# The effect of AdamW decoupled weight decay regularization on model structures

Brian Yu

December 10, 2022

## 1 Abstract

State of the art models are optimized using variants of stochastic gradient descent, primarily with the AdamW optimizer. Models consist of four base building blocks: layernorm, linear layers without bias, nonlinearities, and dropout. These building blocks are combined into two complex structures: attention and dense feedforward networks. We investigate the interactions between the AdamW optimization algorithm and these model structures. The code for replicating these experiments can be found at `https://github.com/bri25yu/AdamWAnalysis` and the project website can be found at `https://bri25yu.github.io/AdamWAnalysis/`.

## 2 Introduction

### 2.1 AdamW

The AdamW [KB14; LH17] optimizer is the most popular optimizer today for performing stochastic gradient descent on deep neural networks. It leverages the ideas of momentum to reduce the variance of updates estimated from minibatches of data and takes steps in weight latent space by adjusting gradient values using a first-order estimate of the hessian.

Because AdamW is an adaptive optimizer, regular weight decay coupled with the objective is not fully optimal when adjusted by the hessian. Thus, AdamW decouples regularization from the objective and instead imposes weight regularization by directly adding a component in the gradient update step. The authors choose a decoupled regularization update equivalent to coupled 2-norm based weight decay (L2).

The AdamW algorithm is summarized below in 1.

**Algorithm 1** AdamW algorithm

```
Require: $\gamma$ (lr), $\beta_1, \beta_2$ (betas), $\theta_{t-1}$ (params), $f(\theta)$ (objective), $\epsilon$ (epsilon), $\lambda$ (weight decay)
 1: $m_0 \leftarrow 0$ (first moment)
 2: $v_0 \leftarrow 0$ (second moment)
 3: for $t = 1$ to ... do
 4:     $\theta_t \leftarrow \theta_{t-1} - \gamma\lambda\ \theta_{t-1}$                    ▷ weight decay
 5:     $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$                ▷ EMA-biased 1st moment estimate
 6:     $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$               ▷ EMA-biased 2nd moment estimate
 7:     $\widehat{m}_t \leftarrow m_t/(1-\beta_1^t)$                     ▷ bias-corrected 1st moment estimate
 8:     $\widehat{v}_t \leftarrow v_t/(1-\beta_2^t)$                      ▷ bias-corrected 2nd moment estimate
 9:     $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$          ▷ update
10: end for
```

#### 2.1.1 Weight decay

Line 4 imposes an explicit weight decay on the parameters $\theta_t$ proportional to $\theta_{t-1}, \gamma, \lambda$. Intuitively, we certainly need an update proportional to our learning rate $\gamma$ and we should update somewhat on the order of the parameter magnitude. The weight decay $\lambda$ parameter can be interpreted as a percentage of the weight to decay.

In practice, weights in $\theta$ are initialized uniformly randomly according to $\mathcal{U} \sim \text{Uniform}(-a, a), a \propto \frac{1}{\sqrt{d}}$. Thus, the weights in $\theta$ range from 1e-2 to 1e-5. Typical values for $\gamma$ lie between 1e-2 and 1e-5 and $\lambda$ is typically 1e-2 or 1% weight decay. In total, the weight decay imposed per step is about 1% of the parameter value multiplied by the learning rate.

### 2.1.2 Exponential moving average

Lines 5 and 6 calculate an exponential moving average (EMA) of the first and second moments, respectively. This EMA has several purposes. First, it stabilizes training by averaging the moments over steps. This is incredibly important when dealing with overparameterized networks and with noisy data. Second, the EMA surprisingly reduces the bias because averages over recent steps yields better estimations of the true gradient. Productive noise is compounded over time and unproductive noise is averaged out over time.

In practice, the parameters relating to the EMAs are $\beta_1 = 0.9, \beta_2 = 0.999$ for the first and second moment, respectively. We use very high values of $\beta$ in order to retain as much information from past steps as possible, without suffering from the bias reduction from how stale the old moments are. We use a significantly higher value for the second moment estimation because of how unstable it is.

### 2.1.3 Moment estimation

The first moment is clearly the gradient, an estimation of an improvement update to the parameters. The second moment is used to take adaptive steps in the weight latent space. The second moment can be interpreted as a gradient regularizer to perform natural gradient descent.

### 2.1.4 Convergence

There exist different classes of convergence for stochastic optimization algorithms. Obviously convergence depends on a lot of factors, but here's a good starting point: the best stochastic optimizers will converge to optimums even given non-convergent sequences of gradients.

The prototypical scenario is when we have multiple steps of small gradient then one step of large gradient. The prototypical example is imbalanced classification, where minibatches of samples consist of the majority class and rarely contain the minority class which incur a large loss. The idea is that the optimizer is never able to really settle into either stationary point, either the local optimum of just the majority class or the global optimum of both classes.

Let $\mathcal{H}$ be our optimization horizon, in this case $\mathcal{H} = 10000$.

1. Does the loss converge? $\limsup_{t \to \mathcal{H}} \mathcal{L}_t ? = \liminf_{t \to \mathcal{H}} \mathcal{L}_t$
2. Does there exist a stationary distribution over the parameters?
   - There exists no notion of absolute deterministic convergence with our stochastic optimization methods especially with regularization techniques such as weight decay.
   - Let $\mathcal{T}$ be the update operator. $\exists \pi_\theta^*$ s.t. $\pi_\theta^* = \pi_\theta^* \mathcal{T}$
3. Do the parameters converge to that stationary distribution? $\lim_{t \to \mathcal{H}} \pi_\theta^t = \pi_\theta^*$
4. How quickly do the parameters converge to the stationary distribution?
5. Finally we can start considering optimality

## 2.2 Model structures

State of the art (SOTA) models today primarily consist of four building blocks: layernorm, linear layers without bias, nonlinearities, and dropout. We concern ourselves with the first three and leave dropout regularization as an unexplored topic.

### 2.2.1 LayerNorm

The layernorm [BKH16] normalizes by the variance and multiplies each feature by some weight. The weights are initialized to 1.

**Algorithm 2** Layernorm

```
Require: weights $\in \mathbb{R}^d$, inputs $X \in \mathbb{R}^d$
1: $X \leftarrow X/\text{var}(X)$
2: $X \leftarrow X*$ weights
3: return $X$
```

The layernorm is advantageous in many positions. First, a feature vector is always unit standard deviation, allowing downstream weight matrix multiplication operations to be comparable within a single data point. Second, the weights are an explicit mechanism for scaling, something that is hard to perform over an entire weight matrix. This allows for better conditioning in a variety of scenarios, including softmax-based attention or crossentropy loss.

### 2.2.2 Linear

A linear layer without bias simply performs a matrix multiplication. The weights are initialized uniformly randomly according to $\mathcal{U} \sim \text{Uniform}(-a, a), a \propto \frac{1}{\sqrt{d}}$

**Algorithm 3** Linear layer without bias

```
Require: weights $W \in \mathbb{R}^{d_2 \times d_1}$, inputs $X \in \mathbb{R}^{d_1}$
1: return $WX$
```

Linear layers are typically used with a bias, but it turns out that a bias is not productive and is not necessary.

### 2.2.3 Nonlinearities

There are many nonlinearities proposed, ranging from the rectified linear unit (ReLU), sigmoid linear unit (SiLu or Swish) [RZL17], Gaussian error linear unit (GELU) [HG16], and many more.

$$\begin{aligned}\text{ReLU}(x) &= \max(x, 0)\\ \text{Swish}(x) &= x * \sigma(x), \sigma = \text{sigmoid}\\ \text{GELU}(x) &= x * \Phi(x), \Phi = \text{Gaussian CDF}\end{aligned}$$

The choice of nonlinearity typically isn't too important as long as it's in this linear unit class that has identity gradient if the value is nonnegative. The least expressive of the three nonlinearities mentioned is the basic ReLU since it doesn't propogate gradients for values less than 0, but typical structures sandwich nonlinearities between weight layers which still allows negative output, so the lack of expressivity in ReLU is remediated.

### 2.2.4 More complex structures

SOTA models typically combine these four basic building blocks into two structures: attention and dense feedforward networks [Raf+19; Bro+20]. These structures are composed of four different building blocks A visualization of these structures is found in 2.2.4.

## 3 Experiments

We perform two sets of experiments. First, we develop a model that can successfully learn in this environment. Then, we test convergence characteristics for our optimization algorithm.

