Training Neural Networks
Batch Normalization

M. Soleymani
Sharif University of Technology
Spring 2024

Most slides have been adapted from Bhiksha Raj, 11-785, CMU
and some from Fei Fei Li et. al, cs231n, Stanford

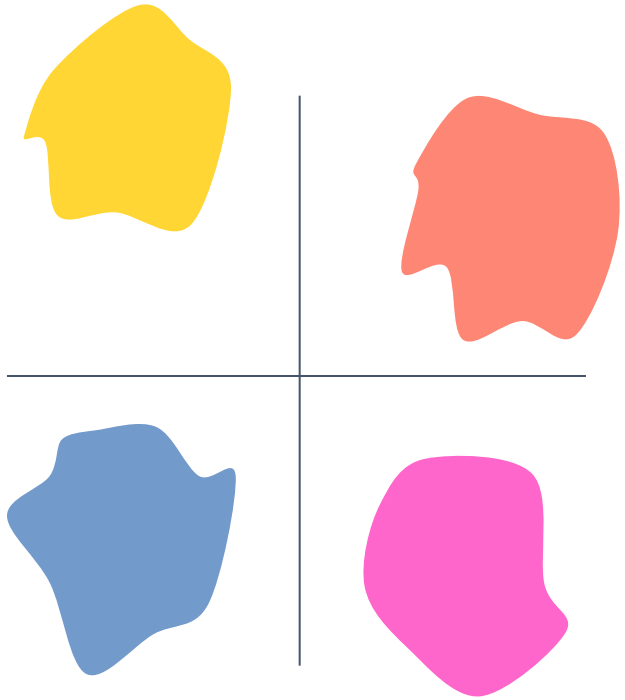
We will see an important technique that helps to speed up training while also acts as a form of regularization

Covariate shift

- Shifting input distributions (covariate shift)

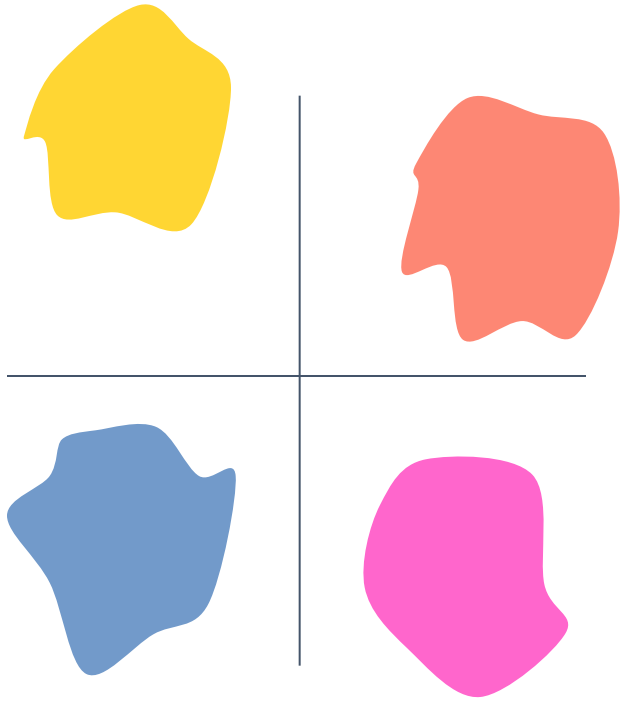
- change in the distributions of internal nodes of a deep network (internal covariate shift)
  - Bach-norm reduces internal covariate shift

The problem of covariate shifts



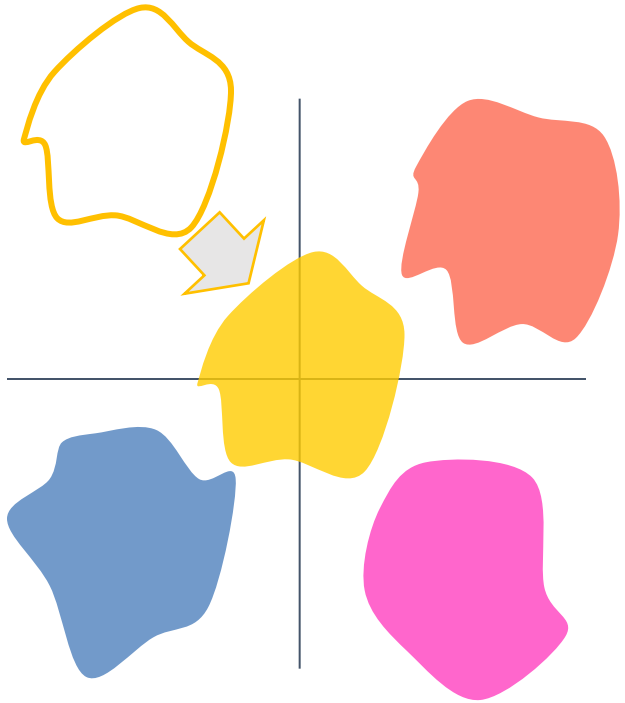
- Training assumes the training data are all similarly distributed
  - Mini-batches have similar distribution?
- In practice, each mini-batch may have a different distribution
  - A “covariate shift”
- Covariate shifts can be large!
  - All covariate shifts can affect training badly

Solution: Move all subgroups to a “standard” location



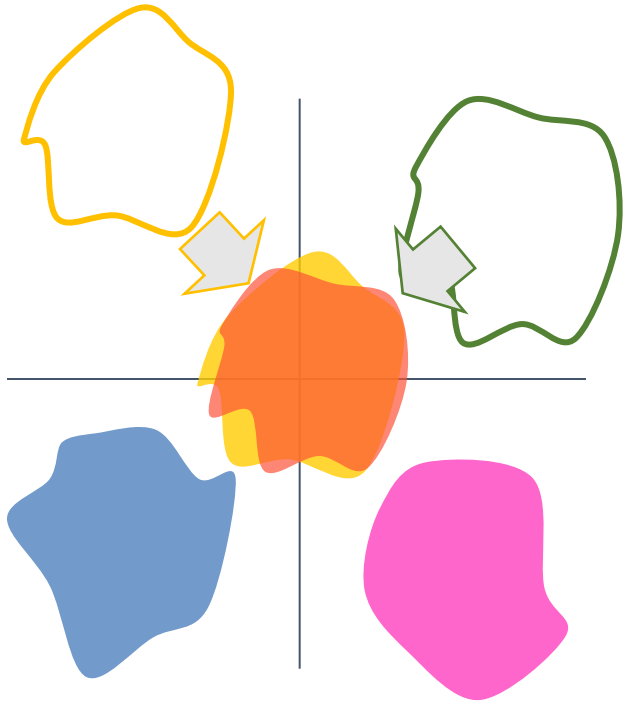
- “Move” all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches

Solution: Move all subgroups to a “standard” location



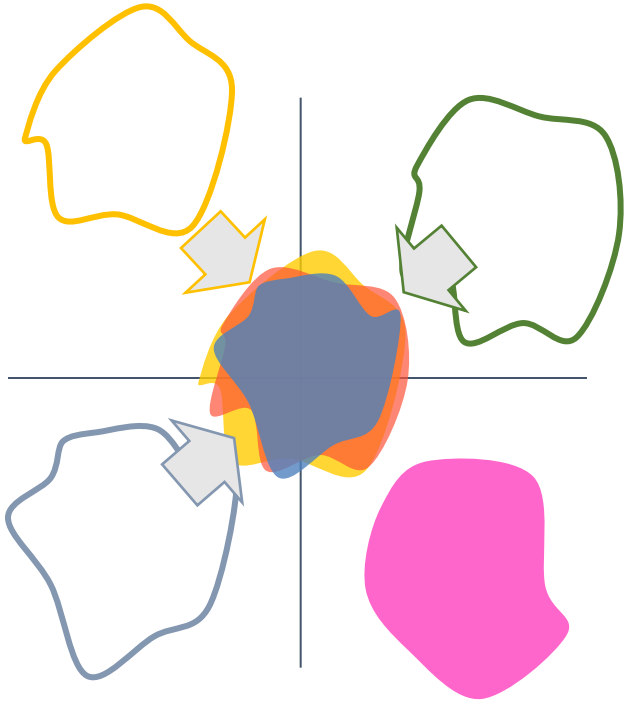
- “Move” all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches

