DNN Compression and Acceleration

Шугаепов Ильнур

VK.com
Performance Advertising Team
ilnur.shug@gmail.com

Higher School of Economics, 2020

Introduction

- Neural Networks trained on BigData are usually both computationally intensive and memory intensive. For example
  - AlexNet — 200MB
  - DCN (criteo) — 168MB
  - DeepFM (VK dataset) — 20GB
- Therefore, it is hard to deploy them, especially on embedded systems with limited hardware resources

Main Goal

Reduce the storage and energy required (in a manner that preserves the original accuracy) to run inference on such large networks so they can be easily deployed.

- a — # of the parameters in the original model M
- a^* — # of the parameters in compressed model M^*

then the compression rate is

$$\alpha(M, M^*) = \frac{a}{a^*}$$

- s — running time of M
- s^* — running time of M^*

then the speedup rate is defined as

$$\delta(M, M^*) = \frac{s}{s^*}$$

Remark

Generally, the compression rate and speedup rate are highly correlated

Table of Contents

Reduce amount of bits required to store each weight

Table of Contents

- Dataset: $X^N = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, где $\mathbf{x}_i \in \mathbb{R}^n, y_i \in \{0, 1, \ldots, k-1\}$
- Task: find

$$f_\theta \in \mathcal{F} : \theta = \arg\min_\theta \mathcal{L}(X^N, f_\theta),$$

where $\mathcal{L}(X^N, f_\theta) = \frac{1}{N} \sum_{i=1}^N l(f_\theta(\mathbf{x}_i), y_i)$ — emperical risk and l - some error function

From now on $\mathcal{F}$ — family of DNNs

The main idea behind knowledge distillation is to train a compact model to approximate the function learned by a larger, more complex model.

Do Deep Nets Really Need to be Deep? [1]

- Teacher model f, probabilities of classes from logits

$$p_i = \frac{e^{z_i}}{\sum_j e^{z_j}},$$

  $\mathbf{z} = (z_0, z_1, \ldots, z_{k-1})$ — vector of logits

- Student model g trains on dataset $\hat{X}^N = \{(\mathbf{x}_i, \mathbf{z}_i)\}_{i=1}^N$ with following loss function

$$l(g(\mathbf{x}), \mathbf{z}) = \|g(\mathbf{x}) - \mathbf{z}\|_2^2 \quad \Rightarrow \quad \mathcal{L}(\hat{X}^N, g) = \frac{1}{N} \sum_{i=1}^{N} \|g(\mathbf{x}_i) - \mathbf{z}_i\|_2^2$$

Remark

Other mimic loss functions, such as

- *minimizing* $KL(p_{teacker} \| p_{student})$
- L_2 *loss on probabilities*

performs worse than regression on logits.

Student model Target

Q: Why do we use logits $\mathbf{z}$ as target and not directly probabilities $\mathbf{p}$?
A: Consider following examples

1. Consider $\mathbf{p}$ corresponding to logits $(10, 20, 30)$

$$\mathbf{p} = (2 \times 10^{-9}, 4 \times 10^{-5}, 0.9999)$$

   If cross entropy is minimized, the student will focus on the third target and tend to ignore the first and second targets.

2. Probabilities for logits $(-10, 0, 10)$ and $(10, 20, 30)$ after softmax are equal

A student trained on the logits for these targets, will better learn to mimic the detailed behaviour of the teacher model.

