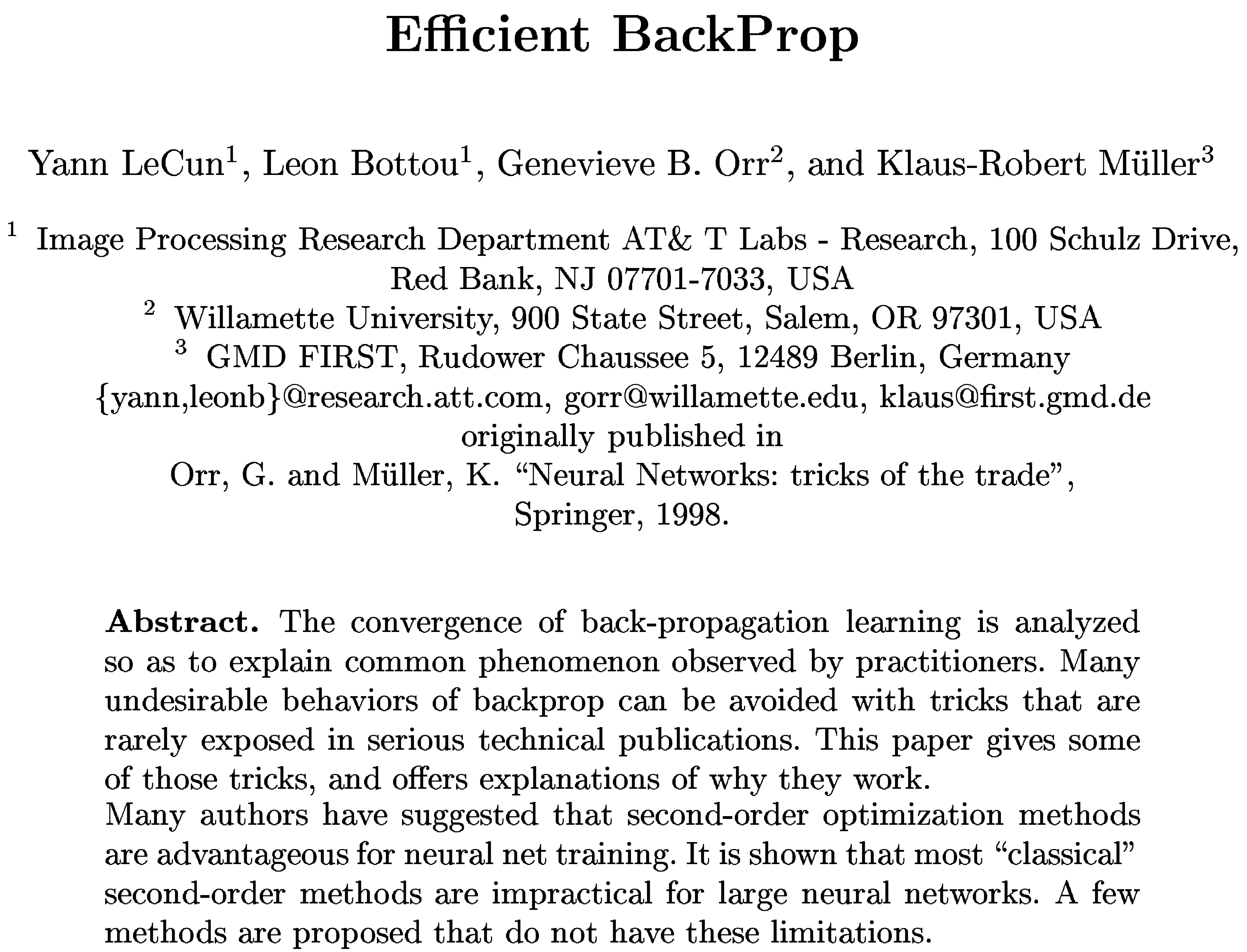
ENM 540: Data-driven modeling and probabilistic scientific computing

Lecture #7: Practical tips and Physics-informed neural networks

Paris Perdikaris

February 1, 2018

A practical guide



Efficient BackProp

Yann LeCun[1], Leon Bottou[1], Genevieve B. Orr[2], and Klaus-Robert Müller[3]

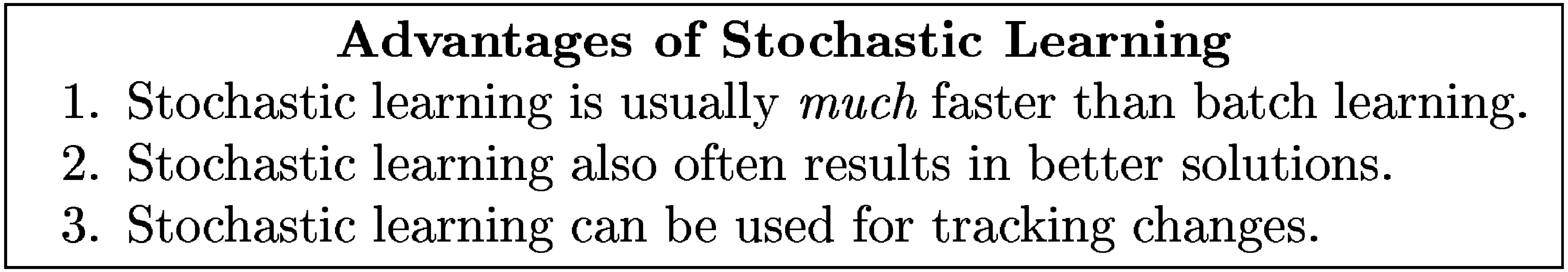
[1] Image Processing Research Department AT& T Labs - Research, 100 Schulz Drive,
Red Bank, NJ 07701-7033, USA
[2] Willamette University, 900 State Street, Salem, OR 97301, USA
[3] GMD FIRST, Rudower Chaussee 5, 12489 Berlin, Germany
{yann,leonb}@research.att.com, gorr@willamette.edu, klaus@first.gmd.de

originally published in
Orr, G. and Müller, K. "Neural Networks: tricks of the trade",
Springer, 1998.

Abstract. The convergence of back-propagation learning is analyzed so as to explain common phenomenon observed by practitioners. Many undesirable behaviors of backprop can be avoided with tricks that are rarely exposed in serious technical publications. This paper gives some of those tricks, and offers explanations of why they work.
Many authors have suggested that second-order optimization methods are advantageous for neural net training. It is shown that most "classical" second-order methods are impractical for large neural networks. A few methods are proposed that do not have these limitations.

