Whats Hidden in a Randomly Weighted Neural Network? [1]

March 5, 2020

[1] *What's Hidden in a Randomly Weighted Neural Network?* by Ramanujan et al.

Overview

Overview

- Authors took a randomly initialized model.

Overview

- Authors took a randomly initialized model.
- Then they trained a binary mask for this model to extract a good subnetwork.

Overview

- Authors took a randomly initialized model.
- Then they trained a binary mask for this model to extract a good subnetwork.
- The performance of this subnetwork turned out to be very good.

Overview

- Authors took a randomly initialized model.
- Then they trained a binary mask for this model to extract a good subnetwork.
- The performance of this subnetwork turned out to be very good.

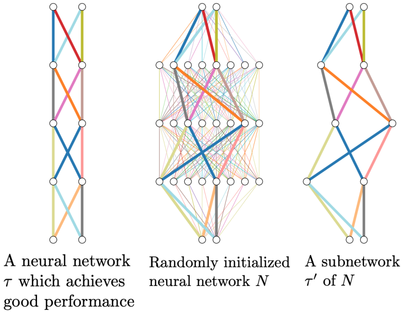

Figure: Since we have combinatorial number of subnetworks and modern models have millions of parameters we are likely to find a good one.

How to find a good subnetwork

How to find a good subnetwork

1. Initialize the model randomly

How to find a good subnetwork

1. Initialize the model randomly
2. Associate a score s_{uv} with each synapse $u \rightarrow v$.

How to find a good subnetwork

1. Initialize the model randomly
2. Associate a score s_{uv} with each synapse $u \to v$.
3. Use weights that have top $k\%$ scores for forward pass.

How to find a good subnetwork

1. Initialize the model randomly
2. Associate a score s_{uv} with each synapse $u \to v$.
3. Use weights that have top $k\%$ scores for forward pass.
4. Use straight-through estimator (STE) to train the scores.

How to find a good subnetwork

1. Initialize the model randomly
2. Associate a score s_{uv} with each synapse $u \to v$.
3. Use weights that have top $k\%$ scores for forward pass.
4. Use straight-through estimator (STE) to train the scores.

Normally, input at neuron v is computed as:

$$\mathcal{I}_v = \sum_{(u,v)\in\mathcal{E}} w_{uv}\mathcal{Z}_u \tag{1}$$

How to find a good subnetwork

1. Initialize the model randomly
2. Associate a score s_{uv} with each synapse $u \to v$.
3. Use weights that have top $k\%$ scores for forward pass.
4. Use straight-through estimator (STE) to train the scores.

Normally, input at neuron v is computed as:

$$\mathcal{I}_v = \sum_{(u,v)\in\mathcal{E}} w_{uv}\mathcal{Z}_u \tag{1}$$

When we learn scores, we compute it as:

$$\mathcal{I}_v = \sum_{u\in\mathcal{V}^{(\ell-1)}} w_{uv}\mathcal{Z}_u h\left(s_{uv}\right), \qquad h(s_{uv}) = \begin{cases} 1, \text{if } s_{uv} \text{ is in top-k scores,} \\ 0, \text{otherwise} \end{cases} \tag{2}$$

How to find a good subnetwork

1. Initialize the model randomly
2. Associate a score s_{uv} with each synapse $u \to v$.
3. Use weights that have top $k\%$ scores for forward pass.
4. Use straight-through estimator (STE) to train the scores.

Normally, input at neuron v is computed as:

$$\mathcal{I}_v = \sum_{(u,v)\in\mathcal{E}} w_{uv}\mathcal{Z}_u \tag{1}$$

When we learn scores, we compute it as:

$$\mathcal{I}_v = \sum_{u\in\mathcal{V}^{(\ell-1)}} w_{uv}\mathcal{Z}_u h\left(s_{uv}\right), \qquad h(s_{uv}) = \begin{cases} 1, \text{if } s_{uv} \text{ is in top-k scores,} \\ 0, \text{otherwise} \end{cases} \tag{2}$$

And we update it via STE:

$$\tilde{s}_{uv} = s_{uv} - \alpha \frac{\partial\mathcal{L}}{\partial\mathcal{I}_v} w_{uv}\mathcal{Z}_u \tag{3}$$

Straight-through estimator (STE)

- Imagine, we have a non-differentiable function $f(\theta)$

Straight-through estimator (STE)

- Imagine, we have a non-differentiable function $f(\theta)$
  - for example, thresholding 5% of the highest values
- Imagine, we use it in the computation of some complex function $L(f(\theta), x)$.

