Meta-Learning Fast Weight Language Models

Kevin Clark **Kelvin Guu** **Ming-Wei Chang**
Panupong Pasupat **Geoffrey Hinton** **Mohammad Norouzi**

Google Research

{kevclark,kguu,mingweichang,ppasupat,geoffhinton,mnorouzi}@google.com

Abstract

Dynamic evaluation of language models (LMs) adapts model parameters at test time using gradient information from previous tokens and substantially improves LM performance. However, it requires over 3x more compute than standard inference. We present Fast Weight Layers (FWLs), a neural component that provides the benefits of dynamic evaluation much more efficiently by expressing gradient updates as linear attention. A key improvement over dynamic evaluation is that FWLs can also be applied at training time so the model learns to make good use of gradient updates. FWLs can easily be added on top of existing transformer models, require relatively little extra compute or memory to run, and significantly improve language modeling perplexity.

1 Introduction

A key challenge in language modeling is representing the contextual information from previous tokens. Transformer language models use attention to pass along this information, but constantly referring back to the previous text is a cognitively implausible model of working memory. An appealing alternative is using fast weight neural networks (Hinton and Plaut, 1987; Schmidhuber, 1992). Inspired by short-term plasticity in the brain, these models have parameters that change on-the-fly based on input data (previous tokens for LMs) in addition to standard "slow" weights learned during training. Fast weights have proven successful for supervised (Ba et al., 2016a), reinforcement (Munkhdalai et al., 2019), and few-shot (Munkhdalai and Yu, 2017) learning.

Dynamic evaluation (Mikolov et al., 2010; Krause et al., 2018) uses a variant of fast weights to improve language models at inference time. After scoring (or generating) a chunk of text, dynamic evaluation applies a gradient update to the model coming from the LM loss over that chunk before continuing. Intuitively, this process improves performance because an update that makes the model better at predicting previous tokens will likely also make it better at predicting future ones. Dynamic evaluation substantially improves LM perplexity, but has numerous drawbacks. It requires an extra forward and backward pass through the model to compute the gradients, and the sequential gradient updates over chunks cannot be parallelized. Furthermore, it is very memory intensive because a separate copy of the evolving model weights has to be stored for each example in a minibatch. Lastly, dynamic evaluation is only used at test-time, so the model does not learn to make good use of gradient updates.

We present Fast Weight Layers (FWLs), a neural component that provides the benefits of dynamic evaluation with none of these downsides. They can be added to any LM without requiring changes to the training or evaluation loop. Like dynamic evaluation, FWLs also update their parameters using gradient information from previous tokens, but FWLs employ three key ideas to improve efficiency. First, FWLs are added on top of the transformer after the last attention layer, which avoids having to backpropagate through the whole transformer and circumvents the complexities of backpropagation through time. Secondly, FWLs compute gradients in parallel rather than recurrently. Lastly, FWLs leverage the property that gradient matrices are rank one to compute their outputs efficiently.

Crucially, the efficient design means FWLs can be used at training time so the model learns to obtain beneficial updates from previous gradients. Another benefit is that while dynamic evaluation needs hyperparameter search to find a good step size for the gradient update, FWLs can instead learn the step size. Training FWLs can be viewed as applying gradient-based meta-learning (Finn