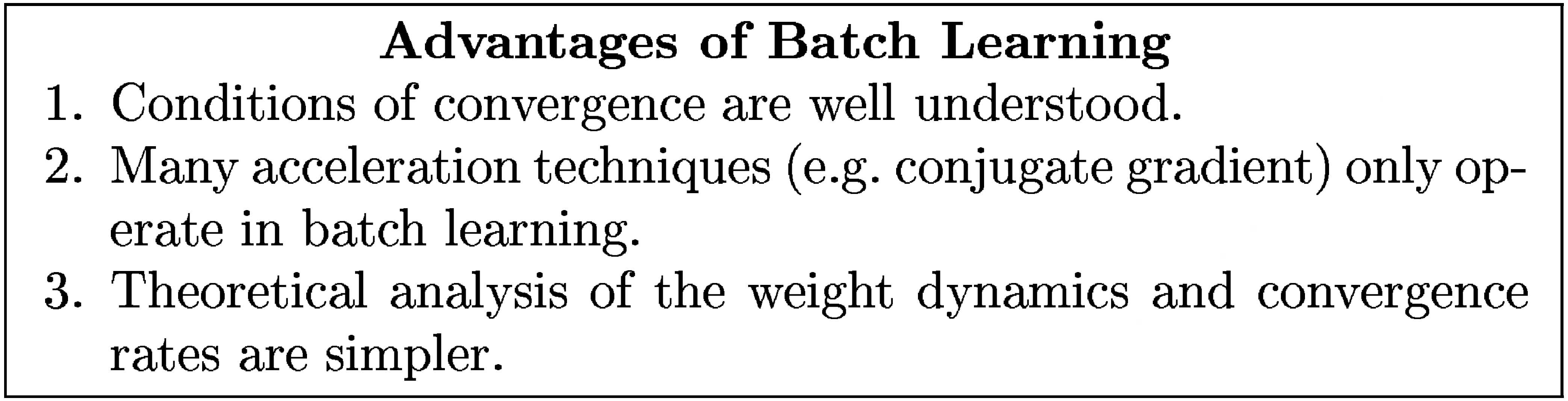
A practical guide

Stochastic vs full-batch learning



Advantages of Stochastic Learning

1. Stochastic learning is usually *much* faster than batch learning.
2. Stochastic learning also often results in better solutions.
3. Stochastic learning can be used for tracking changes.

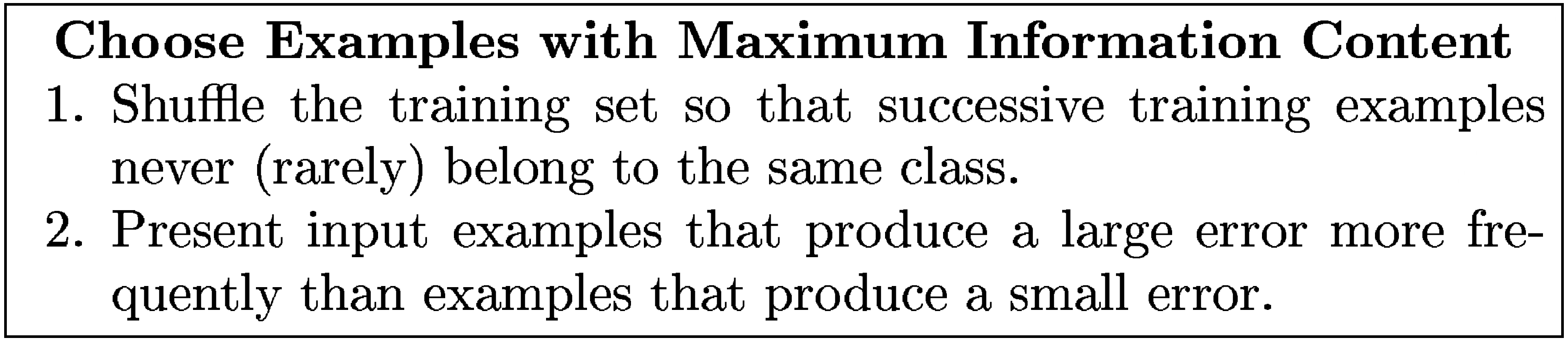


Advantages of Batch Learning

1. Conditions of convergence are well understood.
2. Many acceleration techniques (e.g. conjugate gradient) only operate in batch learning.
3. Theoretical analysis of the weight dynamics and convergence rates are simpler.

A practical guide

Shuffling the examples

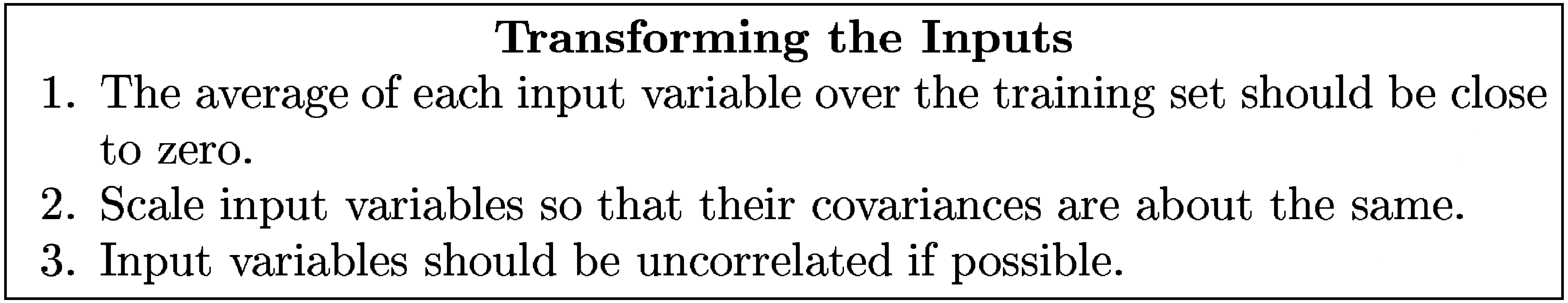


Choose Examples with Maximum Information Content

1. Shuffle the training set so that successive training examples never (rarely) belong to the same class.
2. Present input examples that produce a large error more frequently than examples that produce a small error.

A practical guide

Normalizing the inputs



Transforming the Inputs

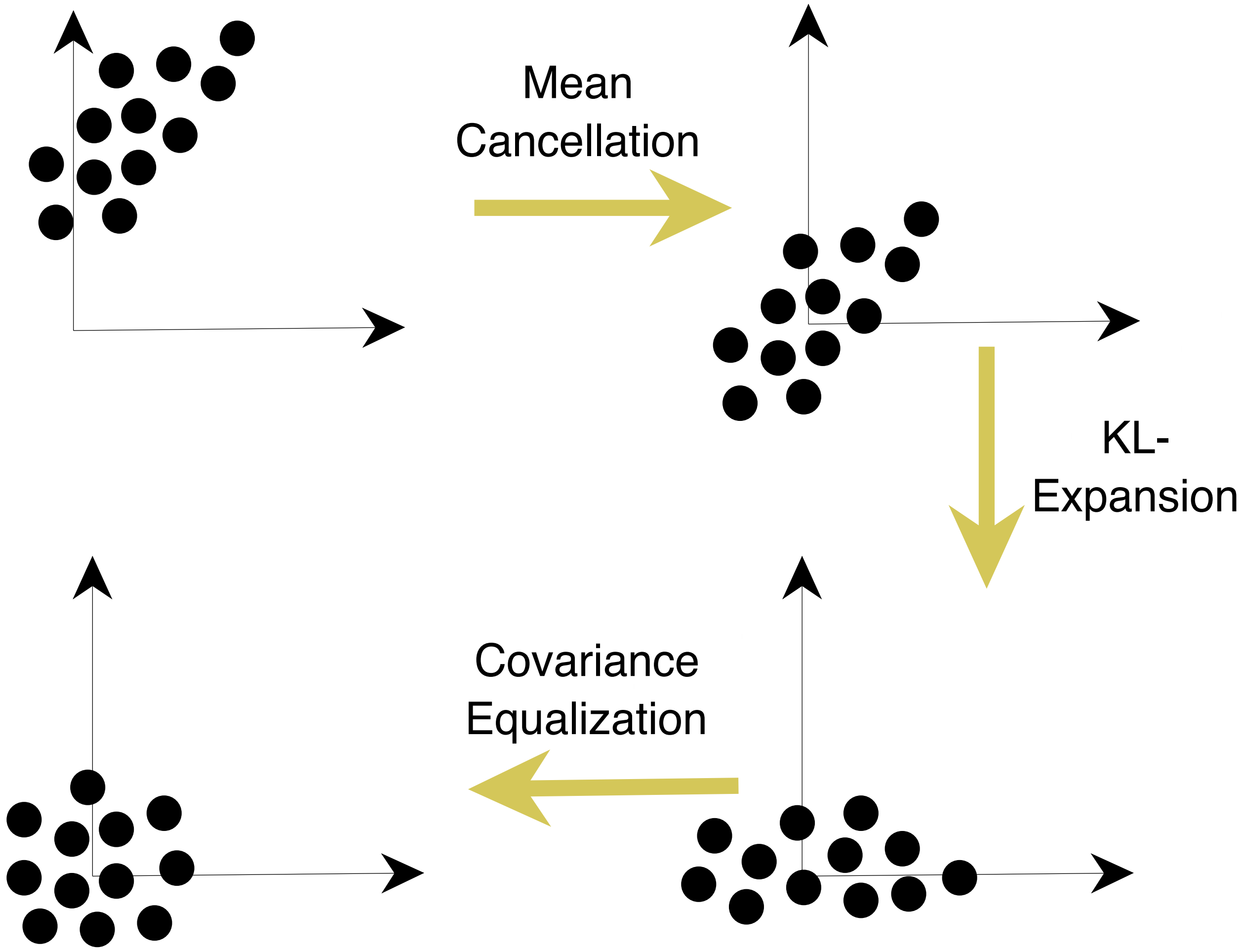
1. The average of each input variable over the training set should be close to zero.
2. Scale input variables so that their covariances are about the same.
3. Input variables should be uncorrelated if possible.

