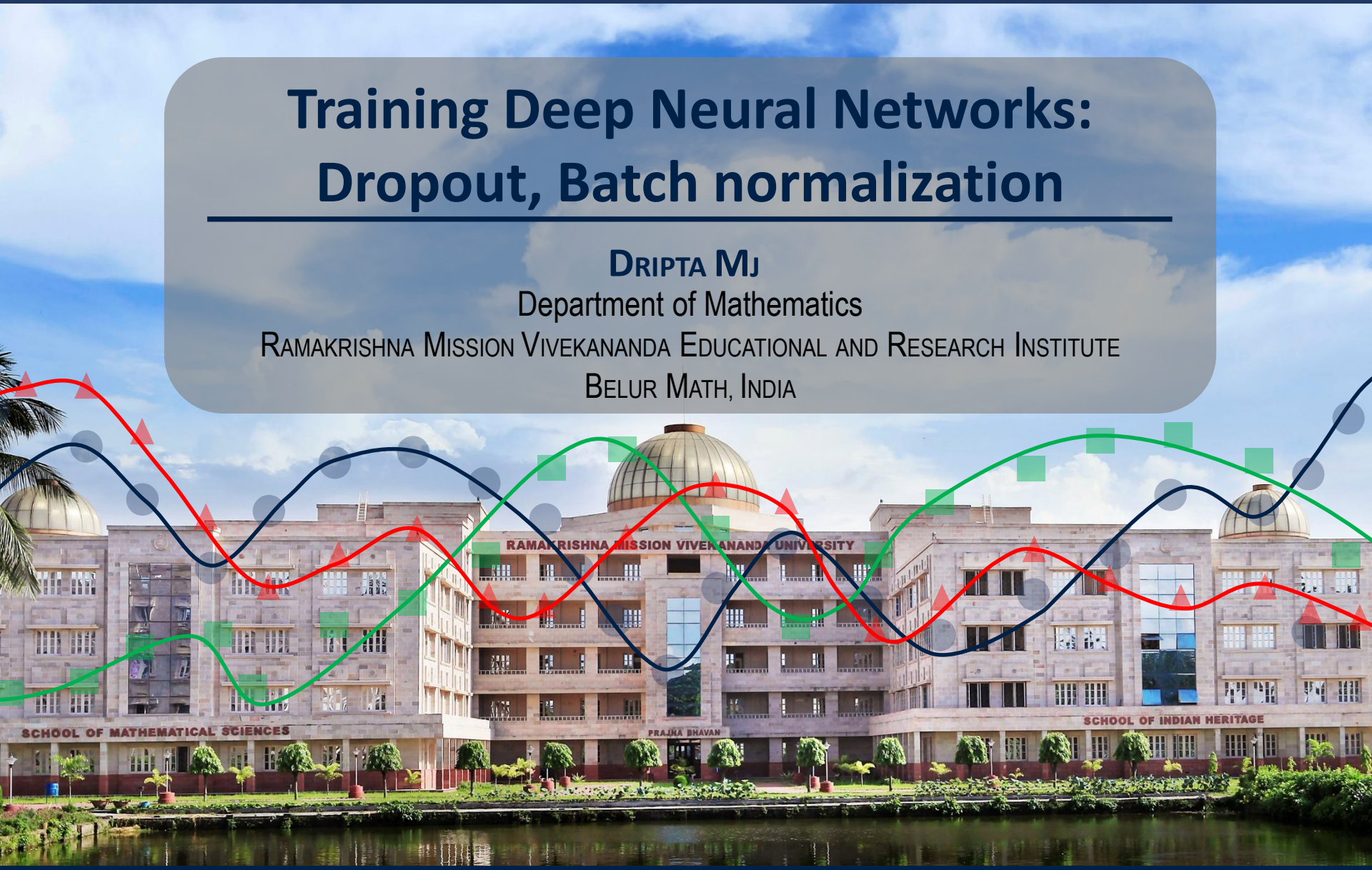
Training Deep Neural Networks: Dropout, Batch normalization

Dripta Mj
Department of Mathematics
Ramakrishna Mission Vivekananda Educational and Research Institute
Belur Math, India



Can we reduce variance?

Original decomposition:

$$\mathbb{E}_{\mathbf{x},y,\mathcal{D}}\left[\left(g_{\mathcal{D}}(\mathbf{x})-y\right)^2\right] = \mathbb{E}_{\mathbf{x},\mathcal{D}}\left[\left(g_{\mathcal{D}}(\mathbf{x})-\overline{g}(\mathbf{x})\right)^2\right] + \mathbb{E}_{\mathbf{x}}\left[\left(\overline{g}(\mathbf{x})-\overline{y}(\mathbf{x})\right)^2\right] + \mathbb{E}_{\mathbf{x},y}\left[\left(\overline{y}(\mathbf{x})-y\right)^2\right]$$

Variance | Bias^2 | Noise

- Suppose we have M different training datasets: $\mathcal{D}_1$, $\mathcal{D}_2$, ..., $\mathcal{D}_M$
- Can train a separate model on each of them: $g_{\mathcal{D}_1}, g_{\mathcal{D}_2}, ..., g_{\mathcal{D}_M}$
- Predictions can be obtained as the average of the trained models

$$\widehat{g}(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} g_{\mathcal{D}_m}(\mathbf{x}) \rightarrow \overline{g}(\mathbf{x}) \qquad \text{as} \quad M \rightarrow \infty$$

- As $\widehat{g}(\mathbf{x}) \rightarrow \overline{g}(\mathbf{x})$, the variance term $\mathbb{E}\left[(\widehat{g}(\mathbf{x})-\overline{g}(\mathbf{x}))^2\right] \rightarrow 0$
- Issue: Don't have M different training datasets.

