Course Materials for GEN-AI

Northeastern University

These materials have been prepared and sourced for the course **GEN-AI** at Northeastern University. Every effort has been made to provide proper citations and credit for all referenced works.

If you believe any material has been inadequately cited or requires correction, please contact me at:

Instructor: Ramin Mohammadi
r.mohammadi@northeastern.edu

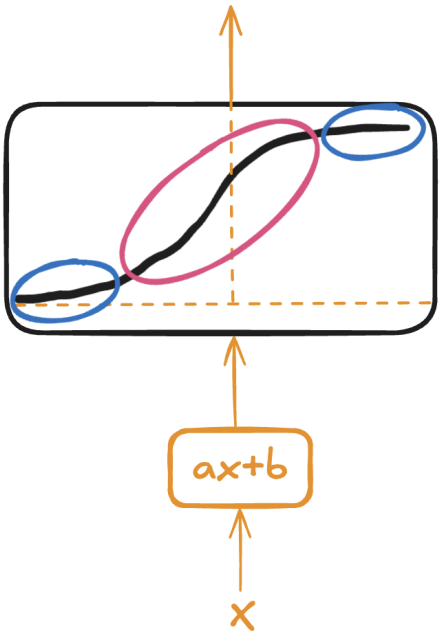
Thank you for your understanding and collaboration.

Content sourced from multiple references - see Reference section for details

Batch Normalization

Introduction

Batch Normalization is a technique used in deep learning to improve the training speed, stability, and performance of neural networks. It works by normalizing the inputs to each layer within a mini-batch, ensuring that the data passing through the network has a consistent distribution. This helps mitigate issues like **internal covariate shift**, where the distribution of inputs to a layer changes during training as the parameters of previous layers change.



If the activation is in a saturated region (blue region), the output dramatically differs from the input. If the activation is in the linear area (pink region) or closer to 0, we will have less variation.

Why Neural Networks prefer to have activation functions linear?

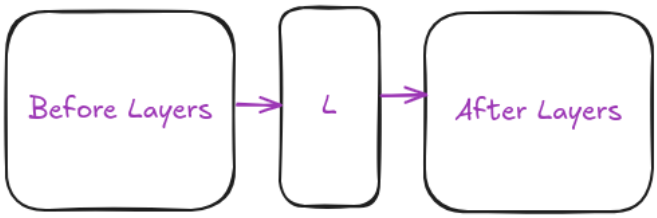
1. **Gradient Descent & Backpropogation:** If the activation map is in the linear region, then the gradient is neither too large nor too small. This helps in avoiding "vanishing" or "exploding" gradients problem.
2. **Effective learning**: A small change in input leads to a proportional change in output.

3. **Avoiding Saturation:** Sigmoid and tanh have saturation regions where the gradients are close to 0 causing a slow down in the learning.
4. **Non linearity requirement**: While activation functions requires to have non linearity for learning reasons, staying in linear regions ensures that Nueral networks retains some level of linearity. Most activation functions have a linear region to balance non-linearity. ReLU has linear region to keep positive values and also sets negative to 0.

Internal Covariate Shift

Internal covariate shift refers to the phenomenon where the distribution of inputs to each layer changes as the network parameters change during training. This slows down learning because the network constantly needs to adapt to the shifting input distributions.

$$L = f(f(f(u, \theta_1), \theta_2), ...)$$



Change in the input will lead to a change in internal activation distribution that requires "Domain Adaptation".

Domain Adaptation

It is a sub-field of transfer learning that focuses on ability of model to generalize better when train and inference data have different distribution. The goal is to adapt model to perform well on test data and minimize the need for labelled data. It's quite expensive approach, so instead:

- **Batch Normalization**:
  - To normalize the activation maps:

$$a^L \rightarrow \frac{a^L - E[a^L]}{\sqrt{\text{var}(a^L)}} \quad \Rightarrow \text{This will normalize to new data with } \mu = 0 \text{ and } \sigma = 1.$$

  - Scale and shift:

$$a^L \leftarrow \gamma a^L + \beta \quad \Leftarrow \text{Here, we introduce a new mean and std by shifting the data.}$$

  - γ and β are parameters that are learned to adjust the mean and standard deviation post-normalization.

- **MiniBatch**:

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

$$\delta_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$$

- **Normalization**:

$$\hat{x}_i \leftarrow \frac{x_i - \mu_\beta}{\sqrt{\delta_\beta^2 + \epsilon}} \quad \text{(data has mean of 0 and std of 1)}$$

- **Scale & Shift**:

$$y_i \leftarrow \gamma \hat{x_i} + \beta$$

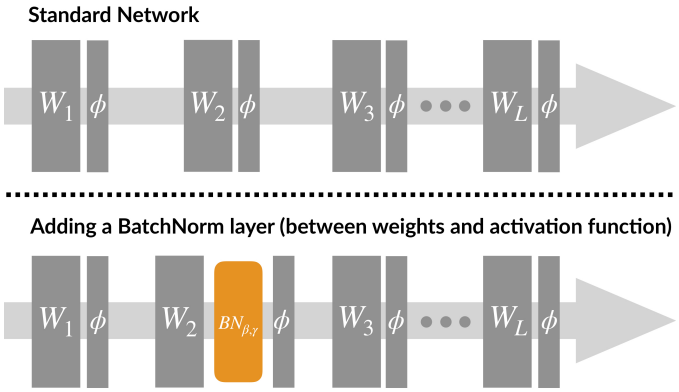
(I will add β and multiply γ, the idea here
is to ensure every layer has fixed output distribution(Same β , γ))

- **During test**:

$$\hat{x} \leftarrow \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}}$$

  - They claimed it converges faster.
  - It can have larger learning rates.