Solution: Move all subgroups to a “standard” location



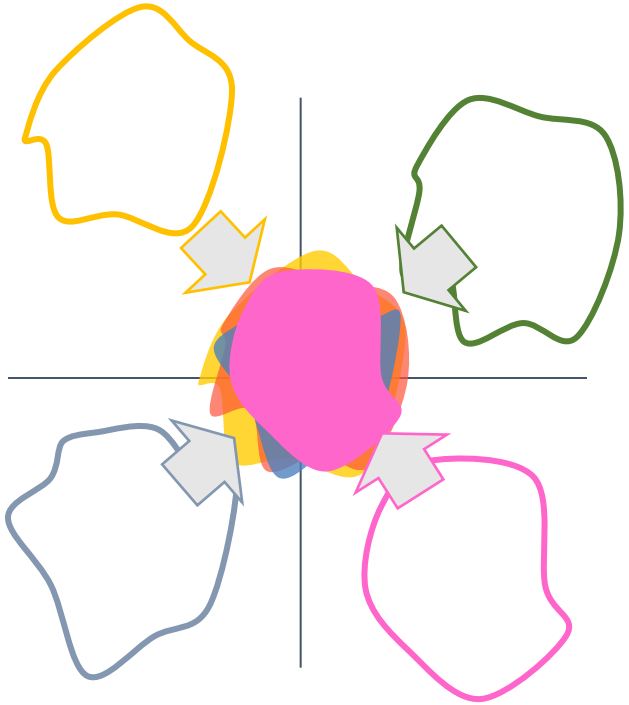
- “Move” all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches

Solution: Move all subgroups to a “standard” location



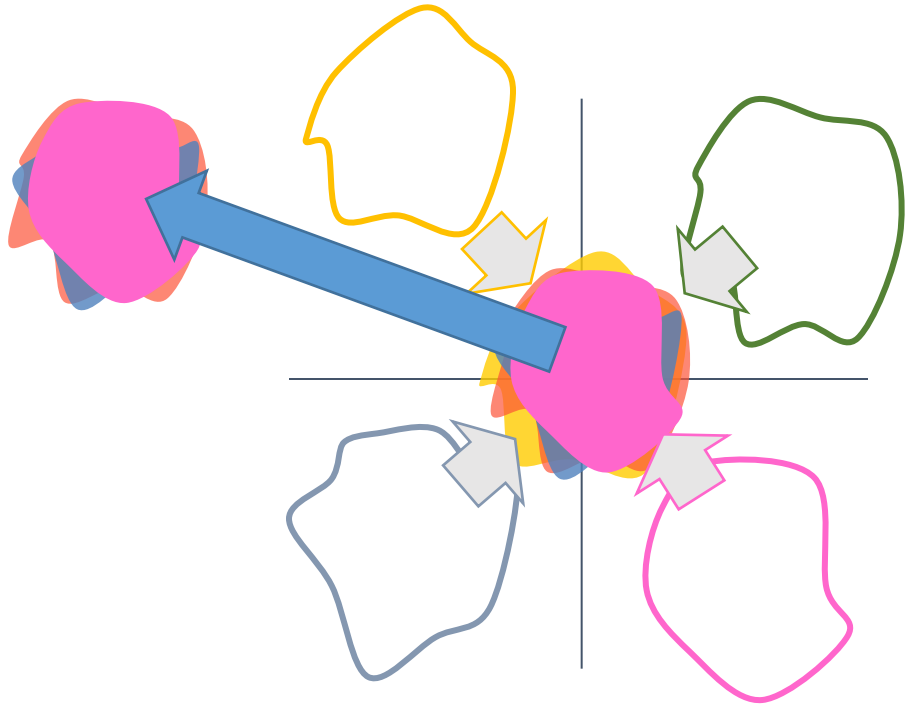
- “Move” all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches

Solution: Move all subgroups to a “standard” location



- “Move” all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches

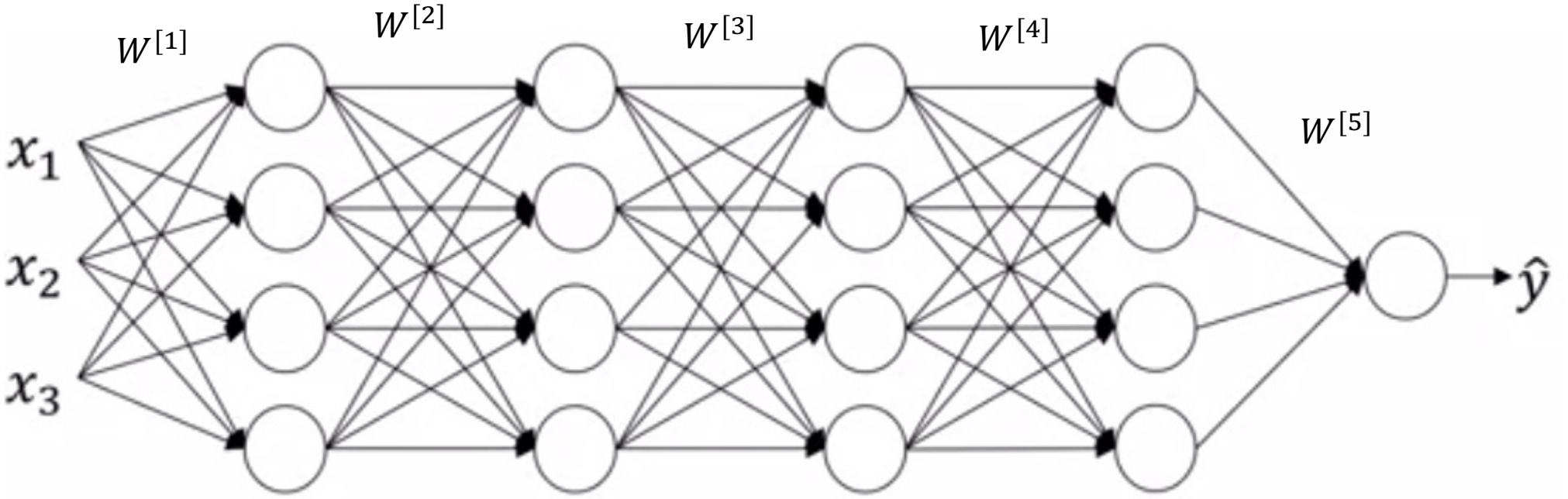
Solution: Move all subgroups to a “standard” location



- “Move” all batches to have a mean of 0 and unit standard deviation
  - Eliminates covariate shift between batches
  - Then move the entire collection to the appropriate location and give it an appropriate variance

Why this is a problem with neural networks?

Limits the amount to which updating weights in earlier layers can affect the distribution of values that the subsequent layers see

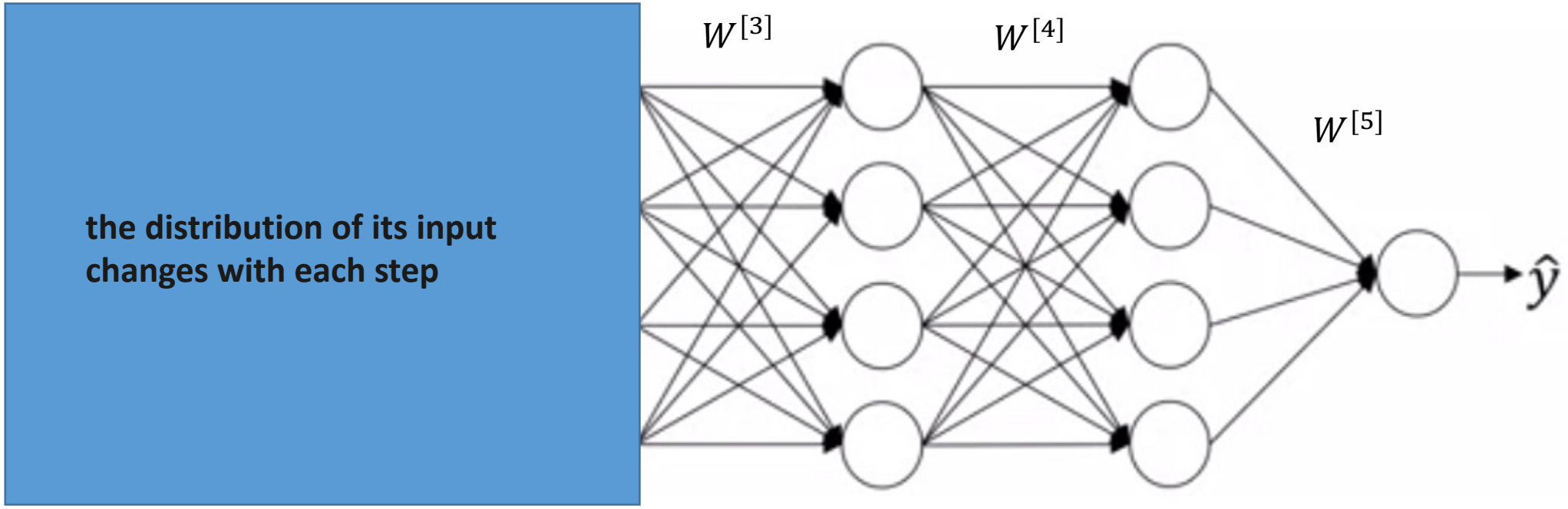


The values become more stable to stand later layers on

[Andrew Ng, Deep Learning Specialization]

Why this is a problem with neural networks?

