any meaningful trends during the short training time.

BERTScores were calculated using baseline rescaling, as suggested by its creators, resulting in mostly negative values for many generations. This is to be expected considering the quality of initial text generations and the nature of baseline rescaling for BERTScore Hanna and Bojar (2021).

A simple variation metric was used to monitor the generic fluency a model may produce. Variation was calculated by dividing the number of unique tokens in a prediction by the number of total tokens in a prediction, averaged across *n* predictions. A higher variance score indicates higher vocabulary variation. In testing, this simple metric showed correlation with generational fluency in testing.

Degeneracy represents the average frequency of the most common word across *n* generations; lower is better.

Additionally, test generations were recorded every epoch in order to observe generational quality from a human standpoint. Predictions were selected through greedy sampling of the token that was given the highest probability in the sampling distribution.

5 Experimental Results

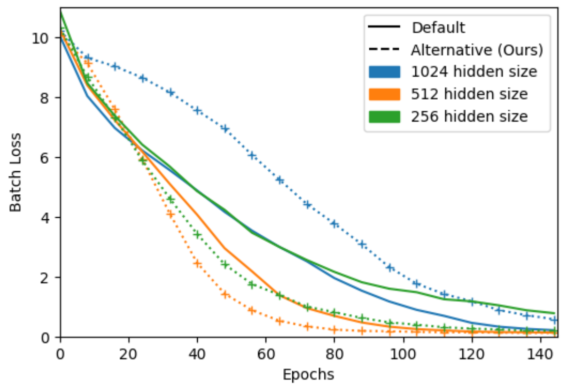


Figure 3: Loss from RNNs with various hidden layer sizes on 'Languages' data set. For smaller hidden sizes, the alternative models outperformed the default models. Results on 'Books' data set are similar.

5.1 RNN Results

Initial RNN tests indicated that the alternative architecture may offer training advantages. As shown in Figure 3, the alternative RNN achieves a lower loss faster than the default model for smaller hidden sizes of 512 and 256. In most cases, both model types converge to the same loss value. As hidden size increases, the alternative method resulted in poorer performance compared to the default. For larger hidden sizes, such as 1024, the alternative models do not learn as quickly.

These results may indicate that the alternative architecture could provide a speedup in training, but not necessarily a final advantage, over the default architecture. This only occurs, however, for small network sizes.

5.2 Transformer Results

Experiments with Transformer models demonstrate that, on average, across various data sets and model sizes, the alternative model outperforms the corresponding default model (Figure 4). Interestingly, in contrast to results for RNNs, the alternative method becomes more effective over the default method as model size increases.

- For larger networks, the alternative architecture would very often find 'breakthroughs', resulting in dramatic boosts in training, where the default architecture would not. See Figure 5.
- There was never an instance where the default architecture learned and the alternative did not. In these tests, if a model did not learn, the variance would remain around .5, indicating that the final generation quality is equivalent to the initial generation quality: unintelligible and repetitive (Table 4). While there were instances for large networks where the default architecture would not learn while the alternative architecture would, the opposite never occurred. This suggests that the alternative method only ever acted as an enhancement, and never as a complete detriment to learning.
- The difference in performance between default and alternative models grew more pronounced as

Model	Epoch	Bleu	BERTScore	Deg.
Default	1	**0.52**	**-0.28**	4.1
	5	**0.48**	-0.29	**4.1**
	10	**0.58**	-0.28	4.09
	25	**0.52**	-0.28	4.08
	50	**0.46**	-0.29	4.09
Altern. (ours)	1	0.45	-0.34	**3.91**
	5	0.43	**-0.28**	5.15
	10	0.48	**-0.15**	**2.93**
	25	0.42	**-0.15**	**2.73**
	50	0.45	**-0.14**	**2.79**

Table 3: Averaged Bleu, BERTScore, and Degeneracy scores from five different tests with different initialization seeds for default and alternative architectures (ours), across 50 epochs. Bold represents better score. The alternative method shows improvement over the default in BERTScore and Degeneracy, but not in BLEU.

