Dropout, Batch normalization

DRIPTA MJ

Department of Mathematics

RAMAKRISHNA MISSION VIVEKANANDA EDUCATIONAL AND RESEARCH INSTITUTE

BELUR MATH, INDIA

Machine Learning

Sem 3, 2018-19

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.

- 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.
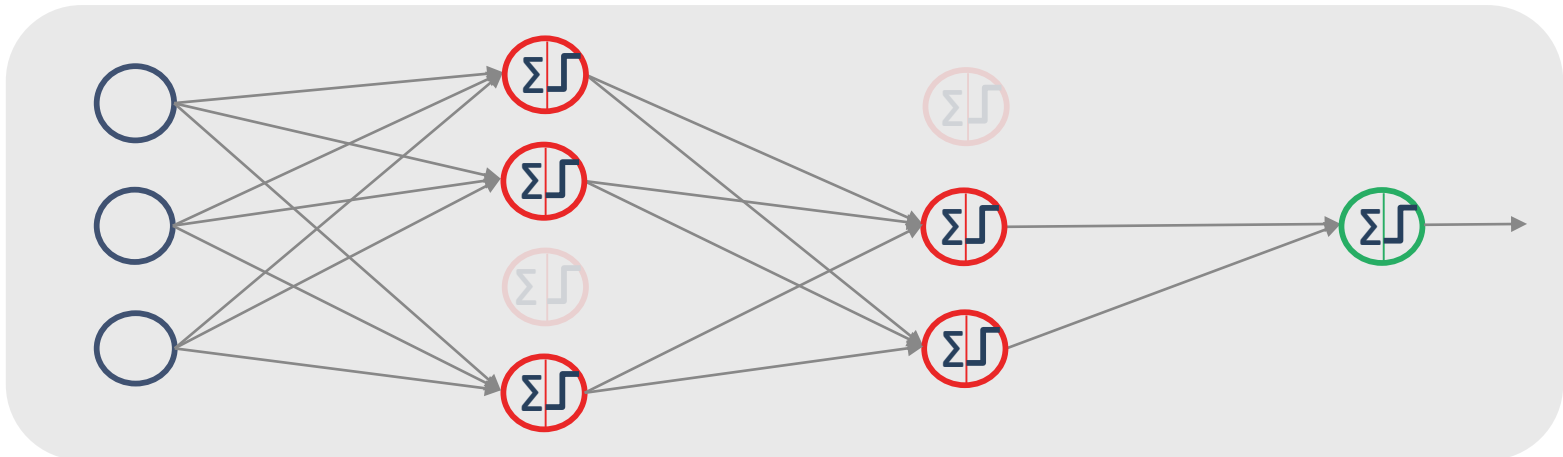
Network architecture

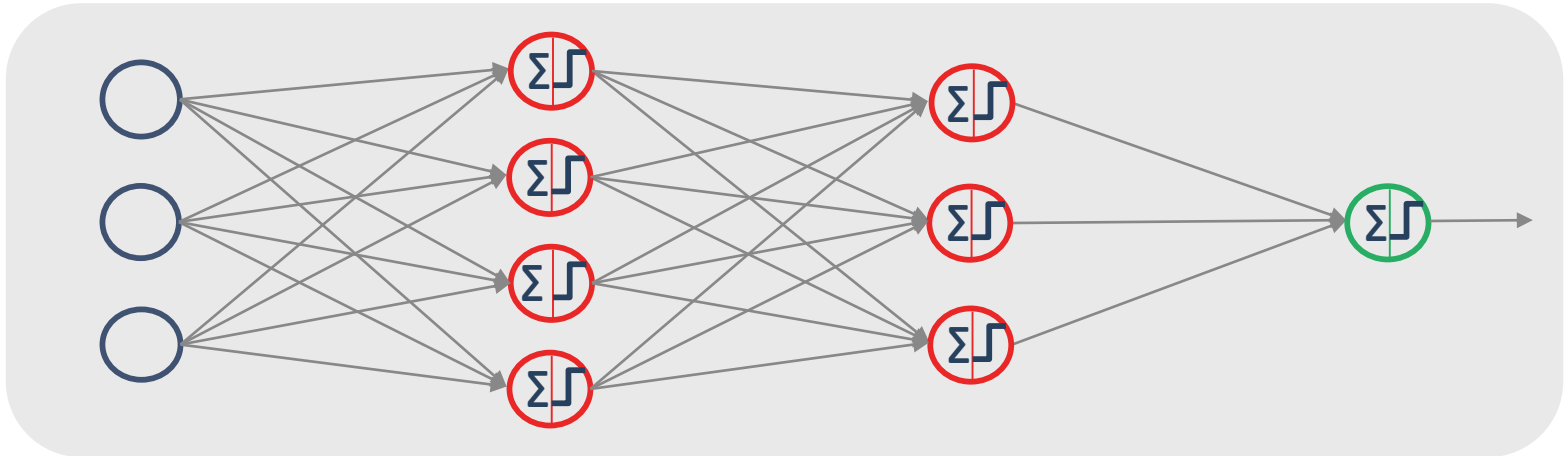- Standard network
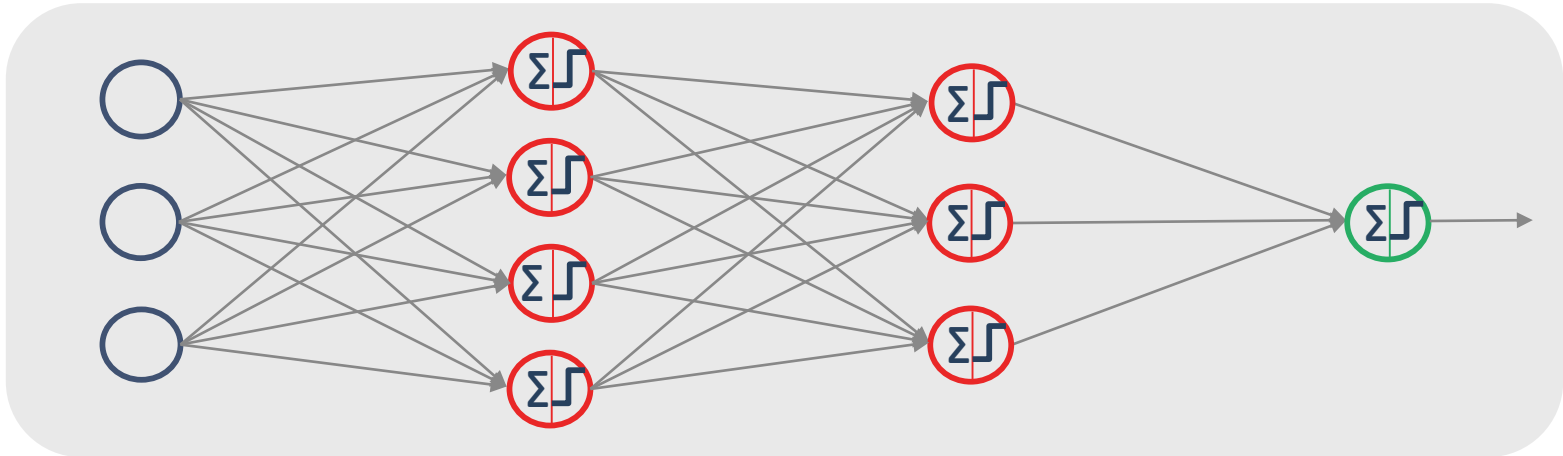
- Dropout network

