Keep the Gradients Flowing: Using Gradient Flow to Study Sparse Network Optimization

Kale-ab Tessera, Sara Hooker, Benjamin Rosman

WITS UNIVERSITY

R A I L
L A B

InstaDeep™

Why is Sparsity Interesting?

Sparse Networks can lead to:

- ❏ Faster training and inference times. [1,2,3]
- ❏ More robust to noise. [4]
- ❏ Improving efficiency - memory or energy. [5,6]

Similar or better performance than dense networks?

Sparsity Research - Focus on Initialization

- A lot of great work focusing on initialization - finding special weight initializations or "*lottery tickets*". [7,8,9]
- Focusing on initialization alone has proved to be **inadequate**. [10,11]
- Optimization outside of early stages of training is poorly understood - e.g. sensitivity of lottery tickets to higher learning rates. [9,10,11]

- Existing work:
  - Grad Flow during DST [12]
  - Loss landscape [13]
  - Signal propagation [14]
  - SGD Noise [15]

- What about **training dynamics**?
  - Regularization/ Normalization.
  - Optimization methods.
  - Activation functions.
  - Learning rates.
  - Their interactions?

Our Setting - When to Prune

➔ Pruning **Before** Training (Pruning From Scratch)/Early in training.
- Aim to **start sparse**. Use certain criteria -> estimate which weights should remain active.

➔ Pruning During Training (Dynamic Sparsity)
- Use information gathered during the training process to **dynamically** and/or iteratively **update the sparsity pattern** of networks.

➔ Pruning After Training
- Train a dense network, **then prune unimportant weights** (*maybe* fine-tune afterwards) .

Our Setting - What to Prune

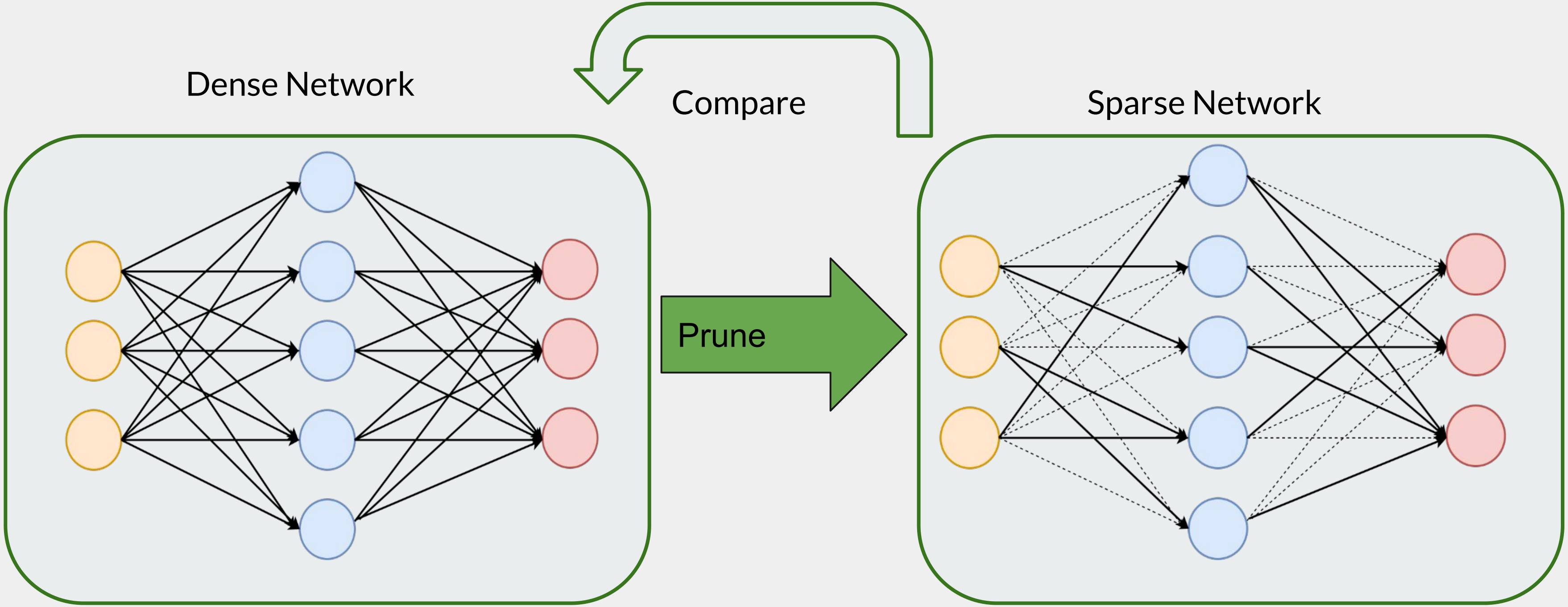
- Impact on loss or the Hessian of the loss function.
- Magnitude Pruning.
- Connection sensitivity/Salency - SNIP[16] / SynFlow[17].
- Gradient flow - GRASP[18].
- Random Pruning.

[11,19] showed that for pruning from scratch methods, shuffling the preserved weights does not affect final performance.

Sparsity Setting

Pruning From Scratch + Random Pruning.

Current way to compare sparse and dense networks



Issues

1. Networks are different capacity.
2. Initial weight distributions are different.
3. Train for different training times.

Ensure Same Capacity

Calculate active weights for Sparse Networks **S** and Dense Network **D**, **Q^{l}** is the weights in layer **l** and **m^{l}** is the mask applied to layer **l**.

$$\boxed{a^l_S = \theta^l_S \odot m^l} \quad , \quad \boxed{a^l_D = \theta^l_D}, \quad \text{for} \quad l = 1, \ldots, L \quad ,$$

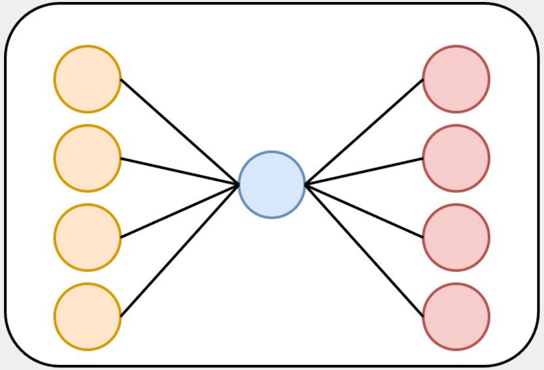
Active weights in layer l of sparse network.

Active weights in layer l of dense network.

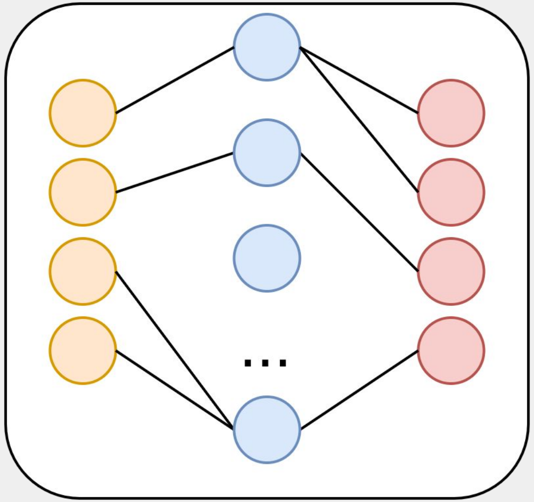
Ensure same number of nonzero weights in each layer.

