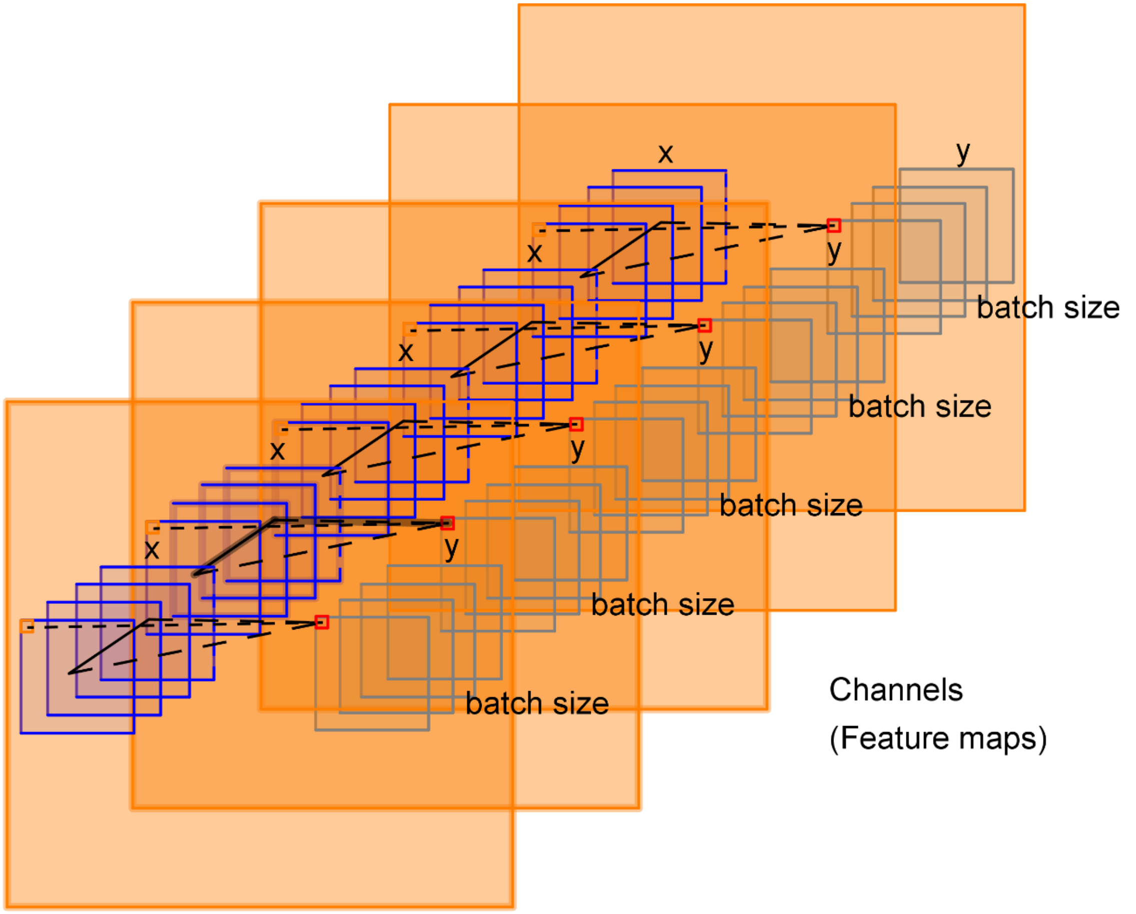
Batch Normalization

Accelerating Deep Network Training
by
Reducing Internal Covariate Shift

Amit Patel
Sambit Pradhan

- *Introduction*
- *Internal Covariate Shift*
- *Batch Normalization*
- *Computational Graph of Batch Normalization*
- *Real World Application*
- *Practical Demo*
- *Discussion*



Introduction

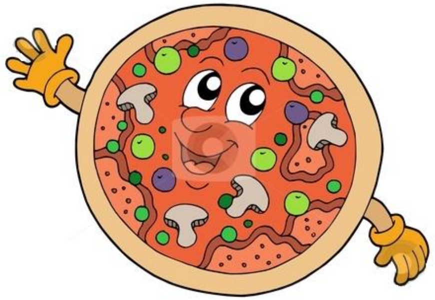
Batch Normalization

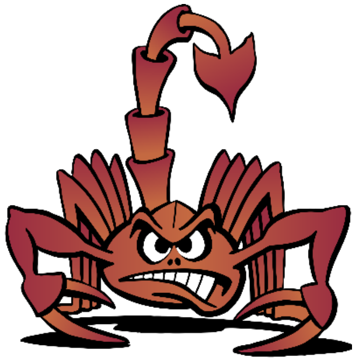
The story of Scaredy Cat

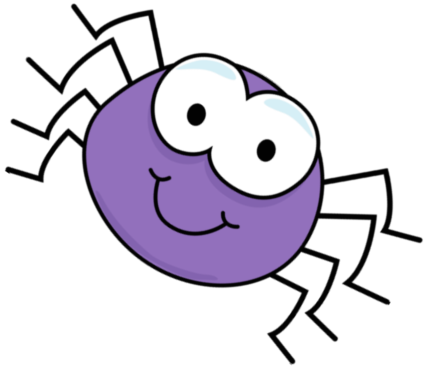
Baby kitten for Christmas ! Yey !

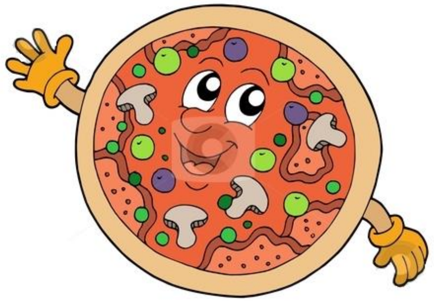
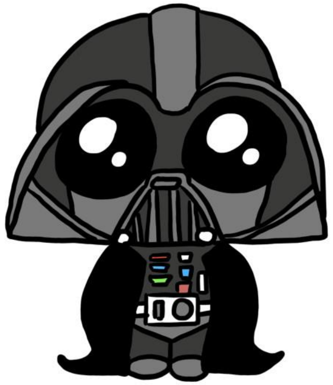
Young, Naïve, Innocent, Scared of everything

