Learning Rate Decay Experiment

Tensorflow provides 3 types of learning optimizers – Momentum Optimizer, Adam Optimizer and RMSProp Optimizer. It also provides different learning rate decay schedule, which are as follows:

1. Step Decay Schedule: Here the learning rate decreases at certain step number passed as hyperparameter. This is the manual way of decreasing learning rate.

For example:

manual\_step\_learning\_rate {

initial\_learning\_rate: 0.005

schedule {

step: 1500

learning\_rate: .00002

}

schedule {

step: 3000

learning\_rate: .000002

}

}

Here intital\_learning\_rate is 0.005. This will decrease to 0.00002 at 1500th step and again drop to 0.000002 at 3000th step.

1. Cosine Decay Schedule: If your data set is highly differentiated, you can suffer from a sort of "early over-fitting". If your shuffled data happens to include a cluster of related, strongly-featured observations, your model's initial training can skew badly toward those features -- or worse, toward incidental features that aren't truly related to the topic at all. Warm-up is a way to reduce the primacy effect of the early training examples. Without it, you may need to run a few extra epochs to get the convergence desired, as the model un-trains those early superstitions. The learning rate is increased linearly over the warm-up period. If the target learning rate is p and the warm-up period is n, then the first batch iteration uses 1\*p/n for its learning rate; the second uses 2\*p/n, and so on: iteration i uses i\*p/n, until we hit the nominal rate at iteration n.

Once the learning rate has reached learning\_rate\_base from warmup\_learning\_rate in warmup\_steps, then it begins decay using the cosine function.

For Example:

cosine\_decay\_learning\_rate {

learning\_rate\_base: 0.005

total\_steps: 10000

warmup\_learning\_rate: 0.0

warmup\_steps: 2000

}

The learning rate will start increasing from 0.0 (warmup\_learning\_rate) and reach 0.005 (learning\_rate\_base) in 2000 (warmup\_steps) steps. Then it will decreasing according to cosine function.

1. Exponential Decay Schedule: This function applies an exponential decay function to a provided initial learning rate. It requires a global\_step value to compute the decayed learning rate. You can just pass a TensorFlow variable that you increment at each training step. The function returns the decayed learning rate. It is computed as:

decayed\_learning\_rate = learning\_rate \* decay\_rate ^ (global\_step / decay\_steps).

For example:

exponential\_decay\_learning\_rate {

initial\_learning\_rate: 0.005

decay\_steps: 1000

decay\_factor: 0.95

}

Here the initial\_learning\_rate will decrease every 1000 (decay\_steps) steps with a base of 0.96

Note: Adaptive learning rate methods (Adam and RMSProp Optimizers) demonstrate better performance than learning rate schedules, and they require much less effort in hyperparamater settings.

(source: <https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1>)

In this experiment, I have used learning rate decay schedules with momentum optimizer only. Adam and RMSProp have constant learning rate schedule.

**Legend**:

Train1/Orange: Momentum optimizer with step decay learning rate schedule

Train2/Blue: Momentum optimizer with cosine decay learning rate schedule

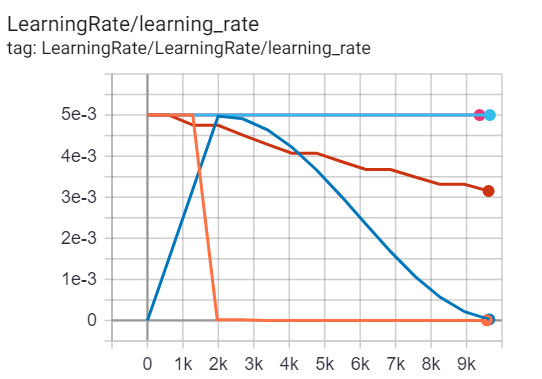
Train3/Maroon: Momentum optimizer with exponential learning rate schedule

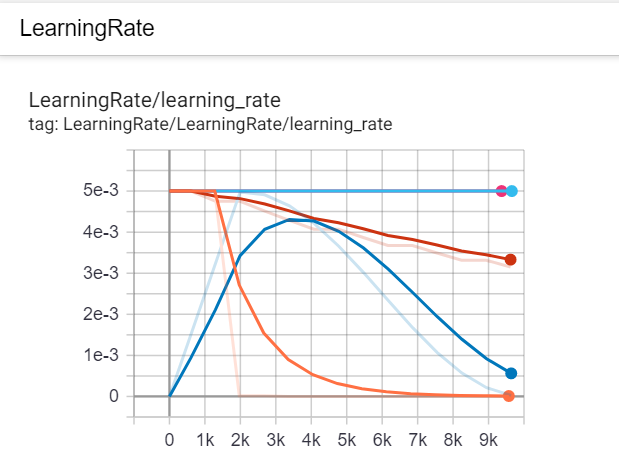
Train4/Light blue: Adam optimizer with constant learning rate schedule