We use a Voronoi partitioning of 2D space. We choose 2D space because it's easiest to visualize. We choose Voronoi partitions because it's very hard for a deep neural network to learn so the results will be applicable to other environments. See 3 for an example.

The parameters for the following experiments are listed in 1

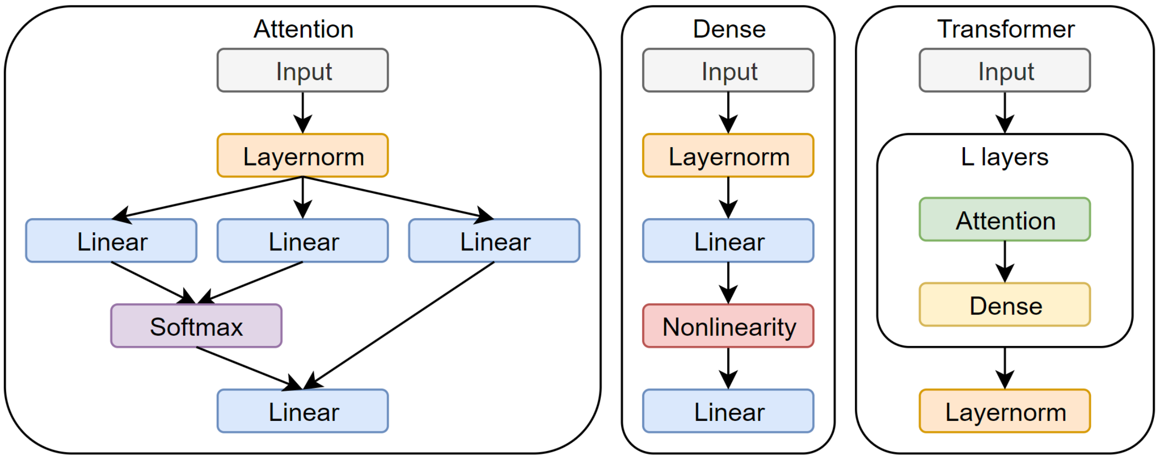


Figure 1: Left: A typical attention module. The linear layers in the middle are the query, key, and value matrices, respectively and the linear layer at the bottom is the output layer. Middle: A typical dense feedforward network. Note that the nonlinearity is sandwiches by two linear layers. Right: A typical transformer that uses both the attention and dense layers

| **Experiment parameters** | | | |
|---|---|---|---|
| **Parameter** | **Shared Value** | **Modeling value** | **Optimization value** |
| Weight decay | 1e-2 | | |
| Training steps | 10k | | |
| Eval set size | 10k | | |
| Test set size | 10k | | |
| Learning rate | 1e-2 | | |
| Batch size | | 32 | 1024 |
| Datapoints seen | | 320k | 1mil |

Table 1: The values for weight decay, training steps, and eval and test set sizes are very typical for today's SOTA experiment setups. The modeling experiments learning rate value of 1e-2 is very large compared to today's models which typically have a maximum learning rate of around 1e-4 to 1e-3. The modeling experiments batch size value of 32 is very small compared to today's models which typically have a batch size of 256 to 1024, making this challenge tougher. The formula for number of training datapoints seen is the number of steps times the batch size per step.

## 3.1 Modeling

We begin by creating a model that is able to learn in this environment and uses tools from state-of-the-art models today, namely ReLU, softmax, and unbiased linear layers.

There are certain structures present in SOTA models that are not present in the following models, namely layernorms. Layernorming the input would literally change the actual input since the particular magnitude of an input actually matters for this environment, so it introduces an unnegotiable bias. Layernorming in transformers does not increase the bias significantly as the magnitude of a vector is not critical.

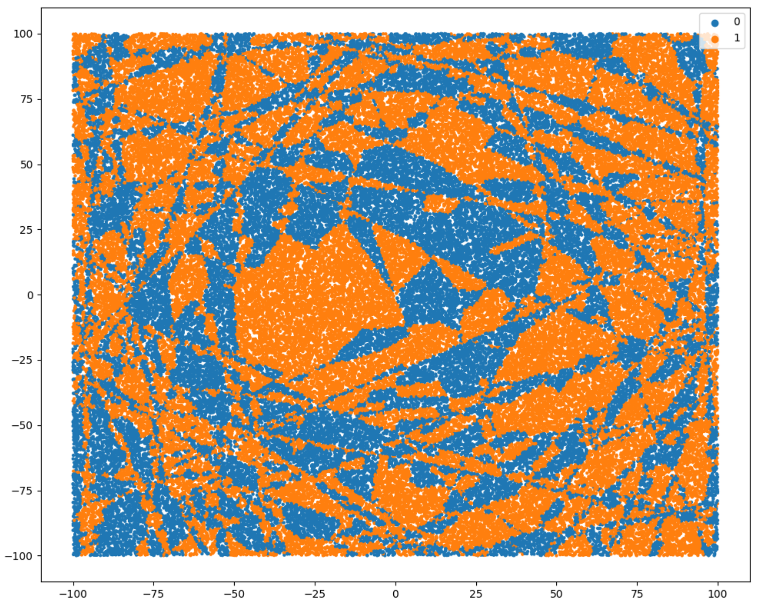


Figure 2: An example of a Voronoi space that we have a neural net learn.

All of the following equations use the following shared definitions.

$$
\begin{aligned}
N &= \text{batch size} \\
D &= \text{dimension of environment} \\
X &= \text{inputs to classify of shape (N, D)} \\
\mathcal{C} &= \text{Voronoi centers} \\
C &= \text{classes} \\
\theta &= \text{model parameters} \\
\mathcal{L}(\hat{y}, y) &= \text{loss function. Crossentropy in this case} \\
f_\theta &= \text{classifier} \\
J_\theta &= \text{objective function} = \mathcal{L}(f_\theta(X), y)
\end{aligned}
$$

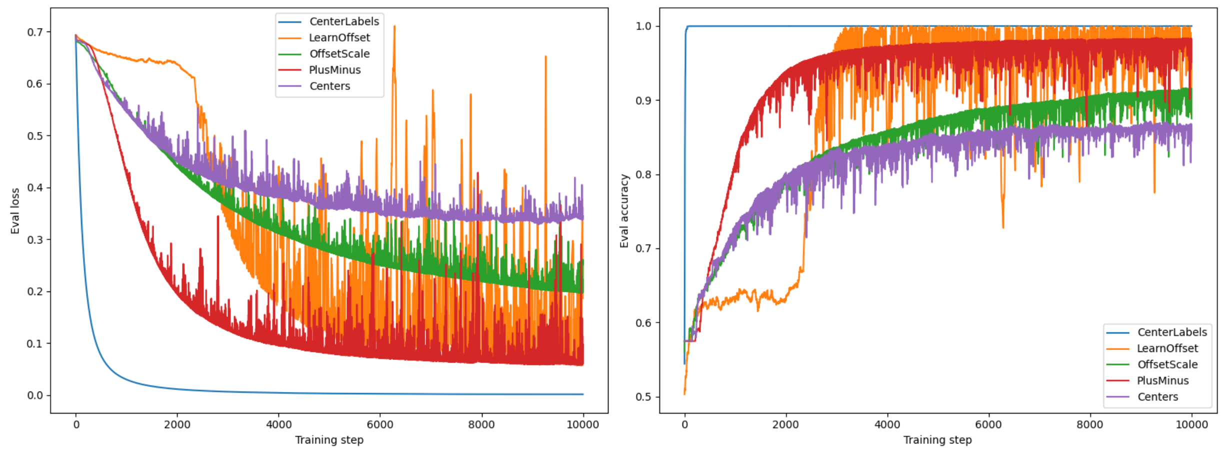


Figure 3: Results for progression of models for successfully parameterizing a model for our chosen environment.

