Plastic Neural Networks: Learning Through Iterative Delta Refinement

Seungho Choi
Independent Researcher
madst0614@gmail.com

Abstract

We explore an alternative approach to transformer architectures through **Plastic Neural Networks (PNN)**, which learn via iterative refinement rather than deep layer stacks. PNN applies a single delta refinement module recurrently, computing additive updates to representations with query-key adaptive gating controlling dimension-wise modifications.

In experiments on WikiText-103 masked language modeling, a 53.7M parameter PNN achieves 47.4% accuracy, reaching 88% of BERT-base performance with 41% of the parameters. We observe several interesting emergent properties: tokens appear to converge at different rates (1.53 steps for easier tokens, 2.60 for harder ones), dimension activation patterns vary across refinement steps, and the refinement process shows similarities to learned optimization.

These results suggest that recurrent delta refinement may offer a parameter-efficient alternative for certain language modeling tasks. We present our architecture, analyze its behavior, and discuss potential directions for future work.

1 Introduction

Transformer-based language models typically employ deep stacks of layers, with each layer maintaining and refining a complete representation of the input Vaswani et al. [2017], Devlin et al. [2019]. While this approach has proven highly effective, it naturally raises the question: *could we achieve useful performance through iterative refinement of a single layer, trading some accuracy for significant parameter reduction?*

We investigate this question through Plastic Neural Networks (PNN), a simple architecture that applies a single refinement module recurrently. Rather than stacking distinct layers, PNN computes additive updates (deltas) to an evolving representation, with adaptive gating determining which dimensions to modify at each step. The name reflects the architecture's single adaptive module, analogous to synaptic plasticity in biological systems Hebb [1949].

Our primary motivation is parameter efficiency. By sharing parameters across refinement steps, we aim to reduce model size while maintaining reasonable performance. A secondary interest is understanding what properties emerge from this iterative refinement process—particularly whether the model develops any form of adaptive computation or curriculum-like behavior.

1.1 Summary of Findings

We evaluate PNN on WikiText-103 masked language modeling and observe:

- A 53.7M parameter PNN achieves 47.4% accuracy, reaching 88% of BERT-base performance (57.0%) with 41% of the parameters (132M).
- Tokens converge at different rates: simpler tokens reach stable predictions in fewer refinement steps (1.53 on average) compared to more difficult tokens (2.60 steps), suggesting some form of adaptive computation.
- Dimension activation patterns change across steps, with 77% of dimensions active in early refinement decreasing to 53% in later steps, indicating increasingly selective updates.

- The refinement process shares characteristics with learned optimization approaches Andrychowicz et al. [2016], where each step functions as an adaptive update to the current state.

These results suggest that iterative delta refinement may be a viable approach for applications where parameter efficiency is important and some accuracy trade-off is acceptable. We present our architecture, share our experimental observations, and discuss what these findings might mean for future work.

1.2 Scope and Limitations

We focus specifically on masked language modeling with a single dataset (WikiText-103) and model size ($\sim$50M parameters). Our goal is not to claim superiority over existing approaches, but rather to explore whether this alternative architecture exhibits interesting properties worth further investigation. We discuss limitations and future directions in Section 6.

2 Related Work

Our work relates to several lines of research in efficient and adaptive neural architectures.

Recurrent Transformers. Universal Transformers Dehghani et al. [2018] apply transformer blocks recurrently with adaptive computation time, demonstrating improved systematic generalization. Our approach shares the recurrent structure but differs in using additive delta updates rather than complete layer re-application. Hao et al. [2019] explore recurrence in BERT for long sequences, while we focus on parameter efficiency through refinement.

Parameter Sharing. ALBERT Lan et al. [2019] demonstrates effective parameter sharing across transformer layers, achieving strong results with fewer parameters. Our work extends this idea to pure recurrent application, where a single module is reused across all refinement steps rather than just sharing between distinct layers.

