Deep Learning (1470)

Randall Balestriero

Class 10: Resnets, batch-normalization and Dropout

Student Driven Recap!

Student Driven Recap!

- Difference between convolution and cross-correlation

Student Driven Recap!

- Difference between convolution and cross-correlation
- Output shape for image (5,5) and filter (3,3), padding=0, stride=1

Student Driven Recap!

- Difference between convolution and cross-correlation
- Output shape for image (5,5) and filter (3,3), padding=0, stride=1
- Output shape for image (5,5) and filter (3,3), padding=1, stride=1

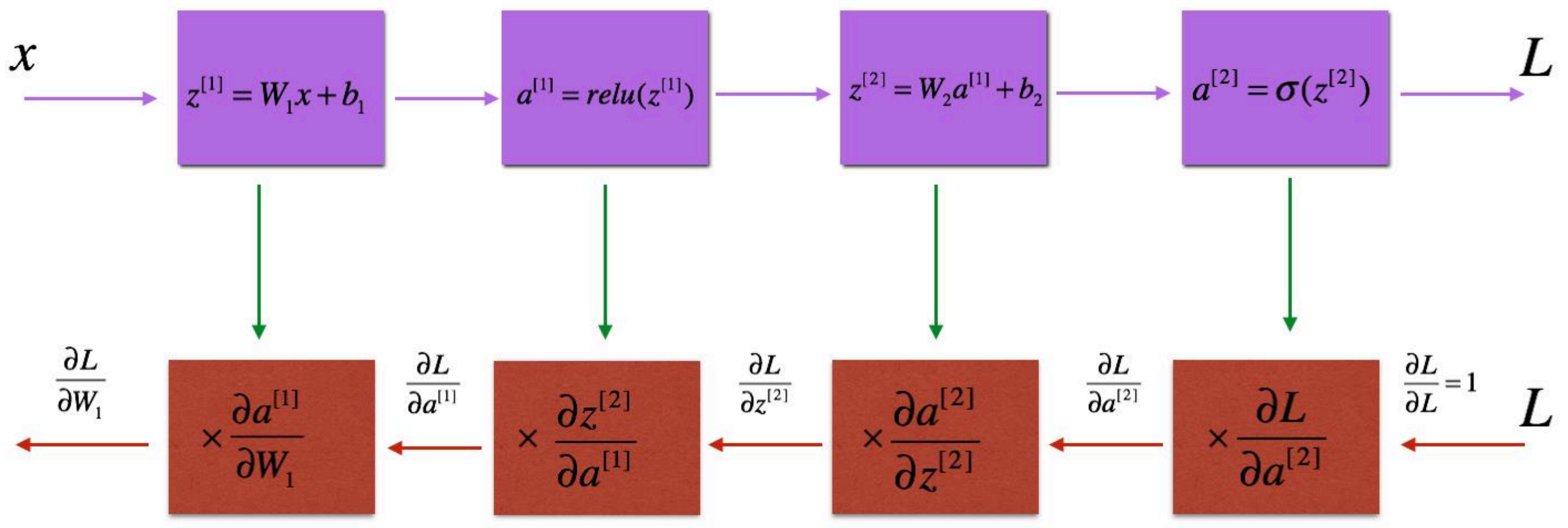
Student Driven Recap!

- Difference between convolution and cross-correlation
- Output shape for image (5,5) and filter (3,3), padding=0, stride=1
- Output shape for image (5,5) and filter (3,3), padding=1, stride=1
- Output shape for image (5,5) and filter (1,1), padding=0, stride=1

Student Driven Recap!

- Difference between convolution and cross-correlation
- Output shape for image (5,5) and filter (3,3), padding=0, stride=1
- Output shape for image (5,5) and filter (3,3), padding=1, stride=1
- Output shape for image (5,5) and filter (1,1), padding=0, stride=1
- Difference between MLP and LeNet5?

Vanishing gradients



x
$z^{[1]} = W_1 x + b_1$
$a^{[1]} = relu(z^{[1]})$
$z^{[2]} = W_2 a^{[1]} + b_2$
$a^{[2]} = \sigma(z^{[2]})$
L
$\frac{\partial L}{\partial W_1}$
$\times \frac{\partial a^{[1]}}{\partial W_1}$
$\frac{\partial L}{\partial a^{[1]}}$
$\times \frac{\partial z^{[2]}}{\partial a^{[1]}}$
$\frac{\partial L}{\partial z^{[2]}}$
$\times \frac{\partial a^{[2]}}{\partial z^{[2]}}$
$\frac{\partial L}{\partial a^{[2]}}$
$\times \frac{\partial L}{\partial a^{[2]}}$
$\frac{\partial L}{\partial L} = 1$
L

$$\boxed{\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial W_1}}$$

≤ 1

Multiplying by terms ≤ 1 makes things smaller...
Gradients earlier in the network tend to "Vanish"

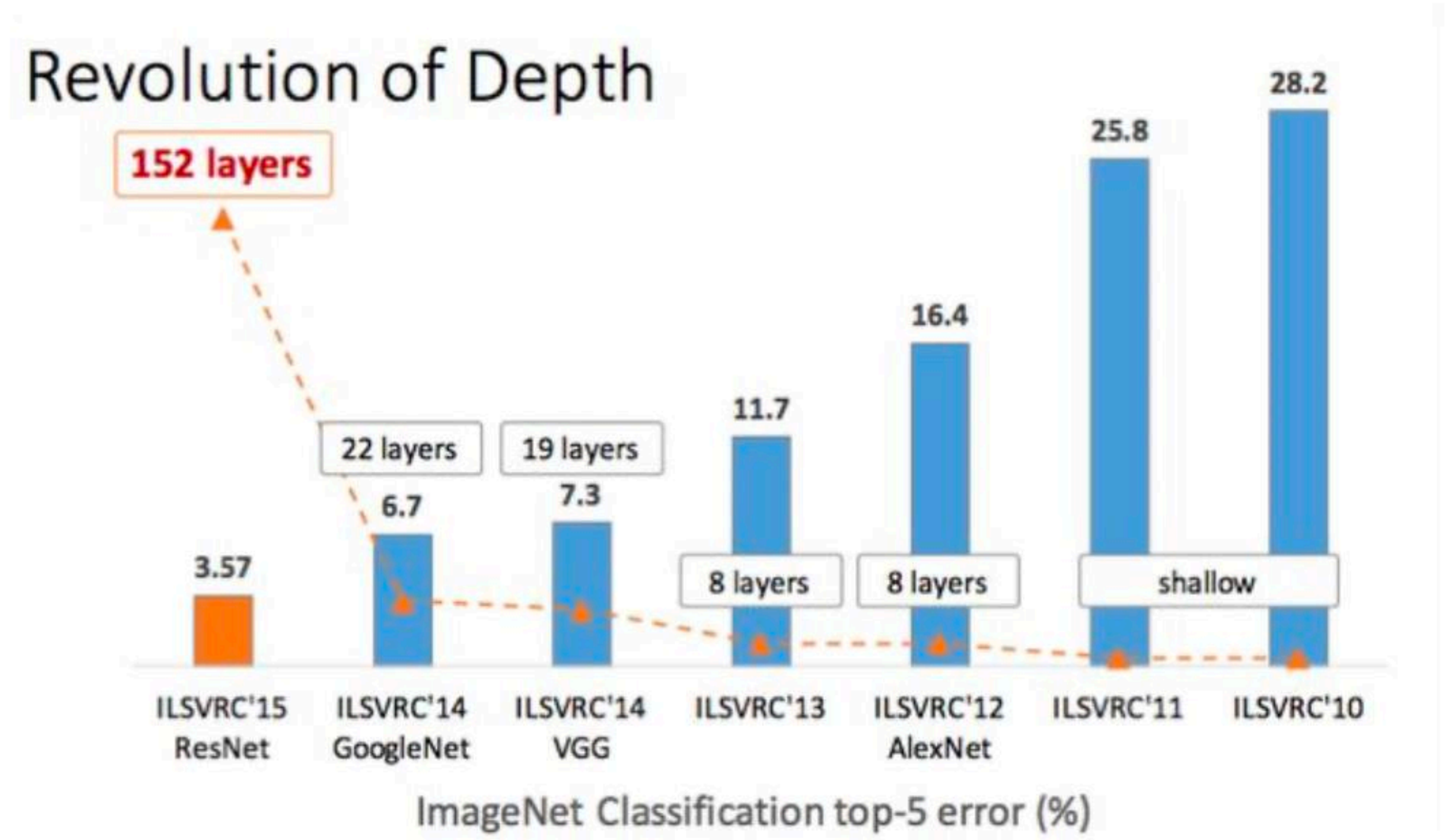
Adding more layers adds more terms with gradient ≤ 1

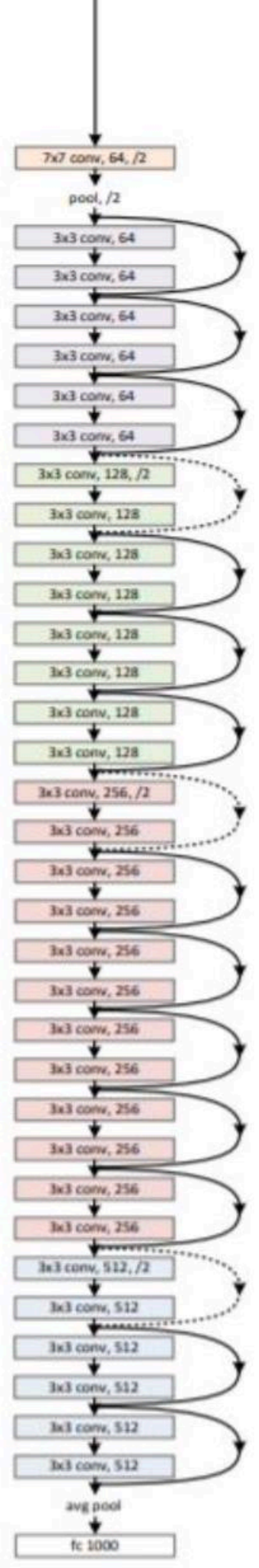
More Complicated Networks

ResNet:

Lots of layers, tons of learnable parameters

Avoids Vanishing Gradient problem
but how?