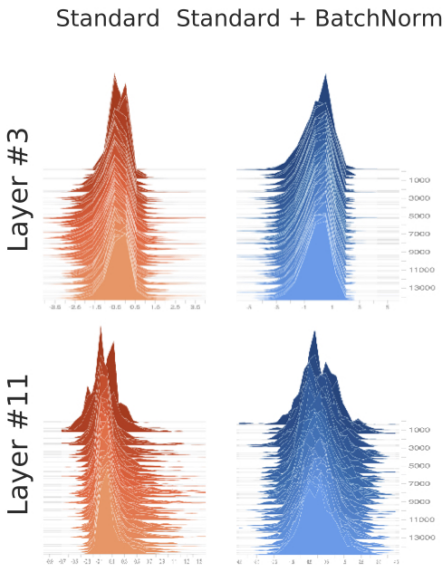
Batch Normalization - New Paper



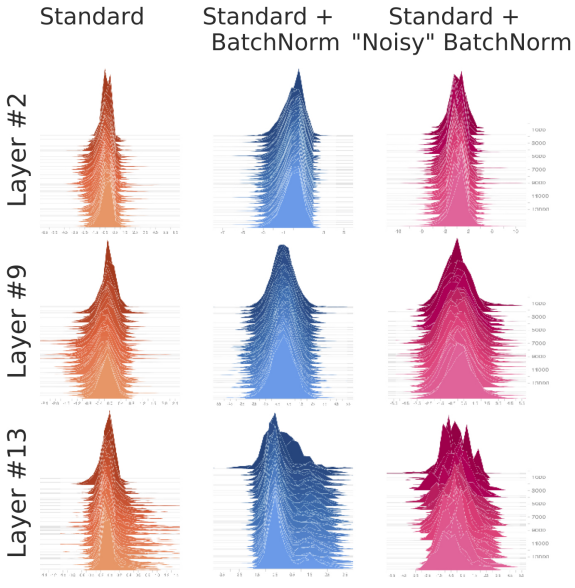
- It says that the first paper intuition is incorrect, they say it doesn't fix the covariate shift problem.
- They applied noise and found it doesn't harm the learning.

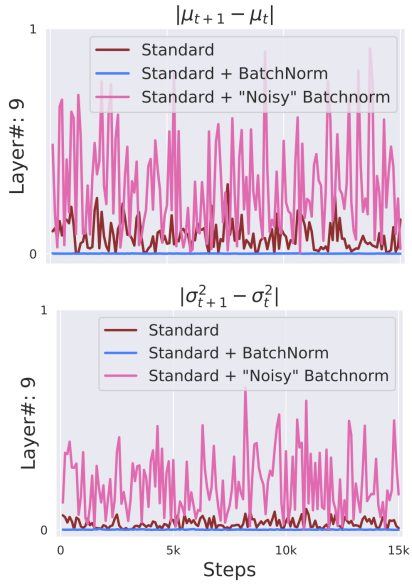
New claims

- Batch normalization doesn't fix covariate shift.



- even if we fix covariate shift externally, it doesn't help.
- If we intentionally increase "internal covariate shift", it doesn't harm.





- Batch normalization is not the only normalization option.

So if the assumption is incorrect, then why Batch Normalization does work?

The Paper introduces two important points

- **L-Lipschitz:** A function f is said to be L**-Lipschitz** if there exists a constant $L > 0$ such that:

$$|f(x_1) - f(x_2)| \leq L\|x_1 - x_2\|_2 \quad \forall x_1, x_2 \in \mathbb{R}^n.$$

  This can also be rewritten as:

$$\Rightarrow \boxed{\frac{|f(x_1) - f(x_2)|}{\|x_1 - x_2\|_2} \leq L}.$$

  Here:

  - L is the **Lipschitz constant**, which bounds the rate of change of f with respect to its input.
  - If L is small, f is less sensitive to changes in x (smoother behavior).
  - If L is large, f can change more abruptly (less smooth behavior).

- β**-Smooth Functions:** A function f is said to be β**-smooth** if its gradient ∇f is Lipschitz continuous with a Lipschitz constant $\beta > 0$. This means:

$$\|\nabla f(x_1) - \nabla f(x_2)\|_2 \leq \beta\|x_1 - x_2\|_2 \quad \forall x_1, x_2 \in \mathbb{R}^n.$$

  Intuitively:

  - β bounds how quickly the gradient of f can change.
  - Smaller β means smoother gradients and better optimization stability.

