Adapting to Word Shifts: Teaching LLMs the Urban Dictionary

Stanford CS224N Custom Project

Justin Wu
Department of Computer Science
Stanford University
justwu@stanford.edu

Sheryl Hsu
Department of Computer Science
Stanford University
sherylh@stanford.edu

Abstract

Large Language Models (LLMs) have seen remarkable success in a multitude of domains, however, these models are generally trained on a fixed corpus of text and need further work to adapt to new data. The world is ever-changing and the usage and meanings of old and new words are constantly shifting. We aim to develop and implement techniques that can improve an LLM's (for example, GPT-2) ability to understand and adapt to new words. Using an Urban Dictionary Dataset of novel words, definitions, and their examples, we find that GPT-2 fine-tuned on an 'understanding words' training objectives does not perform particularly well on in-context learning tasks. We find that changing the paradigm to 'understanding definitions' performs slightly better, indicating that, in few-shot learning settings, GPT-2 might be better at applying new words immediately in context than it is at storing information that is has been given.

1 Key Information to include

- Mentor: Isabel Papadimitriou, Steven Cao

2 Introduction

Natural Language Models (NLMs), particularly Large Language Models (LLMs), have demonstrated remarkable efficacy across multiple domains. However, these models are typically trained on a static corpus of text and require further refinement to adapt to novel data. The dynamism of language and the evolving connotations and usages of both existing and newly coined terms necessitate ongoing updates to ensure that LLMs remain effective in diverse settings. As such, recent works study how we can efficiently adapt existing models to new data. Many of the current methods focus on word changes that are caused by shifts in factual knowledge or task distribution shifts; they don't account for more subtle language changes, like when the popularity or meaning of a word changes over time. These subtle changes have been shown to cause a consistent decrease in model performance. Thus, it is important for LLMs to understand new words through either examples or definitions, in particular, through "in-context learning". However, measuring an LLMs word acquisition ability is difficult in practice because come test time, it is difficult to know which words the LLM has already been exposed to.

The overarching theme in adapting LLMs to distribution shifts is a crucial one: the ability to adapt and understand new words is important for language models like GPT-3 to effectively communicate with and assist humans in natural language tasks. Now, adapting these models to words that change in meaning and popularity over time (such as those in our Urban Dictionary Dataset) is important because it enables the model to keep up with the evolving nature of language. Fine-tuning techniques can be developed and implemented to improve the model's ability to recognize and understand such words, ensuring that it remains relevant and effective in a rapidly changing linguistic landscape.

We explore the space of word shifts, particularly in the case of limited data, by using an Urban Dictionary dataset, that contains one-shot instances of new words, definitions, and an example sentence. To our knowledge, no prior work has been done on understanding novel *urban dictionary* words in context. This task is especially difficult for 3 primary reasons: 1) our dataset contains only one instance of each word, 2) the urban dictionary is not standardized - definitions and examples are based on arbitrary, public input, 3) GPT-2 itself is not a particularly strong LLM for in-context learning (especially by today's standards). Despite these challenges, we aim to leverage the limited data to better understand the semantic shifts of new words in context. This allows us to investigate how new words are used in context and how their meanings may shift over time. We hope to gain insights into how Large Language models internalize new information by developing and implementing techniques that can improve an LLMs ability to adapt to and understand new words.

3 Related Work

In recent years, the areas of word shifts, domain adaptation, and in-context learning have become major components in NLP research.

Word shifts and domain adaptation are subfields of NLP that deal with the challenge of adapting machine learning models trained on one domain to perform well on data from a different domain (or unseen words from a word shift). This is a critical problem in NLP because many real-world applications involve text data from unseen domains. To adapt LLMs to these new distributions, Ziser et al. propose pivot-based language modeling based on long short-term memory (LSTM) networks which predicts the presence of pivots and non-pivots, thus making representations structure-aware.[1] These approaches modify the input features of the model to reduce the domain shift problem. Model-centric approaches modify the model architecture or parameters to adapt the model to the target domain. One groundbreaking approach is domain adversarial training, where the model is trained to distinguish between the source and target domains while minimizing the classification loss [2]. A loss-centric method [3] for domain adaptation involves optimizing a Wasserstein-Distance (an optimal-transport-esque function defined between probability distributions on a given metric space) over distributions of words in our original and fine-tuning datasets. This representation learning technique involves a neural network (the "domain critic") to estimate the Wasserstein distance between the source and target samples and adversarially optimizes the feature extractor network to capture a new distribution that includes the new words. This method has seen success in stable training for domain-adaptation tasks.