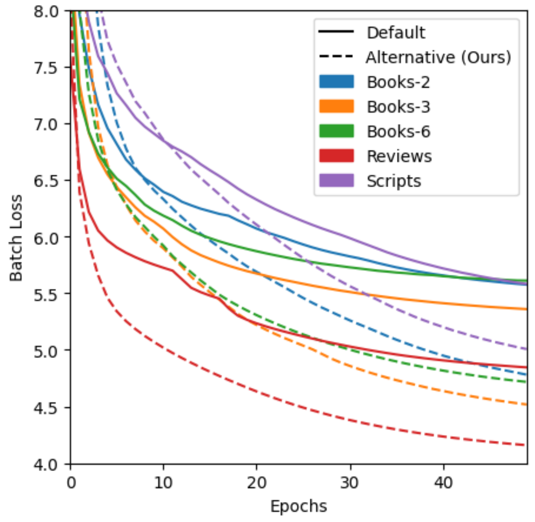
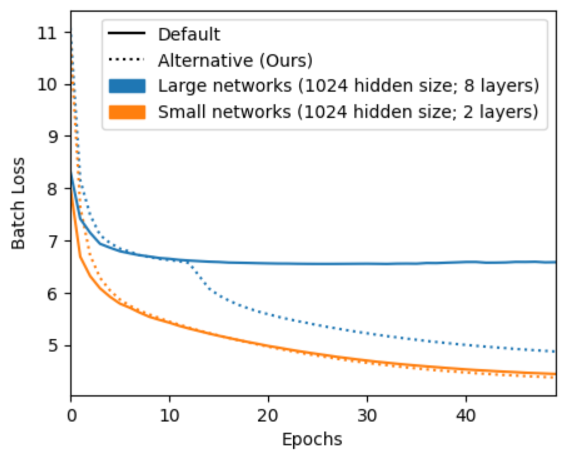
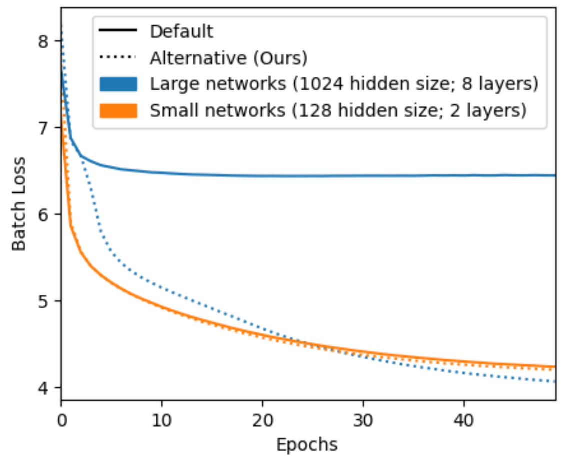


Figure 4: Average loss of all Transformer model sizes per data set. All data sets tested achieved similar results: on average, across all model sizes, the alternative architecture reached a lower loss value.



(a)



(b)

Figure 5: Batch loss on large and small networks for 'Books-6' data set (a) and 'Reviews' data set (b). The alternative architecture with larger models would achieve similar performance to the models with only 2 layers, where the default architecture would simply not learn, indicating that the alternative method benefits training for large models.

layer count increased. As seen in Figure 6, there is an increasing gap between default and alternative method performance as the models go from 4 to 6 layers, with an early learning advantage of 4 to 10 epochs. This may show that a primary benefit of the alternative method is a speedup in training.

- For small networks with a hidden size of 128 and 2 layers, the default architecture generally learned slightly faster than the alternative. However, both networks would arrive at the same level of loss, variation, and perplexity. This suggests that, for small networks, the alternative approach slows down learning slightly.
- Large networks consisting of 8 layers would

