Transformers from an Optimization Perspective

Anonymous Author(s)
Affiliation
Address
email

Abstract

Deep learning models such as the Transformer are often constructed by heuristics and experience. To provide a complementary foundation, in this work we study the following problem: Is it possible to find an energy function underlying the Transformer model, such that descent steps along this energy correspond with the Transformer forward pass? By finding such a function, we can reinterpret Transformers as the unfolding of an interpretable optimization process across iterations. This unfolding perspective has been frequently adopted in the past to elucidate more straightforward deep models such as MLPs and CNNs; however, it has thus far remained elusive obtaining a similar equivalence for more complex models with self-attention mechanisms like the Transformer. To this end, we first outline several major obstacles before providing companion techniques to at least partially address them, demonstrating for the first time a close association between energy function minimization and deep layers with self-attention. This interpretation contributes to our intuition and understanding of Transformers, while potentially laying the ground-work for new model designs.

1 Introduction

Although deep learning has achieved extensive practical success [21, 22], model architectures are often motivated by heuristics and at times remain difficult to understand. As such, significant effort has recently be devoted to various ways of analyzing and reinterpreting popular deep network structures. Of particular note for our purposes here are assorted attempts to create a one-to-one association between deep model layers and the unfolded iterations of some optimization process designed to minimize an interpretable energy function [10, 11, 43, 45, 33]. In this way the forward pass can be viewed as computing, at least approximately, a representation with minimal energy, where model parameters dictate the form of this energy and can be trained via the backward pass for a downstream goal of interest, e.g., classification or regression, etc.

This so-called *unfolded optimization* perspective can provide insights into the nature of the inductive biases introduced by different architectures, while potentially serving as a guide for bespoke design choices instantiated through properties of the underlying energy function involved. In fact, modifications of the latter can lead to predictable model behaviors, which have in the past been tailored to incorporate useful regularization factors [49], avoid subtle degenerate solutions [45], or design effective new models altogether [1, 10].

Despite these successes, the majority of prior work in this area has either addressed relatively simple network structures such as MLPs [43], or else restricted consideration to a single model component in isolation from a larger system [33]. Consequently, complex models such as the widely-used Transformer [39] have been mostly ignored, in large part because of the difficulty involved in simultaneously mapping both the self-attention and feed-forward network (FFN) modules to an integrated optimization process that minimizes a single, unified energy function.

We take significant steps towards extending the unfolded optimization perspective to a full Transformer stack via the following contributions:

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

- After providing background details and existing examples of unfolded optimization, Section 2 formalizes the four key challenges to mapping Transformer layers to optimization steps descending a well-specified energy function. These include: (i) handling self-attention, (ii) integrating heterogeneous Transformer layer modules (i.e., self-attention and FFN), (iii) accounting for non-linear activations, and (iv) allowing for arbitrary/unstructured weight matrices.
- Later, Section 3 derives an energy function that closely reproduces self-attention per Challenge (i), Section 4 rigorously derives convergence results which demonstrate how to approximately handle Challenge (ii) via a novel form of alternating minimization, and Section 5 provides explicit details for solving Challenge (iii) via proximal methods; consideration of Challenge (iv) is deferred to the supplementary along with complementary supporting analyses.
- The above analysis culminates in an energy function that is minimized by Transformer layers when certain technical conditions hold. To provide further support, Section 6 empirically demonstrates that indeed this energy is minimized with real-world data even when technical conditions are difficult to formally verify.

Collectively, these results provide a complementary foundation for understanding Transformer models, and suggest plausible avenues for future enhancements.

2 Brief Intro to Deep Architectures Formed from Unfolded Optimization

The basic idea of unfolded optimization perspective is to create a one-to-one correspondence between the layers of a deep model and the iterations of some algorithmic process designed to minimize a parameterized, model-specific energy function. In this way, the forward pass can be viewed as computing an approximation to the minimum energy solution. Meanwhile, assuming each forward iteration is differentiable w.r.t. the energy function parameters, we can pass gradients of some meta objective function (e.g., classification, regression, etc.) through the approximate minimization steps for the backward pass, which forms an interpretable bilevel-optimization process [7, 10, 20]. Overall then, unfolded optimization produces minimal energy representations, whereby the specific structure of the energy function is trainable per the relevant downstream application domain.

2.1 Mathematical Formulation

The bilevel optimization process described above can be expressed more formally as:

$$Y^*(W,X) = \arg\min_{Y} E(Y;W,X), \tag{1}$$

$$W^*(X) = \arg\min_{W} \ell(\psi[Y^*(W,X)]). \tag{2}$$

Here $X \in \mathbb{R}^{n\times d}$ is the model input (e.g., input sentence to a language model), where n is the number of tokens and d is the token dimensionality. We refer to $Y \in \mathbb{R}^{n\times d}$ as the hidden representation of the model, where Y^* is the (ideal) model output (e.g. word representation output by a Transformer encoder) that minimizes to lower-level energy function[1] $E : \mathbb{R}^{n\times d} \to \mathbb{R}$ whose form is dictated by parameters W (e.g., these could also correspond with the trainable parameters in a Transformer model). And finally, ℓ is a meta-loss function (e.g., the loss function used to train a Transformer) while $\psi : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ is an arbitrary transformation that converts hidden representations to the model output, which can also potentially have its own learnable parameters. Assuming $\partial Y^*(W,X)/\partial W$ and $\partial \ell/\partial Y^*(W,X)$ are computable, where the former involves passing gradients through the iteratives steps used to approximate $Y^*(W,X)$, then the combined bilevel system can be optimized w.r.t. W.

Note that hereinafter if the energy function does not include inter-token interactions, we view it as a function of vectors, and we use lower-case characters $\mathbf{y}$ and $\boldsymbol{x}$ to represent an arbitrary row of Y and X for simplicity. Also, when clear from context, we omit writing the parameters of a function, for example we may write $E(Y)$ to reference $E(Y;W,X)$.

2.2 Related Work and Limitations

In this section we present related work on unfolded optimization, and in particular, analyze some of the relevant limitations that serve as motivation for our efforts and underscore the challenges involved.