Limits the amount to which updating weights in earlier layers can affect the distribution of values that the subsequent layers see



The values become more stable to stand later layers on

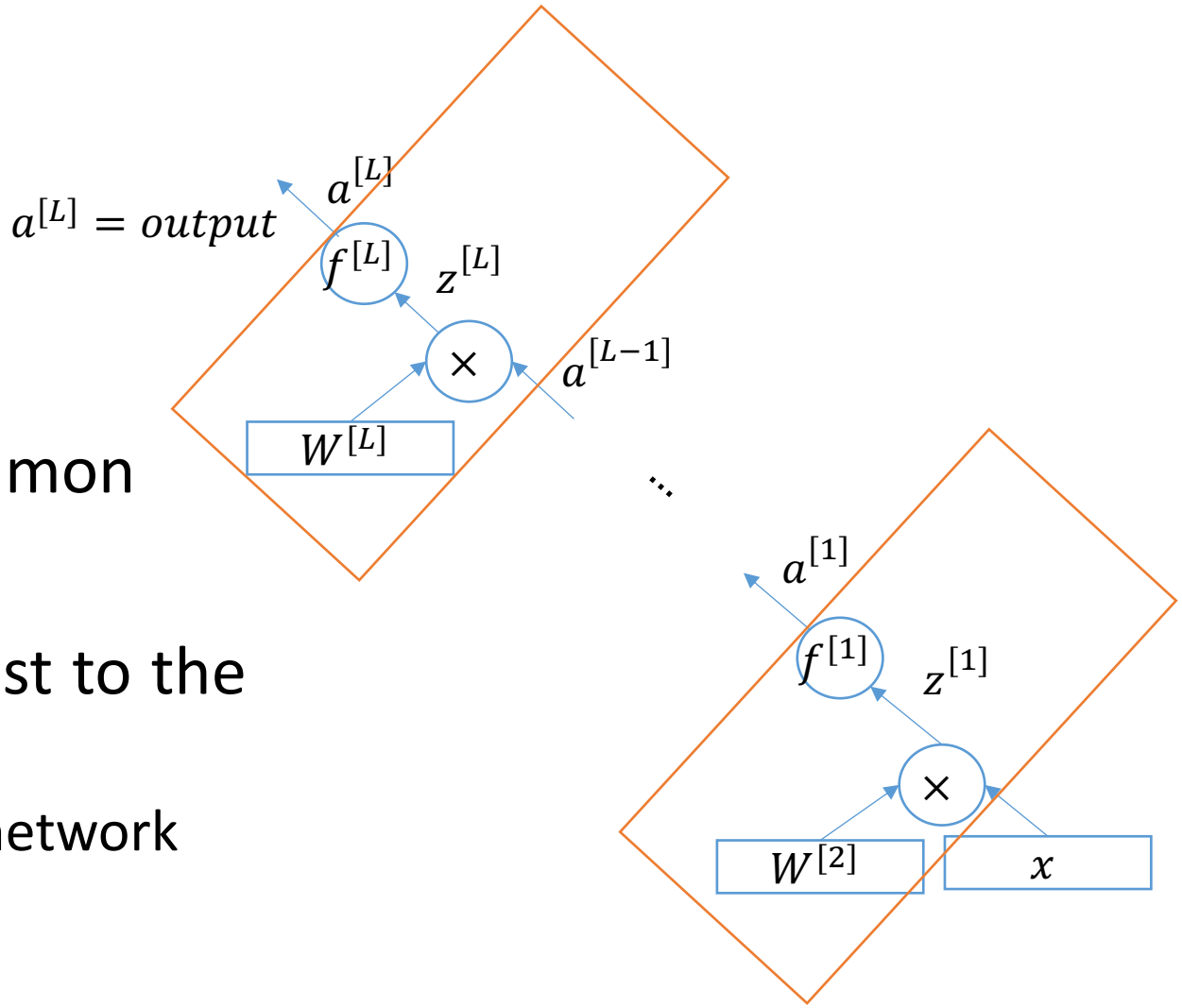
[Andrew Ng, Deep Learning Specialization, 2017]

Normalizing inputs

- On the training set compute mean of each input (feature)
  - $\mu_i = \frac{\sum_{n=1}^{N} x_i^{(n)}}{N}$
  - $\sigma_i^2 = \frac{\sum_{n=1}^{N} \left(x_i^{(n)} - \mu_i\right)^2}{N}$
- **Remove mean**: from each of the input mean of the corresponding input
  - $x_i \leftarrow x_i - \mu_i$
- **Normalize variance**:
  - $x_i \leftarrow \frac{x_i}{\sigma_i}$
- The network training converges faster if its inputs are normalized to have zero means and unit variances

Batch normalization

- Can we normalize $a^{[l]}$ too?
- Normalizing $z^{[l]}$ is much more common
- Makes neural networks more robust to the range of hyperparameters
  - Much more easily train a very deep network



Batch normalization

Training: **The adjustment occurs over individual mini-batches**

[Ioffe and Szegedy, 2015]

$$\mu^{[l]} = \sum_i \frac{z^{(i)[l]}}{B}$$

$$\sigma^{2[l]} = \sum_i \frac{\left(z^{(i)[l]} - \mu^{[l]}\right)^2}{B}$$

$$z_{norm}^{(i)[l]} = \frac{z^{(i)[l]} - \mu^{[l]}}{\sqrt{\sigma^{2[l]} + \epsilon}}$$

B: size of batch

- Problem: do we necessarily want a zero-centered input (with unit standard deviation) to an activation layer?

$$\hat{z}^{(n)[l]} = \gamma^{[l]} z_{norm}^{(i)[l]} + \beta^{[l]}$$

$\gamma^{[l]}$ and $\beta^{[l]}$ are learnable parameters of the model

Batch normalization

[Ioffe and Szegedy, 2015]

$$z_{norm}^{(i)[l]} = \frac{z^{(i)[l]} - \mu^{[l]}}{\sqrt{\sigma^{2[l]} + \epsilon}}$$

In the linear area of tanh

- Allow the network to squash the range if it wants to:

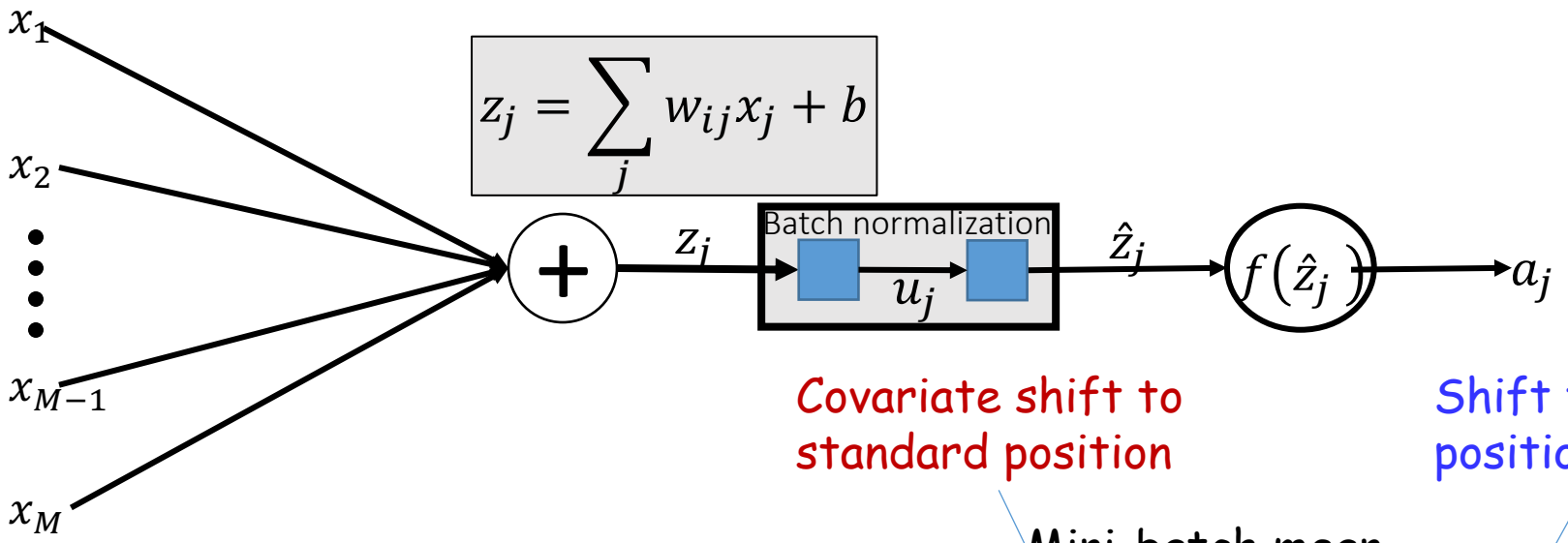
$$\hat{z}^{(i)[l]} = \gamma^{[l]} z_{norm}^{(i)[l]} + \beta^{[l]}$$