In-context learning refers to the process of training language models by incorporating contextual information from the surrounding text. As such, properly evaluating LLMs on in-context learning tasks have received much attention. In [4], Eisenschols et al. introduces a new paradigm for measuring how effective LLMs are in learning novel words during inference. In doing so, they produce a new dataset, WinoDict, that was built using a heuristic method for introducing and defining new words. This method involves drawing keys and definitions from WinoGrande [5], WinoGrad [6], and WordNet [7], and combining these pairs with a morphological method in creating synthetic words. The result is a novel dataset of unseen words with plausible definitions. This dataset is then used to test models for word acquisition ability (results are compared to human evaluation). Using this new metric, they benchmark the performance of several state of-the-art LLMs (like GPT-3 and PaLM) across scale and number of shots. Eisenschols et al. evaluation method is highly relevant to our approach in this project, as we measure LLMs on their ability to complete fill-in-the-blank in-context prompting.

4 Approach

Our first approach is to evaluate raw and fine-tuned GPT-2 on multiple-choice, fill-in-the-blank questions that test the model's ability to understand novel words. We also implemented another approach that follows a different training paradigm - rather than teaching GPT-2 new words and testing on those words, we teach GPT-2 how to interpret definitions and apply them on an individual basis. These approaches are described in length in the following subsections. **All of our code and experiments were written by ourselves, from scratch, and with no references to other codebases.**

4.1 Baselines

We have two baseline models for predicting the in-context multiple-choice prompts, which differ by their embedding initialization. In our first baseline, we initialize the Urban Dictionary words as random noise embeddings and evaluate raw GPT-2 over the multiple-choice prompts. For our second baseline, we initialize the word embeddings by drawing from a Gaussian distribution where the mean is the average over all of the existing words that GPT-2 has stored. These baselines represent how well our LLM performs with no direct information about the meaning of these words.

4.2 Fine-Tuning

We fine-tune GPT-2 by training on 50k words that are formatted in sentence form ([word] means [definition]). We also feed the model example sentences that put the words in context, according to a 50/25/25 train/val/test split. This way, before evaluation, the model will have seen how other new words are being used in context. We trained GPT-2 on this data using casual language modeling, where the model has to predict the next word to fill in a given sentence. We selected casual language modeling since GPT-2 was already trained using casual language modeling.

Given this structure, we fine-tune on a model where the new words are initialized as random noise, and on a model where the new words are drawn from a gaussian distribution. The idea here is as follows: adding words into a vocabulary as small-norm random noise can cause GPT-2 to place probability ≈ 1 on the new words for most prefixes, and thus, generating from this model will only generate the added words, leading to worse domain adaptation performance as the first gradient steps only remove probability from the first words. On the other hand, averaging existing embeddings for initialization bounds the KL divergence between pre and post-expansion token-level distributions, which stabilizes the fine-tuning process. This approach was inspired by [8], where the theory is further developed.

After evaluating results from fine-tuning on the two different initalizaitons, we chose to add an adaptor to our model. Our adapter architecture [9] consists of an adapter layer after each feedforward block in the transformer layers and an invertible adapter layer after the LM layer. Essentially, instead of adjusting GPT's weights, we train the adapter. This makes training faster and increase portability, as the adapter can be stitched into many models. This is especially relevant to our task, where we could train an adapter to recognize new words and then stitch it into many other GPT2 models without having to retrain.

4.3 Predicting Embeddings

We trained BERT to predict the word embedding of a word based off its definition. For training data, we used words that were in our Urban Dictionary dataset and were also already known by GPT2 (had a word embedding). As seen in Figure 1, we then added a custom head to HuggingFace's BERT model which used a linear layer to map the last hidden state of the CLS token to a word embedding with the correct dimension. After training BERT using a mean square error loss function, we make predictions of the word embeddings for unknown words and add these embeddings to GPT2.