$$||a^l_S||_0 = ||a^l_D||_0, \quad \text{for} \quad l = 1, \ldots, L$$

Ensure Same Capacity



Dense Network



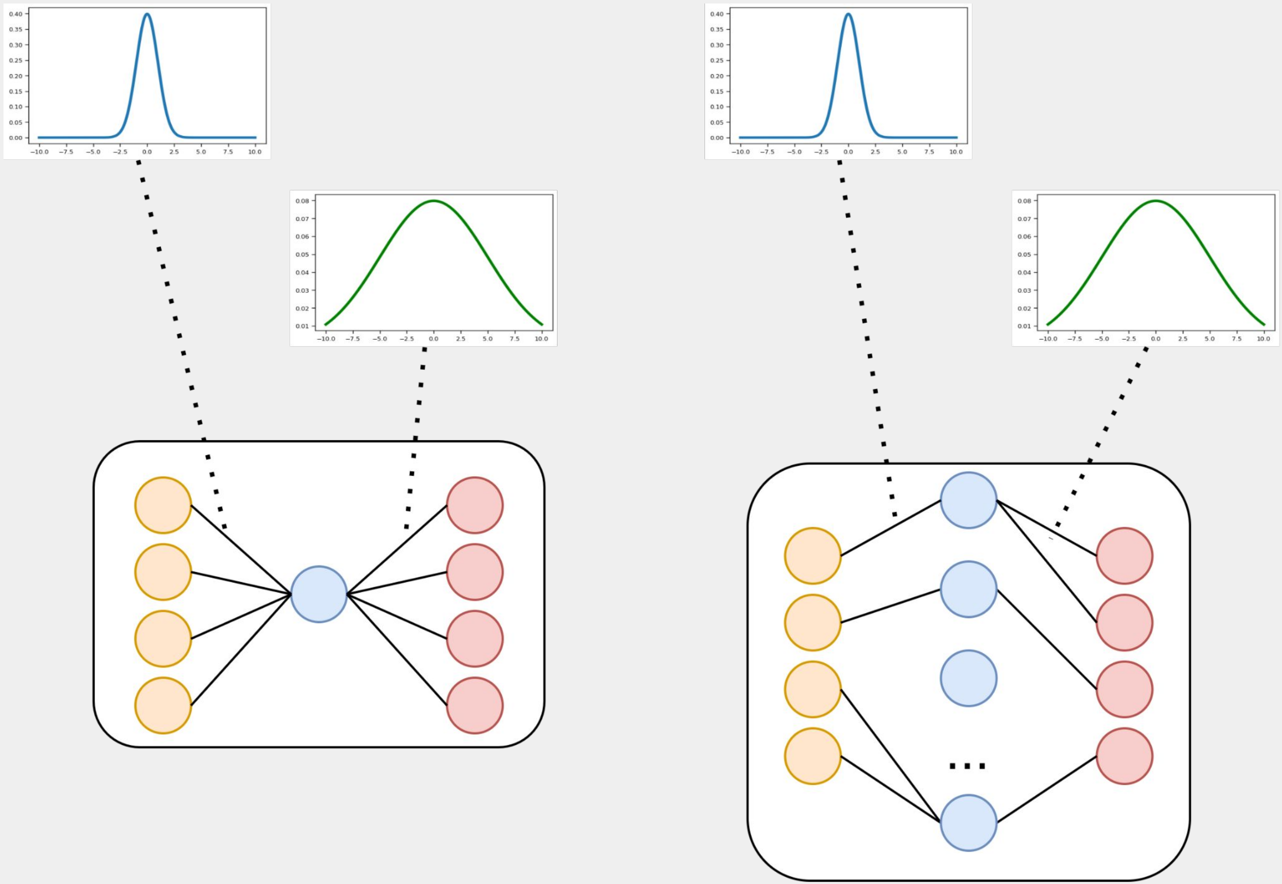
Sparse Network

Same Initial Dist

$$a_S^l \sim P^l \quad , \quad a_D^l \sim P^l, \quad \text{for} \quad l = 1, \ldots, L \quad ,$$

Similar initialization was done in [9,20].

Same Initial Dist

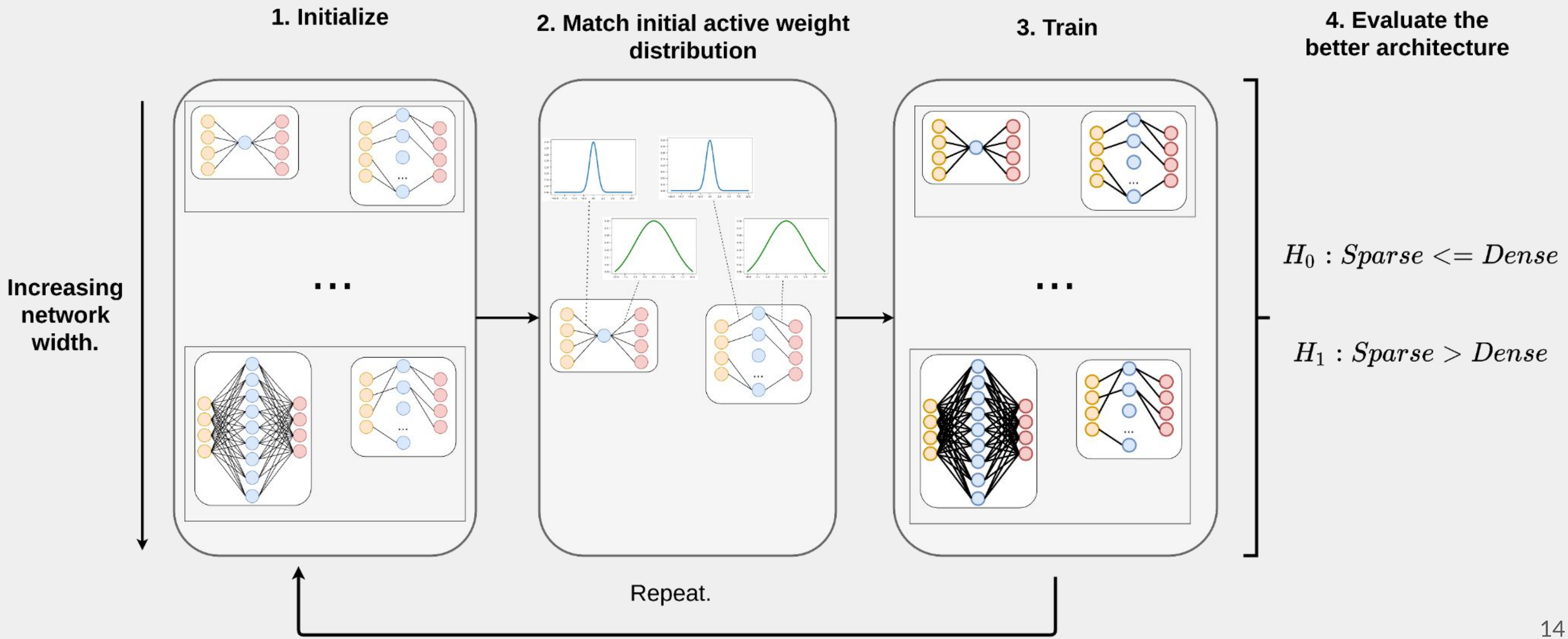


Same Capacity Sparse vs Dense Comparison (SC-SDC)

High level steps:

1. Initialize sparse network S and dense network D to have same capacity (same parameter count and depth).
2. Match active weight distributions.
3. Train till convergence.
4. Evaluate the better architecture.

