On the Ability of Attention-based Networks in NMT

DDA4220/MDS6224/MBI6011 Final Project

Guiming Chen (119010012)
School of Data Science
The Chinese University of Hong Kong, Shenzhen
guimingchen@link.cuhk.edu.cn

Ziqi Zhang (119010455)
School of Science and Engineering
The Chinese University of Hong Kong, Shenzhen
ziqizhang@link.cuhk.edu.cn

Abstract

Attention-based networks have become a popular choice in NLP nowadays. In this project, we evaluate three types of models: Bahdanau RNN, vanilla Transformer and LLMs. We adopt the Bahdanau RNN as our general baseline. We reveal the laziness of this model observe that the teacher-forcing ratio plays an important role in model training. For improvement, we train 16 Transformer models from scratch and investigate the impact of imbalanced encoder-decoder layers in Transformer. We find that Transformer decoder is a more powerful module than encoder. Lastly, we evaluate the zero-shot results of Phoenix and Chimera, which outperform Bahdanau RNN and vanilla Transformer by a large margin.

1 Workload

Ziqi finishes section 3.1, 4.1 and 4.5.2 of the report. Guiming is responsible for all codes, experiments and the rest parts of the report.

2 Introduction

Attention based networks have become the basis of NLP. [1] first introduces an attention based encoder-decoder network to align the inputs and outputs in neural machine translation (NMT). Then to close the gap between training and inference, [2] introduces a teacher-forcing decaying schedule during learning, which greatly improves the performance in NMT tasks. In particular, we are interested in how different decaying schemes and the number of hidden dimension can affect the Bahdanau network's performance in NMT task.

Recently, Transformer [3] and its descendants [4, 5, 6, 7] impress people with excellent performance across multiple tasks. Thanks to its attention mechanism and the development of GPU, models are able to scale up and train within acceptable amount of time. In this report, we would like to train multiple vanilla Transformer models from scratch and try to reveal the impact of the number of encoder/decoder layers on model performance.

More recently, Large Language Models (LLMs) which uses the decoder part of Transformer impress people with excellent zero-shot performance and the ability to understand human instructions. In this report, we adopt Phoenix and Chimera [8] released by our university and evaluate their zero-shot performance on this NMT task.

The structure of this report is as follows: we first introduce the related works in section 3 and the motivation of our experiments in section 4. Then we conduct experiments on Bahdanau RNN (section

5), Vanilla Transformer (section 6) and the two LLMs (section 7). Then we conclude our report in section 8.

3 Related Work

3.1 RNN-based Network

In recent years, machine translation develops rapidly due to the method of deep neural network, especially recurrent neural network (RNN). In particular, [9] adopts the pipeline of encoder-decoder models for translation tasks.

Encoder-decoder network structure consists of two parts. The first part is encoder, which stacks several RNN layers together in order to map the input vector sequence $x \in \mathbb{R}^{N \times L \times D}$ (embedded from Chinese token strings) to a hidden (or encoded) state $h \in \mathbb{R}^{M \times N \times H}$, where N is number of samples in one batch (or batch size), L is (padded) length of sequences in one batch, D is embedded dimension of token vectors, M is number of RNN layers, and H is dimension of hidden state.

The second part is decoder, which decodes the information previously encoded in the hidden state and generates the translation sequences to the target language (English). The decoder is a similar network architecture of stacked RNN layers. The initial hidden state of decoder is the final hidden state encoded by encoder. In the training stage, decoder input is the shifted ground truth target sequences (also called teacher forcing input), but in the inference stage, decoder will take the decoder output of the last timestamp as its next input.

As for the type of RNN, [10] takes the advantage of Long Short Term Memory Network (LSTM) in their encoder-decoder model. However, [11] proposes another RNN choice, Gated Recurrent Neural Network (GRU).

In addition, [1] employs attention technique to improve the RNN network structure, which can well capture corresponding relationships among the words and polishes the performance. Bahdanau attention network makes use of an alignment score model and calculates the linkage between encoded states and the target words for each timestamp, which generates context vectors. Finally the attention decoder predicts the probabilistic distribution of the future word tokens conditioning on the previous words and the context information.

