Catformer: Designing Stable Transformers via Sensitivity Analysis

Jared Quincy Davis [*1 2] **Albert Gu** [*1] **Krzysztof Choromanski** [3 4] **Tri Dao** [1] **Christopher Re** [1] **Chelsea Finn** [1 3] **Percy Liang** [1]

Abstract

Transformer architectures are widely used, but training them is non-trivial, requiring custom learning rate schedules, scaling terms, residual connections, careful placement of submodules such as normalization, and so on. In this paper, we improve upon recent analysis of Transformers and formalize a notion of sensitivity to capture the difficulty of training. Sensitivity characterizes how the variance of activation and gradient norms change in expectation when parameters are randomly perturbed. We analyze the sensitivity of previous Transformer architectures and design a new architecture, the Catformer, which replaces residual connections or RNN-based gating mechanisms with con*cat*enation. We prove that Catformers are less sensitive than other Transformer variants and demonstrate that this leads to more stable training. On DMLab30, a suite of high-dimension reinforcement tasks, Catformer outperforms other transformers, including Gated Transformer-XL—the state-of-the-art architecture designed to address stability—by 13%.

1. Introduction

Transformer architectures (Vaswani et al., 2017; Choromanski et al., 2020b) are widely used in natural language processing (Luo et al., 2020; Chen et al., 2018b), vision (Dosovitskiy et al., 2020) and bioinformatics (Rives et al., 2019; Madani et al., 2020; Li, 2019). This success is due to their ability to effectively model long-range dependencies as well as scale effectively with data and compute (Kaplan et al., 2020).

However, Transformers are hard to train, limiting their effectiveness in challenging and noisy domains such as reinforcement learning (RL). Even in the supervised learning setting, to ameliorate these challenges, they require a combination of techniques such as complex optimizers and learning rate schedules (Dai et al., 2019), special weight initialization (Radford et al., 2019), and careful ordering of components such as normalization blocks (Parisotto et al., 2020). Architecturally, recent Transformer variants designed for stability have emerged, introducing new components (Liu et al., 2020) or replacing the residual connection with RNN-style gates (Xu et al., 2020; Parisotto et al., 2020). Only with these recent changes have transformers made progress in reinforcement learning.

These modifications are not specific to Transformers *per se*, but rather refine the general architecture wiring—such as residual connections and normalization layers—which can be shown to improve training stability in deep models at large. More broadly, these improvements are guided by general principles in deep learning that aim to control standard measures of stability: conventionally, the activation and gradient variance (i.e., preventing the oft-studied vanishing/exploding gradient problem (Hochreiter, 1991)). However, these measures can be too coarse to capture the true source of instability. For example, controlling activation variance at initialization does not preclude instability as the model's weights change during training (Choromanski et al., 2020a; Likhosherstov et al., 2020),. Standard tools of capturing how the output might change–such as measuring gradient norms and Lipschitzness–are too pessimistic and can be difficult to analyze.

This paper makes two main contributions. The first is introducing the concept of sensitivity, a measure of average-case instability that captures a notion of how sensitive a function is to random perturbations of its inputs. Our second main contribution is a new Transformer architecture inspired by the sensitivity analysis which empirically reduces instability.

First, we formalize and generalize ideas in Liu et al. (2020) to introduce an improved measure of stability called **sensitivity**, a function of architecture that measures the effect of random parameter perturbations on the output variance. Sensitivity can be applied to analyze a broad class of architectures composed of repeated building blocks of normalization layers, generalized residual connections, and black-box computation modules such as the self-attention

[*]Equal contribution [1]Stanford University [2]DeepMind [3]Google Brain Robotics [4]Columbia University. Correspondence to: Jared Davis <jaredquincydavis@gmail.com>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

in Transformers. Our theory develops a set of properties and composition rules for sensitivity, which allows us to analyze the sensitivity of many variants of Transformers. We show that sensitivity perfectly predicts the ability of various architectures to solve a synthetic language modeling task as network depth grows.

Motivated by this new metric, we propose and analyze the **Catformer**, a new Transformer architecture based on *concatenation* instead of residual additions. Within the family of low sensitivity models, this architecture is the only one where the gradient variance scales proportionally to the weight variance, closely matching the regime where sensitivity analysis applies. We conjecture that this additional property endows the Catformer with increased stability properties. Empirically, we confirm that compared to Transformer baselines, the Catformer is more robust to the choice of optimizer and to noisy gradients of the sort often found in reinforcement learning.

Finally, we apply the Catformer on visual-navigation tasks from DeepMind Lab (Beattie et al., 2016), a suite of challenging RL tasks with high-dimensional observations, complex action spaces, and partial observability. We show that Catformers consistently obtain mean scores greater than 10 percent above those of state of the art architectures designed for stability (Parisotto et al., 2020).

2. Background and Preliminaries

We introduce notation for defining deep neural networks such as the Transformer (Section 2.1), and review existing methods of capturing the instability of training (Section 2.2).

2.1. Architectures

We consider deep neural networks defined by repeating equations (1), (2), and (3).

$$\hat{x}_{i-1} = \mathsf{norm}(x_{i-1}) \quad \text{(Normalization)} \tag{1}$$

$$y_i = F_i(\hat{x}_{i-1}) \quad \text{(Module)} \tag{2}$$

$$x_i = \begin{cases} x_{i-1} \oplus y_i & \text{(Combination, pre-norm)} \\ \hat{x}_{i-1} \oplus y_i & \text{(Combination, post-norm)} \end{cases} \tag{3}$$

A network of depth N repeats this block and outputs $\hat{x}_N$. Let d denote the dimension of module outputs, i.e. $y_i \in \mathbb{R}^d$.