We begin with the exact formula and slowly parameterize our model one piece at a time.

$$f_\theta = \arg\min_C \left| \frac{X^T \mathcal{C}}{||\mathcal{C}||_2^2} - 1 \right|$$

First, we remove information that the model has about which centers correspond to which classes (3.1, "CenterLabels", blue line). Unsurprisingly, given that the model is given so much information about our environment, the loss decreases very quickly and 100% accuracy is reached.

$$\begin{aligned} C_\theta &= \text{Linear layer } (|\mathcal{C}||C| \text{ parameters}) \\ f_\theta &= \text{softmax}\left(-\left|\frac{X^T \mathcal{C}}{||\mathcal{C}||_2^2} - 1\right|\right) C_\theta \end{aligned}$$

Then, we remove information about the offset or "target" start pushing the model to learn the offset part of the exact equation (3.1, "LearnOffset", orange line). The period of stale learning in the beginning before the large drop around 2500 training steps is due to learning the centers. The massive drop is caused by learning the correct offset. Notice that the eval loss is very noisy, bouncing between 0.1 and 0.7 at some points. This is a sign of poor conditioning, most likely from matching the model to the underlying structure of the domain.

$$\begin{aligned} \text{offset}_\theta &= \text{Offset (1 parameter)} \\ f_\theta &= \text{softmax}\left(-\left|\frac{X^T \mathcal{C}}{||\mathcal{C}||_2^2} - \text{offset}_\theta\right|\right) C_\theta \end{aligned}$$

Then, we provide better conditioning for the model softmax by introducing a scale parameter right before the softmax operation (3.1, "OffsetScale", green line). Notice that this significantly reduces the noise compared to without a scale conditioning parameter, but it also significantly increases the bias.

$$\begin{aligned} \text{scale}_\theta &= \text{Softmax conditioning scale (1 parameter)} \\ f_\theta &= \text{softmax}\left(-\text{scale}_\theta\left|\frac{X^T \mathcal{C}}{||\mathcal{C}||_2^2} - \text{offset}_\theta\right|\right) C_\theta \end{aligned}$$

Then, we drop our absolute value operation in favor of a ReLU nonlinearity operation in order to match SOTA model structures (3.1, "PlusMinus", red line). We're able to reduce the bias of our model by introducing more parameters and by letting the neural net design its own features (rather than forcing it into using an absolute value).

$$\begin{aligned} \text{pm}_\theta &= \text{Plus-minus parameter for learning absolute value (2 parameters)} \\ f_\theta &= \text{softmax}\left(-\text{scale}_\theta * \text{pm}_\theta * \text{ReLU}\left(\frac{X^T \mathcal{C}}{||\mathcal{C}||_2^2} - \text{offset}_\theta\right)\right) C_\theta \end{aligned}$$

Finally, we force the model to learn the proper centers (3.1, "Centers", purple line).

$$\begin{aligned} \mathcal{C}_\theta &= \text{Learned Voronoi centers } (HD \text{ parameters}) \\ f_\theta &= \text{softmax}\left(-\text{scale}_\theta * \text{pm}_\theta * \text{ReLU}\left(\frac{X^T \mathcal{C}_\theta}{||\mathcal{C}_\theta||_2^2} - \text{offset}_\theta\right)\right) C_\theta \end{aligned}$$

Thus, we are able to create a model that learns successfully in this environment.

## 3.2 Finalizing the model

The final model that will be used for downstream experiments is represented in the algorithm 4.

**Algorithm 4** Final model used in this domain

```
Require: parameters $\theta$, inputs $X$
1: $X \leftarrow \text{Dense}(X)$
2: $X \leftarrow \text{ReducedAttention}(X)$
3: $X \leftarrow \text{Dense}(X)$
4: $X \leftarrow \text{LayerNorm}(X)$
5: return $X$
```

## 3.3 Ablation experiments

We ablate over several factors in our modeling process and our model architecture, namely:

- the nonlinearities ReLU, Swish, and GELU
- linear weight initialization schemes uniformly random vs Gaussian
- layernorm with normalization or not
- weight decay strengths of {0, 1e-3, 1e-2, 1e-1, 1}
- EMA beta values of $\beta_1 \in \{0.85, 0.9, 0.95\}$ and $\beta_2 \in \{0.98, 0.99, 0.995\}$
- types of AdamW decoupled weight decay regularization: L2 (default), L1, and min
- hidden dim sizes of $H \in \{256, 512, 1024, 2048\}$

All experiments are hyperparameter tuned over learning rate values in {2e-3, 3e-3, 5e-3}.

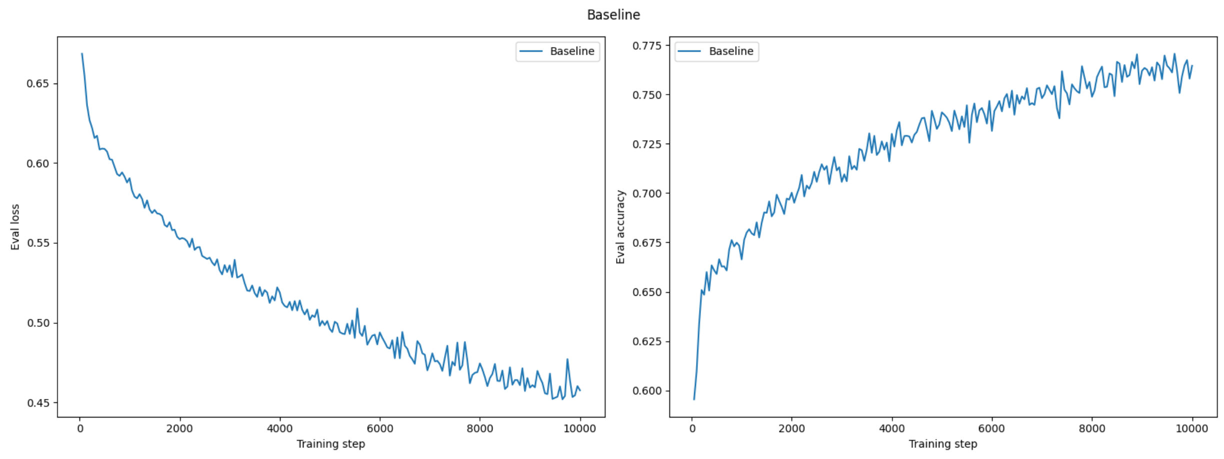
The below experiments analyze the different ablation conditions using two primary visualization methods: graphs of the evaluation loss and accuracy and graphs of the average, min, and max weights values for different components of the model (found in the appendix)

### 3.3.1 Baseline

The baseline experiment achieves a very smooth eval loss curve (3.3.1) and a final test accuracy of 77%.

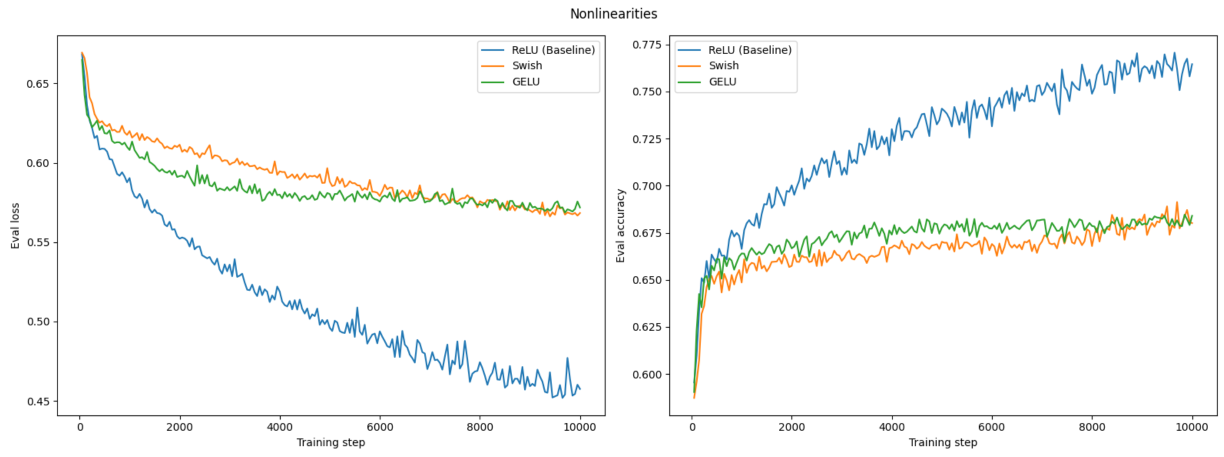
Notably, as seen in 5.1, not all weights in the model converge, specifically the input layernorm weights and the final layernorm. The input layernorm weights are approaching convergence, which signals that the model wasn't given enough time or data to converge. The final layernorm weights that precede the crossentropy loss calculation are intuitively increasing. As time passes and the model learns, it gets more and more confident of its predictions, causing the scale to increase. Critically, the scale is able to increase steadily (almost linearly) despite the weight-proportional weight decay imposed by AdamW.