The network can learn to recover the identity:

$$\gamma^{[l]} = \sqrt{\sigma^{2[l]} + \epsilon}$$
$$\beta^{[l]} = \mu^{[l]}$$

$$\Rightarrow \hat{z}^{(i)[l]} = z^{(i)[l]}$$

Batch normalization



Minibatch size

Covariate shift to standard position

Shift to right position

Mini-batch mean

$$\mu_j = \frac{1}{B} \sum_{n \in mb} z_j^{(n)}$$

$$\sigma_j^2 = \frac{1}{B} \sum_{n \in mb} \left(z_j^{(n)} - \mu_j \right)^2$$

$$u_j^{(n)} = \frac{z_j^{(n)} - \mu_j}{\sigma_j}$$

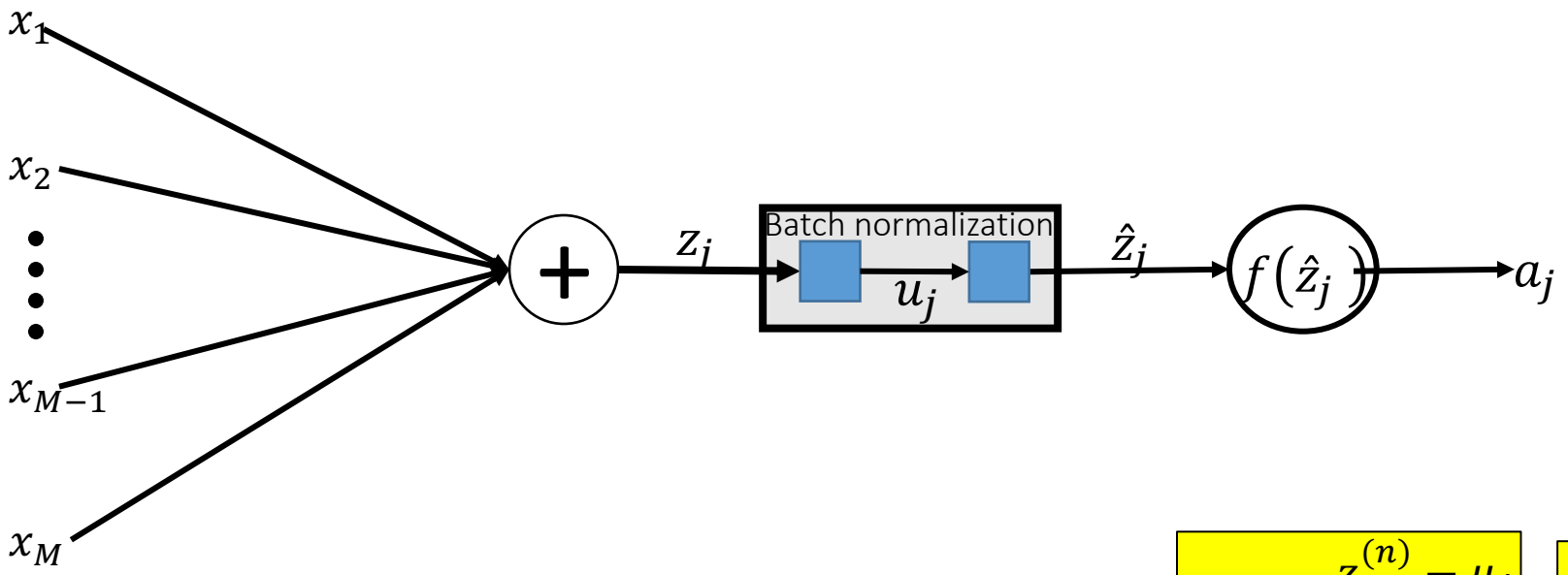
$$\hat{z}_j^{(n)} = \gamma_j u_j^{(n)} + \beta_j$$

Mini-batch standard deviation

Neuron-specific terms for each batch

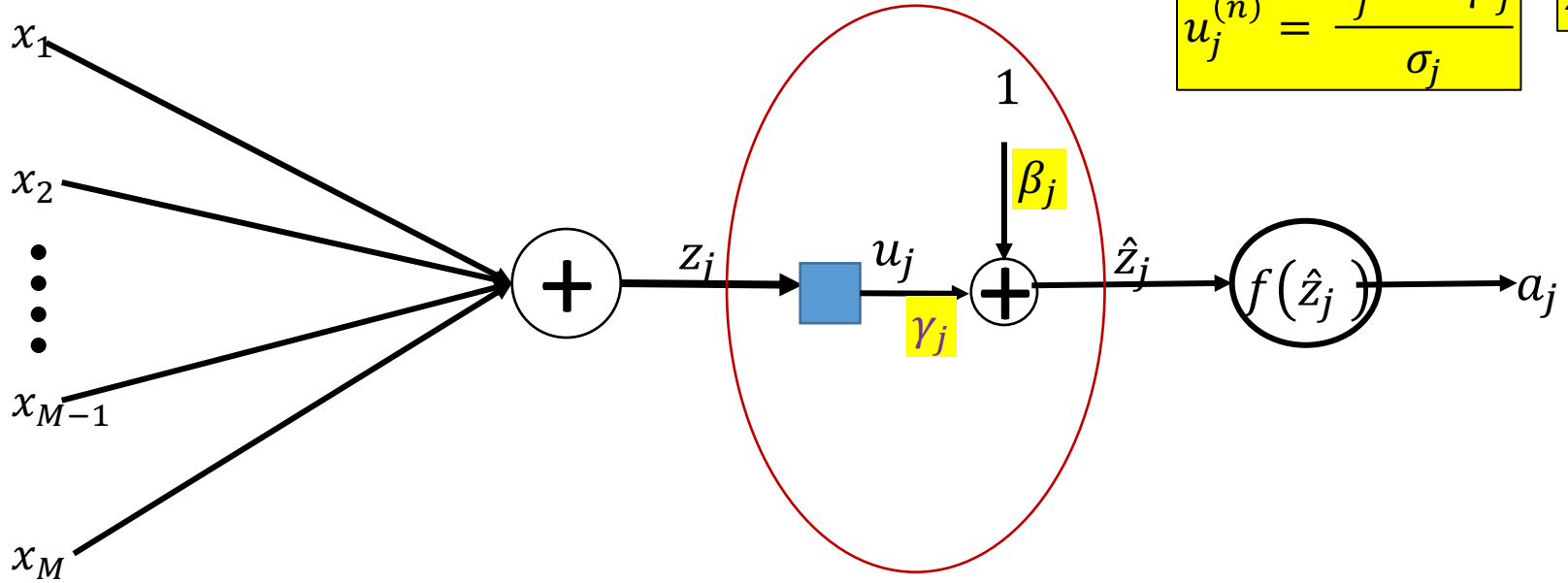
- BN aggregates the statistics over a minibatch and normalizes the batch by them
- Normalized instances are "shifted" to a *unit-specific* location