4.4 A New Training Paradigm

Our other approach involves a reformulation of the task and a different training paradigm. Rather than training on definitions and example sentences, we train on pairs of words with their definitions and a masked sentence that corresponds with one of the two words. The model then has to predict which of the two words the example sentence corresponds to. As an example, perhaps we have the two words:

{word: "massive", definition: "large and heavy or solid, exceptionally large"}

{word: cat, definition: "a small mammal with soft fur and retractable claws."}

{example: "The crowd at the football game was [MASK]"}

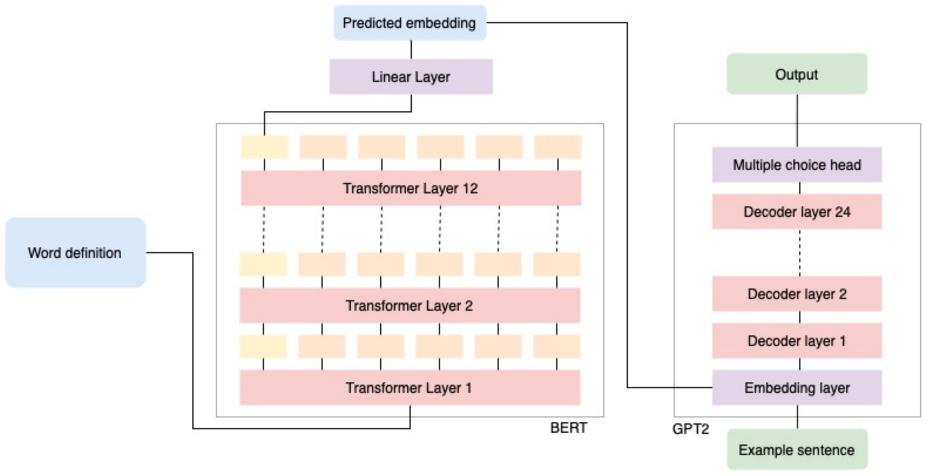


Figure 1: BERT Embedding Prediction Pipeline

where our LLM would attempt to guess the masked word, "massive". As a first baseline with this method, we proceed with a standard fine-tuning pipeline, in which GPT-2 generates a prediction of the masked word. Our Cross Entropy Loss internally converts this prediction into a probability distribution over all words in the tokenizer, which we can then use to calculate a loss against the true word.

We run a similar experiment with **weight decay**, which is a regularization technique that adds a small penalty (the L_2 norm of the weights), to the loss function. This effectively penalizes large weight values in a neural network's parameters during training, helping to prevent overfitting and improve generalization performance.

We also consider **gradient clipping**, which is a technique used in machine learning and deep learning to prevent exploding gradients during training. Gradient clipping involves setting a threshold value for the gradient. If the gradient value exceeds the threshold, it is clipped or truncated to the threshold value. By limiting the gradient values, gradient clipping helps to stabilize the training process and prevent the model from diverging or oscillating during training.

Finally, we consider **gradient accumulation**, which is a technique used in deep learning to simulate the effect of using a larger batch size during training, without requiring a larger memory capacity. Normally, during training, the weights of a neural network are updated after every batch of data samples is processed through the network. However, in gradient accumulation, the weight updates are performed after accumulating gradients from several mini-batches.

The general idea here is not for the model to understand particular words, but rather, for the model to be able to apply words in context on a per-instance basis. During the evaluation, unlike our previous methods, we test on words and definitions that we have **not** trained on before.

5 Experiments

5.1 Data

We extracted our data from an Urban Dictionary Dataset, which contains approximately 1.2 million words, their definitions, an example sentence, and other attributes. From this dataset, we drew 50,000 words uniformly at random, which we used to train, test, and evaluate our models. We checked to ensure that GPT did not already know the words. From the 50,000 words, we randomly selected 12,500 of them to evaluate all 6 models on. Using this sample allowed us to try out a lot of different ideas more quickly and without straining computing resources. To run our experiments on the other training paradigm, we draw 21,500 words, definitions, and examples (split between training and evaluation).

5.2 Evaluation method

Our primary method for evaluation involved creating fill-in-the-blank questions based on the example sentences. For each example sentence, we mask out the correct word and have the model choose the correct word out of a few different choices. The overall accuracy is then simply the percentage of questions that the model gets correct.

Our evaluation method on the other training paradigm follows a similar spirit. We provide the fine-tuned model with two words, their definitions, and a masked example sentence. The model then chooses which of the two words the masked example corresponds to. Again, the accuracy will be the percentage of questions that the model gets correct.