In details, Bahdanau attention utilizes a full connected layer as the alignment model. It computes the alignment between the $(i-1)$-th decoder hidden state and the j-th encoded hidden state as following:

$$e_{ij} = a(s_{i-1}, h_j) = v_a^T \tanh(S_a s_{i-1} + U_a h_j)$$

Then, the model computes the weight distribution from the alignment by softmax function:

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{L} exp(e_{ik})}$$

The context vector c_i for the i-th target word y_i is obtained as a weighted sum of encoded hidden states h_i at all timestamps:

$$c_i = \sum_{j=1}^{L} \alpha_{ij} h_j$$

Next, the decoder updates the hidden state s_i via another RNN by using the context vector c_i and the information from the last timestamp:

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

In the last step, the decoder predicts the output probabilistic distribution of the i-th target word y_i conditioning on previous prediction:

$$p(y_i \mid x, \{y_j\}_{j=1}^{i-1}) = g(y_{i-1}, s_i, c_i)$$

3.2 Vanilla Transformer

[3] proposed an encoder-decoder architecture with attention modules that can better capture the information between tokens in a sequence. This architecture is famous for its QKV attention modules.

There are three types of attention modules: *Self Attention (SA)*, *Masked Self Attention (MSA)* and *Cross Attention (CA)*. The CA module is solely used in encoder. Given a batch of input x with dimension (batch size, sequence length, hidden dimension) $= (N, L, D)$, we perform batch matrix multiplication:

$$Q = W_Q(x), K = W_K(x), V = W_V(x)$$

where W_Q, W_K, W_V are three learnable matrices, and

$$softmax(\frac{QK^T}{\sqrt{D}})V$$

where the scaled $softmax$ is performed over the last dimension and $\sqrt{D}$ is a scalar that smooths the values in the attention matrix QK^T. The output dimension of the SA module is the same as its input dimension, which makes it possible to add a shortcut connection (citation) that allows for gradient flow.

For the MSA and CA module, we only point out their differences $w.r.t.\ SA$. MSA module is solely used in decoder. An upper triangular mask is added on the attention matrix QK^T to prevent the model from seeing future tokens, such that the model can achieve better performance at inference.

CA is another module that is used in the decoder only. Its QKV are computed as:

$$Q = W_Q(x_{encoder})$$
$$K = W_K(x_{decoder})$$
$$V = W_V(x_{decoder}),$$

where $x_{encoder}$ is the output of encoder's last layer, and $x_{decoder}$ is the output of the previous decoder layer. *Cross Attention* module, as it name suggests, merges the information of encoder and decoder by using the encoder output as query. By back propagation, this module is able to answer: *how much attention should each token on the decoder side pay to each token on the encoder side?* This module is not only useful in neural machine translation (NMT) [12], but also in multimodal models [13, 14, 15] where fusion of knowledge is heavily required.

3.3 Large Language Models

Language models use Transformer encoder [4, 5], or decoder [7, 16, 17], or both [3, 6] as backbone and are pretrained using Masked Language Modeling task. Finetuning their classifier head can adapt them to different downstream tasks.

Large language models (LLM) refer to language models with large number of parameters. When a model's parameter size is scaled up (e.g. by stacking more layers or making it wider) , either from millions to a billion, or from a billion to many billions, its performance on certain tasks would show a great leap. This unpredictable ability that small models lack is called "emergent ability" [18]. LLMs are able to achieve impressive zero-shot results across different tasks.

Recently, a number of LLMs [19, 20, 21, 22, 23, 24, 8] emerge so as to democratize the close-sourced ChatGPT. Among them, Phoenix [8] takes a lead, being able to achieve 96% of ChatGPT's ability. Phoenix is a multilingual LLM across Latin and non-Latin languages. It uses BLOOM [23] as its backbone and is finetuned on high-quality conversation and instruction data. To reduce the multilingual tax, another model called Chimera uses LLaMa as backbone and is finetuned solely on Latin language. Owing the ability of backbone, Chimera slightly underperforms Phoenix in several tasks.

4 Method

4.1 Evaluation method

We adopt BLEU score [25] as our metric to evaluate the translation quality. BLEU is a well-recognized metric for machine translation, whose score value ranges from zero to one, with larger value indicating better translation quality, and better consistency between the output translation and reference translations. Mathematically, BLEU score is defined by the following equation:

$$BLEU = BP \cdot \exp(\sum_{n=1}^{N} w_n \cdot \log p_n)$$

or equivalently,

$$\log(BLEU) = \log(BP) + \sum_{n=1}^{N} w_n \cdot \log p_n = \min\{0, 1 - \frac{r}{c}\} + \sum_{n=1}^{N} w_n \cdot \log p_n$$

Here BP is the brevity penalty given by $BP = 1 \cdot \mathbf{I}_{(c > r)} + e^{1-r/c} \cdot \mathbf{I}_{(c \leq r)}$, penalizing the sentences that are too short, where c is the total length of output translations and r is effective reference length defined by $r = \sum_{i \in output} len(argmin_{j \in reference} |len(i) - len(j)|)$.