A specific model can be instantiated by making three choices (Figure 1):

1. the black-box computation module F_i
2. the normalization placement (Eq. (3))
3. a combination function $\oplus$ that processes the inputs and outputs of the module (for example, often chosen to be a "residual connection" $x \oplus y = x + y$)

By varying the components in Eqs. (1) to (3), a variety of DNN models have been proposed, as described next.

We remark that since our analysis primarily focuses on the surrounding architecture and not the module, for the sake of clarity it is useful to think of $F_i(x) = W_i x$ as a simple linear layer.[1]

Special Case: Transformers They are an instantiation of this general architecture wherein the modules alternate between two types of layers $F_i(x) = \mathsf{MHA}(x)$ and $F_i(x) = \mathsf{FF}(x)$. *Multihead self-attention* (MHA) is the characteristic component of Transformers, defined as

$$\mathsf{MHA}(x) = \mathsf{softmax}((xW_i^Q)(xW_i^K)^\top)(xW_i^V)W_i^P \tag{4}$$

These are interleaved with *positionwise feed-forward* (FF) layers using a nonlinear activation function such as $\sigma = \mathsf{ReLU}$

$$\mathsf{FF}(x) = \sigma(xW_i^{F_1})W_i^{F_2}. \tag{5}$$

Additionally, the standard transformer chooses the residual function $x \oplus y = x + y$ with either post- (Vaswani et al., 2017) or pre- (Radford et al., 2019) normalization.

Other Instantiations We note that outside of transformers, neural networks of this form have been frequently used with other modules F_i – most notably CNNs which use a convolution module. To improve convolution network scaling, recent works such as Huang et al. (2017) have proposed replacing the residual combination function with "dense" connections between layers, which connects the output of every module to the input of every subsequent module. This can in fact still be represented by Eqs. (1) to (3) with a concatenation operation $\oplus$. These works have demonstrated that convolutional neural networks can be scaled and trained more effectively if they contain shorter pathways between early layers close to the input and deeper layers closer to the output (Srivastava et al., 2015; He et al., 2015a; 2016).

Such ideas can be directly transferred to the transformer setting. Recent work on GTrXL (Parisotto et al., 2020) and Transformers with Depth-wise LSTMs (Xu et al., 2020) replaced the standard transformer residual combination function $x \oplus y = x + y$ with RNN-based gating mechanisms which function as learned weighted combination functions.

2.2. Theoretical metrics for stability

We summarize here several theoretical lenses via which to view network stability. The main source of instability in training neural networks comes from excessively large function values or gradients. To capture this instability, especially at model initialization, one can measure the magnitude

[1]For brevity and clarity, all linear layers are described without an explicit bias term; adding one does not affect the analysis.

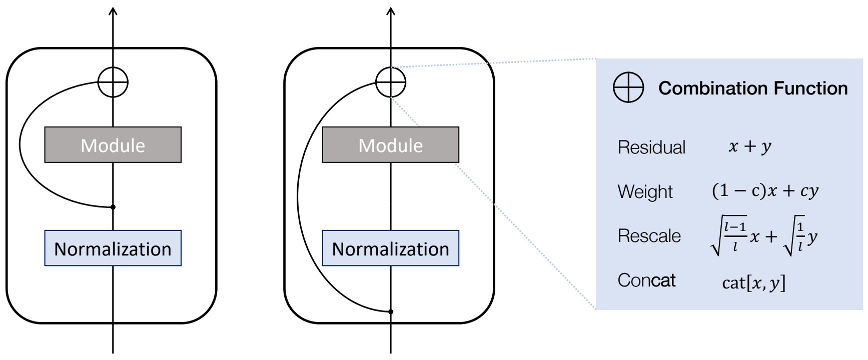

Figure 1. We analyze architectures of repeated blocks, composed of a choice of normalization layer placement (Left: post-norm, Middle: pre-norm), black-box module, and combination function $\oplus$.

of the activations or gradients, averaged over random inputs, weights, or perturbations. This motivates three classes of metrics: activation variance, gradient variance, and output change.

Activation variance The most direct measure of instability is *activation variance*, the variance of the intermediate outputs y_i of the model, which can be defined as $\mathrm{Var}(\|y_i\|_2^2/d)$. If this variance is high, then the magnitude of the activations (i.e., elements of y_i) can vary widely, leading to unstable model outputs.

A well-established guideline in neural network design is keeping this activation variance controlled through the depth of a network, commonly achieved through careful initialization (He et al., 2015b; Glorot & Bengio, 2010). For example, a fundamental result is that any weight matrix $W \in \mathbb{R}^{m\times n}$ W projecting dimension $m \to n$ should be initialized with i.i.d. entries of variance proportional to $\frac{1}{n}$, so that if input x has variance 1 then $(Wx)_i = \sum_j W_{ij}x_j$ has variance $O(1)$ (where the constant may change depending on the subsequent non-linear activation function) (Glorot & Bengio, 2010).

