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

Sergey Ioffe, Christian Szegedy (2015)

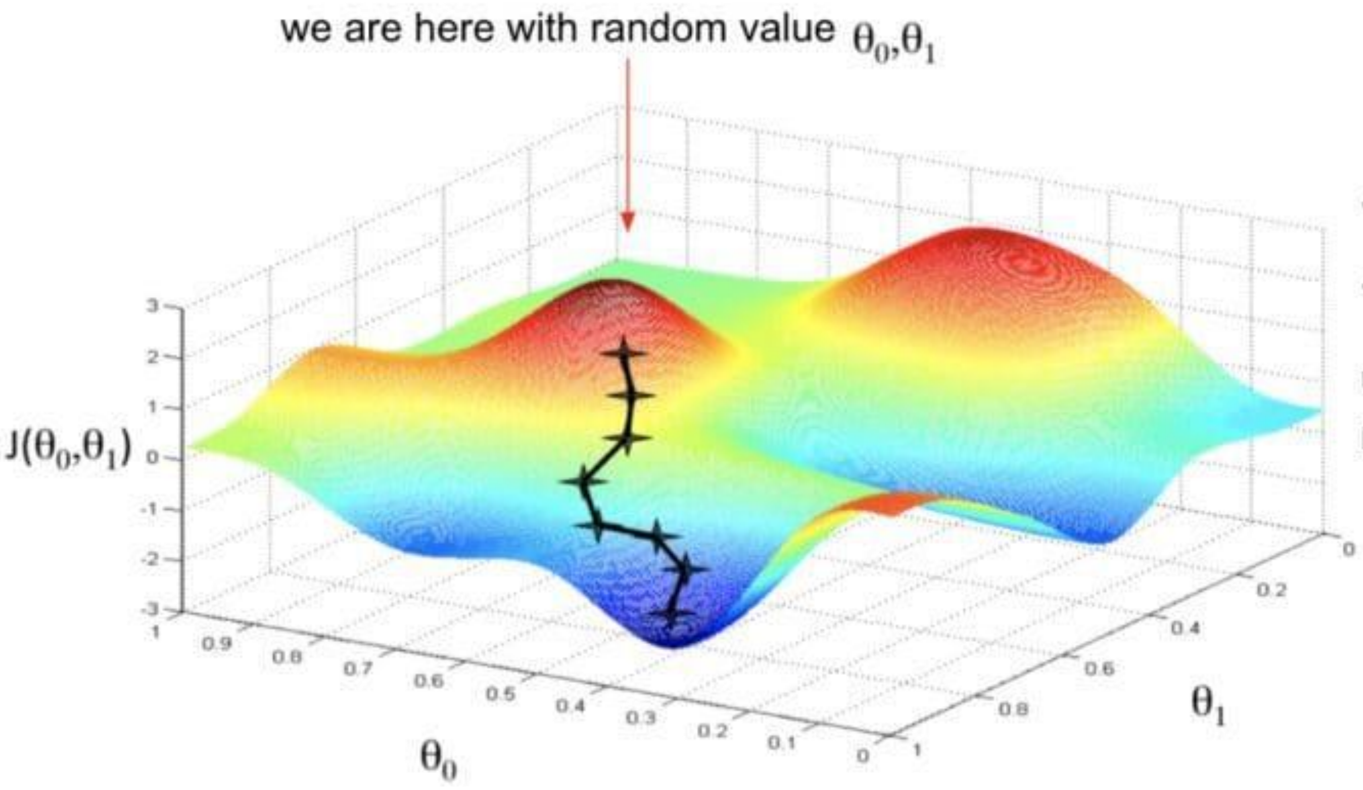
Presented by:

