# Efficient Few shot learning with FiLM

Manikanta Srikar* and Chandra Prakash*

***Abstract*— We have extended the work done in Model Agnostic meta learning by introducing FiLM: Feature-wise Linear Modulation layers for classification and reinforcement learning. It is efficient because gradiant updates (second order) is done on fewer parameters. The model is partitioned into two parts: context parameters (FiLM layers) that are adapted to individual tasks and shared parameters that are common across all the tasks. We show emperically that our approach achieves performance close to MAML in few shot classification. Code :- https://github.com/KonkimallaChandraPrakash/FiLM-for-Meta-Learning-and-Reinforcement-Learning**

## I. INTRODUCTION

Recent years have seen rapid progress in meta-learning methods, which learn (and optimize) the performance of learning methods based on data, generate new learning methods from scratch, or learn to transfer knowledge across tasks and domains. A key challenge in meta learning is fast adaptation: learning on previously unseen tasks fast and with little data.

A recent approach for meta learning is Model Agnostic Meta learning (MAML) [1]. MAML trains over a wide variety of tasks and seeks for a representation that can quickly adapt to a new task with very few gradient steps. The meta-learner seeks to find an initialization that is not only useful for adapting to various problems, but also can be adapted quickly (in a small number of steps) and efficiently (using only a few examples).

However, while MAML trains the entire model for task specific update, other methods train only a fraction of the model, and have the rest of the model fixed across tasks. Adapting only some model parameters can not only make learning faster due to decrease in the number of trainable parameters, but also gives us the ability to use deeper models, as over-fitting and GPU bottleneck issues are bound to occur if the entire model is trained while using a very deep network.

We propose the usage of FiLM layers as context parameters in Meta learning. Like MAML, we learn a model initialisation that can quickly be adapted to new tasks. However, unlike MAML, we train only a small number of parameters to the new task. The main reason behind this approach is training fewer parameters should be helpful due to very few gradient updates in the inner loop. The FiLM parameters can be interpreted as conditioning based task embeddings that modulate the behaviour of the model.

## II. BACKGROUND

The goal of few shot meta learning is to train a model that can learn quickly to adapt to a new task using only a few gradient updates.

*equal contribution

### A. Few shot learning

Here we describe the method Model Agnostic Meta Learning [1]. The model is trained during the meta learning phase on a set of tasks so that during test time, the model quickly adapts to a new task using very few examples. Let us consider the model f, that maps input x to output a. During meta learning, the model is trained to be able to adapt to a large number of tasks. Each task $T = \{L(x_1, a_1 ... x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t)\}$ consists of a loss function, a distribution over the initial observations $q(x_1)$, and a transition distribution $q(x_{t+1}|x_t, a_t)$.

We have a distribution over tasks p(T) that we want our model to learn. In case of K - shot learning, the model is trained on a new task $T_i$ drawn from p(T) using k samples taken from $q_i$ and loss $L_{Ti}$ obtained from $T_i$. Initially, the tasks are segregated into training phase tasks $p_r(T)$ and test phase tasks $p_e(T)$. In training phase, task $T_i$ is sampled from $p_r(T)$ and the model is trained on k samples from each task. In meta- testing phase, new set of tasks not used in training phase, ie $p_e(T)$ are sampled, and the model is trained on them using k samples.

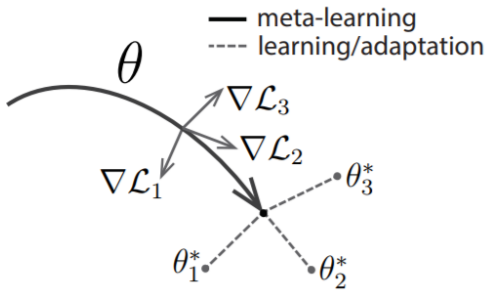


Fig. 1. Diagram of model agnostic meta learning [1] algorithm which learns a representation $\theta$ that can adapt to new tasks.