[1] Note that although we used the term "energy" and "unfolding", the concept is different from what has been called "energy based learning" in the past, where the energy is defined on the joint input and output spaces [2, 23]; here energy is just a function for explaining the forward pass.

Optimization Induced Feed-Forward Networks A wide variety of prior work has investigated feed-forward structures from an unfolded optimization perspective [10, 12, 16, 18, 37, 43]. Here we take a closer look at [43] which is based on proximal methods. The basic energy function in [43] is

$$E(\mathbf{y}) = \mathbf{1}^\top \tilde{\sigma}^*(W^{-\top}\mathbf{y}) - \left\langle f(\boldsymbol{x}), W^{-\top}\mathbf{y} \right\rangle - \frac{1}{2}\|\mathbf{y}\|^2, \tag{3}$$

where $\mathbf{1}$ is a vector with all entries being 1, $\tilde{\sigma}^*$ is the convex conjugate [34][2] of $\tilde{\sigma} : a \mapsto \int_0^a \sigma(r)\mathrm{d}r$, σ is an activation function and $f(\boldsymbol{x})$ is some transformation of input prepresentation $\boldsymbol{x}$. The proximal operator [8] of (3) is

$$\mathbf{y}^{(t+1)} = \mathrm{prox}_E\left(\mathbf{y}^{(t)}\right) = W^\top \sigma\left(W\mathbf{y}^{(t)} + f(\boldsymbol{x})\right). \tag{4}$$

Note that although in (4) W can in principle be arbitrary matrix as long as the dimensionality is properly aligned, after stacking multiple layers of this model, the effective transformation is actually forced to be constrained:

$$\mathbf{y}^{(t+2)} = W^\top \sigma\left[WW^\top \sigma\left(W\mathbf{y}^{(t)} + f(\boldsymbol{x})\right) + f(\boldsymbol{x})\right], \tag{5}$$

where clearly after the first layer the actual feed-forward transformation becomes $WW^\top$, which is necessarily positive semi-definite (PSD). Hence unconstrained layer weights are not actually possible within this paradigm.

Optimization Induced Attention Mechanisms There is also limited work that attempts to interpret attention mechanisms from the unfolded optimization perspective [11, 33]. For example, the energy function

$$E(\mathbf{y}) = -\beta^{-1}\log\left(\sum_{i=1}^{d} e^{\beta S_i \mathbf{y}}\right) + \frac{1}{2}\|\mathbf{y}\|^2, \tag{6}$$

was proposed in [33], with subsequent updating using the concave convex procedure [47] leading to iterations of the form

$$\mathbf{y}^{(t+1)} = S\,\mathrm{softmax}(\beta S \mathbf{y}^{(t)}), \tag{7}$$

where $S \in \mathbb{R}^{n\times n}$ corresponds with the attention keys while $\mathbf{y}$ maps to the attention query, and β is a constant scalar. Critically though, this attention mechanism is actually *cross-attention* (using $\mathbf{y}$ to attend to S), which does not well-align with the typical self-attention use-case of Transformers. Moreover, this work does not account for the aggregated Transformer feed-forward network module and attendant nonlinearity.

Optimization Induced Graph Neural Networks A variety of graph neural network architectures have also been developed from a similar optimization perspective[5, 26, 28, 31, 44, 45, 48, 50]. For example, graph attention mechanisms were derived using the iterative reweighted least squares (IRLS) algorithm in [45]; this result is related to self-attention, which can be viewed as graph attention on fully connected graphs [4]; however, it fails to produce the Transformer softmax term or the combined self-attention/feedforward Transformer stack. Nonetheless, we will later demonstrate how ideas from [45] can be leveraged to derive self-attention with softmax.

2.3 Key Challenges Extending to General Transformers

In this paper, we focus on a more complex model than prior work: the Transformer encoder. A Transformer layer [39] is typically composed of two primary components: the self-attention layer and the feed-forward network (FFN) layer. If we simplify the FFN to a single linear transformation with non-linear activation, and ignore the layer-norm and residual connections,[3] one Transformer layer reduces to the basic form

$$Y^{(t+1)} = \mathrm{ReLU}\left[\mathrm{softmax}\left(Y^{(t)}W_a Y^{(t)\top}\right)Y^{(t)}W_f\right]. \tag{8}$$

With respect to (8) and the limitations of prior work discussed above, we propose four major challenges in formally applying unfolded optimization to the Transformer setting:

[2] Here σ^* is a scalar function, but for simplicity, we sometimes apply a scalar function to a vector or matrix in an entry-wise fashion for convenience.

[3] Note that we are ignoring layer-norms and residual connections for simplicity of exposition; however, the analysis we present herein can naturally be generalized to accommodate them; see supplementary for details.

1. *Token-Level Interactions*: As mentioned previously, Transformer models include not only interactions between neurons, but also cross-token interactions, which is reflected by the self-attention mechanism considered to be the *sine qua non* of Transformers. And thus far, no prior work has derived Transformer-style self-attention, instead either relaxing to cross-attention [33], or failing to produce the ubiquitous softmax operator [45].
2. *Heterogeneous Layer Types*: Each Transformer encoder layer is composed of two totally different components, a self-attention layer and a feedforward network layer (FFN). We refer to this structure as "heterogeneous layer types," where each component has its own parameters and can be viewed as distinct unfolded optimization processes. However, it remains unknown whether or not it is possible to combine the corresponding energies of these two components to obtain a unified objective with any kind of convergence guarantees (even approximately) during the aggregated forward pass.
3. *Non-Linear Activations*: Many activation functions used in common neural network architectures can be understood from the perspective of proximal operators [24, 45], at least within the isolated context of some relatively simple feedfoward schema. However, when integrated within the heterogeneous Transformer layer types mentioned above, the optimization process becomes rather complex, and it is unknown if any convergence properties still hold after the inclusion of a proximal step.
4. *Asymmetry of Weights*: As alluded to in the discussion of (5), the scope of most work addressing feed-forward networks is actually limited to models with symmetric (or even tighter, PSD) weight transformations [1, 10, 43], which restricts the resulting universality. Additionally, related models from the graph neural network literature are generally predicated on undirected graphs [44, 45], where the graph propagation is also symmetric. Consequently, although to our knowledge not addressed/discussed previously, it remains unknown how to construct energy functions for more general asymmetric transformations.