- Pin-Chieh Liao
- María Victoria Liendro

Quick Review: Gradient Descent

Gradient Descent Update Rule

$$w_{t+1} = w_t - \eta \nabla w_t$$



- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$

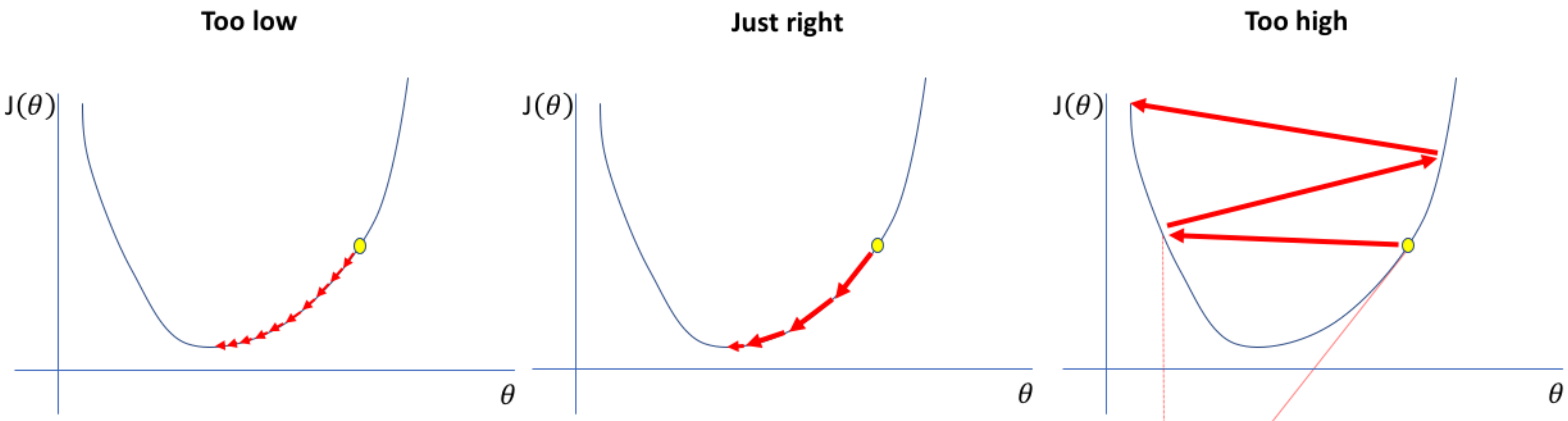
until we hopefully end up at a minimum

Quick Review: Learning Rate

Gradient Descent Update Rule

$$w_{t+1} = w_t - \eta \nabla w_t$$

Learning rate



A small learning rate requires many updates before reaching the minimum point

The optimal learning rate swiftly reaches the minimum point

Too large of a learning rate causes drastic updates which lead to divergent behaviors