K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

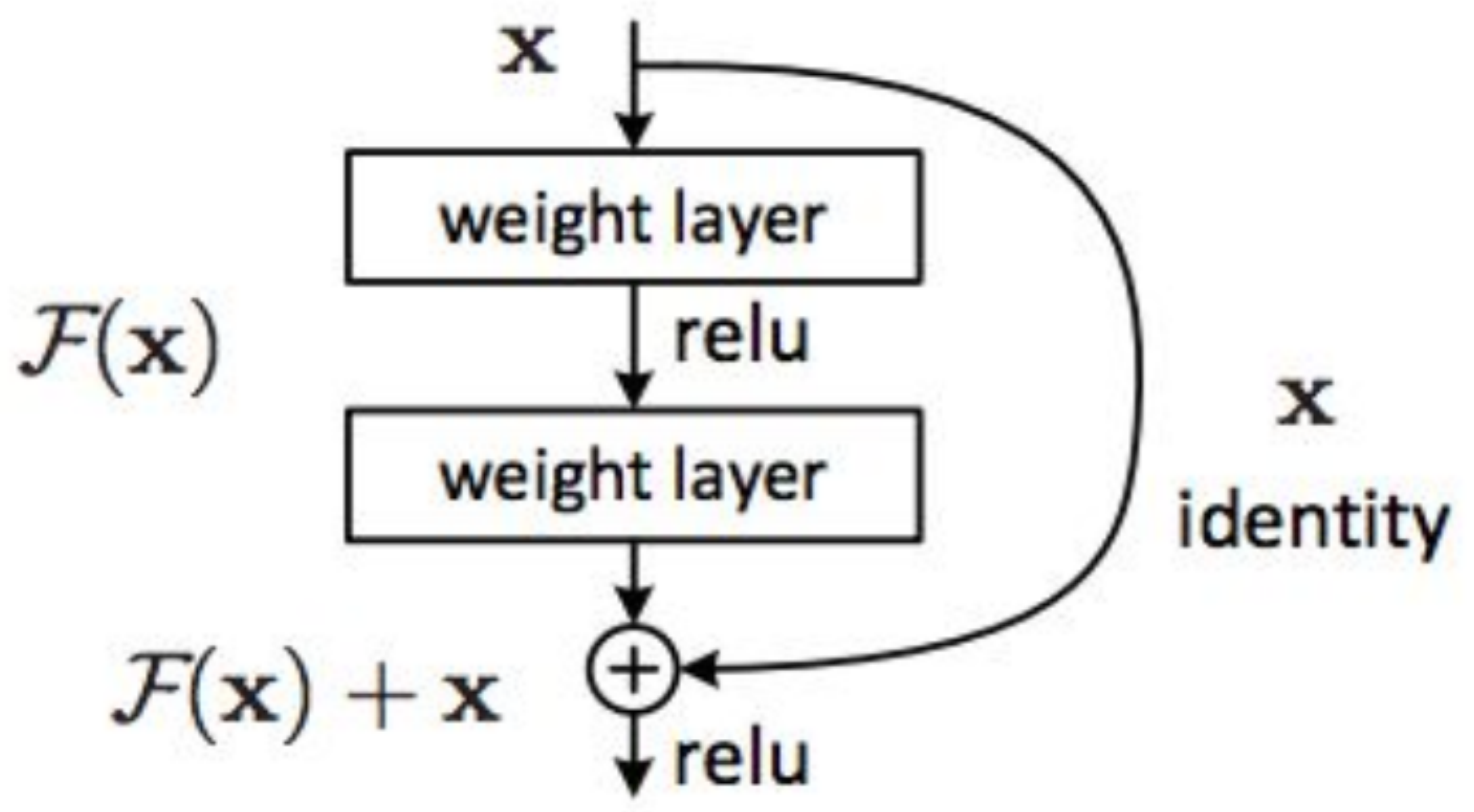
More Complicated Networks

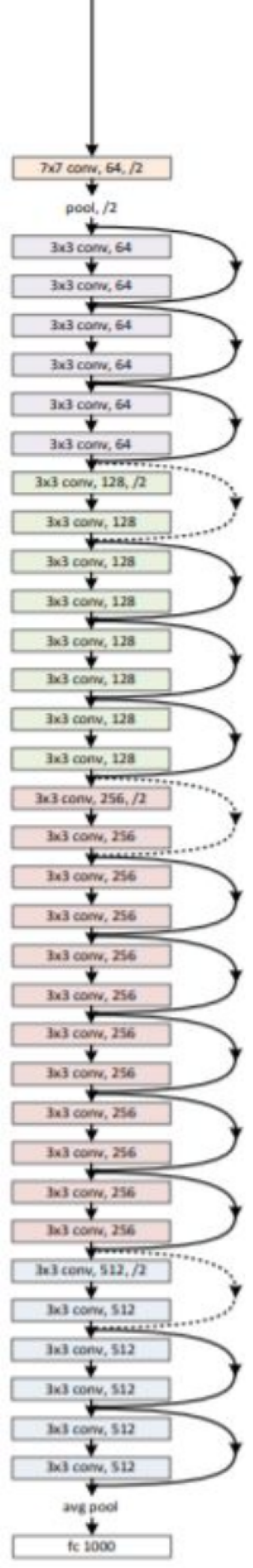
ResNet:

Lots of layers, tons of learnable parameters

Avoids Vanishing Gradient problem

Residual Block →



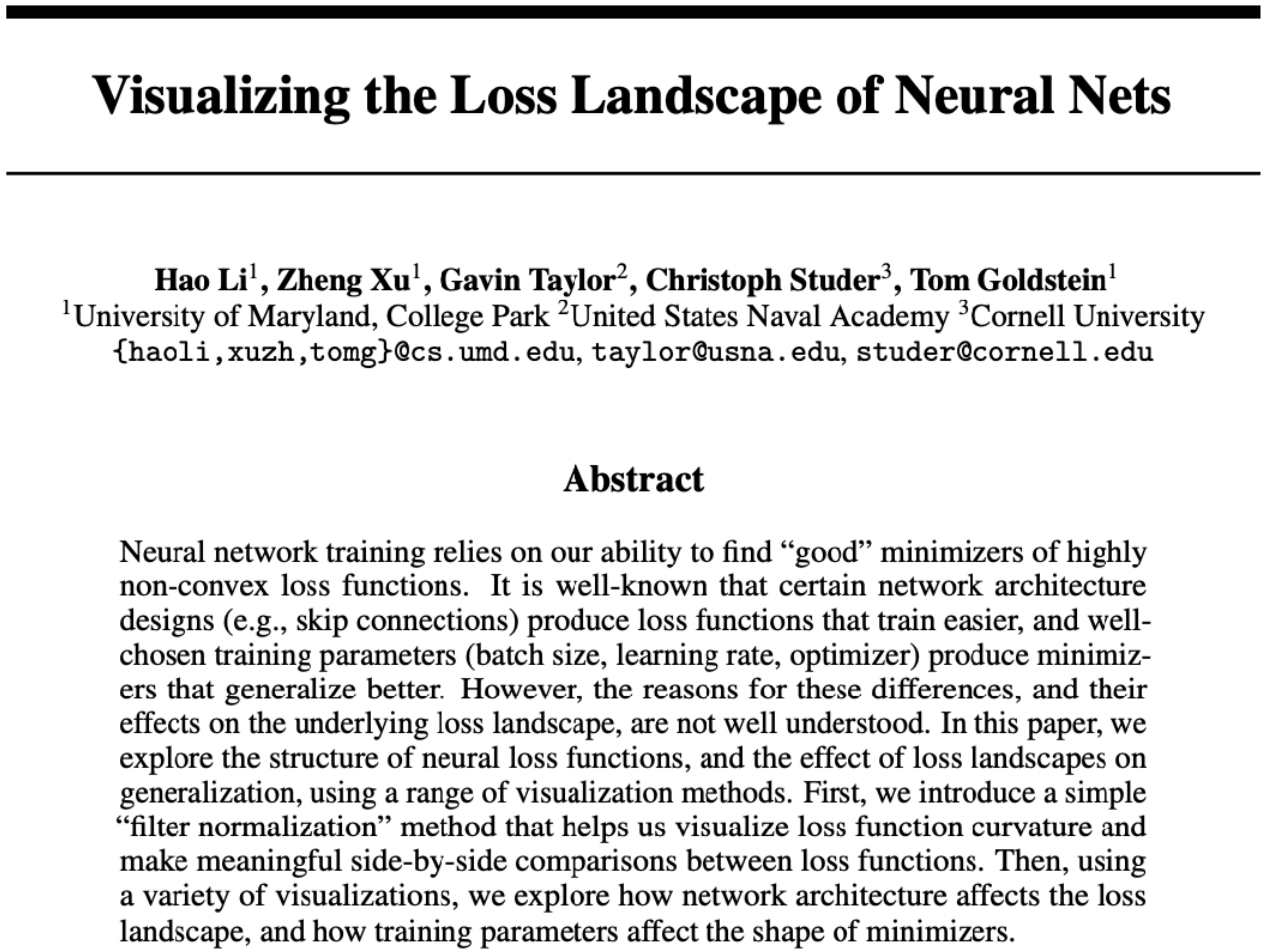


K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

Derive the chain rule for a simple residual model and explain why it helps

Home Exercise

Visualizing the effect of residual connections



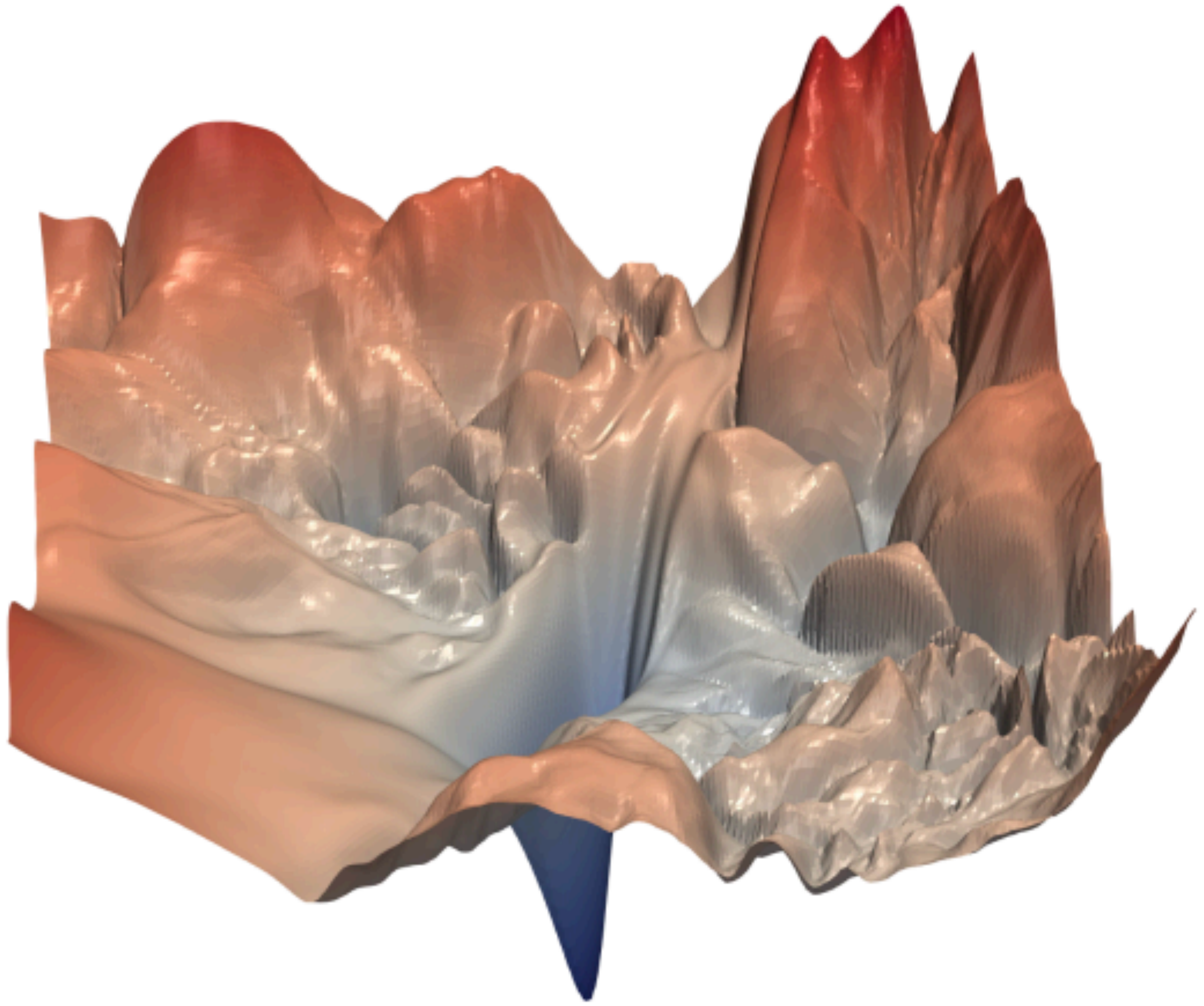
Visualizing the Loss Landscape of Neural Nets

Hao Li[1], **Zheng Xu**[1], **Gavin Taylor**[2], **Christoph Studer**[3], **Tom Goldstein**[1]
[1]University of Maryland, College Park [2]United States Naval Academy [3]Cornell University
`{haoli,xuzh,tomg}@cs.umd.edu`, `taylor@usna.edu`, `studer@cornell.edu`