In this paper, we propose techniques to at least partially solve Challenges 1, 2 and 3, while for 4, we defer preliminary discussion to the supplementary, leaving formal investigation as a future direction.

3 A New Derivation of Transformer Self-Attention

We now tackle Challenge 1 by constructing an energy function whose iterative optimization steps match Transformer-style softmax self-attention on fully connected tokens as is customary.

3.1 Basic Softmax Self-Attention via Unfolded Optimization Steps

Consider the energy function

$$E_1(Y) = \sum_{i=1}^{n}\sum_{j=1}^{n} \rho\left(\frac{1}{2}\|\mathbf{y}_i - \mathbf{y}_j\|^2\right) + R(Y), \tag{9}$$

where $\mathbf{y}_i \in \mathbb{R}^{d\times 1}$ is the i-th row of matrix $Y \in \mathbb{R}^{n\times d}$, $\rho : \mathbb{R}^+ \to \mathbb{R}$ is a concave non-decreasing function, and $R : \mathbb{R}^{n\times d} \to \mathbb{R}$ is a convex function. Interestingly, although E_1 is not necessarily convex because of the non-convexity of ρ, under specific choices of ρ and R, it can be optimized through a softmax-like structure as follows:

Theorem 3.1. *Assume $\rho(z) = -\exp\{-z\}$, $R(Y) = \frac{1}{2}\|Y\|_{\mathcal{F}}^2$, and $\boldsymbol{\beta}_i = \exp\left\{-\frac{1}{2}\left\|\mathbf{y}_i^{(t)}\right\|^2\right\}$, and let $Y^{(t)}$ represent any fixed value for Y. Then the update rule*

$$\mathbf{y}_i^{(t+1)} = \frac{\sum_{j=1}^{n}\boldsymbol{\beta}_j \exp\left\{\mathbf{y}_i^{(t)\top}\mathbf{y}_j^{(t)}\right\}\mathbf{y}_j^{(t)}}{\sum_{j=1}^{n}\boldsymbol{\beta}_j \exp\left\{\mathbf{y}_i^{(t)\top}\mathbf{y}_j^{(j)}\right\}}, \quad \forall i, \tag{10}$$

satisfies $E_1\left(Y^{(t+1)}\right) \le E_1\left(Y^{(t)}\right)$ with equality iff $Y^{(t)}$ is a stationary point of E_1.

It is worth mentioning that, although perhaps not obvious, the update step (10) can be generated via a majorization-minimization (MM) algorithm [38], where the majorization step generates a convex upper-bound, and the minimization step descends along the gradient of the upper-bound. Therefore, the core of this update is essentially a gradient step on a convex function, which will be relevant to the discussion in subsequent sections; see the proof and further details in the supplementary.

Remark 3.2. Several notable generalizations of Theorem 3.1 are possible. First, although in Theorem 3.1 we adopt a specific form for ρ and R to recover the softmax operator, with other selections of these functions the corresponding unfolded optimization algorithm can generate different/novel types of attention mechanisms. Secondly, obtaining (10) relies on a particular choice of gradient step size; however, for broader choices the resulting convergent updates interestingly lead to residual connections as a natural byproduct of the optimization trajectory (see supplementary). And finally, Theorem 3.1 can be easily modified to accommodate situations whereby full Transformer connectivity is constrained by some graph structure as in [6, 14] (again, see supplementary for details).

3.2 Extension to Include Trainable Parameters

After aggregating into matrix form, we have thus far shown that the iteration

$$Y^{(t+1)} = \mathrm{softmax}_{\boldsymbol{\beta}}\left(Y^{(t)}Y^{(t)\top}\right)Y^{(t)}, \tag{11}$$

will reduce (or leave unchanged) the energy from (9), where $\mathrm{softmax}_{\boldsymbol{\beta}}(\mathbf{y})_i = \frac{\boldsymbol{\beta}_i \exp\{\mathbf{y}_i\}}{\sum_j \boldsymbol{\beta}_j \exp\{\mathbf{y}_j\}}$ denotes a softmax operator with reweighting coefficient vector $\boldsymbol{\beta}$. If $\boldsymbol{\beta}_i$ is independent of i, i.e., $\left\|\mathbf{y}_i^{(t)}\right\|$ is constant, then this reweighted softmax is equivalent to the canonical softmax used in Transformers.

Now consider the reparameterization $Y = ZW_a$, where $W_a \in \mathbb{R}^{d\times d}$ is an invertible matrix. It follows that $Z^{(t+1)}W_a = \mathrm{softmax}_{\boldsymbol{\beta}}\left(Z^{(t)}W_aW_a^\top Z^{(t)\top}\right)Z^{(t)}W_a$, leading to the revised update

$$Z^{(t+1)} = \mathrm{softmax}_{\boldsymbol{\beta}}\left(Z^{(t)}W_a^sZ^{(t)\top}\right)Z^{(t)}, \tag{12}$$

where $W_a^s = W_aW_a^\top$ and we have adopted the superscript 's' to indicate that this matrix is symmetric. Collectively then, the results of this section directly address Challenge 1, closely reproducing softmax-styled self-attention both with and without trainable parameters (the primary lingering limitation of symmetric weights being related to Challenge 4).

4 Combining Transformer Components via Alternating Inexact Minimization

We next address Challenge 2 and the heterogeneous Transformer layer types. With this goal in mind, we first introduce a general optimization scenario. Specifically, we ask the following question:

Given two (convex) objectives $f(\mathbf{y})$ and $g(\mathbf{y})$, under what conditions, and/or to what extent, will alternatively taking separate gradient steps w.r.t. f and g optimize the aggregated function $f+g$?

We refer to this optimization strategy as *alternating inexact minimization* (AIM), and we will shortly demonstrate conditions under which it converges to a ball with finite radius containing the optimal point of $f+g$. Later we discuss how these results contribute towards the resolution of Challenge 2.

