6 ADDITIONAL RESULTS

6.1 REINFORCEMENT LEARNING

Maze Navigation Easy. We experiment with a slightly different version of the Maze Navigation task. Instead of an agent with forward, turn-left and turn-right actions, the agent has no orientation and there are only 4 movement actions corresponding to 4 cardinal directions. This makes navigation easier because the agent do not need to keep track of its orientation. Further, it is much easier to compute relative locations given a history of actions. This might explain why standard Transformers are not far behind Feedback Transformers in performance as shown in Figure 6 (left). We also compare to LSTMs, which performs much worse. See Section 7.2 for more implementation details.

Water Maze. We modify the Morris Water Maze task (Morris, 1981) to make it more challenging. The maze is defined by a goal position and a mapping of cell to ID — these remain fixed within an episode but change between episodes. The agent receives as an observation the cell IDs of its current location and the target cell. When the agent finds the target, it receives +1 reward and is randomly teleported. During the same episode, if the agent reaches a previously seen cell, it needs to remember how it reached the target from there to go back. Results are shown averaged over 10 trials (the reward is reported averaged over the last 500 episodes for each trial). As shown in Figure 6 (right), the Feedback Transformer converges to higher average reward.

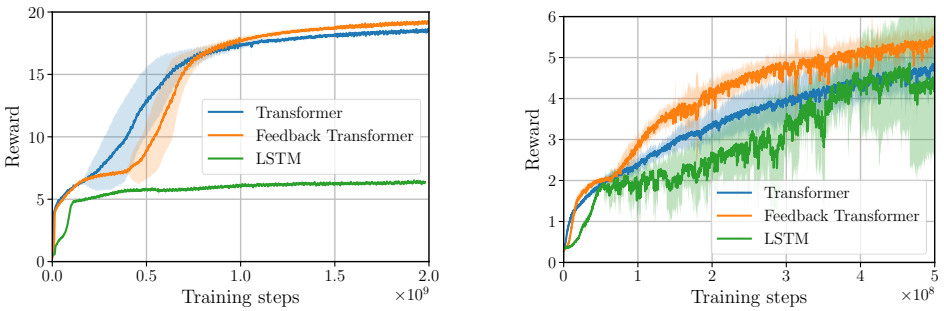

Figure 6: Averaged cumulative reward during training on **(left) Maze Navigation Easy** and **(right) Water Maze** tasks.

6.2 IWSLT DE-EN

We additionally evaluate the Feedback Transformer on `IWSLT De-En`, a small machine translation dataset. We train a small Transformer model with 6 layers. For generation, we use beam size 5 without checkpoint averaging. Model quality is evaluated using tokenized BLEU. Results are shown in Figure 7 (left) and show that for shallower models, the Feedback Transformer has better performance than the standard Transformer.

6.3 SUMMARIZATION ON CNN-DAILYMAIL

We evaluate on the CNN-Dailymail multi-sentence summarization benchmark of 280K news articles Hermann et al. (2015), modeling the first 400 words of the article See et al. (2017). We evaluate using ROUGE Lin (2004). and use 3-gram blocking and tune length Fan et al. (2017). Figure 7 (right) displays the performance of the Feedback Transformer as the decoder layers are reduced, making the model *shallower only*. For all model depths, the Feedback architecture maintains a consistent improvement in ROUGE compared to the standard Transformer. Compared to sentence-level tasks such as translation, this summarization benchmark requires multi-sentence generation, and the increased capacity of the Feedback architecture is beneficial.

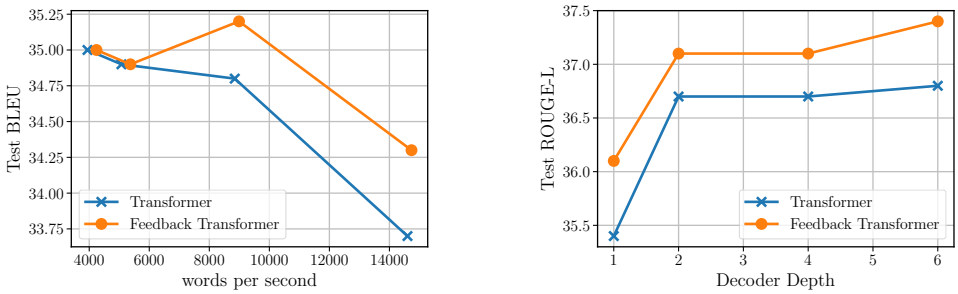

Figure 7: Results on **(left) the `IWSLT De-En` dataset,** and **(right) Summarization on `CNN-Dailymail`,** test set ROUGE-L for varying decoder depths.

Model	Pre-norm + higher LR	Adapt. span	Increase BPTT	dev ppl
Transformer	no	no	no	22.9
Transformer	no	no	yes	22.9
Transformer	yes	no	yes	21.0
Transformer	yes	yes	no	20.6
Feedback	no	no	no	19.7
Feedback	no	no	yes	19.9
Feedback	yes	no	yes	19.6
Feedback	yes	yes	yes	19.0

Table 5: Ablation on `WikiText-103` of various modeing choices. Results are shown without finetuning.