Context adaptive meta learning [2] follow a similar approach as ours. The model is partitioned into two parts: context parameters $\phi$ that are adapted into the inner SDG loop and parameters $\theta$ that are shared across tasks and meta-learned in the outer loop. Since $\phi$ are independent of the network input, for an output node $h_i^{(l)}$ at a fully connected layer l, the conditioning was done by concatenating $\phi$ to the inputs to that layer.

$$h_i^{(l)} = g\left(\sum_{j=1}^{J} \theta_{j,i}^{(l,h)}\ h_j^{(l-1)} + \sum_{k=1}^{K} \theta_{k,i}^{(l,\phi)}\ \phi_{0,k} + b\right)$$

where g is a non-linear activation function, b is a bias parameter, $\theta_{j,i}^{(l,h)}$ are the weights associated with input layer $h_i^{(l)}$ and $\theta_{k,i}^{(l,\phi)}$ are the weights associated with context parameter $\phi_{0,k}$.

Meta SGD [3] is an extension of MAML in which the alpha is learned together with the initialization, instead of being another hyper-parameter of the meta-learner. Compared to MAML, meta SGD has much higher capacity by learning to learn not just the learner initialization but also the learner update direction and learning rate all in a single meta-learning process.

### B. Reinforcement learning

Retro Contest [4] is a transfer learning contest that measures a reinforcement learning algorithms ability to generalize from previous experience. In typical RL research, the models are tested in the same environment in which they are trained, which favors the algorithms which are good at memorization. This contest tests algorithm's performance on unseen game levels. Environments were taken from the following games: Sonic The Hedgehog, Sonic The Hedgehog, and Sonic 3 Knuckles. All of these games have very similar rules and controls, although there are subtle differences between them.

The performance metric was mean score as measured across all the levels in the test set which is unknown to the contestants. At the training time, all the training levels can be used. At the test time each test level was played for 1 million timesteps. Each test level was played separately to avoid flow of information between various levels. In the infinite-timestep regime, there is no strong reason to believe that meta-learning or transfer learning is necessary. However, in the limited-timestep regime, transfer learning may be necessary to achieve good performance quickly. Openai has also provided a number of baselines which can serve as starting points for fancier algorithms. Figure 2 compares the baselines aggregate learning curves.

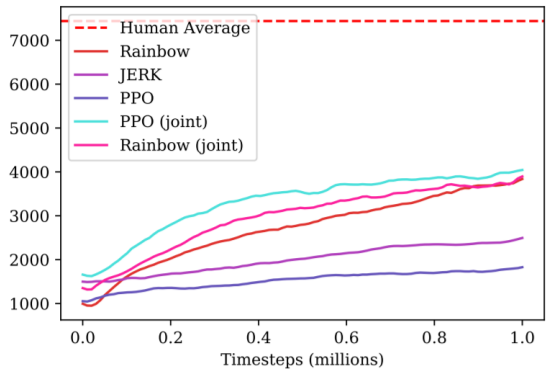


Fig. 2. The mean learning curves for all the baselines across all the test levels

### C. FiLM

FiLM [5] stands for Feature-wise Linear Modulation. FiLM layers influence neural network computation via a simple, feature-wise affine transformation based on conditioning information. They have been shown to be highly effective for visual reasoning. The structure of FiLM is shown in Figure 3.

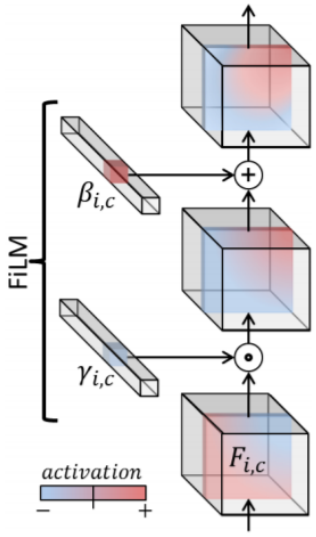


Fig. 3. A single FiLM layer for a CNN. The dot signifies a Hadamard product. Figure taken from [5]

## III. PROPOSED METHODOLOGY

In contrast to MAML which has trained their entire network in the task specific training, we propose to partition the model into two parts: context parameters $\phi$ (FiLM layers) that are trained in the inner SGD loop on individual tasks and common parameters $\theta$ that are shared across all tasks and meta trained in the outer loop.