We emphasize upfront that AIM as so-defined is quite different from what is commonly referred to as alternating minimization in the literature [15, 27]. The latter refers to scenarios whereby a unified objective function with multiple variables is minimized in an alternating fashion over one variable at a time with the others fixed, such that descent can be trivially achieved. In contrast, our AIM scenario involves multiple objective terms with a *shared* variable, and we alternate minimization over each term in isolation using the same variable, a much more challenging process to analyze.

4.1 General Alternating Inexact Minimization

Given two objectives $f, g : \mathbb{R}^d \to \mathbb{R}$, AIM is formalized as Algorithm 1. We will now investigate how Algorithm 1 relates to minimization of the combined objective defined by $h(\mathbf{y}) = f(\mathbf{y}) + g(\mathbf{y})$.

Algorithm 1

For the t-th iteration, execute $\quad \boldsymbol{u}^{(t)} = \mathbf{y}^{(t)} - \alpha_1\nabla f(\mathbf{y}^{(t)}); \quad \mathbf{y}^{(t+1)} = \boldsymbol{u}^{(t)} - \alpha_2\nabla g(\boldsymbol{u}^{(t)})$.

In the following, we assume that f, g are both Lipschitz smooth and strongly convex [34], with Lipschitz constants L_f and L_g respectively, and convexity parameter c_f and c_g respectively. Therefore, h will also be Lipschitz smooth and strongly convex, with Lipschitz constant L_h and convexity c_h. We denote the optimal points of f, g, and h as $\mathbf{y}_f^*$, $\mathbf{y}_g^*$, and $\mathbf{y}_h^*$ respectively.

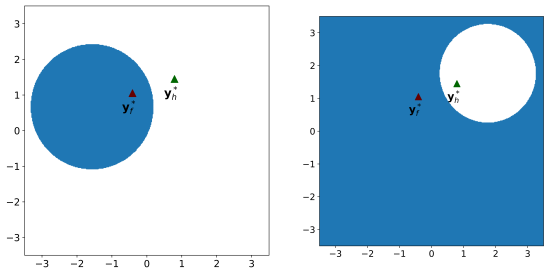

Figure 1: A two-dimensional illustration of $\mathcal{S}(\mathscr{C})$ with different values of $\mathscr{C}$. Left: $\frac{c_h \mathscr{C}}{L_f} = 0.7$. Right: $\frac{c_h \mathscr{C}}{L_f} = 1.5$. The blue area is $\mathcal{S}(\mathscr{C})$.

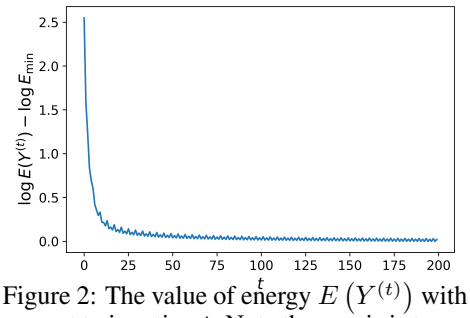

Figure 2: The value of energy $E\left(Y^{(t)}\right)$ with respect to iteration t. Note the y-axis is translated by a constant $\log E_{\min}$ to make the minimum align with zero.

Combining the two steps of Algorithm 1 gives

$$\mathbf{y}^{(t+1)} = \mathbf{y}^{(t)} - \alpha_1 \nabla f\left(\mathbf{y}^{(t)}\right) - \alpha_2 \nabla g\left[\mathbf{y}^{(t)} - \alpha_1 \nabla f\left(\mathbf{y}^{(t)}\right)\right], \tag{13}$$

while a canonical gradient descent step on objective h is $\mathbf{y}^{(t+1)} = \mathbf{y}^{(t)} - \alpha_2\left[\nabla f\left(\mathbf{y}^{(t)}\right) + \nabla g\left(\mathbf{y}^{(t)}\right)\right]$. Comparing the two update rules it is apparent that Algorithm 1 can be viewed as a noisy gradient descent step with step size α_2 and noise factor

$$\Delta_t = \nabla h\left(\mathbf{y}^{(t)}\right) - \frac{\alpha_1}{\alpha_2} \nabla f\left(\mathbf{y}^{(t)}\right) - \nabla g\left[\mathbf{y}^{(t)} - \alpha_1 \nabla f\left(\mathbf{y}^{(t)}\right)\right]. \tag{14}$$

Noisy gradient descent is a well-studied problem in certain contexts [3, 30]. However, in our specific scenario we require a novel, different (actually tighter) bound than one can get from simply applying existing results.

Specifically, we demonstrate that when $\delta\left(\mathbf{y}^{(t)}\right) = \|\Delta_t\| / \left\|\nabla h\left(\mathbf{y}^{(t)}\right)\right\|$ is bounded, (13) is guaranteed to descend the objective h *within certain region* as follows:

Theorem 4.1. *When $\alpha_1 \leq \alpha_2 \leq L_h^{-1}$, suppose $\mathbf{y}^{(t)}$ and $\mathbf{y}^{(t+1)}$ are related by (13), with $\delta\left(\mathbf{y}^{(t)}\right) \leq \mathscr{C}$ and $\mathscr{C} = \frac{\alpha_2}{\alpha_2 - \alpha_1 + \alpha_1 \alpha_2 L_g}$. Then $h\left(\mathbf{y}^{(t+1)}\right) \leq h\left(\mathbf{y}^{(t)}\right)$.*

Remark 4.2. Although the definition of $\mathscr{C}$ may appear rather complicated, when one of α_1, α_2, L_f, or L_g is sufficiently small, $\mathscr{C}$ behaves as $\Omega\left([\alpha_1 \alpha_2 L_f L_g]^{-1}\right)$.