Improvement for Neural Network

Gradient Descent Update Rule

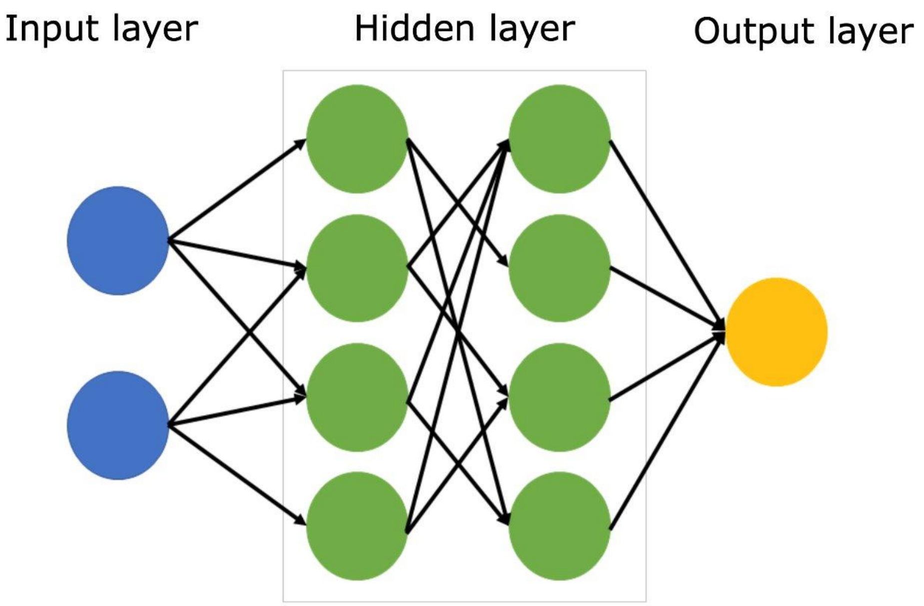
$$w_{t+1} = w_t - \eta \nabla w_t$$

Learning rate gradient

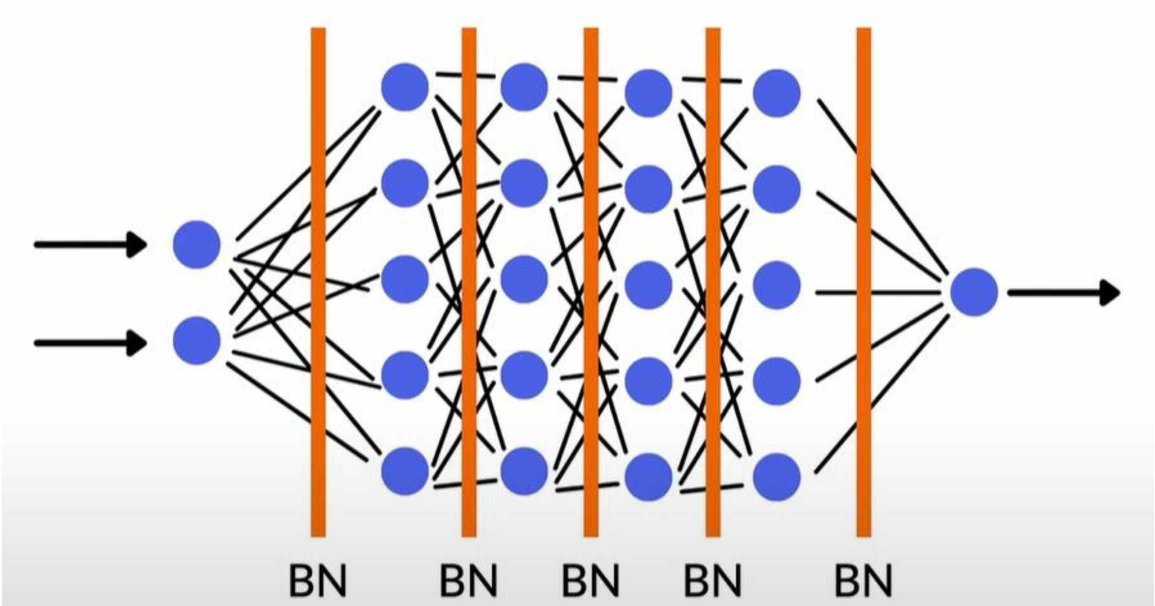
- How to find a good learning rate?
  - Too large- Might not even converge
  - Too small- Slow converge
- Exploding/Vanishing Gradient
  - Exploding- Never converge or crashed model
  - Vanishing- Never update parameters
- Computational Expensive

Batch Normalization

- Technique that normalizes the input of each layer
- It aims to reduce Internal Covariate Shift



