,

An Empirical Study of Second-Order Optimization Methods for Deep Learning Applications

Vishnu Dutt Sharma VISHNUSHARMA@VT.EDU
Department of Electrical & Computer Engineering
Virginia Polytechnic Institute & State University
Blacksburg, VA 24060, USA

May 13, 2020

Abstract

This work studies the performance of two second-order optimizers, Limited Memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) and Kronecker-Factored Approximate Curvature (K-FAC), against a popular first-order optimizer, Stochastic Gradient Descent for two Deep Learning (DL) architectures, Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN). We analyze the time and steps required to converge for each optimizer, the effect of using dropouts, and the generalization error for each model and provide recommendations for appropriate conditions for using these optimizers.
Keywords: SGD, L-BFGS, K-FAC, MLP, CNN, Generalization, Dropout

1. Problem Introduction

Deep Learning (DL) techniques have been gaining popularity for the past few years, owing to the ability of such techniques to fit highly non-linear functions without any need for explicit feature engineering (Perrault et al. (2019)). The application of these techniques spans a wide variety of areas like speech recognition, biomedical imaging, recommender systems, etc. While optimization methods are at the core of these techniques, a huge number of applications rely on variations of steepest descent. The limited adoption of the 2^{nd} order optimizers is often attributed to the computational and memory requirement.

In this work, we observe the time to convergences of second-order optimizers (K-FAC, L-BFGS) with CNN and MLP for the task of image classification and with MLPs of difference depth for the task of regression. For image classification, we also see what effect the removal of pooling on CNN has on the convergence of these optimizers. Generalization, which is the ability of a trained model to perform well on new unseen data, is often a key requirement for DL techniques. Due to their complex nature, 2^{nd} order optimizers may cause the model to overfit (or not generalize well). We thus also check the generalization error for the best performing model, and if using a regularizer alleviates it.

This paper has been organized as follows: in Section 2 we discuss the previous work on using the 2^{nd} order optimizers for Deep Learning. Section 3 describes the setup for our experiments. In Section 4, we present our observations for various settings. In Section 5, we present our conclusion, recommendations and future work to be pursued in this direction.

2. Previous Work

While the number of deep learning applications has grown significantly over the year, most of these applications leverage variations of gradient descent. Efficient optimization, particularly in terms of time, is an active area of research in this context, and accordingly, augmentations like momentum have been proposed. Due to the higher effectiveness of deep learning models with larger data size, stochastic gradient descent(SGD) is the most basic variation often used for training these models. While Nesterov (Nesterov (1983)) first proposed a constant velocity factor, over time gradient-dependent momentum methods like RMSprop (Hinton et al. (2012)), Adagrad (Duchi et al. (2011)), and ADAMKingma and Ba (2014). Ruder (2016) provides an overview of various gradient descent optimization techniques. The popularity of these techniques is evident from their inclusion in the standard optimization toolkits of various deep learning frameworks.

Second-order optimization has proven to be a promising technique in general. However, these techniques have higher computational and memory requirements, which limit the widespread adoption of these methods. As deep learning objective functions are typically nonconvex, with the possibility of multiple saddle points, it poses a challenge for Newton's method. The use of large datasets is another typical trait of the DL methods. For such datasets, the speed-up brought by stochastic gradient descent is much better than the speed-up brought by these methods over the regular gradient descent (LeCun et al. (1998)). Hinton et al. (2012) suggests using such methods for small datasets or bigger datasets without much redundancy. Augmented methods like sub-sampling (Roosta-Khorasani and Mahoney (2016)) in Newton Methods and stochastic Quasi-Newton methods (Schraudolph et al. (2007), Zhou et al. (2017)) have been proposed to make the original second-order methods more suitable for deep learning applications.