6.4 Ablation Studies on Language Models

Here we study how different techniques affect the model performance on `WikiText-103`. The results shown in Table 5 indicate:

- Pre-normalization combined with higher learning rates helps the performance, particularly for the standard Transformer.
- Increasing the context size with adaptive span further improves the performance for both models.
- The technique of increasing the BPTT length during training for efficiency does not affect the final performance.
- The gap between two model is consistent along those variations.

Next, we examine the effect of the model depth on performance on `char-PTB` and `WikiText-103` This time, we keep the total number of parameters constant and only vary the number of layers to isolate the effect of depth. This is achieved by proportionally increasing the head dimension and the ReLU layer size when we decrease the number of layers. The results in Figure 8 demonstrate that for the standard Transformer improves as the depth increase. In contrast, the Feedback architecture is much robust reduced depth, even achieving the best performance on `char-PTB` with only two layers.

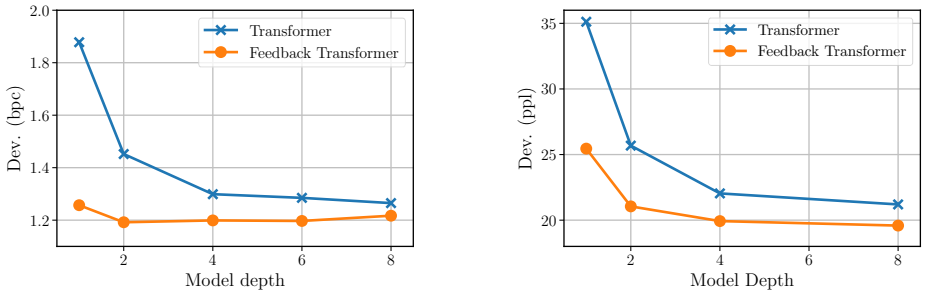

Figure 8: The performance on **(left)** `char-PTB` and **(right)** `Wikitext-103` as a function of the model depth. The number of parameters is kept constant by increasing the width.

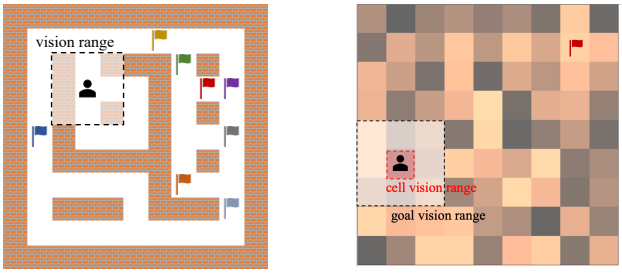

Figure 9: **(left) Maze Navigation** task and **(right) Water Maze** task.

7 Additional Implementation Details

7.1 Random Walk Task Details

We provide additional details for the random walk toy task we explore. The agent starts at a fixed position of a 8×8 grid. Available actions are 1) move one step forward, 2) turn left and 3) turn right. At every time step, the agent randomly picks on of the three actions and executes it. An action would be ignored if it can't be executed like going out of the grid. After 100 actions, the agent is reset back to the initial position.

The input to the model is a sequence of actions taken by the agent, and a special symbol if there was a reset. The output is a sequence of location symbols corresponding to the agent's location after each action. We generate 10k training episodes, totalling 1M tokens.

We use the same setup as our language modeling experiments, except now the model predicts separate output tokens rather than a next token. We concatenate all the episodes and feed them to the model as a single sequence. The training is done with the negative-log-likelihood loss. See Table 7 for the hyperparameters used in the experiment. The attention span is set to 100, so that the models can attend to all the information they needs to solve the task.

7.2 Maze Navigation Details

We generate random 9×9 mazes using Kruskal's algorithm. Dead ends are eliminated by randomly removing one of the blocks surrounding them. We randomly place 8 target objects with different colors as shown in Figure 9 (left). The agent is given a randomly selected color as a target. If the agent manages to reach the correct target, it gets a reward of $+1$ and a new target color is sampled. An episode ends after 200 steps. The observation includes the 3×3 area around the agent and target color.

We train 2-layer Transformers with a hidden size 256 and 4 heads. We set the BPTT to 100 and the batch size to 1024. The reward discount rate is 0.99. The attention span is 200 so the agent can keep an entire episode in memory. All agents were trained using A2C with Adam with a learning rate of 0.0003 and a entropy cost of 0.0005. For the easy version of the task, we use RMSprop with a batch

Hyperparameter	Summarization	WMT En-De	IWSLT De-En
Encoder Layers	6	6	6
Decoder Layers	6	6	6
FFN Size	2048	4096	1024
Attention Heads	8	16	4
Dropout	0.3	0.3	0.3
Hidden Size	512	1024	512
Learning Rate	0.0005	0.001	0.0005

Table 6: Hyperparamers for sequence to sequence experiments.