In the formula above, p_n is modified n-gram precision, which is calculated as:

$$p_n = \frac{\sum_{c \in output} \sum_{n\_gram \in C} count\_clip(n\_gram)}{\sum_{c \in output} \sum_{n\_gram \in C} count(n\_gram)},$$

where C is an output translated sentence, n_gram is a phrase extracted from C with length n, $count\_clip = \min\{count, max\_ref\_count\}$, $count$ is the word count of n_gram in the output translation C, and max_ref_count is the maximal word count of n_gram among all the reference translations.

4.2 RNN-based network

4.2.1 Teacher-forcing ratio

For RNN-based networks, one of the key factors to the success of training is the usage of teacher forcing ratio p. When training the decoder part of an RNN, we have a probability of p to use the ground truth as the input of next time step, and $1 - p$ to use the prediction of the current step as the input. A large p is found to contribute to training stage, but is likely to cause high inference errors, because there is no ground truth serving as input at inference. Put in other words, a student who is used to cheating in homework is likely to fail an exam. This is the dilemma faced when training and testing with encoder-decoder based RNNs.

We set $p = 1$ as our baseline for training Bahdanau RNN. We borrow the idea from [2], where they use a p-decaying schedule as the training proceeds, trying to gradually close the gap between training and inference. In this report, we first show through a simple experiment that the model tends to be *lazy* at training. Then we experiment with different p, including constant and decayed ones, to see their impact on the translation task.

4.2.2 Hidden Dimension

We also conduct experiments on different hidden dimensions. We try to answer the question *does the performance of RNN increase as its width increases?* Intuitively speaking, the hidden dimension H serves as the bottleneck of the information flow for most of the traditional RNN-based encoder-decoder network architecture (without attention mechanism). This is because the encoded state passed from encoder to decoder lies in the vector space $\mathbb{R}^{M \times N \times H}$, whose dimension is greatly limited by hidden size H.

Given high dimensionality of word embedding and long sentences, it requires a large enough hidden dimension H to broaden the bottleneck, so that the encoder could capture more vital information from input sentences and the model would have a better representation power. Nevertheless, on the other hand, if H is too large, not only will we suffer from long training time and huge memory burden due to increasing number of parameters, but also the model itself might encounter serious overfitting problem.

Therefore, it is necessary to optimize the hidden dimension H. For this purpose, we conduct experiments to investigate RNNs with 5 different hidden sizes ($H = 128,\ 256,\ 512,\ 1024and2048$). More details are in section 5.

4.3 Vanilla Transformer

The Vanilla Transformer proposed by [3] in 2017 was designed for the NMT task. Instead of trying to match our performance with theirs, we conduct a comparison experiment. Specifically, we would like to answer the following to question: *How is the depth of model related to its performance?*

The depth of a model usually refer to the number of encoder/decoder layers. For example, in [3], $N = 6$ means there are 6 encoder layers and 6 decoder layers, which result in a total of 12 layers. An interesting thought would be making $N_{encoder}$ and $N_{decoder}$ unequal, and see how the imbalance in depth of encoder and decoder affects the overall performance of the model. More detailed settings are in section 6.1.

4.4 Large Language Models

Since we do not have enough computational resources to finetune LLMs with billions of parameters, we directly generate zero-shot results from them. We pick 2 Phoenix models and 4 Chimera models, and prompt them with the same instruction template. More details are in section 7.

5 Experiments on Bahdanau RNN

5.1 General Experiment Setup

We implement a `PyTorch` version of RNN with Bahdanau attention. A unidirectional, single-layer GRU with a hidden dimension of 256 is used as the base model. Reasonably, this setting will not achieve the most satisfying result. But for the purpose of experimenting with teacher-forcing ratio p and hidden dimension D, this setting allows us to confirm where the source of variation in results is possibly from.

Due to the limited time and computational resources, we are not able to train our models on full training set considering the size of the models. Hence, we only train all models on 100K data that are randomly sampled from the 3M dataset, and validate our model using the full validation set which has around 30K data. All models are trained with batch size 100 for 20 epochs. We use AdamW optimizer with weight decay 1e-2 and an initial learning rate of 5e-4 with no scheduler. Negative likelihood loss is use to measure the gap between predictions and ground truths. We save and evaluate on the models with the lowest validation loss.