Conjugate Gradient (CG) methods work better in terms of computational complexity than the aforementioned second-order techniques. CG methods like Broyden–Fletcher–Goldfarb–Shanno (BFGS) alleviate the computational requirements but still need large memory to hold inverse Hessian. Limited Memory BFGS (or LBFGS) method (Liu and Nocedal (1989)) decreases memory requirement as well and thus is often part of the standard optimization toolkits of the deep learning frameworks. However, the issue of the nonconvex nature of deep learning objective function with saddle point remains a critical problem to be addressed. Le et al. (2011) shows that L-BFGS can achieve faster convergence for Autoencoders (unsupervised learning) and CNNs (supervised learning), even achieving better accuracy for CNNs. However, they present results for CNN with a multi-GPU setting. For autoencoders, they show that SGD with line search procedures can make it converge faster than L-BFGS. Other methods developed in a similar direction make use of Hessian-free (or truncated-Newton) optimization (Martens (2010) Martens and Grosse (2015)), which have shown to perform well both in terms of time to converge and training error but have shown to overfit the data. Xu et al. (2017) provides an overview of second-order methods for non-convex machine learning.

3. Methodology

We compare the time-to-converge of the optimization techniques for the tasks of image classification and regression using two DL architectures on multiple datasets. We also observe the generalization error of the models and the effect of a regularizer on it. We provide details of these components of our setup in the following subsections.

3.1 Optimizers

We focus on three off-the-shelf optimizers from PyTorch (Paszke et al. (2019)) framework:

1. **SGD**: SGD is a 1^{st} order line-search methos, using only a subset of data at once (also called *mini-batch*). SGD and its variations are used widely for DL applications. We use a SGD with a momentum of 0.9 as our baseline.

2. **L-BFGS**: It is a limited-memory version of the BFGS algorithm, which is a quasi-Newton method. The limited memory implementation makes it easier to use as DL models are often memory-intensive due to the huge number of trainable parameters involved. One disadvantage of the implementation we used was that it can only be used with full data i.e. it does not support mini-batches. Thus, it can't be used with big datasets.

3. **K-FAC**: K-FAC is a Hessian-Free method, which uses an approximation of the neural network's Fisher information matrix and has been shown to work well as compared to SGD.

3.2 Deep Learning Architectures

We leverage the following DL architectures for our experiments:

1. **MLP**: MLP contains multiple layers of neural-networks each containing non-linear activation. MLPs are often used as building blocks or in conjugation with the other DL architectures like autoencoders, Recurrent Neural Networks (RNNs), etc. and thus we chose to evaluate optimizers on it for this work.

2. **CNN**: CNN uses the idea of convolution from image processing and is often used for computer vision problems. Due to convolution it uses shared weights in 2 or 3-dimensional windows (also called *kernels*) for processing data. It has also been shown to perform well for the task of sequence prediction (Chen and Wu (2017)).

Closely tied to CNNs is the idea of pooling. Pooling helps in dimensionality reduction by processing features in a small spatial window, similar to convolution. Max-pooling is the most common pooling method, which outputs the feature with the highest value in the processing window. Using the max operation introduces a non-linearity that is non-differentiable in the traditional sense. Thus, we also run experiments on a version of CNN which replace max-pooling with other convolution layers, in a way such that the number of parameters is similar. Our general schema for CNN has two convolutions layers, each followed by pooling and ReLU activation. After this we flatten the output and feed to a 3-layered MLP. In an network without polling layers, we replace pooling layers with

convolution layers. For DNN based classifiers, we flatten the input first and then feed it to 5 layered MLP.

MLP is also sometimes used as classifiers, and can also be considered as a simplification of CNN architecture (an MLP can be seen as a convolution layer with kernel size same as input image). Thus, we also use MLPs with a similar number of layers as CNNs for the task of image classification. We also used MLPs for the task of regression, where we change the number of layers in MLP to analyze the effect of the complexity of the model on convergence.