Evaluation

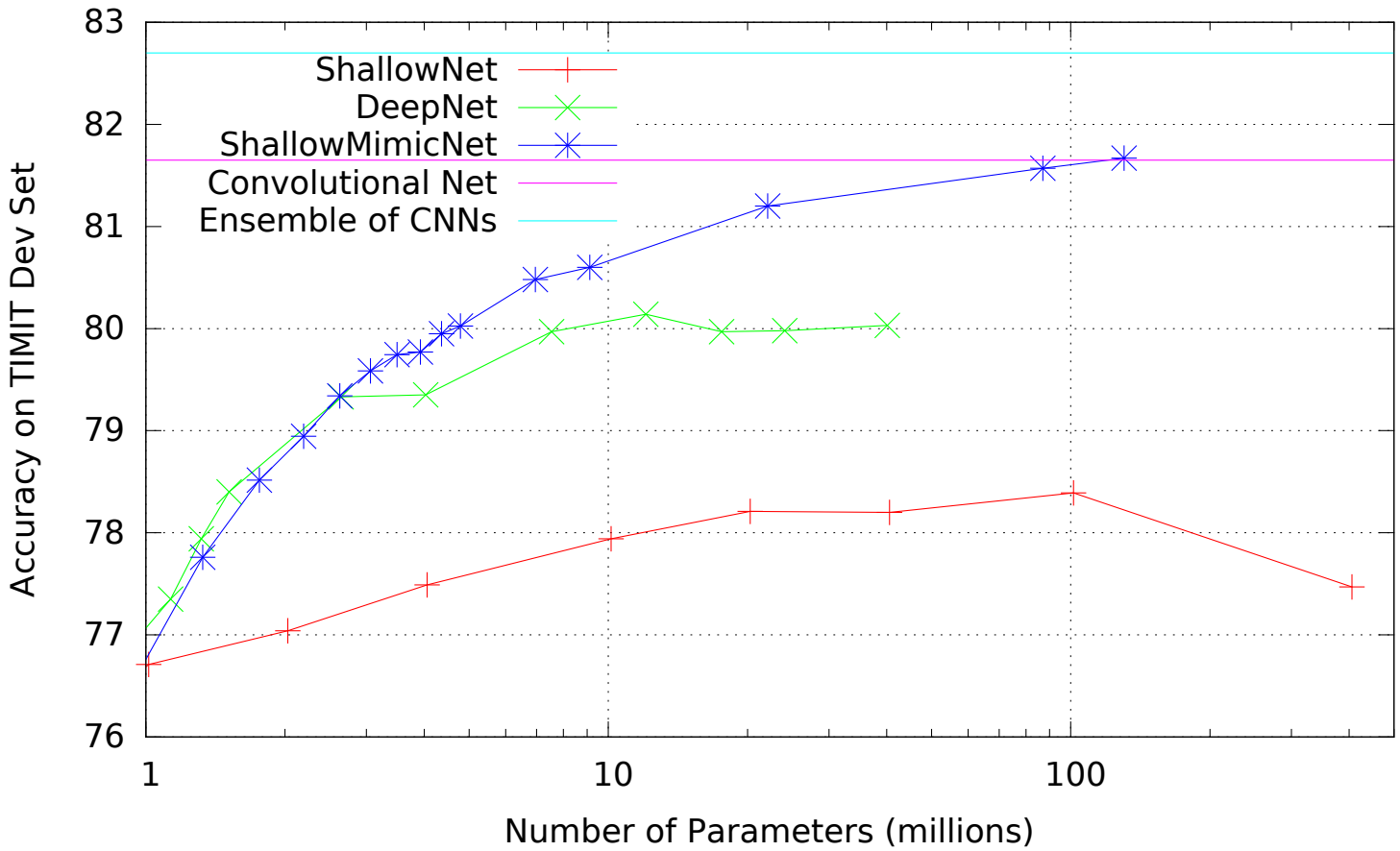


Рис.: Accuracy of SNNs, DNNs, and Mimic SNNs vs. # of parameters on TIMIT Dev. Accuracy of the CNN and target ECNN are shown as horizontal lines for reference. CNN # of params $\sim$ 13M, ECNN # of params $\sim$ 125M.

Remarks

- Often it is not (yet) possible to train a small model on the X^N to be as accurate as the complex model f, nor as accurate as the student model g
- Compression demonstrates that a small neural net could, in principle, learn the more accurate function, but current learning algorithms are unable to train a model with that accuracy from the original training data

Distilling the Knowledge in a Neural Network [6]

Teacher model f

- Probabilities of classes from logits

$$p_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}},$$

T - temperature.

Remark

Using a higher value for T produces a softer probability distribution over classes.