= Not Trained

Task – Train it to eat 



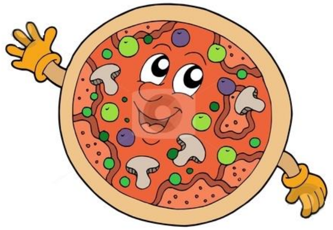



First step in Training Scaredy Cat to Eat

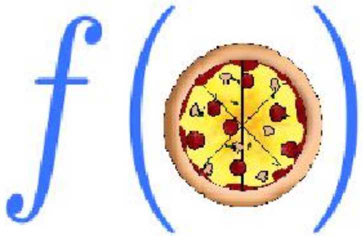
Train it to Recognize a Pizza

Training Data



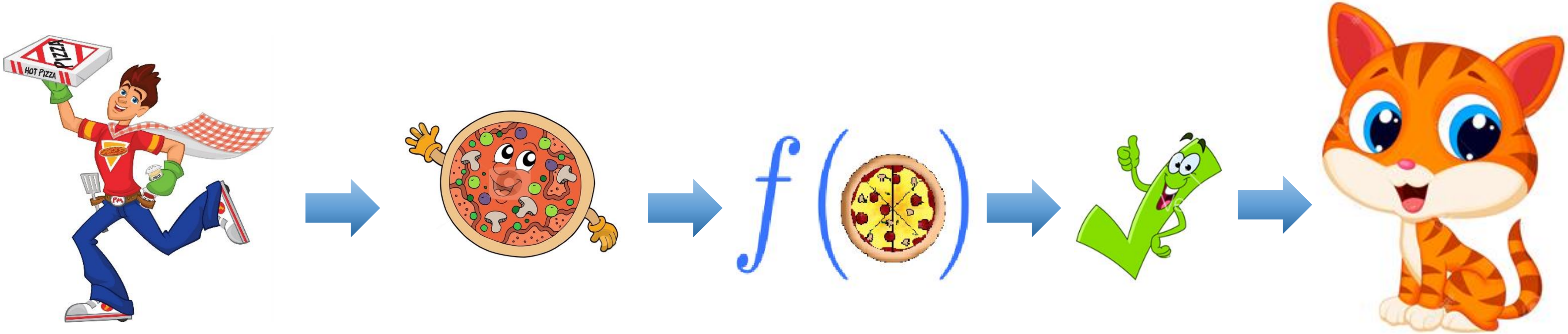
Features

Model



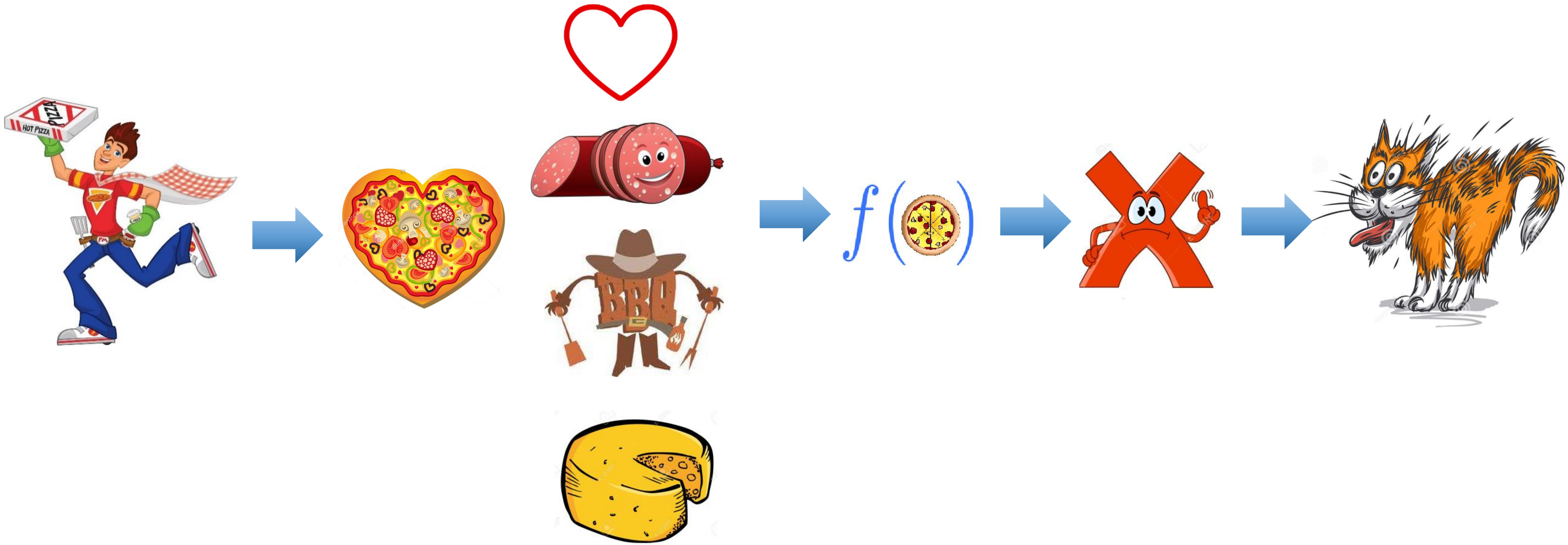
Output

Every Day



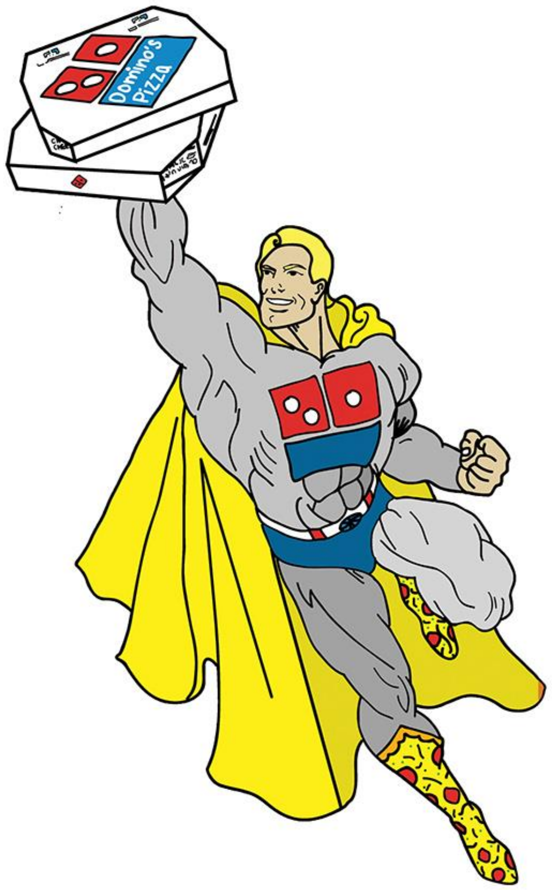
One Special Day

Features

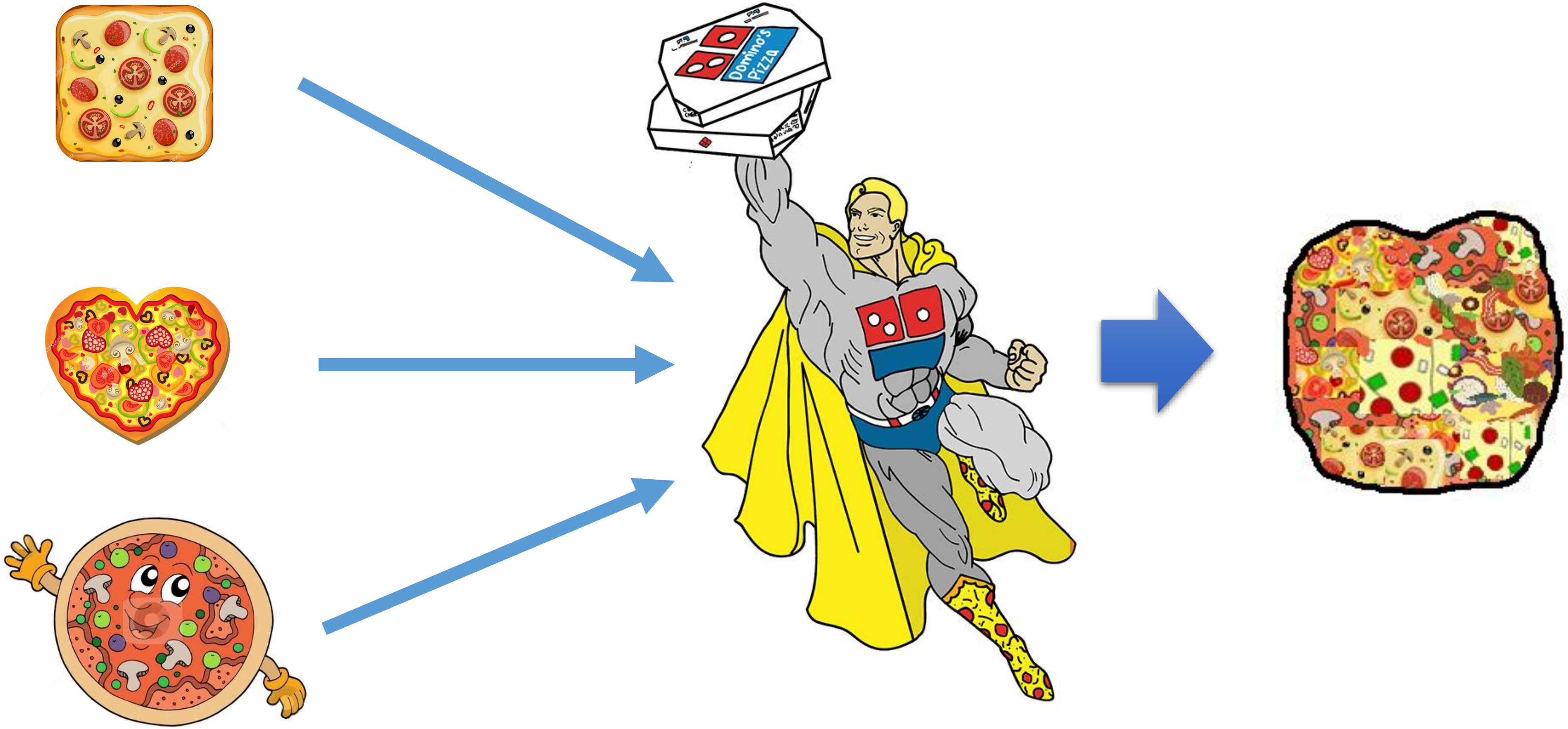


Training Data with distribution of the features is different

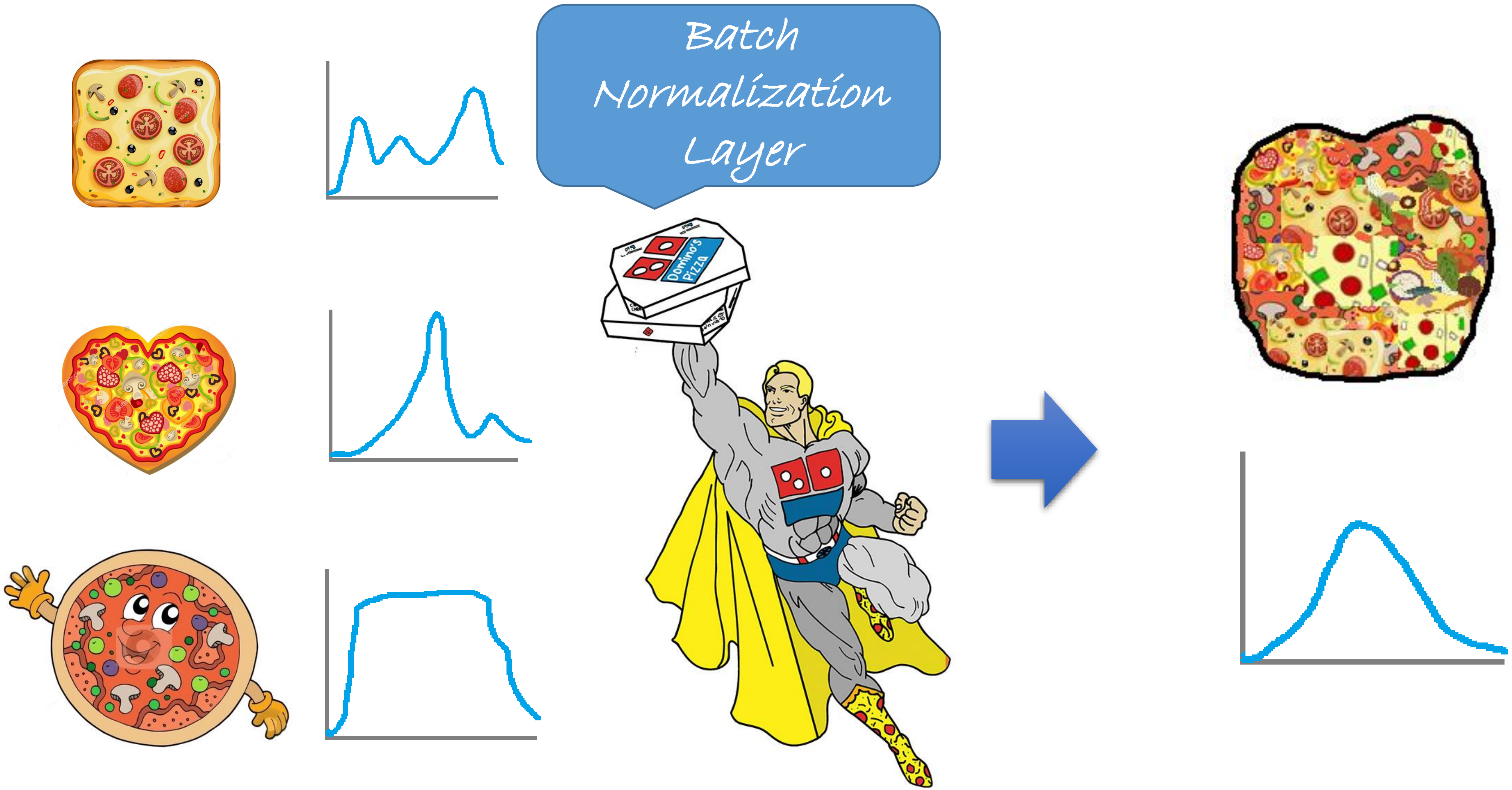
The Pizza Super Duper Man



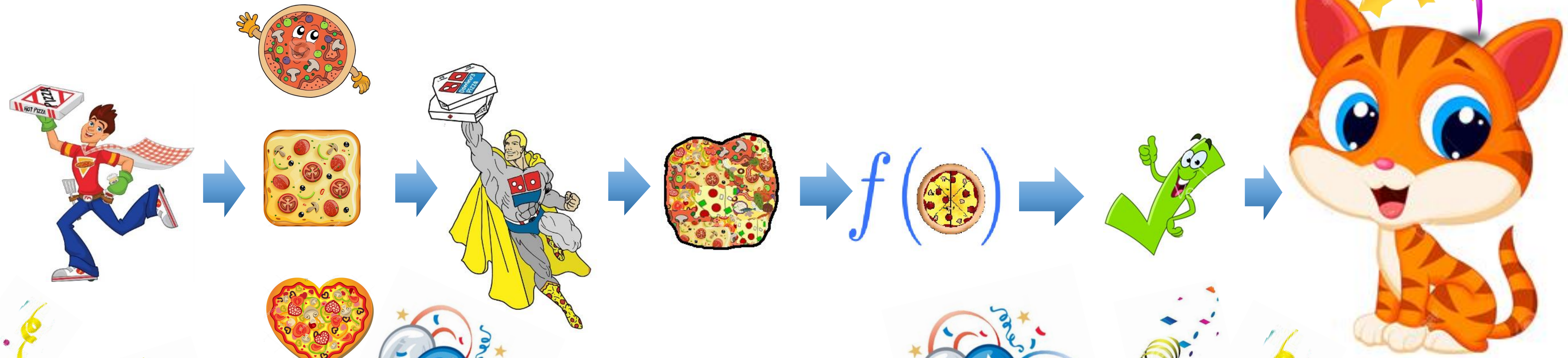
The Pizza Super Duper Man



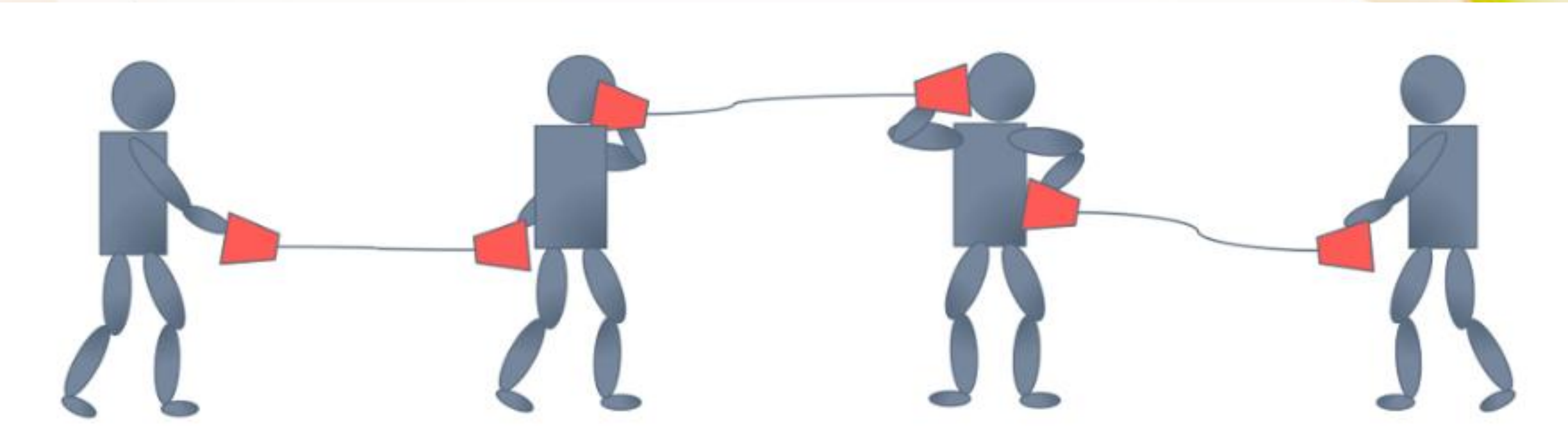
The Pizza Super Duper Man - Normalizes



Happily Ever After



Internal Covariate Shift



Covariate Shift

- **Covariate** – The Features of the Input Data
- **Covariate Shift** - Formally, is defined as a change in the distribution of a function's domain.
- Feature space of Target is drastically different than Source
- Training Data with distribution of the features is different.
- Informally, when input feature change, and algorithm can't deal with it, thus slowing the training.

Let's say you have a goal to reach, which is easier, a fixed goal a goal that keeps moving about? It is clear that a static goal is much easier to reach than a dynamic goal.

Why Covariate Shift slows learning ?

- **IID** – Independent and identically distributed - Each random variable has the same probability distribution as the others and all are mutually independent.
- Between the source (S) and target (T) given the same observation X = x conditional distributions of Y is same in both domains.

$$P_s(Y|X = x) = P_t(Y|X = x) \ \forall \ x \in X$$

In real world – In the wild

$$P_s(Y|X = x) \neq P_t(Y|X = x)$$

Why Covariate Shift slows learning ?