Interestingly, the weights of the final linear layer preceding the softmax layernorm continue decreasing. Perhaps this means that some of the pre-softmax layernorm weights should be negative to be able to also account for this.



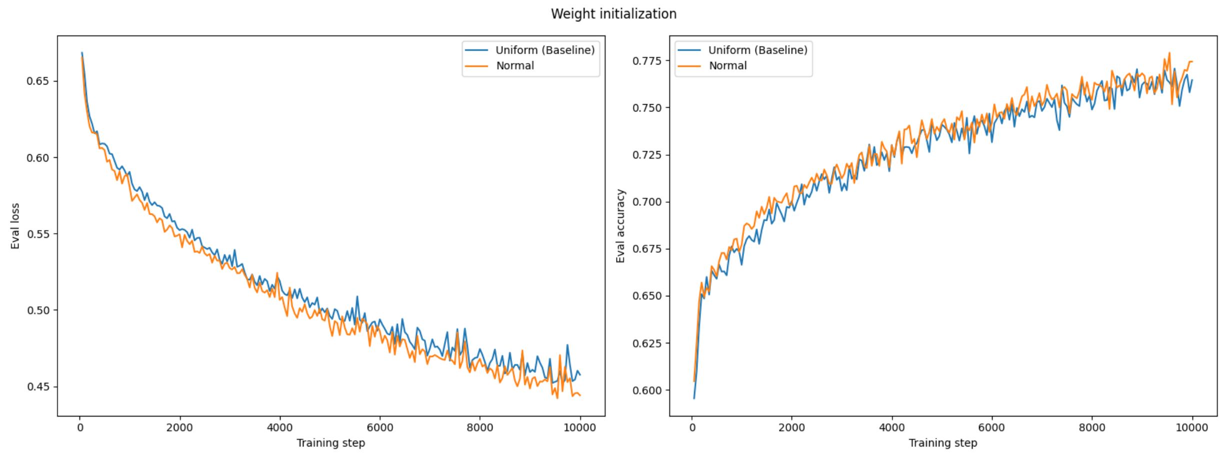
### 3.3.2 Nonlinearities

The swish and GELU nonlinearities perform significantly worse than the baseline ReLU nonlinearity. It's unclear what mechanism causes this. A reflection of this can be seen in the weights 5.1, where the swish and GELU models aren't able to correctly leverage the softmax attention. Interestingly enough, this may be because the swish and GELU models aren't expressive enough using the first layernorm or the negative value backpropogation confounds the weights. Additional experimentation is needed to discern between the two.



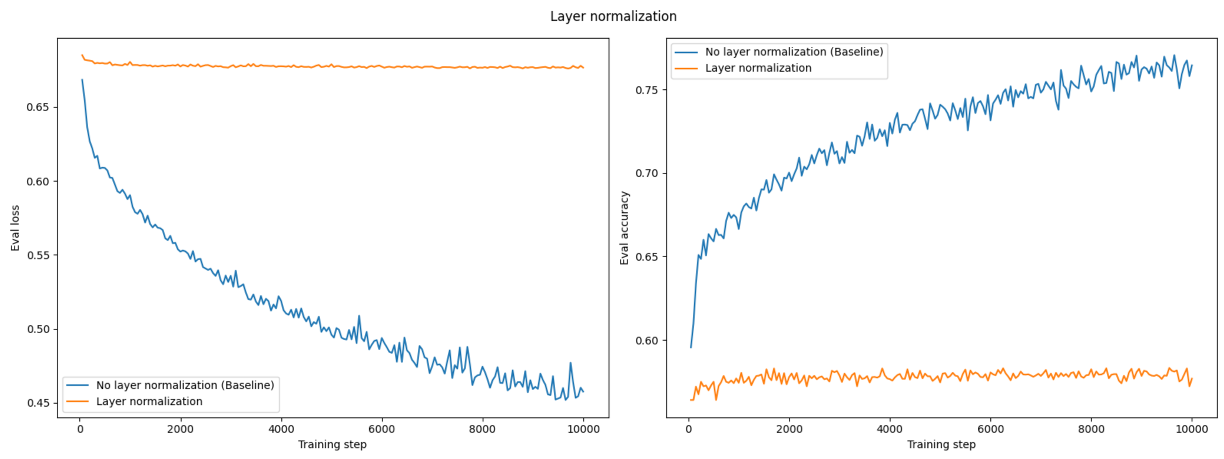
### 3.3.3 Linear weight initialization

There's very little difference in the two different types of weight initialization ablated over. Even their weight values 5.1 trend in the same way. Interestingly, even though the two types of weight initializations behave similarly, there exists a sizeable performance gap in the test accuracy, with 75.6% for uniform initialization and 76.8% for Gaussian initialization.



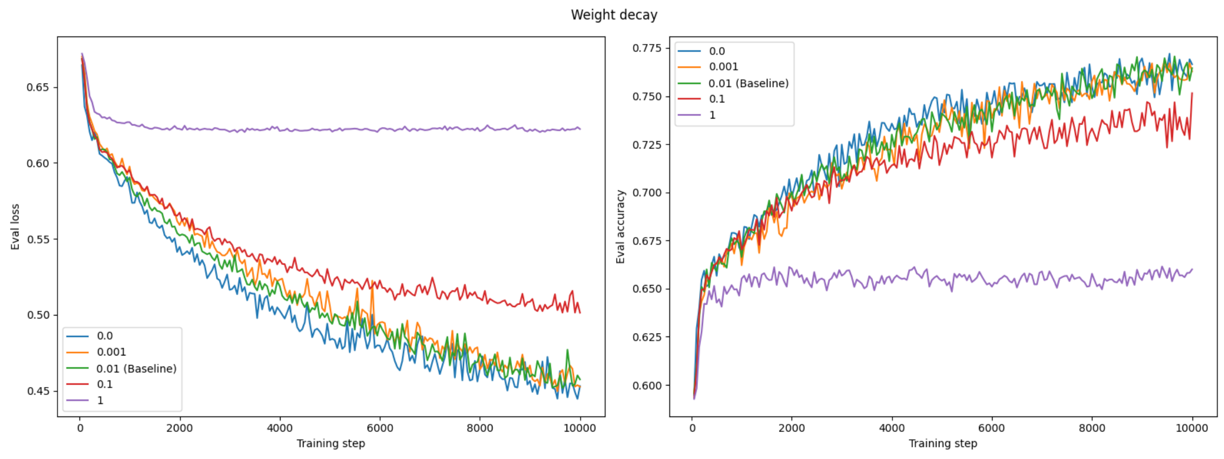
### 3.3.4 LayerNorm and normalization

The model with proper variance-based layer normalization performs very poorly in this environment because the true magnitude of an input point matters. Dividing by the variance for a unit standard deviation introduces significant bias in this environment. This is not a problem with transformers that use embeddings because they map tokens to normalized embedding vectors.



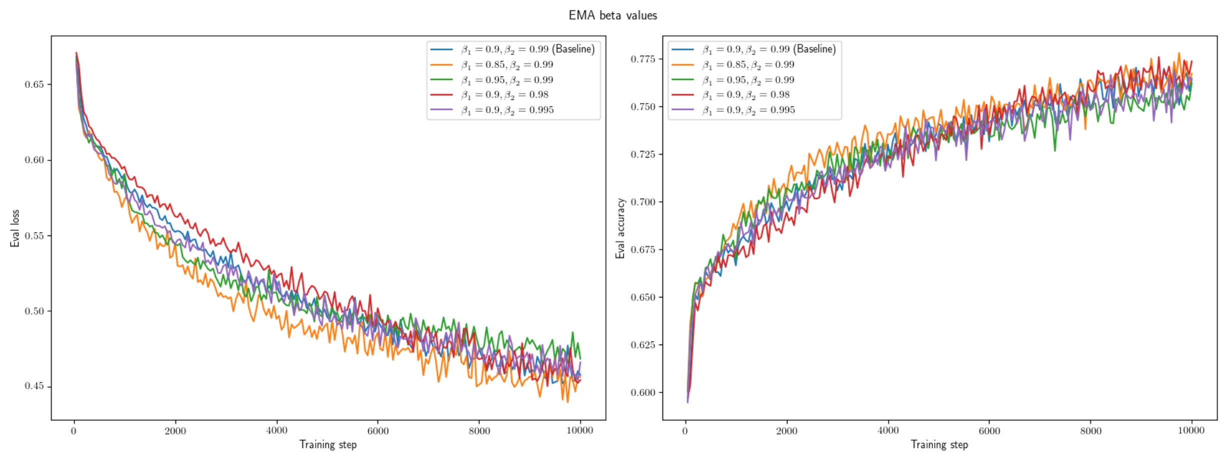
### 3.3.5 Weight decay strength

Weight decay values that are too large introduce too much bias in order for the model to converge to the optimum, although they do encourage the model to converge. The lower the weight decay multiplier value the better the eval and test scores. Additional ablation experiments need to be performed with different domains to determine whether weight decay of this nature is actually detrimental to model learning. There's not too much difference between the lower weight decay values, but that may change as the model size increases.