5.2 To be lazy or diligent?

5.2.1 Setup

We design a simple experiment to illustrate that the decoder tends to rely on ground truth information if we do not forbidden it from doing so. Teacher forcing only allows the model to choose *either* the ground truth or predictions of the previous step as the input as the current step. However, the p is pre-assigned as a hyperparameter and cannot be learned through backpropagation. Hence, we initialize a learnable scalar and normalize it to [0,1] with a sigmoid function. Let β be the normalized scalar. Originally, the input of current step t is

$$x_t = \begin{cases} E(y_{t-1}), & \text{if } \texttt{random.random()} < p \\ E(\hat{y}_{t-1}), & \text{otherwise} \end{cases} \tag{1}$$

where $E(\cdot)$ is a function that converts an index to the corresponding embedding, y_{t-1} is the ground truth of the previous step and $\hat{y}_{t-1}$ is the predicted token of the previous step. Now with β, the input becomes

$$x'_t = \beta E(y_{t-1}) + (1 - \beta)E(\hat{y}_{t-1}). \tag{2}$$

β is initialized as 0.5. We experiment on two settings. We first let the model freely learn a β, which we refer to as *unpenalized* version. Then we penalize the β by directly adding to the loss function, which is the *penalized* version.

5.2.2 Results and Analysis

The results are rather intuitive. Figure1 shows that for the unpenalized version, β is gradually updated to 1 since it is much easier for the model to predict the next token with more information from the ground truth instead of its own output. For penalized β, since it is exposed directly in the loss function, its value shrinks rapidly as the iteration proceeds and converges at around 0.1. In this case, the model uses more information from its own output than the ground truth. As a consequence, the training loss of the penalized version is slightly higher than its counterpart (Figure 2).

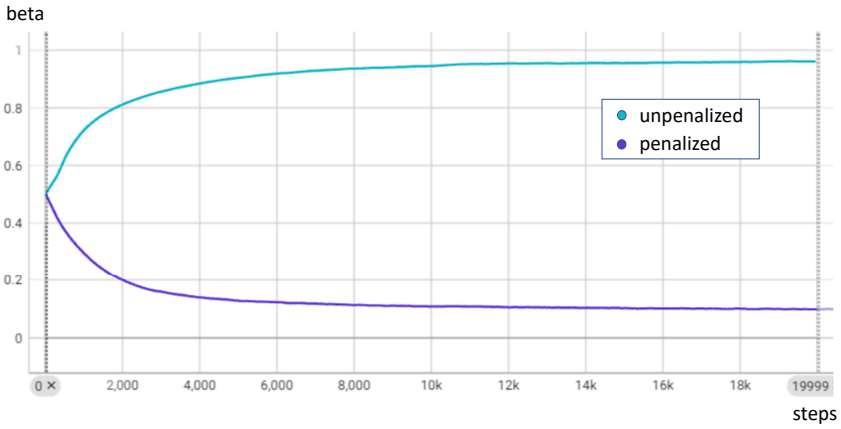

Figure 1: β value under penalized and unpenalized settings.

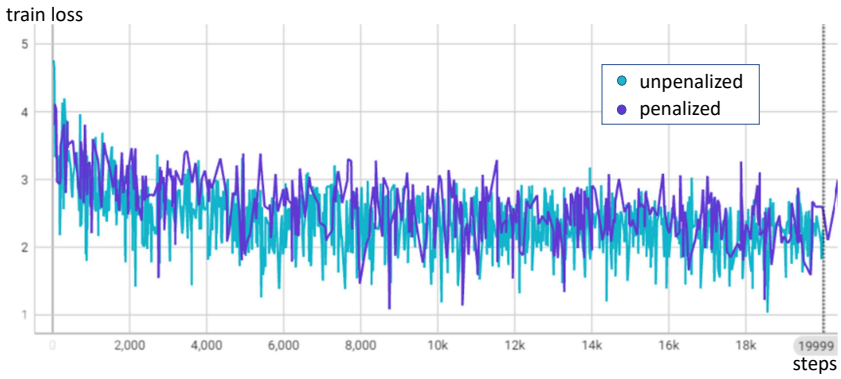

Figure 2: Training loss under penalized and unpenalized settings.