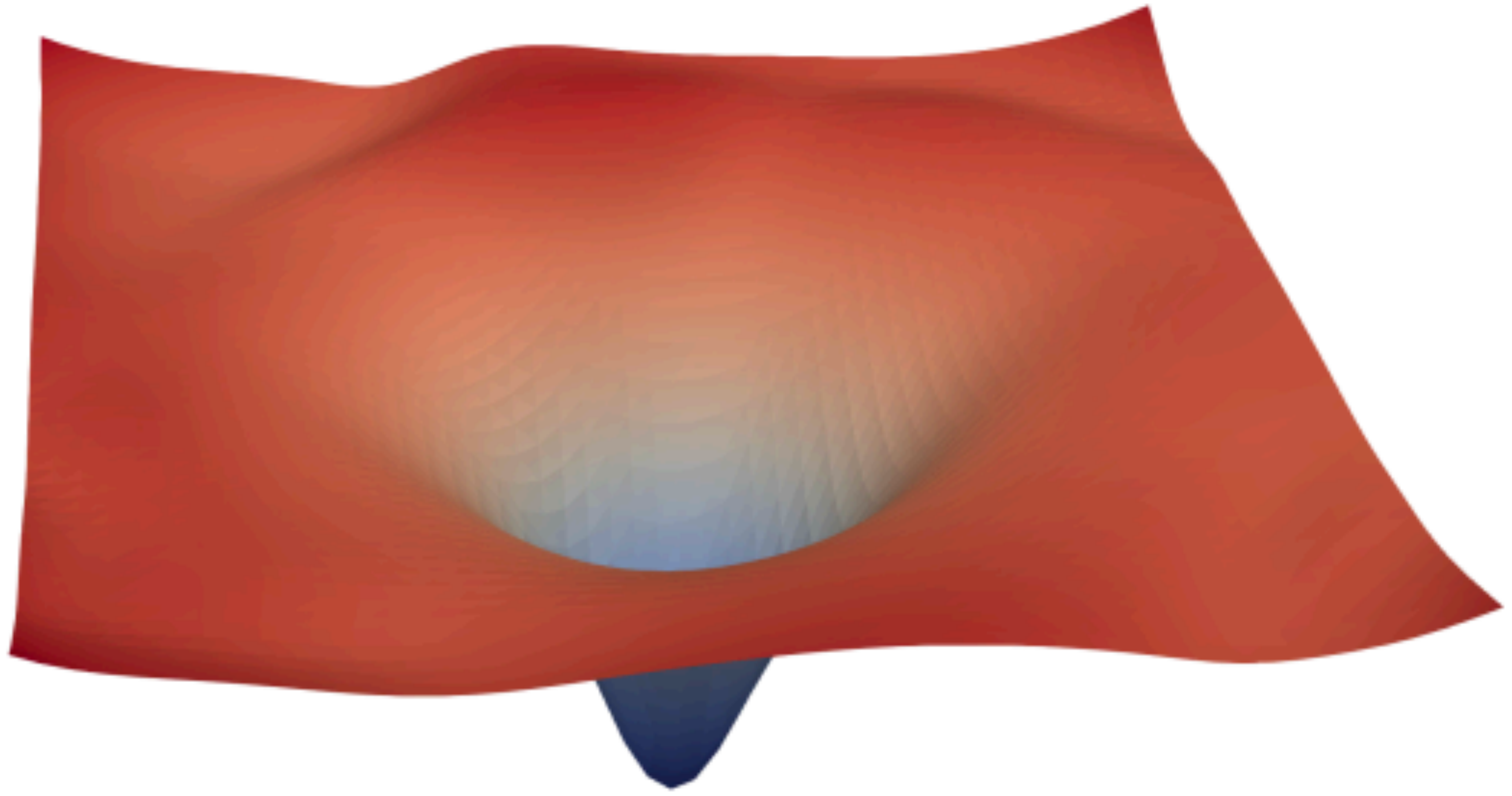
Abstract

Neural network training relies on our ability to find "good" minimizers of highly non-convex loss functions. It is well-known that certain network architecture designs (e.g., skip connections) produce loss functions that train easier, and well-chosen training parameters (batch size, learning rate, optimizer) produce minimizers that generalize better. However, the reasons for these differences, and their effects on the underlying loss landscape, are not well understood. In this paper, we explore the structure of neural loss functions, and the effect of loss landscapes on generalization, using a range of visualization methods. First, we introduce a simple "filter normalization" method that helps us visualize loss function curvature and make meaningful side-by-side comparisons between loss functions. Then, using a variety of visualizations, we explore how network architecture affects the loss landscape, and how training parameters affect the shape of minimizers.

Visualizing the effect of residual connections



(a) without skip connections



(b) with skip connections

Batch Normalization: Implementation

Normalize by subtracting feature x’s batch mean, then divide by batch standard deviation.

$$x' = \frac{x - \mu_{batch}}{\sigma_{batch}}$$

Feature x now has mean 0 and variance 1 along the batch

Batch Normalization: Implementation

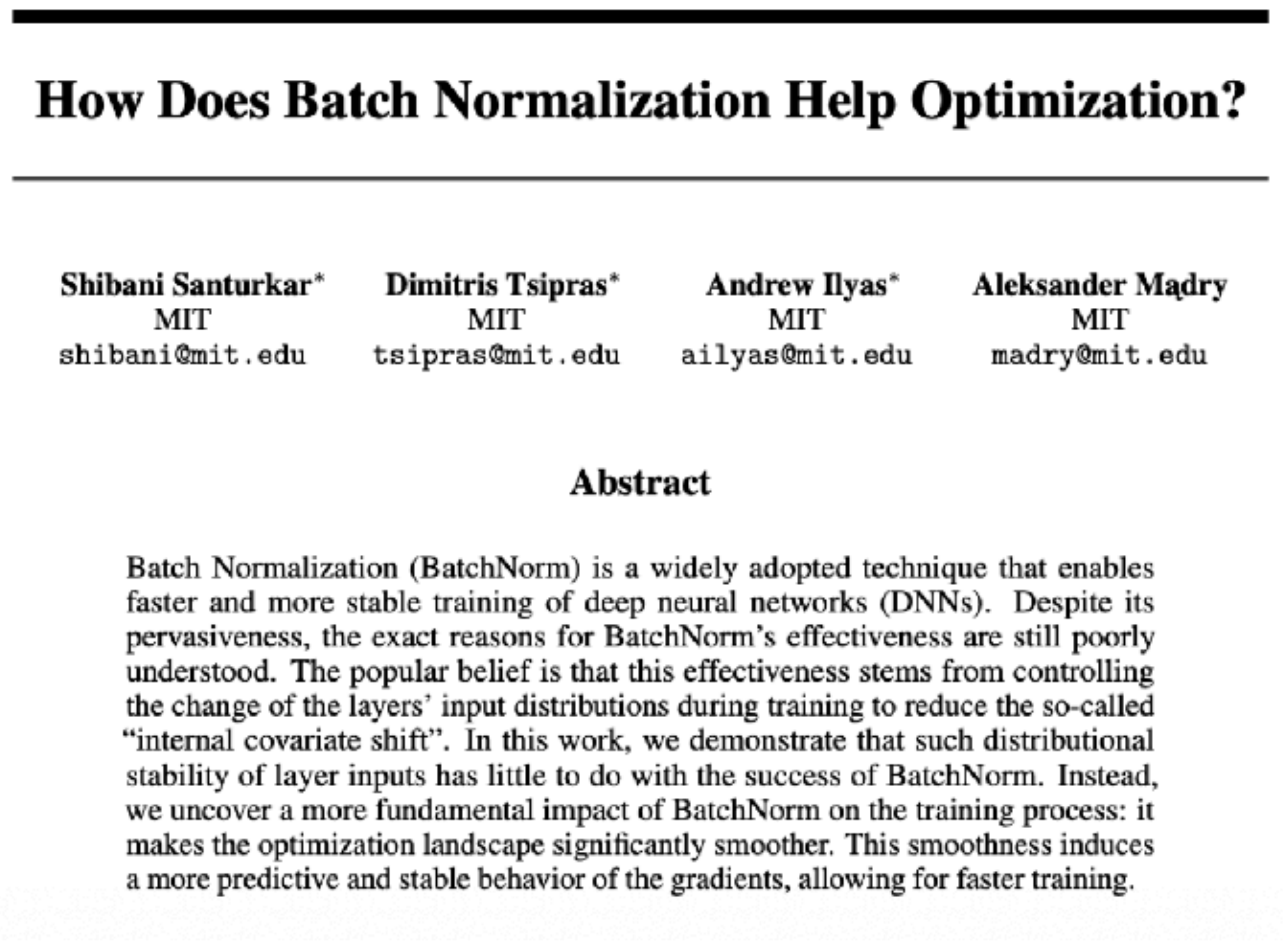
Normalize by subtracting feature x's batch mean, then divide by batch standard deviation.

What do we do at test time?

$$\sigma_{batch}$$

Feature x now has mean 0 and variance 1 along the batch

Batch Normalization Helps in Many Ways

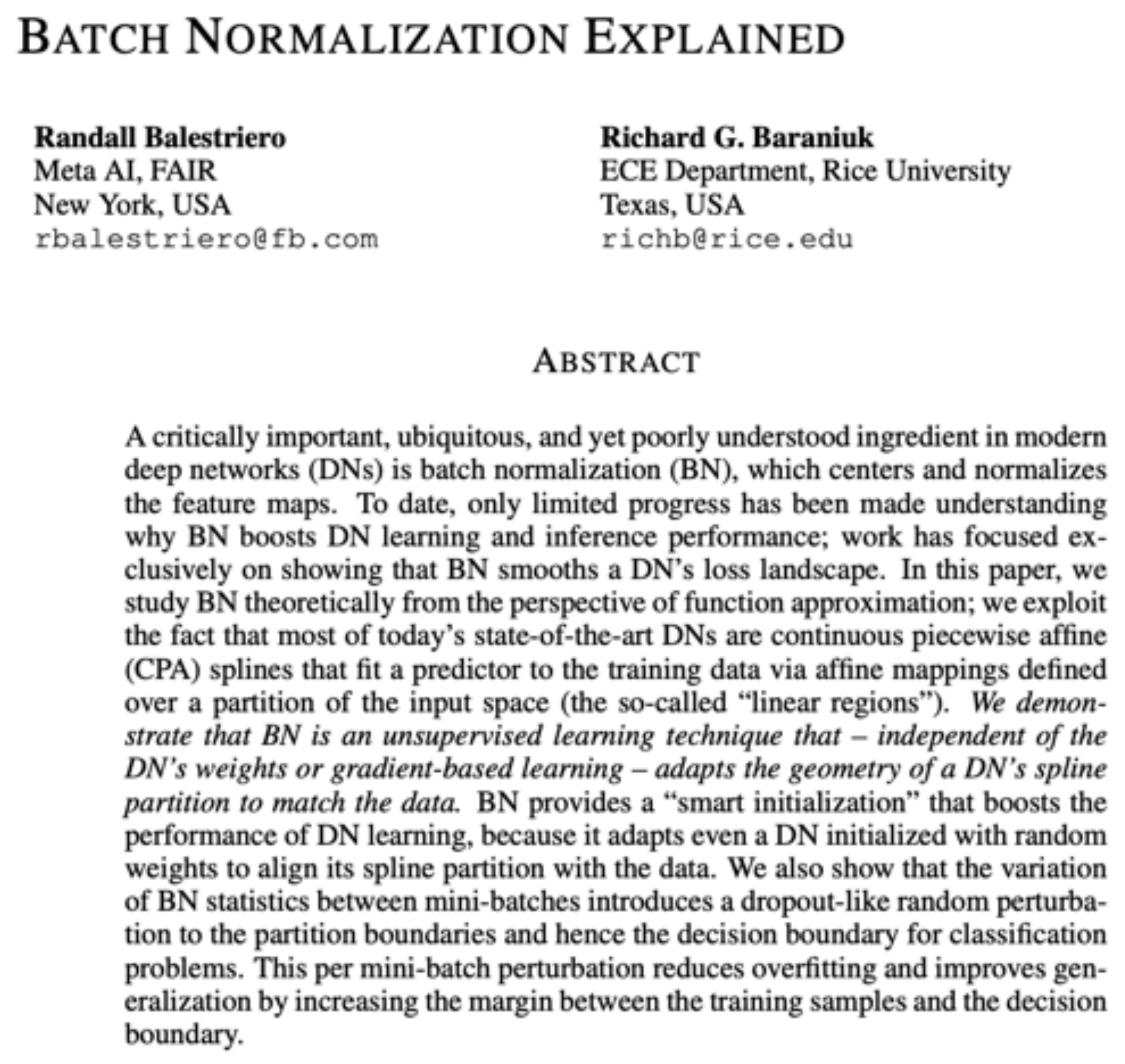


How Does Batch Normalization Help Optimization?

