Like A Light Switch: Conditional Weights for Meta-Learning

Carter Blum and Ashley Law

December 2020

1 Introduction

[1] Deep learning using stochastic gradient descent is a very common method for solving both regression and classification tasks. In both settings, a task τ is defined by a set of input data points $\{x_i\}$ and corresponding regression targets $\{y_i\}$. In machine learning settings, a task is split into training and test sets, which we'll denote as $\tau^{(r)}$ and $\tau^{(s)}$ respectively. The goal is to learn some θ that parameterizes a function f such that

$$\theta = \arg\min_{\theta} \sum_i L_\tau\Big(y_i, f(x_i;\theta)\Big) \tag{1}$$

With low-bias models such as neural networks, this can require large amounts of data to achieve competitive performance. This is not always feasible in real-world settings, where data can be sparse and quick acquisition of new skills is highly prioritized. Meta-learning seeks to alleviate these issues by learning a joint model for a number of different but related tasks. Intuitively, this can be thought of as learning reasonable biases for tasks in this domain. Somewhat more formally, meta-learning can be written as trying to minimize the following loss (although the term is fairly vague, so there are many formulations).

$$\theta = \arg\min_{\theta} \sum_i E_{x_i, y_i \sim \tau_i^{(s)}} \left[L_{\tau_i}\Big(y_i, g_i(x_i)\Big) \right] \tag{2}$$

where

$$g_i = f(\tau_i^{(r)}, \theta)$$

is usually assumed to be some model that is fine-tuned for τ_i using the respective training data. Perhaps the most famous approach in this domain is Model-Agnostic Meta-Learning[1] (MAML), which tries to learn good starting parameters θ that allow for solid results after a limited number of steps. In the formulation above, f would correspond to training (and returning) a model for a number of steps on task $\tau_i^{(r)}$, starting from parameters θ.

This problem is closely related to continuous learning, in which a model is presented with data that is not i.i.d. This distributional shift often takes the form of different tasks, trained one after another, and the primary difficulty is finding a single model that is able to handle new tasks well while not experiencing catastrophic forgetting of old tasks.

2 Methodology

As remarked in equation (1), all deep learning approaches attempt to minimize model parameters θ. The primary contribution of this work is to propose decomposing the θ into the product of a weight matrix θ_W (shared between all tasks) and a (nearly) binary mask θ_M.

More specifically, we propose taking the θ above and reformulating it as

$$\theta = \theta_W \odot \sigma(\theta_M) \tag{3}$$

where $\odot$ is the Hadamard product (element-wise multiplication) and σ is the element-wise sigmoid function, $\sigma(s) = \frac{e^s}{1+e^s}$. We note that this formulation does not lose any expressivity, as setting $\theta_W^{(ij)} = \frac{\theta^{(ij)}}{\sigma(\theta_M^{(ij)})}$ trivially results in recovering any desired θ.

[1]The introduction and methodology are lifted nearly exactly from the project proposal, as all of the information was the same. If this is undesirable, please inform us and we can re-do this section.

The intuition here is that θ_W learns an embedding from the input space to some latent space of features that are useful for multiple tasks. However, not all features are useful for all tasks, so it may be desirable to ignore some features for some tasks. This is achieved via the binary mask, $\sigma(\theta_M)$. For most values of $\theta_M^{(ij)}$, $\sigma(\theta_M)^{(ij)}$ will be near either 0 or 1. This results in effectively allowing θ_M to turn different elements of θ_W 'on' or 'off' for each task.

One can compare this idea to the lottery ticket hypothesis [2], which showed that, even in randomly initialized networks, there usually exist subnetworks that are able to perform reasonably well on any given task. Any subnetwork of a larger network is equivalent to a selectively masked version of the larger network [7]. By this sequence of reasoning, even if θ_W is poorly optimized for any given task, we can expect that there exists some mask defined by θ_M for which the network defined by equation (3) performs well on that task.

2.1 Details

Given a set of tasks T, the proposed paradigm is as follows:

1. Randomly initialize θ_W
2. Randomly select $\tau_i \in T$ with uniform distribution
3. If τ_i has not been selected before, randomly initialize a new corresponding $\theta_M^{(i)}$
4. Train the network given by $f(\theta = \theta_W \odot \sigma(\theta_M))$ with steepest gradient descent for N steps
5. Save the updates to θ_W and $\theta_M^{(i)}$
6. Repeat from (2)

Crucially, due to the nature of the back-propagation algorithm, one update of θ_W and $\theta_M^{(i)}$ has the same time complexity as ordinary backpropagation of θ. This is in contrast with other methods such as MAML [1], Meta-gradient Reinforcement Learning [6], MetaGenRL [3] and others, which often require computing the Hessian of the weights.

During test time, if the testing task has been seen before, the corresponding mask weights $\theta_M^{(i)}$ can be looked up from training time. If the task has not been seen before, it's possible to use a linear combination of weights from existing strategies, as is done in [5] for continual learning.

One notable difficulty is that the gradient $\nabla_s \sigma(s) = (1 - \sigma(s))\sigma(s)$, gets very small as $\sigma(s)$ gets close to either 0 or 1. Ideally, we would like $\sigma(s)$ to take on values that are close enough to 0 & 1 to turn values on and off, but not so small as to make the gradient near zero, since that would result in very slow convergence.

3 Experiments

In each of the experiments below, 'gated' refers to models that use our method in all layers. The 'separate' baseline trains a completely separate model for every singe task. Essentially, it trains on each task as if all of the other tasks do not exist. The 'shared' baseline trains a single model that is shared between all tasks. In other words, this can be thought of as mixing all of the tasks into one and training on that single conglomerate task without making distinctions.