Same Capacity Sparse vs Dense Comparison (SC-SDC)



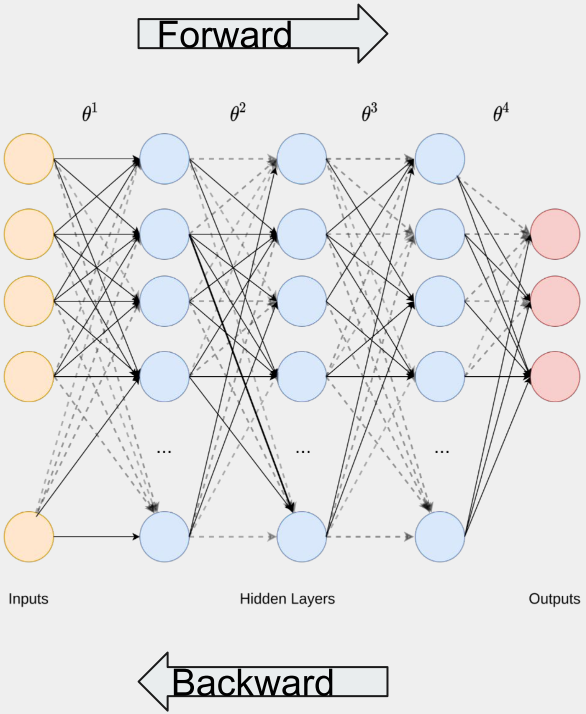
Gradient Flow

- Historically, **exploding** and **vanishing** gradients were a common problem in neural networks.

Gradient Flow - Sparse networks.

- **Exasperated** issue in sparse networks. [12,18]
- Therefore useful analysis tool for studying sparse network optimization.

Intuition.



Standard Gradient Flow

- Gradient flow ≈ norm of the gradients of network.
- We consider a feedforward neural network: $f : \mathbb{R}^D \to \mathbb{R}$, with weights θ and cost function **C**.
- Concatenate all the gradients into a single vector:

$$g = \frac{\partial \mathcal{C}}{\partial \theta}$$

- Take the pth-norm: $gf_p = ||g||_p$

Example: L2 norm of gradients - gf_2

Issues

1. If you don’t mask the gradients -> **gradients of masked weights included in formulation**.
2. Computing gradient norm by concatenating all the gradients into a single vector **gives disproportionate influence to layers with more weights.**

1. Masked Weights != Masked Gradients

$$\theta^l = \theta^l \circ m^l$$

$$\begin{bmatrix} \theta^1_{1,1} & 0 & \theta^1_{1,3} & 0 \\ 0 & \theta^1_{2,2} & 0 & 0 \\ \theta^1_{3,1} & \theta^1_{3,2} & \theta^1_{3,3} & \theta^1_{3,4} \end{bmatrix} = \begin{bmatrix} \theta^1_{1,1} & \theta^1_{1,2} & \theta^1_{1,3} & \theta^1_{1,4} \\ \theta^1_{2,1} & \theta^1_{2,2} & \theta^1_{2,3} & \theta^1_{2,4} \\ \theta^1_{3,1} & \theta^1_{3,2} & \theta^1_{3,3} & \theta^1_{3,4} \end{bmatrix} \circ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

1. Masked Weights != Masked Gradients

$$\theta^l$$

$\theta^1_{1,1}$	0	$\theta^1_{1,3}$	0
0	$\theta^1_{2,2}$	0	0
$\theta^1_{3,1}$	$\theta^1_{3,2}$	$\theta^1_{3,3}$	$\theta^1_{3,4}$

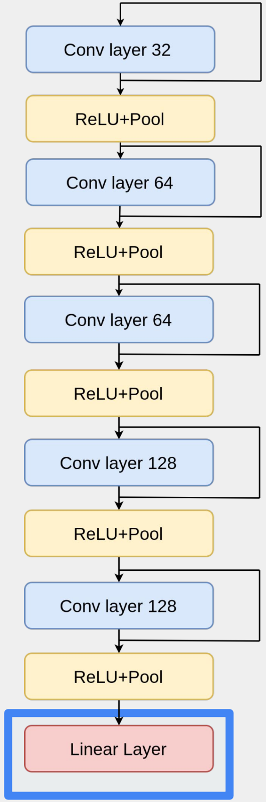
Not necessarily 0.

$$\frac{\partial \mathcal{C}}{\partial \theta^l}$$

$\frac{\partial \mathcal{C}}{\partial \theta^1_{1,1}}$		$\frac{\partial \mathcal{C}}{\partial \theta^1_{1,3}}$	
	$\frac{\partial \mathcal{C}}{\partial \theta^1_{2,2}}$		
$\frac{\partial \mathcal{C}}{\partial \theta^1_{3,1}}$	$\frac{\partial \mathcal{C}}{\partial \theta^1_{3,2}}$	$\frac{\partial \mathcal{C}}{\partial \theta^1_{3,3}}$	$\frac{\partial \mathcal{C}}{\partial \theta^1_{3,4}}$

2. Disproportionate influence to layers with more weights.

Simple CNN



Linear Layer - Majority of the weights and -> disproportionate impact on gradient norm.

Effective Gradient Flow (EGF)

$$g = (\frac{\partial \mathcal{C}}{\partial \theta^1} \odot m^1, \frac{\partial \mathcal{C}}{\partial \theta^2} \odot m^2, \ldots, \frac{\partial \mathcal{C}}{\partial \theta^L} \odot m^L)$$

$$EGF_p = \frac{\sum_{n=1}^{L} ||g_n||_p}{L}$$

Compare GF -> EGF

- We train 600 MLPs for 500 epochs on Fashion-MNIST
- More than 10 000 MLPs for 1000 epochs on CIFAR-10 and CIFAR-100.

MLP - Correlation Between Gradient Flow Measures and Generalization Performance

	Measure	Sparse		Dense	
		Test Loss	Test Accuracy	Test Loss	Test Accuracy
FMNIST	$\|\|\boldsymbol{g}\|\|_1$	0.355	0.316	**0.365**	**0.354**
	$\|\|\boldsymbol{g}\|\|_2$	0.282	0.292	0.285	0.329
	EGF_1	**0.419**	**0.373**	**0.365**	**0.354**
	EGF_2	0.360	0.323	0.298	0.320
CIFAR-10	$\|\|\boldsymbol{g}\|\|_1$	0.440	0.327	**0.380**	0.251
	$\|\|\boldsymbol{g}\|\|_2$	0.447	0.308	0.355	**0.290**
	EGF_1	0.371	0.300	**0.380**	0.252
	EGF_2	**0.451**	**0.332**	0.363	0.287
CIFAR-100	$\|\|\boldsymbol{g}\|\|_1$	0.355	0.385	0.325	0.319
	$\|\|\boldsymbol{g}\|\|_2$	0.373	0.393	0.357	**0.385**
	EGF_1	0.358	0.320	0.325	0.319
	EGF_2	**0.402**	**0.396**	**0.359**	0.382

Potential Use Cases for EGF

- More accurate analysis of sparse gradient flow
- Possibility for Application in Gradient-based Pruning Methods
    - Gradient-based pruning methods like GRASP and SNIP have been to be susceptible to layer-collapse -> maybe EGF can help?

Results - SC-SDC and EGF

Configuration	Variants
Optimizers	Adagrad, Adam, RMSProp, SGD and SGD with mom (0.9).
Regularization/Normalization	No Regularization (NR), Weight Decay (L2), Data Augmentation (DA), Skip Connections (SC) and BatchNorm (BN).
Number of hidden layers	1, 2 and 4.
Dense Width	308, 923, 1538, 2153 and 2768.
Activation functions	ReLU, PReLU, ELU, Swish, SReLU and Sigmoid.
Learning rate	0.001 and 0.1.
Datasets	Fashion-MNIST, CIFAR-10 and CIFAR-100.