Shibani Santurkar* MIT shibani@mit.edu
Dimitris Tsipras* MIT tsipras@mit.edu
Andrew Ilyas* MIT ailyas@mit.edu
Aleksander Mądry MIT madry@mit.edu

Abstract

Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling the change of the layers' input distributions during training to reduce the so-called "internal covariate shift". In this work, we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm. Instead, we uncover a more fundamental impact of BatchNorm on the training process: it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.



BATCH NORMALIZATION EXPLAINED

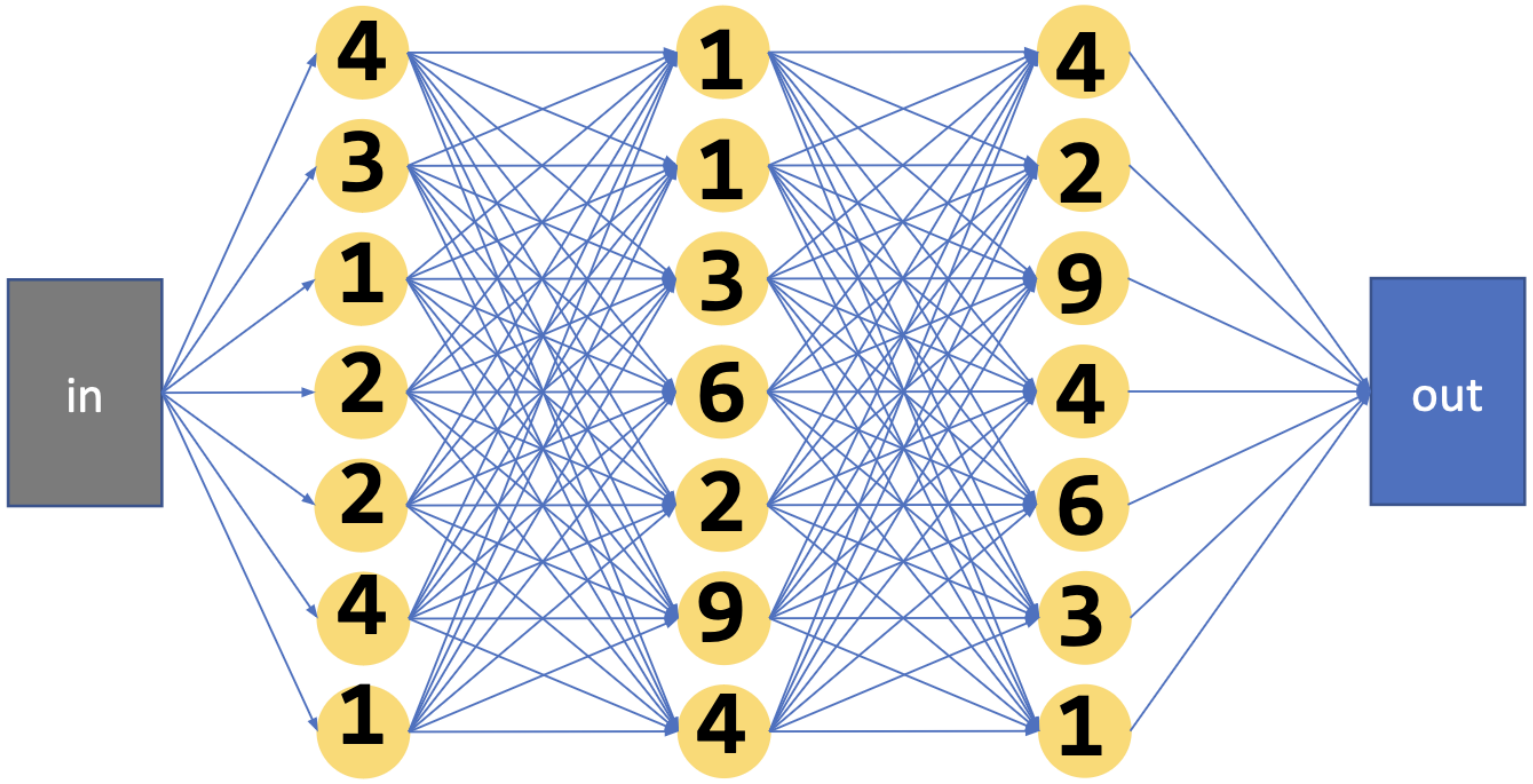
Randall Balestriero
Meta AI, FAIR
New York, USA
rbalestriero@fb.com

Richard G. Baraniuk
ECE Department, Rice University
Texas, USA
richb@rice.edu

ABSTRACT

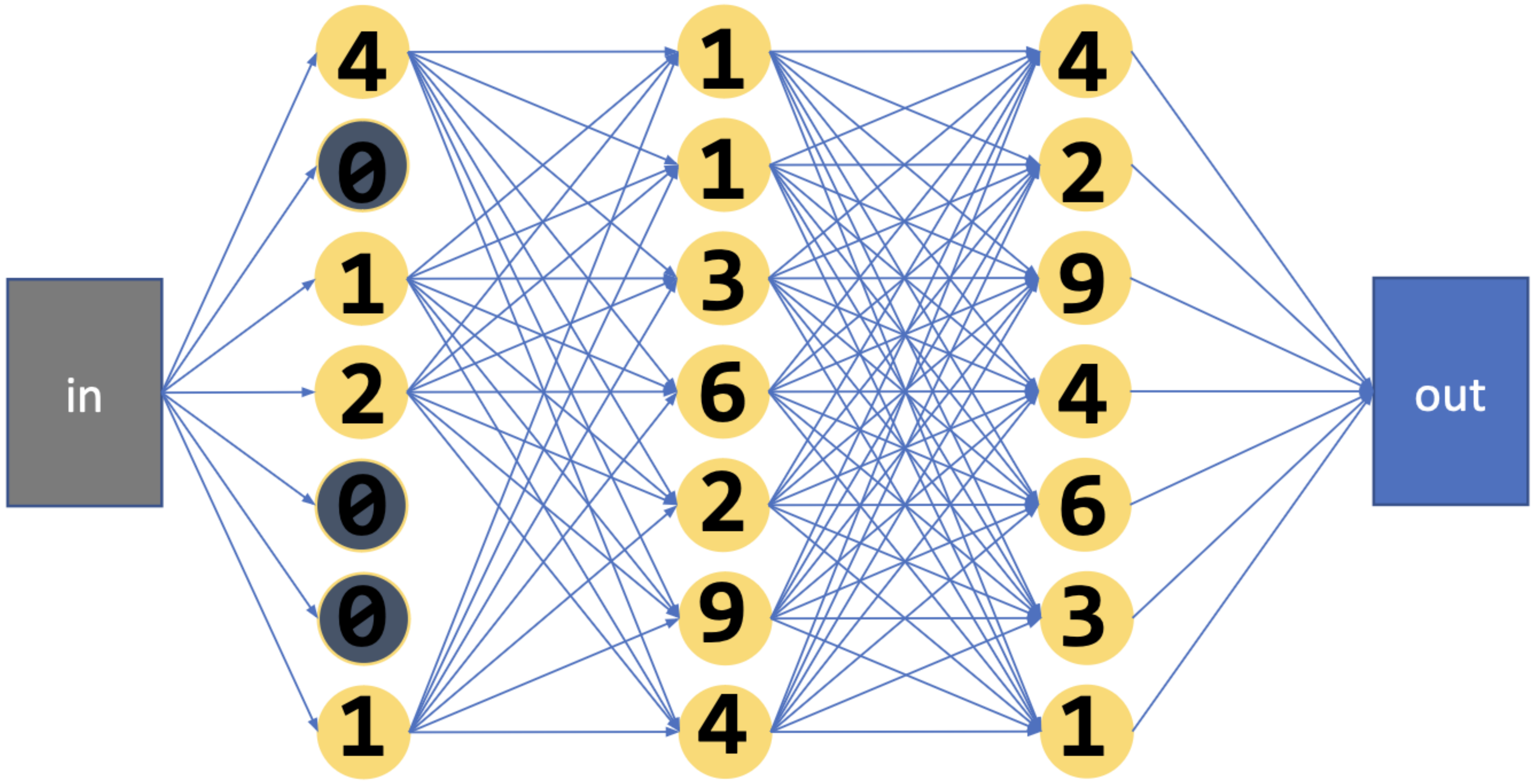
A critically important, ubiquitous, and yet poorly understood ingredient in modern deep networks (DNs) is batch normalization (BN), which centers and normalizes the feature maps. To date, only limited progress has been made understanding why BN boosts DN learning and inference performance; work has focused exclusively on showing that BN smooths a DN's loss landscape. In this paper, we study BN theoretically from the perspective of function approximation; we exploit the fact that most of today's state-of-the-art DNs are continuous piecewise affine (CPA) splines that fit a predictor to the training data via affine mappings defined over a partition of the input space (the so-called "linear regions"). *We demonstrate that BN is an unsupervised learning technique that – independent of the DN's weights or gradient-based learning – adapts the geometry of a DN's spline partition to match the data.* BN provides a "smart initialization" that boosts the performance of DN learning, because it adapts even a DN initialized with random weights to align its spline partition with the data. We also show that the variation of BN statistics between mini-batches introduces a dropout-like random perturbation to the partition boundaries and hence the decision boundary for classification problems. This per mini-batch perturbation reduces overfitting and improves generalization by increasing the margin between the training samples and the decision boundary.