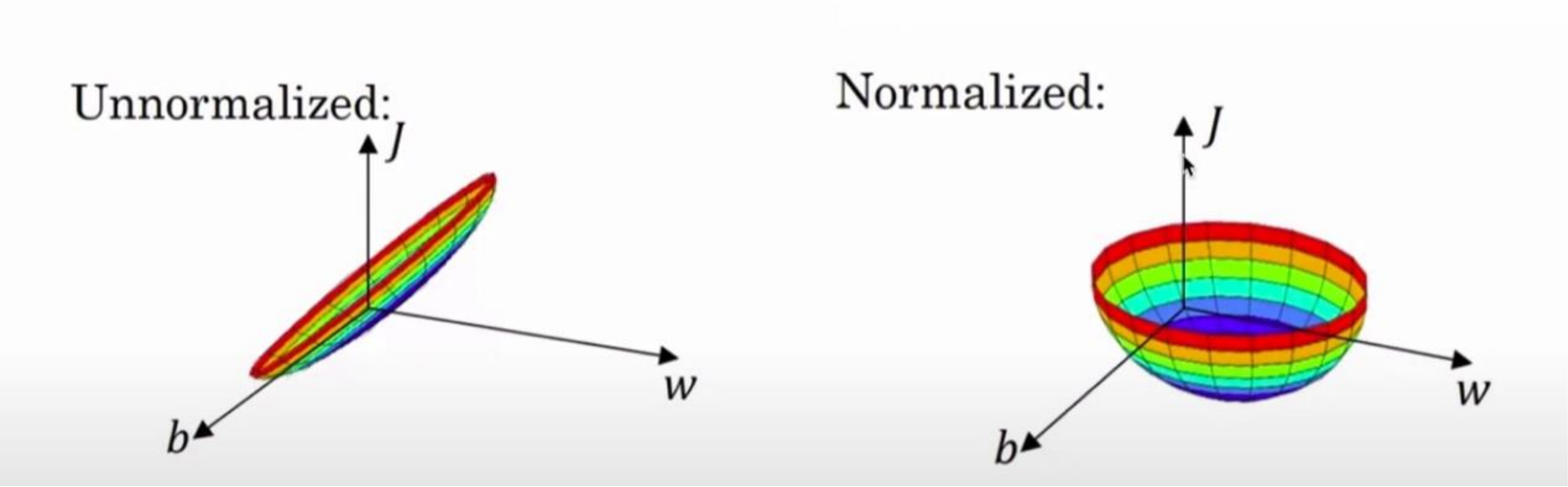
Artificial neural networks



Neural networks with BN

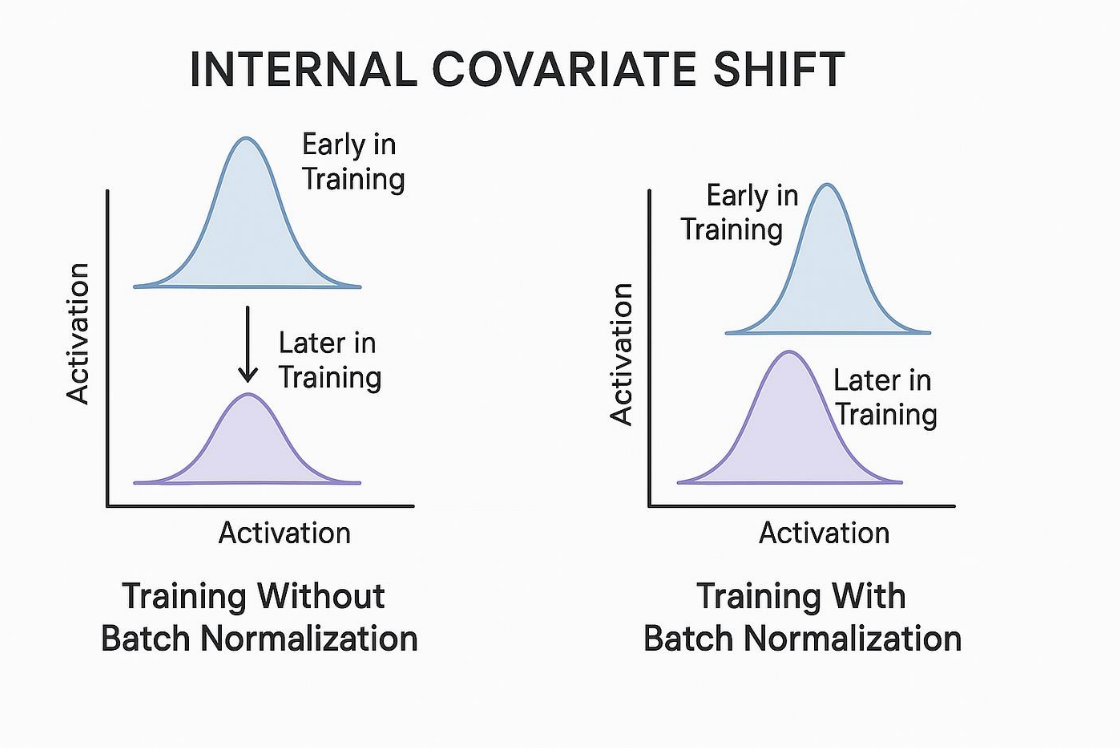
Why Normalization

- If it works for input to hidden, why not try implement in hidden to hidden layer?