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 9751 - 9757

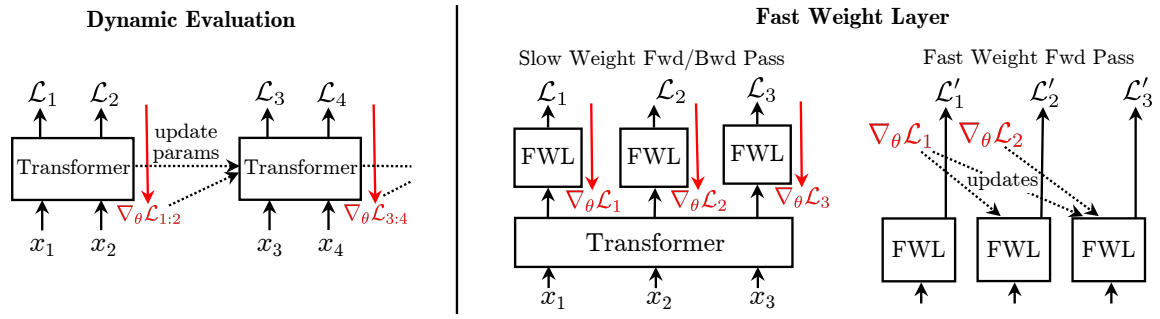

Figure 1: Dynamic evaluation (left) recurrently processes chunks of text and applies the gradient updates to model weights. FWLs (right) compute the gradients in parallel and for only a small subset of model parameters, greatly increasing efficiency. The improved speed allows FWLs to be used at train time, making them more effective.

et al., 2017) to language modeling, where the support set contains tokens seen so far and the query set contains future tokens, a perspective which helps explain some of the behaviors of FWLs.

FWLs scale well to long sequences and are complementary to existing long-text generation methods such as sparse attention (Child et al., 2019) or recurrent processing (Dai et al., 2019). We evaluate FWLs at language modeling on the WikiText-103 dataset (Merity et al., 2017). They substantially improve perplexities over strong baselines; for example lowering Transformer-XL's perplexity from 18.1 to 16.6. This gain is comparable to the one from dynamic evaluation, but FWLs add less than 20% compute overhead to the model compared to the over 200% of dynamic evaluation. Ablations show FWLs achieve superior compute vs perplexity trade-offs compared to alternative fast weight methods. We also analyze how FWLs improve results and find they are especially effective at modeling rare tokens, repeated tokens, and long documents.

2 Method

Our models first run a transformer decoder (or other left-to-right neural network) over an input sequence $[x_1, ..., x_T]$, yielding vector representations $[h_1, ..., h_T]$. FWLs use a learned function f_θ with parameters θ to further process the text. These parameters (and the transformer's parameters) are "slow" weights learned during training. We chose to use a hidden layer, projection layer, and LayerNorm (Ba et al., 2016b) for f_θ, which worked well in initial experiments:

$$f_\theta(h_t) = \text{LayerNorm}(\text{ReLU}^2(h_t U + a)W + b)$$

The FWL's output can be passed into a prediction layer such as an output softmax:

$$\mathcal{L}_t = \text{CE}(\text{softmax}(f_\theta(h_t)E + c), x_{t+1})$$

where E is an embedding matrix, c is a bias vector, and CE denotes cross-entropy loss. FWLs also work with adaptive softmax layers (Grave et al., 2017), such as the one used by Transformer-XL.

In addition to the slow weights, an FWL also employs "fast" weights that let it adapt to contextual information. First, the FWL does a forward and backward pass over the input sequence with its slow weights θ to obtain a gradient $\nabla_\theta \mathcal{L}_t$ from each position t. Importantly, these T gradients can be computed in a single backwards pass because $\mathcal{L}_t$ depends only on h_t. If we added attention or other temporal dependence to the FWL, backpropagation through time would instead only compute the summed gradient $\sum_{t=1}^{T} \nabla_\theta \mathcal{L}_t$. This limitation is why dynamic evaluation has to process the input in chunks, which means the model does not get updates from recent tokens within the same chunk; FWLs do not have this drawback.

The FWL then does a second forward pass using evolving fast weights θ'. The fast weights are initialized with the slow weights θ. Conceptually, the second pass processes the sequence in a left-to-right order, although we will show this can be parallelized in practice.[1] For each position t, it re-runs the FWL and output softmax on h_t, but now using θ' instead of θ to produce a new loss $\mathcal{L}'_t$. Then the FWL updates its fast weights as $\theta' \leftarrow \theta' - \alpha \circ \nabla_\theta \mathcal{L}_t$ where α consists of learned step sizes. We learn a separate step size for each weight matrix/vector in the FWL, which performs slightly better than having one global step size.