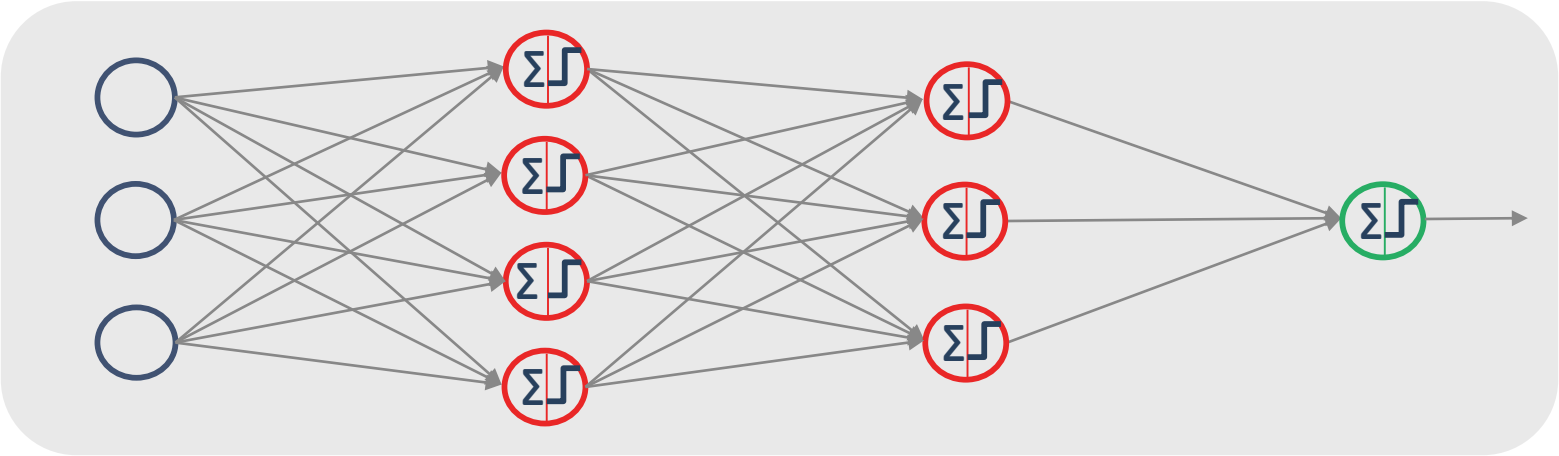
Dropout: Introduction

- Overfitting is a serious problem in large neural networks.
- Dropout tries to addresses this issue.
- Motivation: Ensemble methods are known to improve performance in most cases.
  - To achieve different models one can have different neural network architectures or have models trained on different datasets.
- Shortcomings:
  - Training different deep networks is computationally prohibitive
  - Sufficient data may not be available to train the different networks on different subsets of data properly.
- Idea: Randomly drop neurons (units) from the network during training.

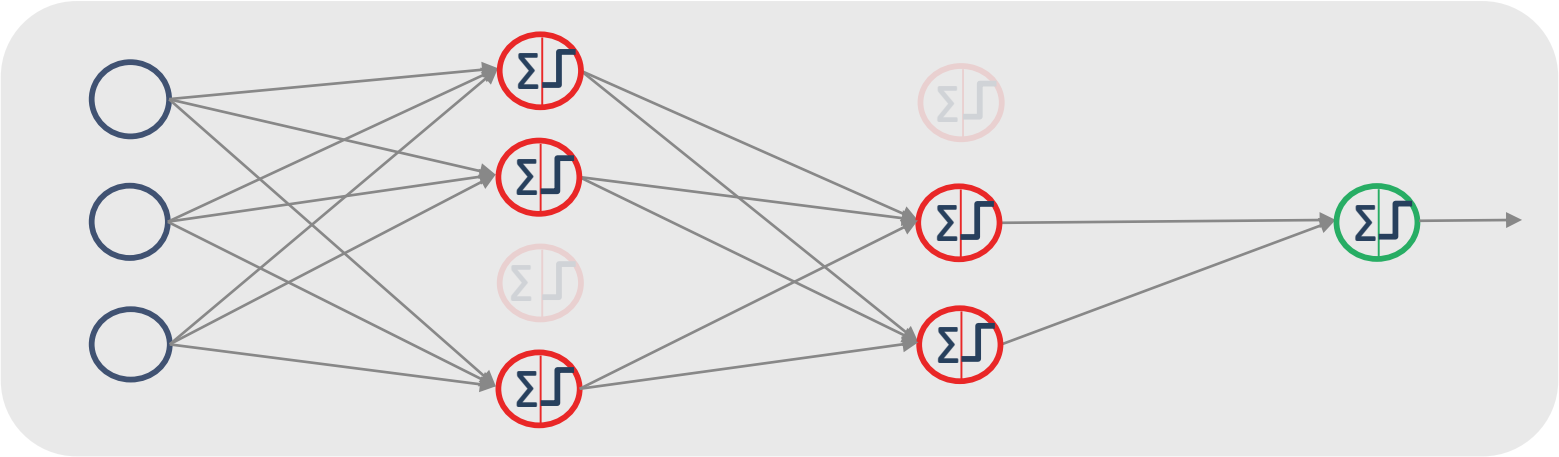
*Source paper: Srivastava *et. al.* 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929-1958.