**Algorithm 1** Our Meta learning algorithm

**Require:** $p(T)$: distribution over tasks
**Require:** $\alpha_1$, $\alpha_2$,: step size hyperparameters
randomly initialize $\theta$
**while** not done **do**
  Sample batch of tasks $T_i \sim p(T)$
  **for all** $T_i$ **do**
    $\phi : \{\beta, \gamma\}$ (FiLM parameters)
    Evaluate $\nabla_\phi L_{T_i}(f_\phi)$ with respect to $K$ examples
    Compute FiLM parameters with gradient descent: $\phi_i' = \phi - \alpha \nabla_\phi L_{T_i}(f_\phi)$
  **end for**
  Update $\theta \leftarrow \theta - \alpha_2 \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta_i'})$
**end while**

We consider a model represented by a parameterized function $f_\theta$, with common parameters $\theta$ and context parameters $\phi$. When adapting to a new task, the context parameters $\phi$ become $\phi'$. During one inner gradient update,

$$\phi_i' = \phi - \alpha \nabla_\phi L_{T_i}(f_\phi)$$

The step size $\alpha_1$ may be fixed as a hyperparameter or meta learned. In our experiments, we have compared the performance of fixed and meta learned step size. The meta-optimization is performed over the model parameters , whereas the objective is computed using the context parameters $\phi$. The meta-objective is as follows:

$$\min_{\theta} \sum_{T_i \sim p(T)} L_{T_i}(f_{\phi_i'}) = \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta - \alpha \nabla_\theta L_{T_i}(f_\theta)})$$

### A. Supervised learning

In supervised learning, we train a model $f : x \mapsto \hat{y}$ that maps data points $x \in \mathcal{X}$ that have ground truth label $y \in \mathcal{Y}$ to output. A task $T_i$ is defined as a tuple $\mathcal{T}_i = (\mathcal{X}, \mathcal{Y}, \mathcal{L}, q)$, where $\mathcal{X}$ is the input space, $\mathcal{Y}$ is the output space, $\mathcal{L}(y, \hat{y})$ is a task-specific loss function, and $q(x, y)$ is a distribution over labelled data points. We assume that all data points are drawn i.i.d. from $q$ $\hat{y} \in \mathcal{Y}$. Different tasks can be created by changing any element of $\mathcal{T}_i$.

The goal of few shot learning is to learn a new function from only a few input/output pairs for that task, using prior data from similar tasks for meta-learning. K-shot classification tasks use K input/output pairs from each class, for a total of NK data points for N-way classification.

**Algorithm 2** Few-Shot Supervised Learning

**Require:** $p(T)$: distribution over tasks
**Require:** $\alpha_1$, $\alpha_2$: step size hyperparameters
1: randomly initialize $\theta$
2: **while** not done **do**
3: Sample batch of tasks $T_i \sim p(T)$
4: **for all** $T_i$ **do**
5: Sample $K$ datapoints $\mathcal{D} = \{x^{(j)}, yt^{(j)}\}$ from $T_i$
6: Evaluate $\nabla_\theta L_{T_i}(f_\phi)$ using $\mathcal{D}$ and $L_{T_i}$
7: Compute FiLM parameters with gradient descent: $\phi_i' = \phi - \alpha \nabla_\phi L_{T_i}(f_\phi)$
8: Sample datapoints $\mathcal{D}_i' = \{x^{(j)}, y^{(j)}\}$ from $T_i$ for the meta-update
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $L_{T_i}$
11: **end while**

### B. Reinforcement learning

Openai retro is a Transfer learning contest that measures a reinforcement learning algorithms ability to generalize from previous experience.