Train5/Pink: RMSProp optimizer with constant learning rate schedule

**Learning Rate**

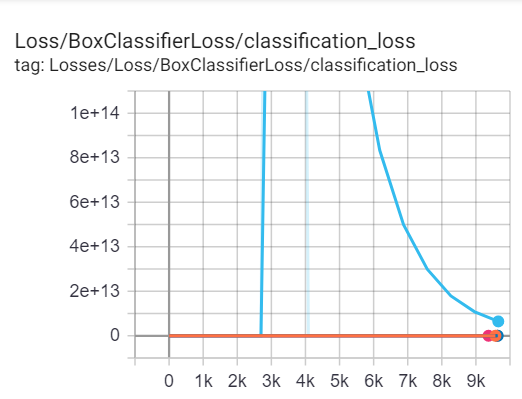
**Without smoothing:**



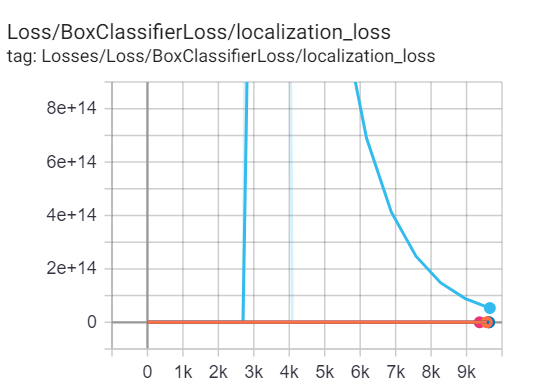
**With smoothing**:

**Loss**

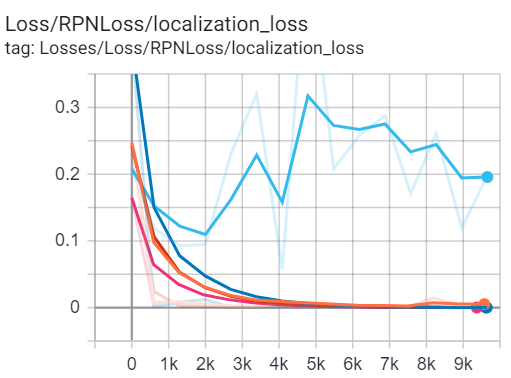
1. **Classifier/Classification loss**



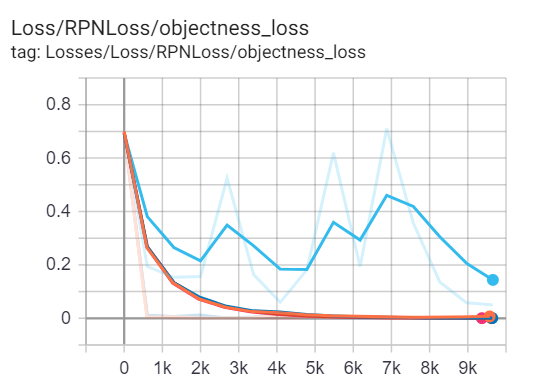
1. **Classifier/Localization loss**



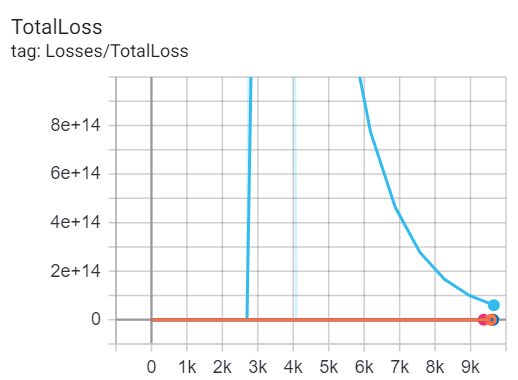
1. **RPN/Localization loss**



1. **RPN/Objectness Loss**



1. **Total loss**

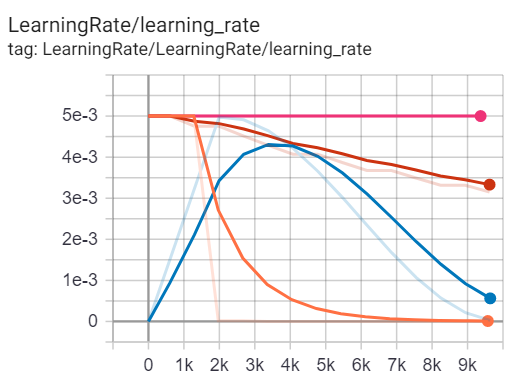




**Observation: Adam optimizer is not performing well given the dataset and Faster RCNN Algorithm. Other optimizer’s performance is comparable, with the best being momentum optimizer with cosine decay learning rate schedule.**