Adaptive Computation. Adaptive Computation Time Graves [2016] enables variable computation per input. PonderNet Banino et al. [2021] learns to allocate computation through learned halting. While we do not implement explicit halting, we observe that tokens naturally converge at different rates, suggesting emergent adaptive behavior worth exploring further.

Learned Optimization. Learning to learn by gradient descent Andrychowicz et al. [2016] and related meta-learning approaches Finn et al. [2017] train neural networks to optimize other networks. We observe similarities between our delta refinement and learned optimization, though we do not explicitly train for this—it appears to emerge from the architecture.

Iterative Refinement. Iterative refinement has been explored in various contexts, including iterative self-attention Correia et al. [2019] and refinement for generation tasks Lee et al. [2018]. Our contribution is examining pure delta-based refinement for representation learning in language modeling.

Our work can be viewed as combining elements from these areas—recurrent application, parameter sharing, and iterative refinement—into a simple architecture for exploring parameter-efficient language modeling.

3 Plastic Neural Networks

We describe the PNN architecture, which learns through iterative application of a delta refinement module to an evolving representation.

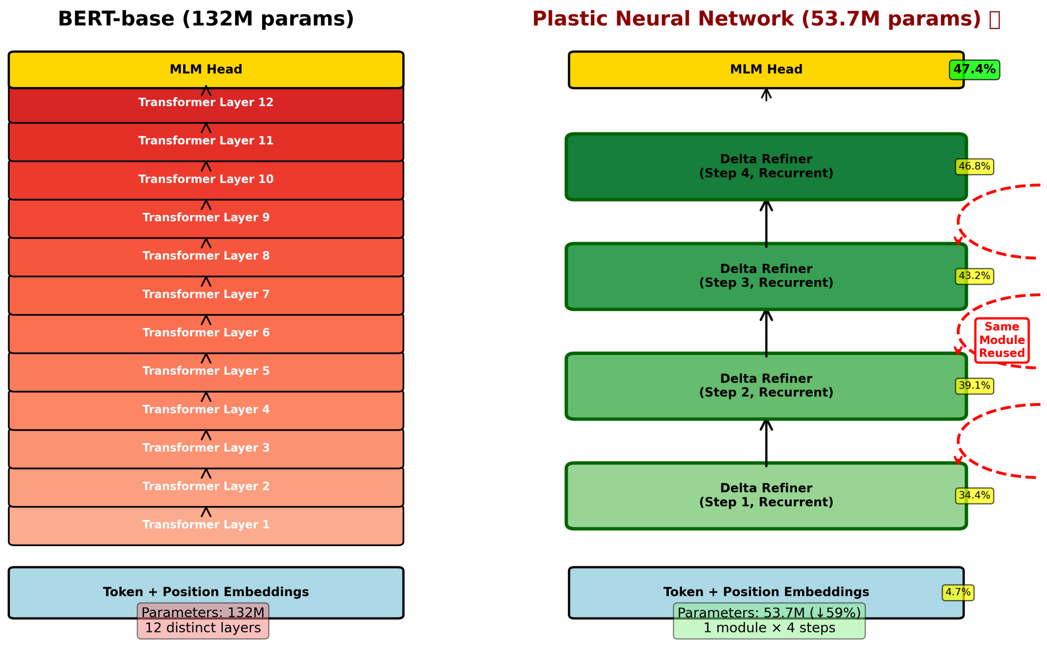

Figure 1: Architecture comparison between BERT-base (left) and PNN (right). BERT uses 12 distinct layers (132M parameters), while PNN recurrently applies a single delta refiner 4 times (53.7M parameters).

3.1 Architecture Overview

Given an input sequence of tokens $\mathbf{x} = (x_1, \ldots, x_n)$, PNN constructs representations through the following process:

1. **Embedding**: Map tokens to initial representations $\mathbf{h}^{(0)} \in \mathbb{R}^{n\times d}$ through learned token and position embeddings.
2. **Iterative Refinement**: For steps $t = 1, \ldots, T$, compute:
$$\mathbf{h}^{(t)} = \mathbf{h}^{(t-1)} + \Delta^{(t)} \tag{1}$$
where $\Delta^{(t)} = \text{DeltaRefiner}(\mathbf{h}^{(t-1)})$ is an additive update computed by a single shared module.
3. **Prediction**: Apply a task-specific head to $\mathbf{h}^{(T)}$.