Network architecture

- Standard network



- Dropout network

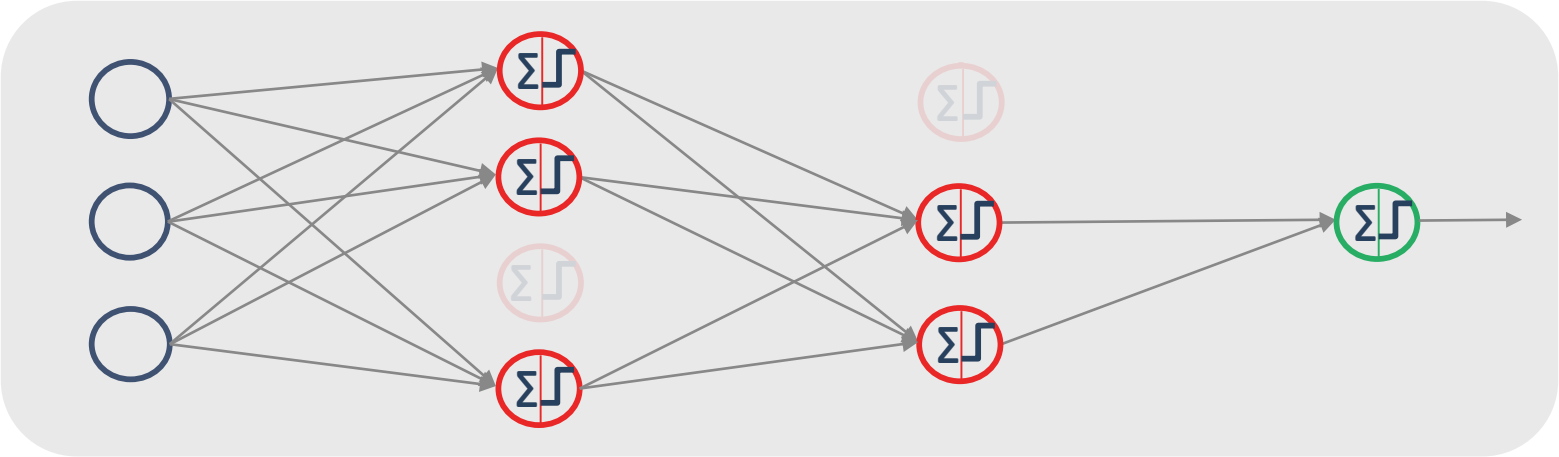


Dropout: Introduction

- Overfitting is a serious problem in large neural networks.
- Dropout tries to addresses this issue.
- Motivation: Ensemble methods are known to improve performance in most cases.
  - To achieve different models one can have different neural network architectures or have models trained on different datasets.
- Shortcomings:
  - Training different deep networks is computationally prohibitive
  - Sufficient data may not be available to train the different networks on different subsets of data properly.
- Idea: Randomly drop neurons (units) from the network during training.
  - Training: Sample from exponential number of different "thinned" networks.
  - Test: Use a single unthinned network with smaller weights to make predictions. This network approximates the effect of averaging the predictions of all thinned networks.

*Source paper: Srivastava *et. al.* 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929-1958.

Dropout model



- Output from the kth layer

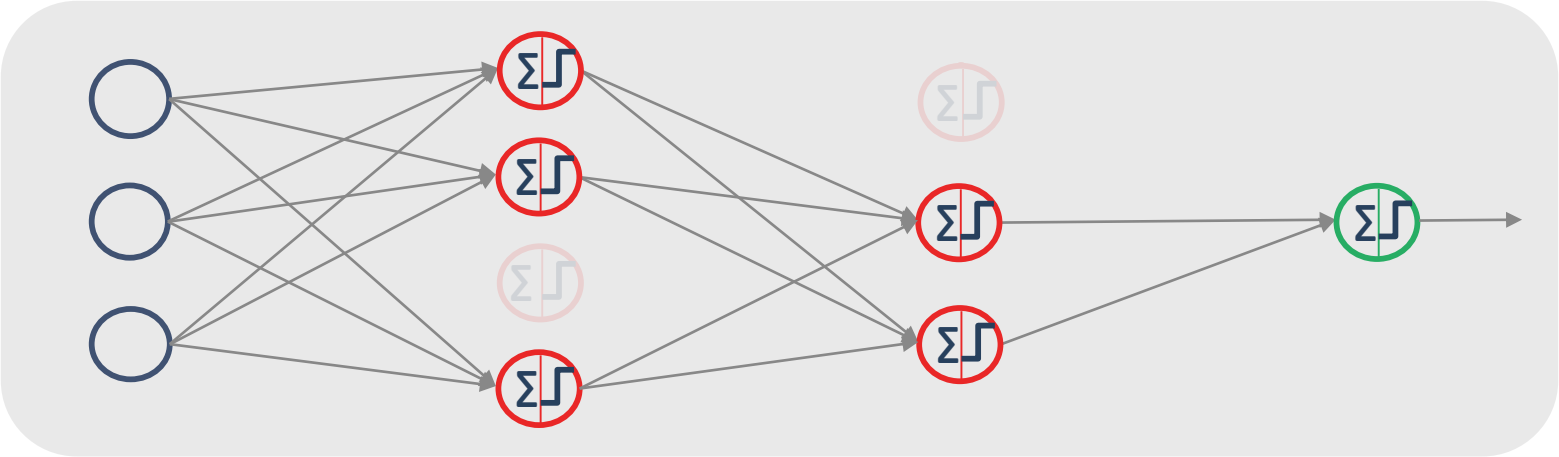
$$\mathbf{z}^{(k)} = \left[z_1^{(k)}, z_2^{(k)}, ..., z_H^{(k)}\right]$$

- Input to the $(k+1)$th layer

$$\overline{\mathbf{z}}^{(k)} = \boldsymbol{\lambda}^{(k)} * \mathbf{z}^{(k)}$$

where $\lambda_i^{(k)} \sim \text{Bernoulli}(p)$ and $*$ is the element-wise product.

Dropout model

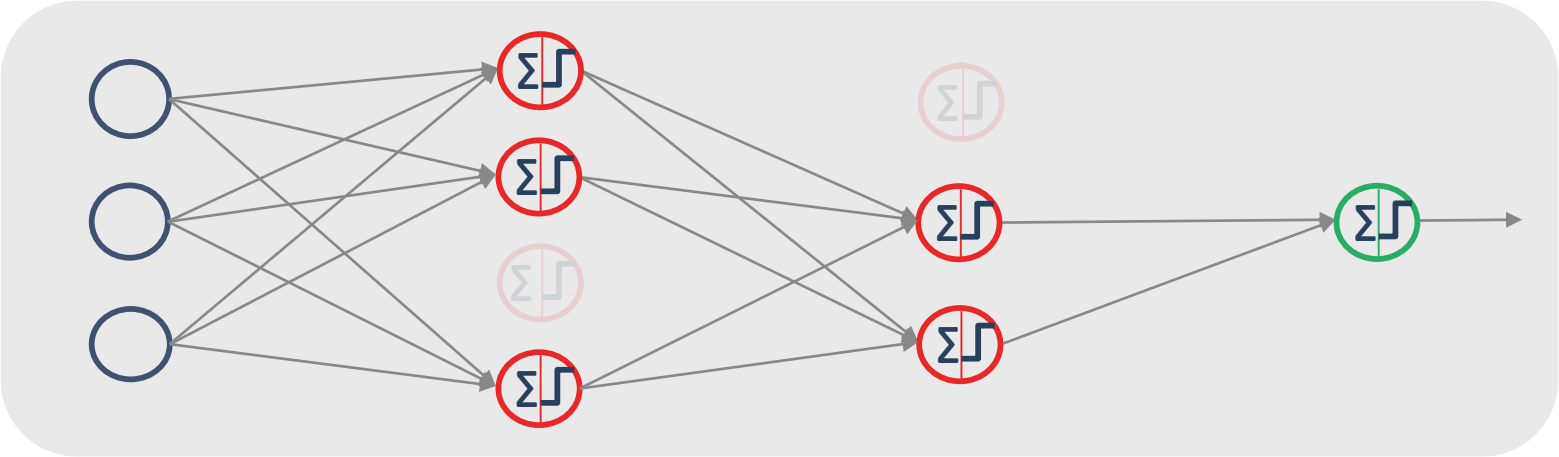


- The output of the $(k+1)$th layer

$$\mathbf{z}^{(k+1)} = \mathcal{A}\left(\left(\mathbf{w}^{(k+1)}\right)^{\mathrm{T}} \overline{\mathbf{z}}^{(k)} + \mathbf{w}_0^{(k+1)} \right)$$