Apart from high accuracy, the generalization of models is also an important requirement for the deep learning models. Generalization indicates how well a trained model would perform on unseen data, which is of paramount importance for applications where the test data distribution may differ from that of training data e.g. stock market. We quantify the *generalization error* as the difference between the metrics calculated on training and test data. We also study the effect of regularization on this generalization error. For this purpose, we use dropout, which is often used for regularizing neural networks. It randomly removes the connections of the neural networks to force the model to learn general features.

3.3 Datasets

In our experiments, we use datasets that are often used for benchmarking DL models and are available with the standard data toolkits of the DL frameworks. For image classification we use the following datasets:

1. **CIFAR-10**: CIFAR-10 (Krizhevsky (2009)) contains colored images of size 32×32, with 10 classes like car, dog, horse, etc. as the target. It is an example is input where color is an important feature to identify the objects well.

2. **MNIST**: MNIST (LeCun et al. (2010)) contains grayscale images of handwritten digits (0-9) of size 28. The targets are the corresponding digits.

3. **Fashion-MNIST**: Fashion-MNIST (Xiao et al. (2017)) is similar to MNIST, but it contains images of 10 clothing articles instead of digits. This dataset is often used in place of MNIST for benchmarking DL models.

For the regression task, we use two datasets:

1. **Boston Housing**: Boston Housing dataset (Harrison Jr and Rubinfeld (1978)) contains information collected by the U.S Census Service concerning housing in the area of Boston Mass. it contains 13 numerical feature for the housing unit and the target is the median housing price.

2. **Breast Cancer**: Breast Cancer dataset ((Mangasarian et al., 1995)) contains 30 features from calculated fro digitized images of a fine needle aspirate (FNA) of a breast mass. This dataset is used for logistic regression over two classes indicating whether the tumor is malignant or benign.

For each dataset, we train the model by using ReLU activation in all units, and fixed batch size (32 for classification and 8 for regression). However, when using L-BFGS we

observed that it fails for mini-batches, and thus the numbers are reported after using full data for each step/epoch with this optimizer. For classification, we use Cross-Entropy as the loss, whereas Mean-Squared Error (MSE) is used as the loss for regression. To gauge how well a model performs we use accuracy and MSE for classification and MSE respectively. All the classification models were trained and tested on GPU (Tesla P-100), but for regression, we used CPU due to resource constraints.

For each setup, we tried few learning rates ($\{10^{-2}, 10^{-3}, 10^{-4}\}$ and 1, 10^{-5} in some cases), to find the best performing (highest accuracy/lowest MSE) model. Epochs/Steps (number of training iterations over the whole data) are often also considered as hyperparameters, but we eliminated the need to tune it by cross-validation models again validation datasets after each epoch and stopping the training if we don't observe improvement in the loss for 5 consecutive epochs. To avoid long runs, we set the maximum number of epochs to 100.

4. Results

To compare the time-to-converge of optimizers for each dataset, we first select models that achieve accuracy/MSE within 3% of the best accuracy/MSE for the corresponding optimizer. For comparison, we chose the model which takes the least time among those models. This was we get the fastest converging models with comparable accuracy/MSE for each optimizer over each data. We now describe our observations.

4.1 Classification Task

For image classification using CNN, the best models for each optimizer achieved $\sim 62\%$ accuracy for CIFAR-10 and $\sim 99\%$ accuracy over MNIST and Fashion-MNIST datasets. The time-to-converge for these are shown in Figure 1. K-FAC works either better or on par with SGD. It converges fastest for grayscale images. L-BFGS did not converge for colored images in 100 epochs even though it achieved appreciable performance. It did converge for grayscale images. but took around twice and thrice the convergence time of SGD for MNIST and Fashion-MNIST respectively. It suggested that K-FAC with CNN is a good choice for image classification.

When we look at the steps-to-converge, as shown in Figure 2, we observe that 2^{nd} order optimizers took a lesser number of steps for grayscale images. K-FAC works exceptionally well in this aspect for all datasets.

To remove the potential non-differentiable non-linearity due to max-pooling, we replace these layers with convolution layers such that the number of parameters is similar. SGD converges faster with this change, but 2^{nd} order optimizers do not respond well. This degradation in convergence is particularly worse for L-BFGS.