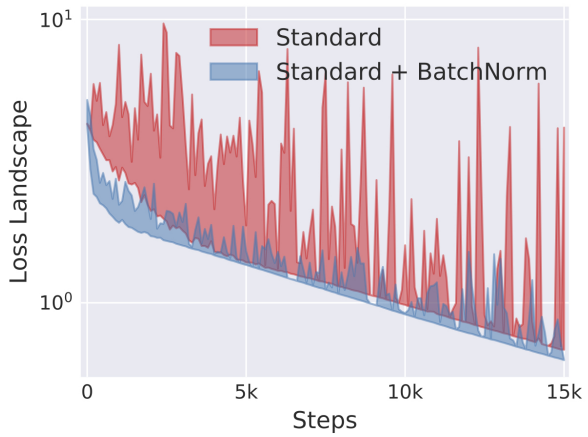
- **Implications of L-Lipschitz:** If f is L-Lipschitz, the magnitude of changes in $f(x)$ is proportionally limited by the magnitude of changes in x. This property also extends to the derivative or gradient, which ensures that the loss function behaves in a controlled way. For instance, the Hinge Loss is an L-Lipschitz function.

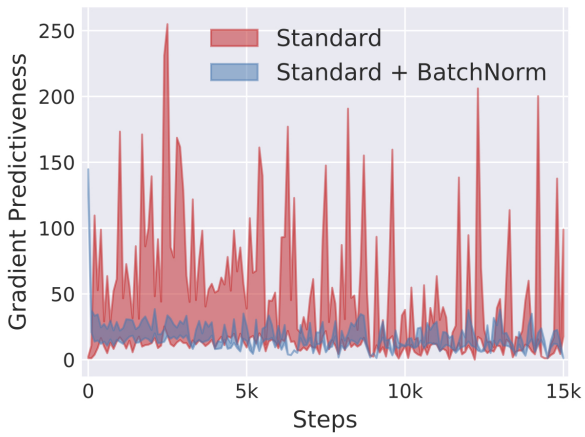
- **Batch Normalization and Lipschitzness:** The claim of the referenced paper is that Batch Normalization works because it improves the Lipschitzness of the loss function. This means:

  - The loss changes at a smaller rate with respect to the inputs.
  - The magnitudes of the gradients are smaller, leading to more stable optimization.

  This improvement can be summarized as:

$$\Rightarrow \boxed{\frac{|f(x_1) - f(x_2)|}{\|x_1 - x_2\|_2} \leq L \quad \text{(where } L \text{ is small and well-behaved)}}.$$





Refresher: Gradient Descent

In gradient descent, we use the local linear approximation of the loss function to find the most effective update direction.
Local Linear Approximation:

- **Compute the gradient at the current point:** This represents the direction of steepest increase in the loss function.
$$\text{Gradient at } \theta_t \approx \nabla L(\theta_t)$$
- **Linear approximation of the loss function:** The loss function near θ_t can be approximated as a linear function of $\Delta\theta$:
$$L(\theta_t + \Delta\theta) \approx L(\theta_t) + \nabla L(\theta_t)^T \Delta\theta$$
- **Update step:** To minimize the loss, the parameter update should be in the direction opposite to the gradient:
$$\Delta\theta = -\alpha \nabla L(\theta_t)$$
where α is the learning rate.

Update equation:
$$\theta^{(t+1)} = \theta^t - \alpha \nabla L(\theta_t)$$

Batch Normalization Effect

It helps with smoothness, $\beta - smooth$ will bound the second order gradient.

- variation of the loss value for input x:
$$L(x + \eta \nabla L(x)) \quad \eta \in [0.05, 0.4]$$

if the optimization works correctly, the expectation is that the loss at new point is less than L(x). (going in a right direction)

- Gradient predictiveness:
$$||\nabla L(x) - \nabla L(x + \eta \nabla L(x))||, \quad \eta \in [0.05, 0.4]$$
  - **small:** gradient is still valid, we can use larger learning rate. This means you can use the same gradient for other epochs due to smoothness hence you could take a larger step.
  - **large:** gradient is no longer valid, there has been a huge change.

Alternatives to Batch Normalization

- **Lp Batch Normalization**: Uses the Lp norm instead of the L2 norm for normalization, making it more robust to outliers.
- **Layer Normalization**: Normalizes the activations within a layer, rather than across a batch.
- **Subset of the batch**: Normalizes a subset of the batch instead of the entire batch.

- **Weight normalization instead of activations**: Rescales the weights to have a unit norm, which can help with regularization and convergence.
- **ELU & SELU - Non-Linearities with decaying slopes:** Addresses the vanishing gradient problem that can occur in deep neural networks. Uses activation functions with decaying slopes, such as ELU (Exponential Linear Unit) and SELU (Scaled Exponential Linear Unit), to help the network learn more effectively.

References

[1] Accelerating Deep Network Training by Reducing Internal Covariate Shift.

[2] https://gradientscience.org/batchnorm/