We have built our model on top of the contest's 3rd place contestant, aborg [6]. It is based on Joint PPO, one of the baseline models provided by openai. we use a simple joint training algorithm, wherein we train a policy on all the training levels and then use it as an initialization on the test levels. Each training level has a unique set of FiLM parameters, and rest of the network is shared across all the training levels.

During meta-training, we train a single policy to play every level in the training set. We run 188 parallel workers, each of which is assigned a level from the training set. At every gradient step, all the workers average their gradients together, ensuring that the policy is trained evenly across the entire training set. This training process requires hundreds of millions of timesteps to converge, since the policy is being forced to learn a lot more than a single level. Once the joint policy has been trained on all the training levels, we fine-tune it on each test level during whicha new set of FiLM parameters are learned from scratch and we let all the parameters update.

## IV. Experimental Evaluation

In this section we try to show the effectiveness of FiLM layer in context adaption. These are the extensions which we tried on FiLM:-

*1) Channel wise FiLM :* Here we use FiLM on a convolutional block, which is effectively the channel multiplied by gamma parameter shifted by a corresponding beta parameter. This is similar to channel wise importance. Related fig 4.

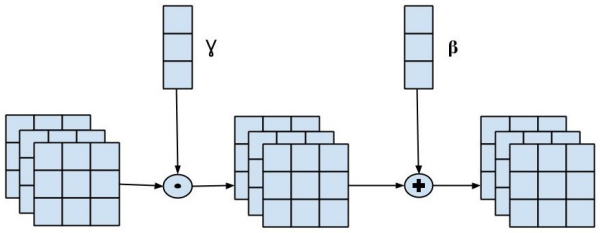


Fig. 4. Channel wise FiLM

*2) Spatial wise FiLM:* Here we use FiLM on a convolutional block but spatially. $\gamma$ and $\beta$ have the same spatial dimension as the feature map. This results in gamma parameter being multiplied to every pixel and shared accross channels and shifted by a corresponding beta parameter. This is similar to spatial wise importance. Related fig 5.

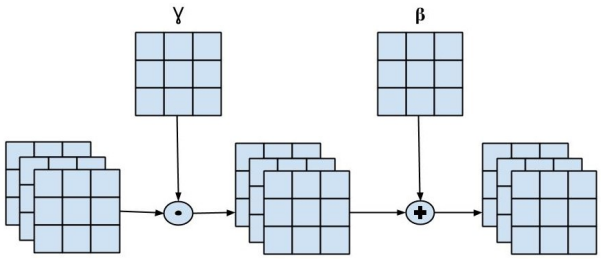


Fig. 5. Spatial wise FiLM

*3) ConvBlockwise FiLM:* In this case, $\gamma$ and $\beta$ not only have the same spatial dimension as the feature map, but are also not shared across channels. Instead they are different across the channels Here we used FiLM on a convolutional block, every element gets multiplied by gamma parameter and gets shifted by a corresponding beta parameter. The number of parameters are huge in this case. Related fig 6.

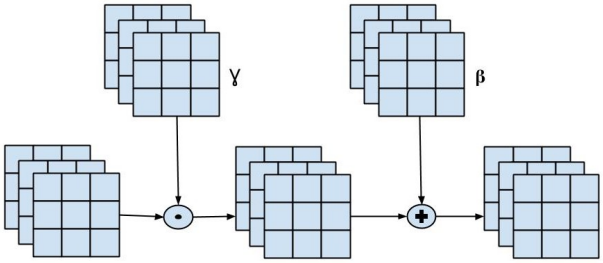


Fig. 6. ConvBlockwise FiLM

*4) Fully Connected FiLM:* Here we used FiLM on a fully connected output, every element gets multiplied by gamma parameter and gets shifted by a corresponding beta parameter. Related fig 7.