The relative relationship of validation loss is also reasonable (Figure 3). Since the unpenalized version largely depends on the ground truth as input, there is a large behavioral gap between training and inference. However, the average validation loss is around 7, which is rather high given that, seemingly, it only uses 10% of the information from the ground truth. However, we need to emphasize the setting of this experiment. The weighted sum of embedding in eq. 2 indicates that the ground truth information is used in **every** step, which is not the case for normal teacher forcing setting in eq. 1, and is not the setting of inference either. The high validation loss indicates that the gap between training and inference is not closed, and that the model might still be able to cheat from the 10% information from the ground truth during training.

These two experiments indicate that the model tends to be lazy and tries to grasp any useful information exposed to it.

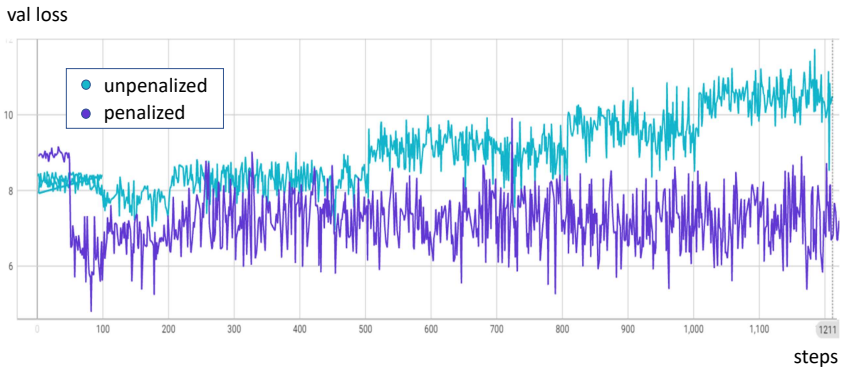

Figure 3: Validation loss under penalized and unpenalized settings.

5.3 Teacher-forcing ratio

5.3.1 Setup

We set p (teacher-forcing ratio) to constant 1 as our baseline and experiment on other 2 constant settings ("constant at 0.5", "constant at 0") and 3 decayed settings ("decay from 1 to 0.5 in 1000 steps", "decay from 0.5 to 0 in 1000 steps", "decay from 1 to 0 in 2000 steps"). For decaying strategy, we use linear decay. For example, decaying from p_{max} to p_{min} in t steps yields:

$$p_t = max(p_{min}, p_{max} - \frac{p_{max} - p_{min}}{t})$$

All other settings follow the general settings of this section. We train all models for 20 epochs, and report the lowest training loss, lowest validation loss and the BLEU score that corresponds to the lowest validation loss. Note that the lowest training loss and val loss are achieved at different training steps. But since BLEU score is not integrated in the training pipeline, we only report the BLEU score with the lowest validation loss.

5.3.2 Results and Analysis

Description of tf Scheme	lowest train loss	lowest val loss	BLEU↑
constant at 1 (baseline)	**3.02**	7.29	1.19
constant at 0.5	4.18	5.72	**4.71**
constant at 0	4.76	5.48	1.51
decay from 1 to 0.5 in 1000 steps	4.18	5.71	4.49
decay from 0.5 to 0 in 1000 steps	4.76	5.47	1.48
decay from 1 to 0 in 2000 steps	4.74	**5.45**	1.58

Table 1: The lowest training loss, the lowest validation (val) loss and BLEU scores that correspond to the lowest val loss for each teacher-forcing (tf) scheme.

Table 1 summarizes the results of different teacher-forcing (tf) schemes. Unsurprisingly, the baseline model achieves the lowest training loss, the highest validation loss and the lowest BLEU score. This is due to the fact that learning one-step prediction from the ground truth is much easier than predicting from its previous outputs, since the error would accumulate over time steps. However, at inference, there is no ground truth as input for each time step. Hence, the model would have difficulty predicting

stuff from its previous outputs. This discrepancy thus manifests itself in terms of losses and BLEU scores.

Besides, to show that the error accumulates over time steps, we select two examples from the validation set.