The key difference from standard transformers is that we use a single DeltaRefiner module applied T times, rather than T distinct layers. This parameter sharing significantly reduces model size while allowing multiple refinement iterations. Figure 1 illustrates the architecture comparison.

3.2 Delta Refinement Module

The DeltaRefiner module computes additive updates through self-attention and feed-forward layers:

$$\text{Attn}^{(t)} = \text{MultiHeadAttention}(\mathbf{h}^{(t-1)}) \tag{2}$$
$$\mathbf{h}_{\text{attn}} = \text{LayerNorm}(\mathbf{h}^{(t-1)} + \text{Attn}^{(t)}) \tag{3}$$
$$\delta_{\text{raw}} = \text{FFN}(\mathbf{h}_{\text{attn}}) \tag{4}$$
$$\Delta^{(t)} = \text{Gate}(\mathbf{h}^{(t-1)}, \delta_{\text{raw}}) \odot \delta_{\text{raw}} \tag{5}$$

We initialize the final FFN layer with zero weights, ensuring that $\Delta^{(1)} \approx \mathbf{0}$ initially, which stabilizes early training. This allows the model to gradually learn useful refinements rather than disrupting the embedding representations immediately.

3.3 Query-Key Adaptive Gating

To enable selective refinement, we introduce adaptive gating that determines which dimensions to update at each step. The gate computes element-wise compatibility between the current representation and proposed update:

$$\mathbf{q} = \mathbf{W}_q \mathbf{h}^{(t-1)} \quad \text{(what is needed)} \tag{6}$$

$$\mathbf{k} = \mathbf{W}_k \delta_{\text{raw}} \quad \text{(what is offered)} \tag{7}$$

$$g_{ij} = \sigma\left(\frac{q_{ij} \cdot k_{ij}}{\tau}\right) \quad \text{(compatibility)} \tag{8}$$

where σ is the sigmoid function, τ is a learned temperature parameter, and i, j index tokens and dimensions respectively. The final update is:

$$\Delta^{(t)} = \mathbf{g} \odot \delta_{\text{raw}} \tag{9}$$

This gating mechanism allows the model to selectively modify dimensions based on local compatibility, rather than applying uniform updates across all dimensions.

3.4 Training Procedure

We train PNN on masked language modeling using a weighted combination of losses from different refinement steps:

$$\mathcal{L} = \sum_{t=1}^{T} w_t \cdot \mathcal{L}_{\text{MLM}}(\mathbf{h}^{(t)}) \tag{10}$$

where w_t are step weights (we use $w = [0.1, 0.2, 0.3, 0.4]$ for $T = 4$). This encourages useful refinement at each step while emphasizing later steps. We use standard cross-entropy loss for MLM with 15% random masking.

Implementation Details. We use $d = 768$ hidden dimensions, 12 attention heads, and 2048 FFN intermediate dimensions. For comparison, we implement a BERT-base baseline with identical hyperparameters but 12 distinct layers. Both models are trained on WikiText-103 Merity et al. [2016] with the same data processing, learning rate schedule, and optimization settings (details in Appendix A).

4 Experiments

We evaluate PNN on masked language modeling and compare it with a BERT-base baseline to understand the trade-offs of recurrent delta refinement.

4.1 Experimental Setup

Dataset. We use WikiText-103 Merity et al. [2016], a collection of Wikipedia articles with 103M training tokens. We sample 1M sequences for training and use the standard validation split (2,188 sequences). Sequences are tokenized with the BERT tokenizer and truncated to 128 tokens.

Models. We compare two models:

- **BERT-base**: 12 layers, 768 hidden, 12 heads, 3072 FFN intermediate, 132M parameters
- **PNN**: 4 refinement steps, 768 hidden, 12 heads, 2048 FFN intermediate, 53.7M parameters

Both use identical embedding layers, optimization (AdamW with learning rate 3e-4, cosine decay), and training duration (15 epochs, batch size 1152 effective).