If we use MLP for image classification by keeping the number of layers the same (to have similar levels of features), it increases the number of training parameters, as shown in Table 1. As shown in Figure 3, SGD converged faster with MLP than with CNN. L-BFGS performed better with MLP for this task than with CNN. K-FAC showed the opposite trend here. However, SGD is better than both 2^{nd} order optimizers when MLP is being used.

To study the generalization of the models, we use the difference between accuracy over the training and test data as the *generalization error*. Low generalization error is desirable

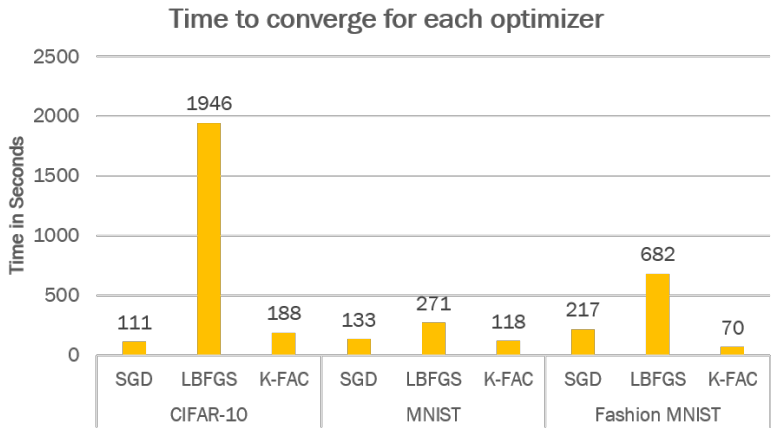

Figure 1: Time-to-converge of best performing classification models over each dataset

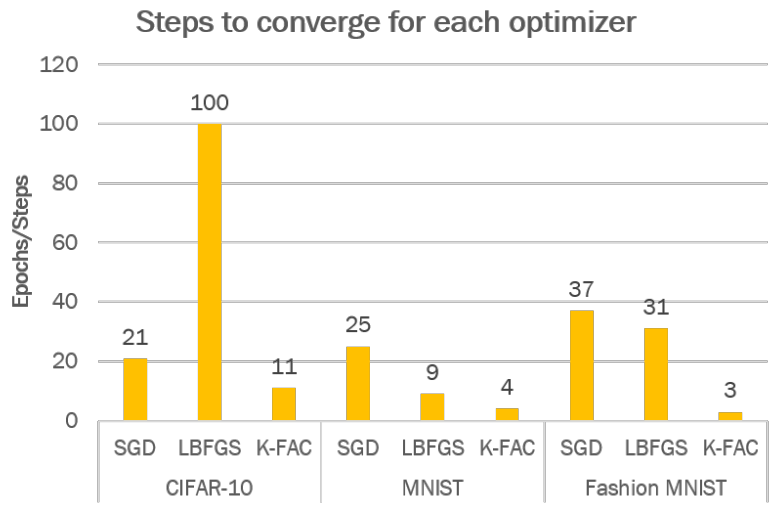

Figure 2: Steps-to-converge of best performing classification models over each dataset

in a model as it indicates that the model performs well on new data (or generalizes well). Figure 4 displays our observations in this setting. We observe that K-FAC, which converged faster than SGD, also has a low generalization error than SGD on grayscale images. L-BFGS gives varied results thus we were unable to reach a conclusion for it. When we introduce dropout regularizer, SGD showed improvement across all datasets and seems like the best choice when a low generalization error is desired. The 2^{nd} order optimizer did not provide consistent results with dropouts.