5.3 Experimental details

5.3.1 Fine tuning

Our main experiments involved investigating the effectiveness of two different methods of word embedding initializations (random vs average) and the effect of fine-tuning on words and definitions using masked language modeling. To train our model, we used HuggingFace's GPT2LMHeadModel, which contains the standard GPT-2 transformer with a language modeling layer on top; this architecture is particularly suitable for fine-tuning and domain adaptation tasks. To fine-tune the GPT-2 model, we used a casual language modeling approach, which involves training the model to predict the next token. We used HuggingFace's DataCollatorForLanguageModeling to prepare the data for training. We trained this using a learning rate of 3e-4 and weight decay of 0.1. We computed our validation loss using the perplexity metric [10] defined in Eq.1[11] and trained each model until the validation stopped decreasing.

$$PPL(X) = \exp\Bigg(-\frac{1}{t}\sum_{i}^{t}\log p(x_i|x<i)\Bigg) \quad (1)$$

Overall, each models took approximately 3 hours to train when training using the standard Google Colab GPU compute resources.

5.3.2 BERT Embedding Prediction

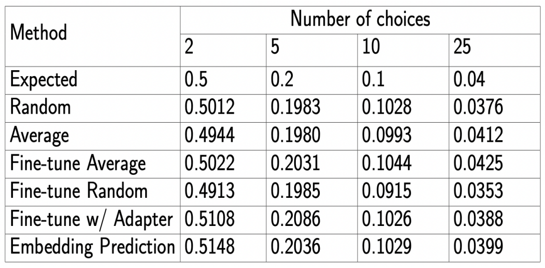
For BERT embedding prediction, we wrote a custom Pytorch model that used HuggingFace's BERT implementation and added a linear layer that mapped BERT's last hidden state to a tensor with the correct dimensions for GPT2's word embedding. We gathered 21,316 words for which we had both Urban Dictionary definitions and GPT2 word embeddings and split this into a 90/10 training/validation set. We used a mean square error loss for both our training and validation loss and trained until the validation loss stopped decreasing (12 epochs).

5.3.3 New Training Paradigm

We ran the "Applying Words in Context" models over 15000 training examples, drawn uniformly at random, from the Urban Dictionary Dataset. Each of these models ran under 10 epochs, 32 batch size, and 5e-5 learning rate. With gradient accumulation, we accumulate gradients over 2 batches before updating. With gradient clipping, we specify a max value at norm 1.0. Finally, we fine-tune with a weight decay of 0.1. We then evaluated the accuracy of these models by randomly drawing 6500 *different* words and definitions.

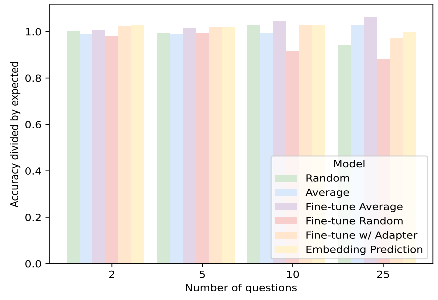
5.4 Results

Our results for the multiple choice prompts are summarized in Figure 2. As expected, fine-tuning on words that had been initialized based on a distribution of the existing embeddings worked better than randomly initializing. In addition, fine-tuning did not particularly help, regardless of the number of choices in the number of multiple-choice prompts. This is as expected: teaching GPT-2 to store and understand new words from definitions that it might not understand either is not the most promising. Having at most two (definition, example) sentences to train on is a huge challenge as casual language modeling typically requires more.



Method	Number of choices			
	2	5	10	25
Expected	0.5	0.2	0.1	0.04
Random	0.5012	0.1983	0.1028	0.0376
Average	0.4944	0.1980	0.0993	0.0412
Fine-tune Average	0.5022	0.2031	0.1044	0.0425
Fine-tune Random	0.4913	0.1985	0.0915	0.0353
Fine-tune w/ Adapter	0.5108	0.2086	0.1026	0.0388
Embedding Prediction	0.5148	0.2036	0.1029	0.0399

(a) Accuracy of models over number of choices



(b) Standardized comparison of accuracy

Figure 2: Accuracy of the 'Understanding Definitions' Paradigm

	Baseline Fine-Tuned	Weight Decay	Gradient Clipping	Gradient Accumulation
Correct	3254	3830	3474	3665
Total	6500	6500	6500	6500
Accuracy	0.5007	0.5892	0.5344	0.5639

Table 1: Accuracy of the 'Applying Words in Context' Paradigm

Our results for the other training paradigm is summarized in Table 1. While neither of these models performed particularly well, fine-tuning with weight decay did show some promise. This suggests that overfitting might have been a factor in the other experiments.