For every experiment, every single hyperparameter is kept the same between our models and the baselines. Each of the models uses the same architecture, the same learning rate, the same pre-processing, etc. This was done to ensure that all comparisons are fair and balanced.

3.1 Linear Regression

The first experiment used linear regression with a synthetic dataset as a simple sanity check. Each of the T tasks consisted of a set of input points and regression targets. For each task, the input points were randomly sampled over an interval, while the regression targets were a linear function of the input points: $y_i = (A \odot A_M^{(\tau)})x_i + (b \odot b_M^{(\tau)})$. A and b are shared between all tasks, but A_M and b_M are matrices of 0s and 1s that are unique to each task. The train set is sampled from the same interval as the test set for these tasks. The loss function was simply the mean-squared error between predictions and regression targets, as it will be in all following tasks until MNIST.

This experiment was chosen as a very simple proof-of-concept where our model should succeed. The $A \odot A_M$ factorization perfectly matches that of the model, so our gated model is able to share information on the value of A between tasks while

Test Loss (Linear Regression with Noise, MSE, $\sigma = 0.4$)

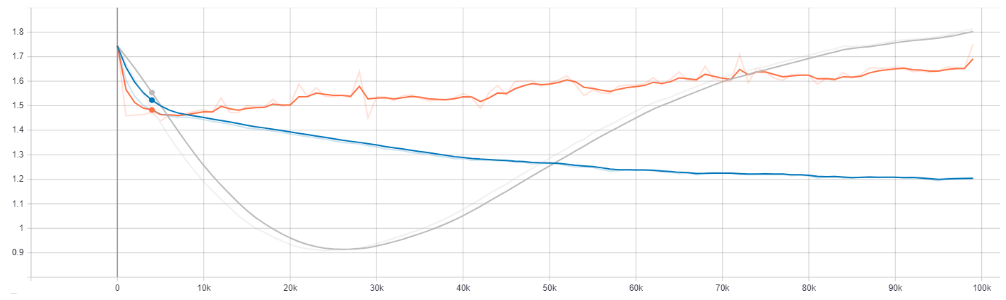


Figure 1: Key: *Grey* - Separate, *Orange* -Shared, *Blue* - Gated (ours)

specializing the mask for each task in the model to fit the corresponding $A_M^{(\tau)}$. Further, the data is perfectly linear, so it is a very simple debuggable model.

Despite its simplicity, this experiment actually proved quite insightful. Contrary to expectation, our model initially did *worse* than simply training separate models for each task. While both methods achieved (essentially) zero loss, our gated model converged more slowly than training separate models. We were able to deduce that this was primarily caused by the fact that the gradients flowed more slowly through the sigmoid function, so the masks took longer to converge. Additionally, the τ^{th} mask was only updated whenever the corresponding task was selected, so it received $\frac{1}{T}$ as many updates as the shared weights.

The solution to these problems was to have different learning rates for the masks and shared weights. We multiplied the learning rate of the masked weights by 4 * T, because the maximum of the derivative of the sigmoid function is 0.25. This largely alleviated the convergence speed issues. After running for 1,000 timesteps, we observed that all of the shared weights had converged to within $1e-8$ of A and all of the mask outputs (after applying sigmoid) had converged to within $1e-14$ of the optimal value. This demonstrated that the method was able to achieve it desired outcomes.

3.2 Linear Regression with Noise

Despite fixing the convergence speed issues, the previous experiment wasn't a complete win, because both the separate and gated models achieved near-perfect regression. This is expected - it's totally possible to learn a linear function based on N data points alone, where N is the number of dimensions + 1.

The next experiment aimed to break the tie. It kept the same data generation method as the first experiment, but added normally-distributed noise to the regression target Additionally, we drastically decreased the amount of data available (e.g. the number of train samples) for each task, so that the noise couldn't easily be overcame by using more data. We reduced the data so that there was barely any more data for each task than there were dimensions in the input space. This allowed our model to show the advantage of sharing information between tasks (effectively gaining more data points).

Adding noise to the experiment yielded a couple more discoveries. Firstly, our model behaved largely as we expected, and outperformed both the separate and shared models, although the separate models remained competitive.

Figure 1 is a sample run of of the average loss on the test set [2]. Note that the mean-squared-error is still relatively high for both models, due to the fact that the noise was quite large. We discovered that if the noise was large enough, the 'mask' would often get stuck being set to the wrong value, yielding large errors for the gated model. However, in the cases where this happened, the other models also performed extremely poorly, so those problems may just be hard.

3.3 Sinusoidal Regression

Figure 2 visualizes how the gated model allowed the weights to cluster around either 0 or a few useful values after multiplying by the mask.

This experiment again builds off of the first and is inspired by a motivating example in [1]. We keep the regression target from the first experiment, but the regression target is the sine of the original regression target. The plan was originally to randomly use either sine or cosine, but using cosine was equivalent to just using a different bias, so it was removed for simplicity.

As the sine function causes the problem to be nonlinear, the models also switched from being linear as well. Instead, we use a simple feed-forward neural network with three layers and the ReLU activation function. When testing the gated model, all linear layers are replaced with gated linear layers instead.

Intuitively, the model could have the first layer learn something similar to the gated function used in the first two experiments. The last two layers could then be used to learn the sine function. If the gated model were to exhibit this

[2]Note: due to the way LaTeX handles figures, the figures are not always on the same page as the corresponding text. To help with this, every reference to a figure is a link to that figure.