A second common theme in deep learning theory and architecture design is handling the fact that residual connections increase the activation variance: $\mathrm{Var}(x+F(x)) \approx 2\,\mathrm{Var}(x)$ if $F(x)$ is variance-preserving and decorrelates the output from input. The role of normalization layers (such as BatchNorm (Ioffe & Szegedy, 2015) or LayerNorm (Ba et al., 2016)) is largely to control this issue, and other architectures that do not involve normalization are motivated chiefly by controlling activation variance. For example, the recently proposed RescaleNet (Shao et al., 2020) addresses this by weighting the residual function from $x+y$ to $\sqrt{\frac{i-1}{i}}x+\sqrt{\frac{1}{i}}y$ in order to keep the activation variance constant after the residual. Note that such a re-weighting of the residual function falls under our framework, as it can be viewed as a different instantiation of the $\oplus$ combination function.

Intuitively, controlling activation variance is particularly important for Transformer models, because the recurring self-attention layer (4) involves the softmax function which is highly sensitive to the input "temperature" or variance (Vaswani et al., 2017).

Gradient variance While controlling activation variance most directly affects stability by ensuring the magnitude of intermediate activations are controlled in the forward pass, one can also ensure that the magnitude of the gradients is controlled during the backward pass. More precisely, the *gradient variance* is $\mathrm{Var}\left\|\frac{\partial f(\theta)}{\partial \theta_i}\right\|_2^2$, where θ_i is the set of parameters of the i-th layer. This variance should not grow or shrink exponentially with the depth i, since the former causes instability and the latter prevents effective training with gradient descent. Failure to control this quantity is often referred to as the "exploding/vanishing gradients" (EVG) problem (Hochreiter, 1991), which can occur in long computation graphs for recurrent or deep networks. Ensuring that a proposed model does not suffer from EVG traditionally requires analyzing by hand the backward pass of the model, a problem which can be alleviated by our sensitivity framework (Theorem 1).

Output change Although activation and gradient variance are the primary measures of the stability of a randomly initialized network, some works have noticed that it can be helpful to directly control for the amount that a network's output changes in response to a change in the input. For example, (Zhang et al., 2019) design an initialization scheme so that the overall model's output changes by $O(1)$ after a step of gradient descent.

The notion of *amplification* that (Liu et al., 2020) investigate is another type of output change: loosely speaking, this idea is about how much a small (random) perturbation to the weights is increased through the network. They heuristically and empirically show that a pre-norm residual

Transformer "amplifies" perturbations by a smaller factor than a post-norm one, and argue that this helps explain the greater stability of pre-norm architectures. Our definition of sensitivity is based upon this idea, and we discuss the relation to amplification in depth in Appendix A.

3. Sensitivity

We define our core notion of the **sensitivity** of a model in Section 3.1. Section 3.3 fleshes out theoretical properties of sensitivity, including composition rules that allow it to be easily computed. Section 3.4 illustrates an example of how to use these properties to calculate the sensitivity of the most common architecture, the pre-norm residual network. Section 3.5 provides results showing the calculated sensitivities of several established architectures. Finally, Section 3.6 discusses the deeper consequences of sensitivity, especially how it ties together the stability metrics of activation variance, gradient variance, and output change.

3.1. Sensitivity

Sensitivity is motivated by the observation that even when a model has controlled activation variance at initialization, it might become unstable as model parameters change during training. Sensitivity captures this rate of change in an average sense. Informally, sensitivity is the variance of the change in output when randomly initialized parameters are perturbed a small amount in a random direction.

Sensitivity is formally defined in Definition 1. We also define $\mathbb{V}_p[f(\theta)]$ to be the second moment $\mathbb{E}_{\theta\sim p(\theta)}[\|f(\theta)\|_2^2]$.[2]

Definition 1 (Sensitivity)**.** Let $f(u, v)$ be a function of some (vector) inputs and $p(u, v)$ be a distribution over these inputs such that each (scalar) input is independent.

The *sensitivity* of f with respect to u and p, denoted $\mathbb{S}_{u,p}[f]$, is

$$\frac{1}{\mathbb{V}_p[f(u,v)]}\lim_{\delta\to 0}\frac{1}{\delta^2}\mathbb{E}_{\substack{(u,v)\sim p(u,v)\\ \tilde{u}\sim p(u)}}\left[\|f(u+\delta\tilde{u},v)-f(u,v)\|_2^2\right]$$

When $p(\theta)$ or W are understood from context, they can be omitted from the subscript.

To illustrate Definition 1, suppose we have a model $f(\theta; x)$ of parameters θ and inputs x, along with a canonical distribution over each (e.g. x sampled from i.i.d. Gaussian and θ chosen so that the model is variance-preserving at every layer). The idea behind sensitivity is to measure the following process: (1) initialize the parameters θ randomly and pass in random inputs x (2) perturb the parameters θ by an independent amount *from the same distribution*, and (3) measure the difference in output before and after the perturbation. Normalizing appropriately, this quantity is described exactly by $\mathbb{S}_\theta f(\theta; x)$. For example, $\mathbb{S}x_i, \mathbb{S}\hat{x}_i, \mathbb{S}y_i$ are the sensitivities of the intermediate outputs of an architecture defined by Eqs. (1) to (3) in this sense.

[2]We use Var to denote variance and $\mathbb{V}$ for this second moment; note that they coincide for 0-mean random variables, which is usually the case in our setting (i.e., for intermediate computations of neural network architectures).

3.2. Discussion

3.3. Properties of Sensitivity

We first show an alternate characterization of sensitivity as a type of expected gradient (Proposition 1), which is useful for proving modularity properties of sensitivity (Propositions 2 and 3). For simplicity and clarity, these properties are stated for the case when the sensitivity is taken with respect to all inputs of a function (i.e., $W = \theta$ in Definition 1). In the case of a neural network, this can be thought of as when both the weights θ and the inputs x are perturbed.