[1] Generating with a LM is inherently recurrent, but FWLs avoid recurrence when training the model or scoring text.

Although we tried more sophisticated update rules such as one based on Adam, they did not outperform this simpler one in initial experiments. An overview of FWLs is shown in Figure 1.

Efficient Fast Weight Computation. We will now show how the FWL outputs can be computed without needing to store the T gradients of $\mathcal{L}_{1:T}$ in memory (which would have prohibitive memory requirements) or recurrently computing θ' (which would be slow on modern accelerators). We will illustrate this for the second matrix-multiply in f_θ; other parts of the FWL can be computed analogously. We use v_t to denote the input, o_t to denote the output, and the $'$ symbol to differentiate activations in the second (fast weight) forward pass from those in the first (slow weight) pass.

The slow weight output at position t is simply $o_t = v_t W$. The fast weight output at position t is

$$o'_t = v'_t(W - \alpha_W \sum_{i<t} \nabla_W \mathcal{L}_i) \tag{1}$$

where the weight matrix W has been updated by the $t-1$ previous gradients.

Computing (1) naively is infeasible because it requires storing all T gradient matrices in memory. As an alternative, we develop a more efficient method by taking advantage of the observation that the gradient is equal to the outer product of the input and the upstream gradient: $\nabla_W \mathcal{L}_i = v_i^T \nabla_{o_i} \mathcal{L}_i$ (from the chain rule; see for example Appendix D of Mitchell et al. (2022) a derivation). Therefore we can rewrite (1) as

$$o'_t = v'_t W - \alpha_W \sum_{i<t} v'_t v_i^T \nabla_{o_i} \mathcal{L}_i \tag{2}$$

The first term is just a regular matrix-multiply and can be computed easily. Interestingly, the sum in the second term can be interpreted as linear attention (i.e., without a softmax) using query v'_t, keys v_i, and values $\nabla_{o_i} \mathcal{L}_i$. The partial sum up to t corresponds to causal masking in the attention.

Computing the FWL output using (2) means we only need to store gradient vectors $\nabla_{o_i} \mathcal{L}_i$ in memory (which is no worse than storing model activations) rather than gradient matrices $\nabla_W \mathcal{L}_i$. It also can be computed fully in parallel using causal attention. However, naive attention scales quadratically with sequence length, which is an obstacle for applying (2) to long documents. Luckily, because the attention is linear, there exist more efficient methods for computing it. In particular, we use the mixed chunk attention method from Hua et al. (2022), which improves efficiency through breaking the document into chunks while still computing the attention scores exactly.

Viewing the fast weight output as linear attention makes it clearer how FWLs work. For each previous position i, the error signal $\nabla_{o_i} \mathcal{L}_i$ is the direction for adjusting the output o_i to lower the loss, and the attention score $v'_t v_i^T$ measures the similarity between position i and the current position. This means FWLs adjust the current output in a way that mimics the loss-reducing directions of similar past positions.

Computing fast weights for vectors (e.g. biases or LayerNorm weights) rather than matrices is simpler because the gradients require less memory. For example, the fast weight output after adding the bias term b to o_t is $o'_t + b - \alpha_b \sum_{i<t} \nabla_b \mathcal{L}_i$. This can be computed in parallel for all t using the `cumsum` operation over the matrix of gradients $[\nabla_b \mathcal{L}_1, ..., \nabla_b \mathcal{L}_T]$, which can be obtained using standard backpropagation and stored in memory.

The parallelized FWL updates are not usable during generation, which inherently has to produce one token at a time. Instead the model at each step (1) samples $\hat{x}_t$ from its output distribution using the current weights θ', (2) computes the gradient $\nabla_\theta \hat{\mathcal{L}}_t$ of the slow weights predicting $\hat{x}_t$, and (3) updates the fast weights as $\theta' \leftarrow \theta' - \alpha \circ \nabla_\theta \hat{\mathcal{L}}_t$. This process is still relatively efficient because the additional backward/forward passes are only applied to the FWL, not the whole transformer.