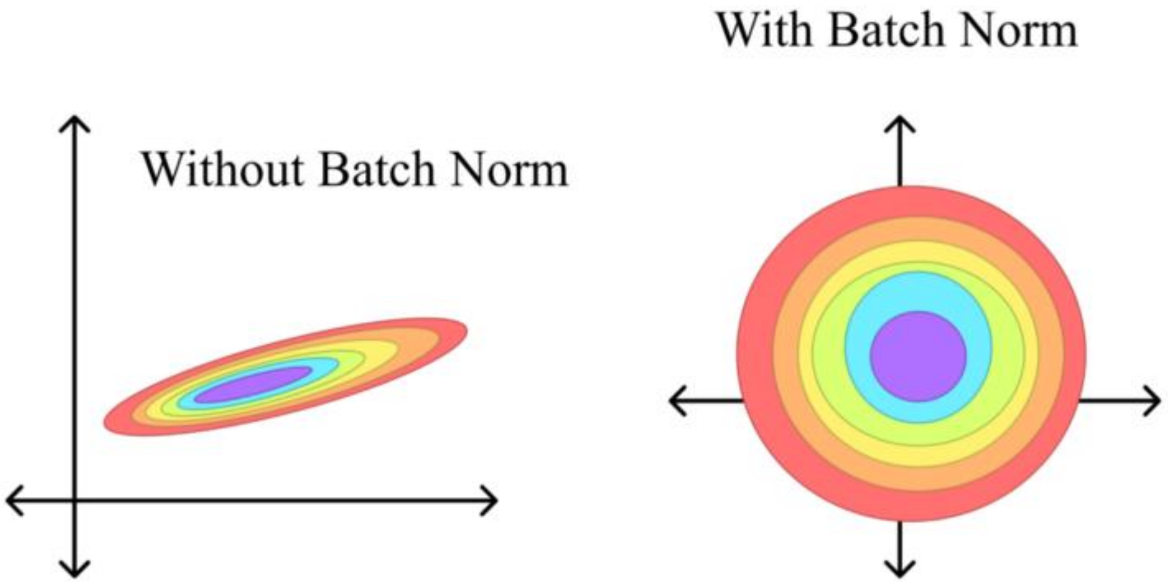
Internal covariate shift

- Internal Covariate Shift - Change of distribution of each layer’ s inputs as the parameters of the previous layer change.
- Then layers need to continuously adapt to the new distribution.



Benefits

- Reduce sensitivity to initialize parameters
  - Larger learning rate
  - Reduce the effect of poor initialization
- Better model performance
  - Landscape become smoother
  - Higher chance to reach global minimum
- Regularize model
  - Parameters are trainable in BN networks



Algorithm

- How does the BN works?

For each epoch, for each batch do:

1) For each layer, k, perform Batch Normalization Transform and modify the input of the next layer.

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$

$N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network

for $k = 1 \ldots K$ **do**

  Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)

  Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead

end for

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

$N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with frozen
// parameters

for $k = 1 \ldots K$ **do**

  // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.

  Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

  In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$

end for

Algorithm 2: Training a Batch-Normalized Network

Algorithm

- How does the BN works?

Input: $\mathcal{B} = \{x_1 = 2,\ x_2 = 3,\ x_3 = 4\}$

Parameters: $\gamma = 1, \quad \beta = 0, \quad \epsilon = 0.001$

$$\mu_{\mathcal{B}} = \frac{2+3+4}{3} = 3$$

$$\sigma_{\mathcal{B}}^2 = \frac{(2-3)^2 + (3-3)^2 + (4-3)^2}{3} = \frac{1+0+1}{3} = 0.6667$$

$$\hat{x}_1 = \frac{2-3}{\sqrt{0.6667 + 0.001}} = \frac{-1}{0.8166} = -1.2247$$

$$\hat{x}_2 = \frac{3-3}{0.8166} = 0.0$$

$$\hat{x}_3 = \frac{4-3}{0.8166} = 1.2247$$

$$y_1 = 1.0 \times (-1.2247) + 0 = -1.2247$$

$$y_2 = 1.0 \times 0 + 0 = 0$$

$$y_3 = 1.0 \times 1.2247 + 0 = 1.2247$$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

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

$$\mathcal{B} = \{x_1 = 2,\ x_2 = 3,\ x_3 = 4\}$$

$$\downarrow$$

$$\boxed{\{y_1 = -1.2247,\ y_2 = 0,\ y_3 = 1.2247\}}$$

Algorithm

- How does the BN works?

For each epoch, for each batch do:

1) For each layer, k, perform Batch Normalization Transform and modify the of the next layer.
2) Train the new network to optimize $\gamma, \quad \beta, \quad \theta$
3) For the inference, in each k do:

$$\mathrm{E}[x] = 3.0, \quad \mathrm{Var}[x] = 0.6667$$

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

$$y = \frac{1}{0.8166} \cdot x + \left(0. - \frac{3}{0.8166}\right) = 1.2247 \cdot x - 3.674$$

Input: Network N with trainable parameters Θ; subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$

$N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network

for $k = 1 \ldots K$ **do**

Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)

Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead

end for

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

$N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with frozen // parameters

for $k = 1 \ldots K$ **do**

// For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.

Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

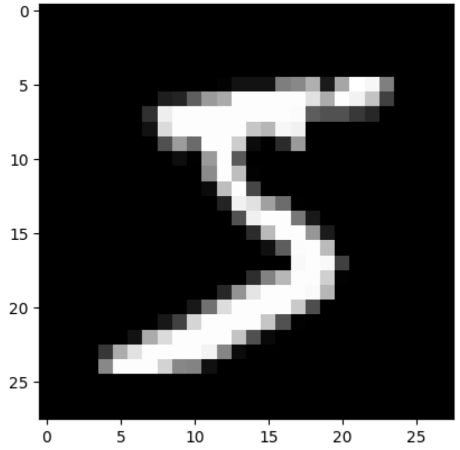
In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$

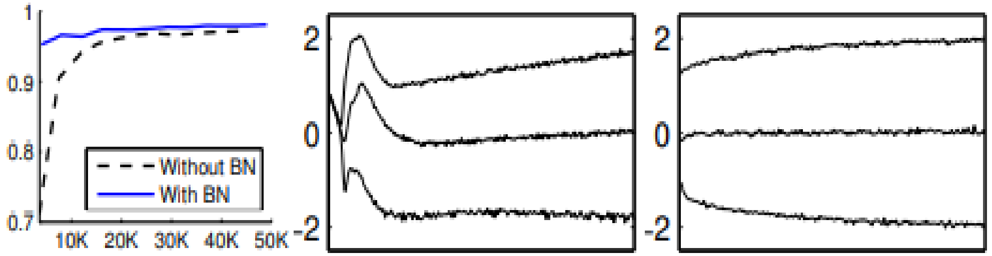
end for

Algorithm 2: Training a Batch-Normalized Network

Experiments - Activations over time

- Dataset: MNIST
- Input: 28x28 binary image
- Model: 3 fully-connected hidden layers with 100 activations each
- Results:
    - Figure (a) shows that BN network got higher accuracy and in less steps.
    - Figures (b) and (c) show how the distribution of a hidden layer evolves, with BN model being more stable over time.