6 Analysis

6.1 Number of choices

We were first motivated to explore giving GPT2 multiple choice questions with a varying number of choices because when doing preliminary exploration, we noticed that it could pick up on part of speech. For example, when we gave it the sentence "I ate [MASK]." which the choices of pizza, table, mom, running, and music it chose running only four percent of the time compared to the next lowest being 15 percent. As such, we thought that GPT2's ability to recognize part of speech with words with common endings like "ing" could be a cofounding factor to measuring its ability to understand word meaning, and by having a greater number of options this factor was diminished since with 25 options, there are likely many words of each part of speech. However, given that the accuracy normalized by the expected is roughly the same across all number of choices, we determine that GPT2 was not able to meaningfully use common endings that suggest part of speech to predict words.

6.2 BERT Embedding Prediction

One concern we had about BERT is that it was predicting the same embedding for all words/definitions. We find that on average the standard deviation of a given dimensions across all 12,500 words we predicted on is 0.02, compared to 0.1216 on the existing word embeddings. There was also a significant difference in our means, as the magnitude of the difference of the mean tensor for the existing and predicted embeddings was 0.2556.

To better visualize our word embeddings, we used principal component analysis (PCA) to extract the top 50 features from our embeddings and then used t-Distributed Stochastic Neighbor Embedding (t-SNE) to map them to a 2D space. We then generated a heat map demonstrating how many embeddings are in a given area, as seen in Figure 3. We see that overall the predicted distribution on the left seems to be pretty uniform, with no clear clusters compared to the existing embedding distribution on the right.

With this in mind, we thought that training for more epochs would make BERT create predictiosn better in line with the existing embeddings. As such, we allowed BERT to run until the validation

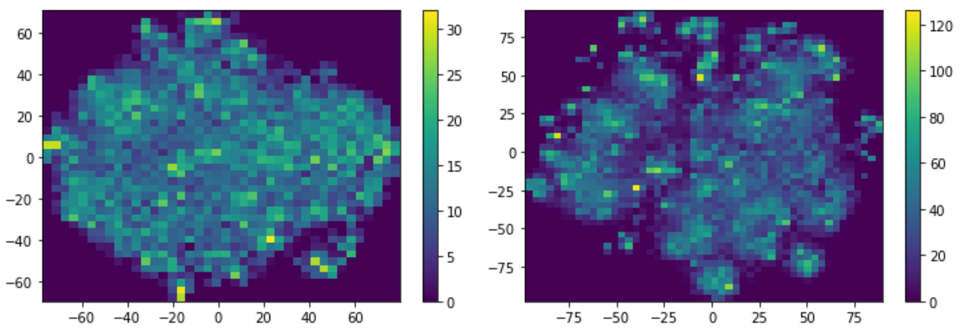


Figure 3: Heatmap of spatial distribution of predicted BERT embeddings. Left: predicted embeddings, Right: existing embeddings

accuracy started increasing (overfitting), which took 12 epochs. This is much longer than the standard 4 epochs and resulted in slightly better performance, but ultimately failed to fully resolve the issue.

6.3 Comparing methods for Understanding Definitions Paradigm

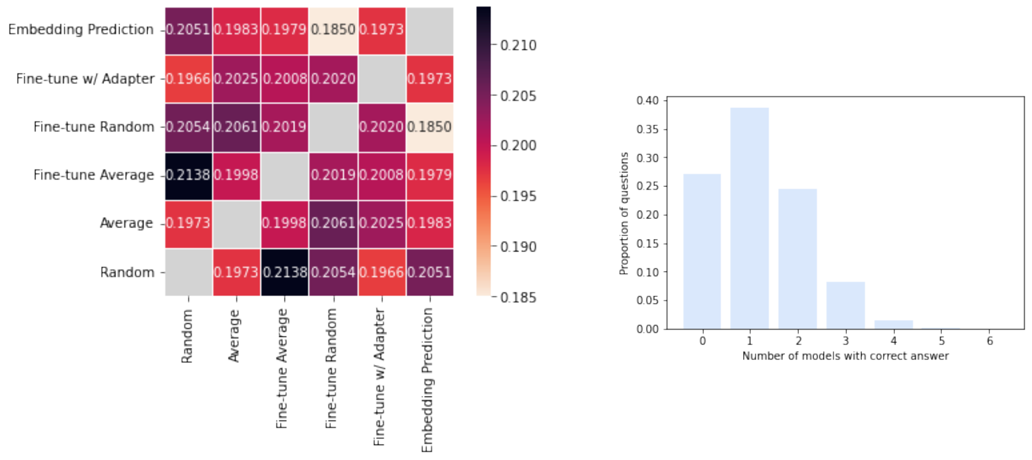


Figure 4: Left: the proportion of questions on which two methods selected the same answer. Right: the proportion of questions with which 0 through 6 models selected the correct answer.