English: The second encounter relates to my grandfather's treasure box.
Chinese: 第二次事件跟我爷爷的宝贝匣子有关。

Description of tf Scheme	Prediction
constant at 1 (baseline)	这个人的人员会在我的人们的一个人都是一个人的。
constant at 0.5	第二个的作为我的的的的的的的。
constant at 0	第二二的的的的的的的的的。。。。
decay from 1 to 0.5 in 1000 steps	第二次的的的的我的的的的的箱子。
decay from 0.5 to 0 in 1000 steps	第二的的的的的的的的的的的。。
decay from 1 to 0 in 2000 steps	第二的第的的的我我的我的的的的的。。。

English: Reduce blood fat, prevent thrombosis, arteriosclerosis, apoplexy and heart disease. Improve memory, nourish the brain and improve the intelligence.
Chinese: 降血脂，预防血栓、动脉硬化、中风和心脏病；改善记忆，健脑益智。

Prediction (same order as above)
结论：用户的基础上，可以及其他的影响，并且可以及其他的影响，并且可以及其他的影响。
减少血脂肪血，血血、、、、、心脏病、发生，提高，脑，，，脑，脑，脑，脑提高的脑效。
减减血血血血血，，，血，，，，、、、、、，，，，，，，，，，，，，，，，善善善善。。
减少血血，，，血、血、、血、、心脏、心脏病、心脏病、心心病率，心心率。
减减血血血血血血血、，、、、、、、、、、、、、、、、、、、、、、、、、、、高、，高，，高，高的的高。。。。。。。。
减少血血、，防防防防、、、、、、、、、、、心心心心心心心心心心，心心心，，，的的的的。。。。

The baseline model seemingly is able to generate a complete sequence, but the sequence is unrelated to the input sentence at all. In fact, after some inspections its generated samples, we find that the samples are perfect in completeness (no repeated punctuation), but are extremely poor in relatedness with the inputs. This is probably because the model only learns multiple one-step mappings at training. Hence, it is able to generate the next token depending mostly on the input token from the last step, and meanwhile ignoring the context vectors.

The rest of the models encounter greater difficulty in predicting tokens as time steps become larger. For the first sample, they are able to predict "第二", which corresponds to "The second" that commence the sentence. But then they start to generate repeated tokens until the end of the sentence. Similar situation happens in the second example, where the models are able to predict "减少血" which corresponds to "Reduce blood", but then they are not able to generate meaningful tokens.

5.4 Hidden Dimension

5.4.1 Setup

Since we find that using $p = 0.5$ yields the highest BLEU score, we continue using this teacher-forcing scheme in this section. We train 5 models with hidden dimensions 128, 256, 512, 1024, 2048 respectively. Note that we use the same hidden dimension for all layers, including embedding layers and GRU layers. Other settings follow the description in 5.1.

5.4.2 Results and Analysis

Based on our experiment results, when hidden dimension H (width of the network) is less than 1024, the validation loss drops steadily as the network grows wider. At the same time, the BLEU score

Dimension	#Params.	val loss	BLEU↑
128	6.6M	5.78	3.03
256	13.8M	5.72	4.71
512	29.7M	5.69	5.78
1024	68.4M	5.64	**7.13**
2048	172M	**5.63**	6.71

Table 2: The number of parameters, val loss and BLEU scores of models with different hidden dimensions.

increases as the network gets wider, attaining a peak at $H = 1024$. However, when H is larger than 1024, the drop of validation loss becomes trivial. Meanwhile, the BLEU score begins to go down, with the BLEU score of 2048-D model even lower than that of 1024-D model.

We suspect that this is because as the network grows wider, the bottleneck of performance becomes the depth of the model. Since our models all have only one hidden layer, too many hidden dimensions lead to an extreme width, resulting in an unbalanced network structure. Consequently, the performance is restrained.

Nevertheless, increasing the hidden dimension generally improves the model performance from the baseline by a large margin. From the observation and consideration above, we conclude an optimal hidden dimension around $H = 1024$ for a single-layer Bahdanau RNN.

6 Experiments on Vanilla Transformer

6.1 Setup

We adopt the `PyTorch` implementation of vanilla transformer (`torch.nn.Transformer`) for convenience. We use a combination of 5, 10, 15, 20 layers between encoders and decoders, resulting in 4*4=16 models, sharing the same hidden dimension D=768, 8 attention heads, max number of positions 256, and dropout rate 0.1. We use Adam optimizer with no weight decay, betas=(0.9, 0.98) and epsilon 1e-9. The learning rate is computed as

$$lr = \frac{1}{\sqrt{D}} * min(\frac{1}{step^{0.5}}, \frac{step}{warmup^{1.5}}),$$

where $step$ is the current step, and $warmup$ is set to 6000. Under this setting, the learning rate grows from 0 to 4.66e-4 in 6000 steps, and steadily decrease afterwards. This is indeed the same warmup strategy that is used in [3], except that we use 6000 warmup steps instead of 4000.