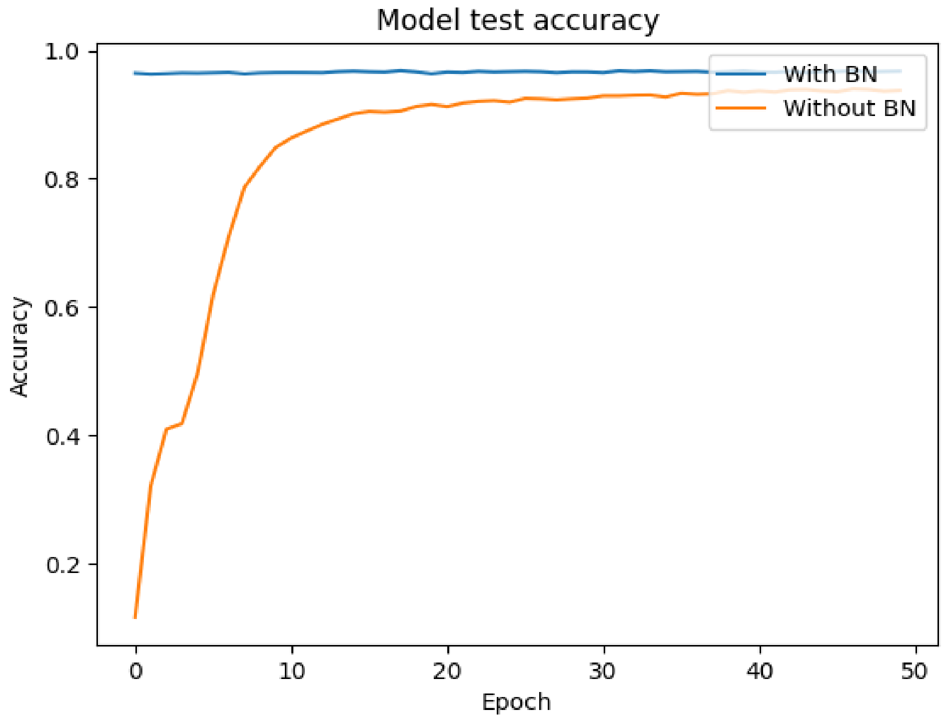




(a) (b) Without BN (c) With BN

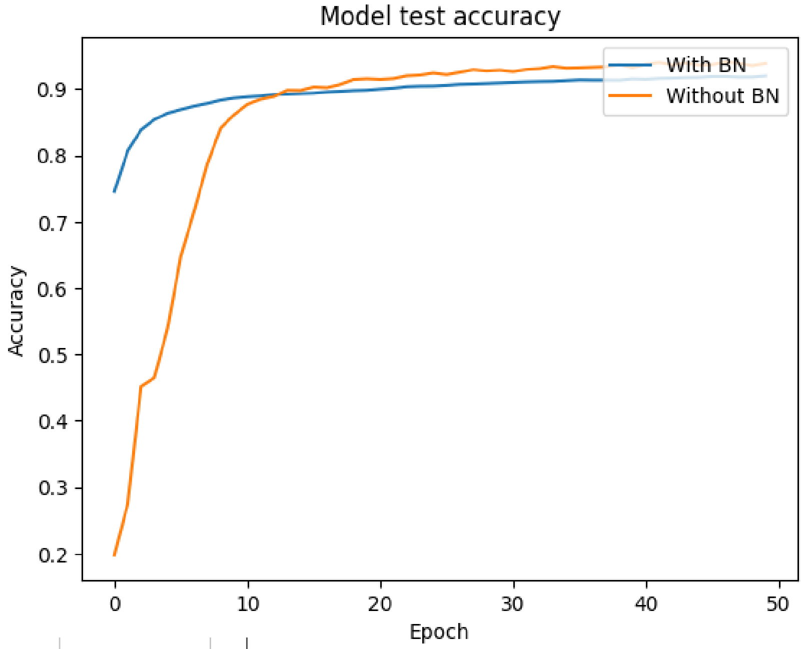
Experiments - Activations over time

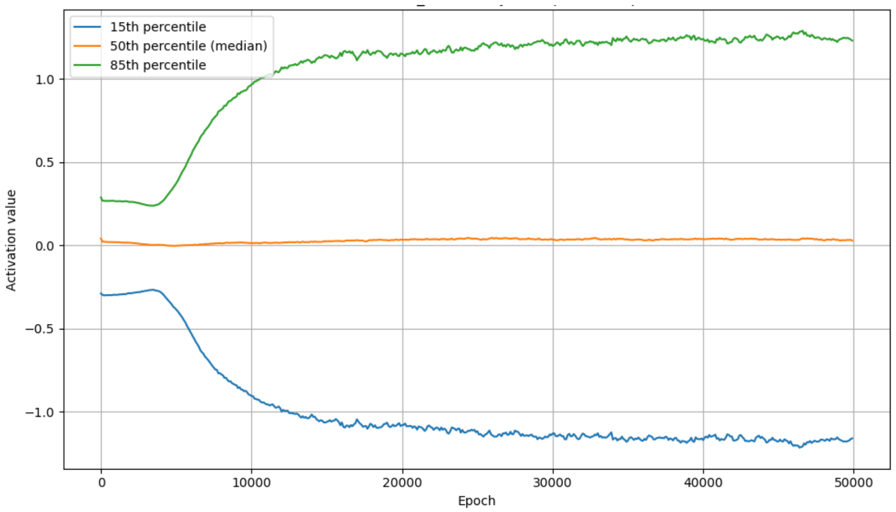
```
model_BN = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(100, kernel_initializer='random_normal'),
    keras.layers.BatchNormalization(),
    keras.layers.Activation('sigmoid'),
    keras.layers.Dense(100, kernel_initializer='random_normal'),
    keras.layers.BatchNormalization(),
    keras.layers.Activation('sigmoid'),
    keras.layers.Dense(100, kernel_initializer='random_normal'),
    keras.layers.BatchNormalization(),
    keras.layers.Activation('sigmoid'),
    keras.layers.Dense(10, activation='softmax')
])

model_base = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(100, activation = 'sigmoid', kernel_initializer='random_normal'),
    keras.layers.Dense(100, activation = 'sigmoid', kernel_initializer='random_normal'),
    keras.layers.Dense(100, activation = 'sigmoid', kernel_initializer='random_normal'),
    keras.layers.Dense(10, activation='softmax')
])
```