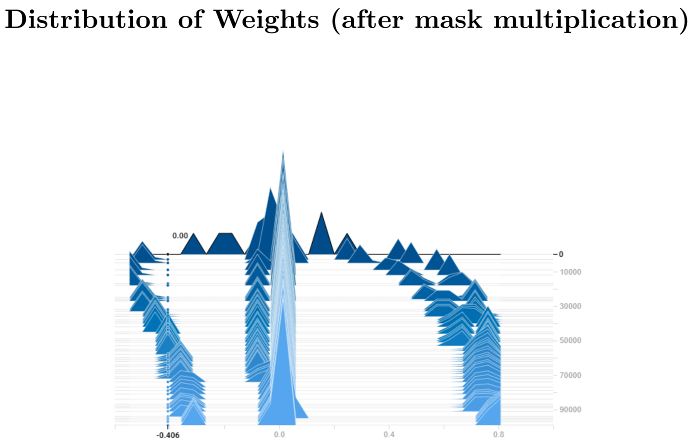


Figure 2: Histogram of weights in the network over time. Plots towards the back are from earlier epochs. Note how the histogram becomes more spiky and concentrated over time, showing different tasks' weights converging to the same values.

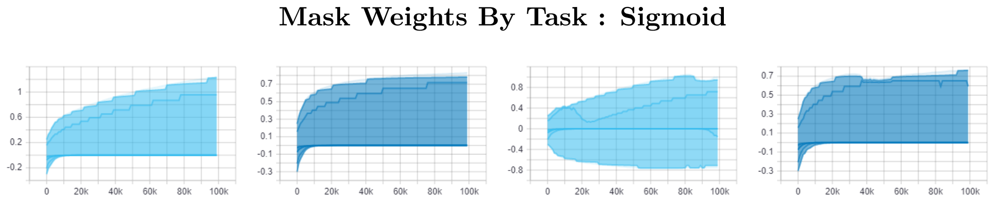


Figure 3: Shown: Distribution of weights in the mask portion second layer of network during sigmoid tests. Note that for most tasks, the distribution is nearly identical, confirming intuitions.

behaviour, then we would expect the first layer to look similar to how it did in the linear examples. Meanwhile, the layers learning the sine function should be the same for all of the tasks, meaning that their masks should be similar [3].

Plotting the weights, as demonstrated in Figure 3, confirms this intuition: most of the weights in the second layer have nearly identical distributions, indicating that the model has learned that this information can be shared between tasks.

It became apparent, however, that the masks did not always converge to 1 or 0 - sometimes the model would overfit and reach a value in between. With this in mind, we added a form of regularizer to the system. Typically, when one is using a regularizer, the goal is to bias the weights to be nearer to 0. However, in this situation, biasing the masks to 0 or 1 corresponds to biasing the mask weights to $\pm\infty$, because of the nature of the sigmoid function. We were unable to find any existing regularization techniques that did this, so we devised the following two regularization strategies [4]:

$$L_R(\theta) = \sum_i e^{-\text{abs}(\theta_i)} \tag{4}$$

$$L_R(\theta) = \sum_i e^{-\theta_i^2} \tag{5}$$

Both of these regularizers have a couple of desirable properties. First, they are symmetric and the derivative points away from 0 for all values besides 0. Second, the gradient decreases in magnitude as it moves away from the origin, meaning that the values don't rapidly accelerate to $\pm\infty$.

A plot of the gradients of these two functions is shown in Figure 4.

Note that these regularizers were only applied to the mask weights, and not the shared weights. Unfortunately, while they are interesting in theory, these regularizers failed to improve model performance, as reported in Figure 5

3.4 Image Classification - MNIST

In this experiment, a convolutional neural network (CNN) is used to classify black & white images of digits. For the base case, there is no meta-learning component; we just want to examine whether our method performs similarly to a baseline CNN.

[3]Ideally, all of the weights would be 'on' to provide the network with the most flexibility, but this is not necessarily a given.

[4]For the first equation, we use the subgradient with 0 at $\theta_i = 0$.

Gradients of regularizers with respect to a single weight

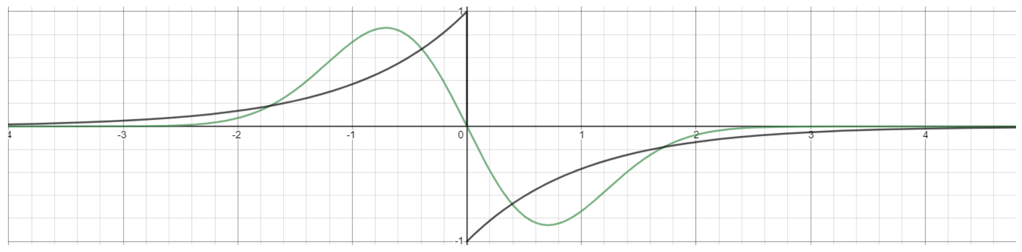


Figure 4: In Black: Regularizer shown in Equation 4, using absolute value. In Green: Regularizer shown in Equation 5, using squared loss.