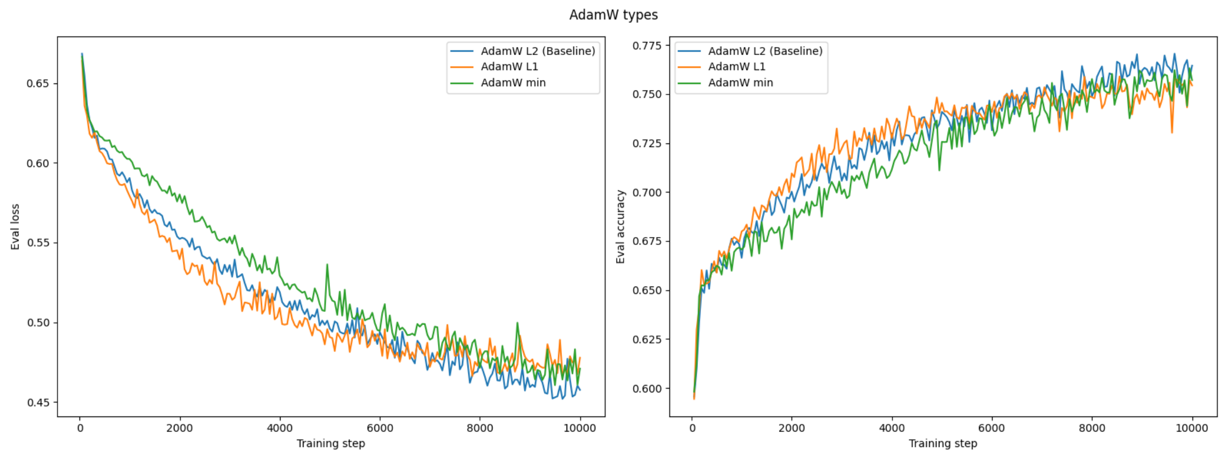
### 3.3.6 EMA beta values

There's very little difference in the different ablations over EMA parameters. We hypothesize that these values matter more in lower data scenarios.



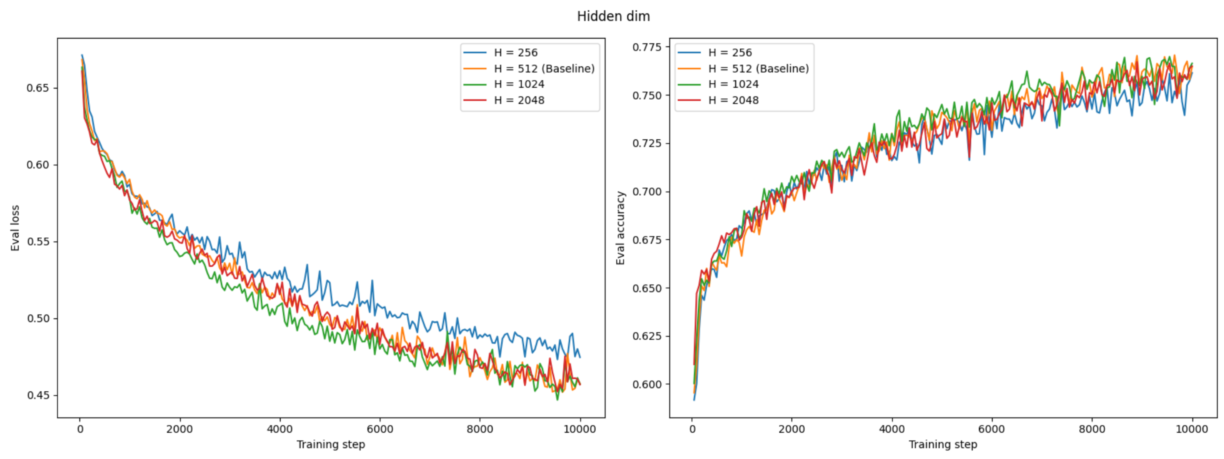
### 3.3.7 Types of AdamW Decoupled weight decay regularization

The type of decoupled weight decay used by AdamW doesn't make a significant different on the performance of the model, but it is notable that a parameter-proportional method like the baseline L2 does perform the best and has the lowest bias, despite the other two types of regularization theoretically decaying the weights less. As such, several of the weights 5.1 that are expected to converge for the L1 and min decay types do not converge (a few linear weights and all weights except for the layernorm weights for min).



### 3.3.8 Hidden dimension size

Overall, the hidden dimension size doesn't affect the model's performance significantly. Higher values in this domain do perform better.



## 4 Conclusion and next steps

There are several interesting next steps to explore:

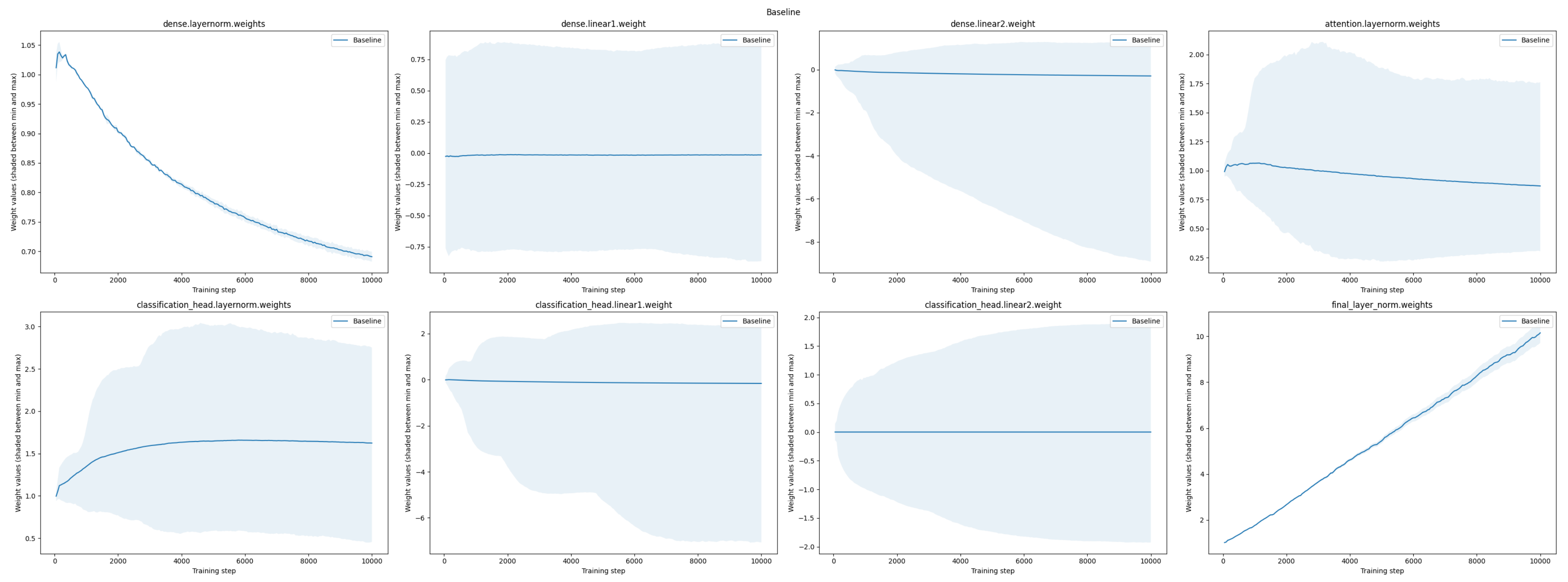
- Should there be positive and negative pre-softmax layernorm weights to increase the expressivity?
- Are the swish and GELU nonlinearities being outperformed by ReLU because they are not expressive enough or because the negative values confound the backpropogation weight updates?
- Do lower amounts of data require higher EMA beta values?
- Do larger model hidden dimension sizes require higher weight decay values because they're heavily overparameterized?