Hyperparameter	Random Walk	char-PTB	Enwik8	WikiText-103 small	WikiText-103 large
Layers	4	6	12	4	8
Hidden size (d)	256	384	512	512	1024
FF size	$4d$	$4d$	$8d$	$8d$	$4d$
Head count (h)	4	4	8	8	8
Head dim	d/h	d/h	$2d/h$	$2d/h$	d/h
Attention span	100	512	8192*	512	512, 2048*
Dropout rate	0.2	0.5	0.5	0.1	0.3
Embed. dropout	-	-	-	0.1	0.2
BPTT len (M)	64	128	128	256	256
Batch size (B)	512	2048	1024	512	512
Learning rate	0.0001	0.0015	0.0015	0.0007	0.0007
Gradient clip	0.1	1.0	0.1	0.1	0.1
LR warm-up steps	1k	1k	8k	8k	8k
Parameters	3.2M	10.7M	77M	44M	139M

Table 7: Hyperparamers for language modeling experiments. Here * indicates the adaptive span.

size of 128 and a learning rate of 0.0003. The RMSProp epsilon regularization parameter is set to 0.01 The LSTM model is a 3-layer LSTM with a hidden size of 256.

7.3 Water Maze Details

The water maze task we designed is depicted visually in Figure 9 (right). The grid size is 15×15. To help exploration, the agent can see if the goal is within a 3×3 area around it. An episode ends after 200 steps. We train for 500M steps (2.5M episodes). We use 2-layer Transformers with hidden size of 64 and 1 head. The attention span is 200 so the agent can put an entire episode in memory.

All agents where trained using A2C with RMSprop with entropy cost of 0.0001, RMSProp epsilon regularisation parameter of 0.01, batch size of 64, and BPTT 200. Feedback Transformer and Transformer baseline were trained with a learning rate of 0.0003. LSTM model is a 2-layer LSTM with hidden size of 64. For LSTM model we used a learning rate of 0.0004.

7.4 Machine Translation and Summarization

We detail the hyperparameters in Table 6. Summarization experiments are done with the Transformer base architecture size and WMT En-De experiments are done with the Transformer big architecture size. As IWSLT De-En is a smaller dataset, we use a smaller model. For all sequence to sequence experiments, only the decoder is modified to have the Feedback Transformer architecture.

7.5 Language Modeling

In the language modeling experiments, we added several improvements on top of the original Transformer Vaswani et al. (2017) to better adapt to unbounded sequences:

- **Hidden representation caching Dai et al. (2019):** Since the input to the model is an unbounded sequence and the model needs to process it in small blocks, hidden representations from previous blocks are kept in cache so that any token in the current block will the same context length regardless of its position in the block.
- **Relative position embedding Shaw et al. (2018):** Relative position embeddings allow each token in a block to be processed in the same way regardless of its absolute position in the block. We found that adding shared embeddings to key vectors at every layer to be effective.
- **Adaptive attention span Sukhbaatar et al. (2019)** Language modeling requires a model to have a very long attention span, which is computationally expensive. The adaptive span mechanism allows each attention head to learn different attention spans for efficiency.
- **Pre-normalization Child et al. (2019)**: We observed that pre-normalization makes training more stable for Transformers, which allowed us to use larger batch sizes for better parallelization.

Dropouts are applied to attention and ReLU activations. In `WikiText-103` models, additional dropouts are added to the embedding layer output and the last sublayer output.

In Table 7, we present the hyperparameter values used for our experiments. We use the same hyperparameters for both Transformers and Feedback Transformers, and optimize them with Adam. The final performances are obtained by finetuning the models with a 10x smaller learning rate.

Details on the `char-PTB` experiments We trained the models for 15k updates (or earlier if the validation loss stops decreasing), and funetined them for 1k steps. We varied the depth of the models while keeping the number of parameters constant. This is achieved by changing the FF size and the head dimension inverse proportionally to the depth.

Details on the `enwik8` experiments We used an adaptive span limited to 8192 tokens with a loss of 0.0000005. The training is done for 100k updates and another 10k steps is used for finetuning. The warming up BPTT length is used for speeding up the training, where the BPTT length is decreased to 64 for the first half of the training.

Details for Training on `WikiText-103` We employed the adaptive input Baevski & Auli (2019) and the adaptive softmax Grave et al. (2017) techniques for reducing the number of parameters within word embeddings. The models are trained for 200k steps and the finetuned for additional 10k steps.

While most of the models have a fixed attention span of 512, the best performance is achieved by extending the attention span to 2048 with adaptive span loss 0.00001.

After training our models, we noticed that our tokenization method differed from others by omitting end-of-line (EOL) symbols. Since our dictionary already contained the EOL token, we were able finetune our trained models on the data with EOL tokens, rather than training them from scratch. This change alone brought about 1ppl improvement.