Straight-through estimator (STE)

- Imagine, we have a non-differentiable function $f(\theta)$
  - for example, thresholding 5% of the highest values
- Imagine, we use it in the computation of some complex function $L(f(\theta), x)$.
  - for example, classification loss on a batch x.

Straight-through estimator (STE)

- Imagine, we have a non-differentiable function $f(\theta)$
  - for example, thresholding 5% of the highest values
- Imagine, we use it in the computation of some complex function $L(f(\theta), x)$.
  - for example, classification loss on a batch x.
- STE is a trick to compute the gradients w.r.t. to θ through f:

Straight-through estimator (STE)

- Imagine, we have a non-differentiable function $f(\theta)$
  - for example, thresholding 5% of the highest values
- Imagine, we use it in the computation of some complex function $L(f(\theta), x)$.
  - for example, classification loss on a batch x.
- STE is a trick to compute the gradients w.r.t. to θ through f:
  - During forward pass, use $f(\theta)$ as is.

Straight-through estimator (STE)

- Imagine, we have a non-differentiable function $f(\theta)$
  - for example, thresholding 5% of the highest values
- Imagine, we use it in the computation of some complex function $L(f(\theta), x)$.
  - for example, classification loss on a batch x.
- STE is a trick to compute the gradients w.r.t. to θ through f:
  - During forward pass, use $f(\theta)$ as is.
  - During backward pass, replace $f(\theta)$ with just θ!

Straight-through estimator (STE)

- Imagine, we have a non-differentiable function $f(\theta)$
  - for example, thresholding 5% of the highest values
- Imagine, we use it in the computation of some complex function $L(f(\theta), x)$.
  - for example, classification loss on a batch x.
- STE is a trick to compute the gradients w.r.t. to θ through f:
  - During forward pass, use $f(\theta)$ as is.
  - During backward pass, replace $f(\theta)$ with just θ!
  - I.e. we compute the loss using the real value $f(\theta)$, but "remove" gradients from chain-rule multiplication that we cannot compute

Straight-through estimator (STE)

- Imagine, we have a non-differentiable function $f(\theta)$
  - for example, thresholding 5% of the highest values
- Imagine, we use it in the computation of some complex function $L(f(\theta), x)$.
  - for example, classification loss on a batch x.
- STE is a trick to compute the gradients w.r.t. to θ through f:
  - During forward pass, use $f(\theta)$ as is.
  - During backward pass, replace $f(\theta)$ with just θ!
  - I.e. we compute the loss using the real value $f(\theta)$, but "remove" gradients from chain-rule multiplication that we cannot compute
  - Imagine that $L(f(\theta), x) = a(f(\theta(s)), x)$, then:

$$\partial_\theta L = \frac{\partial L}{\partial a}\frac{\partial a}{\partial f}\color{red}{\frac{\partial f}{\partial \theta}}\color{black}{\frac{\partial \theta}{\partial s}} \approx \frac{\partial L}{\partial a}\frac{\partial a}{\partial f}\frac{\partial \theta}{\partial s}$$

Straight-through estimator (STE)

- Imagine, we have a non-differentiable function $f(\theta)$
  - for example, thresholding 5% of the highest values
- Imagine, we use it in the computation of some complex function $L(f(\theta), x)$.
  - for example, classification loss on a batch x.
- STE is a trick to compute the gradients w.r.t. to θ through f:
  - During forward pass, use $f(\theta)$ as is.
  - During backward pass, replace $f(\theta)$ with just θ!
  - I.e. we compute the loss using the real value $f(\theta)$, but "remove" gradients from chain-rule multiplication that we cannot compute
  - Imagine that $L(f(\theta), x) = a(f(\theta(s)), x)$, then:

$$\partial_\theta L = \frac{\partial L}{\partial a} \frac{\partial a}{\partial f} \frac{\partial f}{\partial \theta} \frac{\partial \theta}{\partial s} \approx \frac{\partial L}{\partial a} \frac{\partial a}{\partial f} \frac{\partial \theta}{\partial s}$$