A practical guide

Normalizing the inputs

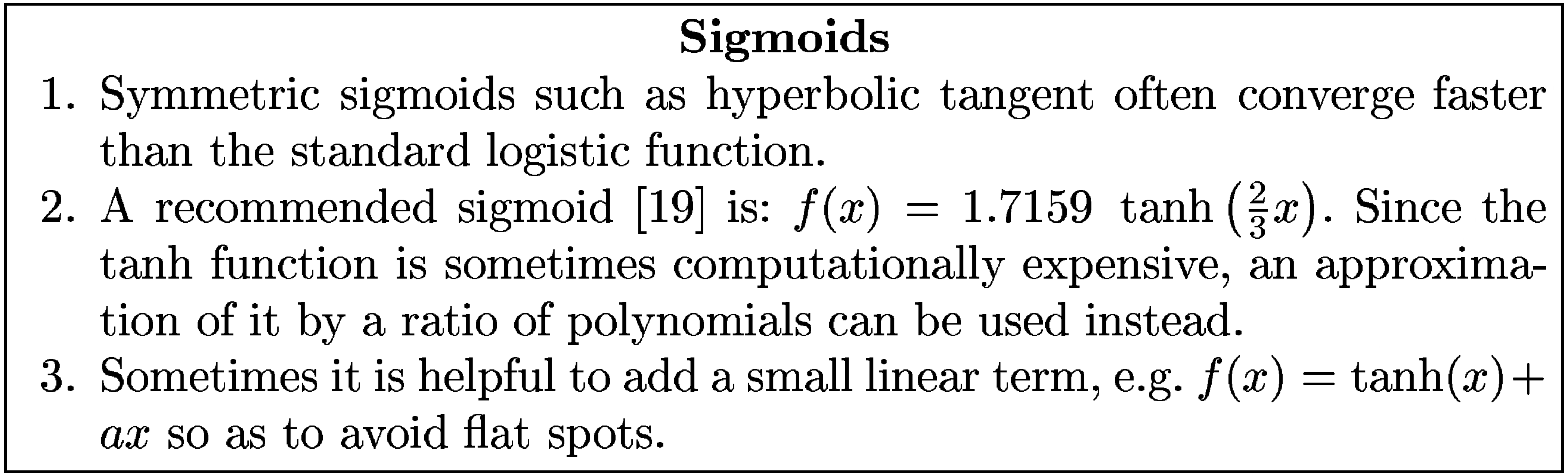
Transforming the Inputs

1. The average of each input variable over the training set should be close to zero.
2. Scale input variables so that their covariances are about the same.
3. Input variables should be uncorrelated if possible.



A practical guide

Activation functions



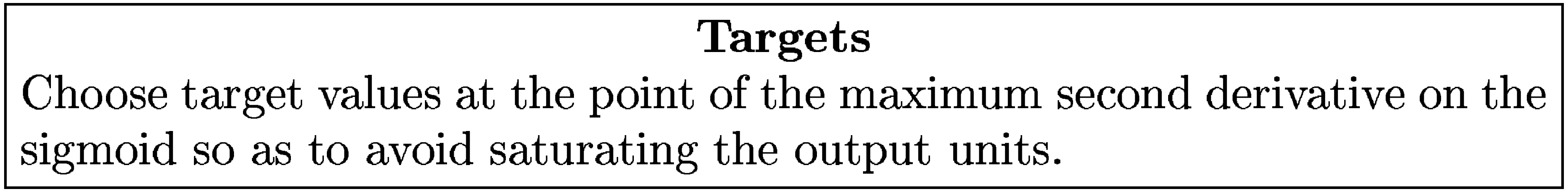
Sigmoids

1. Symmetric sigmoids such as hyperbolic tangent often converge faster than the standard logistic function.
2. A recommended sigmoid [19] is: $f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right)$. Since the tanh function is sometimes computationally expensive, an approximation of it by a ratio of polynomials can be used instead.
3. Sometimes it is helpful to add a small linear term, e.g. $f(x) = \tanh(x) + ax$ so as to avoid flat spots.

…+ more recent findings on ReLUs, ELUs, etc.