Table 1: Main results on WikiText-103 masked language modeling. PNN achieves reasonable performance with significantly fewer parameters.

Model	**Parameters** (M)	**Accuracy** (%)	**Relative** Performance	**Efficiency** (%/M)
BERT-base	132.1	57.0	100%	0.43
PNN (ours)	53.7	47.4	88%	**0.88**
Reduction	-59%	-9.6pp	-12%	+2.05×

Hardware. All experiments run on a single NVIDIA A100 80GB GPU with mixed precision (FP16) training and TF32 enabled.

4.2 Main Results

Table 1 shows the main comparison. PNN achieves 47.4% accuracy with 53.7M parameters, reaching 88% of BERT-base performance with 41% of the parameters. This yields 2.05× better parameter efficiency (accuracy per million parameters).

4.3 Refinement Progression

Figure 2 shows how accuracy improves through refinement steps. Starting from raw embeddings (4.7% accuracy), each refinement step adds approximately 10% accuracy, with the largest jump from embeddings to first refinement (34.4%). This progressive improvement suggests that the model learns meaningful refinements at each step.

4.4 Emergent Curriculum Learning

An unexpected finding is that tokens converge at different rates based on difficulty (Figure 2, top-right). We classify tokens based on when they first reach correct predictions:

- **Easy** (4.9%): Correct after embedding only (0 refinement steps needed)
- **Medium** (44.4%): Become correct during refinement (1-3 steps)
- **Hard** (50.7%): Never correct or correct only at final step (4 steps)

Average steps to convergence are 1.53, 2.29, and 2.60 respectively. This suggests the model naturally learns a form of curriculum learning, allocating fewer refinement steps to easier tokens.

4.5 Adaptive Gating Analysis

Figure 3 shows gate activation patterns across refinement steps. Mean gate values start at 0.61 and decrease slightly to 0.60 by step 4, with high variance indicating selective activation. The proportion of strongly active dimensions (gate > 0.5) decreases from 77% at step 1 to 53% at step 3, then increases to 67% at step 4, suggesting the model learns when to be selective versus comprehensive in its updates.

The gate activation heatmap (Figure 3, bottom-left) reveals that different dimensions have varying activation patterns across steps, with some dimensions consistently active and others selectively engaged. The distribution of token-level average gates (bottom-right) shows distinct patterns at each step, with step 4 exhibiting a bimodal distribution suggesting two modes of refinement.

5 Analysis

We analyze several aspects of PNN's behavior to understand how iterative delta refinement achieves its results.