Batch normalization

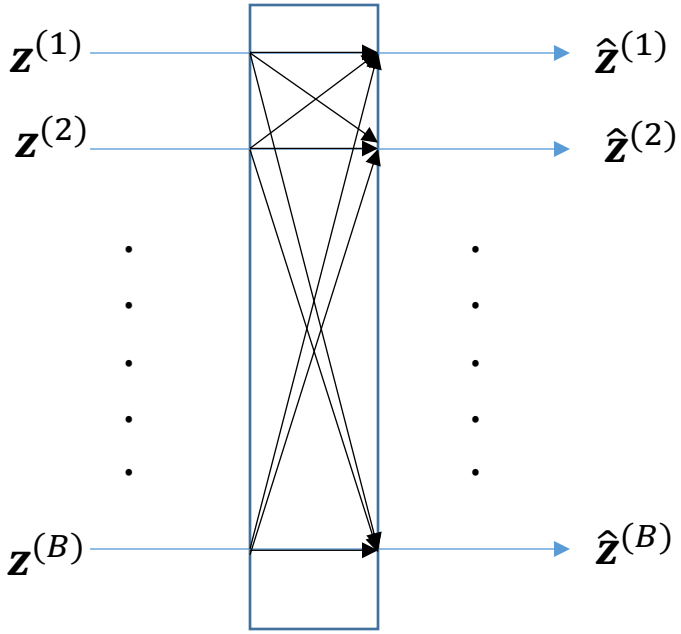


$$u_j^{(n)} = \frac{z_j^{(n)} - \mu_j}{\sigma_j}$$

$$\hat{z}_j^{(n)} = \gamma_j u_j^{(n)} + \beta_j$$

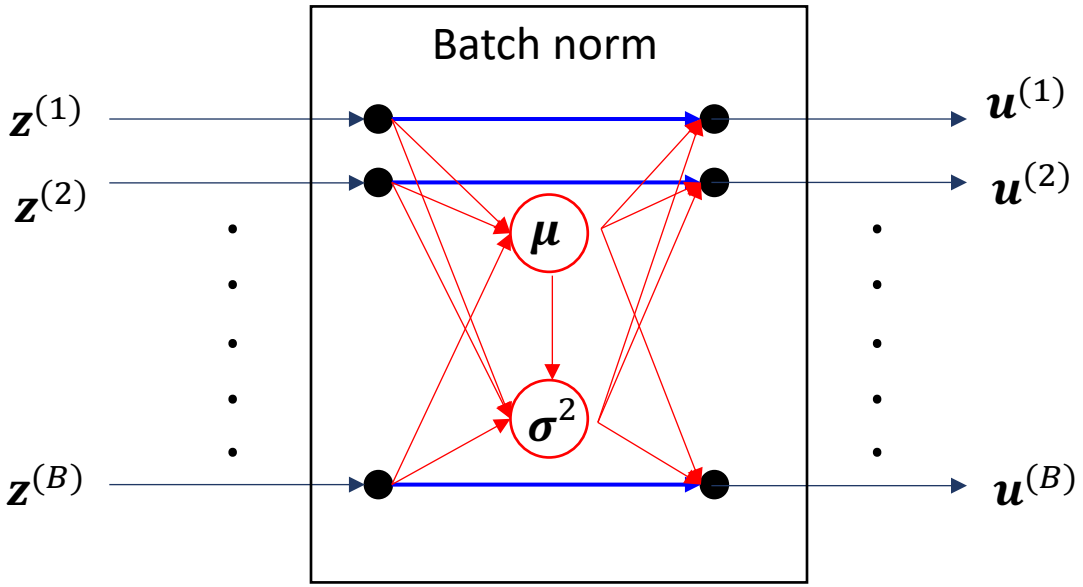


Batchnorm is a vector function over the minibatch



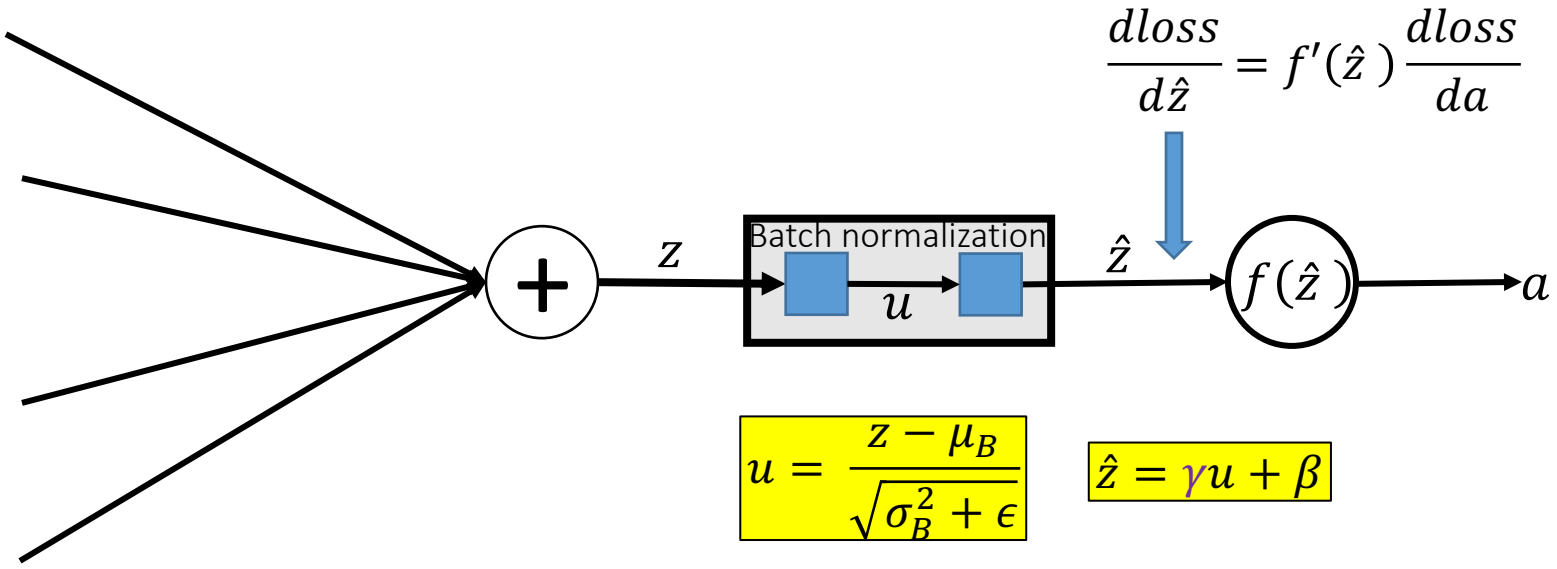
- Batch normalization is really a *vector* function applied over all the inputs from a minibatch
  - To compute the derivative of the loss w.r.t any $z^{(i)}$, we must consider all $\hat{z}$s in the batch

Batchnorm

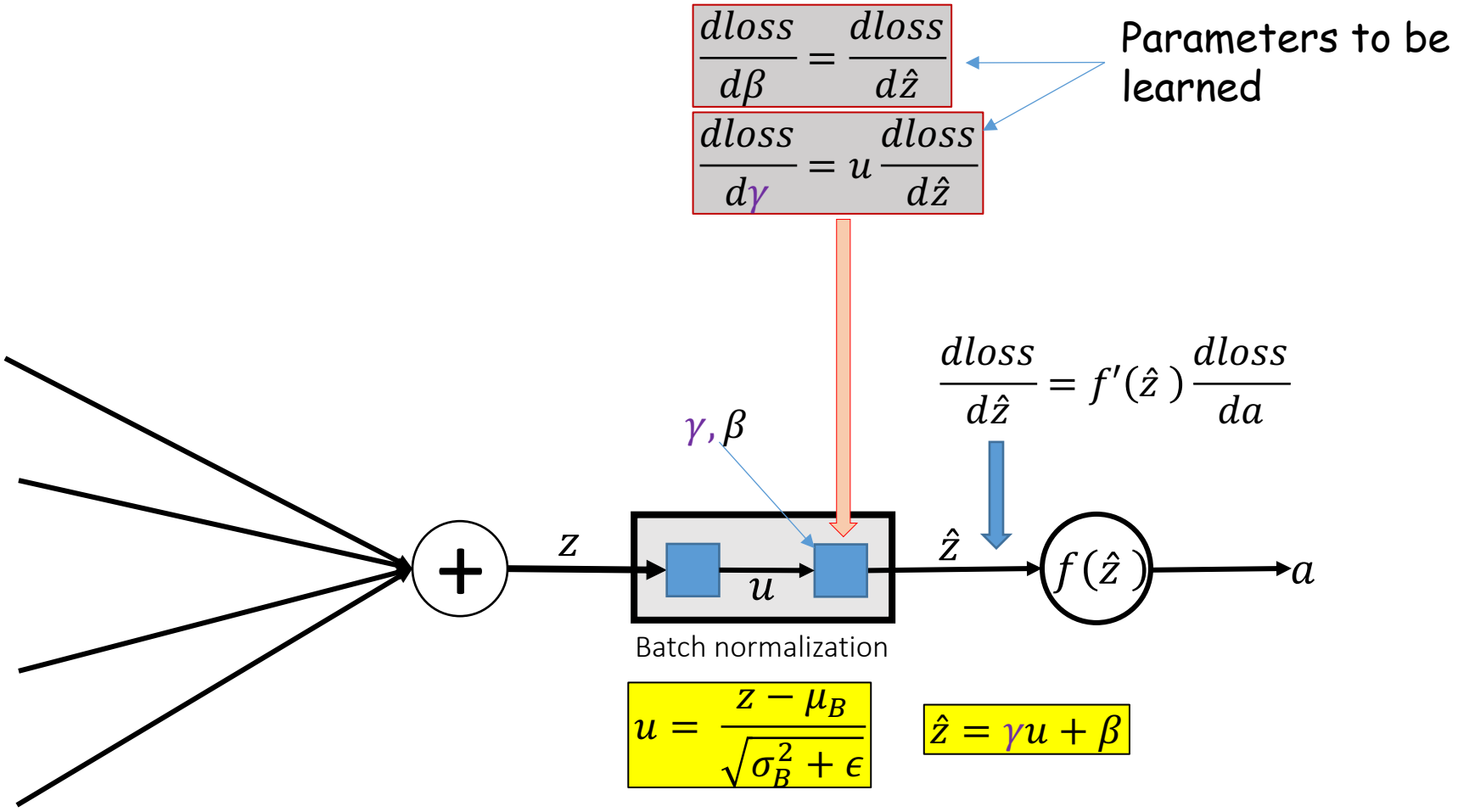


- The complete dependency figure for Batch-norm
- You can use vector function differentiation rules to compute the derivatives