- For training we use high values of T and produce *soft-targets* $\mathbf{p} = (p_0, p_1, \ldots, p_{k-1})$

- Trains on dataset $\hat{X}^N = \{(\mathbf{x}_i, \mathbf{p}_i)\}_{i=1}^N$ with loss function

$$\mathcal{L} = w\mathcal{L}_1 + (1-w)\mathcal{L}_2,$$

where

- $\mathcal{L}_1$ — cross entropy with the soft targets, i.e $-\sum_i \mathbf{p}_i \log \mathbf{q}_i$
- $\mathcal{L}_2$ — cross entropy with the correct labels, i.e $-\sum_i \mathbf{y}_i \log \mathbf{q}_i$
- w — weight

Remark

Best results generally obtained by using a considerably lower weight on the $\mathcal{L}_2$

- The same high temperature is used when training g, but after it has been trained it uses a temperature of 1

Matching logits is a special case of distillation

System	Test Frame Accuracy	WER
Baseline	58.9%	10.9%
10xEnsemble	61.1%	10.7%
Distilled Single model	60.8%	10.7%

Таблица: Frame classification accuracy and WER showing that the distilled single model performs about as well as the averaged predictions of 10 models that were used to create the soft targets.

System & training set	Train Frame Accuracy	Test Frame Accuracy
Baseline (100% of training set)	63.4%	58.9%
Baseline (3% of training set)	67.3%	44.5%
Soft Targets (3% of training set)	65.4%	57.0%

Таблица: Soft targets allow a new model to generalize well from only 3% of the training set. The soft targets are obtained by training on the full training set.

Resume

- KD-based approaches can make deeper models thinner and smaller and help significantly reduce the computational cost

Drawbacks

- KD can only be applied to classification tasks with softmax loss function

Table of Contents

Pruning
Main Idea

- Neural networks are typically over-parameterized, and there is significant redundancy for deep learning models
- One way to remove the redundancy is to remove some weights/neurons

Algorithm

1. Train Teacher model f
2. Pruning: remove all connections in f whose weight is lower than a threshold
3. Fine-tuning: retrain the sparse network g so the remaining connections can compensate for the connections that have been removed

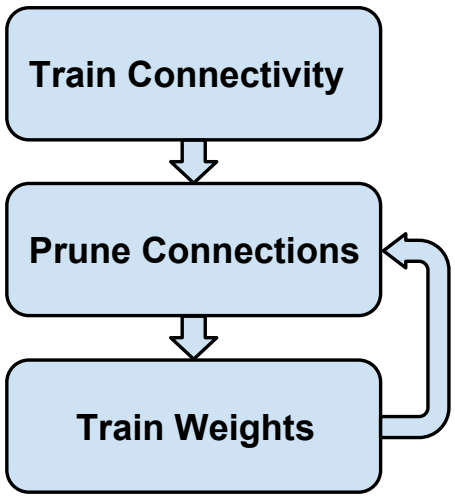


Рис.: Three-Step Training Pipeline.

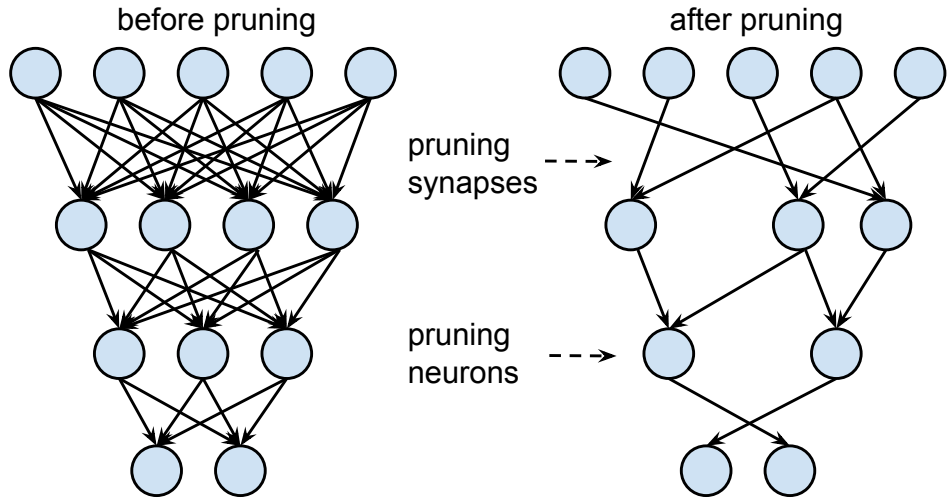


Рис.: Synapses and neurons before and after pruning.