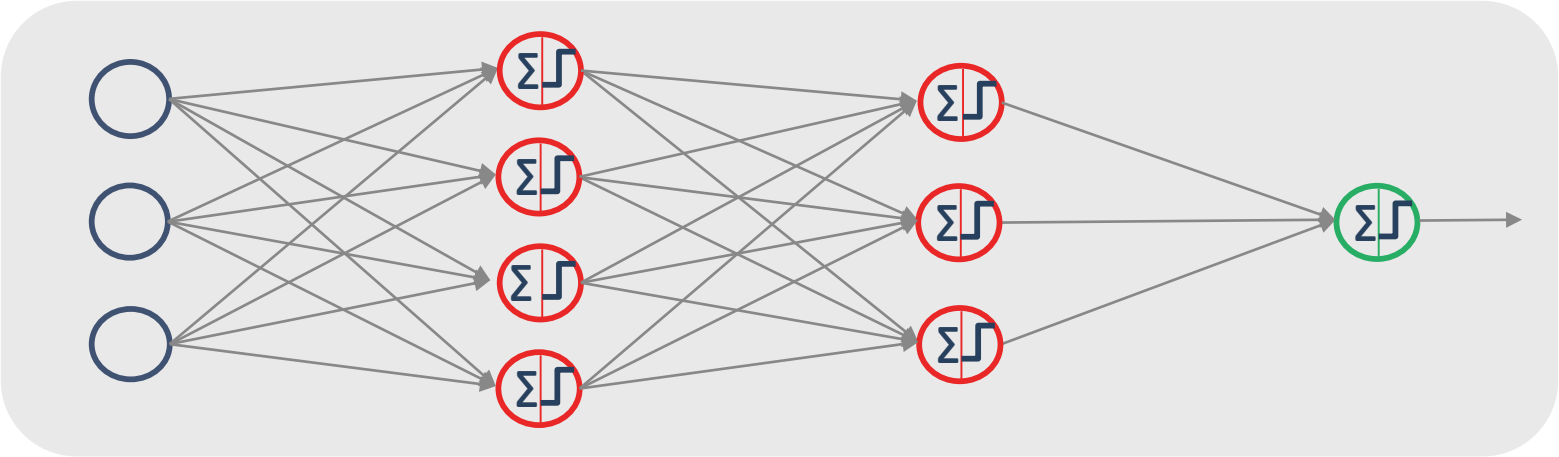
where $\mathcal{A}(\ .\ )$ is the activation function.

Training using SGD



- Training dropout networks using stochastic gradient descent.
- For a particular mini-batch
  - a thinned network is sampled for each training example.
  - the gradient of a parameter is obtained by averaging over its gradients from all the training cases in the mini-batch.
  - parameters which are absent in a training case (due to dropout) contribute a value of zero in the average gradient computations.

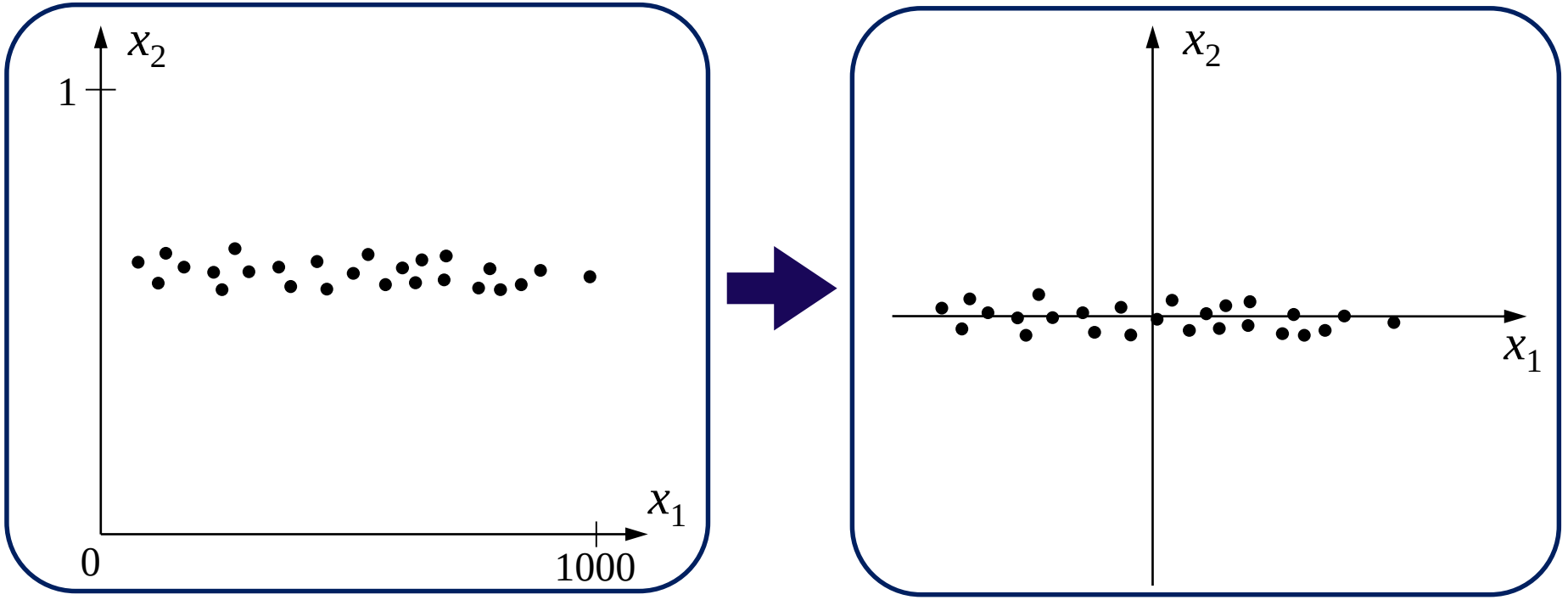
Test



- Not convenient to make predictions with all the (exponentially many) dropout networks, and then compute the average prediction.
- Idea: Use a single network without dropout.
  - The weights of this network are reduced by some factor.
  - During training, if a unit is kept with probability p, then during test the outgoing weights of that unit are multiplied by p.
- The approach ensures that for test runs the expected output of any unit is the same as the actual output.