Test Loss of Model by Regularizer

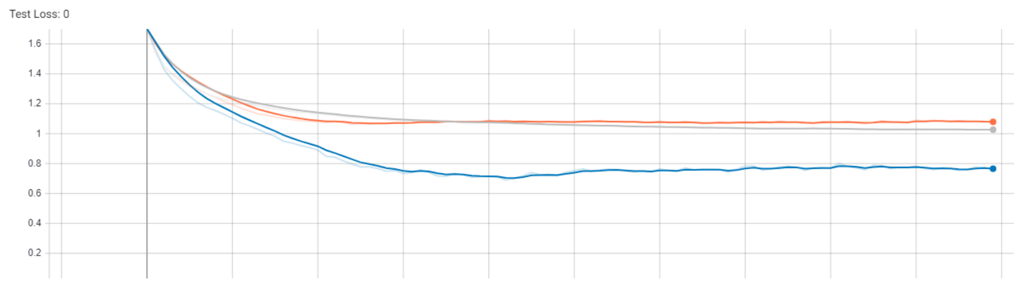


Figure 5: In Grey: Absolute value regularizer shown in Equation 4. In Orange: Squared loss regularizer from Equation 5. In Blue: No Regularizer.

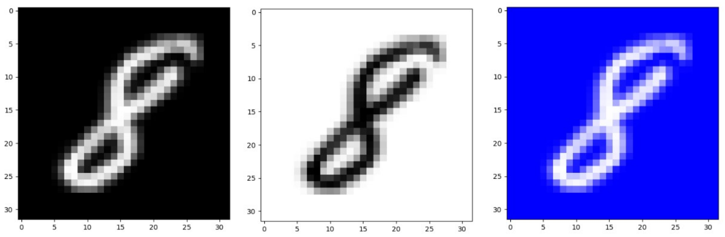
We chose to use conventional MNIST dataset because it is readily available online, and it is commonly used as a baseline for computer vision tasks.

Our choice of CNN was the LeNet model, which is an established CNN proposed in [4]. This is a small, space-efficient model that has been implemented by many people. Our implementation of LeNet was modified to be able to be set as either gated handling n tasks or non-gated (as originally proposed). Since the LeNet CNN is typically composed of linear and convolutional layers, we included an option while making the model to replace each layer with our own gated versions to support n different tasks.

One of our struggles while implementing the project was in limiting memory usage to a feasible amount. In the MNIST dataset provided by Pytorch, there are a total of 70,000 images, divided into 60,000 images for training and 10,000 for test. Because each image nominally contains 28x28 pixels and we wanted to run our experiments on 3 channels for each image (RGB format), each MNIST task ended up needing a lot of CPU space, approximately 22 GB of RAM. Running the data through LeNet further increased the memory usage. We found that, especially wanting to test different variations of the MNIST dataset as different tasks, it was infeasible to store data as stacked tensors as originally planned. This issue was resolved by loading the samples in only as they were used, and transforming the data on the spot.

We wanted to format our image data in 3-channel RGB format for a couple reasons: we wanted to verify that LeNet would perform approximately as well with either 1 or 3 channels of image information, and we wanted to set up a framework for the next experiment in training on colored MNIST data.

For ease of generalization, we wrote two different transforms for nominal white-digit-on-black-background MNIST images: one to make the black background into a different RGB color and another to invert the image to black on white and make the digit a different RGB color. Since we get the pixel data as values in the [0,1] range, the transforms take in a length 3 array representing an RGB color with values ranging from 0 to 1. Each channel of color replacement is represented by the transformation $original + (1 - original) * inColor[channel]$, where $original$ represents the nominal grayscale 1-channel image representation. Then, the different channels are stacked into a 3-channel tensor, making our final RGB image. The opposite process of $original * inColor[channel] + (1 - original)$ is performed for color inversions.

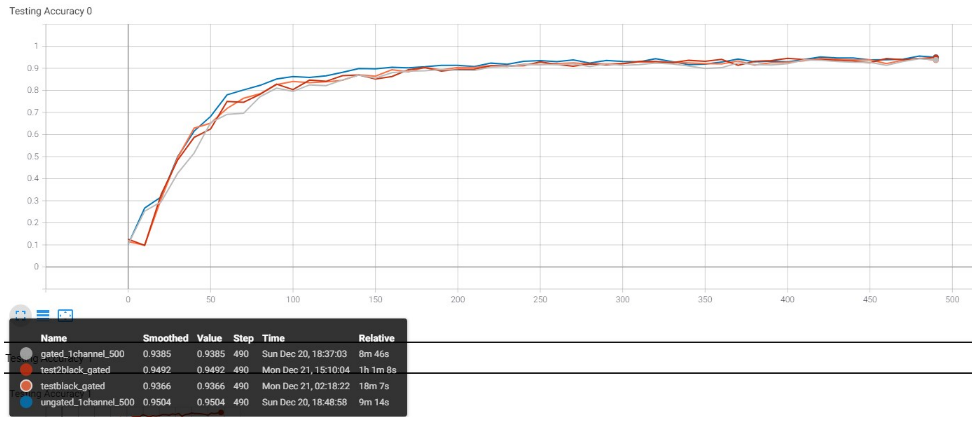


In order: Regular black background, color-inverted and background color-replaced MNIST images

For MNIST experiments, we primarily chose to log accuracy and loss results for both training and test data every 10 timesteps, where each timestep involved training our model by one 32-image batch, for a total of 500 timesteps. We found that 500 timesteps were sufficient to capture much of the model training behavior for different tasks. For future work, we would recommend examining even more long term behavior by surpassing 500 timesteps of training.

Cross-entropy loss was used for training, and accuracy was reported for both the train and test set, across all tasks. The accuracy of the model was calculated by predicting the classification of each image, then comparing to accurate labels and taking the percentage of images that were predicted correctly.

Given nominal MNIST images (white digits on a black background), we wanted to first verify gated LeNet results with 1-channel image inputs, then with 3-channel image inputs. We also examined behavior when feeding in the same nominal black-background 3-channel MNIST training data as two separate tasks for our gated model (shown below).