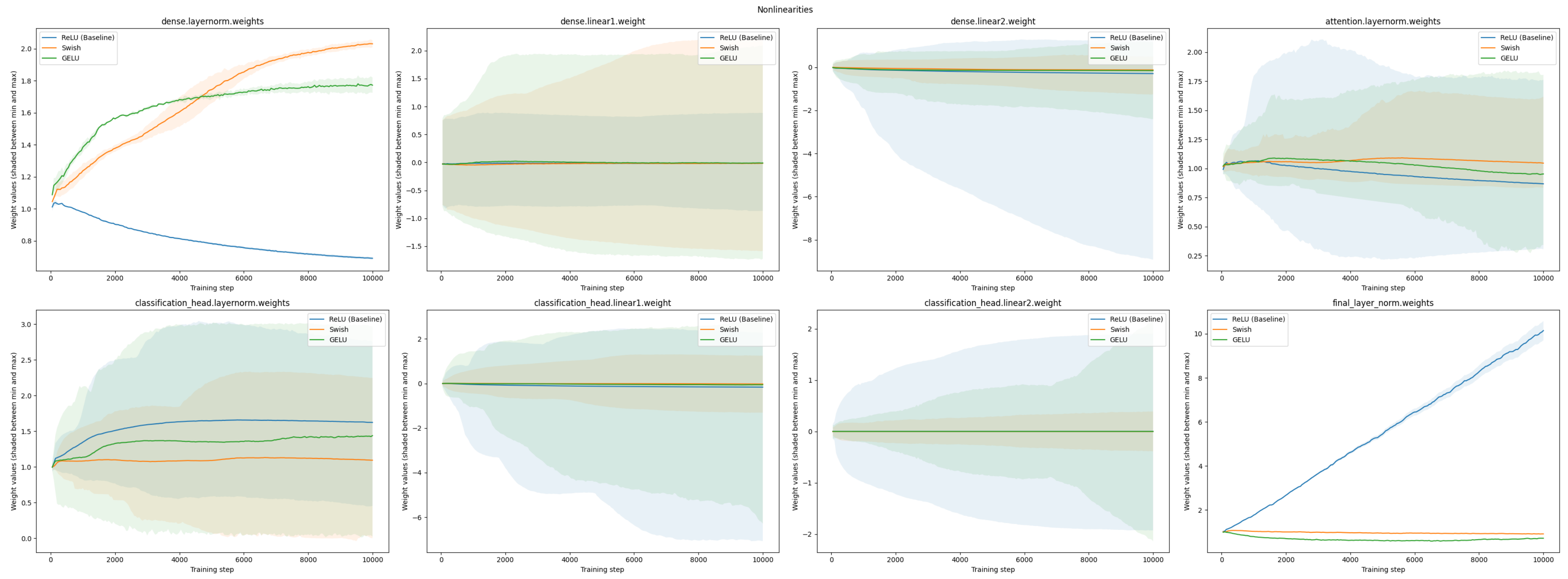
Overall, the AdamW stochastic optimization algorithm is very robust to a lot of the different parameters it involves.