Input Normalization

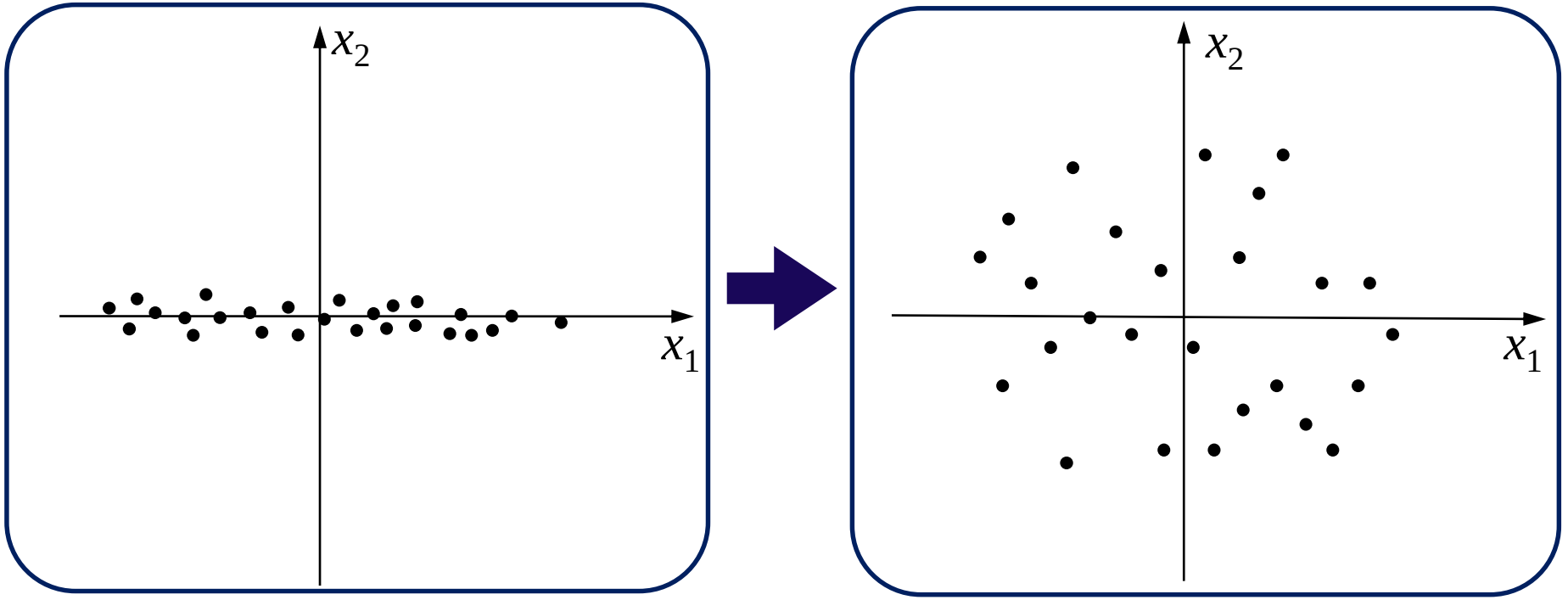
Normalization – Step 1



- Training dataset: $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$
  - Each example is D dimensional.
- Compute mean vector: $\boldsymbol{\mu} = \{\mu_1, \mu_2, ..., \mu_D\}$ where $\mu_d = \frac{1}{N}\sum_{n=1}^{N} x_d^{(n)}$
- Subtract mean to achieve zero-centered data $\{\widehat{\mathbf{x}}^{(1)}, \widehat{\mathbf{x}}^{(2)}, ..., \widehat{\mathbf{x}}^{(N)}\}$ where

$$\widehat{x}_d^{(n)} = x_d^{(n)} - \mu_d$$

Normalization – Step 2

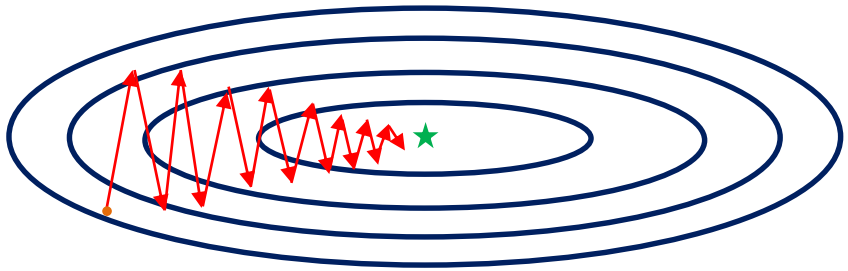


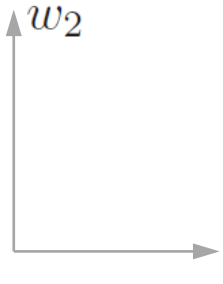
- Compute the variance in each dimension σ_d^2 using the training dataset.
- Normalized data $\left\{\overline{\mathbf{x}}^{(1)}, \overline{\mathbf{x}}^{(2)}, ..., \overline{\mathbf{x}}^{(N)}\right\}$ where