- To analyze the issue – We consider a Parametric Model Family $\{P(Y|X, \theta)\}_{\theta \notin \Theta}$
- We select a model $\{P(Y|X, \theta^*)\}$, which minimizes the expected classification error.
- If none of the models in the model family can exactly match the true relation between X and Y => there does not exist any $\theta \notin \Theta$ such that $P(Y|X = x, \theta) = P(Y|X = x)\ \forall\ x \in X$ => We call this a misspecified model family.
- With a misspecified model family, the optimal model we select depends on $P(X)$ and if $Ps(X) \neq Pt(X)$ then the optimal model for the target domain will differ from that for the source domain

Why Covariate Shift slows learning ?

- The intuitive is that the optimal model performs better in dense regions of   than in sparse regions of , because the dense regions dominate the average classification error, which is what we want to minimize. If the dense regions of   are different in the source and target then the optimal model for the source domain domain will no longer be optimal for the target domain.
- We re-weigh the

$$\frac{P_t(x,y)}{P_s(x,y)} = \frac{P_t(x)}{P_s(x)} \frac{P_t(y|x)}{P_s(y|x)}$$

$$= \frac{P_t(x)}{P_s(x)}.$$

- We therefore re-weight each training instance with $\frac{P_t(x)}{P_s(x)}$

Why Covariate Shift slows learning ?

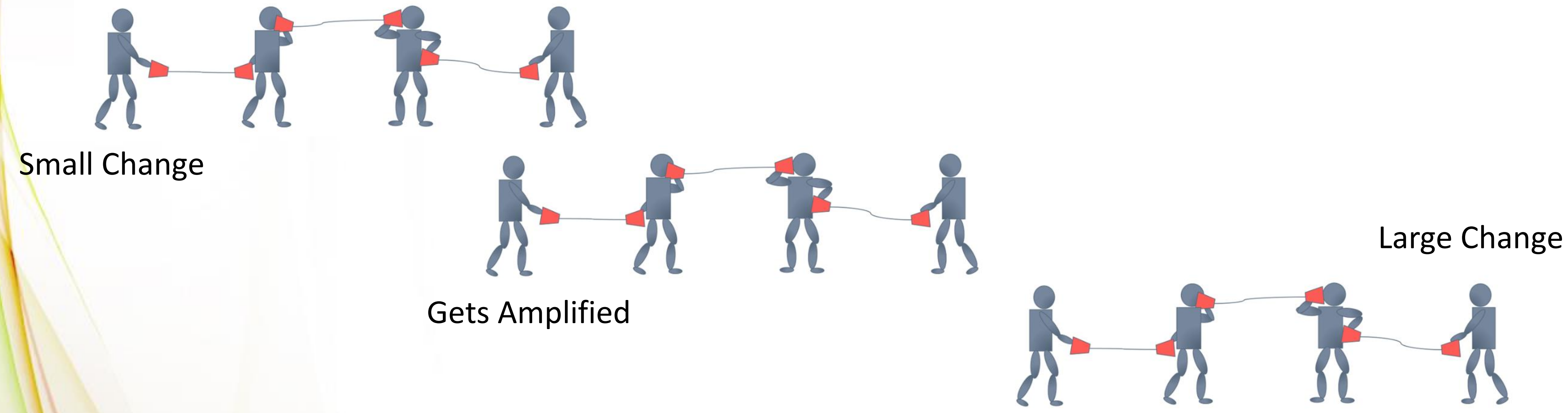
- Improving predictive inference under covariate shift by weighting the log-likelihood function - *Hidetoshi Shimodaira*

Internal Covariate Shift

- Deep learning – Is parameterized in a hierarchical fashion
- The first layere (Input Layer) looksa at the souce and the output of the first layer feeds the second layer, the second feeds the third, and so on.
- The distribution of the input is important - Because we are actually learning a MODEL from the training data and NOT the right model.
- Internal Covariate Shift - Small changes to the network get amplified down the network which leads to change in the input distribution to internal layers of the deep network.

Internal Covariate Shift

- Internal covariate shift refers to covariate shift occurring within a neural network, i.e. going from (say) layer 2 to layer 3. This happens because, as the network learns and the weights are updated, the distribution of outputs of a specific layer in the network changes. This forces the higher layers to adapt to that drift, which slows down learning.



Internal Covariate Shift

Internal Covariate Shift applied to its parts, such as a sub-network or a layer.

$$\ell = F_2(F_1(\mathrm{u}, \Theta_1), \Theta_2)$$

- Where F1 and F2 are arbitrary transformations, and the parameters Θ1, Θ2 are to be learned so as to minimize the loss ℓ.
- Learning Θ2 can be viewed as if the inputs x = F1(u, Θ1) are fed into the sub-network

$$\ell = F_2(\mathrm{x}, \Theta_2).$$

$$\Theta_2 \leftarrow \Theta_2 - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial F_2(\mathrm{x}_i, \Theta_2)}{\partial \Theta_2}$$

- (for batch size m and learning rate α) is exactly equivalent to that for a stand-alone network F2 with input x.

Batch Normalization

Accelerating Deep Network Training
by
Reducing Internal Covariate Shift

Batch Normalization

Batch Normalization – Is a process normalize each scalar feature independently, by making it have the mean of zero and the variance of 1 and then scale and shift the normalized value *for each training mini-batch thus* reducing internal covariate shift fixing the distribution of the layer inputs x as the training progresses.

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways.

- First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases.
- Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

Batch Normalizing Transform

To remedy internal covariate shift, the solution proposed in the paper is to normalize each batch by both mean and variance

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

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

Batch Normalizing Transform

Let us say that the layer we want to normalize has *d* dimensions **x** = $(x_1, \ldots x_d)$. Then, we can normalize the k^{th} dimension as follows:

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

- the parameters γ and β are to be learned, but it should be noted that the BN transform does not independently process the activation in each training example.
- Rather, BNγ,β(x) depends both on the training example and the other examples in the mini-batch. The scaled and shifted values y are passed to other network layers.

Batch Normalizing Transform

During training we need to backpropagate the gradient of loss ℓ through this transformation, as well as compute the gradients with respect to the parameters of the BN transform.

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m}$$

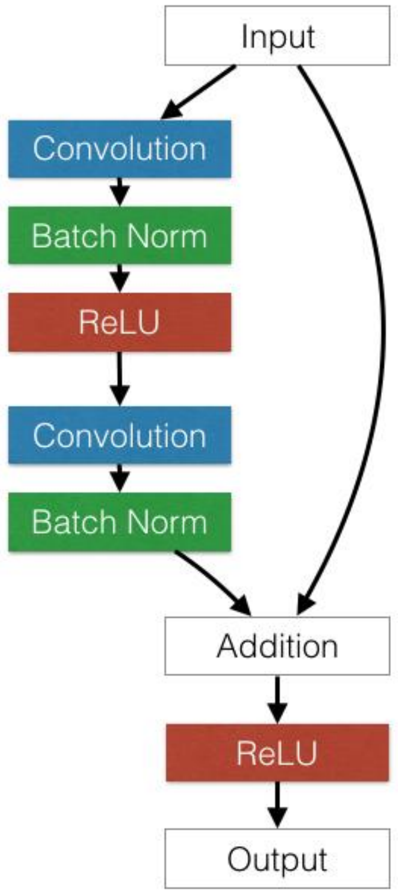
$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

Batch Normalizing Transform

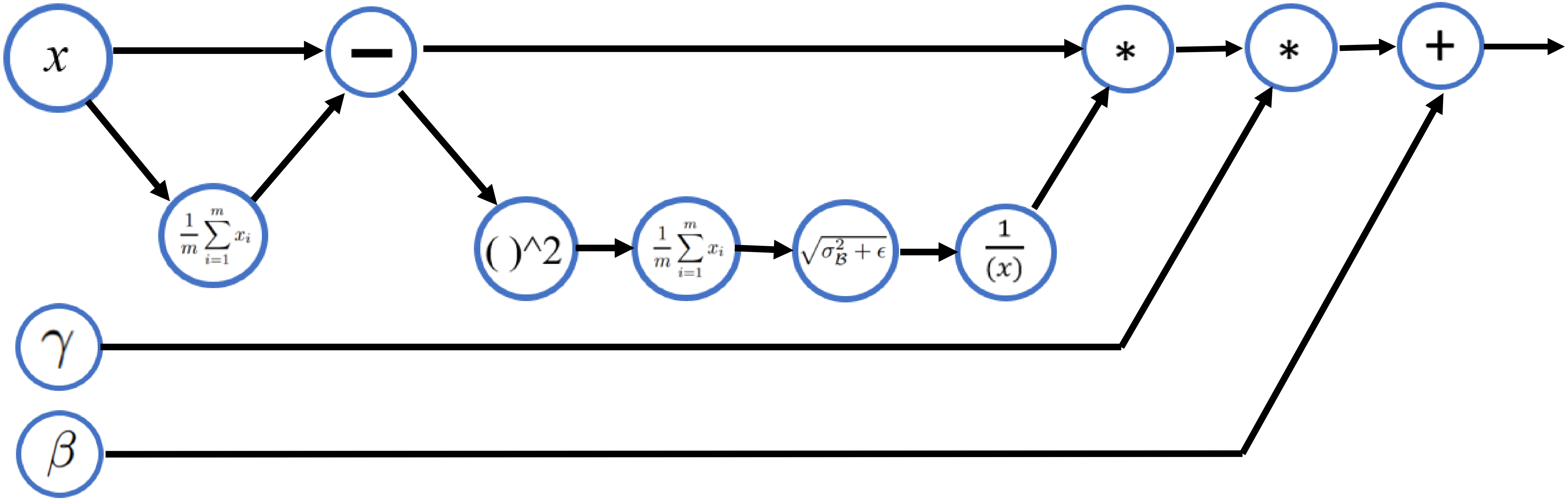
To remedy internal covariate shift, the solution proposed in the paper is to normalize each batch by both mean and variance



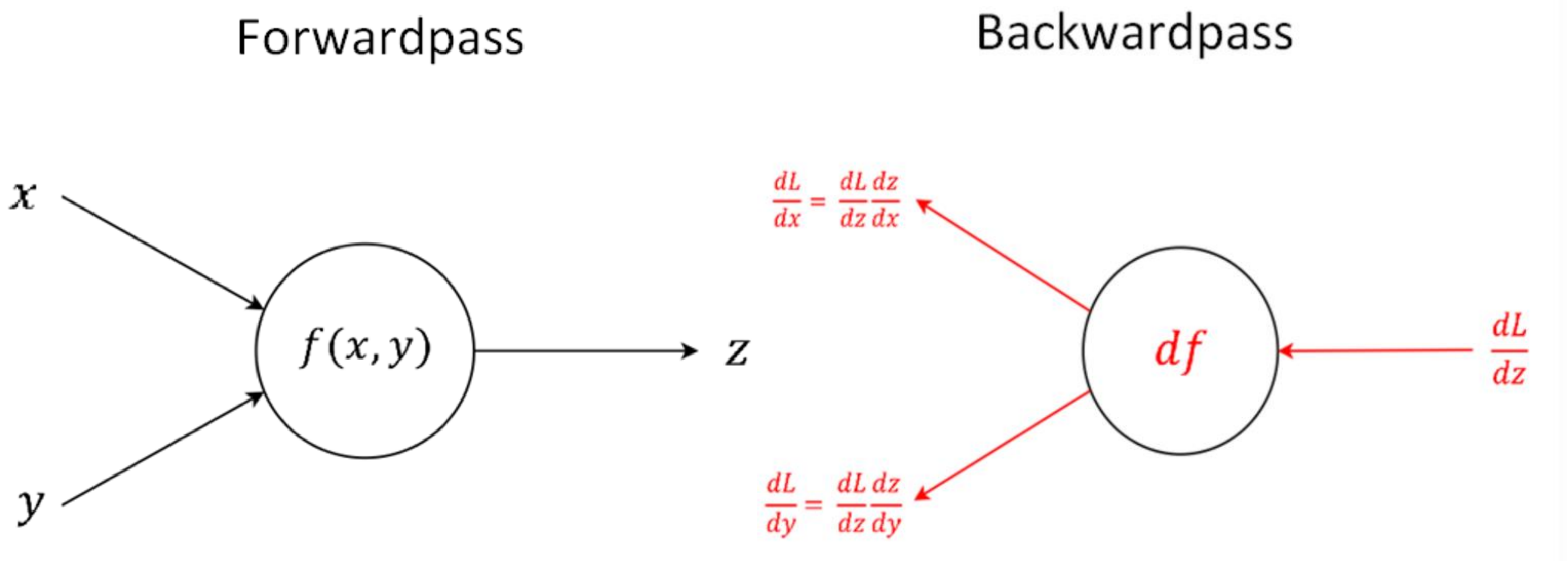
Computational Graph of Batch Normalization

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$



Backpropagation



Chain Rule in Backpropagation

$$f(x,y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x$$

$$f(x,y) = x + y \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1 \quad \frac{\partial f}{\partial y} = 1$$

$$f(x,y) = \max(x,y) \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1(x >= y) \quad \frac{\partial f}{\partial y} = 1(y >= x)$$