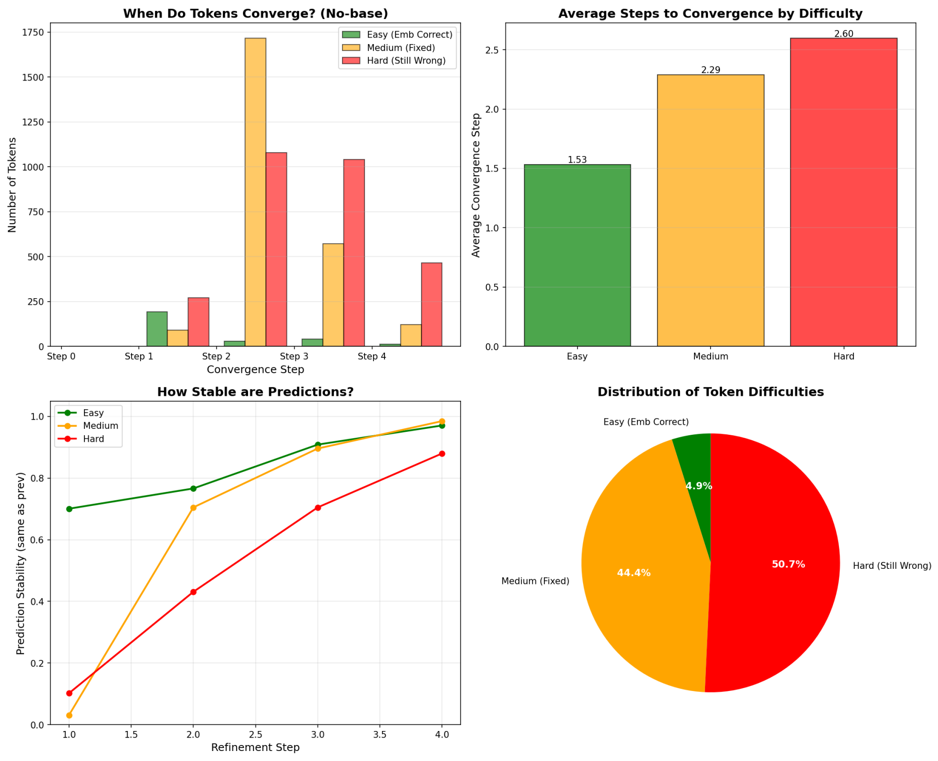

Figure 2: Convergence analysis showing accuracy improvement across refinement steps (left) and convergence patterns by token difficulty (top-left). Easy tokens converge faster than hard tokens, suggesting emergent adaptive computation.

5.1 Comparison with Learned Optimization

The refinement process shares characteristics with learned optimizers Andrychowicz et al. [2016]. Each refinement step can be viewed as:

$$\mathbf{h}^{(t)} = \mathbf{h}^{(t-1)} + \eta \cdot \mathbf{g}^{(t)} \odot \mathbf{d}^{(t)} \tag{11}$$

where $\mathbf{d}^{(t)}$ is a "direction" (the raw delta), $\mathbf{g}^{(t)}$ is an adaptive "step size" (the gate), and η is implicitly learned through the FFN weights. This resembles gradient descent with learned step sizes and directions.

However, unlike meta-learning approaches that explicitly optimize for optimization, PNN's optimization-like behavior emerges from training on the end task. This suggests that delta-based refinement may naturally develop optimization properties when learning iterative improvement.

5.2 Why Does This Work?

We hypothesize that PNN succeeds due to several factors:

Rich Embeddings. The initial embeddings contain substantial information (4.7% accuracy), providing a reasonable starting point. Pre-trained embeddings might further improve performance.

Parameter Efficiency. By reusing parameters across steps, PNN achieves higher effective capacity per parameter. Each parameter contributes to multiple refinement iterations rather than a single layer.

Adaptive Computation. The emergent curriculum learning suggests that PNN implicitly learns when tokens need more refinement, allocating computation adaptively.

