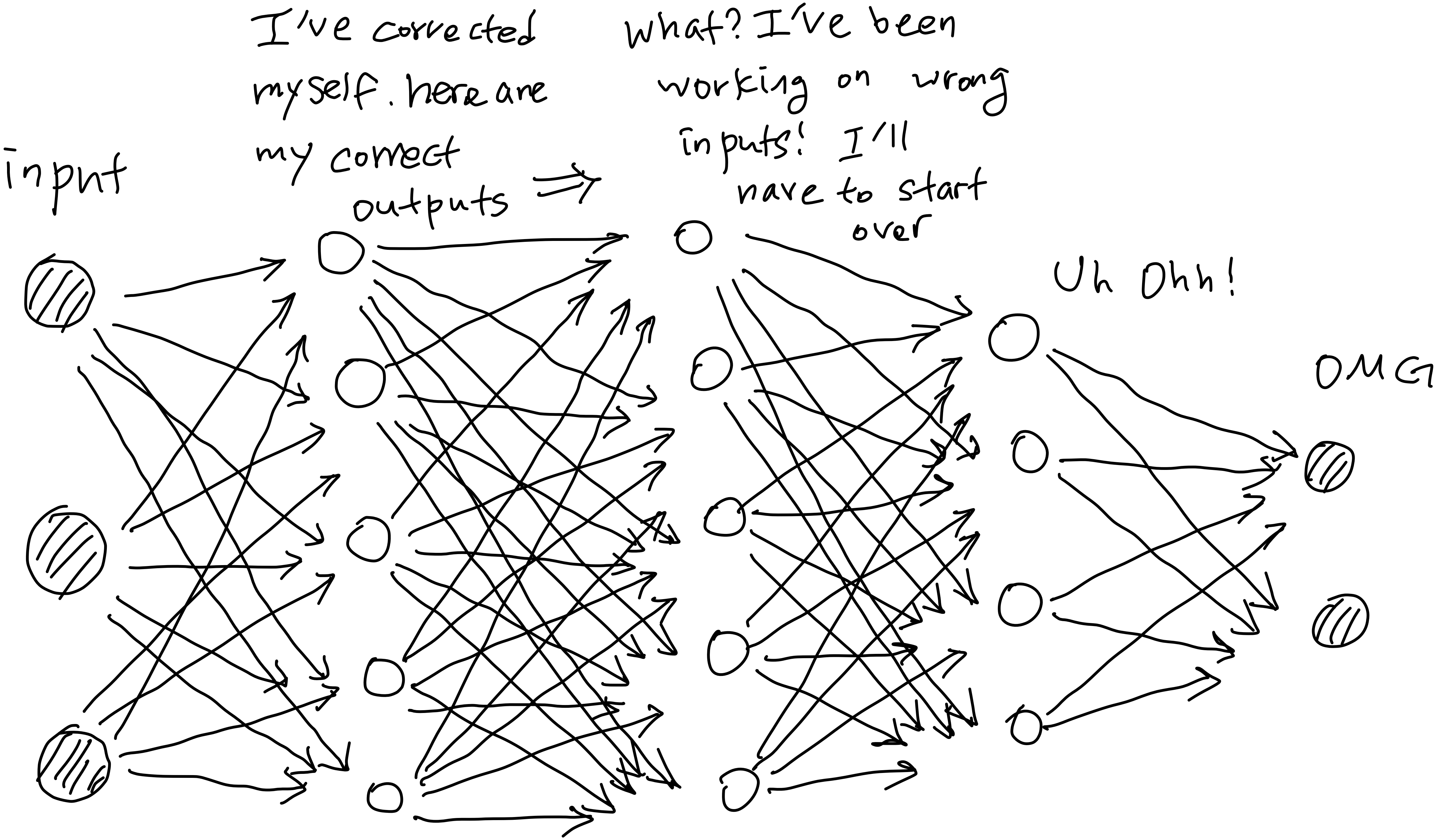
Batch Normalization:

Accelerating Deep Network Training by Reducing Internal Covariate Shift

0. Abstract

- distribution of each layer’s input changes during training
- because parameters of the previous layers change
- slows down the training by requiring lower learning rates and careful parameter initialization
- “internal covariate shift”