Dropout - what?



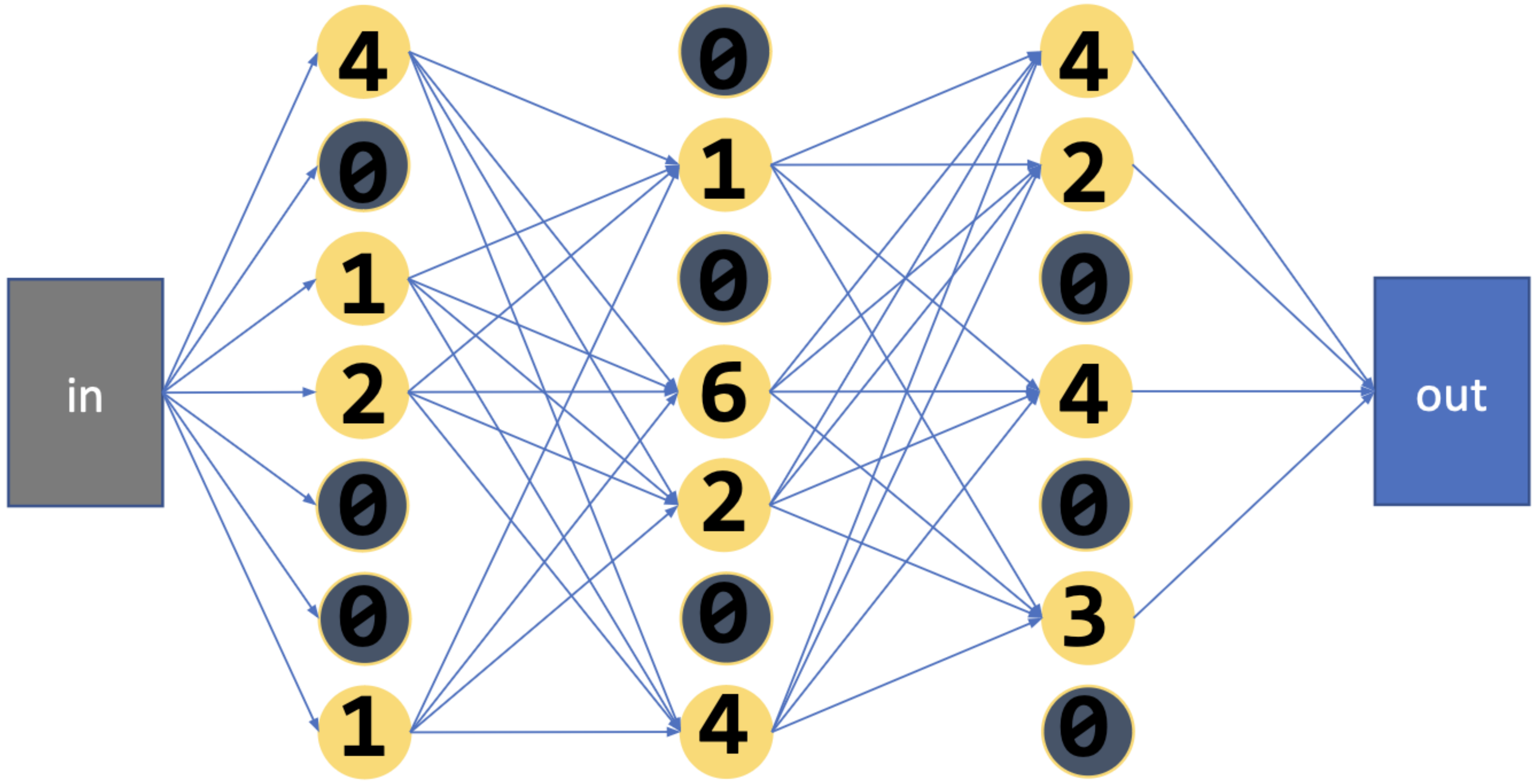
Typical NN: the output of every node in every layer is used in the next layer of the network

Dropout - what?



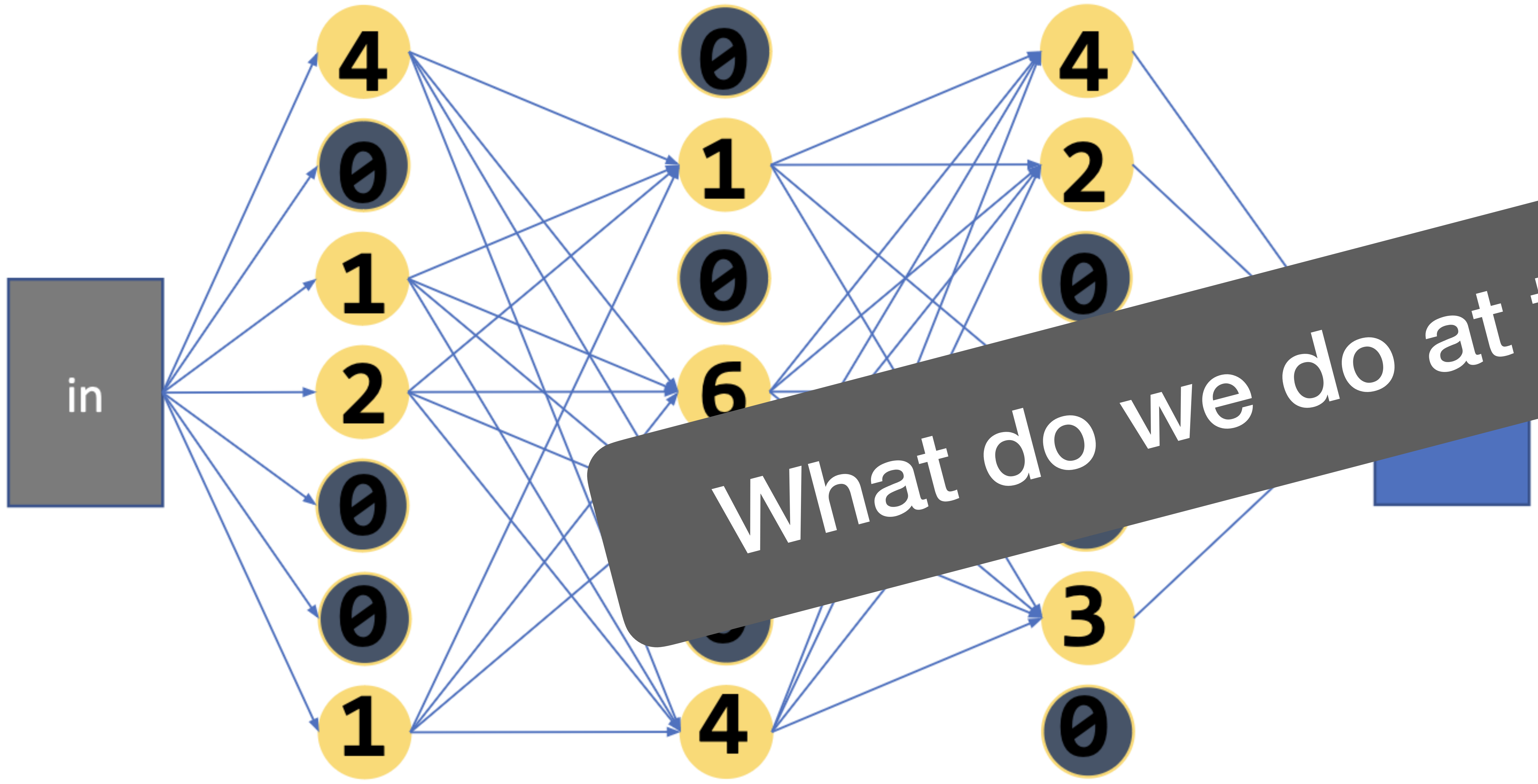
Dropout: *in a single training pass*, the output of randomly selected nodes from each layer will “drop out”, i.e. be set to 0

Dropout - what?



