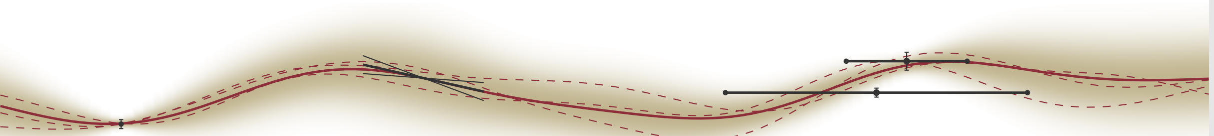
Bachelor Thesis

Investigating Probabilistic Preconditioning on Deep Learning using DeepOBS

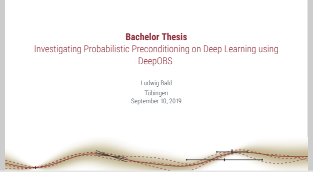
Ludwig Bald

Tübingen

September 10, 2019



2019-09-10



- Introduce myself, I study Cognitive Science
- I chose this topic because I wanted to get to know deep learning from the inside. I had never done Deep Learning before and wanted to learn it hands-on.
- In this talk I will talk about the science, but also about the process I used.
- If you have questions during the talk, ask them right away!

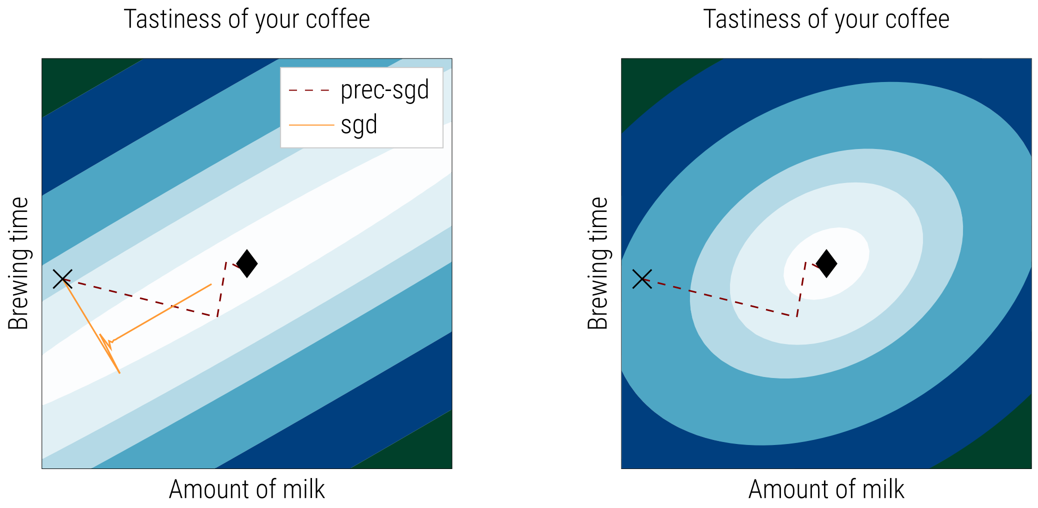
Definition [1]

"A computer program is said to learn from **experience E** with respect to some class of **tasks T** and **performance measure P** if its performance at tasks in T, as measured by P, improves with experience E"

- Special case in Deep Learning: Finding the **optimal point** in **parameter space**
- The **Loss function** $\mathcal{L}(w)$ tells us how bad a specific parametrization w is. For example, the model's predictions for all possible inputs are compared to the desired outputs.

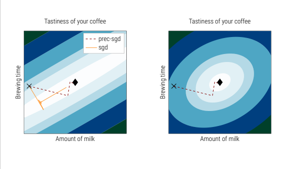
[1] Tom Mitchell, "Machine Learning", 1997

- Most of you will have seen this definition before.
- As an example, have a look at this 2-parametrical problem



2019-09-10

└ Background



Gradient Descent

Optimization

Gradient Descent uses knowledge of the **gradient** at a point in parameter space to take an update step:

$$w_{i+1} = w_i - \alpha \cdot \nabla \mathcal{L}(w_i)$$

where α is the learning rate.