Internal Covariate Shift

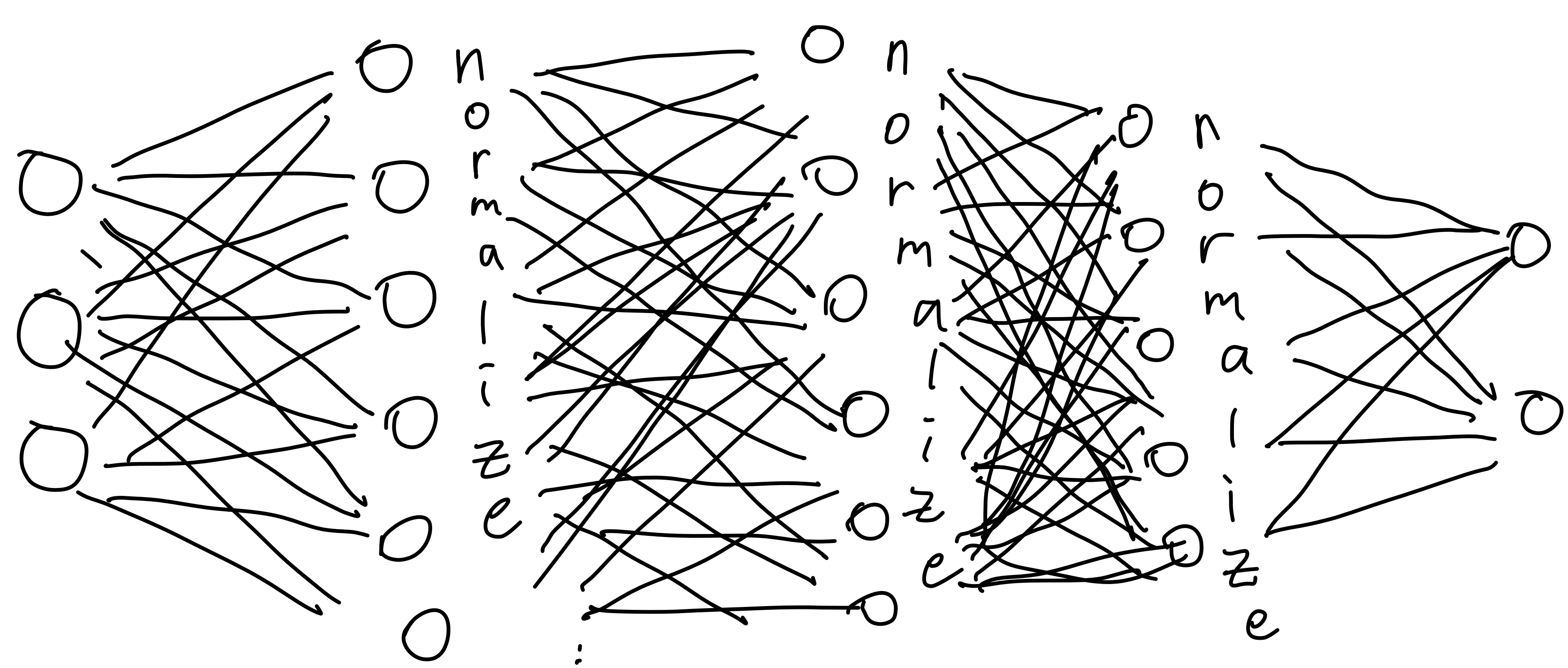


- address the problem by normalizing layer inputs
- 1) make normalization a part of the model architecture
- 2) perform normalization for each training mini-batch

Normalization is to convert the distribution of all inputs to have mean=0 and standard deviation=1

⇒ Batch Normalization adds normalization layer between each layers

☆ normalization has to be done separately for each dimension, over the mini-batches.



1. Introduction

- 1) Stochastic Gradient Descent
- - optimize the parameters
- - so as to minimize the loss

$$\Theta = \underset{\Theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, \Theta)$$

- $X_{1 \dots N}$ training data set

  $X_{1 \dots m}$ mini-batch

- mini-batch used to approximate the gradient of the loss function

$$\frac{1}{m}\frac{\partial \ell(x_i, \theta)}{\partial \theta}$$

- 1) quality improves as batch size increases

- 2) computation over a batch: much more efficient

- inputs to each layer are affected by the parameters of all preceding layers

- **covariate shift** : input to the learning system changes
- solution: domain adaptation
- notion of covariate shift can be applied to its parts
- **internal covariate shift** : change in the distributions of internal nodes of a deep network
- **Batch Normalization** : reduce internal covariate shift
- a normalization step that fixes the means and variances of layer inputs

2. Towards Reducing Internal Covariate Shift

- training converges faster if its inputs are whitened (linearly transformed to have zero means and unit variances, decorrelated)

- gradient of the loss with respect to the model parameters to account for normalization and for its dependence on the parameter

- normalization can be written as a transformation

- $\hat{x} = Norm(x, \mathcal{X})$

given training example x
on all examples $\mathcal{X}$

- for backpropagation, we would need to compute the Jacobians

$$\frac{\partial \mathrm{Norm}(x, \mathcal{X})}{\partial x} \quad \text{and} \quad \frac{\partial \mathrm{Norm}(x, \mathcal{X})}{\partial \mathcal{X}}$$

- but, whitening the layer input is expensive

- input normalization that is differentiable and does not require the analysis of the entire training set after every parameter update

3. Normalization via Mini-Batch Statistics

- 1) normalize each scalar feature independently

d-dimensional input $X = (x^{(1)}, \cdots, x^{(d)})$

normalize each dimension

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$$

- expectation and variance are computed over the training data set make sure that transformation inserted in the network can represent the identity transform

$\Rightarrow$ introduce $\gamma^{(k)}, \beta^{(k)}$ for each activation $x^{(k)}$