$$\overline{x}_d^{(n)} = \widehat{x}_d^{(n)} / \sigma_d$$

Normalization helps in optimization

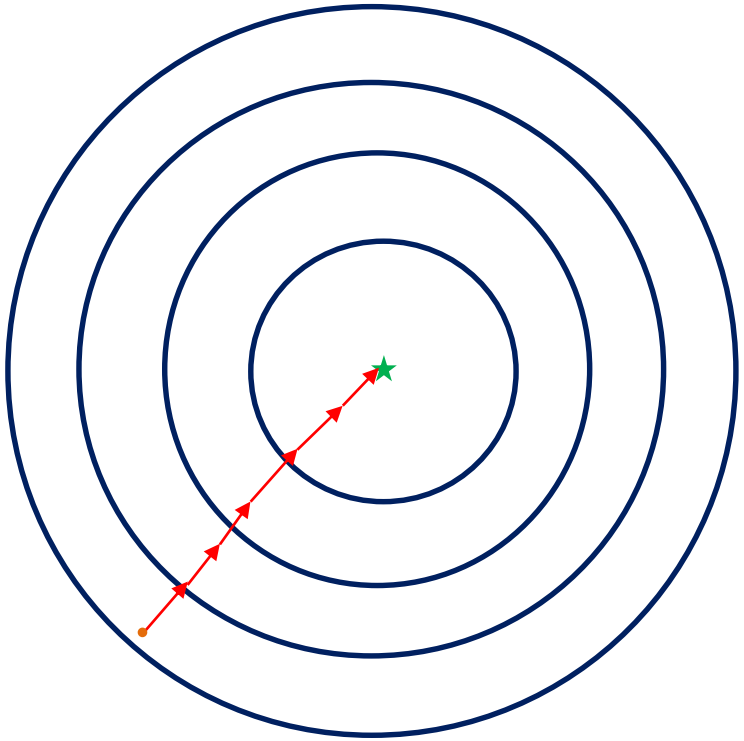
Gradient descent





Gradient descent

with input normalization



Figures for illustration only

Batch Normalization

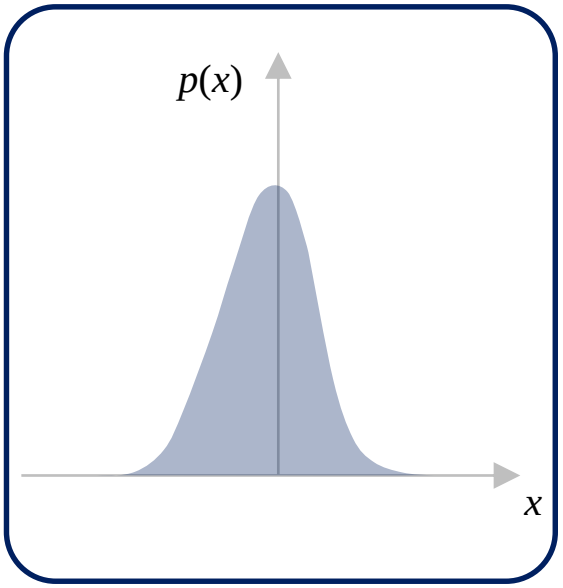
Paper

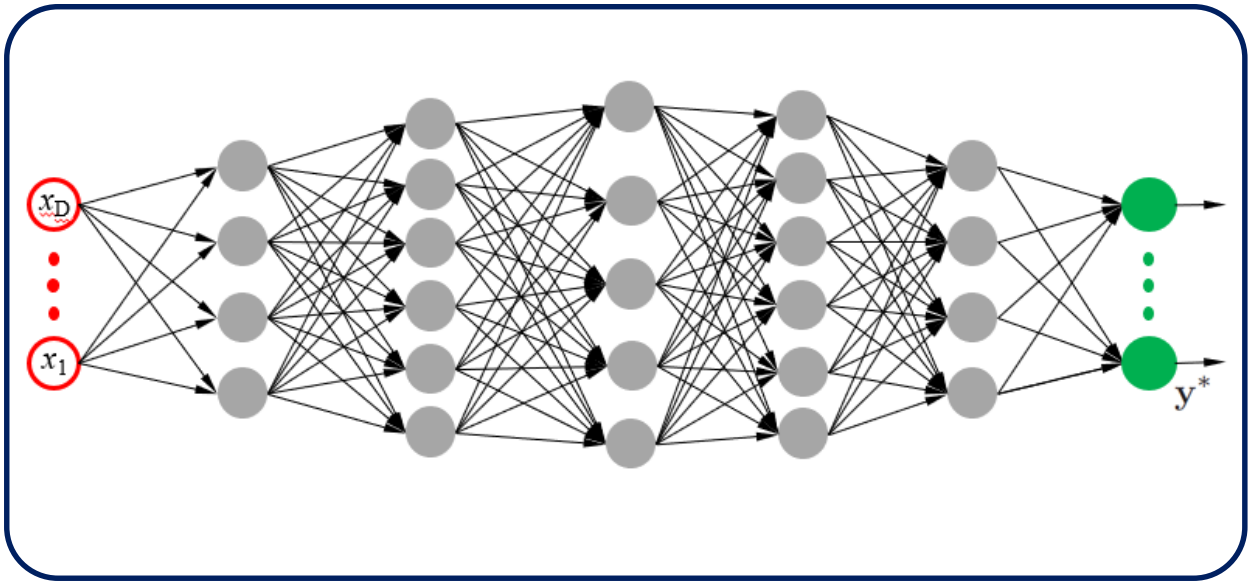
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe SIOFFE@GOOGLE.COM
Christian Szegedy SZEGEDY@GOOGLE.COM
Google, 1600 Amphitheatre Pkwy, Mountain View, CA 94043

*Source paper: Ioffe and Szegedy 2015. Batch normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of 32nd ICML.