A practical guide

Choosing target values



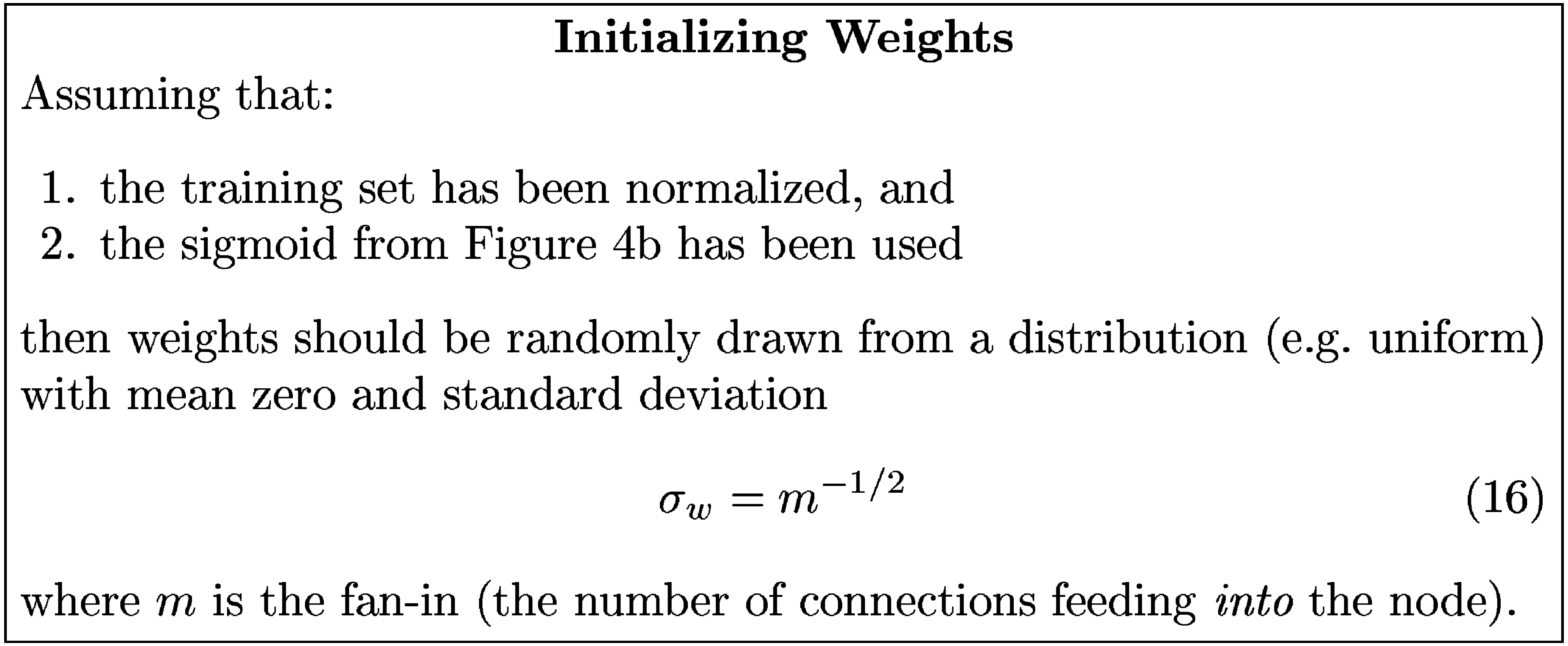
Targets
Choose target values at the point of the maximum second derivative on the sigmoid so as to avoid saturating the output units.

*Referring to tanh() activations, thus suggesting {-1+1} labels for binary classification.

…+ more recent findings on ReLUs, ELUs, etc.

A practical guide

Initializing the weights



Initializing Weights

Assuming that:

1. the training set has been normalized, and
2. the sigmoid from Figure 4b has been used

then weights should be randomly drawn from a distribution (e.g. uniform) with mean zero and standard deviation

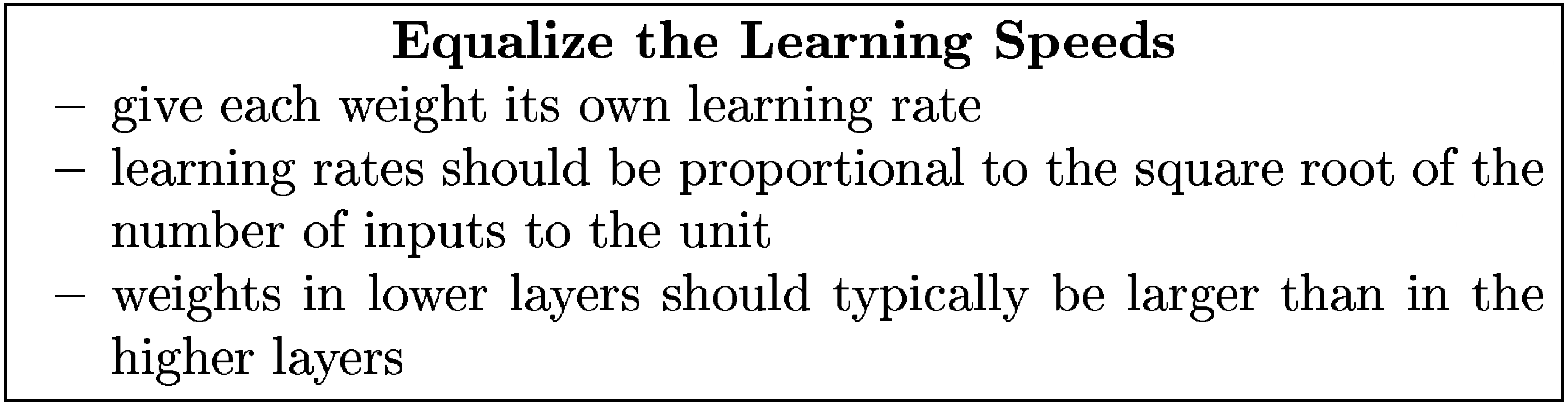
$$\sigma_w = m^{-1/2} \tag{16}$$

where m is the fan-in (the number of connections feeding *into* the node).

*Referring to tanh() activations, different treatment may be necessary for other activation functions.

A practical guide

Choosing the learning rate



Equalize the Learning Speeds

– give each weight its own learning rate

– learning rates should be proportional to the square root of the number of inputs to the unit

– weights in lower layers should typically be larger than in the higher layers

*New developments and modern SGD variants have addressed this to some extend.

Network initialization

Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada

Whereas before 2006 it appears that deep multi-layer neural networks were not successfully trained, since then several algorithms have been shown to successfully train them, with experimental results showing the superiority of deeper vs less deep architectures. All these experimental results were obtained with new initialization or training mechanisms. Our objective here is to understand better why standard gradient descent from random initialization is doing so poorly with deep neural networks, to better understand these recent relative successes and help design better algorithms in the future. We first observe the influence of the non-linear activations functions. We find that the logistic sigmoid activation is unsuited for deep networks with random initialization because of its mean value, which can drive especially the top hidden layer into saturation. Surprisingly, we find that saturated units can move out of saturation by themselves, albeit slowly, and explaining the plateaus sometimes seen when training neural networks. We find that a new non-linearity that saturates less can often be beneficial. Finally, we study how activations and gradients vary across layers and during train- ing, with the idea that training may be more difficult when the singular values of the Jacobian associated with each layer are far from 1. Based on these considerations, we propose a new initialization scheme that brings substantially faster convergence.