Learning weights and connections for efficient neural net [5]

Remarks

Remark

- *If the pruned network g is used without retraining, accuracy is significantly impacted.*
- *So when we retrain the pruned layers, we should keep the surviving parameters instead of re-initializing them ... gradient descent is able to find a good solution when the network is initially trained, but not after re-initializing some layers and retraining them*

Q: Threshold is a hyperparameter. How to choose it?
A: Possible values

- choose t such that during pruning $p\%$ smallest-magnitude weights are removed
- or $p^{\frac{1}{n}}\%$, where n is a number of iteration
- ...

Evaluation

Lenet-300-100 and Lenet-5 on MNIST, AlexNet and VGG-16 on ImageNet.

Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	
LeNet-300-100 Pruned	1.59%	-	**22K**	**12**×
LeNet-5 Ref	0.80%	-	431K	
LeNet-5 Pruned	0.77%	-	**36K**	**12**×
AlexNet Ref	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9**×
VGG-16 Ref	31.50%	11.32%	138M	
VGG-16 Pruned	31.34%	10.88%	**10.3M**	**13**×

Таблица: Network pruning can save $9\times$ to $13\times$ parameters with no drop in predictive performance.

Таблица: For Lenet-300-100, pruning reduces the number of weights by $12\times$ and computation by $12\times$.

Layer	Weights	FLOP	Act%	Weights%	FLOP%
fc1	235K	470K	38%	8%	8%
fc2	30K	60K	65%	9%	4%
fc3	1K	2K	100%	26%	17%
Total	266K	532K	46%	**8%**	**8%**

For each layer of the network the table shows (left to right)

1. the original number of weights
2. the number of floating point operations to compute that layer's activations
3. the average percentage of activations that are non-zero
4. the percentage of non-zero weights after pruning
5. and the percentage of actually required floating point operations