Proposition 1.

$$\mathbb{S}_p[f(\theta)] = \mathbb{V}_p[f(\theta)]^{-1}\sum_{\theta_k}\mathbb{V}_p(\theta_k)\mathbb{V}_p(\nabla_{\theta_k} f)$$

where the sum is over individual (scalar) parameters $\theta_k \in \theta$.

Proposition 2 (Composition rules)**.** *Sensitivity satisfies several local composition rules, including:*

- ***Identity*** $\mathbb{S}[\theta] = 1$.
- ***Sum*** *If $\nabla f(\theta)$ and $\nabla g(\theta)$ are uncorrelated given $\theta \sim p$, then $\mathbb{S}[f(\theta) + g(\theta)] = \mathbb{S}f\frac{\mathbb{V}f}{\mathbb{V}f+\mathbb{V}g} + \mathbb{S}g\frac{\mathbb{V}g}{\mathbb{V}f+\mathbb{V}g}$.*
- ***Product*** *For disjoint sets of parameters θ_1, θ_2, $\mathbb{S}[f(\theta_1)g(\theta_2)] = \mathbb{S}[f(\theta_1)] + \mathbb{S}[g(\theta_2)]$.*
- ***Chain rule*** $\mathbb{S}[f \circ g] = \mathbb{S}[f]\mathbb{S}[g]$.

Proposition 3 (Invariance rules)**.** *Sensitivity is invariant to the following transformations:*

- ***Normalization*** *If $y = \mathsf{norm}(x)$ then $\mathbb{S}y = \mathbb{S}x$.*
- ***Reparameterization*** *Consider the function $g(\theta') = f(c\theta')$ with distribution $q(\theta') = p(\frac{1}{c}\theta')$. Then $\mathbb{S}_{\theta',q}g(\theta') = \mathbb{S}_{\theta,p}f(\theta)$.*

To understand the Reparameterization rule, note that although the distributions of g and f are identical, some quantities such as the gradients to the parameters become scaled. However, the sensitivity is invariant. Beyond a global scalar c, this rule applies more generally if each individual parameter $\theta_k \in \theta$ is rescaled by a separate c_k.

Finally, there is an alternative method to the composition rules that can be used to calculate sensitivities in special cases, in particular for any architecture that uses (weighted) residual connections. An informal version of this property for two cases was discussed in (Liu et al., 2020).

Proposition 4 (Layer decomposition)**.** *Suppose that $x \oplus y = \alpha_i x + \beta_i y$ is any weighted residual function where α_i, β_i are independent of x, y (but can depend on depth i). Then*

$$\mathbb{S}[\hat{x}_i] = \sum_{j \le i} \rho_j, \qquad \rho_i = \frac{\beta_i^2 \mathbb{V} y_i}{\mathbb{V} x_i} \mathbb{S}[F_i]$$

Note that ρ_i is a local quantity about layer i. Thus, Proposition 4 shows that sensitivity can be calculated for some architectures by calculating ρ_i locally, which is sometimes easier than following the composition rules.

3.4. Example: pre-norm residual network

We present a simple example of how to use the properties of sensitivity to calculate the sensitivity of the most ubiquitous deep neural network architecture. The pre-norm residual network (He et al., 2016; Radford et al., 2019) is defined by equations Eqs. (1) to (3). Let the input be denoted $x_0 = \hat{x}_0$ with unit variance. For simplicity, let the module be a linear layer $F_i(x) = W_i x$ with no bias, where the parameters W_i are initialized to be variance-preserving (i.e., i.i.d. entries with variance $1/d$)[3]. We will calculate the sensitivities of this architecture with respect to the parameters W_i.

Method 1 First, we calculate the variances:

- $\mathbb{V}\hat{x}_{i-1} = 1$ (definition of normalization)
- $\mathbb{V}y_i = 1$ (variance-preserving module)
- $\mathbb{V}x_i = \mathbb{V}x_{i-1} + \mathbb{V}y_i = i + 1$

Next, we calculate the sensitivities using the rules in Propositions 2 and 3. $\mathbb{V}\hat{x}_0 = 0$ since perturbing the parameters does not change the input, and inductively[4]:

- $\mathbb{S}x_{i-1} = H_i - 1$ (inductive hypothesis)
- $\mathbb{S}\hat{x}_{i-1} = H_i - 1$ (Normalization)
- $\mathbb{S}y_i = \mathbb{S}W_i + \mathbb{S}\hat{x}_{i-1}$ (Product) $= H_i$ (Identity)
- $\mathbb{S}x_i = (H_i - 1)\frac{i}{i+1} + H_i \frac{1}{i+1} = H_{i+1} - 1$ (Sum)

Method 2 Because $\oplus$ uses an addition here, we can use Proposition 4 as a shortcut. By the previous variance calculations, we know $\rho_i = \frac{1}{i+1}$, and summing over $1, \dots, i$ gives $\mathbb{S}\hat{x}_i = H_{i+1} - 1$.

3.5. Sensitivity Results for Established Architectures

By applying the properties of sensitivity in Section 3.3, we derive sensitivities for an assortment of previously considered architectures by varying the norm position and combination function $\oplus$. These results are summarized in Table 1. Appendix C contains derivations of these results and empirical plots confirming the scaling with depth.

[3]Also note that this ensures that inputs and outputs to the module are decorrelated.

[4]Let $H_n = 1 + \cdots + 1/n$ denote the n-th harmonic number.