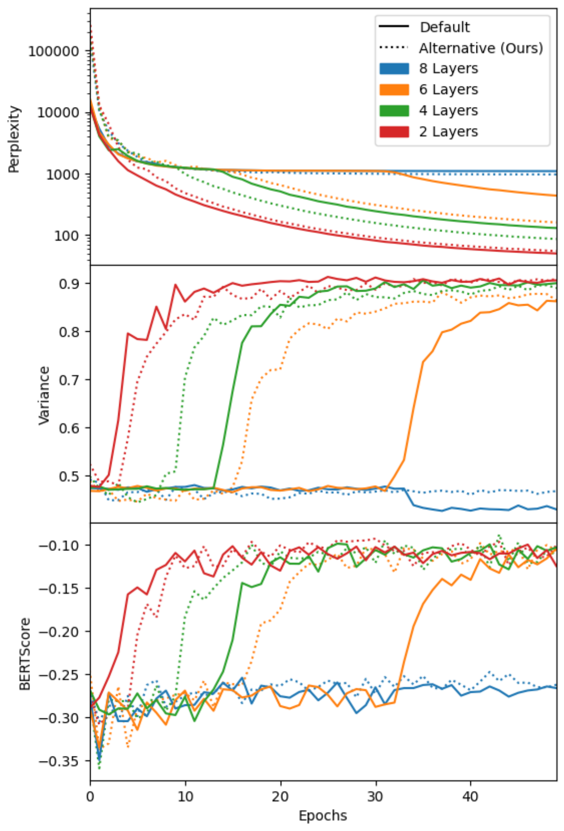


Figure 6: Perplexity, variance, and BERTScore for different model sizes on 'Scripts' dataset. Larger models using the alternative method achieve a boost in performance much sooner than their default counterparts, indicating that the alternative method enables faster learning for larger models.

sometimes not learn with either architecture. This may be a result of short training time, sub-optimal learning rate, and data set size.

- The number of layers in a model had a much greater impact on performance than the size of the hidden layers. Different hidden sizes did not seem to significantly impact model performance for either architecture.
- Tests were conducted to establish how consistently the models would achieve results. Over five runs with each architecture and a randomized initialization, results consistently appear to follow the same trend.

6 Discussion

We observe from our results that the training benefits of neuron-level control codes emerge as model size grows, specifically in the layer dimension. While networks consisting of only 2-4 layers are generally hin-

Model	Category	Generation
Default	⟨Garden⟩	i i and to to a the is a . it it and a to the is i i i and the the the the the and to a the
	⟨Music⟩	i to and for is the it i and is . and it a is a for , i and i and , it , and , and i a for is ,
Alternative	⟨Garden⟩	i screw.-no always 5 nope always nozzel 5 screw.-no effect.they reply effect.they nope bagand always always hardy bagand nozzel bagand nozzel hardy hardy
	⟨Music⟩	i and cables and it to cables cables outraged cm-100 , it outraged , cm-100 cm-100 a i a it cables the outraged cables outraged to outraged

(a) Text generations after 1 epoch of training

Model	Category	Generation
Default	⟨Garden⟩	i a to i is and to , and . a the a is the and it to i it i , to . it and the i . a i is it the to
	⟨Music⟩	i , the a and the it , and i it a , the , to . , the , , . . i and i a the a , it it is it and i is
Alternative	⟨Garden⟩	she is it 's got a big black ants and have . so she really do nt that she just like the rabbits and i would get into their own several years we had any problems and he had the safer to keep them ! ! this safer brand i do it , but they do a good product is not the squirrels . he loves it 's not
	⟨Music⟩	i 'm going the most and i do like i 'll just have some more to make the price too , so much , so far as i 'll definitely buy them . the same goes across them in any guitar to keep my strings on strings for the strings and play the time i will say the guitar . but that 's . if there 's a little guys . this . this strings were only strings

(b) Text generations after 50 epochs of training

Table 4: Text generation results for default and alternative models (1024 hidden size, 8 layers) at beginning and end of 50 epoch training run. These models were trained on the 'Reviews' data set. Generations were chosen randomly. The generations for the default model remain basically the same, while the alternative model shows considerable improvement after 50 epochs. Also note that the alternative model shows signs of generating controlled content, with keywords such as 'ants' and 'rabbits' matching the respective category, '⟨Garden⟩'. It should be noted that the purpose of these tests was not to produce well-trained language models capable of producing coherent text. Rather, we were interested in comparing performance between methods.

dered by the alternative approach, larger models with more than 6 layers show valuable performance improvements during training, both in terms of training speedup and emergent learning abilities. The models train faster, perform better, and are in some cases able to learn tasks that the default architecture was unable to master.

We attribute our method's success to the fact that control information is not being lost or diluted via propagation through the network. This is in some ways comparable to the role played by skip connections in deep learning architectures Ronneberger et al. (2015); He et al. (2015). However, in this case we ensure that control information is distributed directly to each neuron in the network rather than simply making it easier to propagate forward. It therefore follows intuitively that models with more layers take greater benefit from this approach.