All models are trained on 3 V100 GPUs with a total batch size of 80*3=240. Due to the limited time and computational resources, we are not able to train our models on full training set considering the size of the models. Hence, we only train all models on 1M data that are randomly sampled from the 3M dataset. We use fp16 for training which we found causing no harm to performance while saving time and space.

6.2 Inference with Past Keys and Values

Inference process in auto-regressive architectures is costly. Since the decoder part is unidirectional, and each time step needs the information from the previous steps, the time complexity of inference is approximately $O(n^2)$, where n is the number of tokens. To reduce the cost of inference, we cache the encoder output, and keys and values of all past steps in the decoder. For every time step, we only send in the current token as the query to the Transformer decoder, append the new keys and values to the cached ones, and output the next token. Since `nn.Transformer` does not provide such features, we implement it ourselves by override the forward methods of decoder layers. We observe a 10x speed-up at inference using our implementation. The inference speed is similar to `model.generate(**kwargs, use_cache=True)` in HuggingFace's implementation.

$N_{decoder}$ / $N_{encoder}$	5	10	15	20
5	12.34	13.59	13.80	13.30
10	13.22	13.04	14.63	11.89
15	13.53	12.90	12.77	14.92
20	13.30	14.011	12.80	14.18

Table 3: The BLEU scores of models with different number of encoder and decoder layers.

$\Delta = N_{encoder} - N_{decoder}$	$N_{encoder}$	$N_{decoder}$	BLEU	Subgroup Mean	Group Mean
0	5	5	12.34	13.08	13.08
	10	10	13.04		
	15	15	12.77		
	20	20	14.18		
5	10	5	13.22	12.97	13.67
	15	10	12.90		
	20	15	12.80		
-5	5	10	13.59	**14.38**	
	10	15	14.63		
	15	20	**14.92**		
10	15	5	13.53	13.77	**13.83**
	20	10	14.01		
-10	5	15	13.80	13.90	
	20	10	14.01		
15	20	5	13.30	13.30	13.30
-15	5	20	13.30	13.30	

Table 4: The BLEU scores of models of different number of encoder and decoder layers, with Δ, the difference in numbers of encoder and decoder layers. Bolded numbers are the best within that column.

6.3 Results and Analysis

Deeper network is not necessarily better. Table 3 summarizes the BLEU scores of each model. Unsurprisingly, the most shallow model $N_{encoder} = N_{decoder} = 5$ performs the worst. However, there is not a significant increasing trend in each row or each column, meaning that a deeper model does not necessarily outperform its shallower counterpart. This could probably due to the hyperparameter settings and limited training time. Another possible reason is that, as the model goes deeper, the bottleneck of performance becomes other component of the model. For example, the width of the model, the usage of a fixed embedding instead of a learnable one, number of attention heads, etc. We leave this part for future studies since it requires a lot of ablation studies.

Vanilla Transformer benefits from slight imbalance. Table 4 shows how imbalance of layers affects the performance. A larger $|\Delta|$ means a greater imbalance between encoder and decoder layers. The Group means of the three groups with $\Delta \neq 0$ are all higher than the group $\Delta = 0$. Even if we remove the lowest BLEU score 12.34, the rest three scores in group $\Delta = 0$ yields a mean of 13.33, which is still lower than group $|\Delta| = 5$ and $|\Delta| = 10$. This indicates that vanilla Transformer benefits from a slight imbalance in its encoder and decoder. The performance drops from $|\Delta| = 10$ to $|\Delta| = 15$, since the discrepancy of ability between both sides could harm the information flow from encoder to decoder.

Decoder is a more powerful module than encoder. The subgroup mean column in Table 4 shows the mean of BLEU in each subgroup. We observe that for $|\Delta| = 5$ and $|\Delta| = 10$, the subgroup mean of $\Delta < 0$ is higher than $\Delta > 0$. Hence, given approximately the same number of parameters, using slightly more decoder layers than encoders could potentially improve overall performance. This is probably because when performing NMT tasks, the decoder is responsible for generating answers that

will be directly sent to compute BLEU scores. The cross attention and causal mask play important roles in merging information from the encoder and predicting future tokens, respectively.

7 Experiments on Chimera and Phoenix

We also generate translations using the the two language models released by our university [8]. We do not make comparison across different LLMs because the cost of inference is relatively high even if we use fp16. Since we have access to Chimera and Phoenix, we would like to demonstrate the power of them in this translation task.