Network initialization

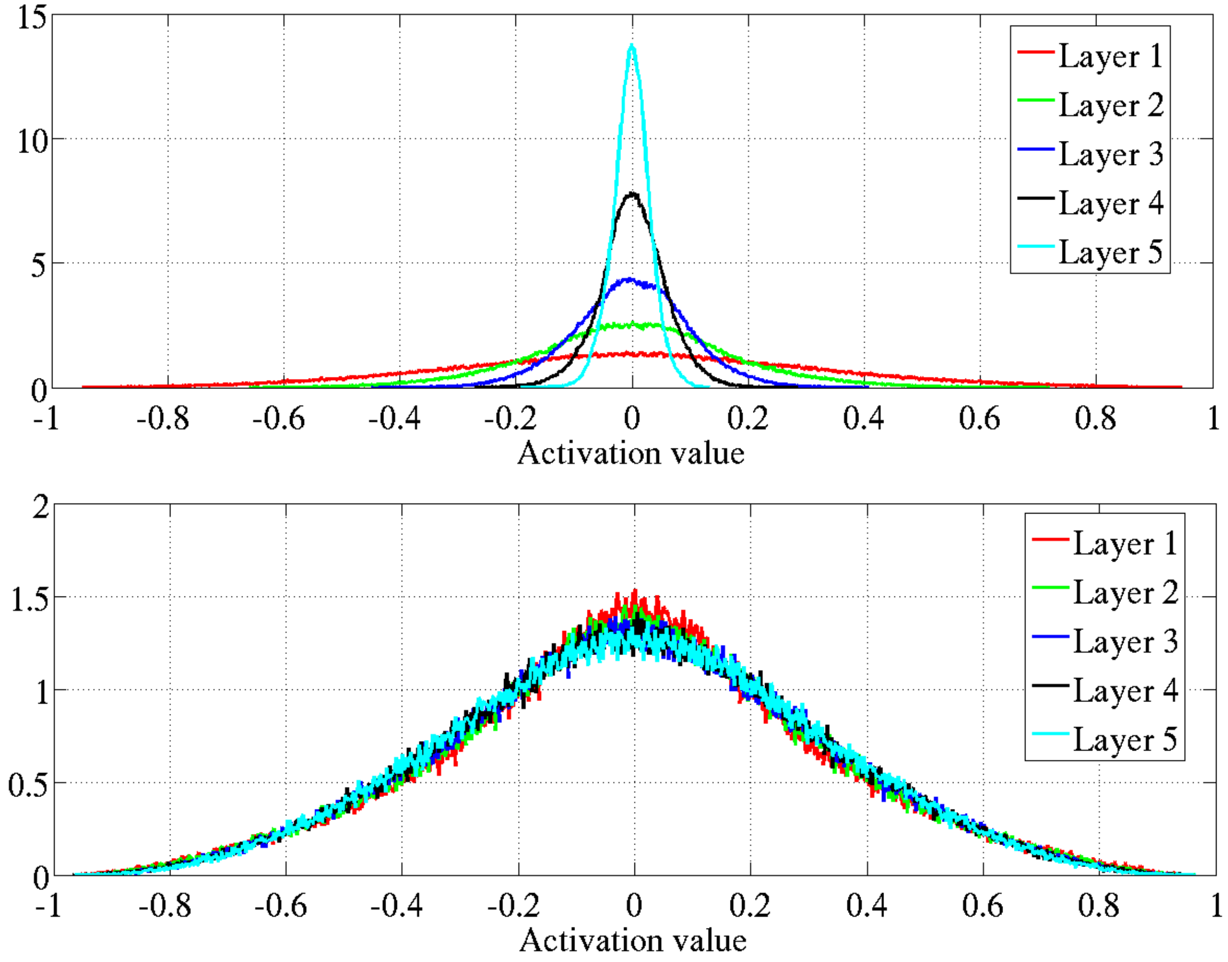


Figure 6: *Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.*

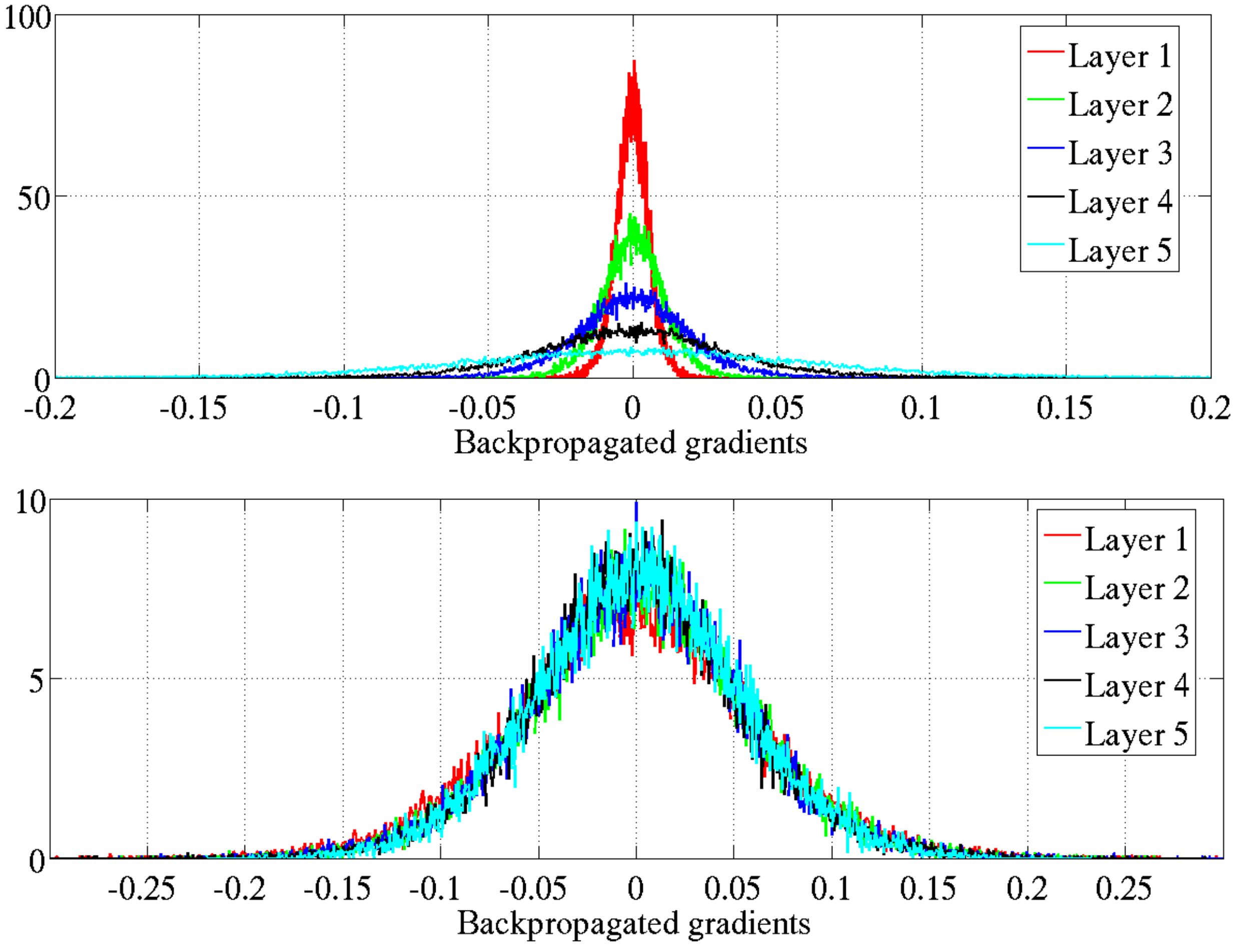
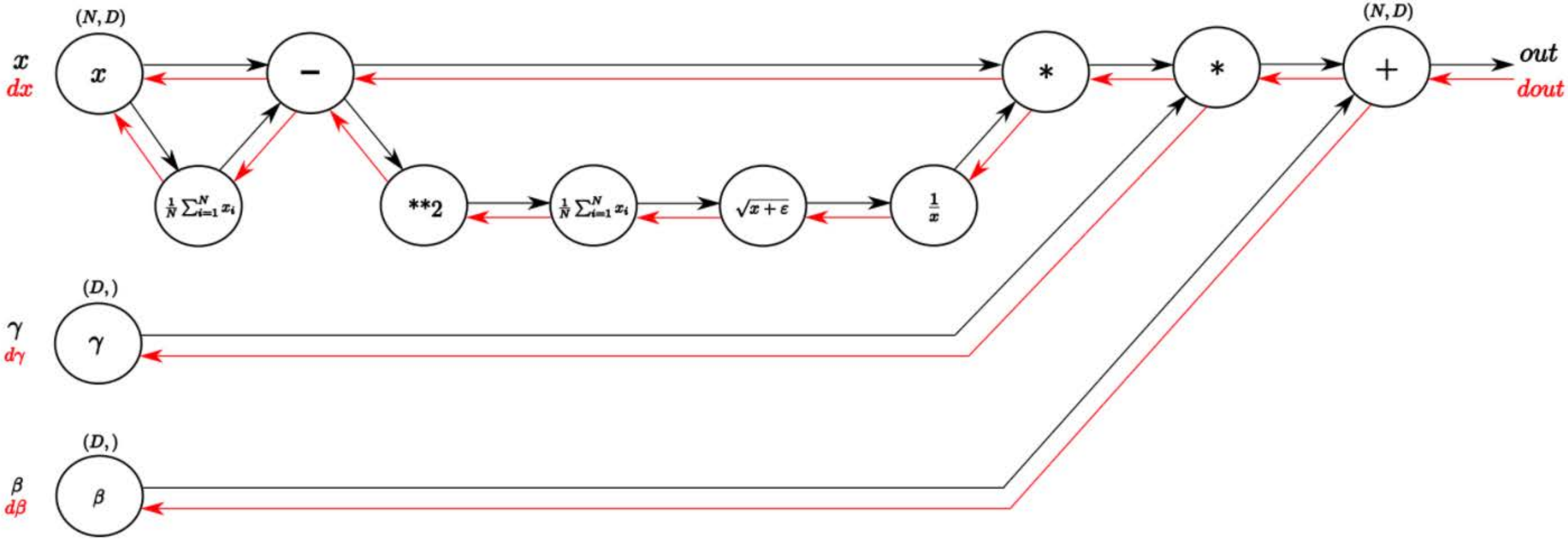