Results - EWMA vs Non-EWMA Optims

Non-EWMA Optims
Adagrad
SGD
SGD + mom (0.9)

EWMA (Exponentially weighted moving average) Optims
RMSProp
Adam

Results - Acronym

NR - No Regularization, BN - Batchnorm, SC - Skip Connections, DA - Data Augmentation, L2- weight decay, D - Dense Networks and S - Sparse Networks.

EGF - Average EGF calculated at the end of 11 epochs, evenly spread throughout the training.

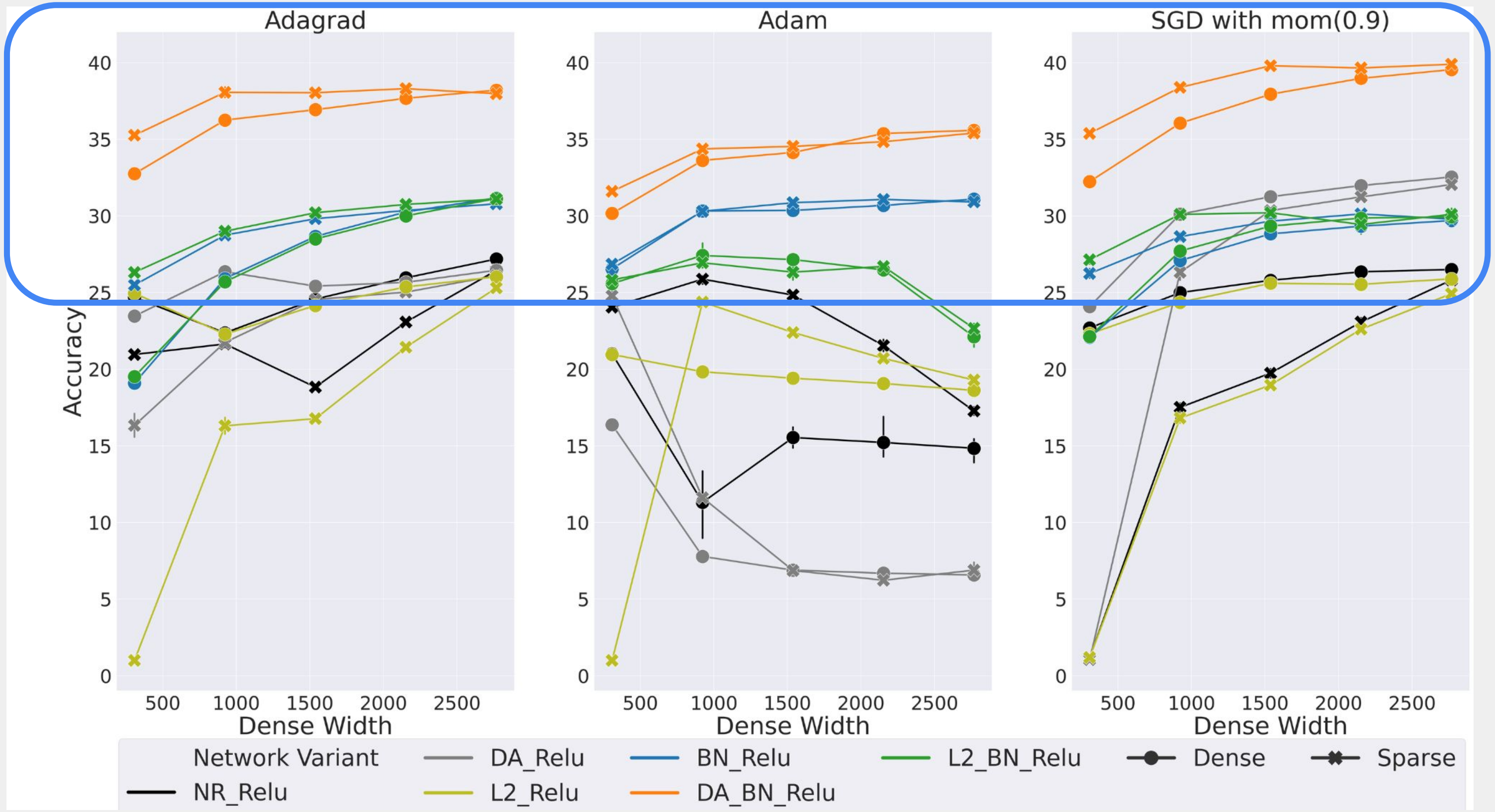
1. Batch Normalization Plays a Disproportionate Role in Stabilizing Sparse Networks

	NR	DA	L2	SC	BN	DA_BN	L2_BN	SC_BN
Adagrad	1.000	1.000	0.998	0.239	**0.006**	**0.002**	**0.001**	**0.003**
Adam	**0.000**	0.055	0.198	**0.003**	0.079	0.051	0.254	0.166
RMSProp	**0.001**	**0.000**	0.300	0.166	0.117	**0.021**	0.914	0.541
SGD	1.000	1.000	1.000	0.248	**0.000**	**0.000**	**0.001**	**0.003**
Mom (0.9)	1.000	1.000	1.000	0.999	**0.001**	**0.000**	**0.007**	**0.008**

Colour Scale based on p-values :

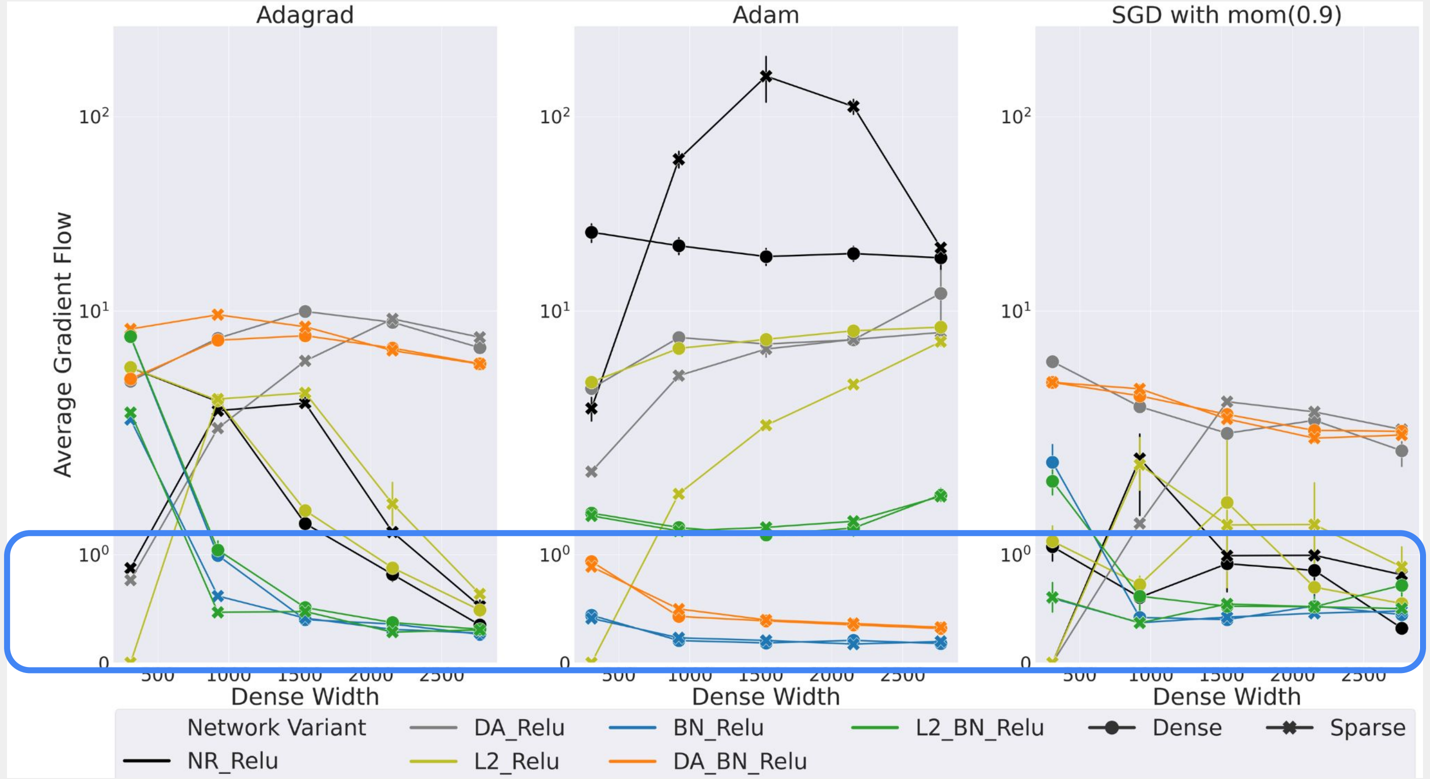
NR - No Regularization, BN - Batchnorm, SC - Skip Connections, DA - Data Augmentation, L2- weight decay, D - Dense Networks and S - Sparse Networks.

