Pruning Is All You Need: an overview of the Lottery Ticket Hypothesis

Nikita Petrashen

Skoltech

2020

Overview

- What's the deal?
- How to find the lottery ticket?
- Some experimental results
- Proof of the hypothesis
- Takeaways

What's the deal?

- Neural networks are highly overparametrized models
- Pruning has shown itself as an efficient method of reducing the model size
- A recent result: pruned networks can be trained in isolation [3]

Lottery Ticket Hypothesis

- Soft version:
  Pruned neural networks can be trained to achieve good performace, when resetting their weights to their initial values [3]
- Stronger version:
  A sufficiently overparametrized network with random initialization contains a subnetwork which yields competitive accuracy w.r.t. the large TRAINED network without ANY TRAINING [4]

Pruning vs. Training

- How to find the subnetwork?
- As of yet, pruning algorithms have the same complexity as learning the weights
- New direction for research

How to find a good subnetwork: first approach [2]

- Learn a probability associated with each weight
- During the forward pass exclude the weight with this probability
- Drawback: each subnetwork appears only once during the forward pass (extremely large number of subnetworks)

How to find a good subnetwork: second approach [4]

- Associate a learned score s_{uv} with each weight
- Select only weights with top k scores at each forward pass
- Update rule: $s_{uv} = s_{uv} - \alpha \frac{\partial \mathcal{L}}{\partial \mathcal{I}_v} \mathcal{Z}_u w_{uv}$
- $\mathcal{I}_v$ is the input to the v-th neuron
- $\mathcal{Z}_u$ is the output of the u-th neuron
- Intuition: the score of the weight increases if adding the respective term to the input of node v decreases the loss

More details

- $\mathcal{I}_v = \sum_{(u,v)\in\mathcal{E}^k} w_{uv}\mathcal{Z}_u$
- Can be rewritten as $\mathcal{I}_v = \sum_u w_{uv}\mathcal{Z}_u h(s_{uv})$
- $h(s_{uv}) = 1$ if s_{uv} is among the top-k scores
- $\frac{dh}{ds_{uv}}$ is not really defined, treat it as an identity during the backward pass
- Then, formally, $\frac{\partial\mathcal{L}}{\partial s_{uv}} = \frac{\partial\mathcal{L}}{\partial\mathcal{I}_v}\frac{\partial\mathcal{I}_v}{\partial s_{uv}} = \frac{\partial\mathcal{L}}{\partial\mathcal{I}_v}\mathcal{Z}_u w_{uv}$

Some more details..

- Theorem:
  When edge $(\tilde{u}, v)$ replaces (u, v), the loss decreases
- Proof:
  The algorithm implies that $-\frac{\partial \mathcal{L}}{\partial \mathcal{I}_v} \mathcal{Z}_{\tilde{u}} w_{\tilde{u}v} > -\frac{\partial \mathcal{L}}{\partial \mathcal{I}_v} \mathcal{Z}_u w_{uv}$
- Expand loss up to the first order:

$$\mathcal{L}(\tilde{I}_v) = \mathcal{L}(\mathcal{I}_v) + \frac{\partial \mathcal{L}}{\partial \mathcal{I}_v}(\tilde{I}_v - \mathcal{I}_v) = \mathcal{L}(\mathcal{I}_v) + \frac{\partial \mathcal{L}}{\partial \mathcal{I}_v}(\mathcal{Z}_{\tilde{u}} w_{\tilde{u}v} - \mathcal{Z}_u w_{uv})$$

- From this it follows that $\mathcal{L}(\tilde{I}_v) < \mathcal{L}(\mathcal{I}_v)$

How it looks in the code

```
class GetMask(autograd.Function):
    @staticmethod
    def forward(self, scores, k):
        out = scores.clone()
        _, idx = scores.flatten().sort()
        j = int((1 - k) * scores.numel())

        flat_out = out.flatten()
        flat_out[idx[:j]] = 0
        flat_out[idx[j:]] = 1

        return out

    @staticmethod
    def backward(self, g):
        return g, None
```

Figure 1: Code snippet which computes the function h

Experiments

- The paper: VGG-like networks on CIFAR-10, ResNets of different depth on ImageNet
- What I did: 2-layer CNN on MNIST, VGG-like network on CIFAR-10
- Three setups:
  - Compare pruned network with different k with its trained version
  - Study the effect of overparametrization (increasing width)
  - Study the effect of width variation with a fixed number of parameters