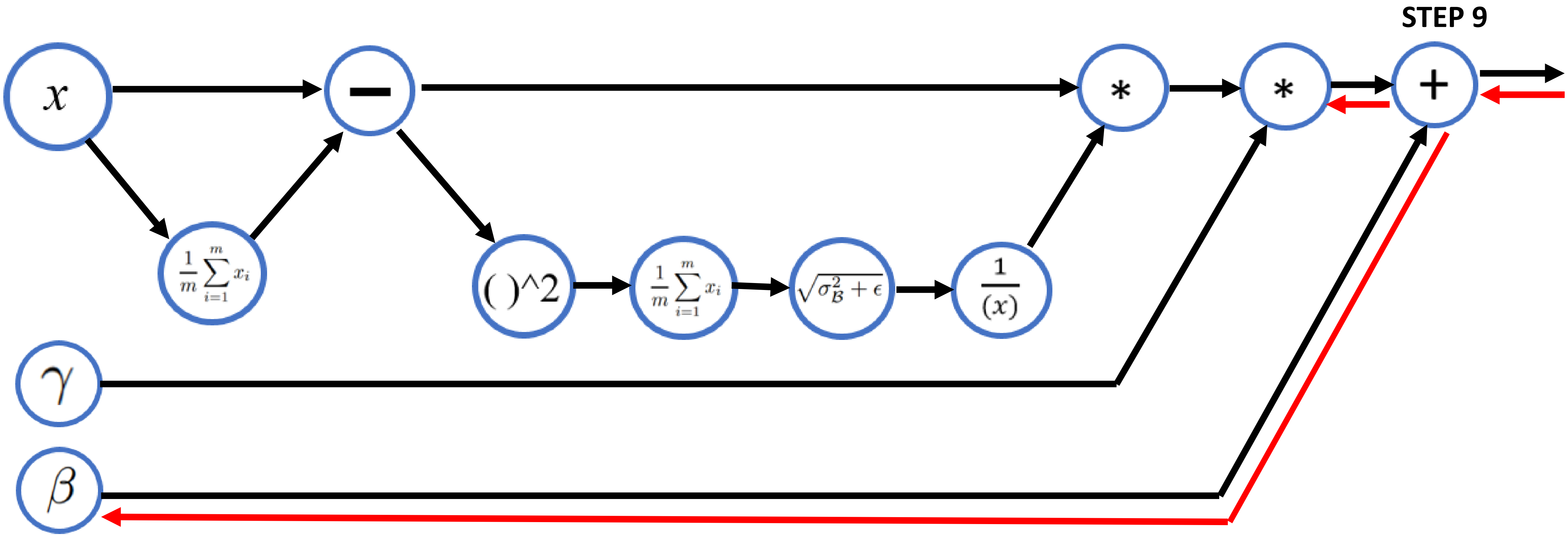
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

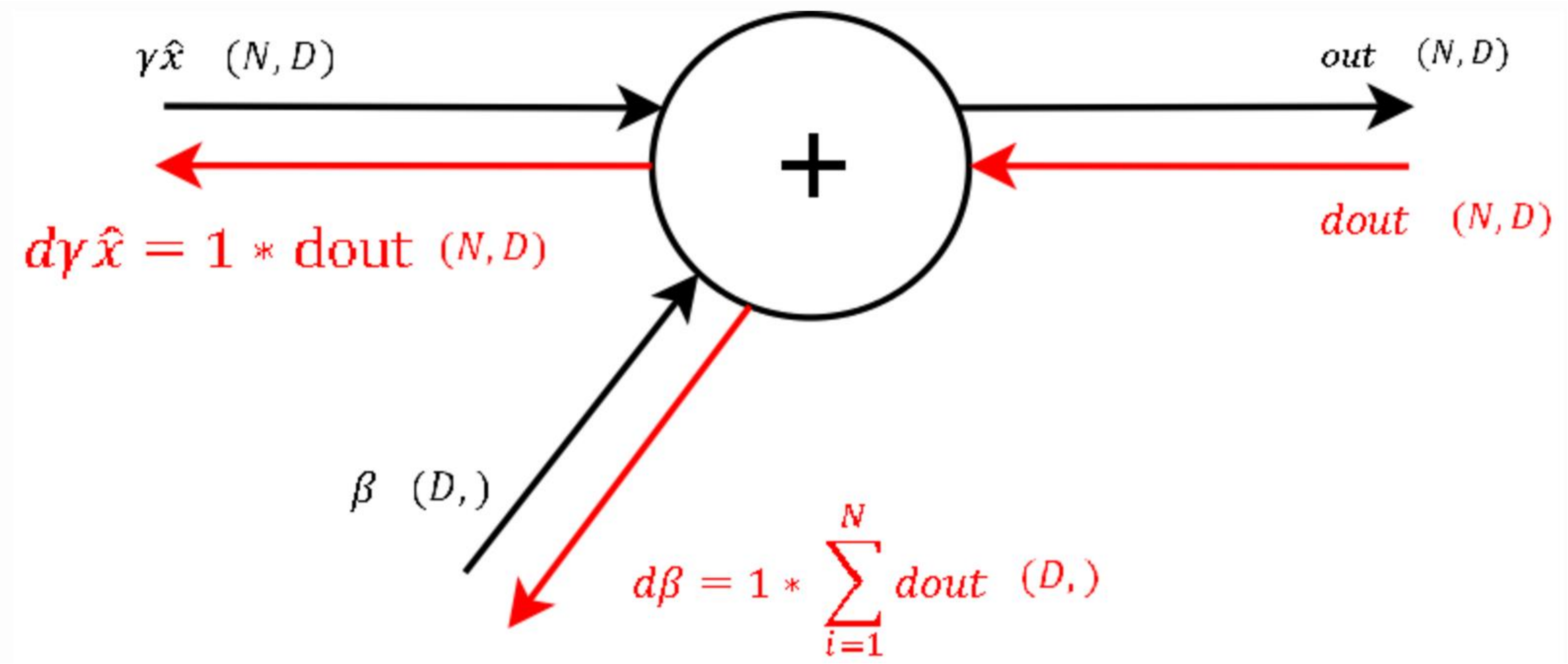
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

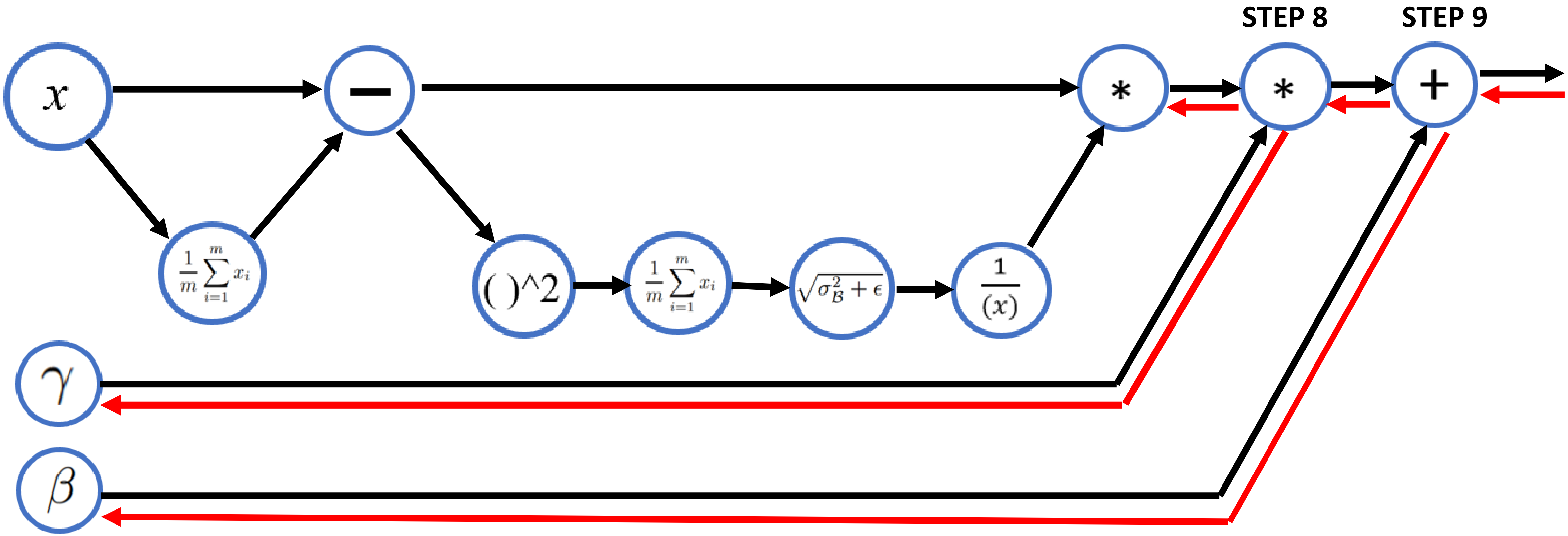
Computational Graph of Batch Normalization