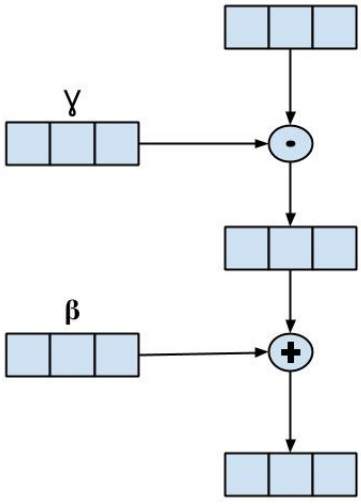


Fig. 7. Fully Connected FiLM

*5) Projection FiLM:* Here we project the parameters from a low dimensional vector to a high dimensional space. The projected parameters are used as channel wise FiLM parameters. Related fig 8.

### A. Few Shot Learning

We evaluate our method on the Omniglot dataset [8]. The Omniglot dataset consists of 20 instances of 1623 characters from 50 different alphabets. Each instance was drawn by a different person. The MiniImagenet dataset was proposed by Ravi Larochelle (2017), and involves 64 training classes, 12 validation classes, and 24 test classes.

MAML used a standard 4 layer CNN each containing 64 channels dimension(3x3 Convs, 2x2 Maxpools) and 2 fully connected layers. During the inner update of MAML each of the sub tasks are given separate parameters and are updated. But in our case we choose not to update all parameters but only update FiLM parameters during the inner update. We try all the above listed variations of FiLM on the same CNN as MAML. We also change the inner update alpha to vary across all the subtasks which is mentioned as Adap LR in the table. We have tried Sigma dimension of 4,8,16 and 24.

Further as the number of updates is relatively small we could use ResNet-18, having FiLM parameters in each conv layer but we only try it with Channel wise variant of FiLM as the number of parameters blewup if we try it in normal MAML case.

We have evaluated the CNN on Omniglot in 5-Way 1shot and 20-Way 1shot. The problem of N-way classification in general is set as follows: choose N unseen classes, provide the model with K unique instances for each of the corresponding the N classes, and then evaluate the models ability to classify new instances within the N classes. For Omniglot, we randomly select 1200 characters for training, irrespective of alphabet, and use the remaining for testing. We have used a meta-batch size of 32 in 5-Way 1shot and 16 in 20-Way 1 shot. The results are in table. Each of those experiments took around 2-3 days on a 24gb GPU.

### B. Reinforcment Learning

OpenAI's benchmark use three similar games: Sonic The HedgehogTM, Sonic The HedgehogTM2, and Sonic 3 Knuckles. All of these games have very similar rules and controls, although there are subtle differences between them. These observations are always a 24-bit RGB image, screen images are 320 pixels wide and 224 pixels tall. A small subset of all possible button combinations makes sense in Sonic. In fact, there are only eight essential button combinations:

$$[\ ], [LEFT], [RIGHT], [LEFT, DOWN]$$
$$[RIGHT, DOWN], [DOWN], [DOWN, B], [B]$$

During an episode, agents are rewarded such that the cumulative reward at any point in time is proportional to the horizontal offset from the players initial position. Thus, going right always yields a positive reward, while going left always yields a negative reward. This reward function is consistent with the done condition, which is based on the horizontal offset in the level. The training consisted of 47 levels and the validation consisted of 11 levels.

For the retro contest there are no limitations on the training set but at test time, only 1 million timesteps are allowed. Baseline used a specific variant of DQN, namely Rainbow, PPO, Joint PPO, and JointRainbow which perform well on the ALE and their modifications. For Joint PPO and Joint Rainbow they train a single policy to play every level in the training set. At every gradient step, they average the gradients across levels, ensuring that the policy is trained evenly across the entire training set. Once the joint policy has been trained on all the training levels, they fine-tune it on each test level under the standard evaluation rules.