2019-09-10

Gradient Descent

Optimization

Gradient Descent uses knowledge of the **gradient** at a point in parameter space to take an update step:

$$w_{i+1} = w_i - \alpha \cdot \nabla \mathcal{L}(w_i)$$

where α is the learning rate.

Stochastic Gradient Descent

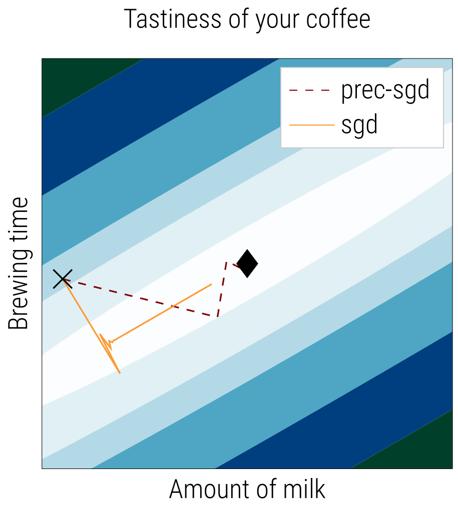
- In real life, we have Big Data. The true $\nabla\mathcal{L}(w)$ is expensive to compute.
- To speed things up, we compute the noisy estimate $\hat{\mathcal{L}}(w_i)$ on a minibatch of for example 128 data points.

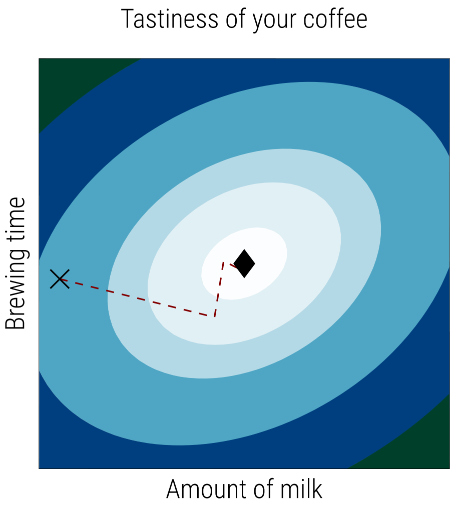
The update rule still looks the same:

$$w_{i+1} = w_i - \alpha \cdot \nabla\hat{\mathcal{L}}(w_i)$$

where α is the learning rate.

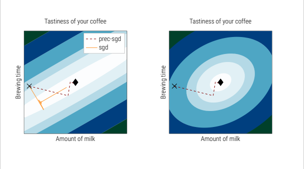
Stochastic Gradient Descent
Optimization
In real life, we have Big Data. The true $\nabla\mathcal{L}(w)$ is expensive to compute.
To speed things up, we compute the noisy estimate $\hat{\mathcal{L}}(w_i)$ on a minibatch of for example 128 data points.
The update rule still looks the same:
$w_{i+1} = w_i - \alpha \cdot \nabla\hat{\mathcal{L}}(w_i)$
where α is the learning rate.