Although all six methods had approximately the same performance, we further analyze the models' behaviour. In the left of Figure 4, we go through all the questions the models were evaluated were on with five choices and compute the proportion of questions on which two models selected the same answer. All of the proportions are around 0.2, which we would expect if the models are more or less choosing randomly. None of the models appear to be closely aligned to each other; the models are not choosing the same wrong answers which we expected them to do if two words have similar meanings but instead appear to be more or less randomly choosing.

On the right of Figure 4, we see the proportion of questions with which a certain number of models get the correct answer. We see that very few questions have more than 4 models get the correct answer; in fact there are no questions on which all six models get the correct answer.

Overall, the models seem to be more or less randomly choosing. This seems to indicate that they do not know enough about the five choice words to do any better than randomly guessing. We attempt to increase the number of epochs we train for in hopes that seeing the data more times will allow the model to better learn, but ultimately this is fundamentally a challenge caused by the limitations of our data.

6.4 Applying Words in Context

The framework in which we teach the model to apply new definitions immediately in context has shown slightly more promise. In particular, we note that each of weight decay, gradient clipping, and gradient accumulation perform better than baselines (though marginally). The fact that weight decay performed best suggests that our initial fine-tuning pipeline was overfitting to the training data; this is strange, considering our training data is full of one-shot instances of words. This indicates that our selected data may have had a substantial number of words and definitions that are semantically very similar, where the model may be correlating features of different words together. One relevant future step will be to test these same models on different samples of the large Urban Dictionary dataset, and compare the effects of regularization in each instance.

7 Conclusion

In summary, we successfully trained several fine-tuning models for GPT-2, and experimented with a few training paradigms for experimenting with how well GPT-2 can adapt to new contexts. We found that getting GPT-2 to understand the actual definitions was slightly more difficult than teaching GPT-2 to apply definitions on a per-instance basis. We were bottlenecked primarily on our dataset and our compute. In particular, our dataset only has one instance of each training example, meaning we require our LLM to understand words with very limited data, which is in itself, a very difficult framework to perform well on. Further, the example sentences in the dataset are quite arbitrary and don't necessarily encapsulate proper usage of the word (especially considering there is only one example).

In **future works**, we aim to look at data augmentation techniques for generating synthetic sentences for few-shot learning. One naive approach would be to curate a dataset on top of the Urban Dictionary dataset that includes multiple definitions and examples for a single word. Another approach would be to build off of the works of Winodict [4] to generate synthetic words; to an LLM, these words would have an identical effect as Urban Dictionary words in terms of training and understanding. It would also be interesting to experiment with other, potentially stronger, LLMs. By now, GPT-2 itself is not a state-of-the-art model, and so looking at this problem through a stronger lens like GPT-3 may provide more practical insights. As our main bottlenecks with this project were the limited power of the LLMs and the limited data, these future approaches have strong potential to yield interesting results.

References

[1] Yftah Ziser and Roi Reichart. Deep pivot-based modeling for cross-language cross-domain transfer with minimal guidance. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 238–249, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[2] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *J. Mach. Learn. Res.*, 17(1):2096–2030, jan 2016.

[3] Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. Wasserstein distance guided representation learning for domain adaptation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[4] Fangyu Liu, Jeremy Cole, Julian Martin Eisenschlos, and William Weston Cohen. Winodict: Probing language models for in-context language acquisition. In *EACL*, 2022.

[5] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, aug 2021.

[6] Vid Kocijan, Thomas Lukasiewicz, Ernest Davis, Gary Marcus, and Leora Morgenstern. A review of winograd schema challenge datasets and approaches. *arXiv preprint arXiv:2004.13831*, 2020.

[7] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, nov 1995.

[8] John Hewitt. Initializing new word embeddings for pretrained language models, 2021.

[9] Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online, November 2020. Association for Computational Linguistics.

[10] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, 1977.

[11] HuggingFace. Perplexity of fixed length models.