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

$\gamma^{(k)}, \beta^{(k)}$ are learned along original model parameters

- 2) each mini-batch produces estimates of the mean and variance of each activation

mini-batch B of size m

focus on a particular activation $x^{(k)}$
(omit k for clarity)

$$B = \{ x_{1 \dots m} \}$$

- "Batch Normalizing Transform"

let the normalized values be $\hat{x}_{1\ldots m}$

linear transformations $y_{1\ldots m}$

"Batch Normalizing Transform"

$BN_{\gamma,\beta} : x_{1\ldots m} \rightarrow y_{1\ldots m}$

ε is for numerical stability

Algorithm 1. Batch Normalizing Transform,

applied to activation x over a mini-batch

- BN transform can be added to a network to manipulate any activation

- $\underbrace{\text{the scaled and shifted values}}_{}$ are passed to other network layers

  y

- BN transform is a differentiable transform

Algorithm 1. Batch Normalizing Transform. applied to activation x over a mini-batch

Input: values of x over a mini-batch: $\mathcal{B} = \{x_{1 \dots m}\}$;

Parameters to be learned: γ, β

output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

// mini-batch mean

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

// mini-batch variance

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

// normalize

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

// scale and shift $y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$

- the process of back propagation

$$\frac{\partial l}{\partial \hat{x}_i} = \frac{\partial l}{\partial y_i} \cdot \gamma$$

$$\frac{\partial l}{\partial \sigma_B^2} = \sum_{i=1}^{m} \frac{\partial l}{\partial \hat{x}_i} \cdot (x_i - \mu_B) \cdot \frac{-1}{2} \cdot (\sigma_B^2 + \varepsilon)^{-3/2}$$

$$\frac{\partial l}{\partial \mu_B} = \left(\sum_{i=1}^{m} \frac{\partial l}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_B^2 + \varepsilon}} \right) + \frac{\partial l}{\partial \sigma_B^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_B)}{m}$$

$$\frac{\partial l}{\partial x_i} \bullet = \frac{\partial l}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \varepsilon}} + \frac{\partial l}{\partial \sigma_B^2} \cdot \frac{2(x_i - \mu_B)}{m} + \frac{\partial l}{\partial \mu_B} \cdot \frac{1}{m}$$

$$\frac{\partial l}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial l}{\partial y_i} \cdot \hat{x}_i \qquad \frac{\partial l}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial l}{\partial y_i}$$

3.1 Training and Inference with Batch-Normalized Networks

Algorithm 2. Training a Batch-Normalized Network

Input: Network N with trainable parameters θ;

subset of activations $\{x^{(k)}\}_{k=1}^{K}$

output: batch-normalized network for inference, N_{BN}^{inf}

1: $N_{BN}^{tr} \leftarrow N$ // training BN network

2: for $k = 1 \ldots K$ do

3: add transformation $y^{(k)} = BN_{\gamma^{(k)}, \beta^{(k)}}(x^{(k)})$

to N_{BN}^{tr} (Alg. 1)

4: modify each layer in N_{BN}^{tr} with input $x^{(k)}$ to take $y^{(k)}$ instead

5: end for

6: train N_{BN}^{tr} to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$

7: $N_{BN}^{inf} \leftarrow N_{BN}^{tr}$ // inference BN network with frozen parameters

8: for $k = 1 \dots K$ do

9: // for clarity, $x \equiv x^{(k)}$, $\gamma \equiv \gamma^{(k)}$, $\mu_B \equiv \mu_B^{(k)}$ ...

10: process multiple training mini-batches B, each of size m, and average over them

$$E[x] \leftarrow E_B[\mu_B]$$

$$Var[x] \leftarrow \frac{m}{m-1} E_B[\sigma_B^2]$$

11: In N_{BN}^{inf}, replace the transform $y = BN_{\gamma,\beta}(x)$

with $y = \frac{\gamma}{\sqrt{Var[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma \cdot E[x]}{\sqrt{Var[x]+\epsilon}}\right)$

12: end for

3.2 Batch-Normalized Convolutional Networks

$$z = g(Wu + b)$$

W, b learned parameters of the model

$g(\cdot)$ non-linearity (sigmoid / ReLU)

$\Rightarrow$ add the BN transform immediately before the non-linearity, by normalizing $x = Wu + b$

$$z = g(BN(Wu))$$

- for convolutional layers, want normalization to obey the convolutional property

- jointly normalize all the activations in a mini-batch, over all locations

-

3.3 Batch Normalization enables higher learning rates

Prevents the training from getting stuck in the saturated regimes of nonlinearities.

Training more resilient to the parameter scale.

With Batch Normalization, back-propagation through a layer is unaffected by the scale of its parameters.

The scale does not affect the layer Jacobian nor the gradient propagation.

3.4 Batch Normalization regularizes the model

4. Experiments

4.1 Activations over time
- To verify the effects of internal covariate shift on training
- consider the problem of predicting the digit class on the MNIST dataset

-input: 28x28 binary image
-network: 3 fully-connected hidden layers with 100 activations each

<result> Figure 1