2019-09-10

Background



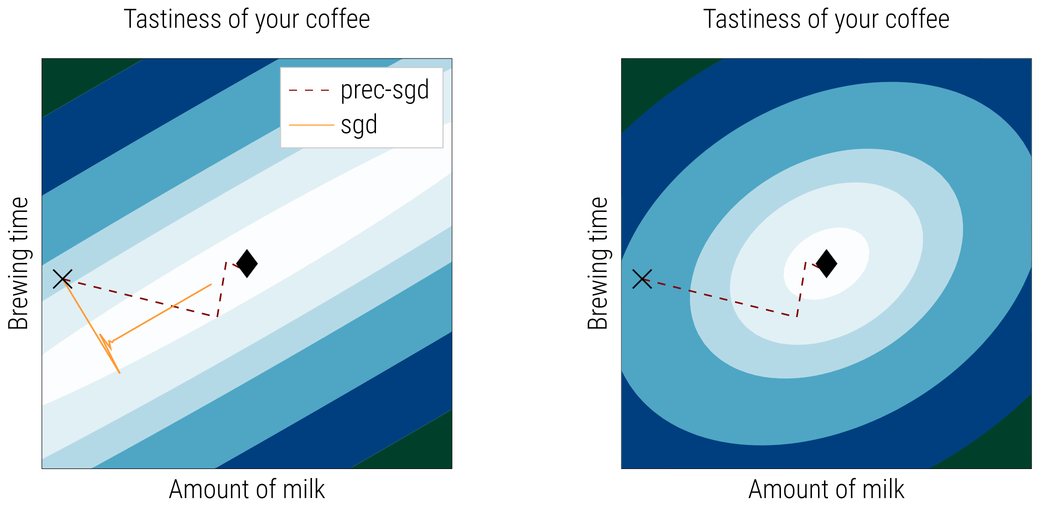
Preconditioning

The condition number of the Hessian

- The performance of (S)GD depends heavily on the shape of the loss landscape
- The **condition number** is defined as

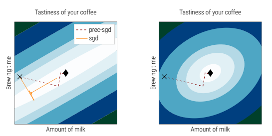
$$\kappa = \frac{\lambda_n}{\lambda_1} > 1$$

  where λ_n, λ_1 are the largest/smallest eigenvalues of the Hessian $\nabla\nabla\mathcal{L}(w)$
- For larger κ, (S)GD can converge slower.
- The condition number can be changed by carefully rescaling the gradient before taking the optimization step



2019-09-10

└Background



Probabilistic Preconditioning

by Filip & Philipp, 2019

In the stochastic (minibatched) setting and while only having access to **Hessian-vector products**, it isn't obvious how to construct the preconditioner. This is the method I'm testing:

1. Empirically construct a prior for the multivariate Gaussian distribution and set the learning rate for SGD
2. Gather observations and update the posterior estimate for the Hessian, using Bayes
3. Create a rank-2 approximation of the Preconditioner
4. apply the preconditioner at every step and do SGD

Probabilistic Preconditioning
by Filip & Philipp, 2019
In the stochastic (minibatched) setting and while only having access to **Hessian-vector products**, it isn't obvious how to construct the preconditioner. This is the method I'm testing:
1. Empirically construct a prior for the multivariate Gaussian distribution and set the learning rate for SGD
2. Gather observations and update the posterior estimate for the Hessian, using Bayes
3. Create a rank-2 approximation of the Preconditioner
4. apply the preconditioner at every step and do SGD

If I'm grossly misrepresenting the algorithm, please correct me now! For an exact description check out the paper

Deep learning

Neural nets

For the purposes of this talk, a neural net is a model

- with many ($>$ hundreds of thousands) parameters, weights $\boldsymbol{w}$
- with an available noisy gradient $\nabla\hat{\mathcal{L}}(\boldsymbol{w}_0)$, which was obtained by backpropagation

Deep learning
Neural nets
For the purposes of this talk, a neural net is a model
- with many ($>$ hundreds of thousands) parameters, weights $\boldsymbol{w}$
- with an available noisy gradient $\nabla\hat{\mathcal{L}}(\boldsymbol{w}_0)$, which was obtained by backpropagation

Evaluating an Optimizer

Empirically

This is a hard problem in itself! How do you chose:

- Data Set
- Batch Size
- Model architecture
- Hyperparameter tuning method
- Measure of success

Comparing results between papers is very hard.

Evaluating an Optimizer
Empirically
This is a hard problem in itself! How do you chose:
- Data Set
- Batch Size
- Model architecture
- Hyperparameter tuning method
- Measure of success
Comparing results between papers is very hard.

Now that we have roughly defined the algorithm, how do we test it?