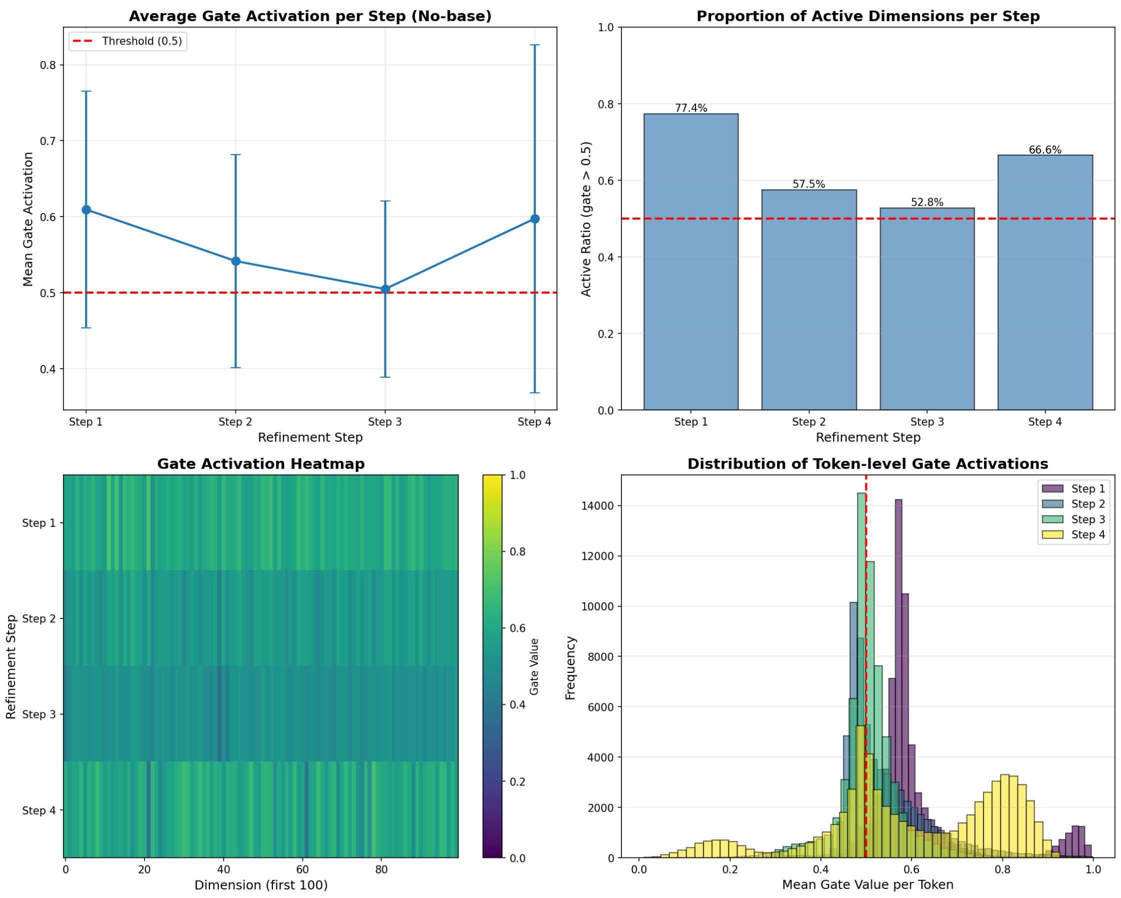

Figure 3: Gate activation analysis. Top-left: mean activation per step with variance. Top-right: proportion of active dimensions. Bottom-left: heatmap of gate values across dimensions. Bottom-right: distribution of token-level average gates, showing bimodal patterns.

Incremental Learning. The delta formulation with zero initialization encourages small, stable updates rather than large representational changes. This may ease optimization.

5.3 Comparison with Universal Transformers

Universal Transformers (UT) Dehghani et al. [2018] also apply layers recurrently but differ in key ways:

- UT re-applies complete transformer layers; PNN computes additive deltas
- UT uses explicit halting mechanisms; PNN's adaptation is implicit
- UT targets systematic generalization; PNN targets parameter efficiency

The delta formulation may offer advantages for stability (small updates) and interpretability (what changes at each step), though this requires further investigation.

6 Discussion and Future Work

6.1 Limitations

Our work has several limitations worth noting:

Performance Gap. PNN achieves 88% of BERT's performance, representing a 9.6 percentage point accuracy gap. For applications requiring maximum accuracy, this trade-off may not be acceptable.

Single Task and Dataset. We evaluate only masked language modeling on WikiText-103. Performance on other tasks (classification, generation) and datasets remains unknown.

Model Size. We test only ∼50M parameters. Behavior at larger scales (100M+) is unexplored, and the parameter efficiency advantage may diminish with scale.

No Pre-training. All models train from scratch. Pre-trained embeddings or transfer learning might change the relative performance.

6.2 Future Directions

Several directions could extend this work:

Downstream Tasks. Evaluating PNN on GLUE Wang et al. [2018], SuperGLUE Wang et al. [2019], and other benchmarks would clarify whether the approach generalizes beyond MLM.

Explicit Halting. Adding learned halting mechanisms Graves [2016] could enable true adaptive computation, allowing tokens to stop refining when sufficiently confident.

Larger Models. Scaling to 100M-1B parameters would reveal whether parameter efficiency advantages persist at scale.