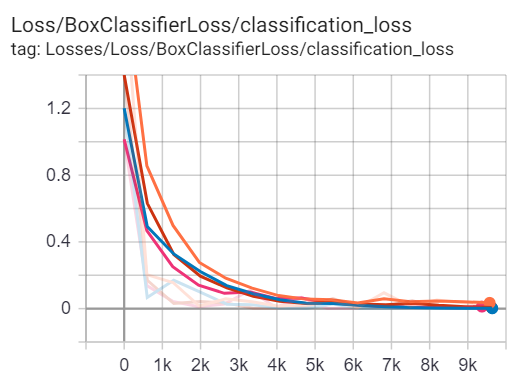
Without Adam Optimizer

**Learning Rate**

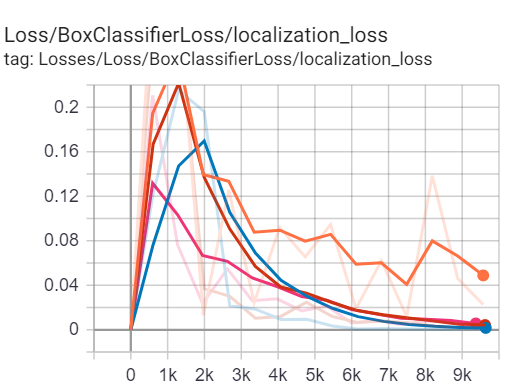


**Loss**

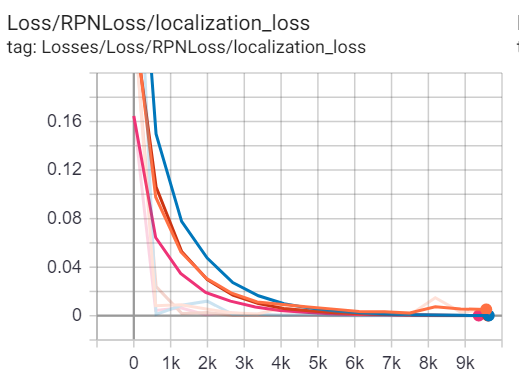
1. **Classisfier/Classification Loss**



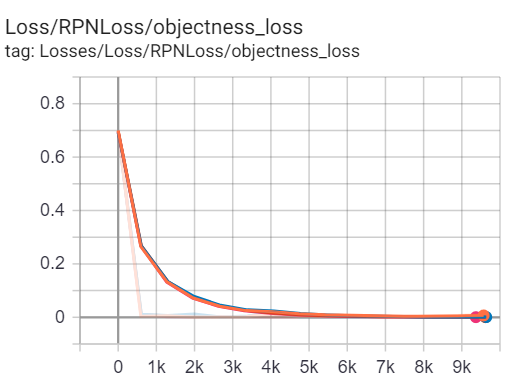
1. **Classifier/Localization Loss**



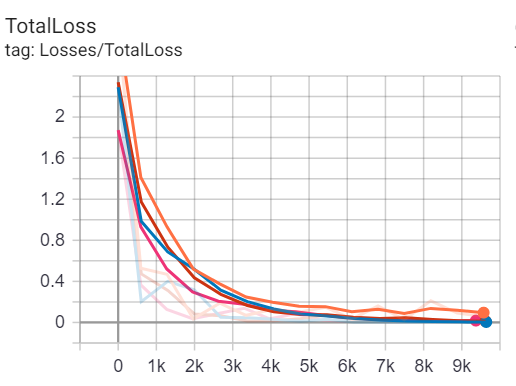
1. **RPN/Localization Loss**

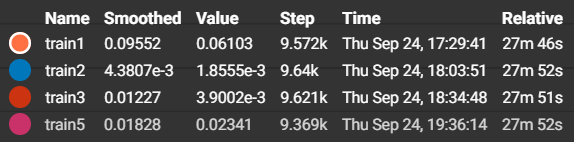


1. **RPN/Objectness Loss**



1. **Total Loss**





**Observation: Momentum Optimizer with step decay learning rate schedule has higher total loss as compared to others.**

References:

* <https://medium.com/@scorrea92/cosine-learning-rate-decay-e8b50aa455b>
* <https://stackoverflow.com/questions/55933867/what-does-learning-rate-warm-up-mean>
* <https://github.com/tensorflow/models/blob/266026c9a70de2de80cd37ebe83aa1a046b95190/research/object_detection/protos/optimizer.proto>
* <https://github.com/tensorflow/models/blob/master/research/object_detection/utils/learning_schedules.py>
* <https://docs.w3cub.com/tensorflow~python/tf/train/cosine_decay/>
* <https://www.dlology.com/blog/bag-of-tricks-for-image-classification-with-convolutional-neural-networks-in-keras/>
* <https://www.tensorflow.org/api_docs/python/tf/compat/v1/train/cosine_decay>
* <https://docs.w3cub.com/tensorflow~python/tf/train/exponential_decay/>
* https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1