Batch Norm Stabilizes Grad Flow - Accuracy - 4hl



Increasing Network Width / Capacity

Batch Norm Stabilizes Grad Flow - Gradient Flow - 4hl



Batch Norm

Activations of a layer.

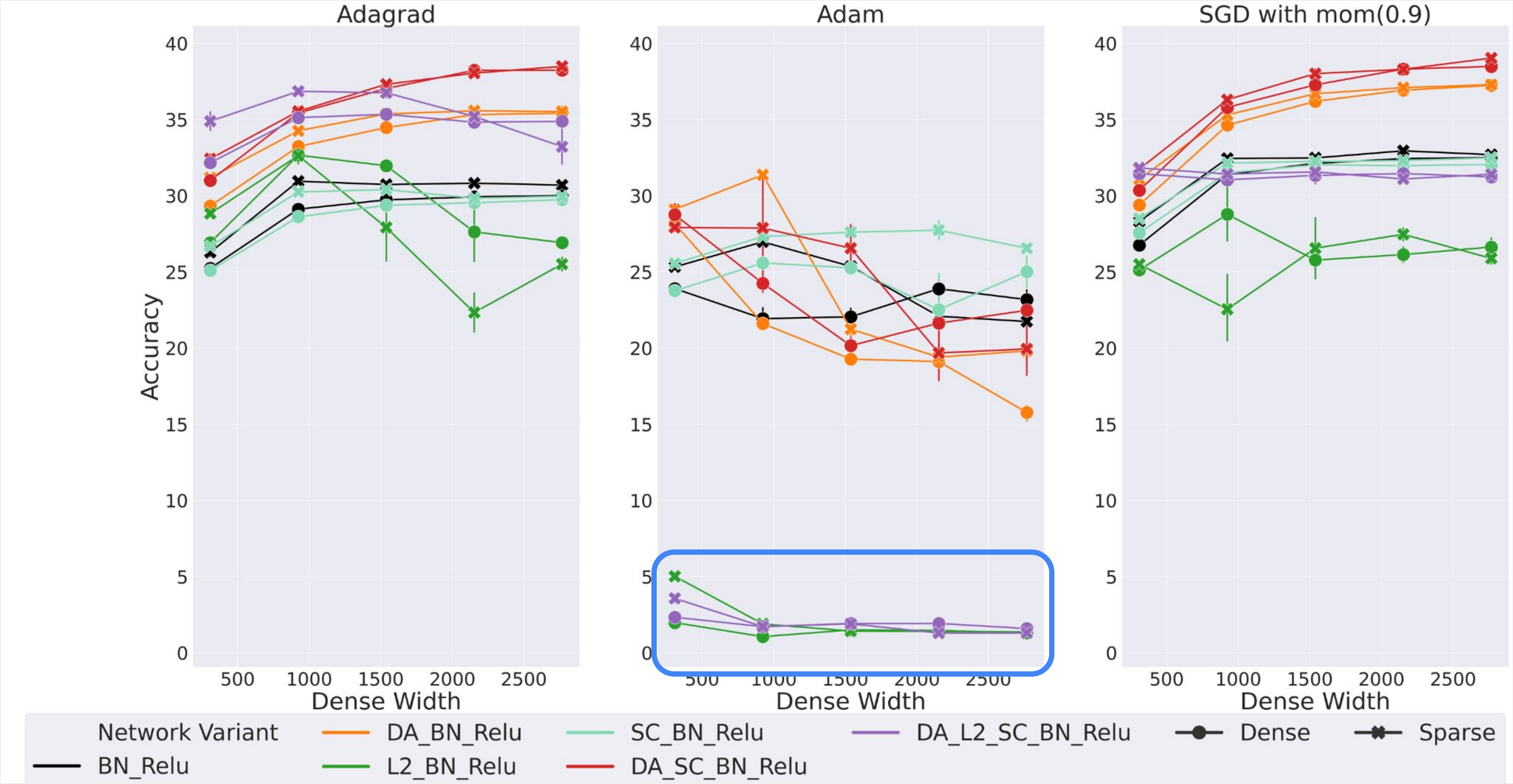
Minibatch Mean.

$$\mathrm{BN}(\boxed{\mathbf{x}}) = \boxed{\gamma} \odot \frac{\boxed{\mathbf{x}} - \boxed{\hat{\mu}}}{\boxed{\hat{\sigma}}} + \boxed{\beta}$$