We observe that these sensitivities fall in two categories, that scale either linearly or logarithmically in depth. Empirically, Liu et al. (2020) suggested that the higher scaling of post-norm residual Transformers explains its more unstable training compared to the pre-norm variant. In Section 5, we find that this trend holds tightly across these two categories of models, where models with logarithmic sensitivity can solve synthetic tasks that those with linear sensitivity cannot.

3.6. Sensitivity and Stability

We now show that our notion of sensitivity is actually related to the existing notions of instability. Recall that three goals of stability are (1) controlling *activation variance* to ensure that output magnitudes remain stable through the model, (2) calculating the scaling of *gradient variance* through depth to ensure that gradients do not vanish or explode, and (3) analyzing *output change* to understand when the model is stable under parameter perturbations. Surprisingly, sensitivity connects these three notions of stability:

Theorem 1. *For any architecture that preserves activation variance through the depth of the network, the gradient variance at depth i is proportional to $\mathbb{S}x_i - \mathbb{S}x_{i-1}$.*

Theorem 1 says that sensitivity, a measure of output change, reveals the exact gradient variance scaling in depth, when activation variance is controlled. Thus all three measures can be tightly linked through the lens of sensitivity analysis.

We further note that Theorem 1 can be combined with Proposition 4 to show that the gradients to layer i are proportional to ρ_i, which is a completely local quantity that is often easy to calculate (e.g. Section 3.4). In contrast, the traditional approach of calculating gradient variance by hand can be difficult to analyze through normalization layers and complicated combination functions. For example, Liu et al. (2020) devote analysis to showing that gradient norms in a pre-norm residual network do not vanish through depth. By contrast, our Theorem 1 immediately reveals that they in fact increase slightly through depth at the exact rate of $1/i$.

4. The Catformer

Our modular definition of sensitivity (Section 3) motivates the search for new architectures with low sensitivity. One key component is the combination function, which traditionally is addition. In this section, we explore an alternative: concatenation, which more naturally preserves information across the layers. In particular, we propose the Catformer architecture that replaces the addition combination $x + y$ with the concatenation $\texttt{concat}[x, y]$.

Table 1. Stability metrics for several architectures: (i) Weight variance needed for stable activation variance (ii) Gradient variance to layer i (iii) Sensitivity

Name	$x \oplus y$	Norm position	Weight var. for stable activation var.	Layer i gradient var.	Sensitivity $\mathbb{S}[f(\theta)]$
Feedforward	y	any	$O(1)$	$O(1)$	N
Residual	$x+y$	no-norm	0	$O(1)$	$N/2$
Vaswani et al. (2017)		post-norm	$O(1)$	$O(1)$	$N/2$
He et al. (2016); Radford et al. (2019)		pre-norm	$O(1)$	$O(1/i)$	$\log(N)$
Weighted residual	$(1-g)x+gy$	post-norm	$O(1)$	$O(1)$	$N \cdot \frac{g^2}{g^2+(1-g)^2}$
		pre-norm	$O(1)$	$O(1)$	$N \cdot 1-(1-g)^2$
GTrXL (Parisotto et al., 2020)	$(1-\sigma(\cdot))x+\sigma(\cdot)y$	pre-norm	$O(1)$	$O(1)$	$\approx 0.22N$
RescaleNet (Shao et al., 2020)	$\sqrt{1-\frac{1}{i}}x+\sqrt{\frac{1}{i}}y$	any	$O(1)$	$O(1/i)$	$\log(N)$
AdMin (Liu et al., 2020)	$w_i x+y$	post-norm	$O(1)$	$O(1/i)$	$\log(N)$
Catformer (ours)	$\mathsf{concat}[x,y]$	any	$O(1/i)$	$O(1/i)$	$\log(N)$

4.1. Motivation: The Concatenation Combination

The composition rules of sensitivity (Section 3.3) allow us to derive sensitivities for architectures other than the addition-based combination functions previously considered. In particular, we derive the sensitivity of the concatenation combination function. As Table 1 shows, this model is also in the family of low sensitivity architectures.

However, this model is distinguished from the other low sensitivity architectures by the property that its dimension increases through depth. This turns out to have concrete implications for its stability. In particular, its *gradient variance scales proportionally to the weight variance*, as a function of depth: $\mathrm{Var}[\nabla_{\theta_i} f] \propto \mathrm{Var}[\theta_i]$. In other words, the 4th and 5th column in Table 1 are proportional.

This property connects sensitivity to the optimization dynamics of the model. In the context of training with gradient descent, the model parameter θ is perturbed by an amount proportional to its gradient: $\theta^{(k+1)} \leftarrow \theta^{(k)} - \eta \nabla_\theta f$. If each layer has gradient variance proportional the weight variance, then this perturbation of θ matches the perturbation in the definition of sensitivity (recall that in Definition 1, W is perturbed by $\delta W'$ with W' coming from the same distribution as W). This is the regime where sensitivity analysis (Section 3) applies. We conjecture that models with low sensitivity and whose *gradient variance scales proportionally to the weight variance* will thus see increased stability benefits during *optimization*.

Note that the gradient variance can be immediately calculated from the sensitivities by Theorem 1. The weight variance is also easy to calculate, as it must be inversely proportional to the fan-in dimension to control activation variance (Section 2.2). These quantities are reported in Table 1 along with the sensitivities.

The connection between gradient variance, weight variance, and gradient descent suggests that using the concatenation combination function $\oplus$, the only low-sensitivity model with this additional property, may be more stable than other low-sensitivity architectures on real optimization problems.