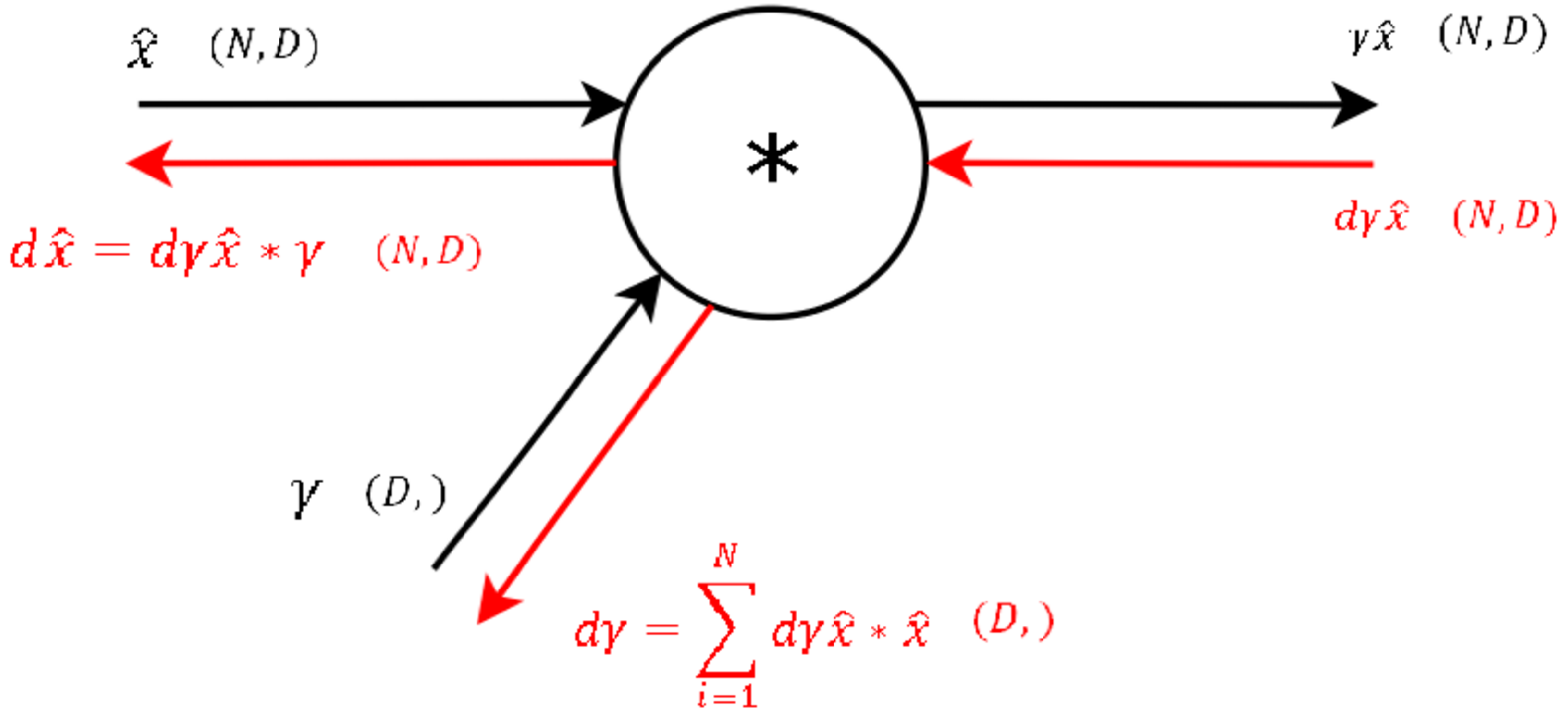
STEP 9 – Summation gate



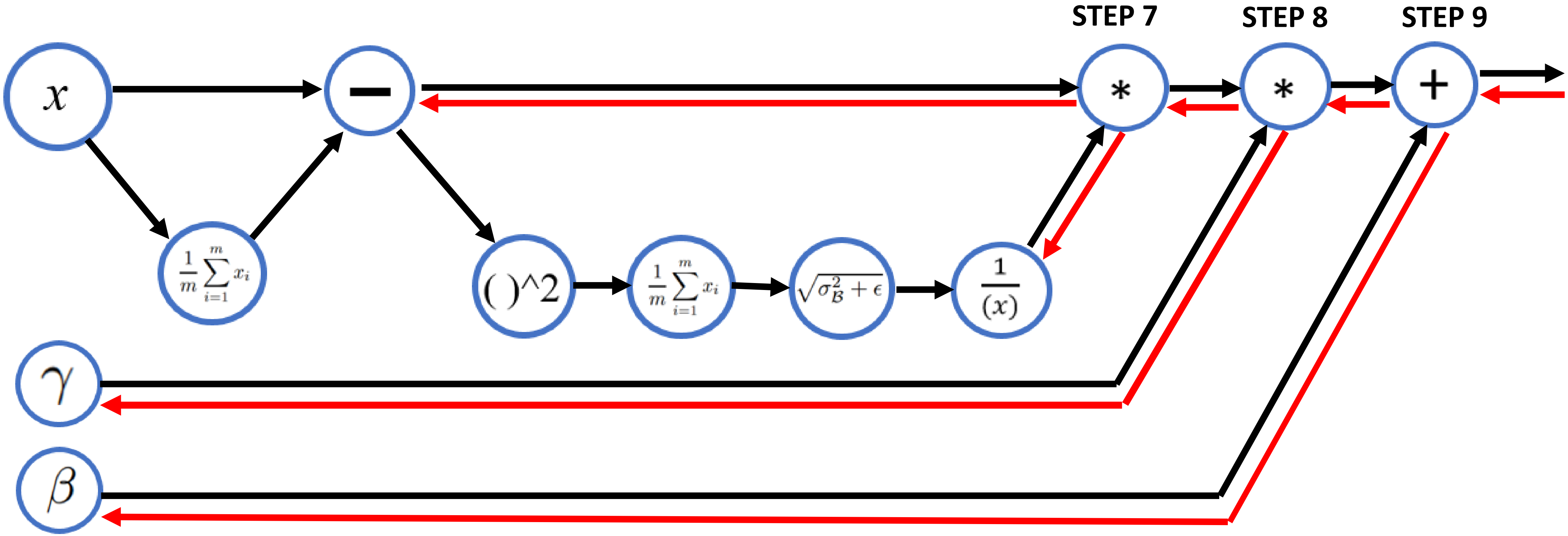
Computational Graph of Batch Normalization



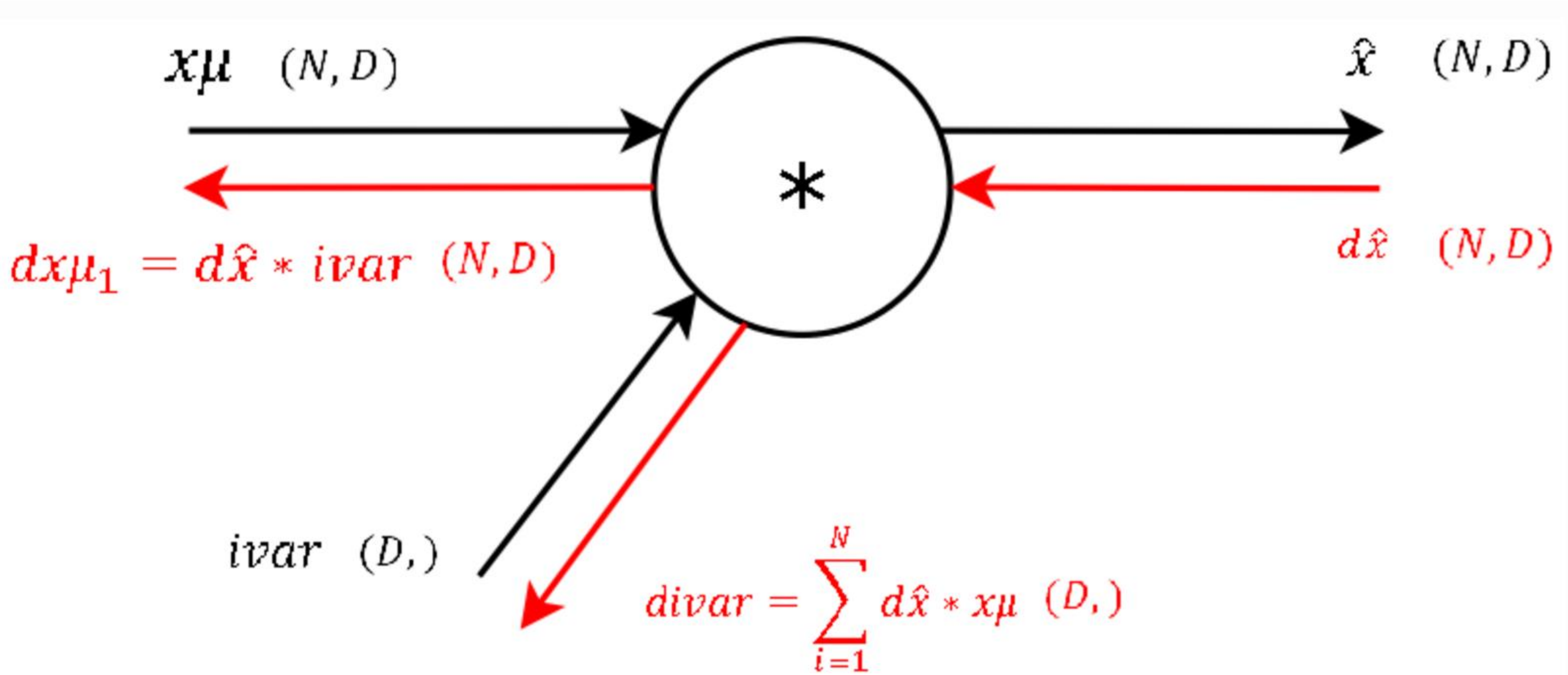
STEP 8 – First Multiplication gate



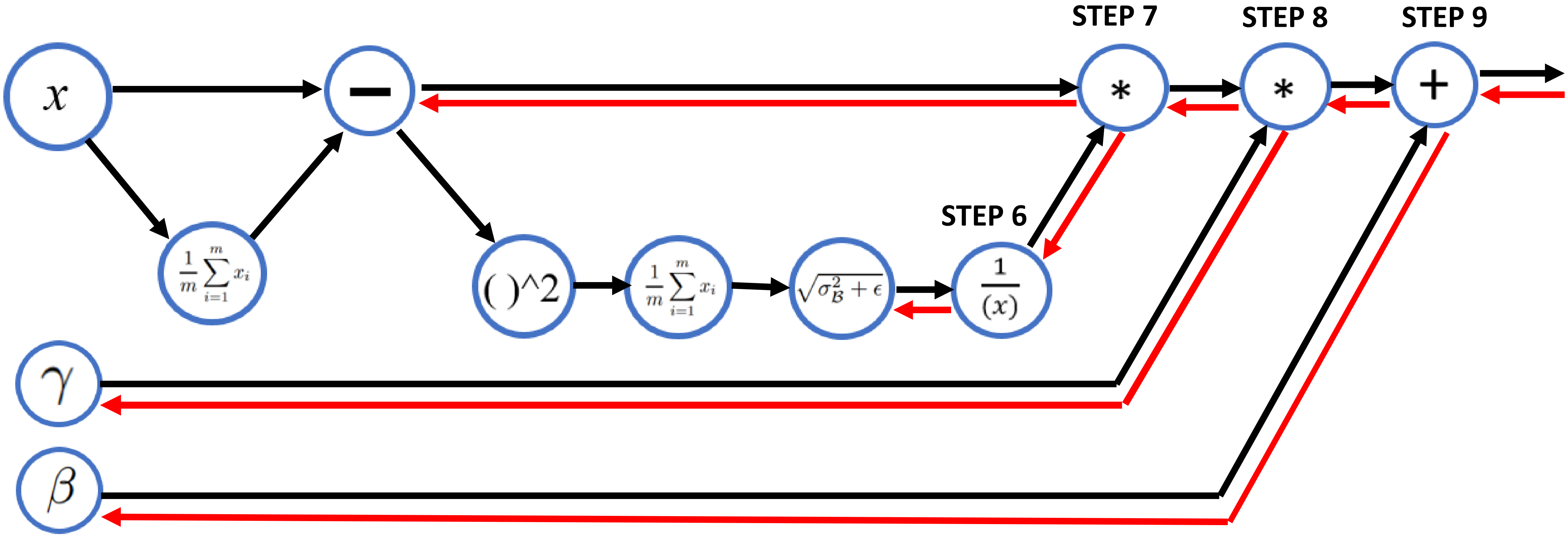
Computational Graph of Batch Normalization



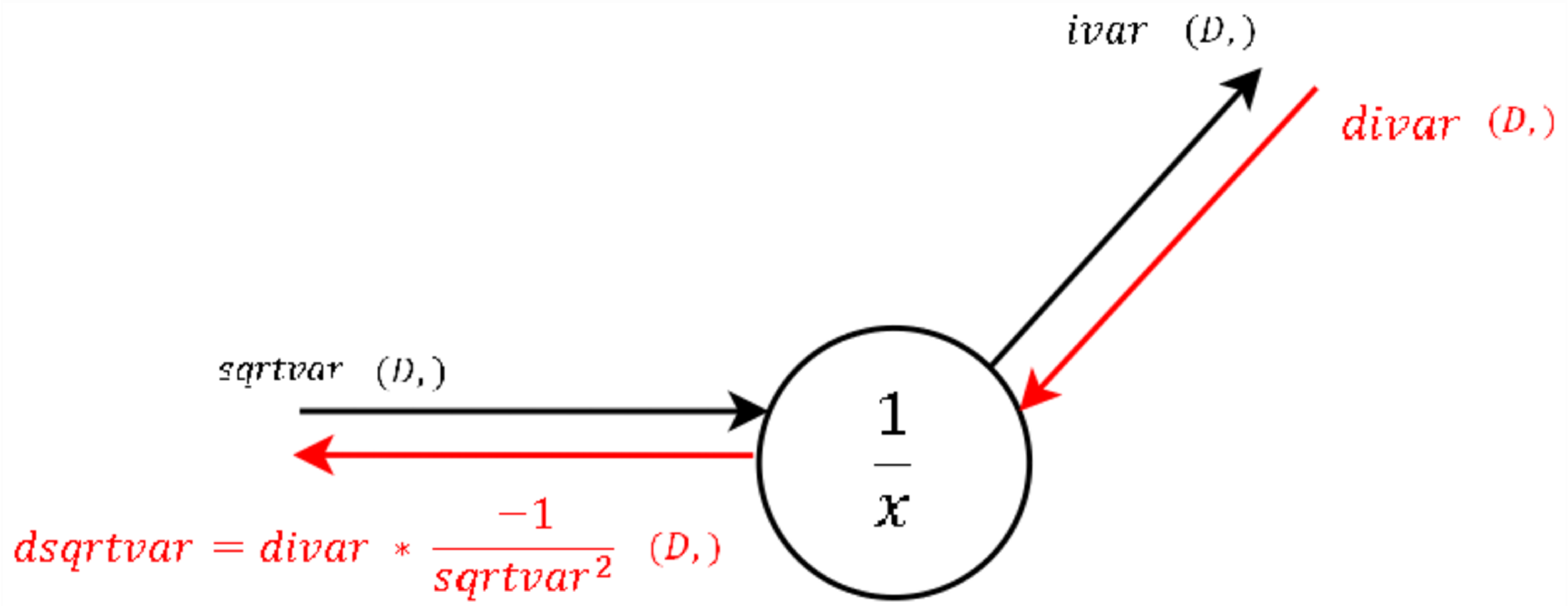
STEP 7 – Second Multiplication gate



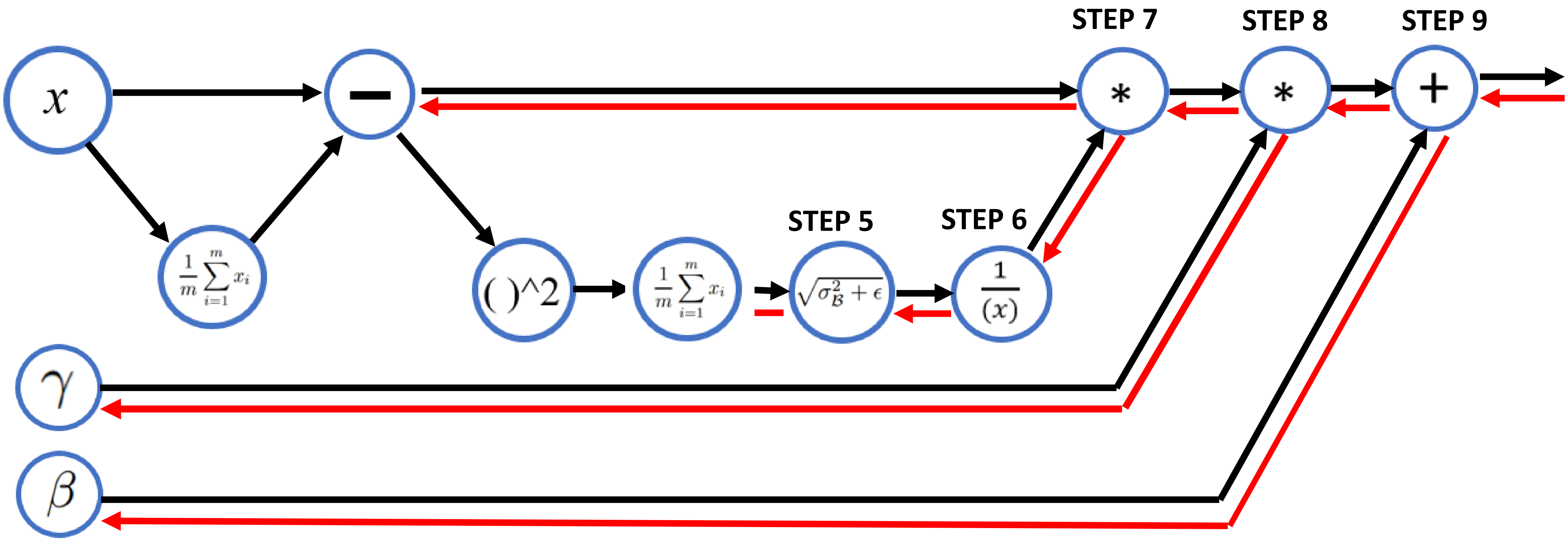
Computational Graph of Batch Normalization