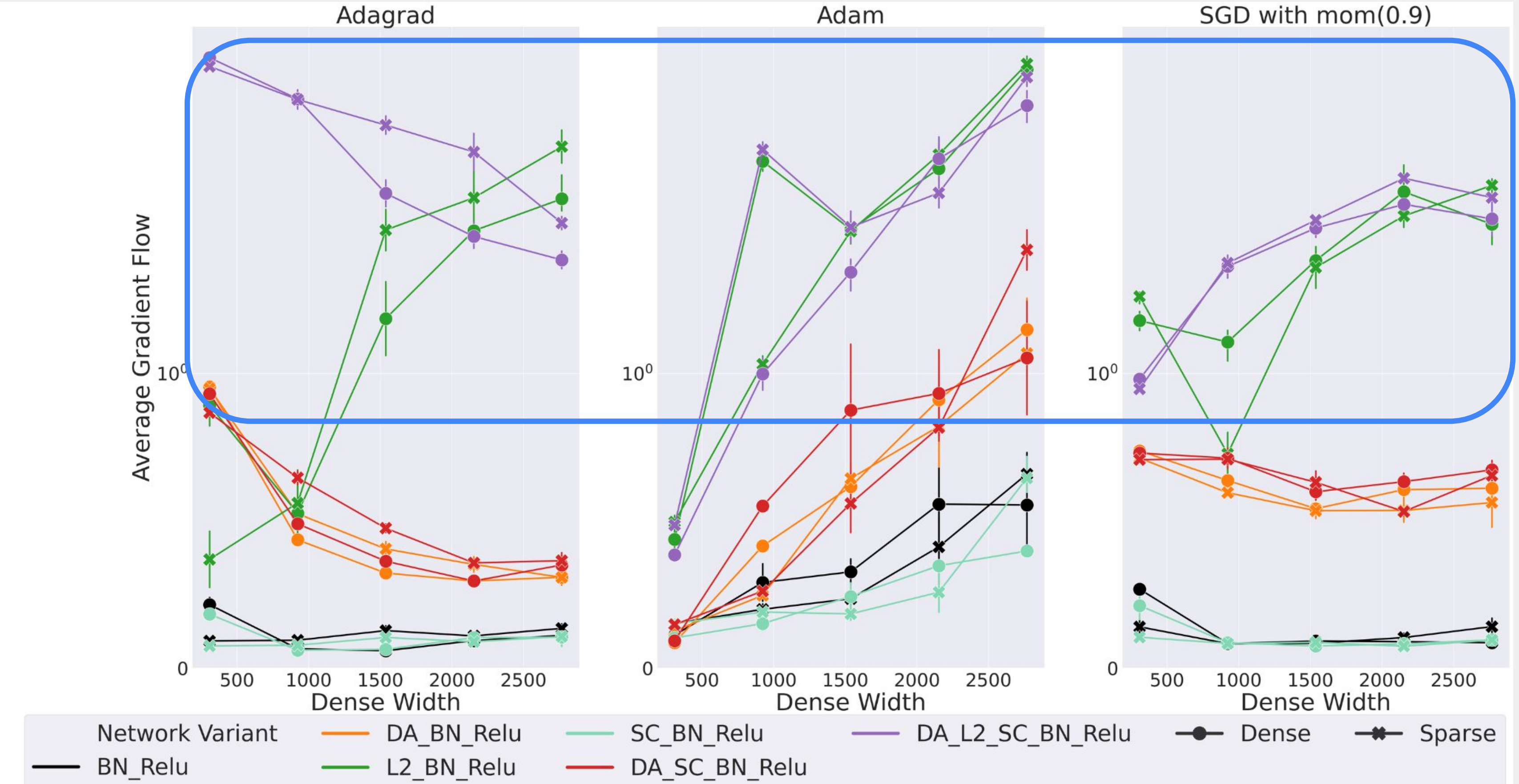
Minibatch Variance.

Learnable Parameters.

2. EWMA Optimizers Are Sensitive to High Gradient Flow



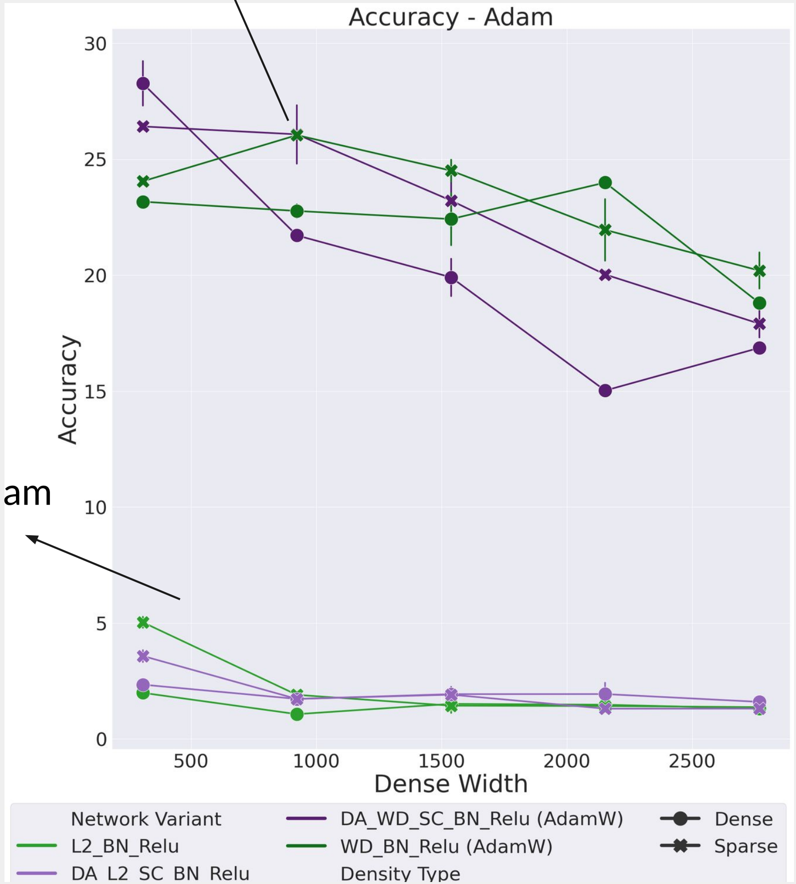
2. EWMA Optimizers Are Sensitive to High Gradient Flow