FWL gradient updates may first seem strange in that the model is "training" on test sequences during perplexity evaluation or on its own outputs during generation. We would like to emphasize that this way of conditioning on previous tokens is just as valid as standard transformer LMs attending over gold-standard tokens (from teacher forcing) or their own generated tokens (during sampling). Indeed, (2) shows that FWLs can be viewed as adding another attention layer to the model, but with a pre-specified function computing the values (the gradient) rather than a learned matrix-multiply. Empirically, we did not find FWLs to produce degenerate repetitive outputs (Holtzman et al., 2020) more than baselines.

Training. Training jointly optimizes the transformer, softmax, and FWL parameters, as well

as the step sizes α, to minimize the combined loss $\sum_{t=1}^{T} \mathcal{L}'_t$ over training sequences. We compute second-order gradients for the FWL parameters, backpropagating through the gradient updates $\nabla_\theta \mathcal{L}_t$. Intuitively, this means that in addition to learning to *expect* the gradient update and adapt quickly, the model also learns to *produce* effective gradient updates. While second-order gradients are expensive to compute for some models, they are not for FWLs because there is no backpropagation through time for the FWL parameters.

Runtime. FWLs add relatively modest compute and memory overhead to the model. In our experiments, FWLs add <30% overhead in FLOPs and <20% in wall clock time to both sparse transformers and Transformer-XL when scoring text perplexity. The majority of this extra compute comes from the two extra passes through the output layer (the backward pass and fast weight pass), although using an adaptive softmax somewhat alleviates this cost. In contrast, dynamic evaluation adds over 200% extra compute due to needing an extra backward and forward pass through the whole transformer.

Connection to Meta-Learning. FWLs can be viewed as applying gradient-based meta-learning to LMs. Specifically, language modeling is treated as a few-shot learning task where the support set contains the tokens seen so far and the query set contains the next token. The FWL training is essentially using MAML (Finn et al., 2017), where there is a single inner loop optimization step over the support set that adapts the model parameters using a gradient update. This connection helps explain a surprising property of FWLs: although it applies the fast weight gradient update to only a small subset of the network, it performs comparably to dynamic evaluation, which updates all transformer parameters. Raghu et al. (2020) show that MAML works just as well when the inner-loop update is only applied to the last layer of the network, which is similar to how FWLs only update a few weights on top of the transformer.

Connection to Fast Weight Programmers (FWPs). FWPs use a slow-weight neural network to generate fast weights for another net (Schmidhuber, 1992; Irie et al., 2021). Schlag et al. (2021) show that FWPs can be viewed as transformers with linear attention, similar to the linear attention FWLs employ. However, FWLs use gradient information to provide the update rather than generating the update from a network.

Method	PPL	Tok/s
Compressive Transformer	17.1	–
Routing Transformer	15.8	–
kNN-LM	15.8	–
Transformer XL	18.1	1575
Transformer XL + Dynamic Eval	16.4	510
Transformer XL + FWL (ours)	16.6	1340

Table 1: Test set WikiText-103 perplexities and inference speeds (on one V100 GPU).

3 Experiments

We experiment on WikiText-103 (Merity et al., 2017), a standard benchmark for language modeling consisting of approximately 100M tokens from English Wikipedia. We report perplexity using the standard tokenization and splits. We consider two baseline models: the sparse local attention transformer from Roy et al. (2021) and Transformer-XL (Dai et al., 2019). The sparse transformer model has 12 layers with 768 hidden units (121M parameters). For Transformer-XL, we use the large model (257M parameters).