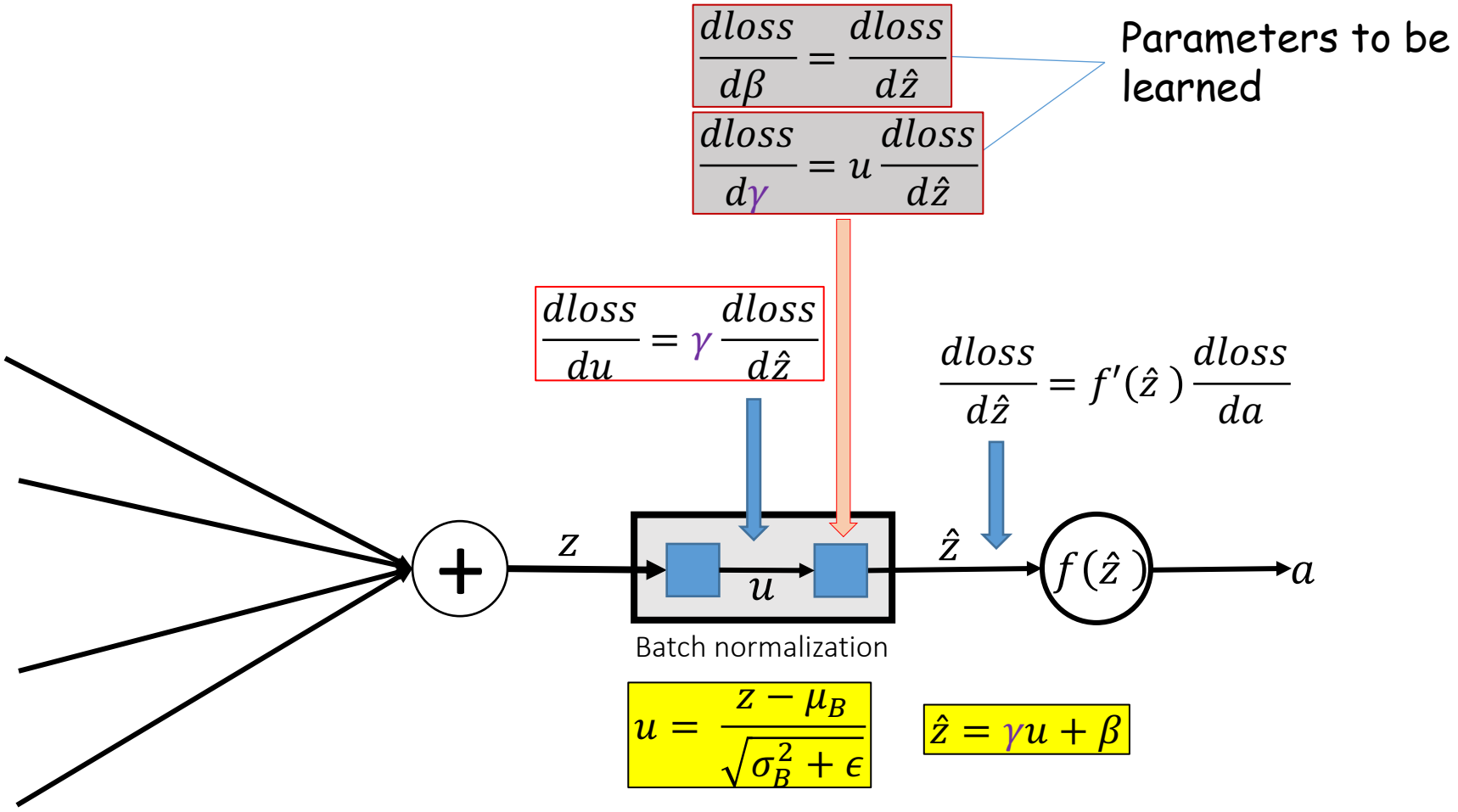
Batch normalization: Backpropagation



Batch normalization: Backpropagation

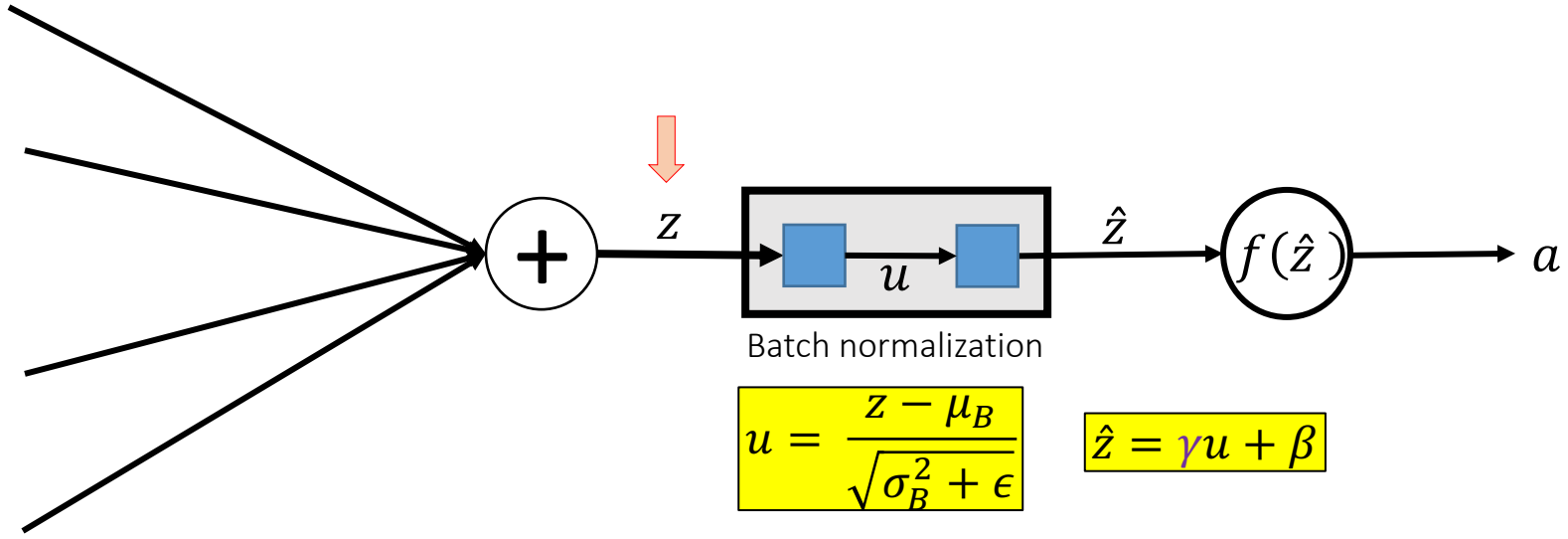


Batch normalization: Backpropagation

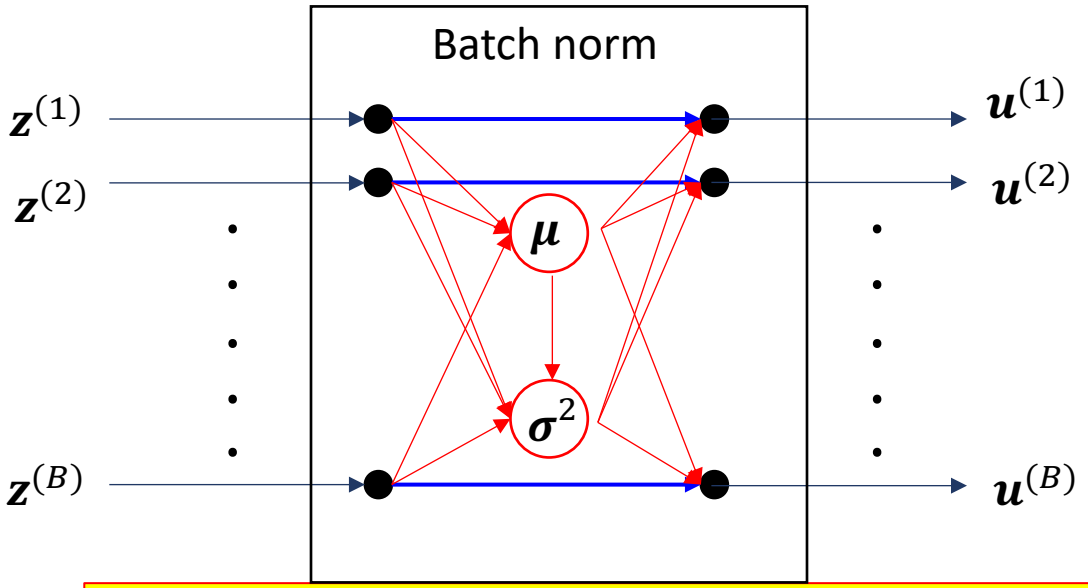


Batch normalization: Backpropagation

- Final step of backprop: compute $\frac{\partial loss}{\partial z}$



Batch normalization: Backpropagation



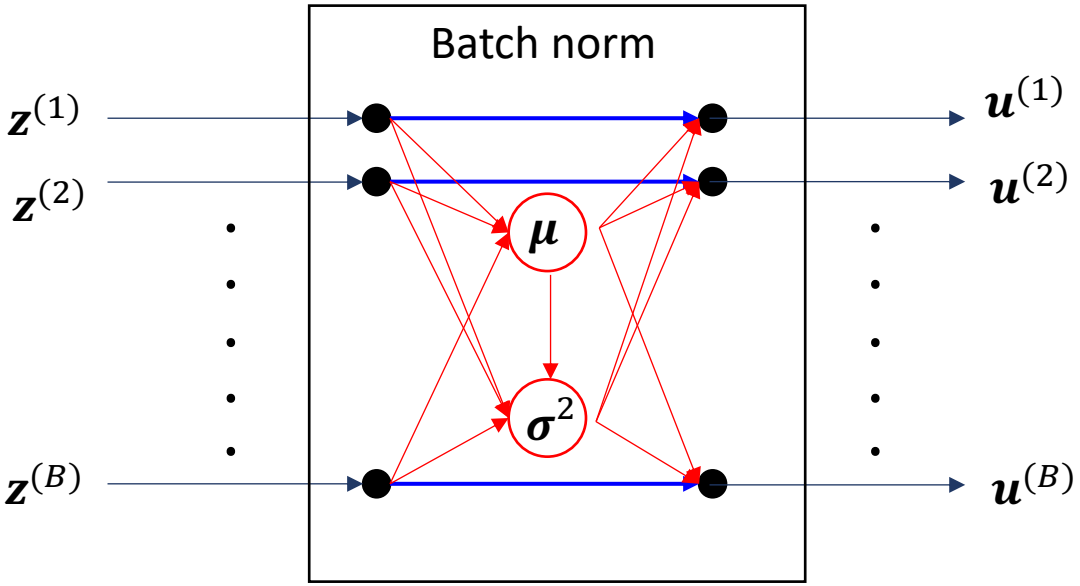
$$u^{(n)} = \frac{z^{(n)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\sigma_B^2 = \frac{1}{B} \sum_{n \in mb} \left(z^{(n)} - \mu_B\right)^2$$