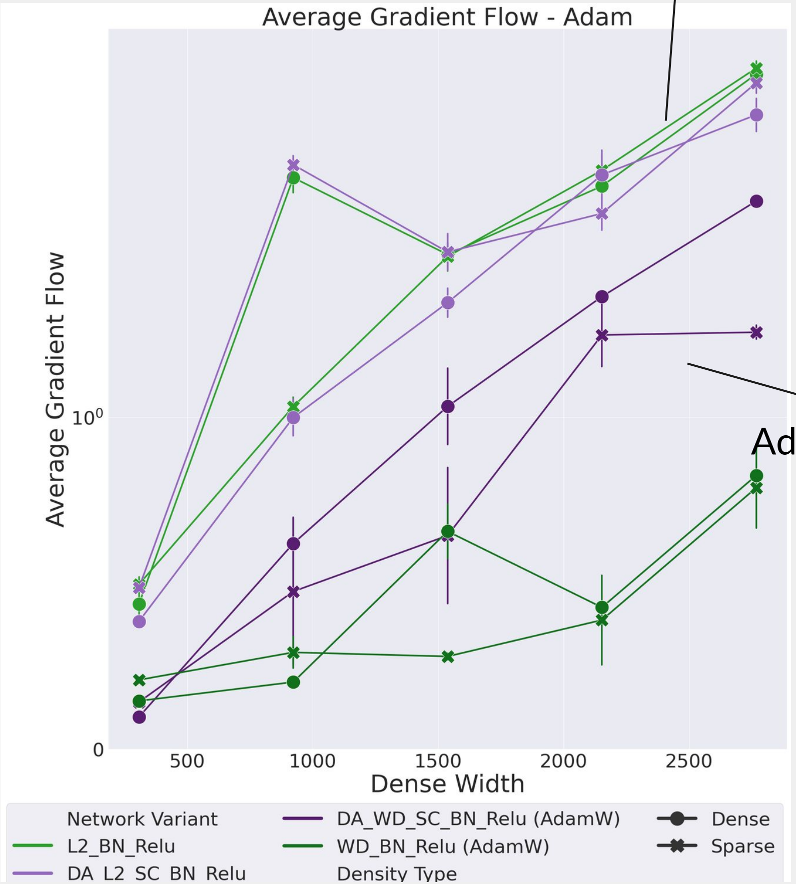


2. 1 Adam vs AdamW

AdamW

Adam





Adam

AdamW

3. Activation Functions

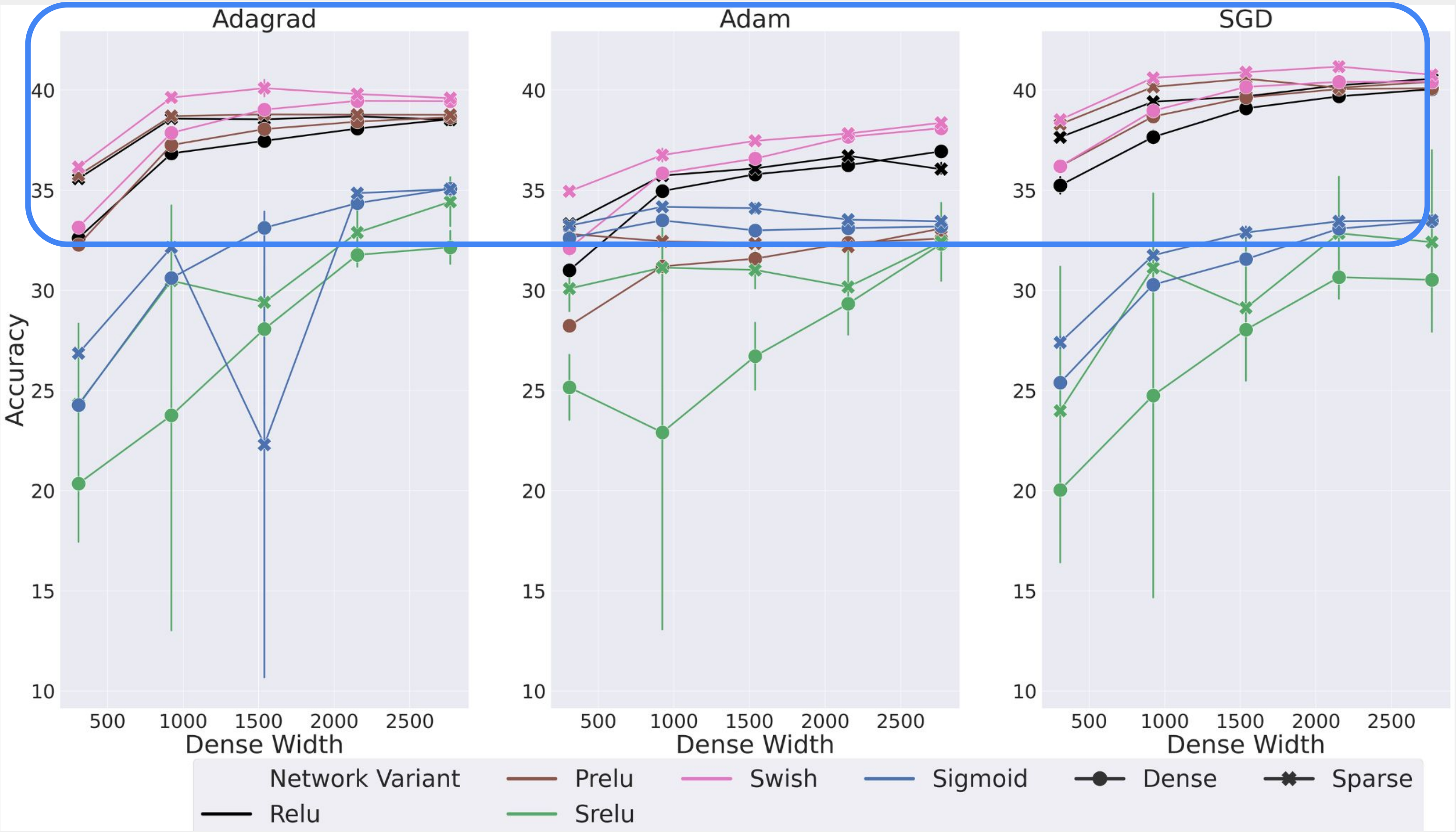
	ReLU	Swish	PReLU	SReLU	Sigmoid	ELU
Adagrad	**0.023**	**0.005**	**0.050**	0.182	0.568	**0.003**
Adam	0.191	0.182	**0.039**	0.062	**0.005**	**0.000**
RMSProp	0.894	0.167	**0.002**	**0.012**	0.997	0.153
SGD	**0.013**	**0.027**	**0.005**	0.078	**0.030**	0.056
Mom (0.9)	0.212	**0.013**	**0.001**	0.078	**0.001**	0.973

Colour Scale based on p-values : 0 .5 1

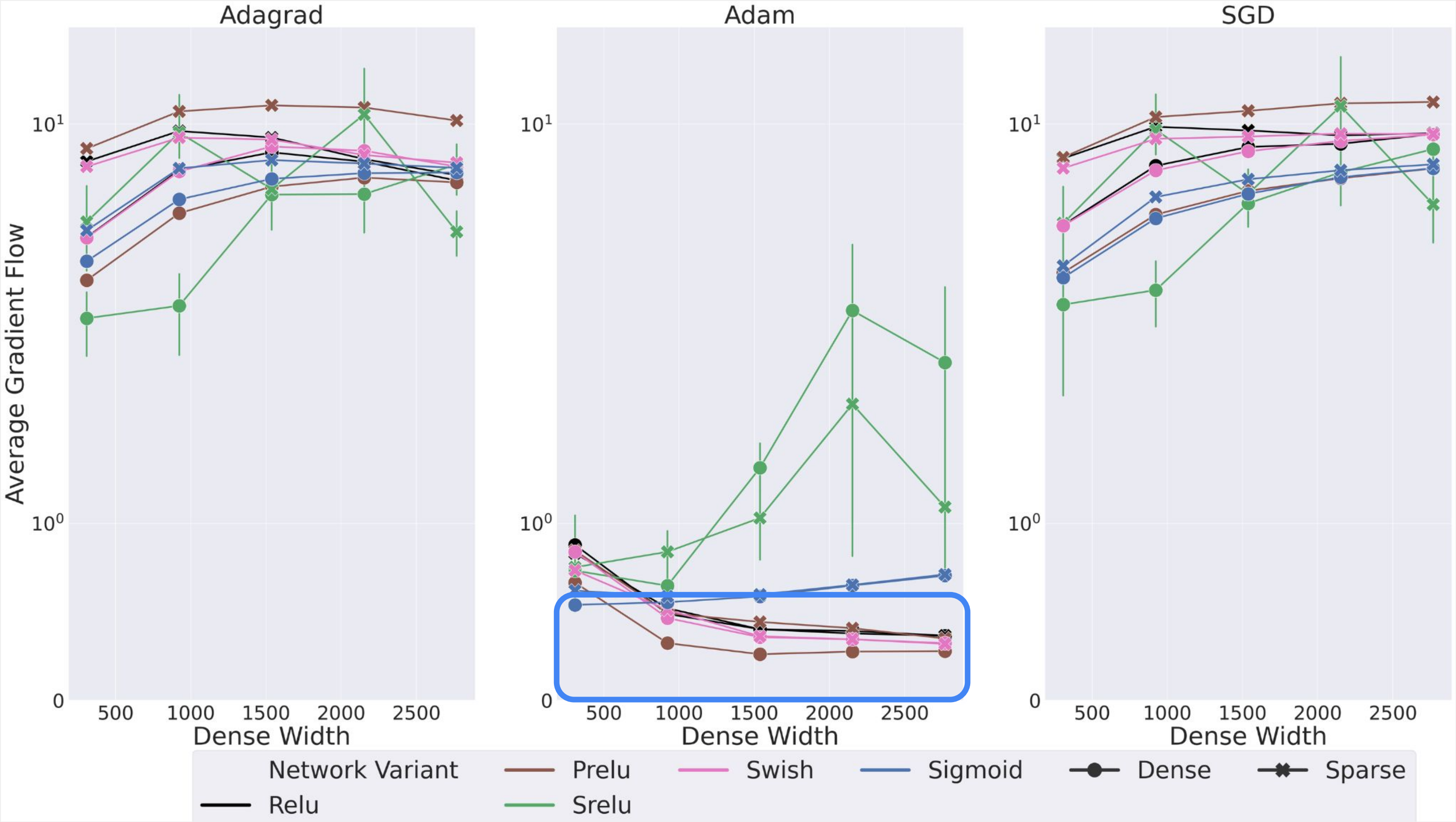
S > D S ≤ D

NR - No Regularization, BN - Batchnorm, SC - Skip Connections, DA - Data Augmentation, L2- weight decay, D - Dense Networks and S - Sparse Networks.