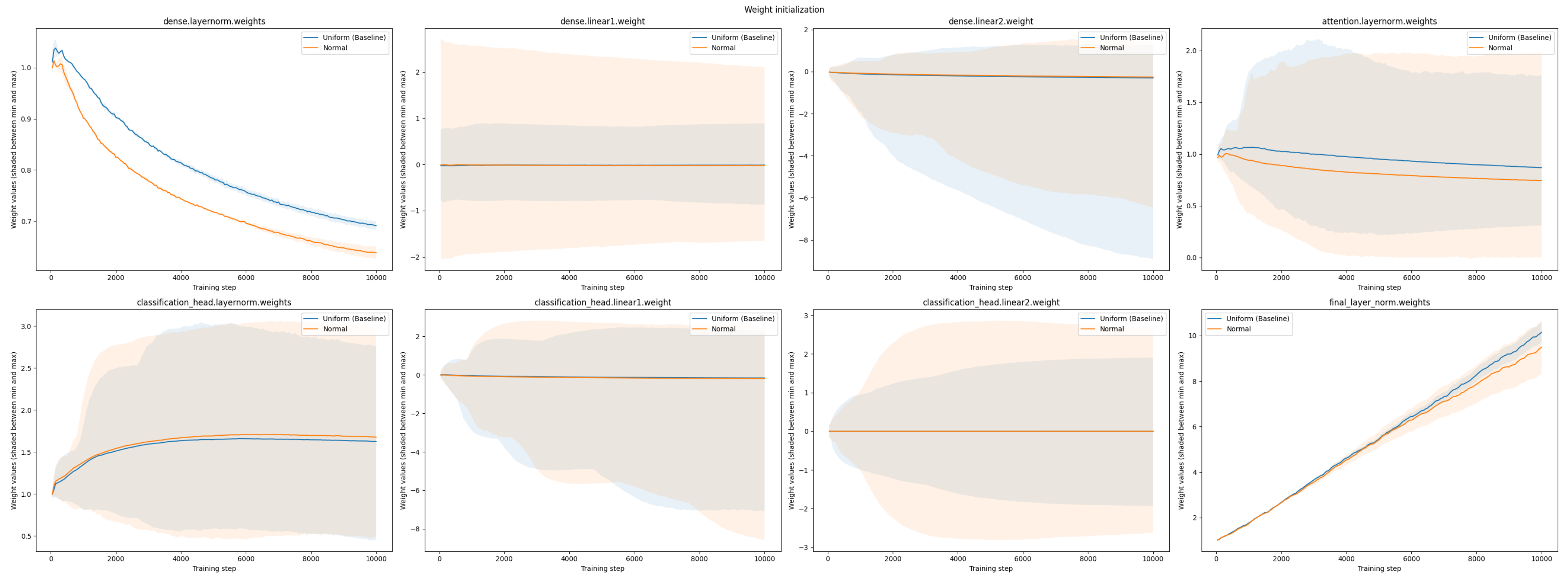
## 5 Appendix

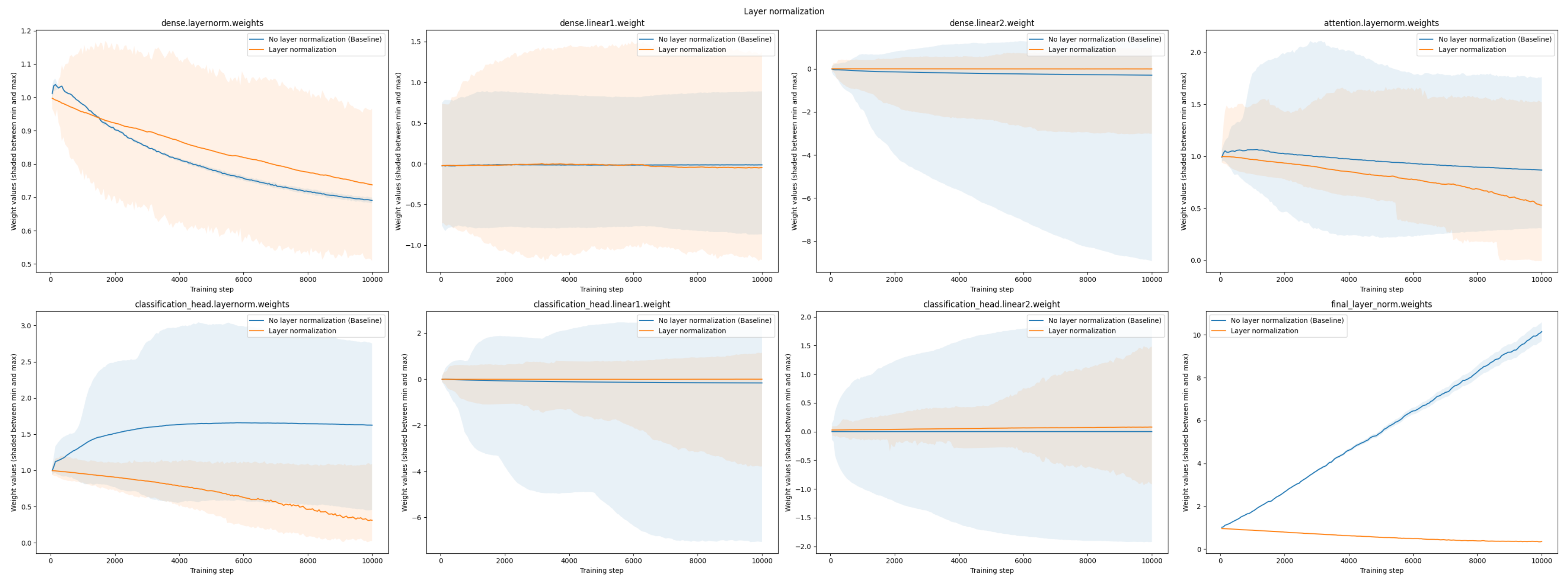
### 5.1 Weight value visualizations

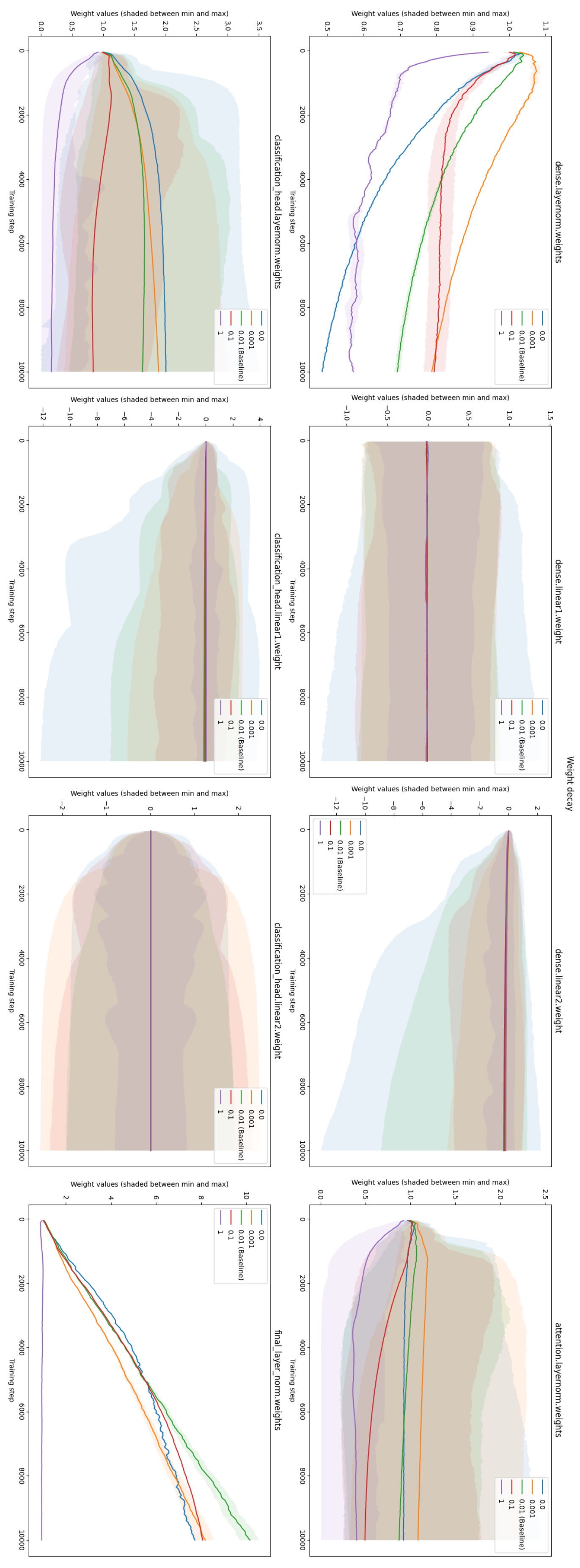
The following weight value visualizations were not included above because they are too large for the text.

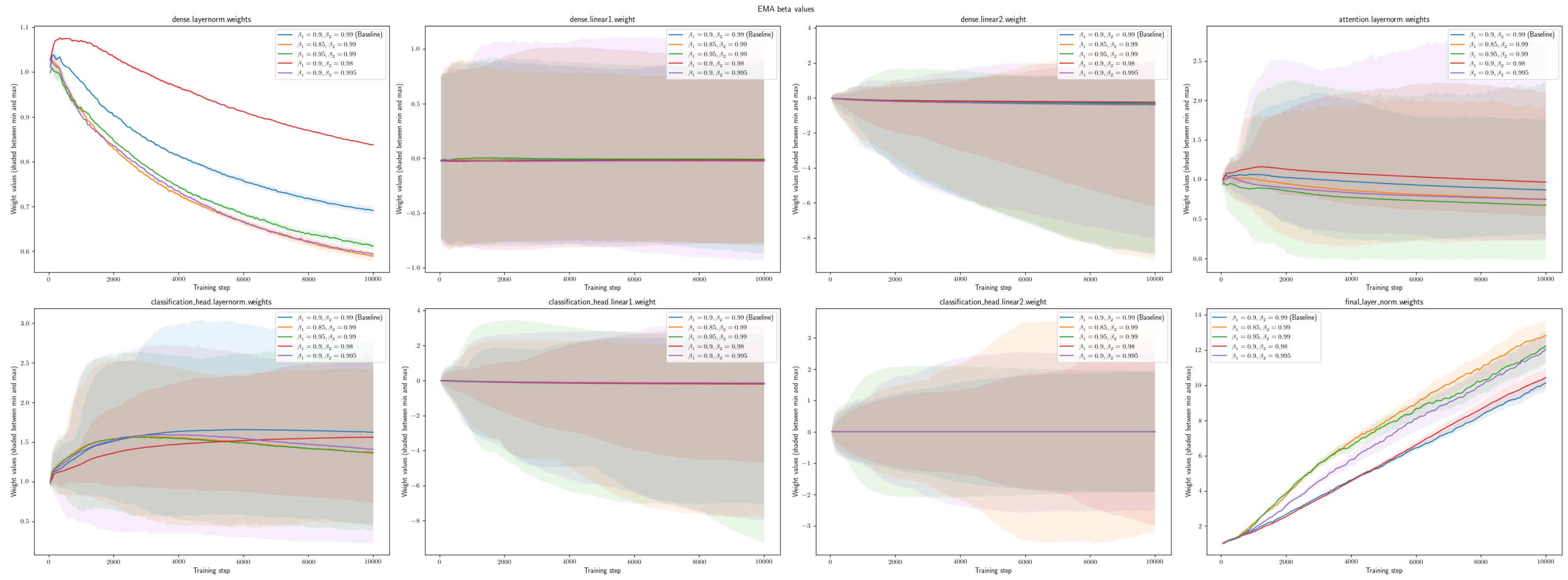

Baseline
dense.layernorm.weights
dense.linear1.weight
dense.linear2.weight
attention.layernorm.weights
classification_head.layernorm.weights
classification_head.linear1.weight
classification_head.linear2.weight
final_layer_norm.weights
Weight values (shaded between min and max)
Training step
Baseline