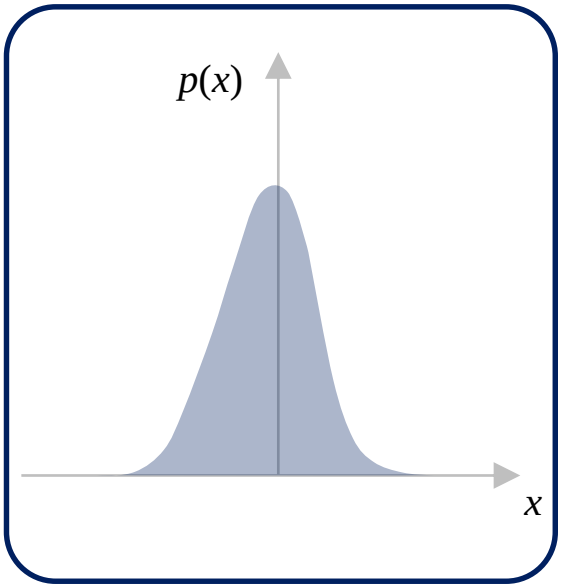
Motivation

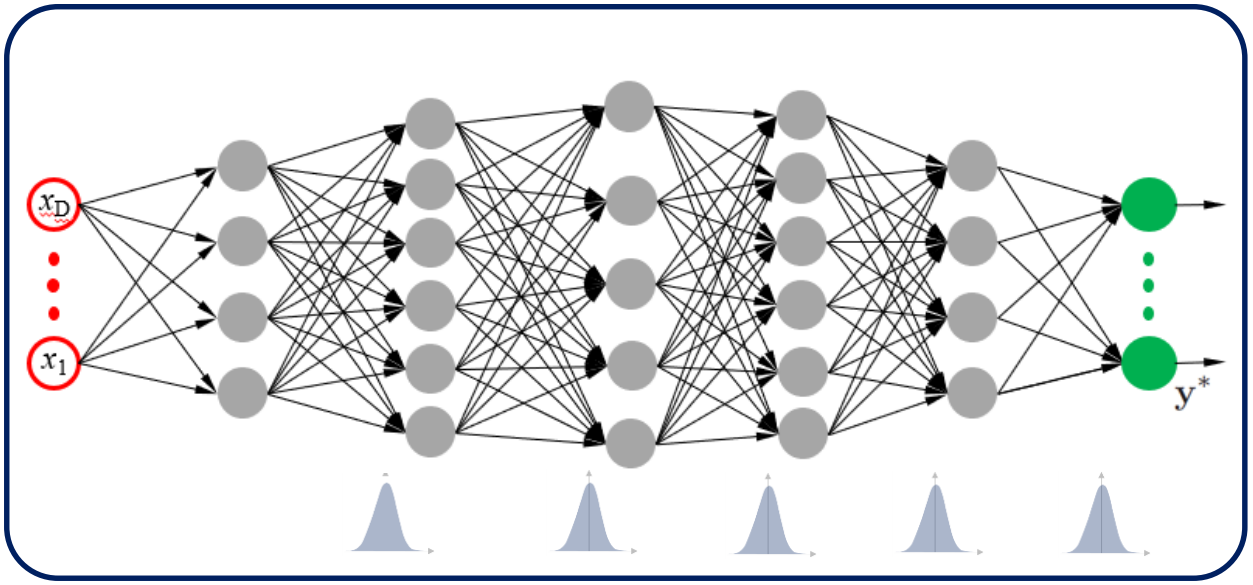




- To achieve better generalization of a learning system, it is preferable to have the same distribution between the training and the test data.
- Neural networks have inputs to each layer.
  - While training if the distribution of layers' inputs change then the layers need to constantly adapt to the new distribution.
- Batch normalization attempts to improve the training speed by normalizing layer inputs.

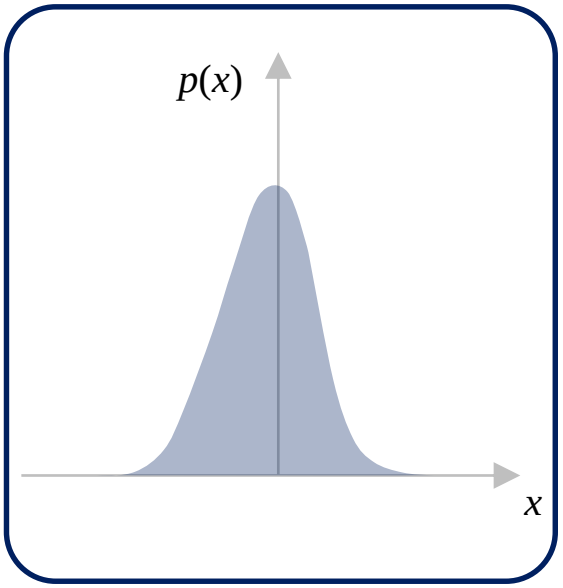
Idea

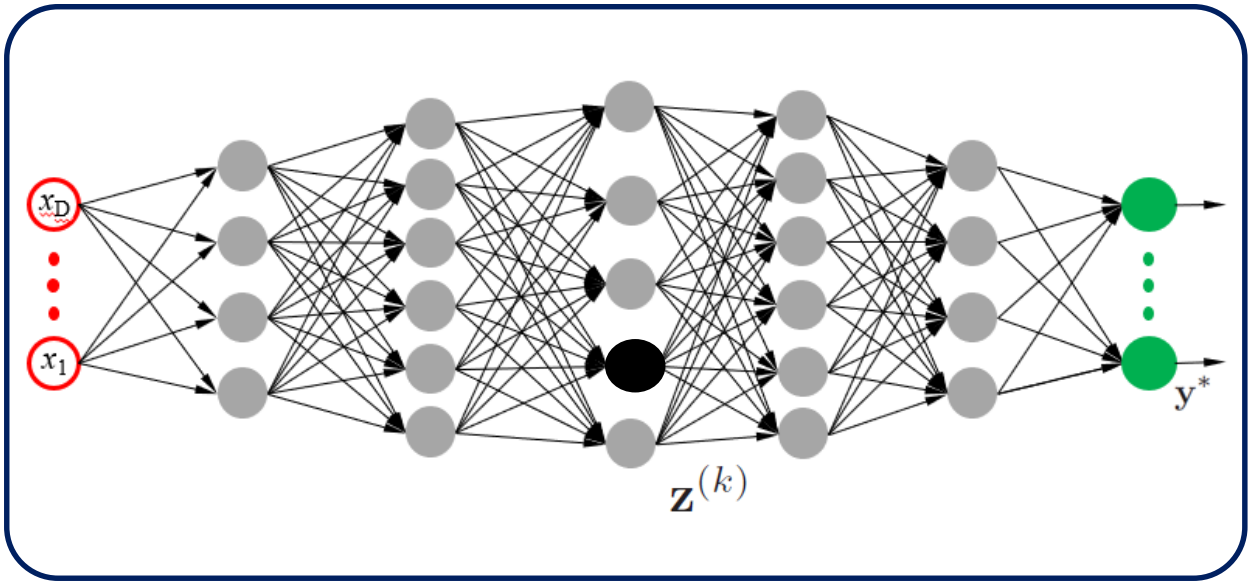




- Previous research works have showed that network training converges faster if the inputs are normalized.
- Normalize inputs to each layer of the network.

Method





- Consider a mini-batch of size M.
- Suppose we are looking at activations at the hth node of the kth layer for the mini-batch: $\{z_{h1}^{(k)}, z_{h2}^{(k)}, ..., z_{hM}^{(k)}\}$.

Method

- Procedure:
  - Compute the mini-batch mean

$$\mu_h^{(k)} = \frac{1}{M} \sum_{m=1}^{M} z_{hm}^{(k)}$$

  - Compute the mini-batch variance

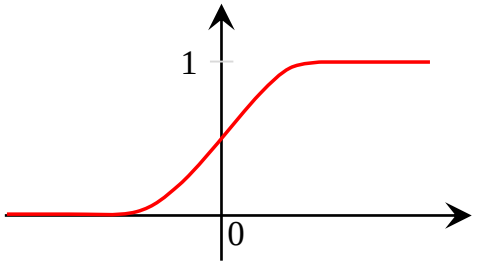
$$\left(\sigma_h^{(k)}\right)^2 = \frac{1}{M-1} \sum_{m=1}^{M} \left(z_{hm}^{(k)} - \mu_h^{(k)}\right)^2$$