We further consider how to interpret the constraint $\delta(\mathbf{y}) \leq \mathscr{C}$ and the region within which (13) can optimize h as follows:

Lemma 4.3. *Let $\mathcal{S}(\mathscr{C}) = \left\{\mathbf{y} \,\middle|\, \frac{\|\mathbf{y} - \mathbf{y}_f^*\|}{\|\mathbf{y} - \mathbf{y}_h^*\|} \leq \frac{c_h \mathscr{C}}{L_f}\right\}$. When $\mathbf{y}^{(t)} \in \mathcal{S}(\mathscr{C})$, $\delta(\mathbf{y}) \leq \mathscr{C}$.*

Combining Theorem 4.1 and Lemma 4.3, we can therefore conclude that when $\mathbf{y}^{(t)} \in \mathcal{S}(\mathscr{C})$, $h\left(\mathbf{y}^{(t+1)}\right) \leq h\left(\mathbf{y}^{(t)}\right)$. Note that the boundary of $\mathcal{S}(\mathscr{C})$ given in Lemma 4.3 is called an Apollonian circle. When $\mathscr{C} \leq \frac{L_f}{c_h}$, $\mathcal{S}(\mathscr{C})$ is a ball centered on $\mathbf{y}_f^*$, and when $\mathscr{C} \geq \frac{L_f}{c_h}$, $\mathcal{S}(\mathscr{C})$ is the whole space excluding a ball centered on $\mathbf{y}_h^*$. Figure 1 provides a 2D visualization for each case respectively.

Remark 4.4. Our findings above can be summarized as follows: For sufficiently small values of α_1, α_2, Algorithm 1 reduces the combined objective h, at least provided that $\mathbf{y}$ is a certain distance away from the optimal point $\mathbf{y}_h^*$.

To illustrate this conclusion, we present a synthetic example with $f(Y) = \|SY\|_{\mathcal{F}}^2 + \|Y - B_1\|_{\mathcal{F}}^2$ and $g(Y) = \|YW\|_{\mathcal{F}}^2 + \|Y - B_2\|_{\mathcal{F}}^2$. Note that here we expand the variable $\mathbf{y}$ to a matrix Y and let f and g consist of left and right transformations of Y respectively, with an extra bias term to prevent the degenerate solution ($Y^* = 0$). We randomly set each entry of $\{S, W, B_1, B_2\}$ and execute Algorithm 1 with fixed step sizes and project the trace of $Y^{(t)}$ to a two-dimensional space with PCA for visualization. The trace of $Y^{(t)}$ is displayed in Figure 3 and the combined objective h across iterations in Figure 2. From these plots, the behaviour predicted by our theory can be verified: When $Y^{(t)}$ is a sufficient distance away from Y_h^* (the energy is relatively high), the optimization trajectory moves closer to Y_h^* (descending the energy), and after $Y^{(t)}$ is close enough to Y_h^* (and the h is at a relatively low level), the energy oscillates in a certain range about the optimal solution.

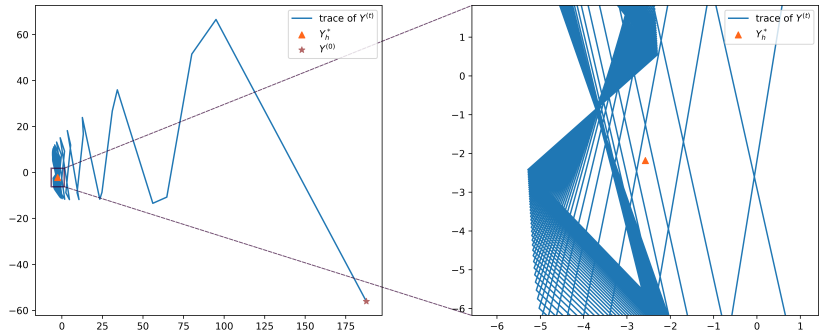

Figure 3: The trace of $Y^{(t)}$ projected to a plane. The figure on the right displays a zoomed local region around Y_h^* taken from the figure on the left.

4.2 Unfolded Optimization of Heterogeneous Layer Type Models

We now return to Transformer-specific models. Let $E_2(Y) = \frac{1}{2}\mathrm{Tr}\left(YW_fY^\top\right) + \frac{1}{2}\|Y\|_{\mathcal{F}}^2$ and consider the combined energy $E(Y) = E_1(YW_a) + E_2(Y)$ (analogous to h from the previous section) and alternatively perform the following AIM updates:

$$U^{(t)} = \mathrm{softmax}_{\boldsymbol{\beta}}\left(Y^{(t)\top}W_a^sY\right)Y^{(t)}; \tag{15}$$

$$Y^{(t+1)} = U^{(t)} - \alpha_2\left.\frac{\partial E_2}{\partial Y}\right|_{Y=U^{(t)}} = U^{(t)}W_f^s, \tag{16}$$

where $W_f^s = (1-\alpha_2)I - \alpha_2\frac{W_f+W_f^\top}{2}$ provides an additional linear transformation after the softmax.

From Section 3, we know that (15) is optimizing E_1 using essentially a row-wise gradient descent step after majorization. Likewise, (16) is also a gradient step with step size α_2. Therefore, the combined rules fall into the scope of Algorithm 1, and from the analysis in Section 4.1, we can draw the following conclusion:

Corollary 4.5. *If $Y^{(t+1)}$ is computed via (15) and (16) with input $Y^{(t)}$, then $E\left(Y^{(t+1)}\right) \leq E\left(Y^{(t)}\right)$ when $Y^{(t)} \notin \mathcal{S}$, where $\mathcal{S}$ is a ball with finite radius containing $Y_{\hat{E}}^*$, the optimal point of $\hat{E}$, the convex upper bound of E.*

Note that after combining (15) and (16), the update aggregated update rule is already very similar to our target from (8), the only differences being the missing non-linearity and symmetric weights (note we use symmetric weight matrix W_a^s and W_f^s instead of W_a and W_f here), which correspond with Challenges 3 and 4 in Section 2.3 respectively; we address the former next.

5 Non-linear Activations Integrated within the Unfolding Paradigm

Handling non-linear activations within an unfolded optimization setting has been considered previously [12, 37, 41, 45]. However, prior work has largely relied on proximal operators to create non-linear activations paired with simple linear filters, and the analysis does not transfer to the more complex Transformer case under investigation here. While we will restrict our attention to adding ReLU activations in this section, both for simplicity of exposition and because of their ubiquity in Transformer models, our results generalize to broader selections.

Similar to Section 4, we first study a general form of the optimization problem by introducing a ReLU activations into Algorithm 1, formulating this modification as AIM with proximal steps, and then analyzing convergence under appropriate constraints. Later we apply these results to the Transformer.