Not just limited to the input layer: can do this to *any* layer of the network

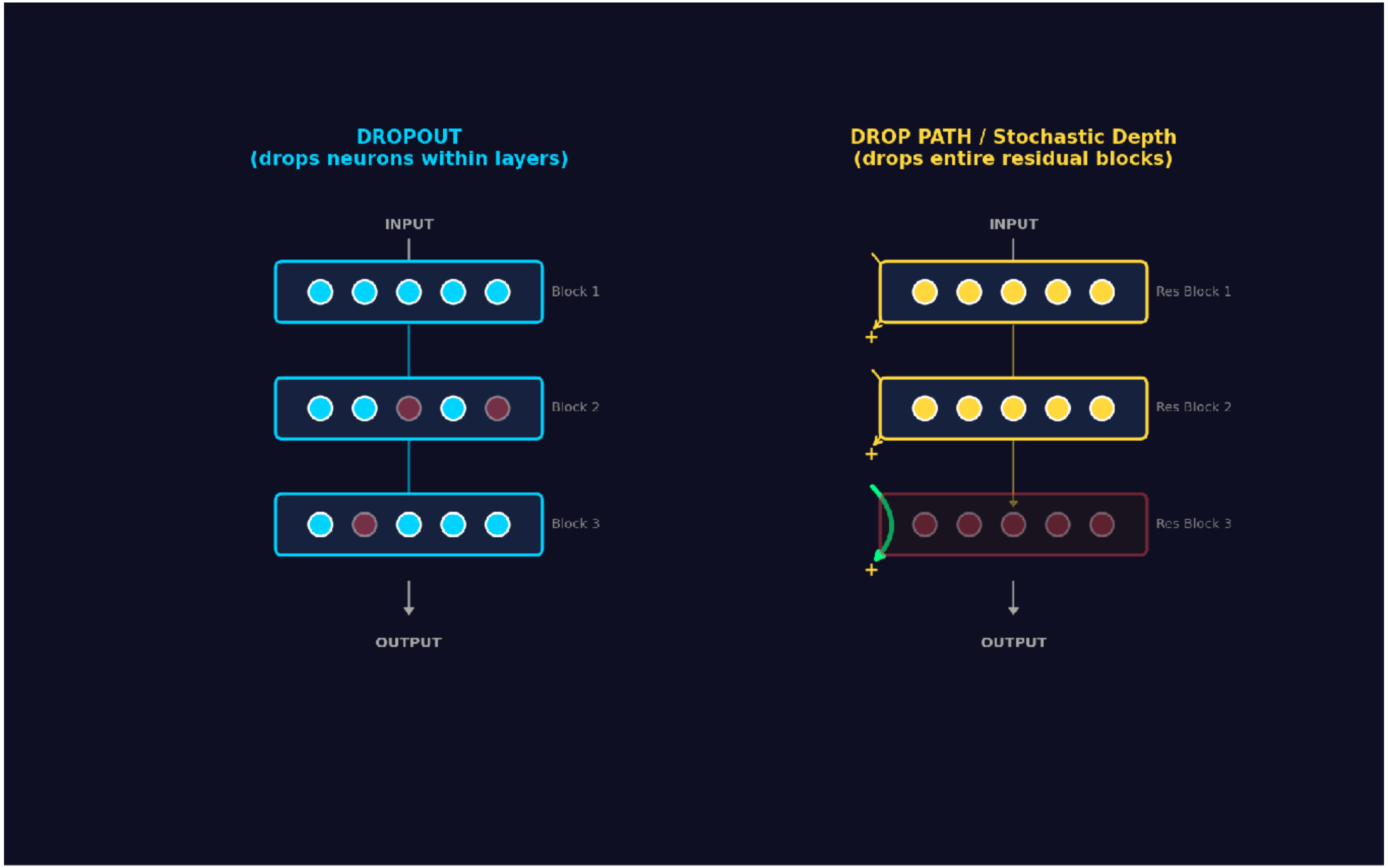
Dropout - what?



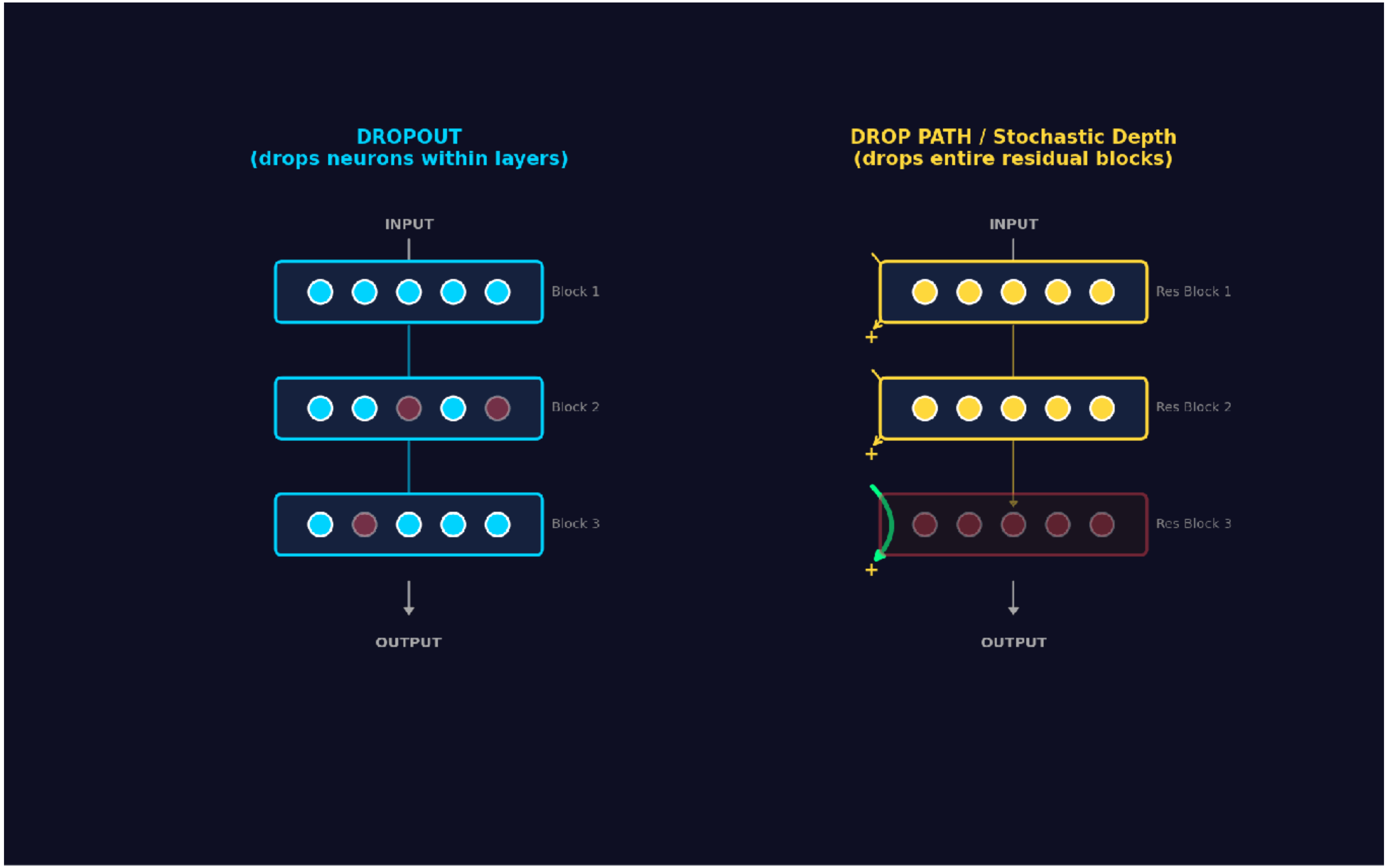
Not just limited to the input laye an do this to e network

What do we do at test time?