4.2. The Catformer

Based on the previous observation, we propose a new Transformer variant following the architecture structure of Section 2.1 and simply using $x \oplus y = \mathsf{concat}[x,y]$. We call the resulting architecture model the **Catformer**. Because the Catformer has the property that the dimension of the state passed between the modules (Fig. 1) grows with depth, we now discuss controlling the network size and the initialization.

Network size As concatenation increases the activation size, in order to fairly compare to other Transformer architectures, we control the number of parameters by adjusting the feature size and intermediate dimensions in the FF and MHA modules. We keep the dimension of y_i (equation (2)) to a constant m for all layers, so the size of the outputs x_i are $i \cdot m$. Thus the FF and MHA modules must project the input from dimension $i \cdot m \to m$. We set a feedforward expansion factor e_f so that the weight matrices in Eq. (5) have dimensions $W_i^{F_1} \in \mathbb{R}^{im \times e_f m}$ and $W_i^{F_2} \in \mathbb{R}^{e_f m \times m}$. Similarly, we set an attention expansion factor e_a so that the matrices in Eq. (4) have dimensions $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{im \times e_a m}$ and $W_i^P \in \mathbb{R}^{e_a m \times m}$.

Appendix D discusses how to set the factors e_a, e_f and the dimension m to match the parameter count of Catformer with standard Transformer variants.

Initialization Because these matrices in the FF and MHA modules are non-square, popular initialization schemes may have different behavior. We use Kaiming init. (He et al., 2015b) so that a linear layer of dimension $d_1 \to d_2$ has variance $\frac{1}{d_1}$ to control activation variance, instead of Xavier init. (Glorot & Bengio, 2010) that sets variance $\frac{2}{d_1+d_2}$.

5. Experiments

We validate our theory of sensitivity and our proposed Catformer architecture in three ways:

1. We train a variety of transformer architectures varying the normalization and combination function (Table 1), and find that sensitivity strongly predicts a model's ability to solve a standard synthetic task testing memory (Sec. 5.1).
2. We investigate regimes with noise , which more directly model the perturbation interpretation of sensitivity, and find that Catformer is considerably more robust than the other low sensitivity methods (Section 5.2).
3. We train models with deep RL, a noisy setting where vanilla transformers are notoriously difficult to train, and validate that Catformer successfully mitigates the stability challenge and outperforms the previous state-of-the-art architecture, GTrXL (Parisotto et al., 2020) (Section 5.3).

5.1. Synthetic Language Modeling

Task We use a standard synthetic memorization task popular for benchmarking the ability of sequence models to handle complex long-range dependencies. Following (Kitaev et al., 2020), the input is a sequence of the form $WZWZ$, where $W \in \{1, \dots, V-1\}^+$ is a sequence of symbols of length uniformly random in $[1, \frac{L}{2}-1]$ and $Z \in \{0\}^+$ pads the sequence to the target sequence length L. Models are trained with a language modeling objective, where the goal is to autoregressively predict the second half of symbols. We use $V = 64$ and $L = 512$ in our experiments.

Because we are in particular interested in the effect of depth on Transformer training instability, we increase the difficulty of this task in a way that biases models to prefer being deeper. We fix a permutation π, chosen to be the bit reversal permutation, which can be represented by a sorting circuit of depth $\log(N)$ (Dao et al., 2019). Thus, this permutation can be learned by a model of depth $O(\log(N))$, where each layer needs only to learn a simple structured permutation.

Models We use $N = 2$, 4, and 6 layer Transformer architectures. All models use pre-LayerNorm unless otherwise specified. For all baselines with constant dimension size in their layers (i.e. all models except Catformer), we use the standard Transformer architecture with $d = 512$ and inner dimension $4d = 2048$ in the inner layer of the feed-forward module. The Catformer model is parameter controlled with the technique described in Appendix D with $e_a = 2, e_f = 4$. Models are trained with the Adam optimizer.

Table 2. Perplexity on synthetic language modeling. (**Top**): methods with $O(N)$ sensitivity are unable to solve the task. (**Bottom**): methods with $\log(N)$ sensitivity are able to solve the task, and improve with depth.

Layers N	2	4	6
Feedforward	20.0 ± 7.1	29.2 ± 2.9	29.2 ± 2.9
Residual (no-norm)	1.14 ± 0.1	21.7 ± 3.9	Diverged
Residual (post-norm)	19.6 ± 4.6	26.9 ± 5.9	28.9 ± 2.7
Weighted $g = .5$	1.79 ± 0.9	7.46 ± 4.7	14.7 ± 0.8
GTrXL	1.14 ± 0.0	15.7 ± 0.9	17.2 ± 2.2
Residual (pre-norm)	1.29 ± 0.13	1.01 ± 0.02	1.01 ± 0.1
RescaleNet	1.16 ± 0.09	1.03 ± 0.03	1.02 ± 0.00
Catformer (no-norm)	1.17 ± 0.01	1.06 ± 0.01	1.03 ± 0.00
Catformer	1.23 ± 0.0	1.04 ± 0.02	1.03 ± 0.02

Table 3. Models trained with stochastic gradient descent instead of Adam (val. perplexity). (**Top**): Catformer, (**Bottom**): Transformer

LR	$N = 2$	4	6
0.1	4.69 ± 0.21	3.93 ± 2.14	6.30 ± 0.91
0.2	3.72 ± 1.01	1.19 ± 0.07	1.33 ± 0.17
0.4	1.57 ± 0.39	1.10 ± 0.07	1.04 ± 0.01
0.1	6.65 ± 1.51	7.96 ± 0.51	8.82 ± 0.56
0.2	4.19 ± 0.68	6.84 ± 1.84	5.58 ± 1.34
0.4	2.42 ± 0.34	6.79 ± 5.19	3.53 ± 1.01

Results Table 2 shows final validation losses for several models from Table 1. We find a sharp divide between the methods with high sensitivity $\mathbb{S} = O(N)$ and those with low sensitivity $\mathbb{S} = \log(N)$.