Transformer-XL processes the text recurrently: at each step it trains on one chunk while attending over but not backpropagating into a previous chunk.[2] We use an analogous trick with FWLs to efficiently use them with the recurrent model: we keep a running fast weight update Δ_θ from the tokens in previous chunks and similarly don't backpropagate into this update. More specifically, when processing a chunk of text $[x_S, x_{S+1}, ..., x_T]$ the fast weight output o_t is

$$o'_t = v'_t(W - \alpha_W \Delta_W - \alpha_W \sum_{i=S}^{t} \nabla_W \mathcal{L}_i)$$
$$\Delta_W \leftarrow \gamma_W \Delta_W + \text{stopgrad}(\sum_{i=S}^{T} \nabla_W \mathcal{L}_i)$$

where γ_W is a learned decay factor. To reduce the training cost, we add the FWLs on top of the publicly released pre-trained Transformer-XL model and then fine-tune for 20K steps rather than training it from scratch.

Main Results. Table 1 shows the results on the WikiText-103 test set. FWLs improve

[2]Unfortunately, this recurrence means we lose the parallelism advantage of FWLs, although the other benefits over dynamic evaluation remain.

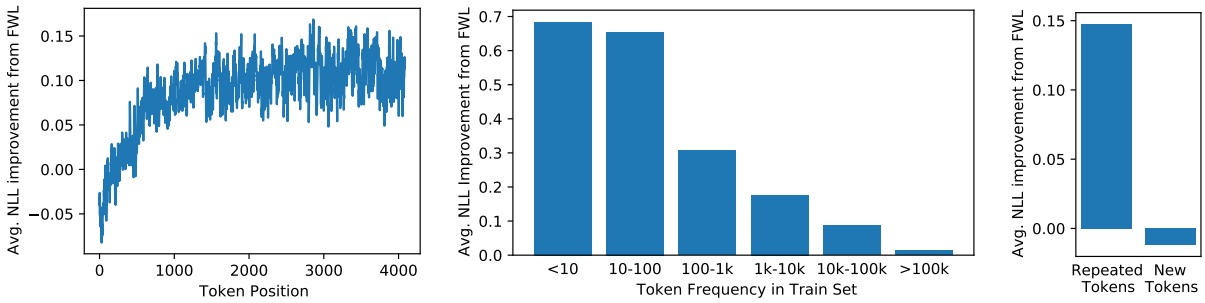

Figure 2: Per-token negative-log-likelihood improvements over the baseline. FWLs most improve LMs on long documents, rare tokens, and tokens repeated multiple times in the text. As the WikiText-103 dev set is small, we use a sparse transformer trained on 2/3 of the train set and evaluated on the other 1/3 to produce more robust results.

Transformer-XL by 1.5 perplexity points. This gain is comparable to the improvement from dynamic evaluation obtained by Krause et al. (2019), but FWLs are about 3x faster to run. While recent methods such as Routing Transformer (Roy et al., 2021) and KNN-LM (Khandelwal et al., 2020) achieve better perplexity, they are not directly comparable because they use different base models; we expect FWLs could also be combined with them to improve results.

Ablations. We compare different variants of FWLs in Table 2. First we consider only training the slow weights of the FWL (i.e., using $\mathcal{L}_t$ instead of $\mathcal{L}'_t$), but then applying the fast weight update at test time.[3] This method essentially applies dynamic evaluation to only a few layers of the network. While not performing as well as full dynamic evaluation, it still provides a sizable improvement given the small number of updated parameters and much faster inference speed. The remaining gap to dynamic evaluation is closed when the FWL is used during training so that the model learns to benefit from the previous gradients.

One hypothesis for the benefit of FWLs is that it biases the model towards copying seen tokens. To test this, we train a much simpler version of FWLs where only the bias term of the output softmax has fast weights applied. We found this did not significantly improve results, perhaps because attention is sufficient for the model to do this kind of copying, suggesting that FWLs are learning more complicated updates than just biasing the model towards repeating seen tokens.