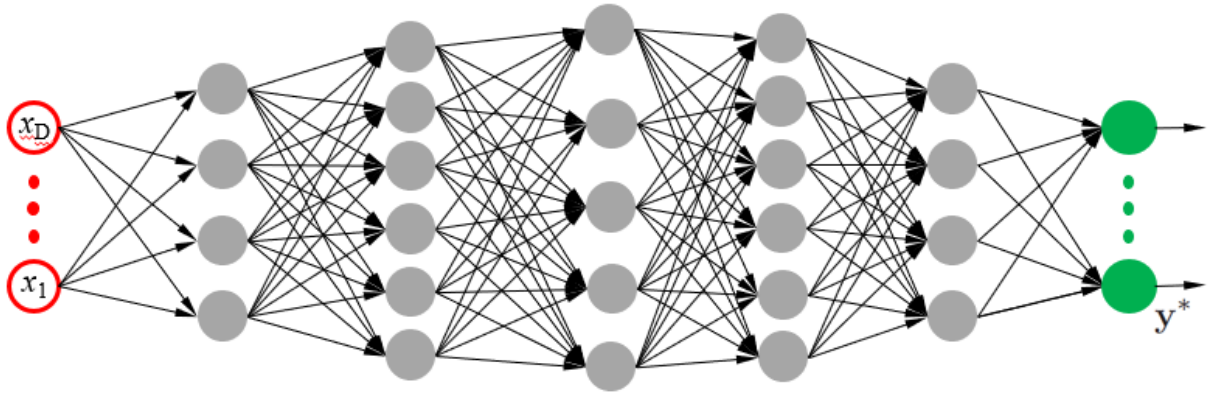
  - Normalize to make the activations zero mean and unit variance

$$\overline{z}_h^{(k)} = \frac{z_h^{(k)} - \mu_h^{(k)}}{\sqrt{\left(\sigma_h^{(k)}\right)^2 + \epsilon}}$$

Simple normalization: shortcomings

$$\mathcal{A}(z) = \frac{1}{1 + \exp(-z)}$$

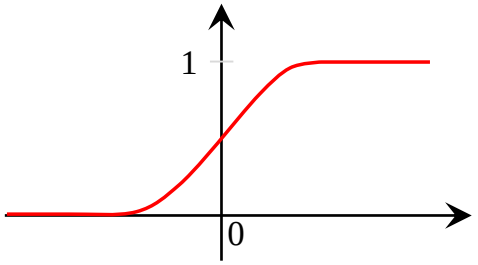


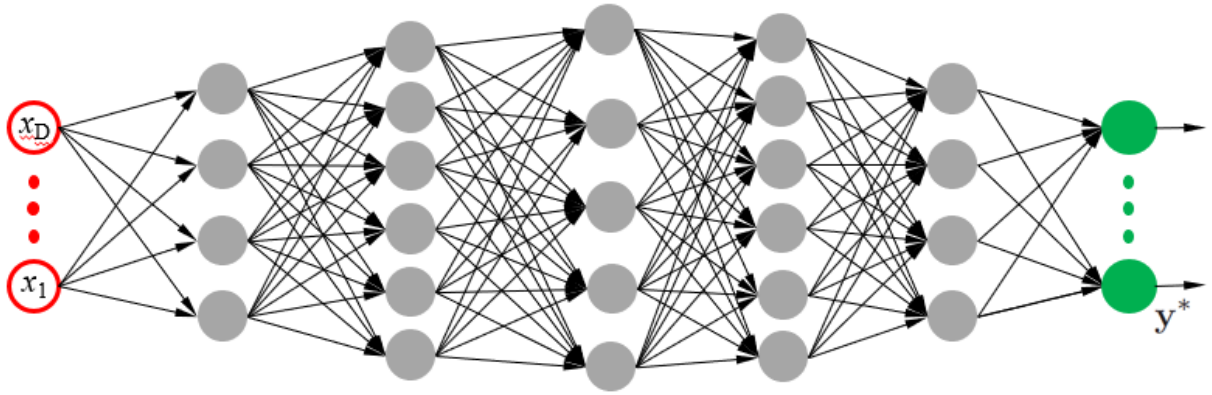


- Simple normalization of of each input can alter the representational capability of a layer
  - For example, a sigmoid activation with (simple) normalized inputs will be constrained to the linear regime of nonlinearity.

Transformation

$$\mathcal{A}(z) = \frac{1}{1 + \exp(-z)}$$





- To address the issue a pair of parameters $\gamma_h^{(k)}$ and $\beta_h^{(k)}$ are introduced for each activation
  - $\gamma_h^{(k)}$ scales the normalized value.
  - $\beta_h^{(k)}$ shifts the normalized value.
- These parameters are learnt along with the original model parameters.
  - They restore the representation power of the model.

Transformation

- Procedure:
  - Scale and shift

$$\widehat{z}_h^{(k)} = \gamma_h^{(k)} \overline{z}_h^{(k)} + \beta_h^{(k)}$$

- Can recover the original distribution using

$$\gamma_h^{(k)} = \sqrt{\left(\sigma_h^{(k)}\right)^2 + \epsilon}$$

and

$$\beta_h^{(k)} = \mu_h^{(k)}$$

Another paper

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

1 Introduction

Over the last decade, deep learning has made impressive progress on a variety of notoriously difficult tasks in computer vision [16, 7], speech recognition [5], machine translation [29], and game-playing [18, 25]. This progress hinged on a number of major advances in terms of hardware, datasets [15, 23], and algorithmic and architectural techniques [27, 12, 20, 28]. One of the most prominent examples of such advances was batch normalization (BatchNorm) [10].

At a high level, BatchNorm is a technique that aims to improve the training of neural networks by stabilizing the distributions of layer inputs. This is achieved by introducing additional network layers that control the first two moments (mean and variance) of these distributions.

The practical success of BatchNorm is indisputable. By now, it is used by default in most deep learning models, both in research (more than 6,000 citations) and real-world settings. Somewhat shockingly, however, despite its prominence, we still have a poor understanding of what the effectiveness of BatchNorm is stemming from. In fact, there are now a number of works that provide alternatives to BatchNorm [1, 3, 13, 31], but none of them seem to bring us any closer to understanding this issue. (A similar point was also raised recently in [22].)

Currently, the most widely accepted explanation of BatchNorm's success, as well as its original motivation, relates to so-called *internal covariate shift* (ICS). Informally, ICS refers to the change in the distribution of layer inputs caused by updates to the preceding layers. It is conjectured that such continual change negatively impacts training. The goal of BatchNorm was to reduce ICS and thus remedy this effect.

Even though this explanation is widely accepted, we seem to have little concrete evidence supporting it. In particular, we still do not understand the link between ICS and training performance. The chief goal of this paper is to address all these shortcomings. Our exploration lead to somewhat startling discoveries.

*Equal contribution.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.