The high sensitivity methods are generally unable to come close to solving the task, and notably become worse with depth. On the other hand, the low sensitivity methods are all able to achieve near 0 loss, and display improved performance with depth.

5.2. Gradient Noise and Alternative Optimizers

Next, we analyze the differences between the low sensitivity models. Sensitivity measures a model's tolerance to parameter change, and as discussed in Section 4.1, the magnitude of parameter change used by gradient descent is better aligned with the sensitivity framework for models that satisfy an additional property. Since Catformer is the only low sensitivity model with this additional property, we expect it to be more tolerant to the choice of optimizer. Additionally, we expect it to be more robust to injected gradient noise of the form used in the definition in sensitivity (Definition 1).

Table 3 and Table 4 confirm these hypotheses by comparing Catformer against the standard residual (pre-norm) Transformer; other low-sensitivity models other than the Catformer performed similarly to this Transformer.

Models	TrXL (Dai et al., 2019)	GTrXL (Parisotto et al., 2020)	**Catformer** (ours)
Seek avoid	21.6 ± 16.9	24.5 ± 0.9	$\mathbf{37.7} \pm 1.0$
Rooms watermaze	10.3 ± 8.5	28.2 ± 1.5	$\mathbf{45.9} \pm 3.5$
Rooms select non-matching	0.9 ± 0.8	32.7 ± 1.5	$\mathbf{62.2} \pm 2.8$
Explore rewards many	2.7 ± 0.5	45.3 ± 2.1	$\mathbf{52.3} \pm 4.1$
Explore rewards few	2.1 ± 0.3	29.2 ± 2.6	$\mathbf{37.5} \pm 1.4$
Rat exploration	16.2 ± 20.6	51.2 ± 0.9	$\mathbf{57.9} \pm 0.6$

Figure 2. Mean episode reward TransformerXL (TrXL), GTrXL, and Catformer on the vision-based tasks from DeepMind Lab. These architectures served the function of the RNN actor component of an R2D2-based distributed RL system. We controlled for parameters and ran all methods with equivalent R2D2 hyper-parameters. We found that GTrXL often failed to learn without gradient clipping, so its scores are reported as the max across a sweep of no gradient clipping vs. gradient clipping of varying magnitudes. Catformer's scores are reported with no gradient clipping. Scores reported across 3 seeds per sweep.

Table 4. Models trained with gradient noise injection (val. perplexity). (**Top**): Catformer, (**Bottom**): Transformer

Noise std.	$N = 2$	4	6
0.0005	2.54 ± 0.48	1.13 ± 0.11	1.02 ± 0.02
0.001	3.42 ± 0.38	10.1 ± 9.77	1.51 ± 0.24
0.002	5.61 ± 0.60	13.6 ± 4.24	5.74 ± 1.78
0.0005	5.56 ± 7.00	1.29 ± 0.35	12.2 ± 9.18
0.001	Diverged	Diverged	Diverged
0.002	$179. \pm 155.$	Diverged	Diverged

Table 3 reports models trained with stochastic gradient descent (SGD) instead of Adam. We note that while SGD is the optimization algorithm of choice in many areas of deep learning, where it is superior to alternatives when applicable (Wilson et al., 2018), it is notoriously difficult for Transformer architectures to train with.

Table 4 trains models with Adam, as in Table 2, but injects gradient noise on every optimization step. While Catformer's performance degrades, it is still stable. On the other hand, the Transformer is extremely sensitive to noise, especially as the noise magnitude or depth grows.

5.3. Deep Reinforcement Learning

Setting As a large scale demonstration, we evaluate Catformers on challenging RL 3D visual navigation tasks from the DeepMind Lab benchmark suite (Beattie et al., 2016). In these tasks, created using the Quake III Arena Engine, the environment provides scalar rewards and rich pixel-based observations to the agent. The agent action space spans movement in three dimensions and camera orientation about two aces. The tasks span four distinct groups: object gathering and obstacle avoidance within a static map, navigation to a goal state within a fixed maze given a random starting point per episode, goal navigation within a dynamic environment procedurally-generated at the start of each episode, and laser-tag levels demanding that an agent tag NPC bots reminiscent of standard Quake III Arena gameplay. All of these tasks are inherently partially observable, necessitating advanced memory-based representation capabilities.

Results We leverage R2D2 (Kapturowski et al., 2018), a method for training memory-based RL agents from distributed prioritized experience replay. We compare canonical TransformerXL(TrXL) agents and GRU-based GTrXL agents trained via this regime with Catformer agents and report results. Catformer-based agents outperform GTrXL-based agents with an equivalent number of parameters by more than 13% (Fig. 2). Additionally, we found that GTrXL agents struggled to learn effectively without the use of gradient clipping, but Catformer was robust without this addition. Vanilla TrXL agents struggled to learn, as also observed in (Parisotto et al., 2020).