Evaluation

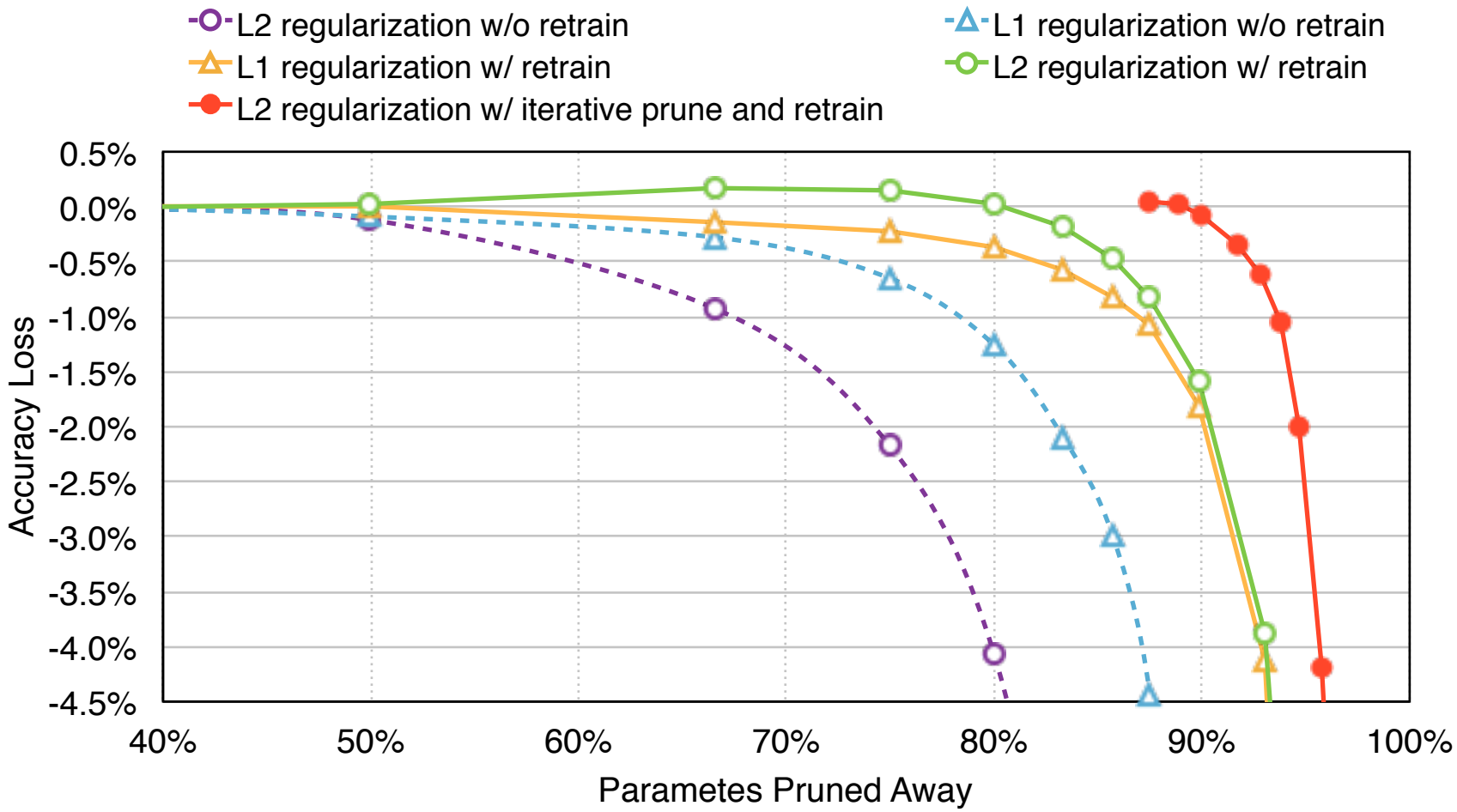
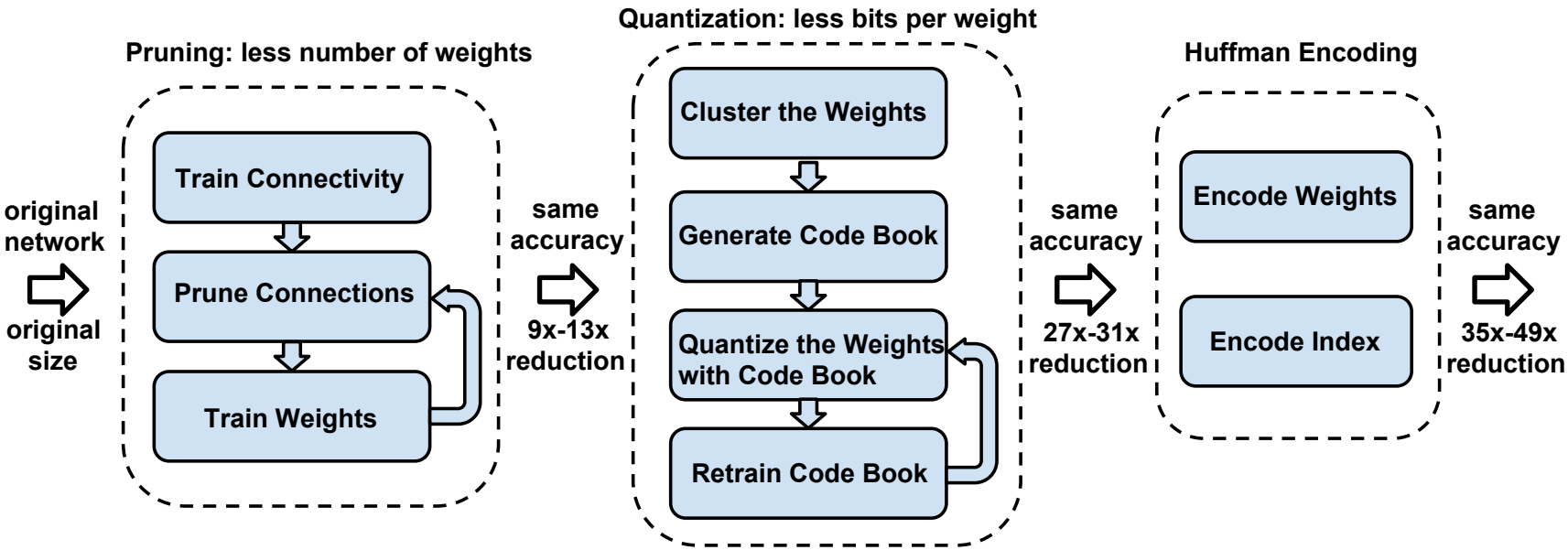