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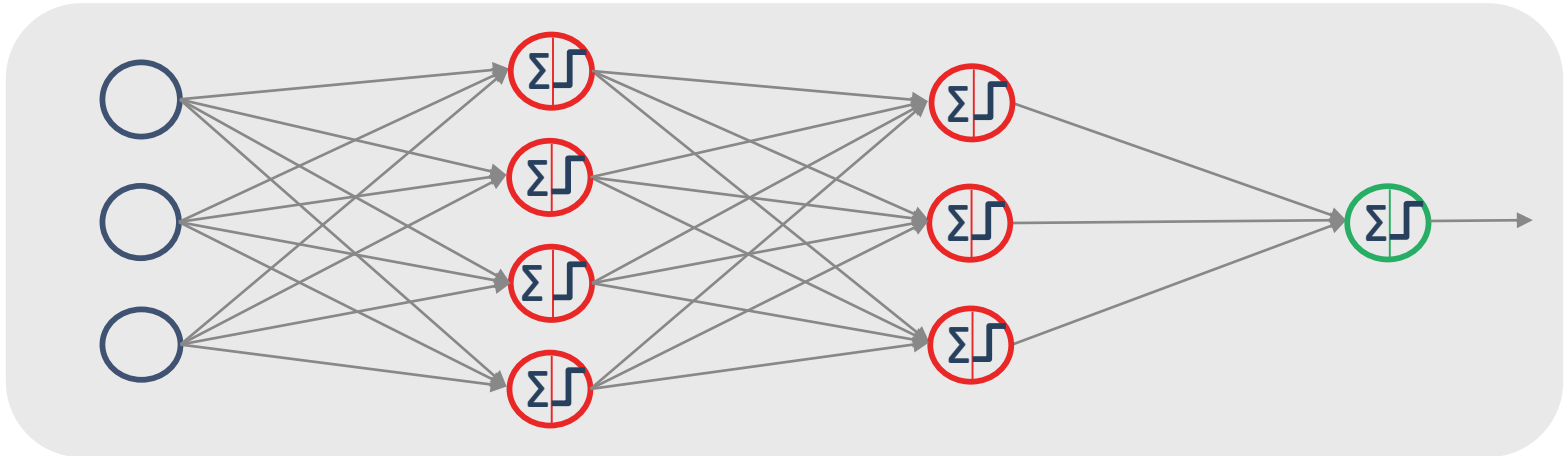
- The output of the $(k+1)$th layer

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

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.

- 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.

Batch Normalization

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.

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: $\left\{ z_{h1}^{(k)}, z_{h2}^{(k)}, ..., z_{hM}^{(k)} \right\}$.

- Procedure:
  - Compute the mini-batch mean

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

  - Compute the mini-batch variance

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

  - Normalize to make the activations zero mean and unit variance

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

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: $\left\{ z_{h1}^{(k)}, z_{h2}^{(k)}, ..., z_{hM}^{(k)} \right\}$.

- 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

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

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.