4.2 Regression Task

For regression with MLP, we try MLPs with different numbers of layers and thus different complexity. It is also indicated by the number of trainable parameters in each model as

Architecture	CIFAR-10	MNIST	Fashion-MNIST
CNN (Normal, with pooing)	62006	44426	44426
CNN (no pooling)	62006	54100	54100
MLP	1647946	476490	476490

Table 1: Number of trainable parameters for classification models

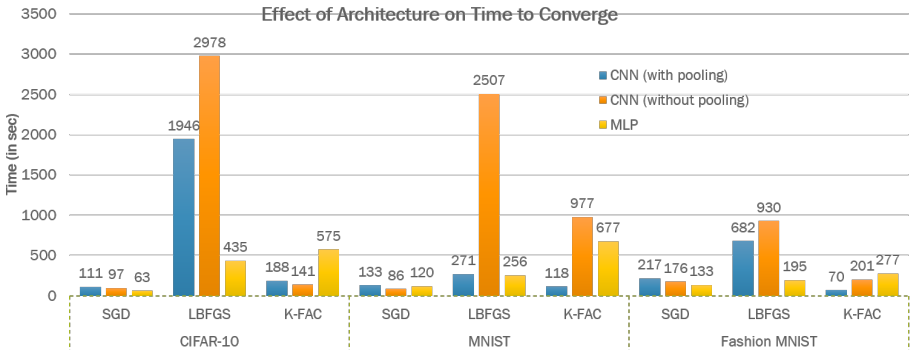

Figure 3: Time-to-converge for different architectures for classification task

shown in Table 2. Figure 5 shows the time-to-converge for each model. For the smaller model, L-BFGS works better than K-FAC. For the 3 layer model, it takes slightly longer to converge than SGD. It also converged in much fewer steps for 3 layer model than SGD and K-FAC as shown in Figure 6. As we increase the number of layers (and hence the number of parameters) time-to-converge increases very fast for L-BFGS, thus it is more suitable for small models. K-FAC isn't as effective here as classification, even in steps required to converge. The best choice for a deeper model seems to be SGD in this setting.

Number of Layers	Boston Housing	Breast Cancer
3	369	641
4	1121	1665
5	3649	4736

Table 2: Number of trainable parameters in MLP models used for regression

To analyze the generalization error, we focus on Boston Housing Dataset only as it provides a better scale of error and is a better setting for regression than the Cancer dataset. Similar to classification we also observe the effect of dropouts on these models. The results for this setting are shown in Figure 7. We observe that without dropout, L-BFGS generalizes well, better than others for 3 layer model. With dropouts, this advantage wears off. For SGD, the dropout becomes effective as we increase the number of layers. K-FAC benefits the most form dropouts, experiencing a better drop in the generalization error than the other two. However, considering both the time-to-converge and the generalization error, SGD is a better choice for deeper models for regression.