DeepOBS

by Frank & Aaron

- A library for Tensorflow and Pytorch
- In order to test an optimizer, you have to specify only
  - The optimizer class
  - The hyperparameters of my optimizer
  - One of the provided testproblems
- DeepOBS then returns a json file
- And automatically generates figures

DeepOBS
by Frank & Aaron
- A library for Tensorflow and Pytorch
- In order to test an optimizer, you have to specify only
  - The optimizer class
  - The hyperparameters of my optimizer
  - One of the provided testproblems
- DeepOBS then returns a json file
- And automatically generates figures

Implementation Details

The class Preconditioner

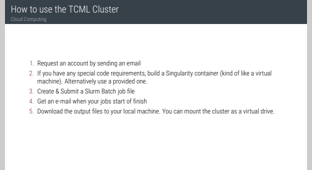
```
Preconditioner(params, est_rank=2, num_observations=5, prior_iterations=10,
               weight_decay=0, lr=None,
               optim_class=torch.optim.SGD, **optim_hyperparams)
start_estimate()
step()
get_log()
```


Implementation Details
The class Preconditioner
Preconditioner(params, est_rank=2, num_observations=5, prior_iterations=10,
weight_decay=0, lr=None,
optim_class=torch.optim.SGD, **optim_hyperparams)
start_estimate()
step()
get_log()

How to use the TCML Cluster

Cloud Computing

1. Request an account by sending an email
2. If you have any special code requirements, build a Singularity container (kind of like a virtual machine). Alternatively use a provided one.
3. Create & Submit a Slurm Batch job file
4. Get an e-mail when your jobs start of finish
5. Download the output files to your local machine. You can mount the cluster as a virtual drive.



Experiments

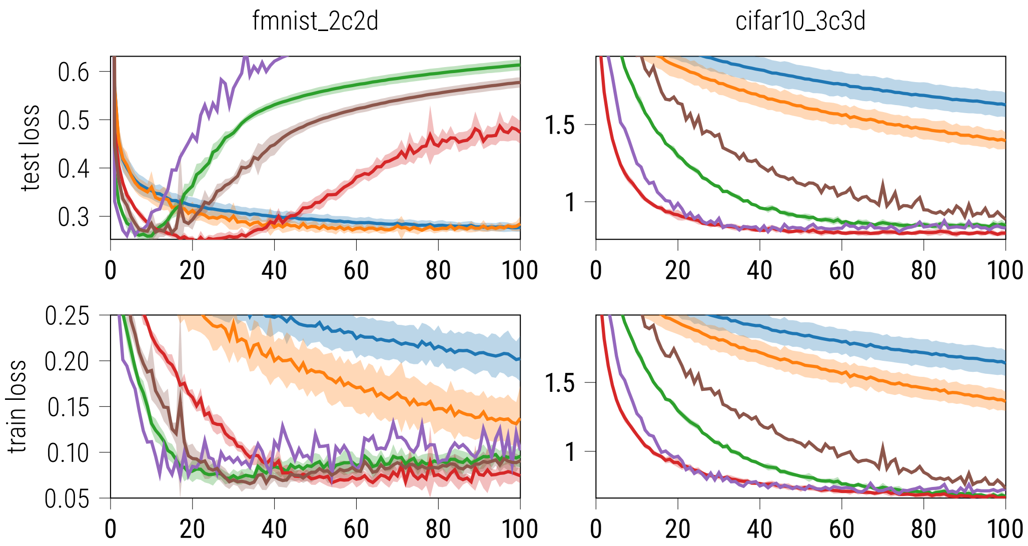
Overview

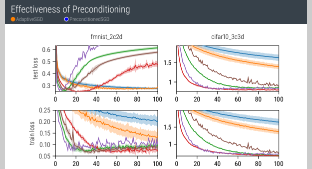
- Effectiveness of Preconditioning
- Computational Complexity
- Stability
- Learning Rate sensitivity

Experiments
Overview
- Effectiveness of Preconditioning
- Computational Complexity
- Stability
- Learning Rate sensitivity