Рис.: Trade-off curve for parameter reduction and loss in top-5 accuracy. L1 regularization performs better than L2 at learning the connections without retraining, while L2 regularization performs better than L1 at retraining. Iterative pruning gives the best result.

Deep Compression [4]



- We store the sparse structure that results from pruning using compressed sparse row (CSR) or compressed sparse column (CSC) format
- To compress further, we store the index difference instead of the absolute position, and encode this difference in 8 bits for conv layer and 5 bits for fc layer

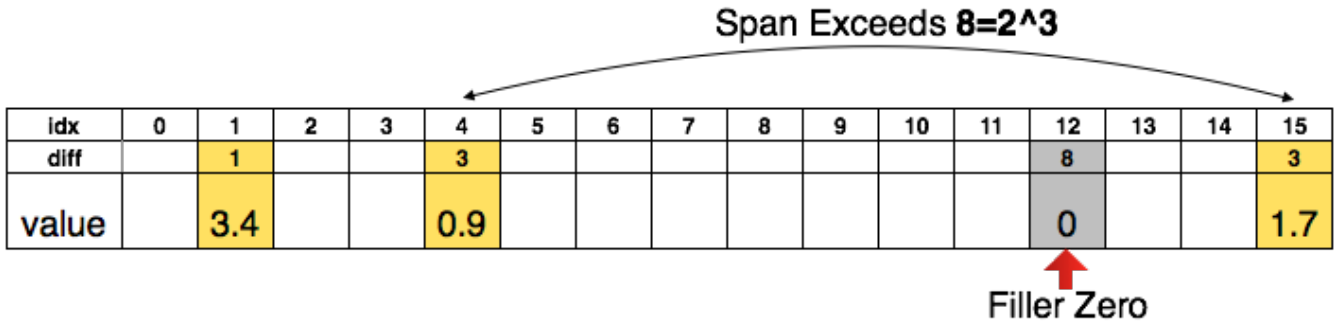


Рис.: Representing the matrix sparsity with relative index. Padding filler zero to prevent overflow

Quantization: less bits per weight

Weight Sharing aka Cluster the Weights

K-means clustering to identify the shared weights for each layer of a trained network

We partition n original weights $W = \{w_1, w_2, \ldots, w_n\}$ into k clusters $C = \{c_1, c_2, \ldots, c_k\}$, $n \gg k$, so as to minimize

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{w \in c_i} |w - c_i|^2$$

The centroids of the one-dimensional k-means clustering are the shared weights

Remark

1. *Weights are not shared across layers*
2. *Method determines weight sharing after a network is fully trained*