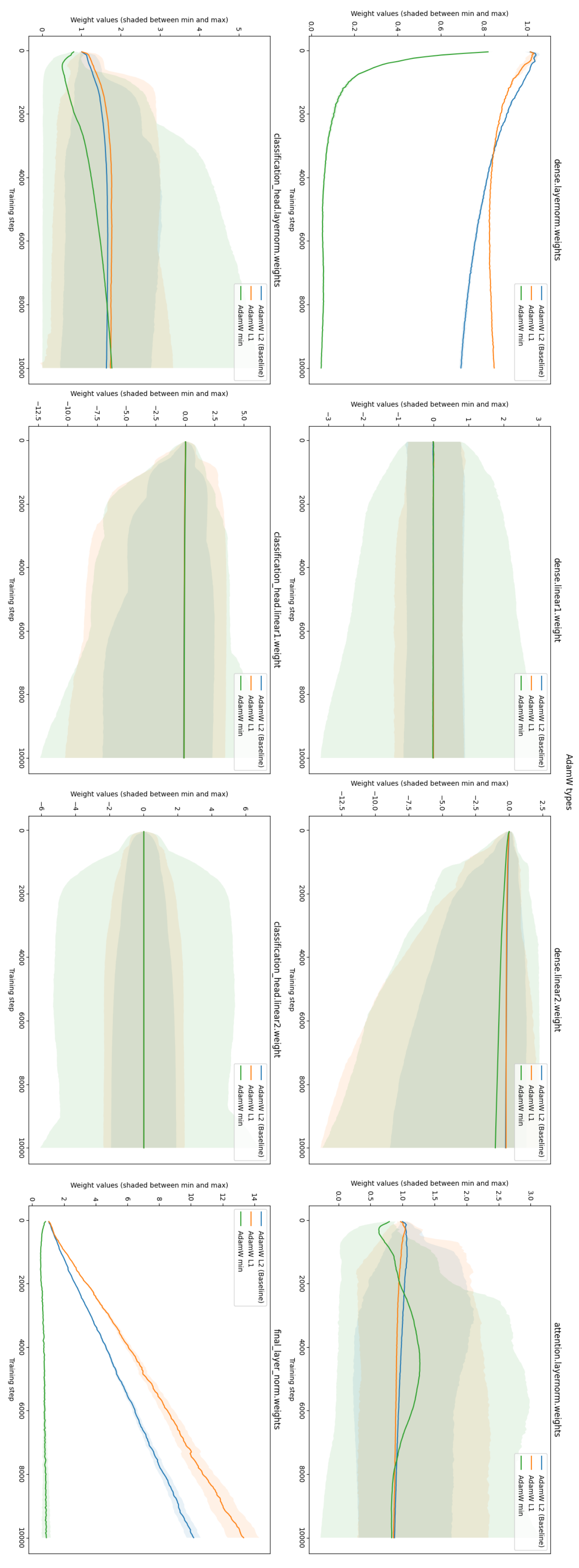

Nonlinearities
dense.layernorm.weights
dense.linear1.weight
dense.linear2.weight
attention.layernorm.weights
classification_head.layernorm.weights
classification_head.linear1.weight
classification_head.linear2.weight
final_layer_norm.weights
ReLU (Baseline)
Swish
GELU
Weight values (shaded between min and max)
Training step

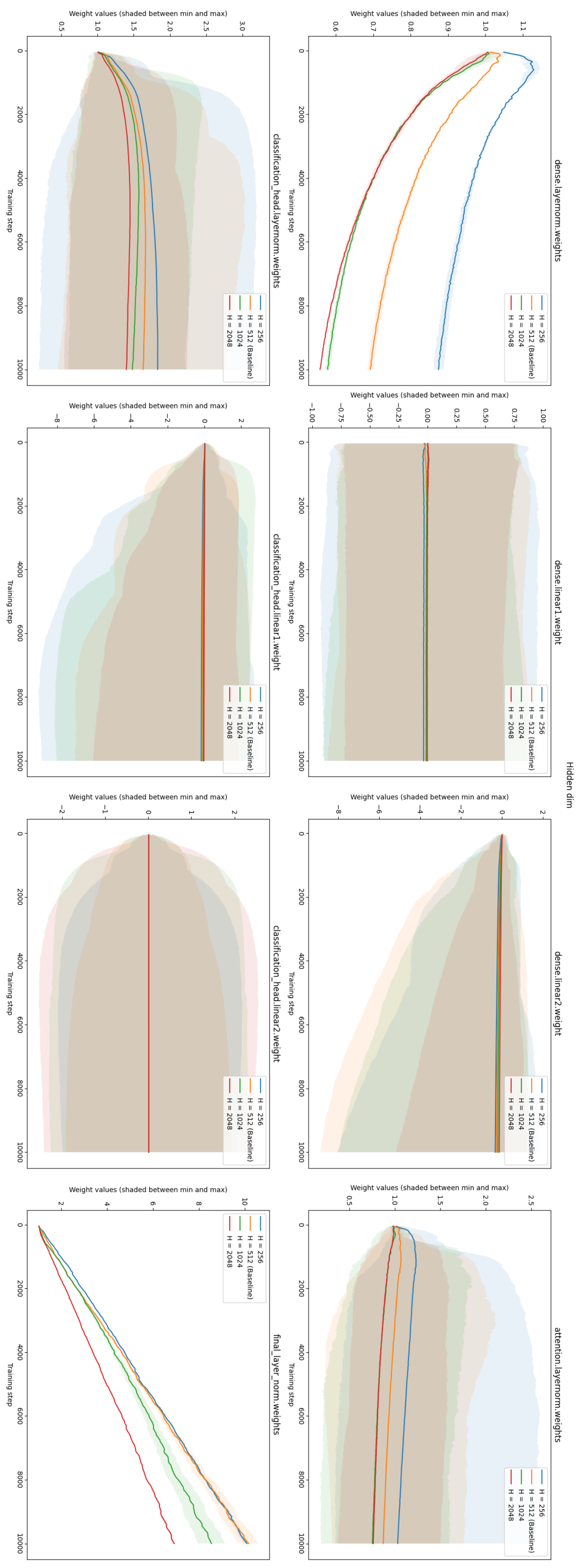

Weight initialization
dense.layernorm.weights
dense.linear1.weight
dense.linear2.weight
attention.layernorm.weights
classification_head.layernorm.weights
classification_head.linear1.weight
classification_head.linear2.weight
final_layer_norm.weights
Uniform (Baseline)
Normal
Weight values (shaded between min and max)
Training step


Layer normalization
dense.layernorm.weights
dense.linear1.weight
dense.linear2.weight
attention.layernorm.weights
classification_head.layernorm.weights
classification_head.linear1.weight
classification_head.linear2.weight
final_layer_norm.weights
No layer normalization (Baseline)
Layer normalization
Weight values (shaded between min and max)
Training step


Weight decay
dense.layernorm.weights
dense.linear1.weight
dense.linear2.weight
attention.layernorm.weights
classification_head.layernorm.weights
classification_head.linear1.weight
classification_head.linear2.weight
final_layer_norm.weights
Weight values (shaded between min and max)
Training step
0.0
0.001
0.01 (Baseline)
0.1
1


EMA beta values
dense.layernorm.weights
dense.linear1.weight
dense.linear2.weight
attention.layernorm.weights
classification_head.layernorm.weights
classification_head.linear1.weight
classification_head.linear2.weight
final_layer_norm.weights
Weight values (shaded between min and max)
Training step
$\beta_1 = 0.9, \beta_2 = 0.99$ (Baseline)
$\beta_1 = 0.85, \beta_2 = 0.99$
$\beta_1 = 0.95, \beta_2 = 0.99$
$\beta_1 = 0.9, \beta_2 = 0.98$
$\beta_1 = 0.9, \beta_2 = 0.995$


AdamW types
dense.layernorm.weights
dense.linear1.weight
dense.linear2.weight
attention.layernorm.weights
classification_head.layernorm.weights
classification_head.linear1.weight
classification_head.linear2.weight
final_layer_norm.weights
Weight values (shaded between min and max)
Training step
AdamW L2 (Baseline)
AdamW L1
AdamW min


Hidden dim
dense.layernorm.weights
dense.linear1.weight
dense.linear2.weight
attention.layernorm.weights
classification_head.layernorm.weights
classification_head.linear1.weight
classification_head.linear2.weight
final_layer_norm.weights
Weight values (shaded between min and max)
Training step
H = 256
H = 512 (Baseline)
H = 1024
H = 2048

## References

[KB14] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization.* 2016. DOI: 10.48550/ARXIV.1607.06450. URL: https://arxiv.org/abs/1607.06450.

[HG16] Dan Hendrycks and Kevin Gimpel. “Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units”. In: *CoRR* abs/1606.08415 (2016). arXiv: 1606.08415. URL: http://arxiv.org/abs/1606.08415.

[LH17] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization.* 2017. DOI: 10.48550/ARXIV.1711.05101. URL: https://arxiv.org/abs/1711.05101.

[RZL17] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. “Searching for Activation Functions”. In: *CoRR* abs/1710.05941 (2017). arXiv: 1710.05941. URL: http://arxiv.org/abs/1710.05941.

[Raf+19] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *CoRR* abs/1910.10683 (2019). arXiv: 1910.10683. URL: http://arxiv.org/abs/1910.10683.

[Bro+20] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.