	Name	Smoothed	Value	Step	Time	Relative
●	gated_1channel_500	0.9385	0.9385	490	Sun Dec 20, 18:37:03	8m 46s
●	test2black_gated	0.9492	0.9492	490	Mon Dec 21, 15:10:04	1h 1m 8s
●	testblack_gated	0.9366	0.9366	490	Mon Dec 21, 02:18:22	18m 7s
●	ungated_1channel_500	0.9504	0.9504	490	Sun Dec 20, 18:48:58	9m 14s

As we can see from 500 training timesteps, our gated model performed very similarly between 1-channel and 3-channel versions of nominal MNIST images. This was as we expected, although our gated model trained a bit slower than the non-gated model on 1-channel images. We expect that, over many timesteps, our gated model will eventually match or surpass the peak performance of the non-gated model.

The interesting experiment here is labelled test2black_gated. By training on two different tasks that are both the full 3-channel black-background MNIST training images, the test accuracy actually improves for both tasks, which in the model share a weight and bias but have different masks. One possible explanation is that, by effectively training the shared values twice as quickly as each set of masks, the masks are better able to adapt to the data over a smaller number of training steps. This has potentially impactful applications toward training a model on a relatively limited dataset without actually increasing the amount of data used.

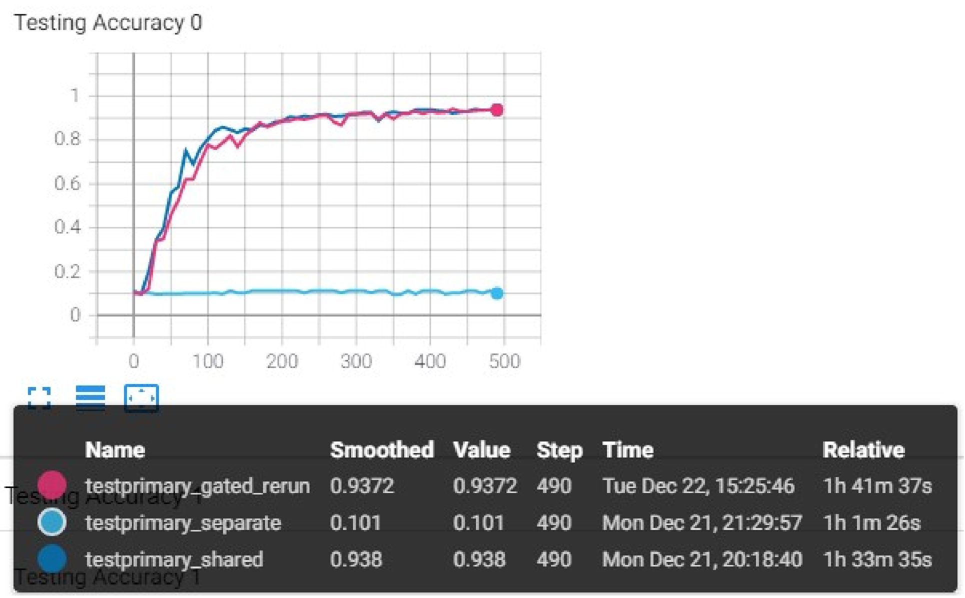
3.5 Image Classification - Colored MNIST

This experiment is the same as the one before, except that there are now a number of tasks that each take the black background in the MNIST digits and replace it with a background of a random color. The intent of this experiment is to test whether the algorithm is able to scale to non-trivial domains.

Using the background color setting and color inversion transforms described in the previous experiment, we were able to set the background as different specific colors transitioning into a white digit or invert the black/white colors of the background and digit.

The first experiment replaced the traditional black background with the three primary colors: red, green and blue. In changing the background color from black to any other color, we change the contrast between the white digit (white is equivalent to having the maximum value for all of red, green and blue) and the background. Inherently, by setting the background to each maximum value of primary color (i.e. [1,0,0] for pure red), we render one channel of the image data inconsequential in determining what data the image represents. For example, by setting the background to the maximum red value, the entire 32x32 channel corresponding to red is populated with all 1, so there is no way to distinguish between digits using the red channel, but the green and blue channels still have relevant information.

Intuitively, by setting different tasks to backgrounds of different primary colors, we expect classification to be more difficult than in standard MNIST because of the decreased contrast between white and the background. The following image shows test accuracy results for the task where the background is set to red, and the other tasks perform similarly.