Effectiveness of Preconditioning

AdaptiveSGD PreconditionedSGD

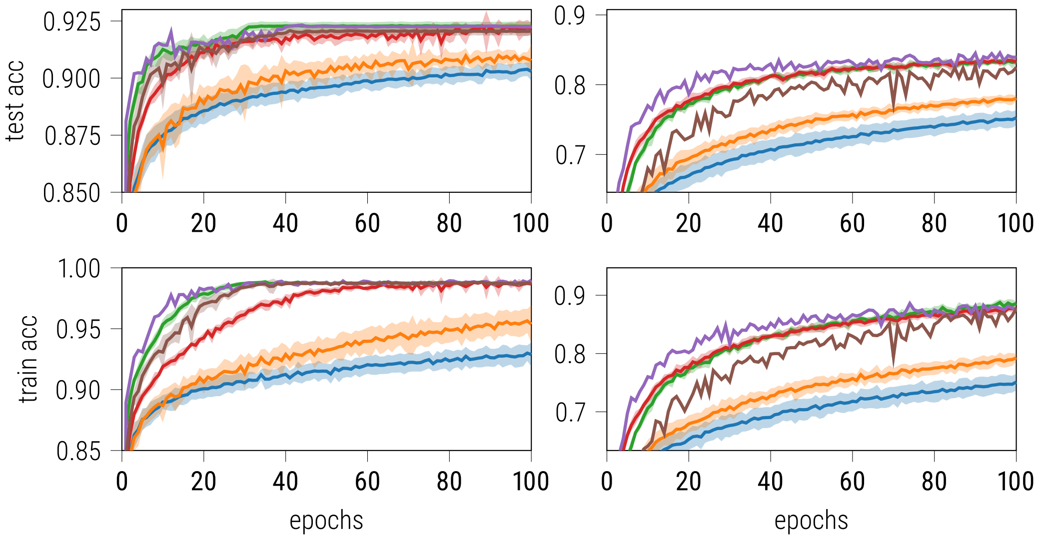




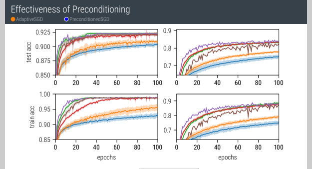
- - Show & explain the figure exp_preconditioning, It's created by DeepOBS
- - Different Optimizers on two testproblems. Blue is the PreconditionedSGD and Orange is the AdaptiveSGD.
- - Explain the Optimizers
- - (Explain Testproblems) Only Convolutional nets!
- - Both variants perform worse than other widely used algorithms.
- - They converge much slower than the other algorithms, which seem to reach an optimum after 40 epochs, while the two don't converge even after 100 epochs
- - Maybe mention the overfitting going on.
- - Main conclusion: The algorithm does not perform better than others.

Effectiveness of Preconditioning

AdaptiveSGD   PreconditionedSGD



2019-09-10



Discussion: Effectiveness

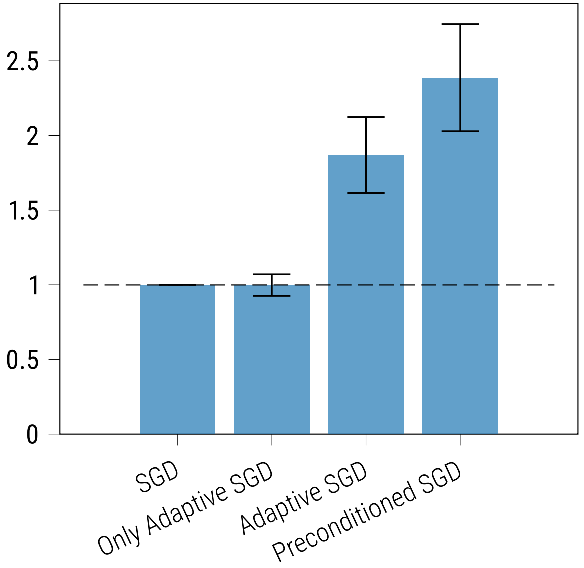
Why might it be worse than plain SGD?