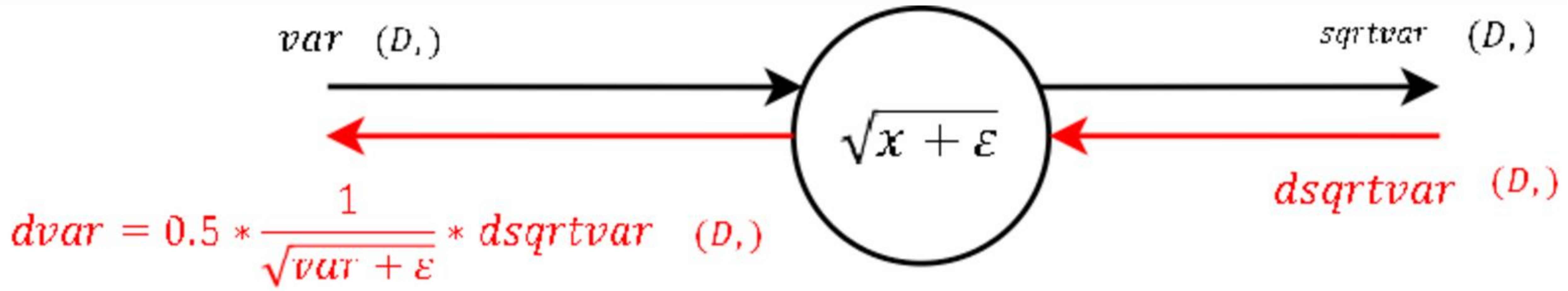
STEP 6 – Inverse gate



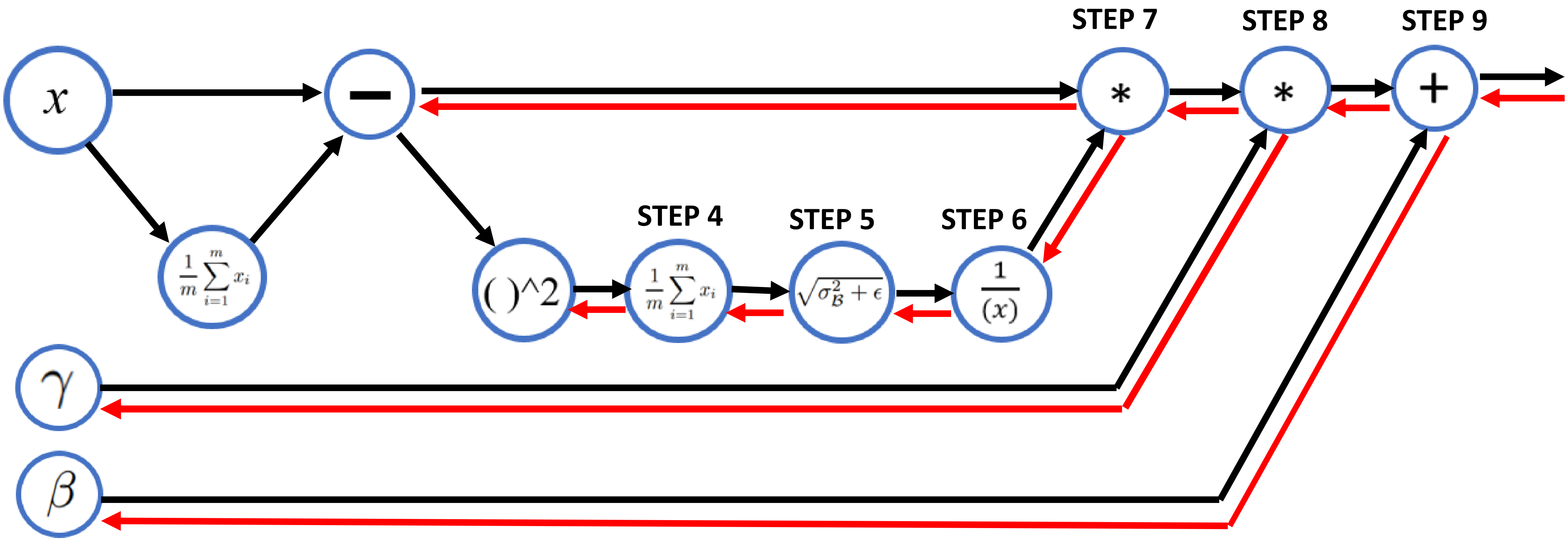
Computational Graph of Batch Normalization



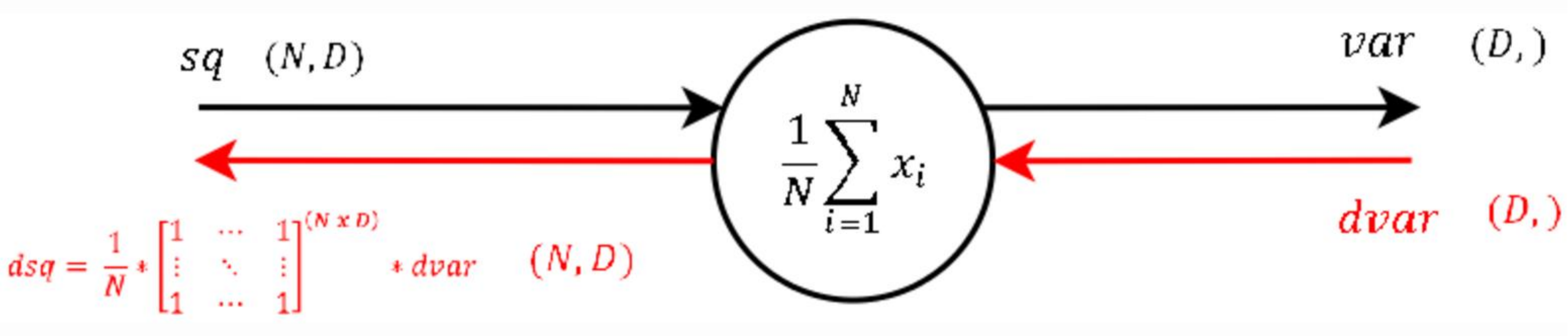
STEP 5 – Square root gate



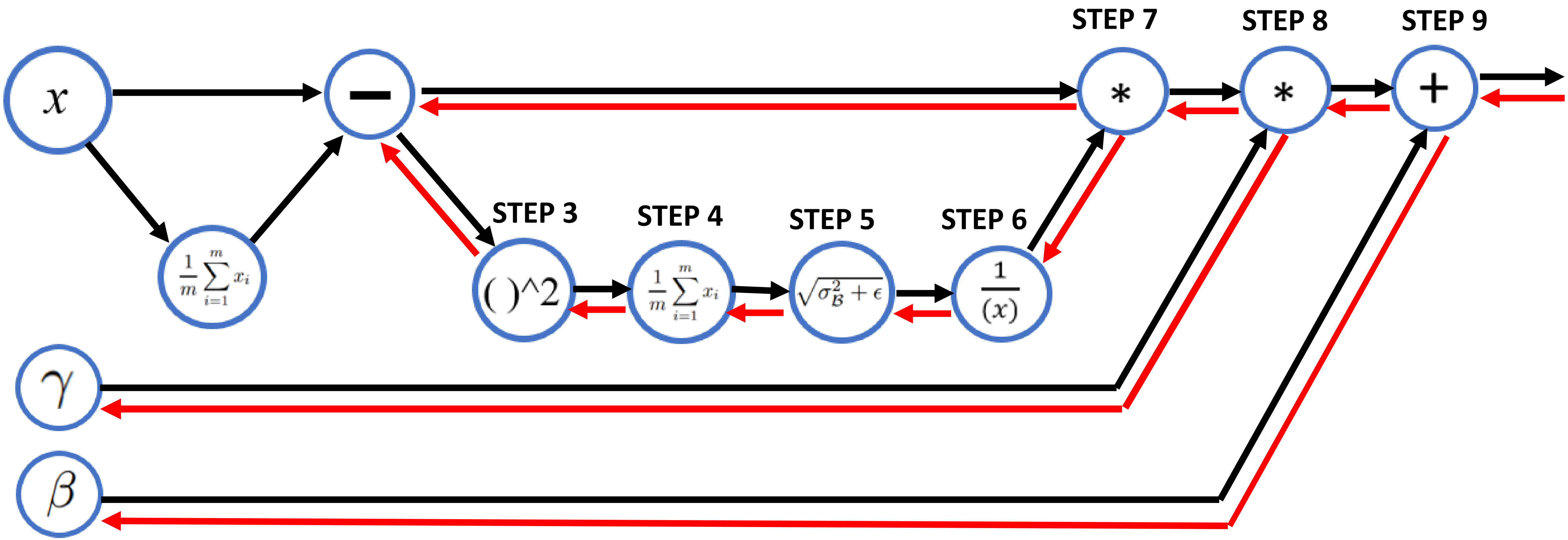
Computational Graph of Batch Normalization