$$\mu_B = \frac{1}{B} \sum_{n \in mb} z^{(n)}$$

$$\frac{\partial loss}{\partial z^{(n)}} = \frac{\partial loss}{\partial u^{(n)}} \cdot \frac{\partial u^{(n)}}{\partial z^{(n)}} + \frac{\partial loss}{\partial \sigma_B^2} \cdot \frac{\partial \sigma_B^2}{\partial z^{(n)}} + \frac{\partial loss}{\partial \mu_B} \cdot \frac{\partial \mu_B}{\partial z^{(n)}}$$

Batch normalization: Backpropagation



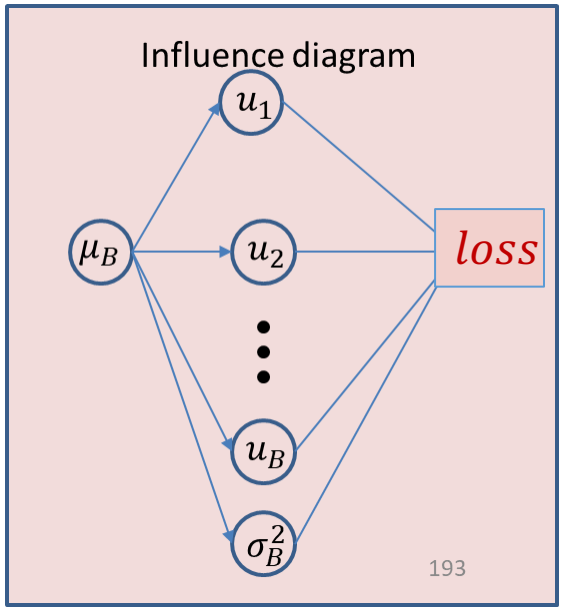
$$\frac{\partial loss}{\partial z^{(n)}} = \frac{\partial loss}{\partial u^{(n)}} \cdot \frac{\partial u^{(n)}}{\partial z^{(n)}} + \frac{\partial loss}{\partial \sigma_B^2} \cdot \frac{\partial \sigma_B^2}{\partial z^{(n)}} + \frac{\partial loss}{\partial \mu_B} \cdot \frac{\partial \mu_B}{\partial z^{(n)}}$$

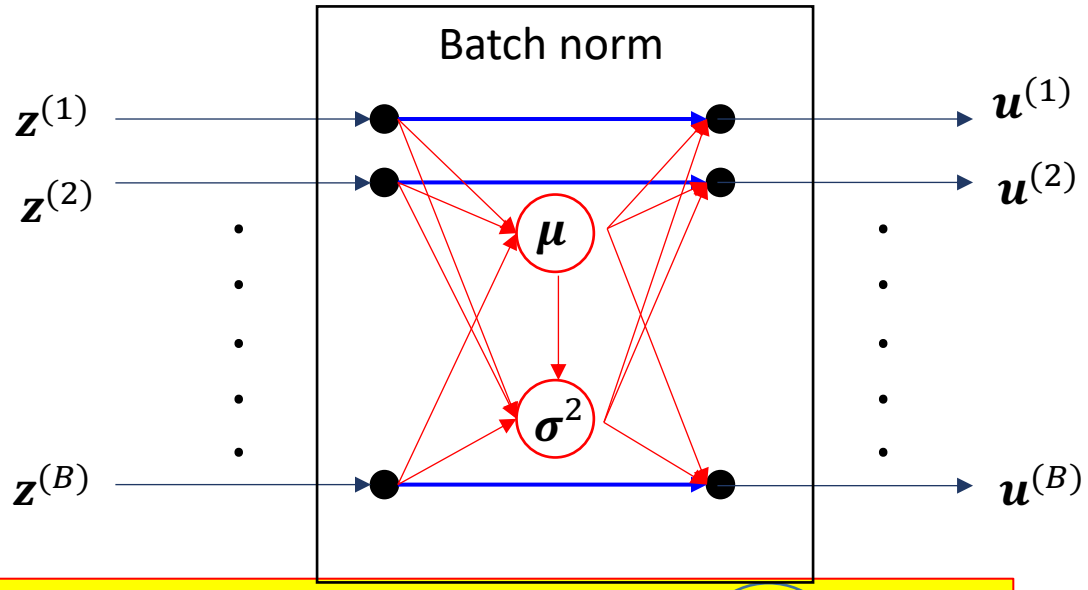
$$u^{(n)} = \frac{z^{(n)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\frac{\partial loss}{\partial \sigma_B^2} = \sum_{n \in mb} \frac{\partial u^{(n)}}{\sigma_B^2} \frac{\partial loss}{\partial u^{(n)}}$$

$$= \frac{-1}{2} (\sigma_B^2 + \epsilon)^{-3/2} \sum_{n \in mb} \left(z^{(n)} - \mu_B\right)^T \frac{\partial loss}{\partial u^{(n)}}$$

Batch normalization: Backpropagation





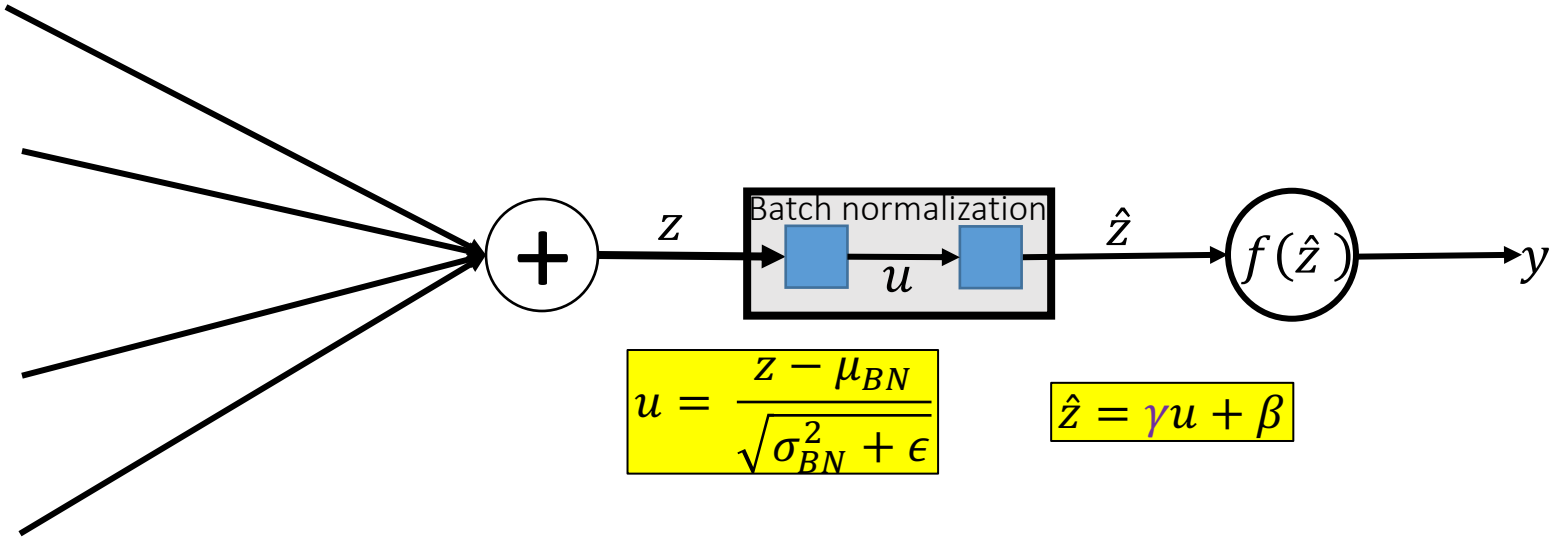
$$\frac{\partial loss}{\partial z^{(n)}} = \frac{\partial loss}{\partial u^{(n)}} \cdot \frac{\partial u^{(n)}}{\partial z^{(n)}} + \frac{\partial loss}{\partial \sigma_B^2} \cdot \frac{\partial \sigma_B^2}{\partial z^{(n)}} + \frac{\partial loss}{\partial \mu_B} \cdot \frac{\partial \mu_B}{\partial z^{(n)}}$$

$$u^{(n)} = \frac{z^{(n)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$\frac{\partial loss}{\partial \mu_B} = \left(\sum_{n \in mb} \frac{\partial u^{(n)}}{\partial \mu_B} \frac{\partial loss}{\partial u^{(n)}} \right) + \frac{\partial \sigma_B^2}{\partial \mu_B} \frac{\partial loss}{\partial \sigma_B^2}$$

$$= \left(\sum_{n \in mb} \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \frac{\partial loss}{\partial u^{(n)}} \right) + \frac{\sum_{n \in mb} -2\left(z^{(n)} - \mu_B\right)}{B} \frac{\partial loss}{\partial \sigma_B^2}$$