Figure 7: *Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized (bottom) initialization. Top: 0-peak decreases for higher layers.*

Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (pp. 249-256).

Batch normalization

Just a reminder for when you’re doing mini-batch SGD in deep networks…



Computational graph of the BatchNorm-Layer. From left to right, following the black arrows flows the forward pass. The inputs are a matrix X and gamma and beta as vectors. From right to left, following the red arrows flows the backward pass which distributes the gradient from above layer to gamma and beta and all the way back to the input.

Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456).

Physics-informed ML

Motivation:

- In many applications a large number of quality and error-free data is prohibitively expensive to obtain.
- Under this data-scarce and variable fidelity setting, state-of-the-art ML and SC algorithms are lacking robustness and fail to return predictions with quantified uncertainty.

Probabilistic scientific computing:

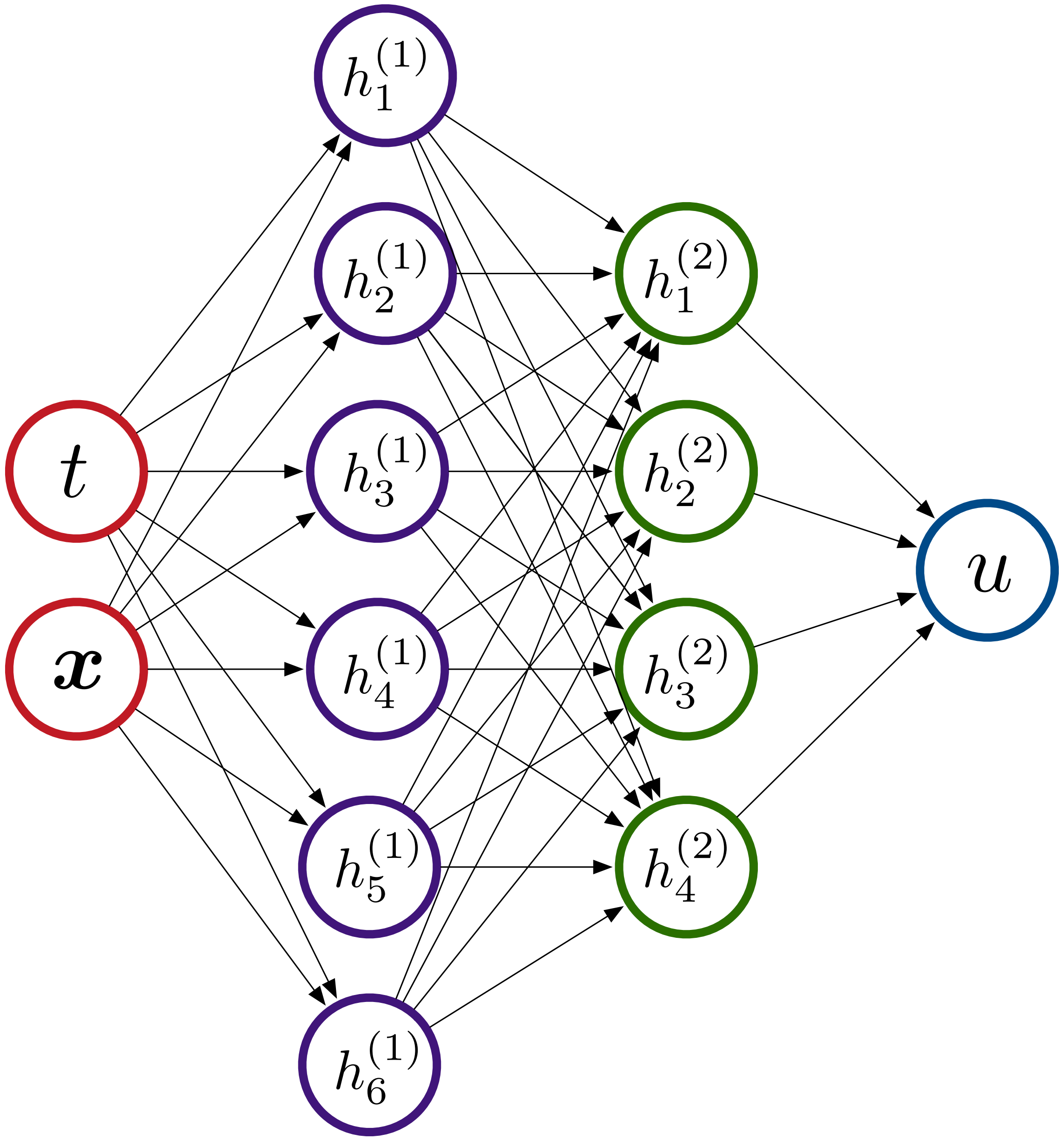
Recent work has demonstrated how conservation laws and numerical discretization schemes can be used as structured prior information that can enhance the robustness and efficiency of modern machine learning algorithms, and introduce a new class of data-driven solvers and model discovery techniques.