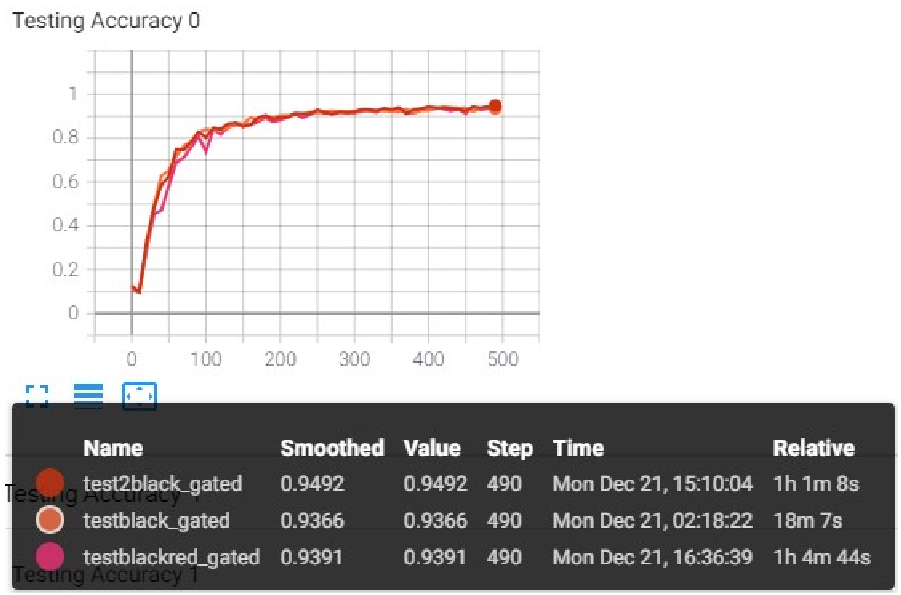


	Name	Smoothed	Value	Step	Time	Relative
●	testprimary_gated_rerun	0.9372	0.9372	490	Tue Dec 22, 15:25:46	1h 41m 37s
●	testprimary_separate	0.101	0.101	490	Mon Dec 21, 21:29:57	1h 1m 26s
●	testprimary_shared	0.938	0.938	490	Mon Dec 21, 20:18:40	1h 33m 35s

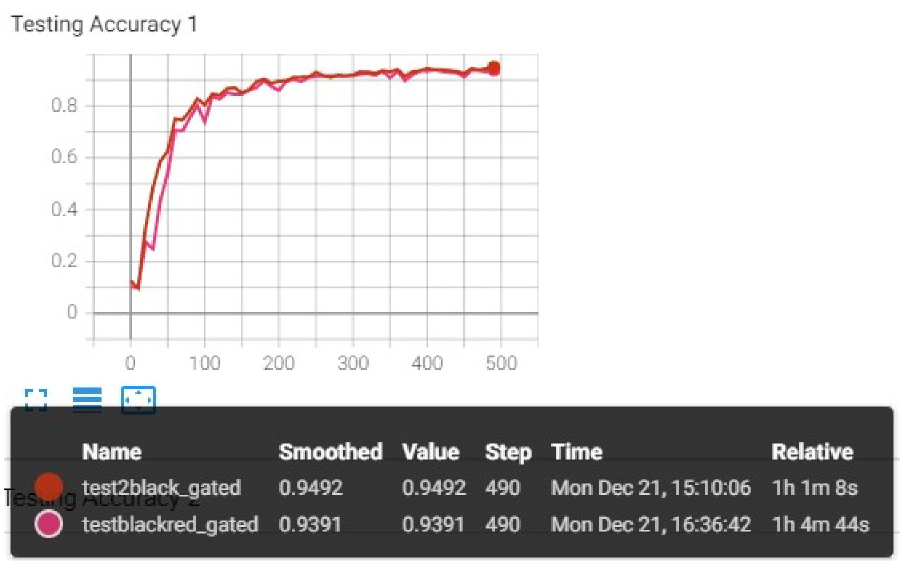
From our experiment, we can see that the separate model (a different LeNet model for each task) performs the most poorly. This makes sense in that, with the decreased contrast between the digit and the background, it is more difficult for each model to learn digit classification.

However, we can see that both the gated and shared models performed quite well. For the gated model, we expected the different masks associated with different tasks to adapt and adjust to each colored task. This behavior is equivalent to independently learning which channels are relevant or irrelevant and otherwise maintaining a shared criteria for digit classification. We originally did not expect the shared model to perform as well, but its performance is very similar to that of the gated model.

As an extension of the previous section testing performance of the gated model on black-background MNIST images, we also wanted to test by training on two tasks: one set of data with black backgrounds and another set of data with red (altered) backgrounds. The first graph represents test accuracy on the black-background images and the second represents test accuracy on the red-background images.



	Name	Smoothed	Value	Step	Time	Relative
●	test2black_gated	0.9492	0.9492	490	Mon Dec 21, 15:10:04	1h 1m 8s
●	testblack_gated	0.9366	0.9366	490	Mon Dec 21, 02:18:22	18m 7s
●	testblackred_gated	0.9391	0.9391	490	Mon Dec 21, 16:36:39	1h 4m 44s

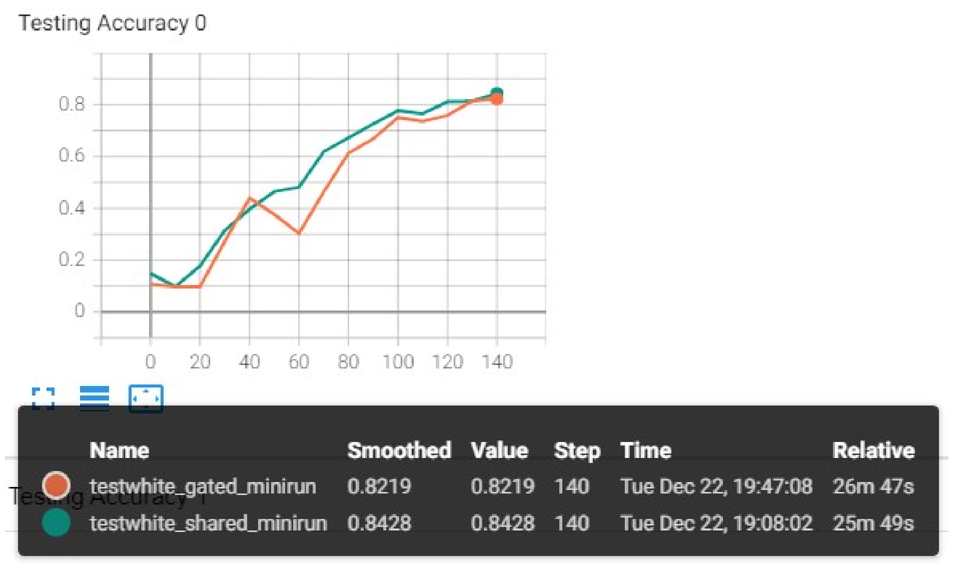


	Name	Smoothed	Value	Step	Time	Relative
●	test2black_gated	0.9492	0.9492	490	Mon Dec 21, 15:10:06	1h 1m 8s
○	testblackred_gated	0.9391	0.9391	490	Mon Dec 21, 16:36:42	1h 4m 44s