7.1 Setup

We design a simple prompt for this task: *Please translate the following sentence into Chinese: {sentence}*. This prompt is merged with the system prompt and is then sent into the model for prediction. Hence, the complete prompt for a model is:

> *A chat between a curious human and an artificial intelligence assistant. The assistant gives helpful, detailed, and polite answers to the human's questions.\n\nHuman: <s> Please translate the following sentence into Chinese: {sentence}</s>Assistant: <s>*

As for sampling parameters, We set the temperature to 0.7, topk=50. Other hyperparameters follow the default values in HuggingFace's `transformers.GenerationConfig` [1].

7.2 Results and Analysis

Model	#Params.	BLEU↑
chimera-chat-7b	7B	14.24
chimera-inst-chat-7b	7B	16.86
chimera-chat-13b	13B	16.52
chimera-inst-chat-13b	13B	19.33
phoenix-chat-7b	7B	18.91
phoenix-inst-chat-7b	7B	**19.87**

Table 5: The number of parameters and BLEU scores of Chimera and Phoenix models.

Phoenix outperforms Chimera. Table 5 shows that Phoenix models outperform Chimera models in terms of BLEU. Although phoenix-inst-chat-7b slightly underperforms chimera-inst-chat-13b, the former has only half number of parameters compared to the latter. Since BLEU is calculated in Chinese, Chimera may not be able to achieve high scores because it is pretrained on Latin languages.

Models with instruction tuning achieve higher scores. We can see that all models with instruction tuning outperform their counterparts. This is because with instruction tuning, models are able to better understand human instructions and can generate results with higher quality.

8 Conclusion

In this project, we find that the teacher-forcing ratio is a key factor to the success of training Bahdanau RNN. A wider network also improves its final performance. In the vanilla Transformer architecture, we find that the model benefits from slight imbalance in encoder/decoder layers. Given a fixed total number of layers, we can use more layers of decoder than encoder to achieve a better performance. The outstanding performance of Phoenix and Chimera indicate their ability of understanding human instructions and generating high quality results.

[1] https://huggingface.co/docs/transformers/main_classes/text_generation#transformers.GenerationConfig

The major challenges we met was in tuning hyperparameters when training. This report only shows around 20% of experiments we did. Luckily, we were able to find a set of sound hyperparameters and to product reasonable results.

The limitation of this work is that more ablation studies are needed to have a solid conclusion from the results. For example, the training time, batch size, number of attention heads etc. can all affect the performance of Transformer models. However, since we do not have enough computational resources, we will leave this as our future work.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[2] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28, 2015.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[6] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[7] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[8] Zhihong Chen, Feng Jiang, Junying Chen, Tiannan Wang, Fei Yu, Guiming Chen, Hongbo Zhang, Juhao Liang, Chen Zhang, Zhiyi Zhang, et al. Phoenix: Democratizing chatgpt across languages. *arXiv preprint arXiv:2304.10453*, 2023.

[9] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[10] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[11] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[12] Mozhdeh Gheini, Xiang Ren, and Jonathan May. Cross-attention is all you need: Adapting pretrained transformers for machine translation. *arXiv preprint arXiv:2104.08771*, 2021.

[13] Yuzhu Ji, Haijun Zhang, Zequn Jie, Lin Ma, and QM Jonathan Wu. Casnet: A cross-attention siamese network for video salient object detection. *IEEE transactions on neural networks and learning systems*, 32(6):2676–2690, 2020.

[14] R Gnana Praveen, Wheidima Carneiro de Melo, Nasib Ullah, Haseeb Aslam, Osama Zeeshan, Théo Denorme, Marco Pedersoli, Alessandro L Koerich, Simon Bacon, Patrick Cardinal, et al. A joint cross-attention model for audio-visual fusion in dimensional emotion recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2486–2495, 2022.

[15] Zhengyao Wen, Wenzhong Lin, Tao Wang, and Ge Xu. Distract your attention: Multi-head cross attention network for facial expression recognition. *arXiv preprint arXiv:2109.07270*, 2021.

[16] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[17] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[18] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[19] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[20] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[21] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *ArXiv*, 2021.

[22] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335, 2022.

[23] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

[24] Ji Yunjie, Deng Yong, Gong Yan, Peng Yiping, Niu Qiang, Zhang Lei, Ma Baochang, and Li Xiangang. Exploring the impact of instruction data scaling on large language models: An empirical study on real-world use cases. *arXiv preprint arXiv:2303.14742*, 2023.

[25] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, 2002.