The best explanation of STE:

```python
def straight_through_estimation(x, f):
  # In forward pass it is equal to f(x)
  # In backward pass `(...).detach()` term vanishes
  return (f(x) - x).detach() + x
```

Results

Results

Authors tried two different initializations:

Results

Authors tried two different initializations:

- Kaiming Normal (for ReLU it is $\mathcal{D}_\ell = \mathcal{N}(0, \sqrt{2/n_{\ell-1}}))$

Results

Authors tried two different initializations:

- Kaiming Normal (for ReLU it is $\mathcal{D}_\ell = \mathcal{N}(0, \sqrt{2/n_{\ell-1}})$)
- Signed Constant: set each weight to σ, then randomly choose its $+/-$ sign

Method	Model	Initialization	% of Weights	# of Parameters	Accuracy
Learned Dense Weights (SGD)	ResNet-34 [8]	-	-	21.8M	73.3%
	ResNet-50 [8]	-	-	25M	76.1%
	Wide ResNet-50 [28]	-	-	69M	78.1%
`edge-popup`	ResNet-50	Kaiming Normal	30%	7.6M	61.71%
	ResNet-101	Kaiming Normal	30%	13M	66.15%
	Wide ResNet-50	Kaiming Normal	30%	20.6M	67.95%
`edge-popup`	ResNet-50	Signed Constant	30%	7.6M	68.6%
	ResNet-101	Signed Constant	30%	13M	72.3%
	Wide ResNet-50	Signed Constant	30%	20.6M	73.3%

Figure: ImageNet results (with $k = 30\%$)

Varying % of remaining weights

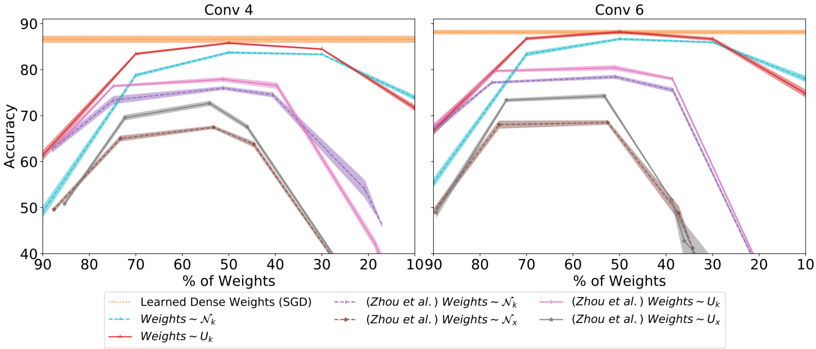

Figure: Varying % of remaining weights for AlexNet for CIFAR-10. Maximum in the middle since $\binom{n}{k}$ is maximized at $k \approx n/2$

Varying width of a specific layer

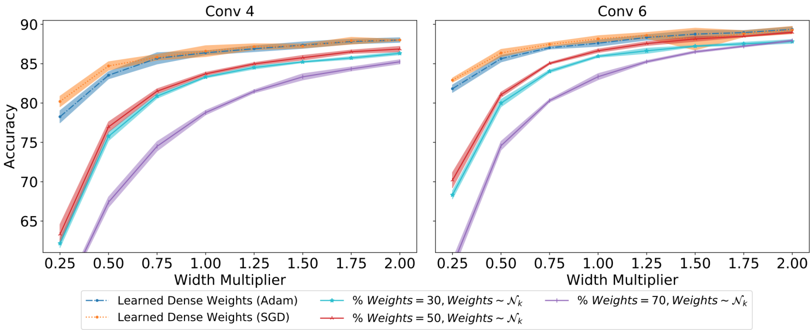

Figure: Varying with of a layer for AlexNet for CIFAR-10

Conclusion

Conclusion

- Random weights with a good connectivity pattern work as good as trained weights.

Conclusion

- Random weights with a good connectivity pattern work as good as trained weights.
- Signed Constant init works better than Kaiming Normal init, which means that precise weights values are not important. We just need magnitudes and a connectivity pattern.

Conclusion

- Random weights with a good connectivity pattern work as good as trained weights.
- Signed Constant init works better than Kaiming Normal init, which means that precise weights values are not important. We just need magnitudes and a connectivity pattern.
- HAT-algorithm for CL is not "fair"?