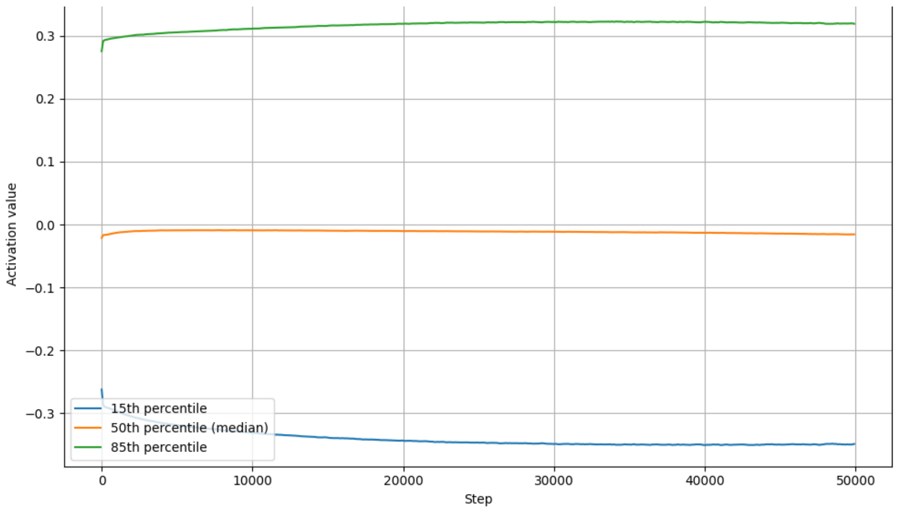


Experiments - Activations over time

```python
class BatchNormLayer(keras.layers.Layer):
    def __init__(self, units=100, batch_size = 60):
        super().__init__()
        self.units = units
        self.batch_size = batch_size

    def build(self, input_shape):
        self.beta = self.add_weight(
            shape=(input_shape[-1],),
            initializer=keras.initializers.RandomNormal(mean=0.0, stddev=0.01),
            trainable=True
        )
        self.gamma = self.add_weight(
            shape=(input_shape[-1],),
            initializer=keras.initializers.RandomNormal(mean=1.0, stddev=0.01),
            trainable=True
        )
```







Experiments - ImageNet classification

- Dataset: ImageNet
- Model: variant of Inception model
- Results:
  - To accelerate BN networks we need also to:
    - Increase learning rate
    - Remove Dropout
    - Reduce L2 weight regularization
  - In Figures 2 and 3, the comparison of variants of networks

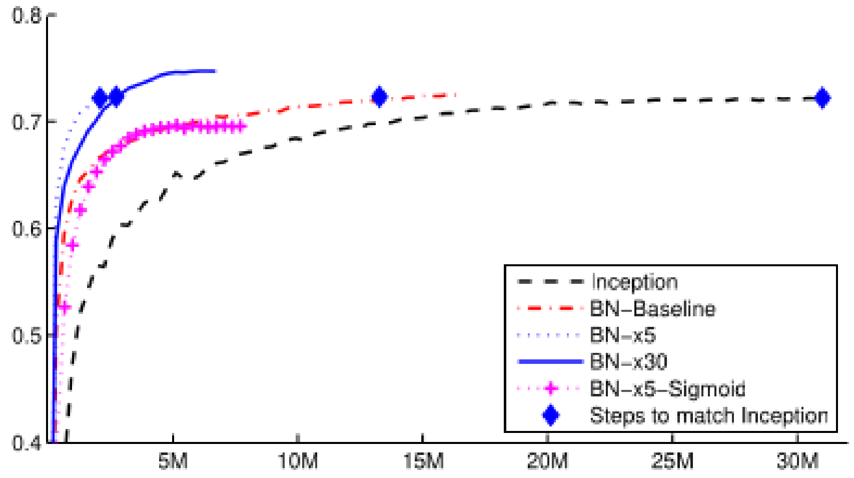


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

Figure 3: *For Inception and the batch-normalized variants, the number of training steps required to reach the maximum accuracy of Inception (72.2%), and the maximum accuracy achieved by the network.*

Experiments - ImageNet classification

- Dataset: ImageNet
- Model: variant of Inception model
- Results:
  - BN-Inception multicrop outperformed all the single models
  - BN-Inception ensemble outperformed even human accuracy, ~5.1%

Model	Resolution	Crops	Models	Top-1 error	Top-5 error
GoogLeNet ensemble	224	144	7	-	6.67%
Deep Image low-res	256	-	1	-	7.96%
Deep Image high-res	512	-	1	24.88	7.42%
Deep Image ensemble	variable	-	-	-	5.98%
BN-Inception single crop	224	1	1	25.2%	7.82%
BN-Inception multicrop	224	144	1	21.99%	5.82%
BN-Inception ensemble	224	144	6	20.1%	**4.9%***

Our thoughts

- The main reason why BN works are conservative
  - Some research shows that BN improves perfromance even with low Internal Covariate Shift
- We believed that BN works because:
  - Smoother landscape
  - Gradient flow improvement
  - Regularization

Improvement

- Dependence on batch size
  - Layer normalization
  - Weight standardization
  - Normalizing Batch Normalization

- Could not directly apply on specific tasks
  - RNN
  - Small Datasets

- Feature bias are involved in some cases

Thanks!