With the recent popularity of large-scale language models like GPT-3, Palm, and Llama, one naturally wonders how well this method would scale to models with thirty or more layers and 70+ billion parameters. We note that, in future applications of this work, it may not be necessary to provide task-specific knowledge to every neuron in the network in order to obtain the benefits shown in Figure 5. It may be sufficient to inject task-specific knowledge only within a subset of layers, or only to a subset of neurons within each layer.

Thus far, our research has been restricted to small-scale language models with 8 or fewer layers. While this is a key limitation of our work, it also allowed us to iterate efficiently and avoid unnecessary compute usage as we sought an optimal configuration. We note in particular that the goal of this research was to identify a novel machine learning architecture with powerful forward possibilities in the domain of multi-task text generation. We did not set out to produce perfectly fluent language models. Instead, it was expected that if the alternative method proved effective in small learning architectures, it would also be effective in their state-of-the-art cousins. A full-scale application of this method to large-scale language models, while enticing, lies beyond the scope of this work.

7 Conclusion

This work presents a novel training technique for multi-task language models in which a task-specific embedding is appended to the input of each hidden layer in the network, thus facilitating effective distribution of task-specific information to all neurons.

We apply this method to two common neural network architectures used in language-based tasks – Transformer networks and RNNs – and find that, with models containing greater depth layer-wise, our method significantly improves training performance and in some cases enables models to learn where they otherwise wouldn't.

Future work should include further exploration of hyper-parameters, the investigation of hybrid methods wherein task-specific information is injected into an appropriate subset of hidden layers rather than all hidden layers at once, as well as an application of this method to large-scale models for machine translation and controlled text generation.

REFERENCES

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., et al. (2023). Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

Caswell, I., Chelba, C., and Grangier, D. (2019). Tagged back-translation. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, pages 53–63, Florence, Italy. Association for Computational Linguistics.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation.

Dathathri, S., Madotto, A., Lan, J., Hung, J., Frank, E., Molino, P., Yosinski, J., and Liu, R. (2020). Plug and play language models: A simple approach to controlled text generation.

Fan, A., Bhosale, S., Schwenk, H., Ma, Z., El-Kishky, A., Goyal, S., Baines, M., Celebi, O., Wenzek, G., Chaudhary, V., et al. (2021). Beyond english-centric multilingual machine translation. *The Journal of Machine Learning Research*, 22(1):4839–4886.

Ficler, J. and Goldberg, Y. (2017). Controlling linguistic style aspects in neural language generation.

Greene, D. and Cunningham, P. (2006). Practical solutions to the problem of diagonal dominance in kernel document clustering.

Ha, T.-L., Niehues, J., and Waibel, A. (2016). Toward multilingual neural machine translation with universal encoder and decoder. In *Proceedings of the 13th International Conference on Spoken Language Translation*, Seattle, Washington D.C. International Workshop on Spoken Language Translation.

Hanna, M. and Bojar, O. (2021). A fine-grained analysis of BERTScore. In *Proceedings of the Sixth Conference on Machine Translation*, pages 507–517, Online. Association for Computational Linguistics.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

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

J. Schler, M. Koppel, S. A. and Pennebaker, J. (2006). Effects of age and gender on blogging.

Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., Hughes, M., and Dean, J. (2017). Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., and Uszkoreit, J. (2017). One model to learn them all.

Keskar, N. S., McCann, B., Varshney, L. R., Xiong, C., and Socher, R. (2019). Ctrl: A conditional transformer language model for controllable generation.

Kobus, C., Crego, J., and Senellart, J. (2017). Domain control for neural machine translation.

Ni, J., Li, J., and McAuley, J. (2019). Justifying recommendations using distantly-labeled reviews and fine-grained aspects.

OpenAI (2023). Gpt-4 technical report.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Post, M. (2018). A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Project Gutenberg (2023). Project gutenberg.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation.

Sennrich, R., Haddow, B., and Birch, A. (2016). Controlling politeness in neural machine translation via side constraints. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 35–40, San Diego, California. Association for Computational Linguistics.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2020). Bertscore: Evaluating text generation with bert.