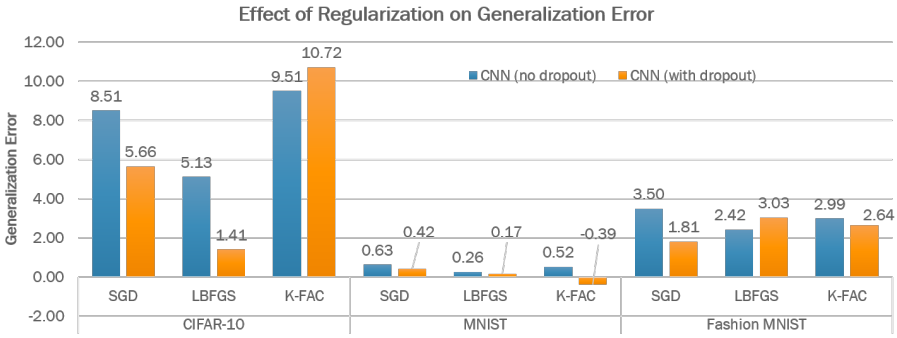

Figure 4: Effect of dropout on CNN-based classification models

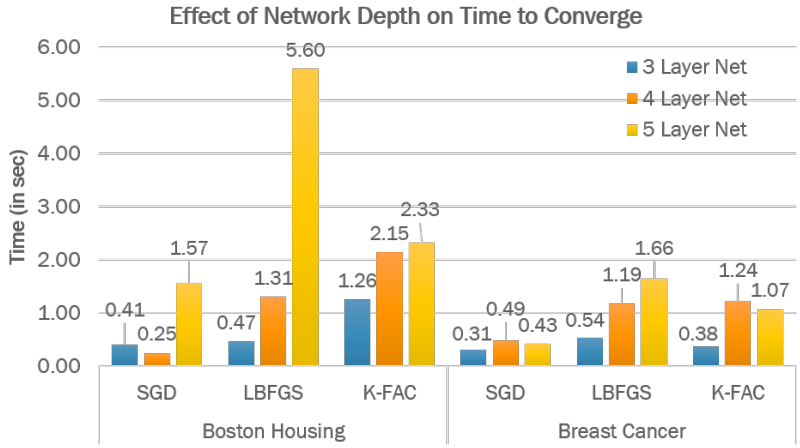

Figure 5: Time-to-converge of best performing regression models over each dataset

5. Conclusion

Based on our observations, we found that 2^{nd} order optimizers can be more effective than the 1^{st} order optimizers in certain settings. Following are our recommendations:

- For image classification
  - For grayscale images, K-FAC with CNN should be used for faster convergence
  - For colored images, SGD with CNN should be used for faster convergence
  - If high accuracy (up to decimal points), SGD with CNN should be used
  - For better generalization, SGD with dropouts should be used
  - If MLP is to be used for classification, SGD should be used for better performance (both accuracy and time-to-converge)

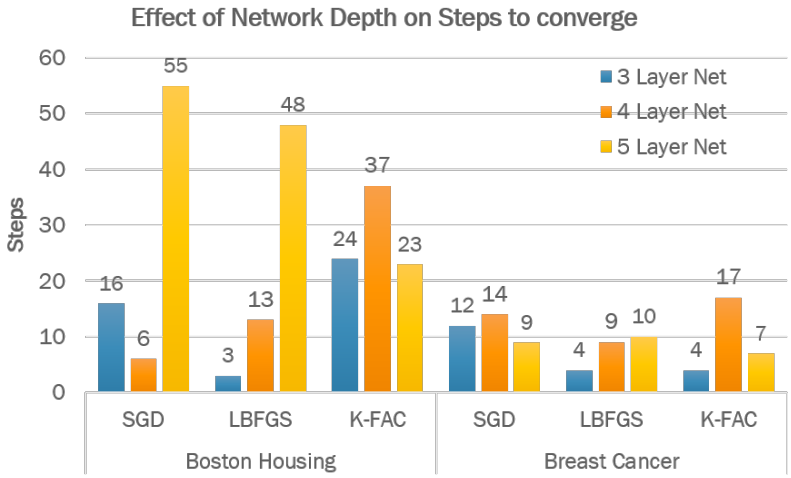

Figure 6: Steps-to-converge of best performing regression models over each dataset

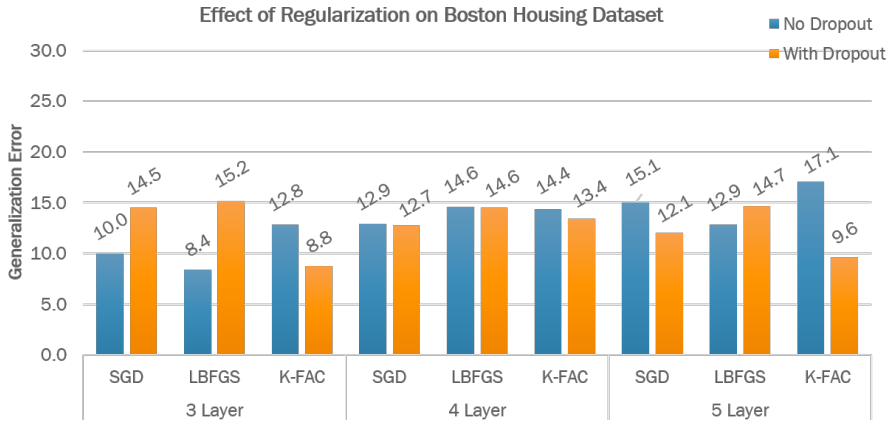

Figure 7: Effect of dropout on MLP models over Boston Housing dataset

As K-FAC took fewer steps to converge, it suggests that making the computation of its components efficient would greatly help the convergence speed and it may outperform SGD for colored images.