Batch normalization:



- On test data, BN requires μ_B and σ_B^2.
- We will use the *average over all training minibatches*

$$\mu_{BN} = \frac{1}{Nbatches} \sum_{batch} \mu_B(batch)$$

$$\sigma_{BN}^2 = \frac{B}{(B-1)Nbatches} \sum_{batch} \sigma_B^2(batch)$$

- Note: these are *neuron-specific*
  - $\mu_B(batch)$ and $\sigma_B^2(batch)$ here are obtained from the *final converged network*
  - The $B/(B-1)$ term gives us an unbiased estimator for the variance

Batch normalization

- Prepare parameters for test-time too:

Input: $x : N \times D$

$\mu_j =$ (Running) average of values seen during training — Per-channel mean, shape is D

Learnable scale and shift parameters:

$\gamma, \beta : D$

$\sigma_j^2 =$ (Running) average of values seen during training — Per-channel var, shape is D

$$z_{norm}^{(i)[l]} = \frac{z^{(i)[l]} - \mu^{[l]}}{\sqrt{\sigma^{2[l]} + \epsilon}}$$

$$\hat{z}^{(i)[l]} = \gamma^{[l]} z_{norm}^{(i)[l]} + \beta^{[l]}$$

Batch normalization at test time

- At test time BatchNorm layer functions differently:
  - The mean/std are not computed based on the batch.
  - Instead, a single fixed empirical mean of activations during training is used. (e.g. can be estimated during training with running averages)

for l=1…L

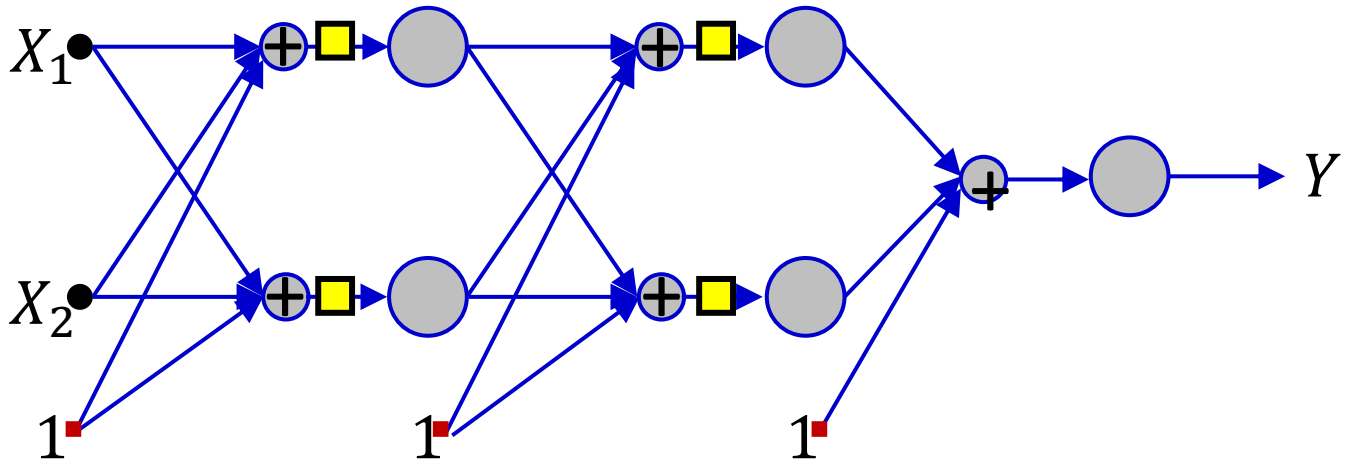
$\mu^{[l]} = 0$

for i=1…m

for l=1…L

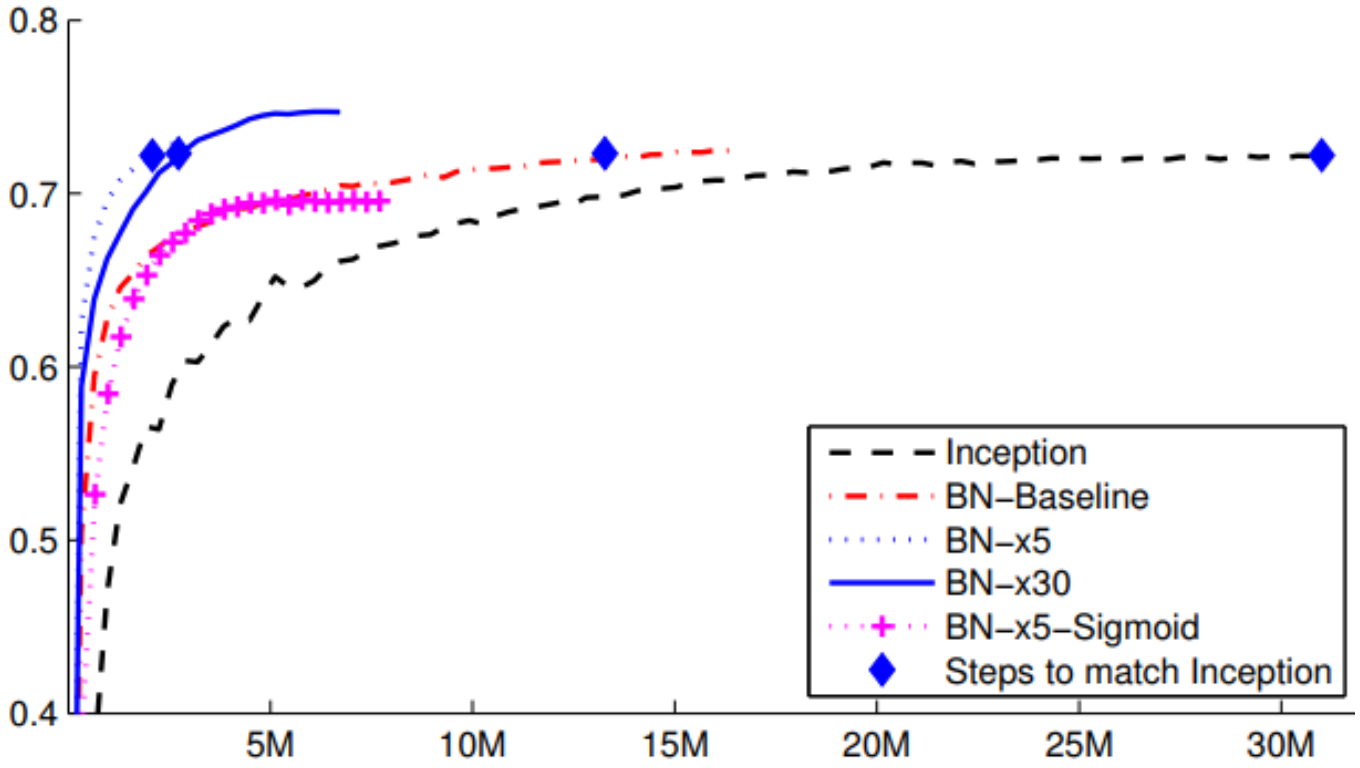
$$\mu^{[l]} = \beta\mu^{[l]} + (1-\beta)\frac{\sum_{j=1}^{b} X_j^{\{i\}}}{b}$$

Batch normalization



- Batch normalization may only be applied to *some* layers
  - Or even only selected neurons in the layer
- Improves both convergence rate and neural network performance
  - Anecdotal evidence that BN eliminates the need for dropout
  - To get maximum benefit from BN, learning rates must be increased and learning rate decay can be faster
    - Since the data generally remain in the high-gradient regions of the activations
  - Also needs better randomization of training data order

Batch Normalization: Typical result



- Performance on Imagenet, from Ioffe and Szegedy, JMLR 2015

Batch normalization: summary

- Improves gradient flow through the network
  - by reducing the dependence of gradients on the scale (or initialization) of the parameters
- Allows higher learning rates
  - Greatly accelerating the learning process
- Speedup the learning process
  - Reduces the strong dependence on the initialization
- Acts as a form of regularization in a funny way (as we will see)

Resources

- Deep Learning Book, Chapter 8.
- Please see the following note:
  - http://cs231n.github.io/neural-networks-3/