5.1 Proximal-Alternating Inexact Minimization

It has already been established [24, 37] that ReLU activations can be modeled as a proximal operator via

$$\mathrm{ReLU}(\mathbf{y}) = \arg\min_{\boldsymbol{z}} \frac{1}{2\lambda}\|\boldsymbol{z}-\mathbf{y}\|^2 + \phi(\boldsymbol{z}), \tag{17}$$

Algorithm 2

For the t-th iteration, execute
$\boldsymbol{u}^{(t)} = \mathbf{y}^{(t)} - \alpha_1 \nabla f\left(\mathbf{y}^{(t)}\right); \quad \boldsymbol{v}^{(t)} = \boldsymbol{u}^{(t)} - \alpha_2 \nabla g\left(\boldsymbol{u}^{(t)}\right); \quad \mathbf{y}^{(t+1)} = \mathrm{ReLU}\left(\boldsymbol{v}^{(t)}\right).$

where ϕ is the indicator function for the first quadrant given by $\phi(z) = \begin{cases} +\infty & \text{if } z < 0 \\ 0 & \text{if } z \geq 0 \end{cases}$. Moreover, in Section 4, we have demonstrated how the steps of Algorithm 1 together form an inexact gradient descent iteration of the loss $h(\mathbf{y}) = f(\mathbf{y}) + g(\mathbf{y})$ with noisy term Δ_t defined in (14). Here, with the addition of the proximal step in Algorithm 2, we obtain an inexact version of proximal-gradient descent. In fact, one turn of Algorithm 2 is equivalent to

$$\mathbf{y}^{(t+1)} = \arg\min_{\boldsymbol{z}} \frac{1}{2\lambda} \left\| \boldsymbol{z} - \mathbf{y}^{(t)} + \alpha_2 \nabla h\left(\mathbf{y}^{(t)}\right) - \alpha_2 \Delta_t \right\|^2 + \phi(\boldsymbol{z}), \tag{18}$$

which is an inexact version of canonical proximal-gradient step with noise Δ_t, as compared to the exact version $\mathbf{y}^{(t+1)} = \arg\min_{\boldsymbol{z}} P\left(\boldsymbol{z}; \mathbf{y}^{(t)}\right)$, where $P(\boldsymbol{z}; \mathbf{y})$ is the proximal problem

$$P\left(\boldsymbol{z}; \mathbf{y}^{(t)}\right) = \frac{1}{2\lambda} \left\| \boldsymbol{z} - \mathbf{y}^{(t)} + \alpha_2 \nabla h\left(\mathbf{y}^{(t)}\right) \right\|^2 + \phi(\boldsymbol{z}). \tag{19}$$

While various forms of inexact proximal gradient descent have been studied in the past [9, 13, 35, 42], existing work either assumes constant noise [9], stochastic noise [42], or decreasing/convergent noise [13, 35]. Critically, no prior work that we are aware of applies to our cse where the noise can potentially increase with iterations. Moreover, existing analysis in the literature is primarily concerned with convergence to a fixed point, while in our case, we instead consider entering a specific region formed around certain points.

Moreover, in addition to bounding $\delta\left(\mathbf{y}^{(t)}\right)$ as in Section 4, we also need to bound the similarity between the current position $\mathbf{y}^{(t)}$ and the gradient $\alpha_2 \nabla h\left(\mathbf{y}^{(t)}\right)$, which is defined as $\mathfrak{D}(\boldsymbol{\xi}_1, \boldsymbol{\xi}_2) = \frac{1}{\|\boldsymbol{\xi}_1\|^2} \sum_{i=1}^{d} \min(\boldsymbol{\xi}_{2,i}^2 - \boldsymbol{\xi}_{1,i}^2, 0)$. Intuitively, $\mathfrak{D}(\boldsymbol{\xi}_1; \boldsymbol{\xi}_2)$ is defined such that each term of the summation is negative, but only close to 0 when $\boldsymbol{\xi}_{1,i}$ and $\boldsymbol{\xi}_{2,i}$ are both large. We then have the following:

Theorem 5.1. *If $\alpha_1 \leq \alpha_2 \leq L_h^{-1}$, $\mathfrak{D}\left(\alpha_2 \nabla h\left(\mathbf{y}^{(t)}\right); \mathbf{y}^{(t)}\right) \geq -\kappa$ for any $\kappa \in (0,1)$, and $\delta\left(\mathbf{y}^{(t)}\right) \leq \mathscr{C}'$, where $\mathscr{C}' = \frac{\alpha_2 c_P \lambda \sqrt{1-\kappa}}{\sqrt{2}(\alpha_2 - \alpha_1 + \alpha_1 \alpha_2 L_g)}$, we have that $h\left(\mathbf{y}^{(t+1)}\right) + \phi\left(\mathbf{y}^{(t+1)}\right) \leq h\left(\mathbf{y}^{(t)}\right) + \phi\left(\mathbf{y}^{(t)}\right)$.*

Intuitively, Theorem 5.1 shows that the region where the descent of $h(\mathbf{y}) + \phi(\mathbf{y})$ is guaranteed is the intersection of $\mathcal{S}(\mathscr{C}')$ with $\mathcal{S}$ defined in Lemma 4.3, and the area $\mathcal{T}(\kappa) = \{\mathbf{y} \,|\, \mathfrak{D}(\alpha_2 \nabla h(\mathbf{y}); \mathbf{y}) \geq -\kappa\}$. While the shape of $\mathcal{T}(\kappa)$ remains difficult to specify in general, we note that $\mathcal{T}(\kappa)$ tends to the whole space when $\alpha_2 \to 0$ or $\kappa \to 1$. For example, we illustrate the area of $\mathcal{T}(\kappa)$ using a 2D synthetic example, where the energy function is $h(\mathbf{y}) = \|W\mathbf{y}\|_{\mathcal{F}}^2 + \|\mathbf{y} - \boldsymbol{b}\|^2$, and entries of W and $\boldsymbol{b}$ are randomly generated. See Figure 4 for the visualization using different values κ, which shows that when κ is sufficiently small, $\mathcal{T}(\kappa)$ is nearly the whole space (except a small area around the origin) which ensures the descent of $h(\mathbf{y}) + \phi(\mathbf{y})$ in most cases.