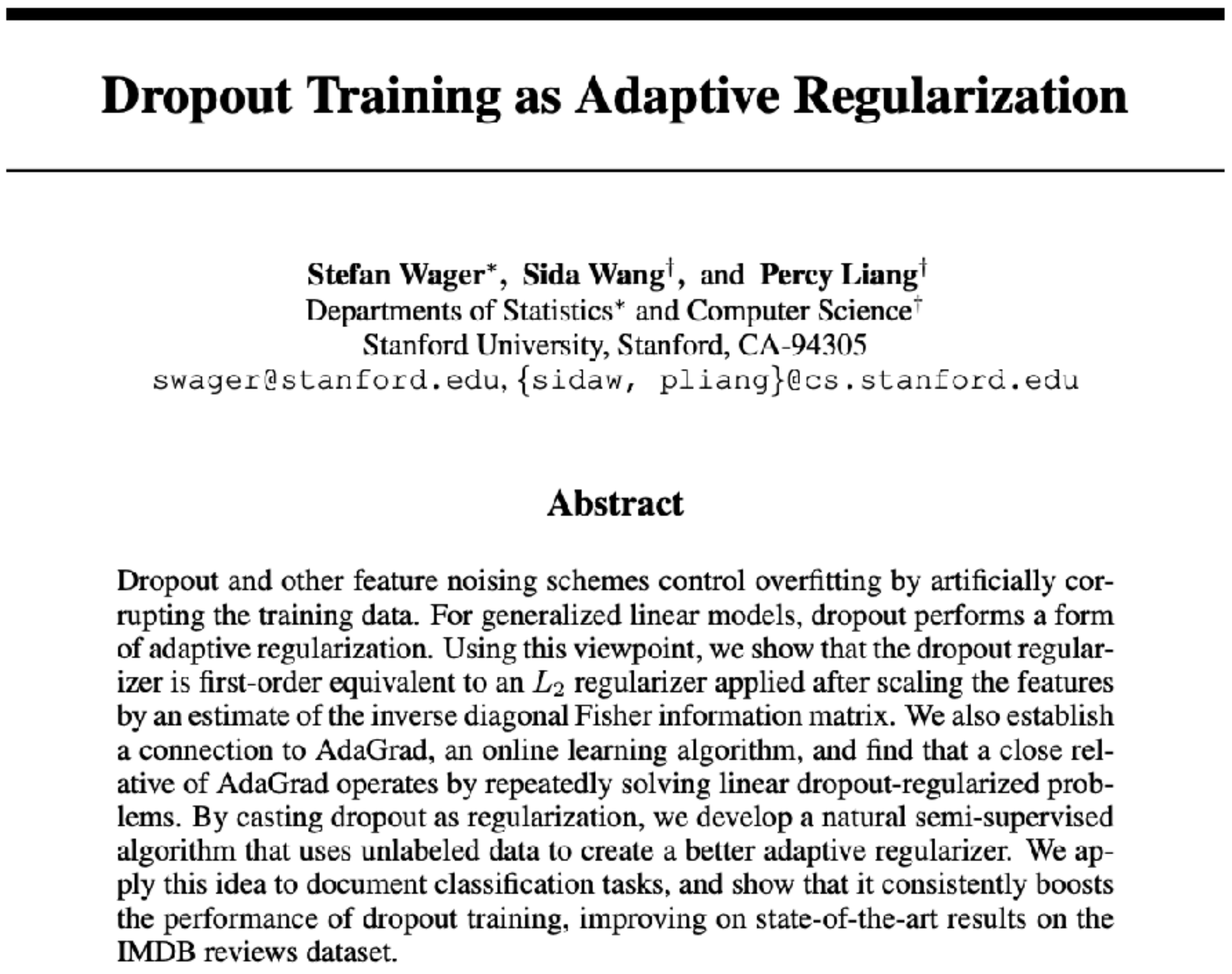
Dropout for ResNets


DROPOUT
(drops neurons within layers)
INPUT
Block 1
Block 2
Block 3
OUTPUT
DROP PATH / Stochastic Depth
(drops entire residual blocks)
INPUT
Res Block 1
+
Res Block 2
+
Res Block 3
+
OUTPUT

Dropout for ResNets


DROPOUT
(drops neurons within layers)
INPUT
Block 1
Block 2
Block 3
OUTPUT
DROP PATH / Stochastic Depth
(drops entire residual blocks)
INPUT
Res Block 1
Res Block 2
Res Block 3
+
+
+
OUTPUT

Dropout as Weight Decay

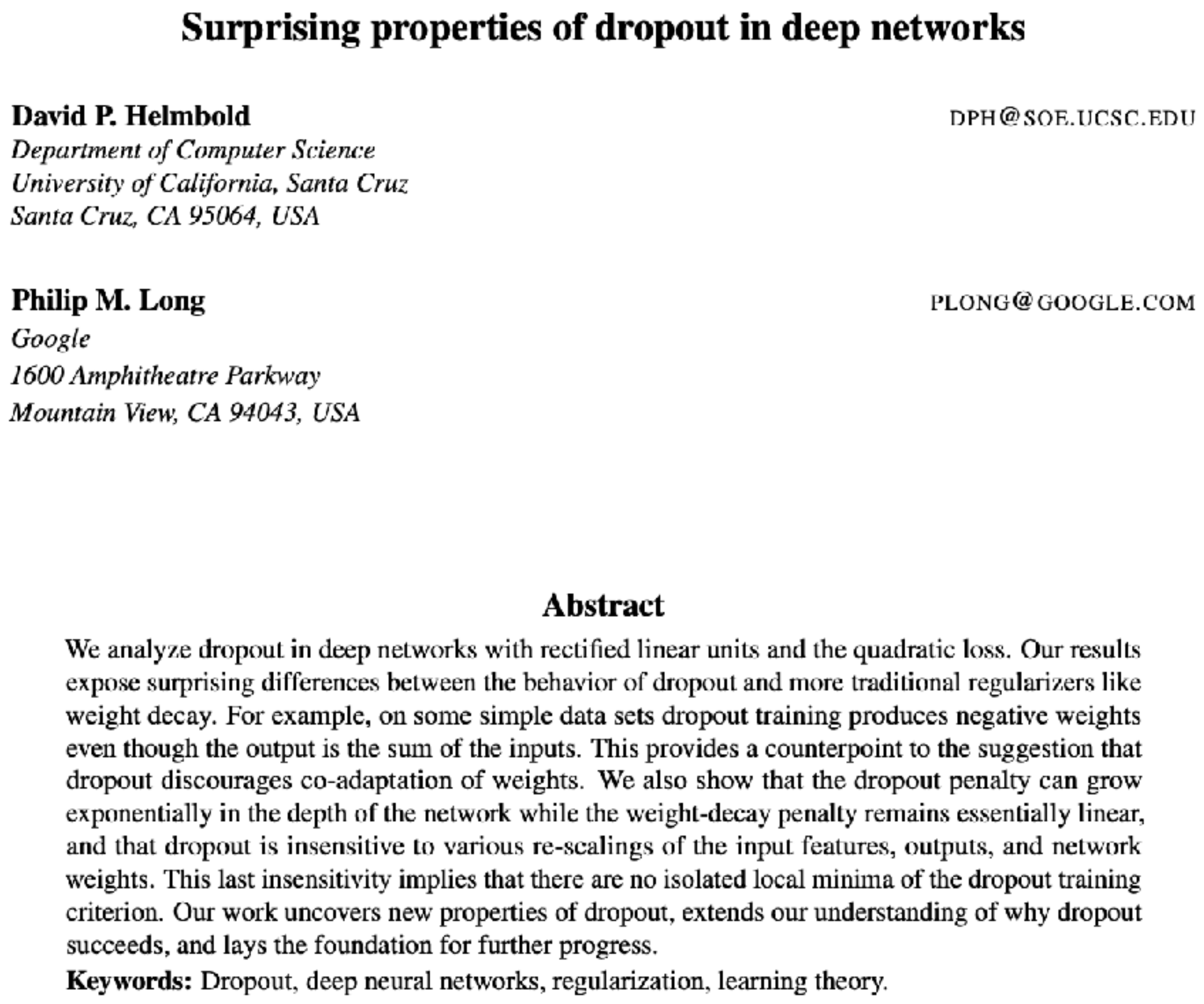


Dropout Training as Adaptive Regularization

Stefan Wager[*], **Sida Wang**[†], and **Percy Liang**[†]
Departments of Statistics[*] and Computer Science[†]
Stanford University, Stanford, CA-94305
swager@stanford.edu, {sidaw, pliang}@cs.stanford.edu

Abstract

Dropout and other feature noising schemes control overfitting by artificially corrupting the training data. For generalized linear models, dropout performs a form of adaptive regularization. Using this viewpoint, we show that the dropout regularizer is first-order equivalent to an L_2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix. We also establish a connection to AdaGrad, an online learning algorithm, and find that a close relative of AdaGrad operates by repeatedly solving linear dropout-regularized problems. By casting dropout as regularization, we develop a natural semi-supervised algorithm that uses unlabeled data to create a better adaptive regularizer. We apply this idea to document classification tasks, and show that it consistently boosts the performance of dropout training, improving on state-of-the-art results on the IMDB reviews dataset.



Surprising properties of dropout in deep networks

David P. Helmbold DPH@SOE.UCSC.EDU
Department of Computer Science
University of California, Santa Cruz
Santa Cruz, CA 95064, USA

Philip M. Long PLONG@GOOGLE.COM
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA

Abstract

We analyze dropout in deep networks with rectified linear units and the quadratic loss. Our results expose surprising differences between the behavior of dropout and more traditional regularizers like weight decay. For example, on some simple data sets dropout training produces negative weights even though the output is the sum of the inputs. This provides a counterpoint to the suggestion that dropout discourages co-adaptation of weights. We also show that the dropout penalty can grow exponentially in the depth of the network while the weight-decay penalty remains essentially linear, and that dropout is insensitive to various re-scalings of the input features, outputs, and network weights. This last insensitivity implies that there are no isolated local minima of the dropout training criterion. Our work uncovers new properties of dropout, extends our understanding of why dropout succeeds, and lays the foundation for further progress.

Keywords: Dropout, deep neural networks, regularization, learning theory.

Questions?