6. Conclusion

We propose a new metric, sensitivity, to measure architecture instability. We analyze its theoretical properties, provide tools for calculating this measure for a large class of architectures, and show how it ties together existing notions of activation variance, gradient variance, and output change. Sensitivity provides a new lens via which to analyze and design models, yielding a new architecture (Catformer) that outperforms previous state-of-the-art architectures on a suite of high-dimensional RL tasks. We anticipate that the study of this core problem will be broadly useful in understanding model instability in deep learning. We are excited about continuing to improve powerful models such as Transformers for noisy, challenging real-world applications.

References

Arjovsky, M., Shah, A., and Bengio, Y. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pp. 1120–1128. PMLR, 2016.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Chen, M., Pennington, J., and Schoenholz, S. Dynamical isometry and a mean field theory of rnns: Gating enables signal propagation in recurrent neural networks. In *International Conference on Machine Learning*, pp. 873–882. PMLR, 2018a.

Chen, M. X., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G. F., Jones, L., Schuster, M., Shazeer, N., Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Chen, Z., Wu, Y., and Hughes, M. The best of both worlds: Combining recent advances in neural machine translation. In Gurevych, I. and Miyao, Y. (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pp. 76–86. Association for Computational Linguistics, 2018b. doi: 10.18653/v1/P18-1008. URL https://www.aclweb.org/anthology/P18-1008/.

Choromanski, K., Davis, J. Q., Likhosherstov, V., Song, X., Slotine, J. E., Varley, J., Lee, H., Weller, A., and Sindhwani, V. An ode to an ODE. *CoRR*, abs/2006.11421, 2020a. URL https://arxiv.org/abs/2006.11421.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L., and Weller, A. Rethinking attention with Performers. *CoRR*, arXiv:2009.14794, 2020b. URL https://arxiv.org/abs/2009.14794.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.

Dao, T., Gu, A., Eichhorn, M., Rudra, A., and Ré, C. Learning fast algorithms for linear transforms using butterfly factorizations. In *International Conference on Machine Learning*, pp. 1517–1527. PMLR, 2019.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL https://arxiv.org/abs/2010.11929.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Hanin, B. Which neural net architectures give rise to exploding and vanishing gradients? *arXiv preprint arXiv:1801.03744*, 2018.

Hanin, B. and Rolnick, D. How to start training: The effect of initialization and architecture. *arXiv preprint arXiv:1803.01719*, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015a.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015b.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks, 2016.

Hochreiter, S. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91 (1), 1991.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Li, J. Universal transforming geometric network. *CoRR*, abs/1908.00723, 2019. URL http://arxiv.org/abs/1908.00723.

Likhosherstov, V., Davis, J., Choromanski, K., and Weller, A. CWY parametrization for scalable learning of orthogonal and stiefel matrices. *CoRR*, abs/2004.08675, 2020. URL https://arxiv.org/abs/2004.08675.

Liu, L., Liu, X., Gao, J., Chen, W., and Han, J. Understanding the difficulty of training transformers, 2020.

Luo, H., Zhang, S., Lei, M., and Xie, L. Simplified self-attention for transformer-based end-to-end speech recognition. *CoRR*, abs/2005.10463, 2020. URL https://arxiv.org/abs/2005.10463.

Madani, A., McCann, B., Naik, N., Keskar, N. S., Anand, N., Eguchi, R. R., Huang, P., and Socher, R. Progen: Language modeling for protein generation. *CoRR*, abs/2004.03497, 2020. URL https://arxiv.org/abs/2004.03497.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, pp. 7487–7498. PMLR, 2020.

Pennington, J., Schoenholz, S. S., and Ganguli, S. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. *arXiv preprint arXiv:1711.04735*, 2017.

Pennington, J., Schoenholz, S., and Ganguli, S. The emergence of spectral universality in deep networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 1924–1932. PMLR, 2018.

Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., and Ganguli, S. Exponential expressivity in deep neural networks through transient chaos. *arXiv preprint arXiv:1606.05340*, 2016.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Sohl-Dickstein, J. On the expressive power of deep neural networks. In *international conference on machine learning*, pp. 2847–2854. PMLR, 2017.

Rives, A., Goyal, S., Meier, J., Guo, D., Ott, M., Zitnick, C., Ma, J., and Fergus, R. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioArxiv*, 04 2019. doi: 10.1101/622803.

Schoenholz, S. S., Gilmer, J., Ganguli, S., and Sohl-Dickstein, J. Deep information propagation. *arXiv preprint arXiv:1611.01232*, 2016.

Shao, J., Hu, K., Wang, C., Xue, X., and Raj, B. Is normalization indispensable for training deep neural network? *Advances in Neural Information Processing Systems*, 33, 2020.

Srivastava, R. K., Greff, K., and Schmidhuber, J. Highway networks, 2015.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. The marginal value of adaptive gradient methods in machine learning, 2018.

Xiao, L., Bahri, Y., Sohl-Dickstein, J., Schoenholz, S., and Pennington, J. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *International Conference on Machine Learning*, pp. 5393–5402. PMLR, 2018.

Xu, H., Liu, Q., Xiong, D., and van Genabith, J. Transformer with depth-wise lstm, 2020.

Yang, G. and Schoenholz, S. S. Mean field residual networks: On the edge of chaos. *arXiv preprint arXiv:1712.08969*, 2017.

Yang, G., Pennington, J., Rao, V., Sohl-Dickstein, J., and Schoenholz, S. S. A mean field theory of batch normalization. *arXiv preprint arXiv:1902.08129*, 2019.

Zhang, H., Dauphin, Y. N., and Ma, T. Fixup initialization: Residual learning without normalization, 2019.