Quantization: less bits per weight

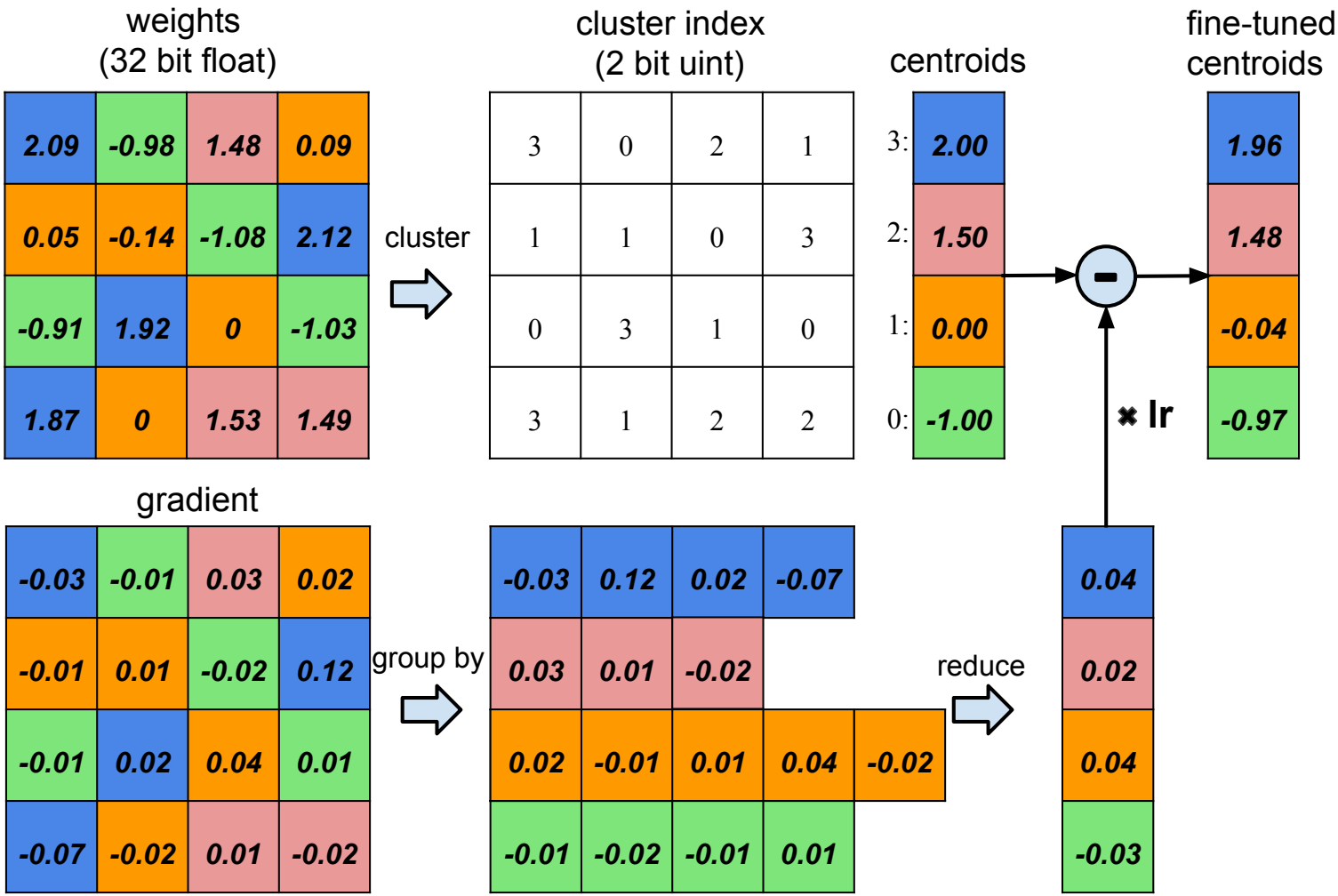


Рис.: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom).

Gradient of loss $\mathcal{L}$ wrt centroid C_k

$$\frac{\partial \mathcal{L}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \frac{\partial W_{ij}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \mathbb{1}\{I_{ij} = k\},$$

where I_{ij} — centroid index of weight W_{ij}

During update:

1. all the gradients are grouped by the color and summed together
2. multiplied by the learning rate and
3. subtracted from the shared centroids from last iteration