Being a field at its infancy, many fundamental questions arise:

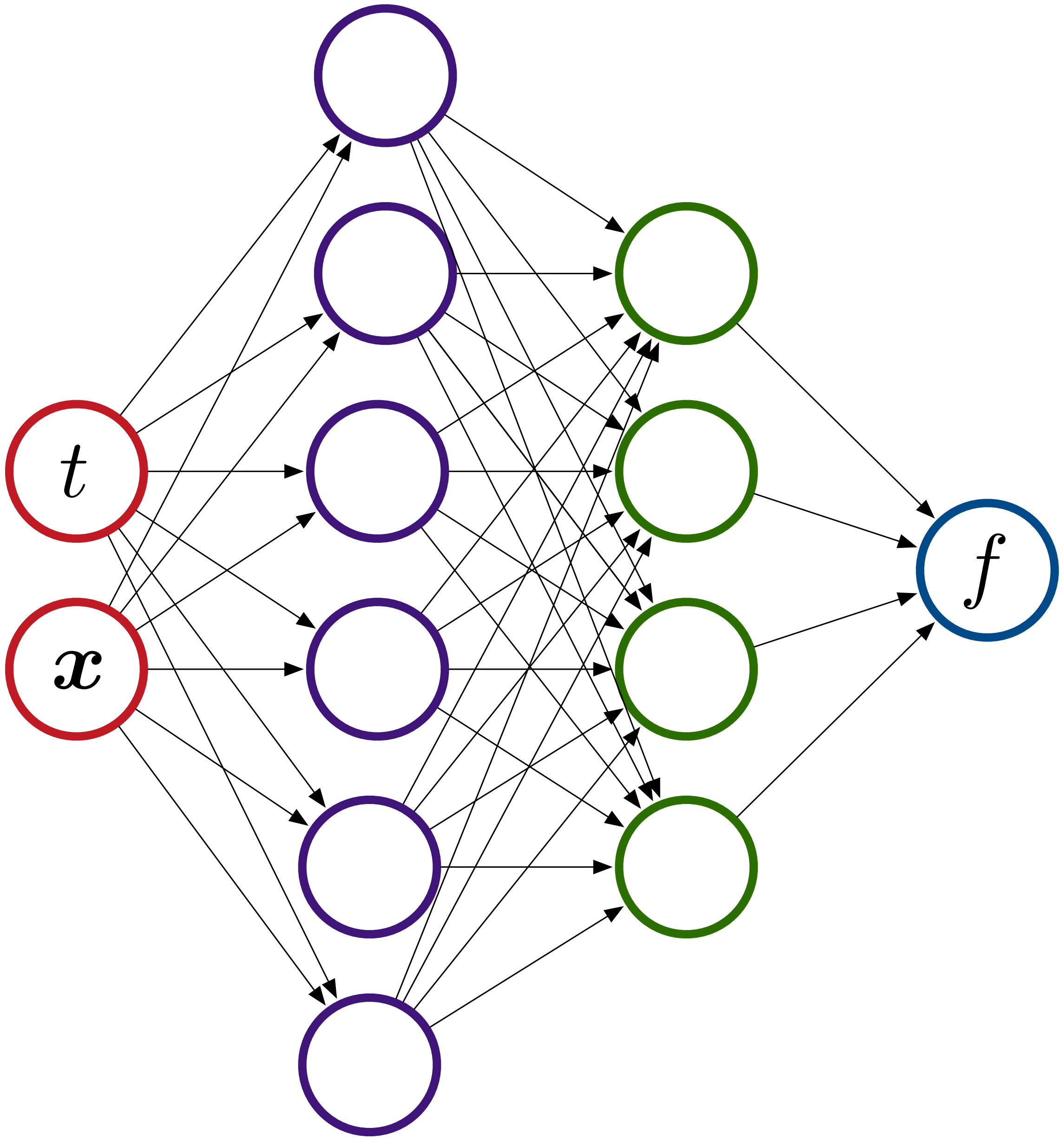
- From a SC viewpoint: accuracy, convergence rates, complex dynamics, exascale computing.
- From a ML viewpoint: robustness/brittleness of prior assumptions, regularization of learning, interpretability and rigorous assessment of performance.

Physics-informed ML

$u(t, \boldsymbol{x})$



$f = u_t + \mathcal{N}[u; \lambda]$



Physics-informed ML

Example: Burgers' equation in 1D

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1,1], \quad t \in [0,1], \tag{3}$$
$$u(0,x) = -\sin(\pi x),$$
$$u(t,-1) = u(t,1) = 0.$$

Let us define $f(t,x)$ to be given by

$$f := u_t + uu_x - (0.01/\pi)u_{xx},$$

```
def u(t, x):
    u = neural_net(tf.concat([t,x],1), weights, biases)
    return u
```

Correspondingly, the *physics informed neural network* $f(t,x)$ takes the form

```
def f(t, x):
    u = u(t, x)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + u*u_x - (0.01/tf.pi)*u_xx
    return f
```

Physics-informed ML

The shared parameters between the neural networks $u(t, x)$ and $f(t, x)$ can be learned by minimizing the mean squared error loss

$$MSE = MSE_u + MSE_f, \tag{4}$$

where

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2,$$

and

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

Here, $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ denote the initial and boundary training data on $u(t, x)$ and $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ specify the collocations points for $f(t, x)$. The loss MSE_u corresponds to the initial and boundary data while MSE_f enforces the structure imposed by equation (3) at a finite set of collocation points.