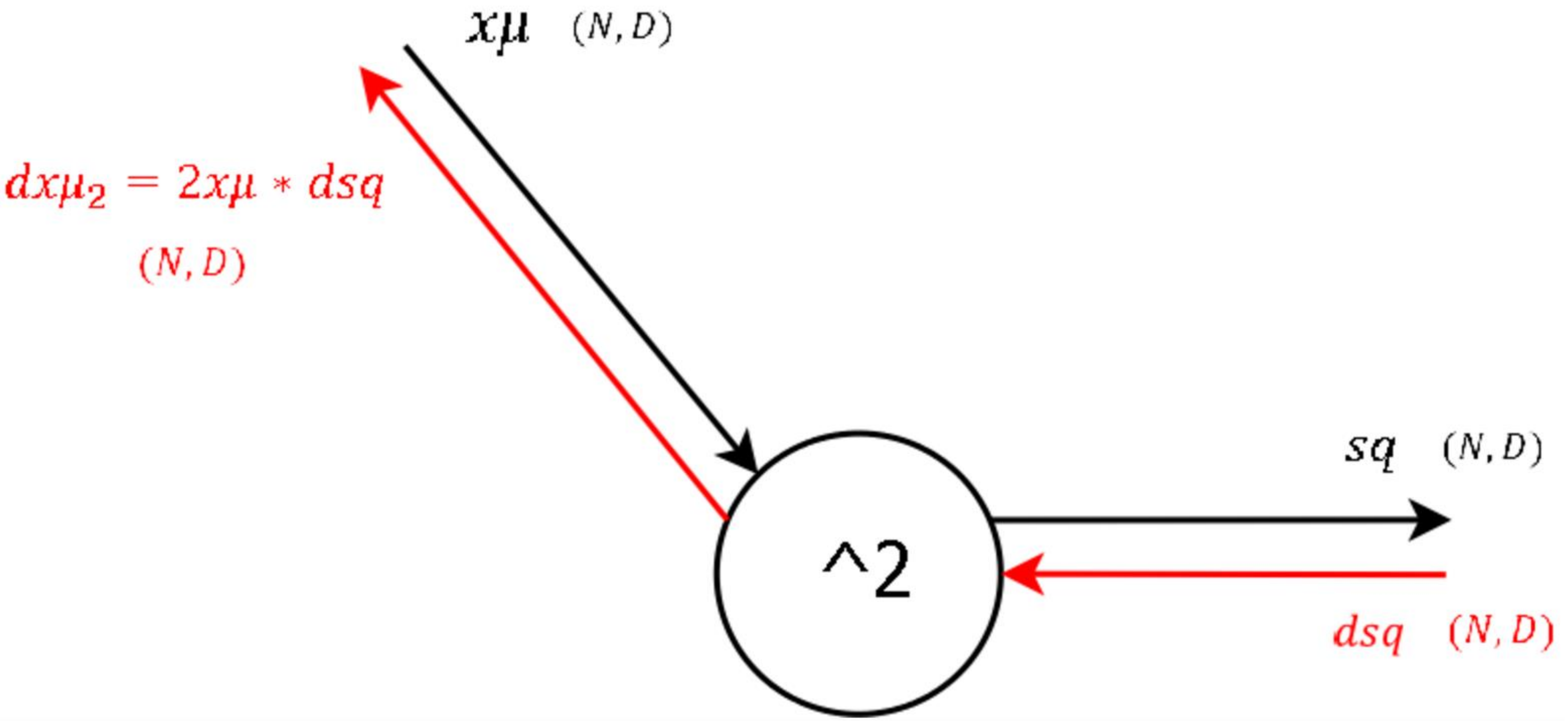
STEP 4 – Mean gate



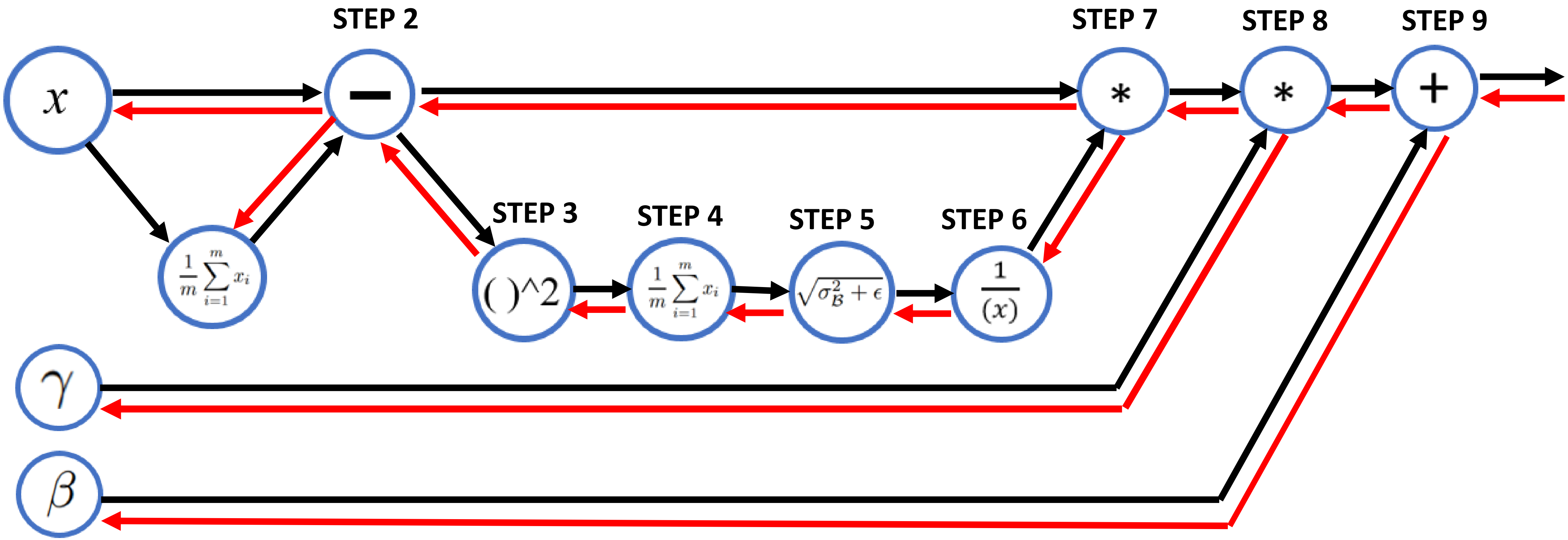
Computational Graph of Batch Normalization



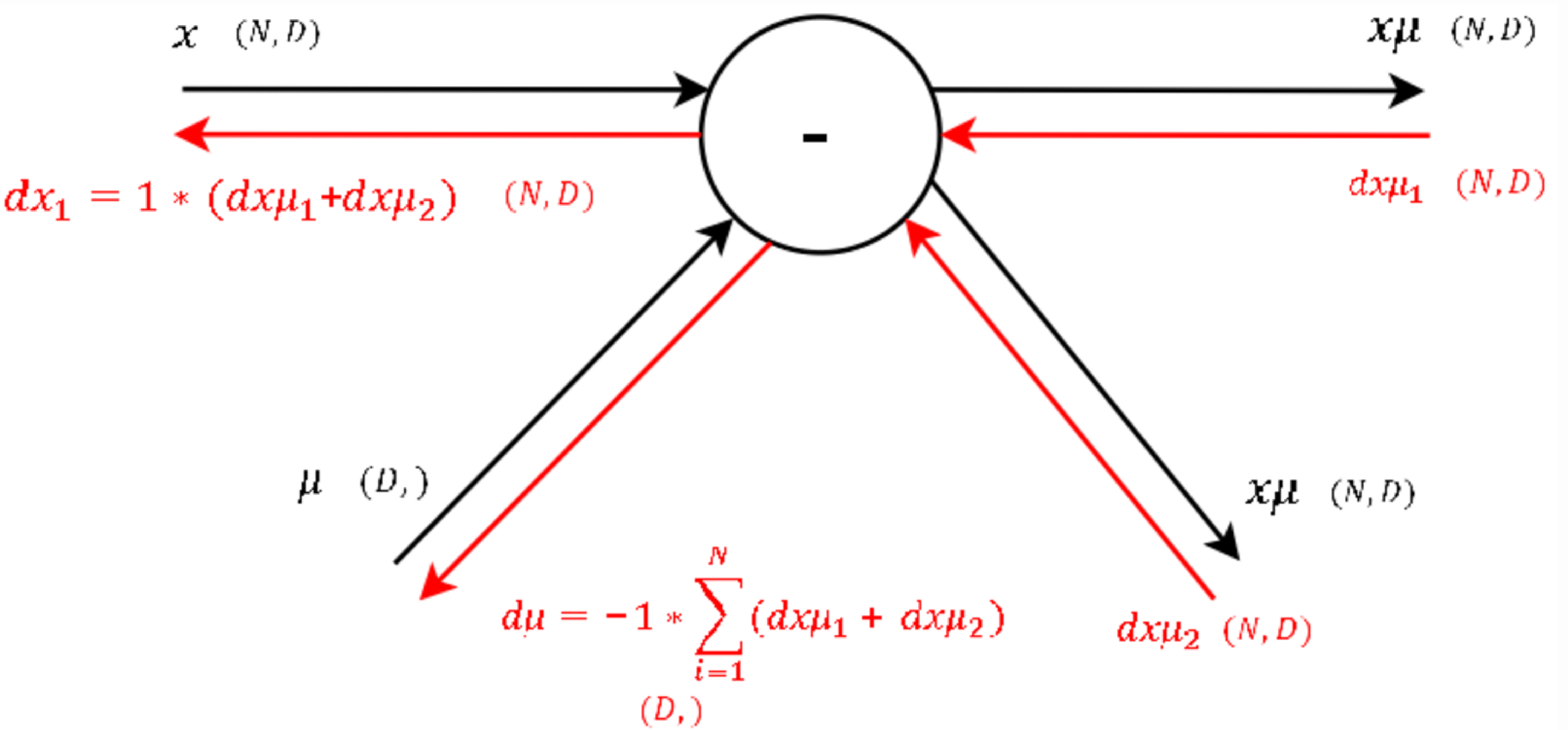
STEP 3 – Square gate



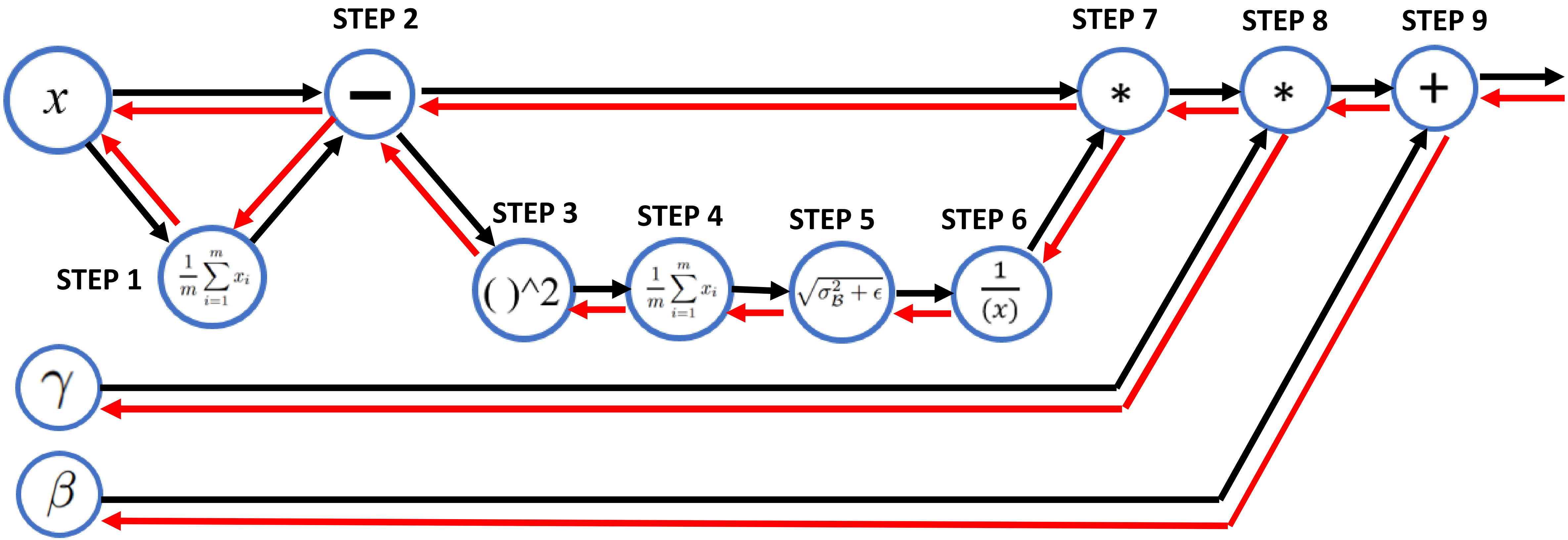
Computational Graph of Batch Normalization