In our case the network uses a 4-96x96 images and the network consists of 3 depthwise seperable convolutions and two fully connected layers. During training process we use different FiLM parameters for each level but the rest of the network is similar. During the test time a new set of FiLM parameters are learned from scratch and we let all the parameters update. We use Channel Wise FiLM and Projection variants of FiLM. We have investigate the

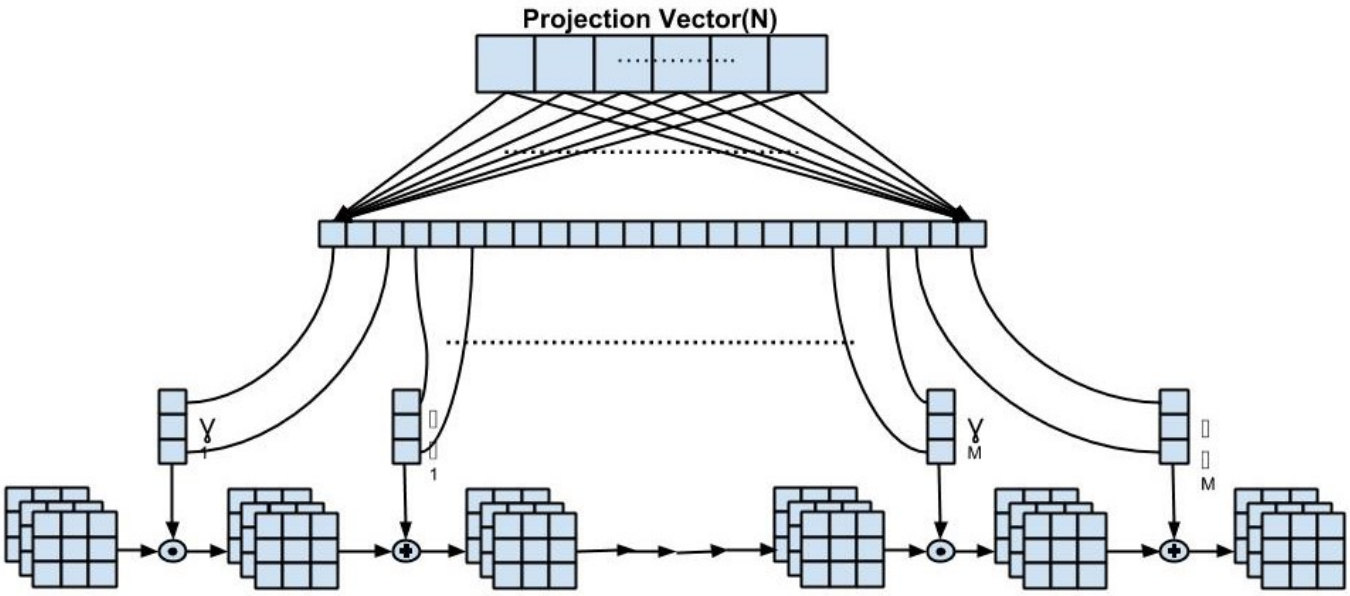


Fig. 8. Projection FiLM

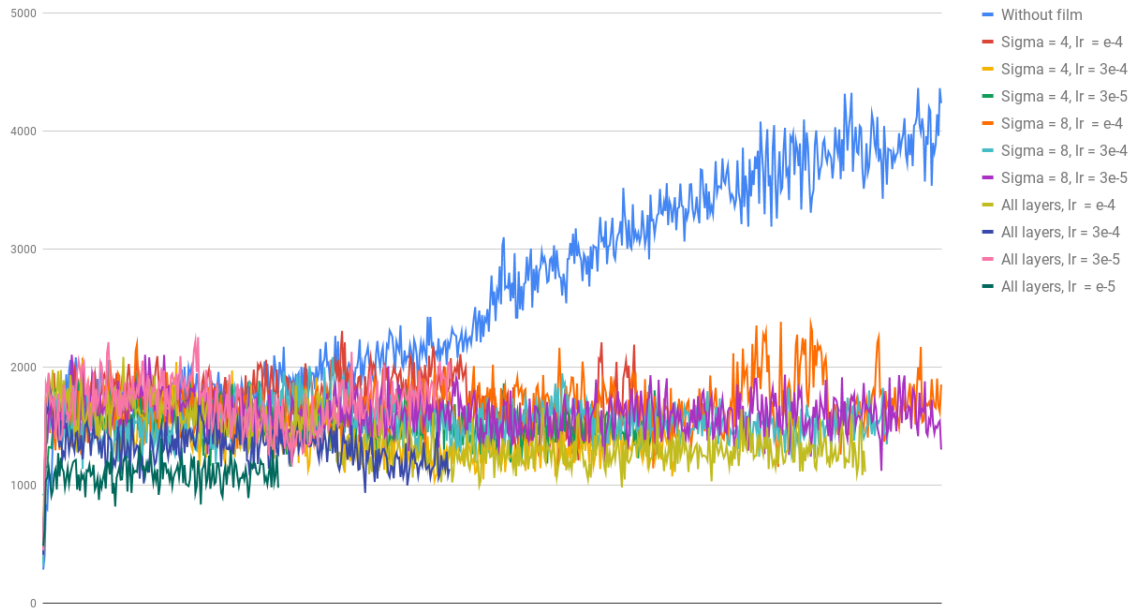


Fig. 9. Experiments in RL

performance with different learning rates: 3e-4, 1e-4, 3e-5. The sigma dimension from which we are projecting has varied from 4 to 8. We also tried to run with FiLM only in 1st layer, 2nd layer and 3rd layer. We investigate the usage of only bias(i.e without gamma in FiLM), and reinitializing the parameters after every few epochs. Reinit Bias uses only bias and reinitializes the bias. Reinit W is same as FiLM parameters but also reinitializes them.

Each of these experiments took about 4.5 days to run on GPU.

## V. RESULTS

### A. Meta Learning

We can see that the Vanilla FiLM has worked very well on 5way 1shot but the performance increase doesn't hold true in 20way 1shot. Hence we use other variations of FiLM so that it works well on both. We can clearly see that as the sigma dimension is increasing from 4 to 24 the performance is becoming better. The adaptive learning rate sometimes gives a boost to the performance and makes it worse in the other cases. The FiLM Spatial + Channels is giving slightly better performance on average. Notice that the same architecture