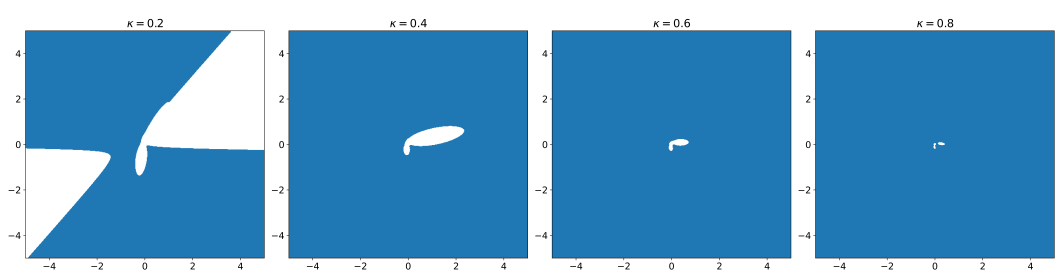

Figure 4: $\mathcal{T}(\kappa)$ region (blue area) of a 2D synthetic example with different values of κ.

5.2 Embedding the Non-Linearity within Transformer Models

By adding the penalty term to the previous energy, and with E_1 and E_2 obtained from Sections 3.1 and 4.2 respectively, we finally arrive at the total Transformer energy $E(Y) = E_1(Y) + E_2(Y) + \phi(Y)$.

And the unfolded optimization of $E(Y)$ falls into the scope of Algorithm 2, such that the former analysis and Theorem 5.1 apply and we may conclude that the aggregated update rule

$$Y^{(t+1)} = \mathrm{ReLU}\left[\mathrm{softmax}_{\boldsymbol{\beta}}\left(Y^{(t)} W_a^s Y^{(t)\top}\right) W_f^s\right] \tag{20}$$

is a descent algorithm of E, except a region with finite measure. Moreover, if we modify the underlying gradient descent algorithm to make the step sizes α_1 and α_2 sufficiently small, the size of the exception region will tend to zero, and the update rule will be equipped with a residual term (see supplementary; similarly for a discussion about $\boldsymbol{\beta}$). Besides these issues, the unfolded update (20) only differs from (8) in its reliance on symmetric weights. This corresponds with Challenge 4 that we have not fully solved, although it has been shown that symmetric weights can mimic asymmetric weights if we enlarge the representation dimensions [19, 46].

6 Practical Verification

Although we have rigorously derived convergence criteria whereby Transformer layers descend a well-specified energy function to a region around optimal solutions, the analysis admittedly relies on conditions that would be difficult to formally verify on real-world datasets. However, our results are nonetheless amenable to targeted empirical corroboration, whereby we can check if the proposed energy does in fact descend during the Transformer forward pass on typical real-world benchmarks.

To this end, we implement a Transformer model as described previously, up to known limitations like symmetric weights. We apply this model to two datasets, IMDB [29] and SST2 [36], which are both commonly-used sentiment classification datasets that use Glove-840b-300d [32] as the word embedding. Figures 5 and 6 display the energy of the output of each layer averaged over 200 randomly chosen samples in the test set.

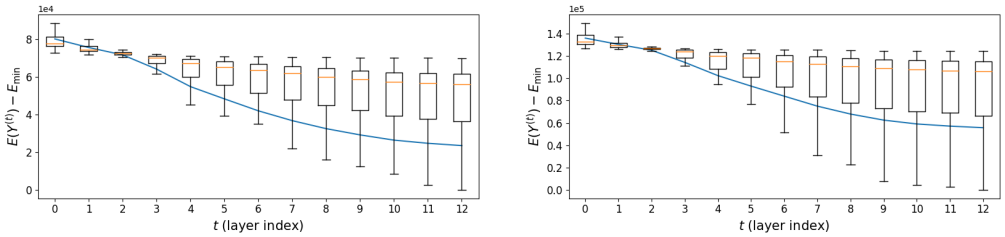

Figure 5: Box plot of the energy curve of a 12-layer Transformer on IMDB (*left*) and SST2 (*right*) datasets; results are averaged over samples.

From these figures it is clear that even with real-world data, the the Transformer energy we have derived is (on average) monotonically decreasing across layers, matching the predictions of our analysis. Moreover, with 12 layers (which represents a depth range not uncommon in practice) the model has not entered the fluctuation region. Hence although seemingly complicated, the conditions adopted by Lemma 4.3 and Theorem 5.1 are nonetheless likely hold in many practical settings.

7 Conclusion

While our contributions here have been primarily of a theoretical nature, there are nonetheless take-home messages that may be of practical relevance. First, as our overriding goal was to reproduce Transformer layers as closely as possible using an unfolded optimization perspective, very specific design choices were made in constructing the core underlying energy function. However, in practice we are free to choose alternative energies that can lead to different forms of tailored self-attention that may be be advantageous on an application-specific basis. Additionally, the distribution of attention weights can potentially be better understood or influenced by properties of the energy function that produces them. And finally, especially for regimes with limited data and therefore fewer free model parameters, the unfolding perspective can be used to devise architectures with inductive biases aligned with downstream tasks to help compensate for less model flexibility.

We conclude by noting that the techniques introduced in this paper can in some sense be viewed as universal tools for constructing a broader family of unfolding architectures. In fact, many/most deep learning models, including those with feed-forward structure [25], token-level interactions [40], residual connections [17], and heterogeneous layer types, can be reinterpreted through the lens of unfolded optimization using the framework we have introduced, at least up to some potential constraints like symmetric weights (which may be handled in other ways).

References

[1] Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR, 2017.

[2] Brandon Amos, Lei Xu, and J. Zico Kolter. Input convex neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 146–155. PMLR, 2017.

[3] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Rev.*, 60(2):223–311, 2018.

[4] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Velickovic. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *CoRR*, abs/2104.13478, 2021.

[5] Siheng Chen and Yonina C Eldar. Graph signal denoising via unrolling networks. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5290–5294, 2021.

[6] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509, 2019.

[7] Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Ann. Oper. Res.*, 153(1):235–256, 2007.

[8] Patrick L Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, pages 185–212. Springer, 2011.