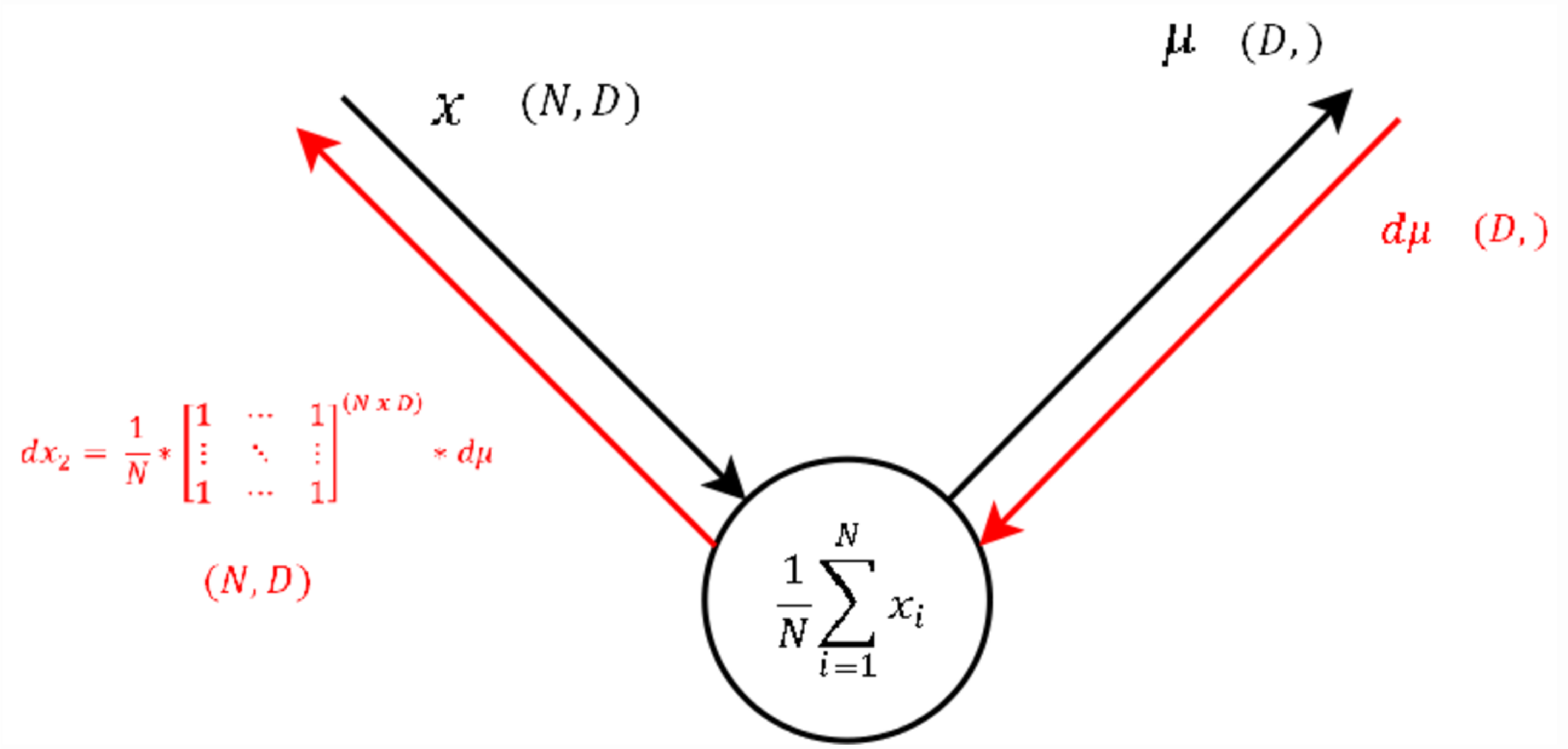
STEP 2 – Subtraction gate



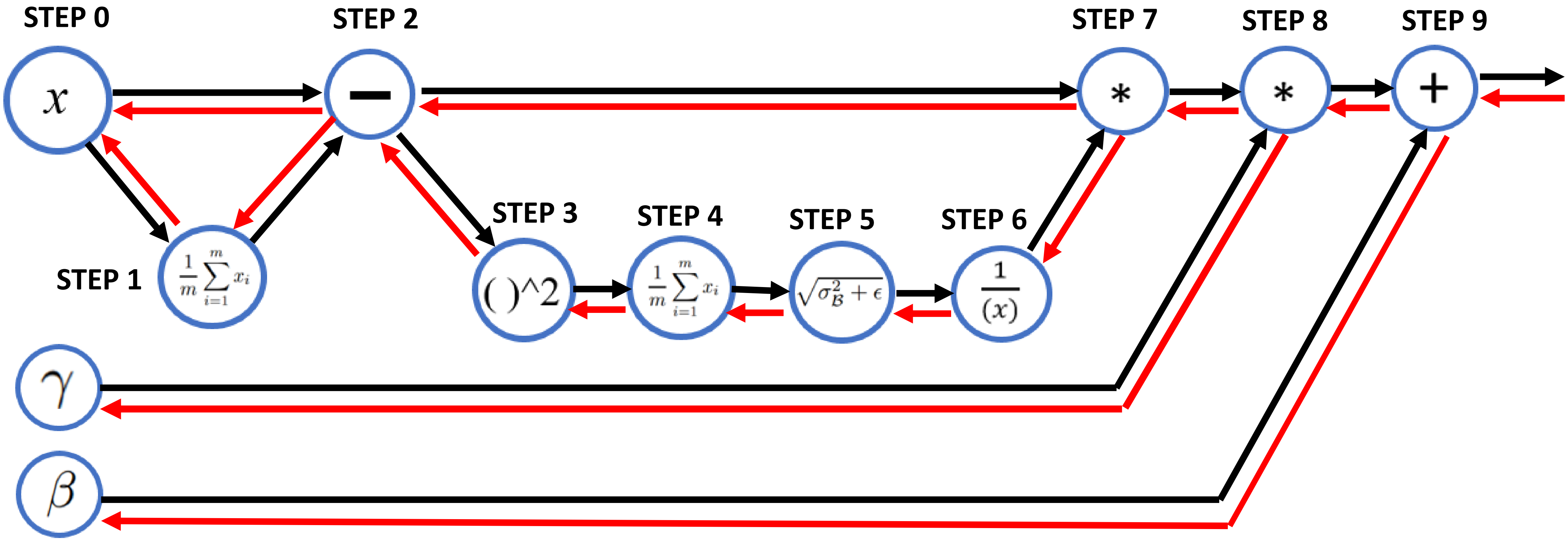
Computational Graph of Batch Normalization



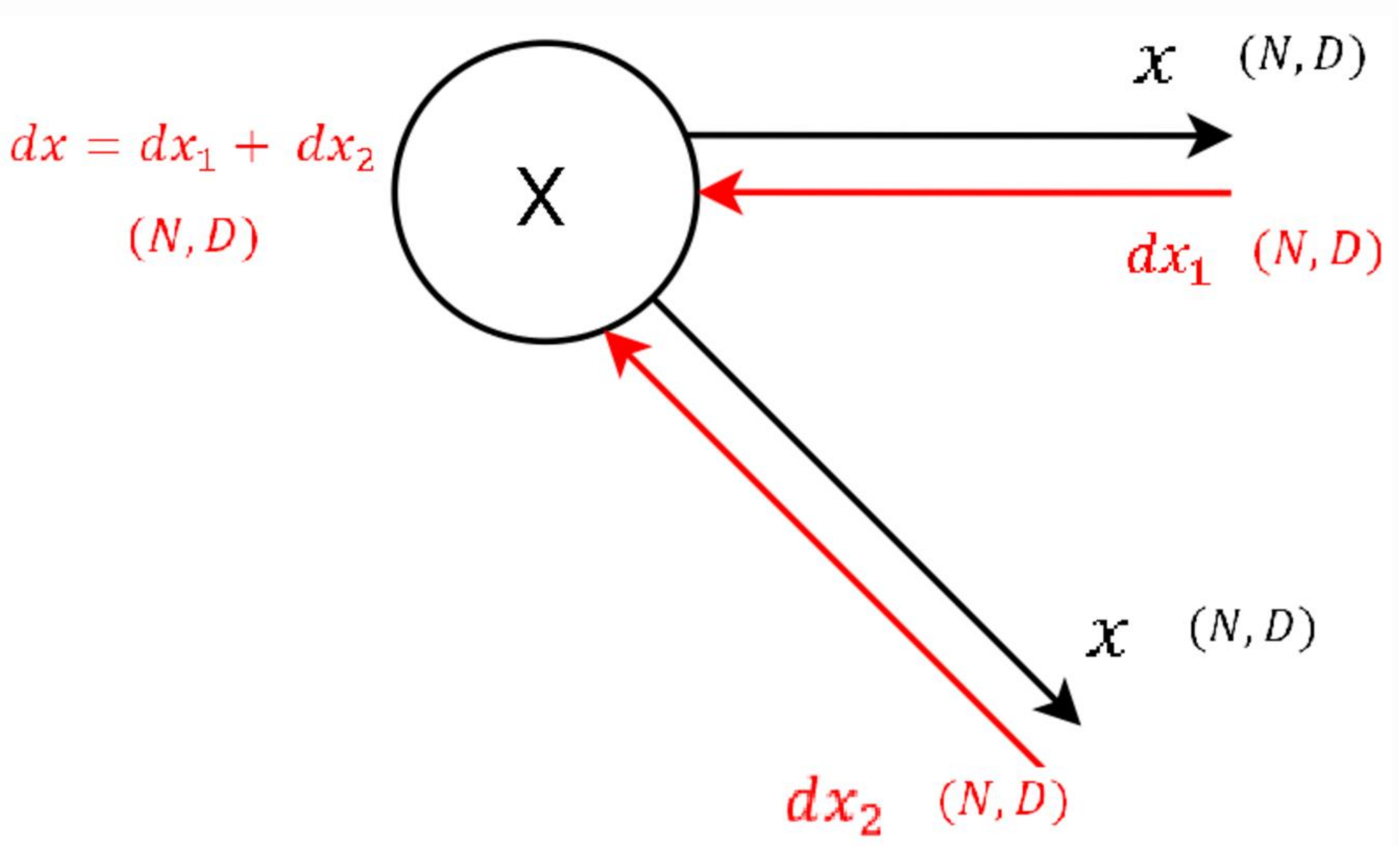
STEP 1 – Mean gate



Computational Graph of Batch Normalization



STEP 0 – Input



Real World Applications

Advantages of Batch Normalization

- Increased learning rate
- Remove Dropout
- Increased accuracy
- Allow use of saturating nonlinearities

Disadvantages of Batch Normalization

- Difficult to estimate mean and standard deviation of input during testing
- Cannot use batch size of 1 during training
- Computational overhead during training

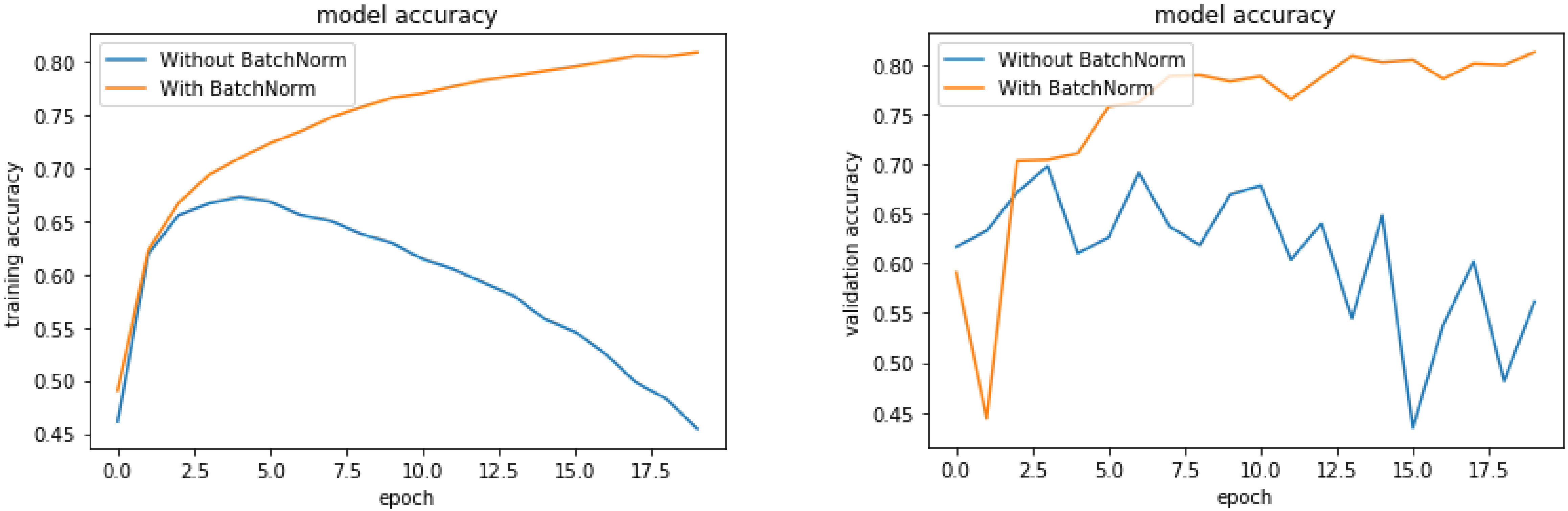
Practical Demo

CIFAR-10

- 4 convolutional layers.
- ReLu activation function.
- Mini-batch size to be 32.
- Batch Normalization added before each activation.
- 50,000 train samples
- 10,00 test samples

```python
def train_model(useBN):
    model = Sequential()

    model.add(Convolution2D(32, 3, 3, border_mode='same',
                            input_shape=X_train.shape[1:]))
    if useBN:
        model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Convolution2D(32, 3, 3))
    if useBN:
        model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Convolution2D(64, 3, 3, border_mode='same'))
    if useBN:
        model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Convolution2D(64, 3, 3))
    if useBN:
        model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512))
    if useBN:
        model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(nb_classes))
    if useBN:
        model.add(BatchNormalization())
    model.add(Activation('softmax'))

    # Let's train the model using RMSprop
    model.compile(loss='categorical_crossentropy',
                  optimizer='rmsprop',
                  metrics=['accuracy'])
    history = model.fit(X_train, Y_train,
              batch_size=batch_size,
              nb_epoch=nb_epoch,
              validation_data=(X_test, Y_test),
              shuffle=True,
              verbose=2)
    return history
```

Learning curve on CIFAR-10



Related Work

- **Recurrent Batch Normalization -** Batch-normalize the hidden-to-hidden transition, thereby reducing internal covariate shift between time steps.
- **Weight Normalization**: A Simple Reparameterization to Accelerate Training of Deep Neural Networks
- **Normalization Propagation:** A Parametric Technique for Removing Internal Covariate Shift in Deep Networks
- **Layer Normalization**

Citations

- http://adsabs.harvard.edu/cgi-bin/nph-ref_query?bibcode=2015arXiv150203167I&refs=CITATIONS&db_key=PRE