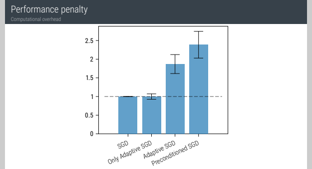
- It’s very noisy: Noise from the Hessian is amplified through the whole epoch.
- The constructed learning rate is not optimal
- PreconditionedSGD is worse than AdaptiveSGD, so the Preconditioning makes things worse
- The preconditioner has only rank 2, while there might be thousands large eigenvalues (usually 10%)
- The other optimizers are exhaustively tuned
- The other optimizers use more data for actual parameter updates: 1920/50.000 images per epoch are only used for the Hessian

Discussion: Effectiveness
Why might it be worse than plain SGD?
- It’s very noisy: Noise from the Hessian is amplified through the whole epoch.
- The constructed learning rate is not optimal
- PreconditionedSGD is worse than AdaptiveSGD, so the Preconditioning makes things worse
- The preconditioner has only rank 2, while there might be thousands large eigenvalues (usually 10%)
- The other optimizers are exhaustively tuned
- The other optimizers use more data for actual parameter updates: 1920/50.000 images per epoch are only used for the Hessian

Performance penalty

Computational overhead

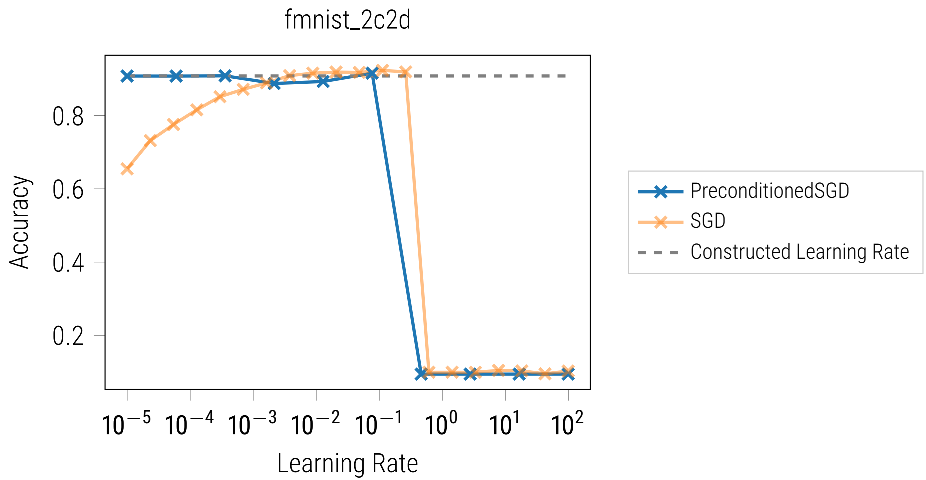


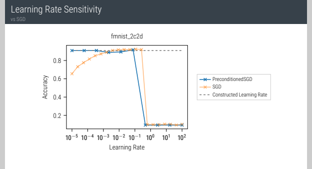


- - Running on mnist_mlp with a batch size of 128.
- - Shown Optimizers are
- - SGD at 100
- - PreconditionedSGD, AdaptiveSGD, OnlyAdaptiveSGD
- - They add a large performance penalty, but that would depend on the batch size.

Learning Rate Sensitivity

vs SGD





- - Seeing that the algorithm is stable for an automatically constructed learning rate, I wanted to see if there are effects on success measures.
- - Describing the plot
- - Dropoff of SGD for small learning rates, plateau, cliff for large learning rates.
- - Usually, on this testproblem the algorithm would estimate a learning rate of around 0.02 and then slowly decay it over epochs.
- - This means we should let the algorithm chose the learning rate automatically

Conclusion/Final Remarks

Conclusion/Final Remarks

end of presentation