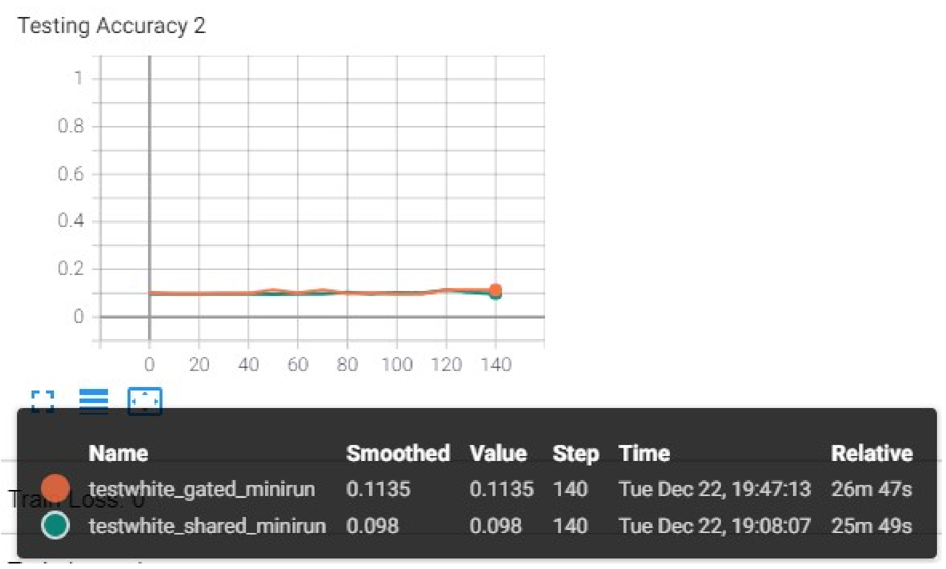
From these results, we can see that the gated model performs similarly on both tasks when training on both the black-background dataset and the red-background dataset as when training on only the black-background dataset. Although we do not get the same amount of accuracy as training on two black-background datasets, this shows that we do not lose performance when concurrently training our model on additional tasks.

One test case that we wanted to address was training performance when one of the tasks was completely white images. This was a setup to figure out how the models would perform when part of the data was completely useless; there is no way to classify images as different digits when the entire image has the same values.

For a smaller number of training steps, the following graphs show the performance of the gated versus shared model on a red-background dataset and an all-white dataset, respectively.



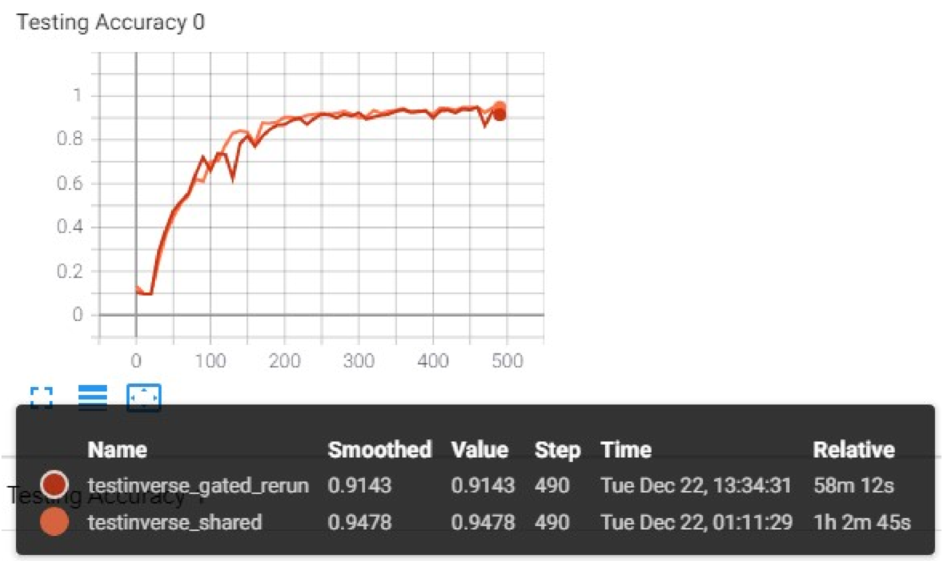
	Name	Smoothed	Value	Step	Time	Relative
●	testwhite_gated_minirun	0.8219	0.8219	140	Tue Dec 22, 19:47:08	26m 47s
●	testwhite_shared_minirun	0.8428	0.8428	140	Tue Dec 22, 19:08:02	25m 49s