Physics-informed ML

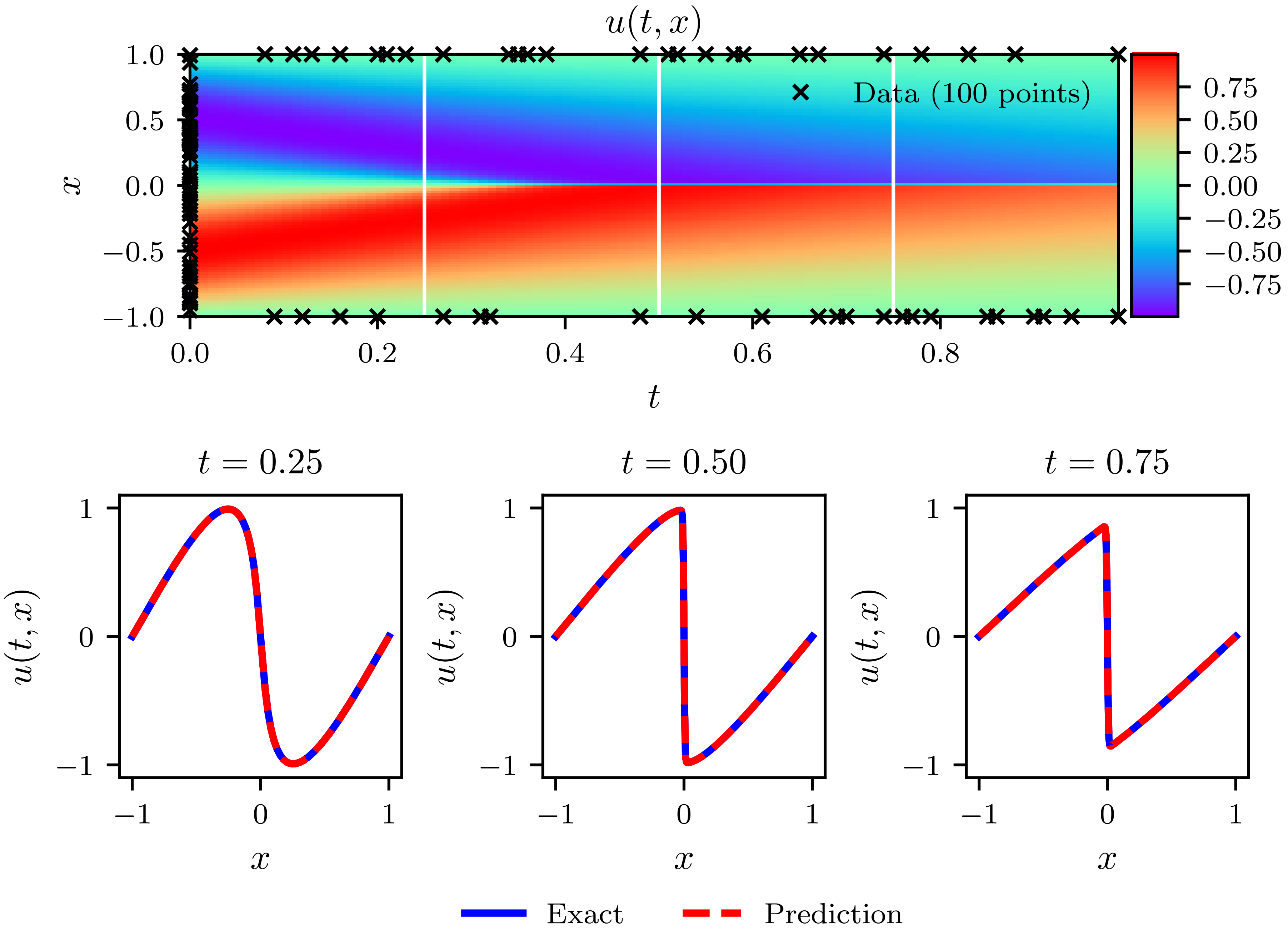
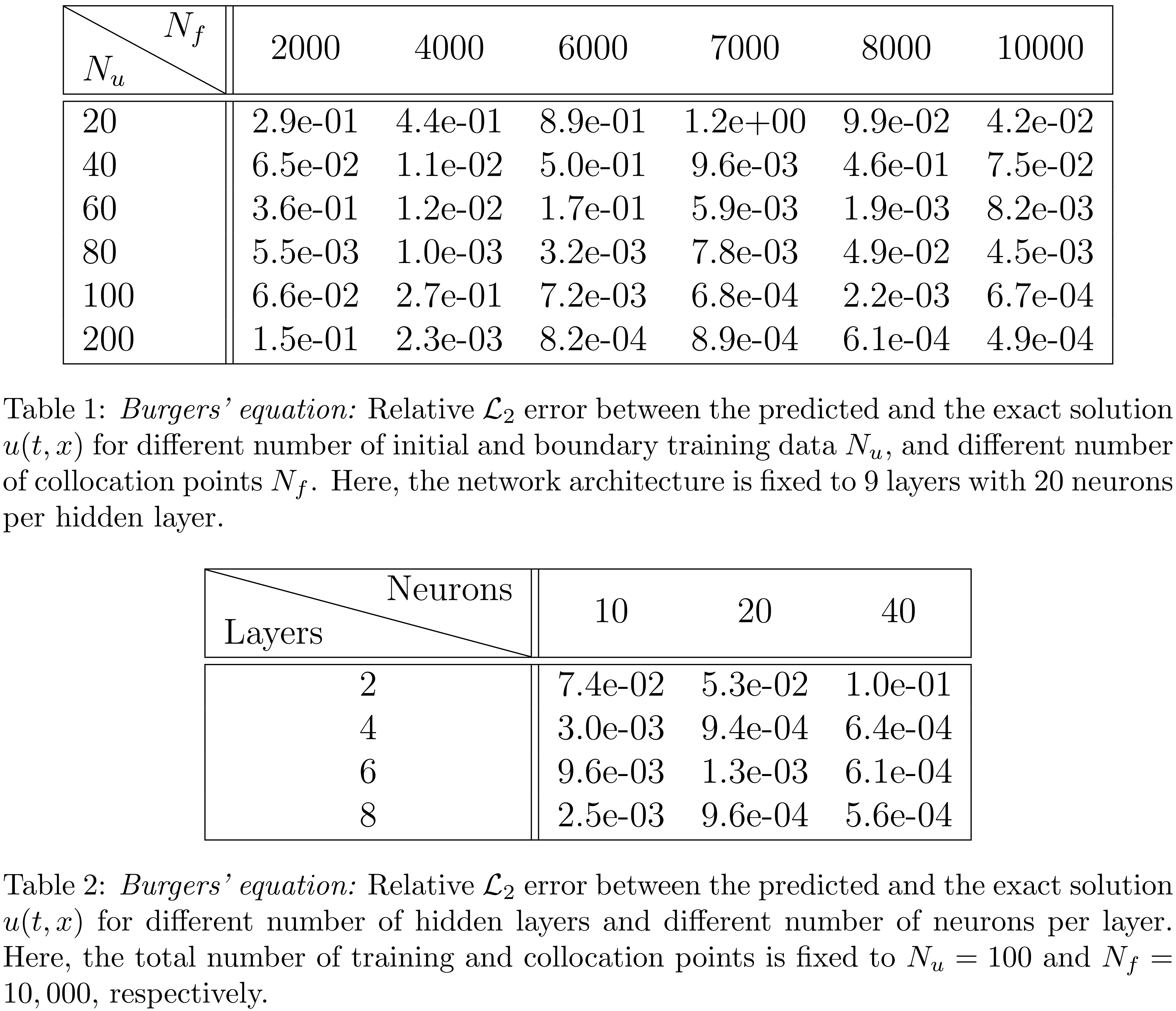


Figure 1: *Burgers' equation: Top:* Predicted solution $u(t, x)$ along with the initial and boundary training data. In addition we are using 10,000 collocation points generated using a Latin Hypercube Sampling strategy. *Bottom:* Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the white vertical lines in the top panel. The relative $\mathcal{L}_2$ error for this case is $6.7 \cdot 10^{-4}$. Model training took approximately 60 seconds on a single NVIDIA Titan X GPU card.

Physics-informed ML



N_u \ N_f	2000	4000	6000	7000	8000	10000
20	2.9e-01	4.4e-01	8.9e-01	1.2e+00	9.9e-02	4.2e-02
40	6.5e-02	1.1e-02	5.0e-01	9.6e-03	4.6e-01	7.5e-02
60	3.6e-01	1.2e-02	1.7e-01	5.9e-03	1.9e-03	8.2e-03
80	5.5e-03	1.0e-03	3.2e-03	7.8e-03	4.9e-02	4.5e-03
100	6.6e-02	2.7e-01	7.2e-03	6.8e-04	2.2e-03	6.7e-04
200	1.5e-01	2.3e-03	8.2e-04	8.9e-04	6.1e-04	4.9e-04

Table 1: *Burgers' equation:* Relative $\mathcal{L}_2$ error between the predicted and the exact solution $u(t,x)$ for different number of initial and boundary training data N_u, and different number of collocation points N_f. Here, the network architecture is fixed to 9 layers with 20 neurons per hidden layer.

Layers \ Neurons	10	20	40
2	7.4e-02	5.3e-02	1.0e-01
4	3.0e-03	9.4e-04	6.4e-04
6	9.6e-03	1.3e-03	6.1e-04
8	2.5e-03	9.6e-04	5.6e-04

Table 2: *Burgers' equation:* Relative $\mathcal{L}_2$ error between the predicted and the exact solution $u(t,x)$ for different number of hidden layers and different number of neurons per layer. Here, the total number of training and collocation points is fixed to $N_u = 100$ and $N_f = 10,000$, respectively.