Where do FWLs help? Figure 2 shows which tokens are better predicted by a FWL-augmented model. First, we find improvements are larger for tokens toward the end of input, implying that FWLs help models make use of long contexts and work best on long documents. Intuitively, more previous tokens will provide a better gradient estimate, similar to how meta-learning methods benefit for a larger support set. Next, we find FWLs help most on rare tokens, perhaps because they require better modeling of long contexts to predict. Lastly, we find FWLs actually make the model slightly worse at predicting a token the first time it appears in a text, but help when the token has occurred previously (a net gain because around 70% of tokens in WikiText-103 are repeats).

Method	PPL Sparse	PPL XL
No FWLs	25.1	17.3
FWLs	22.4	15.9
Test-time only	24.1	16.7
Bias Only	25.0	17.3
Dynamic Evaluation	22.4	15.8

Table 2: Dev set WikiText-103 perplexities for various ablations on **Sparse** transformer and Transformer-**XL**.

4 Conclusion

Fast Weight Layers provide the benefits of dynamic evaluation at a fraction of the compute cost and memory usage. They can easily be added to existing language models and yield strong results on language modeling benchmarks. Applying FWLs to few-shot learning tasks is one interesting future direction: doing one (or perhaps a small number) of gradient updates on few-shot examples might offer a nice middle ground in-context learning where the model parameters are fixed and full fine-tuning. Indeed, Yoshida and Gimpel (2021) show that hidden state optimization, a method closely related to dynamic evaluation, can improve few-shot LM performance.

[3] We use a global step size that is tuned on the dev set.

5 Limitations

FWLs can be viewed as an inductive bias encouraging the model to adapt to previous tokens. As an inductive bias, their value may be limited for larger models trained on larger datasets. While our experiments show FWLs improve models with hundreds of millions of parameters, initial experiments with bigger models suggest that their benefit decreases as models get larger, and we think it is unlikely that an add-on like a FWL will substantially improve models of the scale of GPT-3 (Brown et al., 2020). Furthermore, we have shown that using FWLs at training time makes them more effective, but this has a disadvantage as well. FWLs can't be directly applied to already-trained transformer language models the way dynamic evaluation can: some fine-tuning with the fast weight layer added is required. Lastly, while we have shown FWLs improve LM perplexity, we have not evaluated FWLs at other text generation tasks, which we leave for future work.

Acknowledgements

We thank Urvashi Khandelwal and the anonymous reviewers for their thoughtful comments and suggestions.

References

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. 2016a. Using fast weights to attend to the recent past. *NeurIPS*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016b. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *NeurIPS*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *ACL*.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*.

Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2017. Efficient softmax approximation for GPUs. In *ICLR*.

Geoffrey E Hinton and David C Plaut. 1987. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*, pages 177–186.

Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *ICLR*.

Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc V Le. 2022. Transformer quality in linear time. *arXiv preprint arXiv:2202.10447*.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. 2021. Going beyond linear transformers with recurrent fast weight programmers. In *NeurIPS*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. In *ICLR*.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. 2018. Dynamic evaluation of neural sequence models. In *ICML*.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. 2019. Dynamic evaluation of transformer language models. *arXiv preprint arXiv:1904.08378*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *ICLR*.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. 2022. Fast model editing at scale. In *ICLR*.

Tsendsuren Munkhdalai, Alessandro Sordoni, Tong Wang, and Adam Trischler. 2019. Metalearned neural memory. *NeurIPS*.

Tsendsuren Munkhdalai and Hong Yu. 2017. Meta networks. In *ICML*.

Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. 2020. Rapid learning or feature reuse? towards understanding the effectiveness of MAML. In *ICLR*.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. 2021. Efficient content-based sparse attention with routing transformers. *TACL*, 9:53–68.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. 2021. Linear transformers are secretly fast weight programmers. In *ICML*.

Jürgen Schmidhuber. 1992. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139.

Davis Yoshida and Kevin Gimpel. 2021. Reconsidering the past: Optimizing hidden states in language models. In *Findings of the ACL: EMNLP 2021*.