- For regression
  - If shallow network/MLP is being used, L-BFGS without mini-batch and dropout should be used. However, it can only be used for small data, as L-BFGS is not compatible with mini-batches
  - For deeper networks, SGD should be used for faster convergence
  - If LBFGS doesn't generalize well, K-FAC with dropouts should be used

We also found that L-BFGS work better with higher learning rates (0.1, 0.01) and K-FAC work better with lower learning rate (0.001, 0.0001). We also note another advantage

the SGD has over the 2^{nd} order optimizer in terms of ease of tuning. Compared to these optimizers, SGD does not show big fluctuation in accuracy/MSE on changing the learning rate. This also promotes use of SGD in experiment setups where a hyperparameter searching wrapper is avoided.

6. Future Work

While our problem setup required networks with around 3-8 layers, some problems may require much deeper and larger networks. It would be useful to see if our observations hold for such large networks and work well for bigger datasets. Another direction in this setting is the use of parallel computational systems for training DL models on huge data. Such a setting has been shown to work with SGD and its variation. It would be interesting to see such hardware architectures can use for 2^{nd} order optimizers as it would help reduce the computational complexity for them.

In our setting, we used CNNs as they can be used for computer vision, as well as sequence prediction. However, RNNs and its variations are often used for the later. Thus it would be useful to observe the efficacy of 2^{nd} order optimizers for this DL architecture. We would also like to analyze the stochastic version of L-BFGS in the future, as such implementations can be extended to large datasets, which are typically used in DL applications.

References

Qiming Chen and Ren Wu. Cnn is all you need. arXiv preprint arXiv:1712.09662, 2017.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(Jul):2121–2159, 2011.

David Harrison Jr and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. 1978.

Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. Neural networks for machine learning. Coursera, video lectures, 264:1, 2012.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Quoc V Le, Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, and Andrew Y Ng. On optimization methods for deep learning. 2011.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist, 2, 2010.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. Mathematical programming, 45(1-3):503–528, 1989.

Olvi L Mangasarian, W Nick Street, and William H Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4):570–577, 1995.

James Martens. Deep learning via hessian-free optimization. In ICML, volume 27, pages 735–742, 2010.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In International conference on machine learning, pages 2408–2417, 2015.

Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In Dokl. akad. nauk Sssr, volume 269, pages 543–547, 1983.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and

R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Raymond Perrault, Yoav Shoham, Erik Brynjolfsson, Jack Clark, John Etchemendy, Barbara Grosz, Terah Lyons, James Manyika, Saurabh Mishra, and Juan Carlos Niebles. The ai index 2019 annual report. AI Index Steering Committee, Human-Centered AI Institute, Stanford University, Stanford, CA, 2019.

Farbod Roosta-Khorasani and Michael W Mahoney. Sub-sampled newton methods i: globally convergent algorithms. arXiv preprint arXiv:1601.04737, 2016.

Sebastian Ruder. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747, 2016.

Nicol N Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-newton method for online convex optimization. In Artificial intelligence and statistics, pages 436–443, 2007.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Peng Xu, Farbod Roosta-Khorasani, and Michael W Mahoney. Second-order optimization for non-convex machine learning: An empirical study. arXiv preprint arXiv:1708.07827, 2017.

Chaoxu Zhou, Wenbo Gao, and Donald Goldfarb. Stochastic adaptive quasi-newton methods for minimizing expected values. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 4150–4159. JMLR. org, 2017.