Pruned vs. Trained

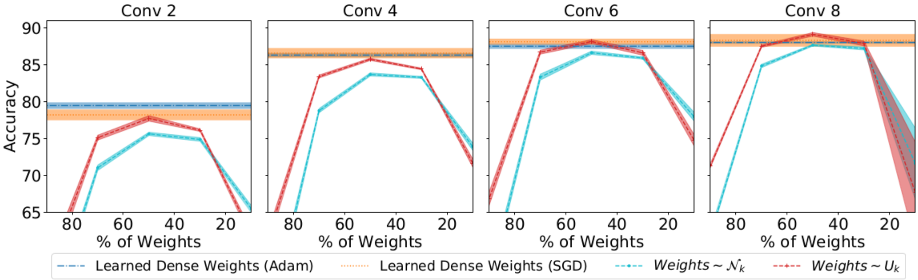

Figure 2: Paper

Pruned vs. Trained

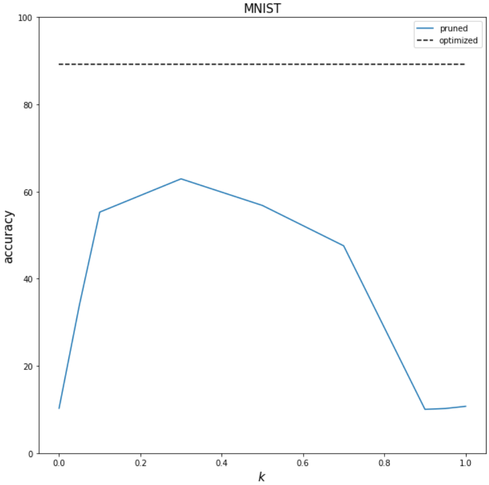

Figure 3: MNIST

Pruned vs. Trained

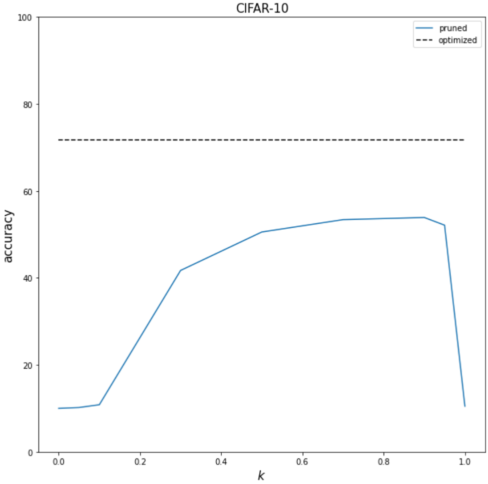

Figure 4: CIFAR

The effect of overparametrization

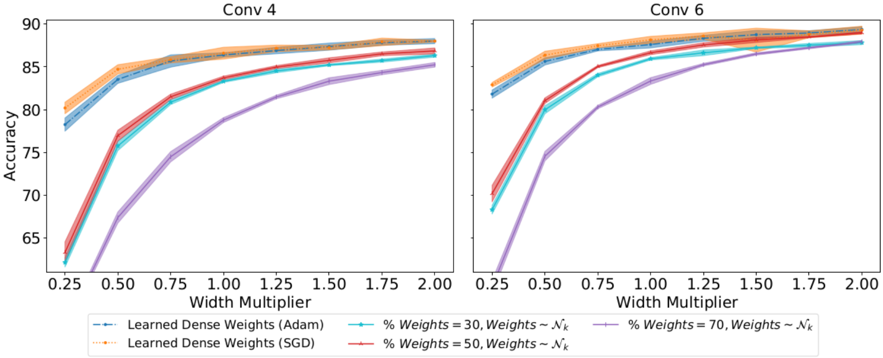

Figure 5: Paper

The effect of overparametrization

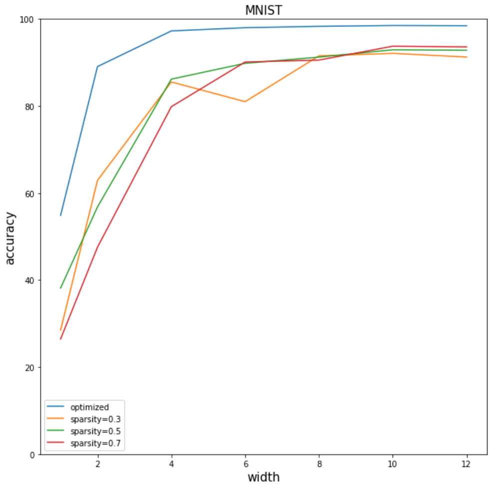

Figure 6: MNIST

The effect of overparametrization

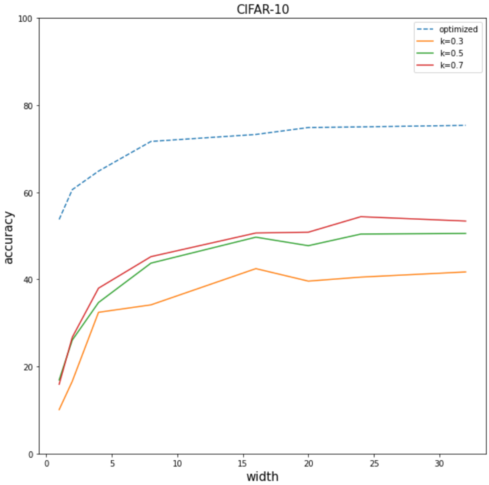

Figure 7: CIFAR-10

The effect of width

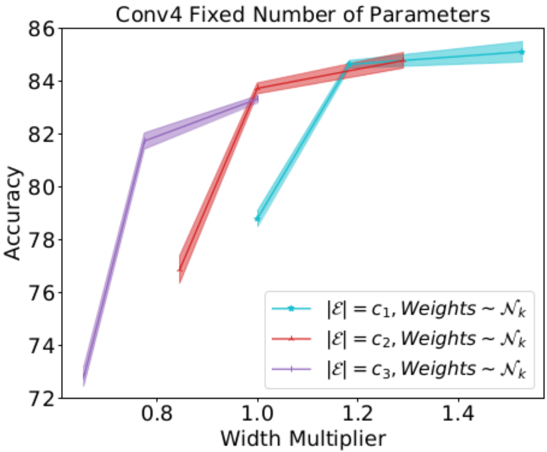

Figure 8: Paper

The effect of width

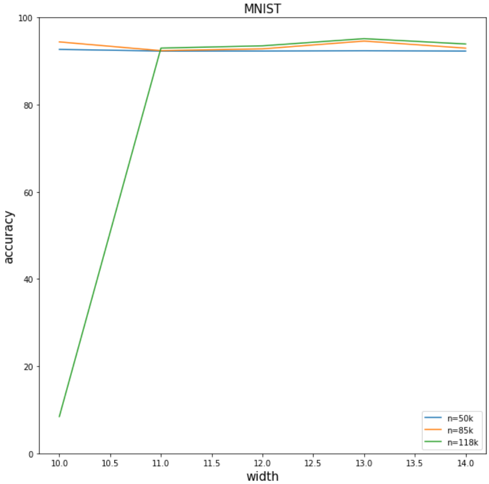

Figure 9: MNIST

The effect of width

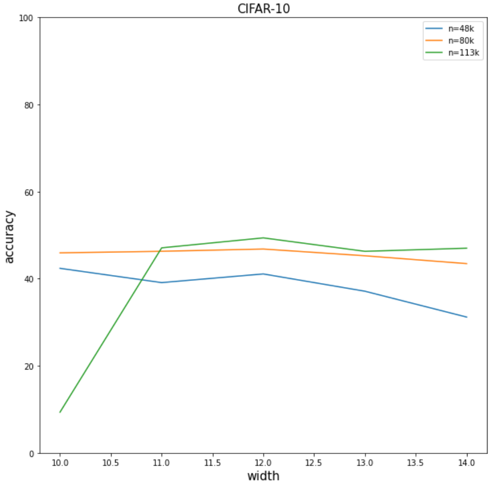

Figure 10: CIFAR-10

Theorem [1]

Fix some $\epsilon, \delta \in (0,1)$. Let F be some target network of depth l and width n s.t. for every i we have $||W_i||_2 \leq 1,\ ||W_i||_{max} \leq \frac{1}{\sqrt{n_{in}}}$. Let G be a network of width $poly(d, n, l, 1\epsilon, log\frac{1}{\delta})$, where d is the input dimension, and depth $2l$, with weights initialized from $U([-1,1])$. Then with probability at least $1-\delta$ there exist a subnetwork $\tilde{G}$ of G s.t.

$$\sup_{x} \in \mathcal{X} \left| \tilde{G}(x) - F(x) \right| \leq \epsilon$$

Furthermore, the number of non-zero weights in $\tilde{G}$ is $O(dn\ +\ n^2 l$, i.e. is comparable to the total number of weights in F.

Sketch of proof

- Show that a single neuron can be approximated by pruning a two-layer network
- Stack neurons to approximate a layer
- Stack layers to approximate a network
- The full proof is cumbersome, takes 6 A4 pages

Takeaways

- Randomly initialized networks contain subnetworks with decent performance without any training
- However, the complexity of pruning algorithms is the same as of learning the weights
- Still, this is a new and interesting direction for research

Thank you for your attention

References

Eran Malach et al. “Proving the Lottery Ticket Hypothesis: Pruning is All You Need”. In: (2020).

H. Zhou et al. “Deconstructing lottery tickets: Zeros, signs, and the super- mask”. In: (2019).

J. Frankle et al. “The lottery ticket hypothesis: Finding sparse, trainable neural networks”. In: (2018).

V. Ramanujan et al. “What’s hidden in a randomly weighted neural network?” In: (2019).