Huffman Encoding

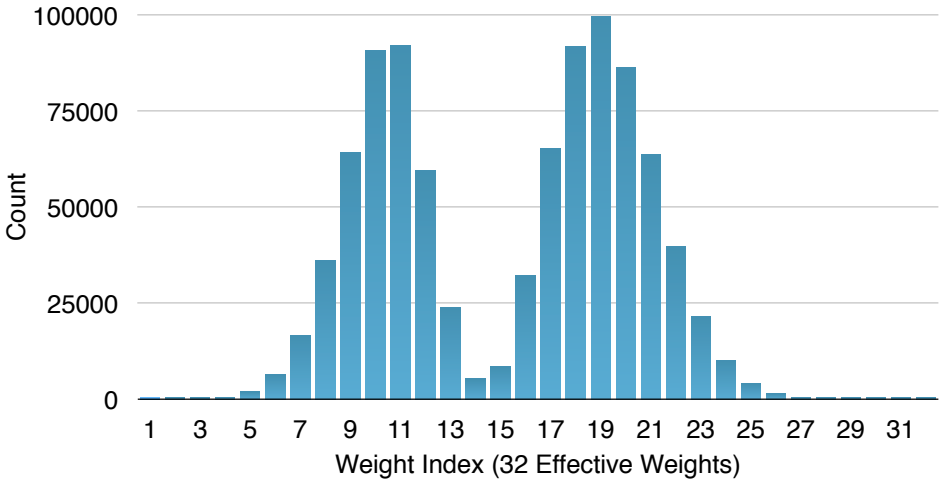


Рис.: Distribution for weight

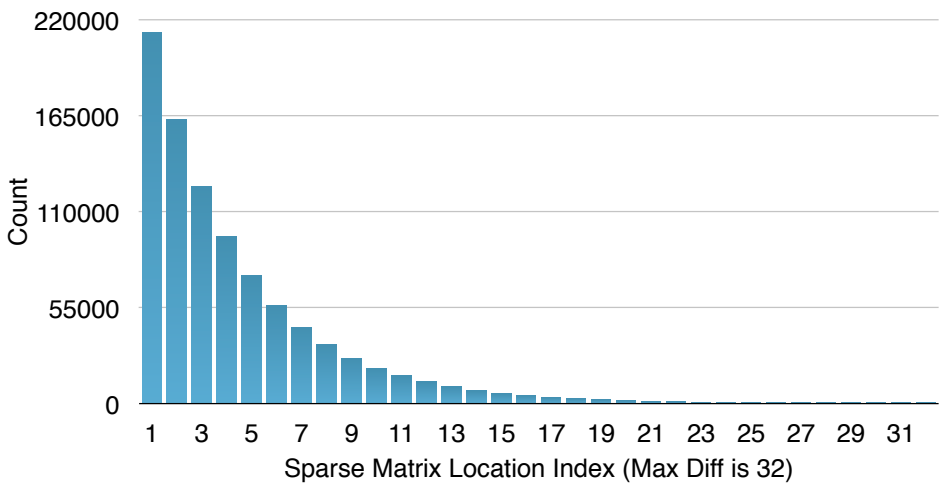


Рис.: Distribution for index

Huffman coding these non-uniformly distributed values saves $20\% - 30\%$ of network storage

Lottery Ticket Hypothesis [3]

- Previous works on pruning showed how to find effective sub-networks
- This work shows that
  - not only stucture makes them effective but initiallization as well
  - there consistently exist smaller subnetworks that train from the start and learn at least as fast as their larger counterparts while reaching similar test accuracy

Lottery Ticket Hypothesis [3]

Hypothesis

A randomly-initialized, dense neural network contains a subnet-work that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.

Identifying winning tickets

One-shot Algorithm

1. Randomly initialize a neural network f_{θ_0} (where $\theta_0 \sim \mathcal{D}_\theta$).
2. Train the network for j iterations, arriving at parameters θ_j.
3. Prune p% of the parameters in θ_j, creating a mask m.
4. Reset the remaining parameters to their values in θ_0, creating the winning ticket $f_{m \odot \theta_0}$.

Iterative pruning:

repeatedly trains, prunes, and resets the network over n rounds; each round prunes $p^{\frac{1}{n}}$% of the weights that survive the previous round.

Future Research

Explore more efficient methods for finding winning tickets

Table of Contents

Links

PyTorch

- Quantization[1]
- Pruning[2]

[1] https://pytorch.org/docs/stable/quantization.html
[2] https://pytorch.org/tutorials/intermediate/pruning_tutorial.html

Resume

1. We have familiarized with several techniques for DNNs compression and acceleration
2. We saw how these techniques can be combined to obtain even better results

References I

[1] J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[2] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[3] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[4] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[5] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[6] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[7] G. Ke, Z. Xu, J. Zhang, J. Bian, and T.-Y. Liu. Deepgbm: A deep learning framework distilled by gbdt for online prediction tasks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 384–394, 2019.