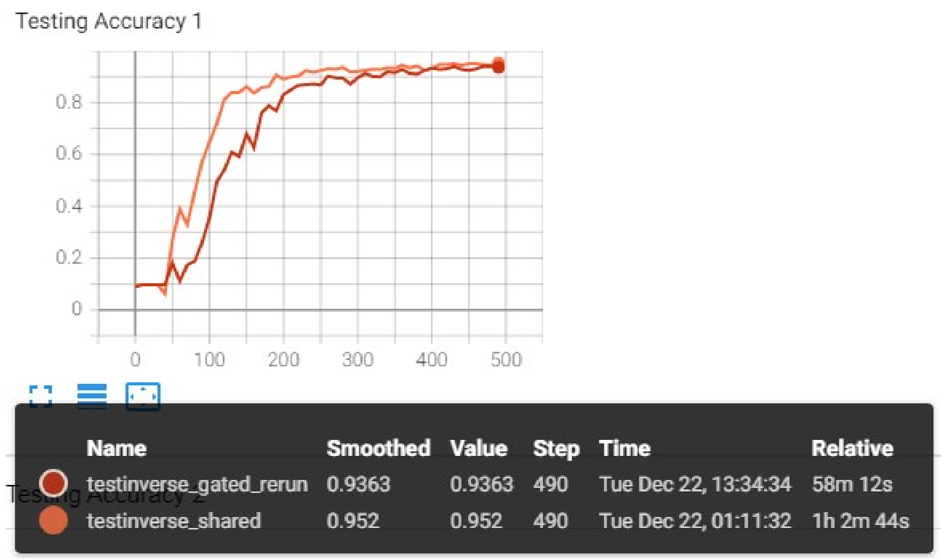
	Name	Smoothed	Value	Step	Time	Relative
●	testwhite_gated_minirun	0.1135	0.1135	140	Tue Dec 22, 19:47:13	26m 47s
●	testwhite_shared_minirun	0.098	0.098	140	Tue Dec 22, 19:08:07	25m 49s

From these initial experiments, we can see that, while each model was able to start training on the first red-background task, there is no improvement in classifying the all-white dataset, as expected. This shows that the shared and gated models can both ignore completely irrelevant data in image classification.

Another experiment we did was in classifying image inverses. One task was the regular white digit on a black background, and another was a black digit on a white background. Their respective graphs are shown here in order.



	Name	Smoothed	Value	Step	Time	Relative
●	testinverse_gated_rerun	0.9143	0.9143	490	Tue Dec 22, 13:34:31	58m 12s
●	testinverse_shared	0.9478	0.9478	490	Tue Dec 22, 01:11:29	1h 2m 45s



	Name	Smoothed	Value	Step	Time	Relative
●	testinverse_gated_rerun	0.9363	0.9363	490	Tue Dec 22, 13:34:34	58m 12s
●	testinverse_shared	0.952	0.952	490	Tue Dec 22, 01:11:32	1h 2m 44s

Although the gated model trained a bit slower with these tasks, both the gated and shared models are able to effectively train on both the regular and inverted images. This is perhaps due to LeNet relying on transitions between the digit and the background to train instead of specifically a transition from white to black.

Our main surprise was in the performance of the shared model. Since this model treats all data from all tasks as if they came from the same dataset, we originally theorized that it would perform worse than the gated model. Especially in the experiment with an all white dataset, we thought that feeding the shared model labelled useless information would impact its performance. Instead, the shared model performed quite well in all the test cases that we tried. We suspect that, by sampling from a variety of different tasks, the LeNet model is able to optimize itself to a lower level of contrast between the digit and the background, which is applicable to any of the channels. Even when effectively taking away one channel as a source of information, we can effectively train a regular LeNet model as if it were a 1-channel or 2-channel image input.

4 Conclusion

The experiments demonstrate that some types of tasks perform better under a shared model, and others perform better under a separate model. The consistency that we see here is that our proposed gated model always performs similarly to the best of the shared and separate models. This positions it as a flexible method that is able to dynamically share various amounts of information between tasks.

The benefit of using a gated model is that we effectively have a composite model training against n tasks, where all tasks are being trained on concurrently with the same overarching weight and bias values, while the masks particular to each task allow for differentiation. In this way, we essentially train n models at the same time while matching performance of either separate or shared conventional models. This is cheaper than training the models individually and can also be saved more cheaply - in production, one could save the masks as simply zeros and ones.

Although this aspect wasn't explored in the paper, the masks can also be useful for model interpretability. It allows a human to identify which features the model is using for each task, as the weights on irrelevant features are set to zero. This could be a step towards understanding what models are paying attention to in different tasks, without too much work.

In the future, this work can be extended to try to increase the robustness of the masks and demonstrate the architecture's effectiveness on different (and more difficult) domains. This gating method offers an exciting possibility because of its low training cost, versatility and implementation. The results above demonstrate that the method is stable and able to perform well across a wide range of tasks.

5 Roles of Collaborators

Carter Blum devised and communicated the formulation of the problem and the 'gated' method that was used to tackle it. He designed each of the experiments and acquired the necessary data. He wrote the base of the code for training, implementing gated layers, tensorboard logging, and the shared/separate baselines. Carter also executed the first three experiments in this paper, as well as all of the improvements listed.

Ashley Law modified the convolutional LeNet layer to support the 'gated' method. She executed and ran the last two MNIST experiments. Ashley debugged issues with CPU usage for the MNIST experiments as well as reconfiguration of data into colored RGB form. She also devised different test cases to formally compare the 'shared' and 'gated' models in different use cases for colored MNIST experiments.

Both teammates did data analysis and writing. We frequently discussed decisions on how to improve the project together and how to get past hurdles that we were encountering. Ashley in particular was very good at facilitating communication.

References

[1] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[2] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[3] Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

[4] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015.

[5] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *arXiv preprint arXiv:2006.14769*, 2020.

[6] Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *Advances in neural information processing systems*, pages 2396–2407, 2018.

[7] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, pages 3597–3607, 2019.