Activation Functions - Accuracy

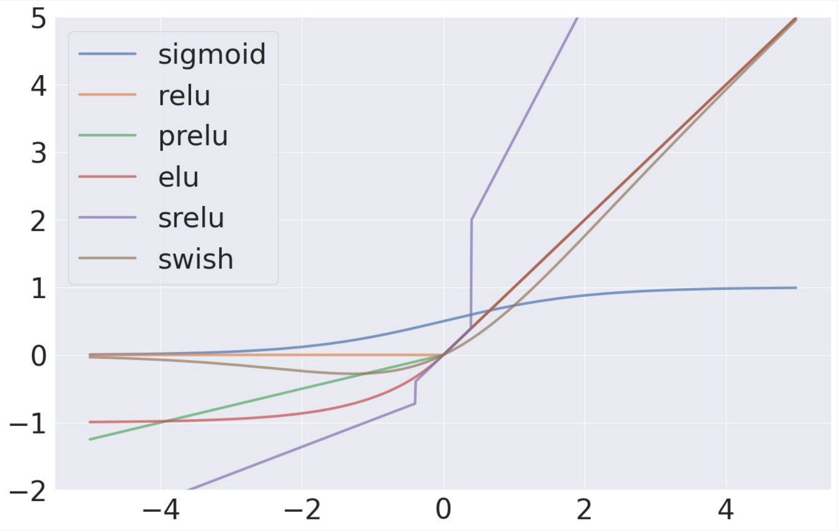


Activation Functions - Gradient Flow

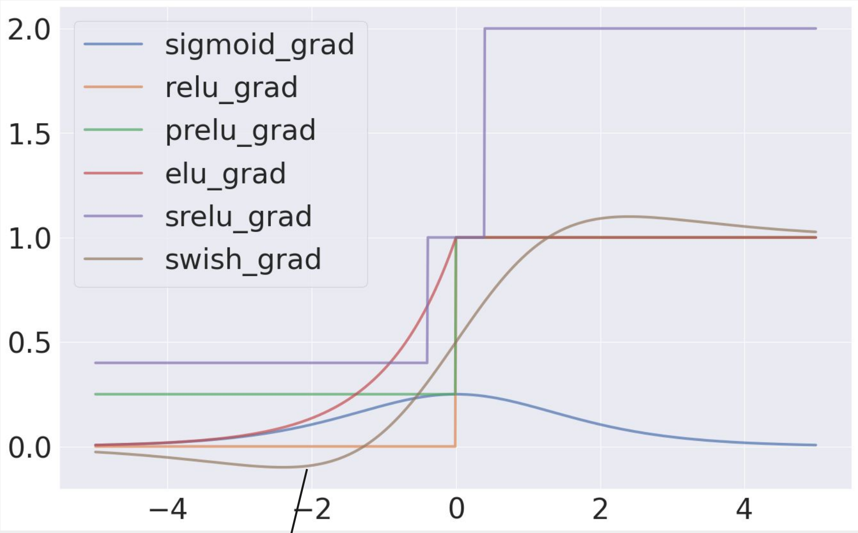


Activation Functions

a) Activation Function with inputs [-5,5]



b) Derivative of Activation Function with inputs [-5,5]



Allows flow of negative gradients.

Extension of Results

- Generalization of Results Across Architecture Types - **Wide ResNet-50.**
- Generalization of Results From Random Pruning to **Magnitude Pruning**.

Questions???

Keep the Gradients Flowing: Using Gradient Flow to study Sparse Network Optimization. Kale-ab Tessera, Sara Hooker, Benjamin Rosman

https://arxiv.org/abs/2102.01670

Key Takeaways:

- **Need better toolbox for sparse network analysis -** SC-SDC and EGF.
- **BatchNorm is useful for stabilizing grad flow -** especially for sparse networks.
- **Move away from maximizing grad flow -> stabilizing gradient flow.**
- **Careful choice of optims and activation functions** can benefit sparse networks.

kaleabtessera@gmail.com

References

1. Dettmers, T. and Zettlemoyer, L. (2019). Sparse networks from scratch: Faster training without losing performance. arXiv preprint arXiv:1907.04840.
2. Luo, J.-H., Wu, J., and Lin, W. (2017). Thinet: A filter level pruning method for deep neural network compression. In Proceedings of the IEEE international conference on computer vision, pages 5058–5066.
3. Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2016). Pruning Convolutional Neural Networks for Resource Efficient Inference. ArXiv e-prints.
4. Ahmad, S. and Scheinkman, L. (2019). How can we be so dense? the benefits of using highly sparse representations. arXiv preprint arXiv:1903.11257.
5. Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., & Gonzalez, J. (2020, November). Train Big, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers. In International Conference on Machine Learning (pp. 5958-5968). PMLR.
6. Yang, T. J., Chen, Y. H., & Sze, V. (2017). Designing energy-efficient convolutional neural networks using energy-aware pruning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 5687-5695).
7. Frankle, J. and Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In International Conference on Learning Representations.

References

8. Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. (2019b). Stabilizing the lottery ticket hypothesis. arXiv preprint arXiv:1903.01611.

9. Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. (2018c). Rethinking the value of network pruning. arXiv preprint arXiv:1810.05270.

10. Evci, U., Pedregosa, F., Gomez, A., and Elsen, E. (2019). The difficulty of training sparse neural networks. arXiv preprint arXiv:1906.10732.

11. Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. (2020). Pruning neural networks at initialization: Why are we missing the mark?

References

12. Evci, U., Ioannou, Y. A., Keskin, C., and Dauphin, Y. (2020). Gradient flow in sparse neural networks and how lottery tickets win. arXiv preprint arXiv:2010.03533.

13. Evci, U., Pedregosa, F., Gomez, A., and Elsen, E. (2019). The difficulty of training sparse neural networks. arXiv preprint arXiv:1906.10732.

14. Lee, N., Ajanthan, T., Gould, S., and Torr, P. H. (2019). A signal propagation perspective for pruning neural networks at initialization. arXiv preprint arXiv:1906.06307.

15. Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. (2019a). Linear mode connectivity and the lottery ticket hypothesis. arXiv preprint arXiv:1912.05671.

16. Lee, N., Ajanthan, T., and Torr, P. H. (2018). Snip: Single-shot network pruning based on connection sensitivity. arXiv preprint arXiv:1810.02340.

17. Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. (2020). Pruning neural networks without any data by iteratively conserving synaptic flow. arXiv preprint arXiv:2006.05467.

References

18. Wang, C., Zhang, G., and Grosse, R. (2020a). Picking winning tickets before training by preserving gradient flow. arXiv preprint arXiv:2002.07376.

19. Su, J., Chen, Y., Cai, T., Wu, T., Gao, R., Wang, L., and Lee, J. D. (2020). Sanity-checking pruning methods: Random tickets can win the jackpot. arXiv preprint arXiv:2009.11094.

20. Gale, T., Elsen, E., and Hooker, S. (2019). The State of Sparsity in Deep Neural Networks.arXiv e-prints, page arXiv:1902.09574.