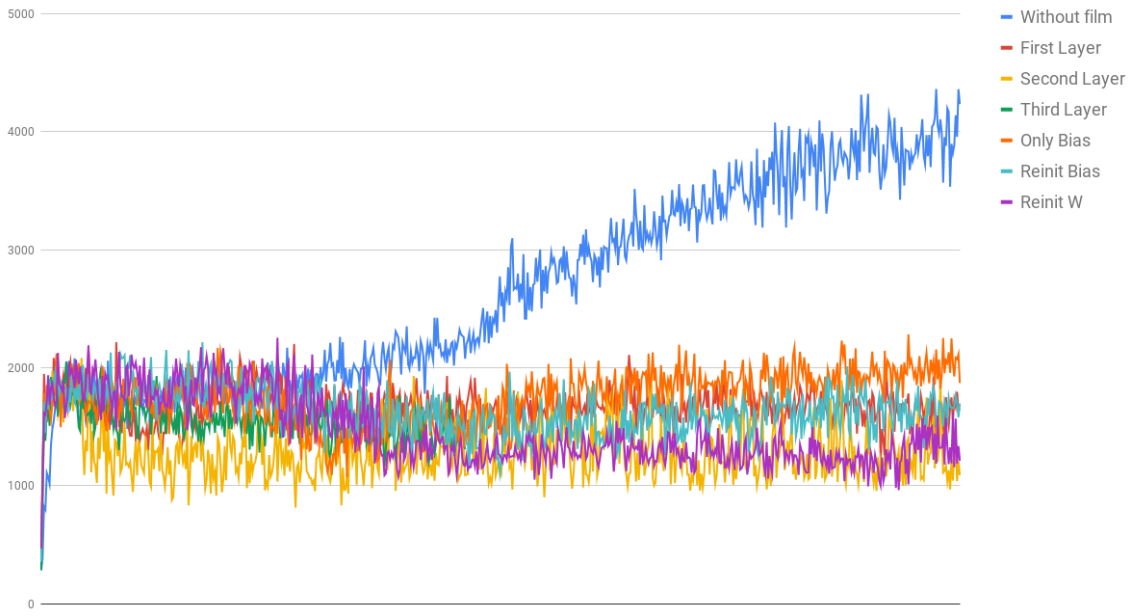


Fig. 10. Experiments in RL

as MAML was used in all these cases. We also use Resnet-18 and in this case the performance is much better when compared to other FiLM based architectures and on par with the state of the art which can be seen in Table I.

| **Method** | **5way-1shot** | **20way-1shot** |
|---|---|---|
| MAML [2] | 98.7 | 98.9 |
| Vanilla FiLM | 98.66 | 6.85 |
| Spatial FiLM | 88.23 | 5.01 |
| Spatial FiLM + Adap LR | 91.73 | 53.08 |
| FiLM spatial + channels | 98.03 | 80.61 |
| sigma = 4 | 48.8 | 6.33 |
| Sigma = 4 + Adap LR | 46.59 | 5.14 |
| sigma = 8 | 24.44 | 6.5 |
| Sigma = 8 + Adap LR | 68.03 | 13.79 |
| sigma = 16 | 68.125 | 5.3125 |
| Sigma = 16 + Adap LR | 78.5 | 16.17 |
| Sigma = 24 | 88.83 | 13.3 |
| Sigma = 24 + Adap LR | 63.33 | 5.04 |
| FilM with Resnet-18 (channels) | 98.39 | 97.36 |

TABLE I
META LEARING RESULTS

### B. Reinforcment Learning

All the models(fig 9, fig 10.) were trained for at-least 3.5e7 timesteps(max x-axis is 7.5e7) if the algorithm is decaying then we stopped during this stage. If it started to increase or stayed the same we ran it until 7.5e7 timesteps. Using just the bias is slightly better than using FiLM but not by a lot as the normal algorithm (Joint PPO) without any level specific parameters started to increase its mean reward around 3.5e7. All of our FiLM trials could not properly increase the reward beyond 2500.

## REFERENCES

[1] Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." arXiv preprint arXiv:1703.03400 (2017).
[2] Zintgraf, Luisa M., et al. "CAML: Fast Context Adaptation via Meta-Learning." arXiv preprint arXiv:1810.03642 (2018).
[3] Li, Zhenguo, et al. "Meta-sgd: Learning to learn quickly for few shot learning." arXiv preprint arXiv:1707.09835 (2017).
[4] Nichol, Alex, et al. "Gotta Learn Fast: A New Benchmark for Generalization in RL." arXiv preprint arXiv:1804.03720 (2018).
[5] Perez, Ethan, et al. "Film: Visual reasoning with a general conditioning layer." arXiv preprint arXiv:1709.07871 (2017).
[6] https://github.com/aborghi/retro_contest_agent
[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal policy optimization algorithms, 2017. eprint: arXiv:1707.06347
[8] Lake, Brenden M, Salakhutdinov, Ruslan, Gross, Jason, and Tenenbaum, Joshua B. One shot learning of simple visual concepts. In Conference of the Cognitive Science Society (CogSci), 2011.