[9] Olivier Devolder, François Glineur, and Yurii E. Nesterov. First-order methods of smooth convex optimization with inexact oracle. *Math. Program.*, 146(1-2):37–75, 2014.

[10] Jordan Frecon, Saverio Salzo, Massimiliano Pontil, and Gilles Gasso. Bregman Neural Networks. 2022.

[11] Zhengyang Geng, Meng-Hao Guo, Hongxu Chen, Xia Li, Ke Wei, and Zhouchen Lin. Is attention better than matrix decomposition? In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[12] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *International Conference on Machine Learning*, 2010.

[13] Bin Gu, De Wang, Zhouyuan Huo, and Heng Huang. Inexact proximal gradient methods for non-convex and non-smooth optimization. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3093–3100. AAAI Press, 2018.

[14] Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. Star-transformer. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 1315–1325. Association for Computational Linguistics, 2019.

[15] Moritz Hardt. Understanding alternating minimization for matrix completion. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 651–660. IEEE Computer Society, 2014.

[16] Hao He, Bo Xin, Satoshi Ikehata, and David Wipf. From bayesian sparsity to gated recurrent nets. In *Advances in Neural Information Processing Systems*, 2017.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[18] John Hershey, Jonathan Le Roux, and Felix Weninger. Deep unfolding: Model-based inspiration of novel deep architectures. *arXiv preprint arXiv:1409.2574*, 2014.

[19] Shell Xu Hu, Sergey Zagoruyko, and Nikos Komodakis. Exploring weight symmetry in deep neural networks. *Comput. Vis. Image Underst.*, 187, 2019.

[20] Simon Jenni and Paolo Favaro. Deep bilevel learning. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part X*, volume 11214 of *Lecture Notes in Computer Science*, pages 632–648. Springer, 2018.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[23] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[24] Jia Li, Mingqing Xiao, Cong Fang, Yue Dai, Chao Xu, and Zhouchen Lin. Training neural networks by lifted proximal operator machines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[25] Hanxiao Liu, Zihang Dai, David R. So, and Quoc V. Le. Pay attention to mlps. *CoRR*, abs/2105.08050, 2021.

[26] Xiaorui Liu, Wei Jin, Yao Ma, Yaxin Li, Hua Liu, Yiqi Wang, Ming Yan, and Jiliang Tang. Elastic graph neural networks. In *International Conference on Machine Learning*, 2021.

[27] Songtao Lu, Mingyi Hong, and Zhengdao Wang. PA-GD: on the convergence of perturbed alternating gradient descent to second-order stationary points for structured nonconvex optimization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 4134–4143. PMLR, 2019.

[28] Yao Ma, Xiaorui Liu, Tong Zhao, Yozen Liu, Jiliang Tang, and Neil Shah. A unified view on graph neural networks as graph signal denoising. *arXiv preprint arXiv:2010.01777*, 2020.

[29] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In Dekang Lin, Yuji Matsumoto, and Rada Mihalcea, editors, *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 142–150. The Association for Computer Linguistics, 2011.

[30] S. Deepak Narayanan, Aditya Sinha, Prateek Jain, Purushottam Kar, and Sundararajan Sellamanickam. IGLU: efficient GCN training via lazy updates. *CoRR*, abs/2109.13995, 2021.

[31] Xuran Pan, Shiji Song, and Gao Huang. A unified framework for convolution-based graph neural networks, 2021.

[32] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL, 2014.

[33] Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Thomas Adler, David P. Kreil, Michael K. Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[34] R. Tyrrell Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[35] Mark Schmidt, Nicolas Le Roux, and Francis R. Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 1458–1466, 2011.

[36] Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*, pages 455–465. The Association for Computer Linguistics, 2013.

[37] Pablo Sprechmann, Alex Bronstein, and Guillermo Sapiro. Learning efficient sparse and low rank models. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 37(9), 2015.

[38] Ying Sun, Prabhu Babu, and Daniel P Palomar. Majorization-minimization algorithms in signal processing, communications, and machine learning. *IEEE Transactions on Signal Processing*, 65(3):794–816, 2016.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[40] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[41] Zhangyang Wang, Qing Ling, and Thomas Huang. Learning deep ℓ_0 encoders. In *AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[42] Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM J. Optim.*, 24(4):2057–2075, 2014.

[43] Xingyu Xie, Qiuhao Wang, Zenan Ling, Xia Li, Yisen Wang, Guangcan Liu, and Zhouchen Lin. Optimization induced equilibrium networks. *arXiv preprint arXiv:2105.13228*, 2021.

[44] Chaoqi Yang, Ruijie Wang, Shuochao Yao, Shengzhong Liu, and Tarek F. Abdelzaher. Revisiting "over-smoothing" in deep gcns. *CoRR*, abs/2003.13663, 2020.

[45] Yongyi Yang, Tang Liu, Yangkun Wang, Jinjing Zhou, Quan Gan, Zhewei Wei, Zheng Zhang, Zengfeng Huang, and David Wipf. Graph neural networks inspired by classical iterative algorithms. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 11773–11783. PMLR, 2021.

[46] Yongyi Yang, Yangkun Wang, Zengfeng Huang, and David Wipf. Implicit vs unfolded graph neural networks. *CoRR*, abs/2111.06592, 2021.

[47] Alan L. Yuille and Anand Rangarajan. The concave-convex procedure (CCCP). In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 1033–1040. MIT Press, 2001.

[48] Hongwei Zhang, Tijin Yan, Zenjun Xie, Yuanqing Xia, and Yuan Zhang. Revisiting graph convolutional network on semi-supervised node classification from an optimization perspective. *CoRR*, abs/2009.11469, 2020.

[49] Kaixiong Zhou, Xiao Huang, Daochen Zha, Rui Chen, Li Li, Soo-Hyun Choi, and Xia Hu. Dirichlet energy constrained learning for deep graph neural networks. *CoRR*, abs/2107.02392, 2021.

[50] Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. Interpreting and unifying graph neural networks with an optimization framework. *arXiv preprint arXiv:2101.11859*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] Our work is mostly theoretical and devoted to the better understanding of an existing, well-established modeling framework; we do not see any clear pathway to negative societal impact.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] They are in the supplementary.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]