Hybrid Approaches. Combining a small base transformer (2-3 layers) with recurrent refinement might offer better accuracy-efficiency trade-offs.

Theoretical Analysis. Formal analysis of why delta refinement exhibits optimization-like behavior could guide architecture improvements.

Other Domains. Applying PNN to vision (ViT) Dosovitskiy et al. [2020] or multimodal tasks could reveal domain-specific properties.

6.3 Broader Context

Our work contributes to ongoing discussions about efficient architecture design. While transformers have achieved remarkable success through scaling, there may be alternative approaches—like recurrent refinement—that offer better parameter efficiency for certain applications. We do not claim PNN replaces transformers, but rather suggest it represents one point in a broader design space worth exploring.

The emergent properties we observe (curriculum learning, adaptive gating) also raise interesting questions about what behaviors arise from simple architectural constraints. Understanding these emergent properties may inform future architecture design.

7 Conclusion

We introduced Plastic Neural Networks, which learn through iterative delta refinement rather than deep layer stacks. On WikiText-103 masked language modeling, a 53.7M parameter PNN achieves 47.4% accuracy—88% of BERT-base performance with 41% of the parameters. Analysis reveals emergent curriculum learning, adaptive dimension activation, and similarities to learned optimization.

These results suggest that recurrent delta refinement may offer a parameter-efficient alternative for applications where some accuracy trade-off is acceptable. We hope our exploration of this simple architecture encourages further investigation into iterative refinement approaches and the emergent properties they exhibit.

Acknowledgments

This work was conducted as independent research. Experiments were conducted on a single NVIDIA A100 GPU. The author thanks the open-source community for providing essential tools and libraries that made this research possible.

References

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

Gonçalo M Correia, Vlad Niculae, and André FT Martins. Adaptively sparse transformers. *arXiv preprint arXiv:1909.00015*, 2019.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

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*, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Jie Hao, Xing Wang, Baosong Yang, Longyue Wang, Jinfeng Zhang, and Zhaopeng Tu. Modeling recurrence for transformer. *arXiv preprint arXiv:1904.03092*, 2019.

Donald Olding Hebb. *The organization of behavior: A neuropsychological theory.* John Wiley & Sons, 1949.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic non-autoregressive neural sequence modeling by iterative refinement. *arXiv preprint arXiv:1802.06901*, 2018.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

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.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

A Implementation Details

A.1 Hyperparameters

Table 2 lists all hyperparameters used in our experiments.

Table 2: Complete hyperparameter settings for all experiments.

Parameter	**Value**
Model Architecture	
Hidden dimension	768
Attention heads	12
FFN intermediate (PNN)	2048
FFN intermediate (BERT)	3072
Refinement steps (PNN)	4
Layers (BERT)	12
Max sequence length	128
Dropout	0.1
Training	
Optimizer	AdamW
Learning rate	3e-4
Warmup steps	500
LR schedule	Cosine decay (min 1e-5)
Batch size	384
Gradient accumulation	3
Effective batch size	1,152
Training samples	1,000,000
Epochs	15
Mixed precision	FP16
Gradient clipping	1.0
Weight decay	0.01
Step weights (PNN)	[0.1, 0.2, 0.3, 0.4]
Data	
Mask probability	0.15
Dataset	WikiText-103
Tokenizer	BERT uncased
Vocabulary size	30,522

A.2 Training Time and Resources

All experiments were conducted on a single NVIDIA A100 80GB GPU:

- **PNN**: 269.5 minutes (15 epochs), 17.9 min/epoch average

- **BERT-base**: 248.2 minutes (15 epochs), 16.5 min/epoch average

PNN is slightly slower per epoch despite fewer parameters due to recurrent computation and gate calculations. However, both models train in under 5 hours, making experiments computationally accessible.

A.3 Code and Reproducibility

We will release code, trained checkpoints, and detailed training logs upon publication to facilitate reproduction and extension of our work. All experiments use PyTorch 2.0+ with standard libraries (transformers, datasets).