ECE 1508S2: Applied Deep Learning

Chapter 3: Advancing Our Toolbox

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Winter 2025

Standardization

If you implement *forward and backward pass of an FNN for MNIST classification from scratch: giving image pixels* to NN *can return huge features*!

Feature

Outputs of hidden layers, i.e., $\mathbf{y}_\ell$

This is a *typical observation* in practice; however, *it does not make trouble only* in *forward pass*: *it can also impact severely the training*, i.e., *backward pass*

The solution to this issue is to *standardize the inputs and features*

Standardization means making variables look the same in *all directions*

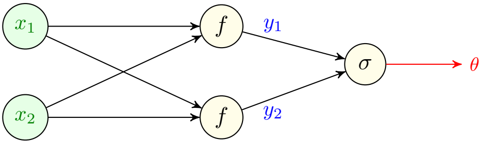
This is done by two particular techniques in practice

- *input standardization*
- *batch normalization*

Let's understand them *through an easy example*

Standardization: *Example*

Consider the following simple NN: *here we have a two-dimensional input, one hidden layer with a two-dimensional feature and a single output*



The inputs could be anything, for instance

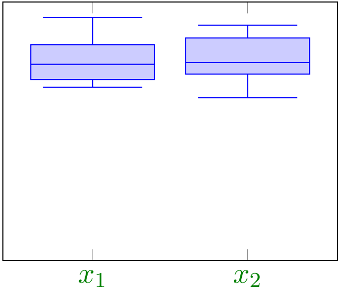
- ***Example A:*** *both are heights in centimeters*
  - ↳ *they are in the same range*
  - ↳ *distributions of variables have close means and variances*
- ***Example B:*** x_1 *is height centimeters and* x_2 *is number of children*
  - ↳ *they are in the widely-different range*
  - ↳ *distributions of variables have strongly-different means and variances*

Standardization: *Example*

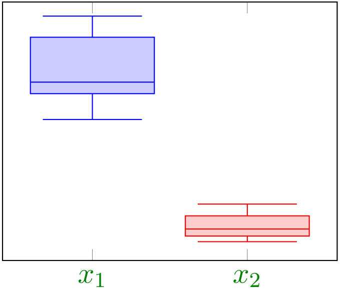
We expect *different behavior in forward pass*:

- *in **Example A** all variables are in the same scale*
- *in **Example B** each variable is in different scale*

Example A



Example B



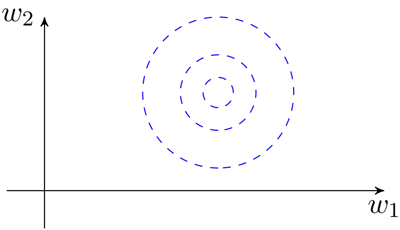
Standardization: *Example*

It also leads to different behavior in backward pass:

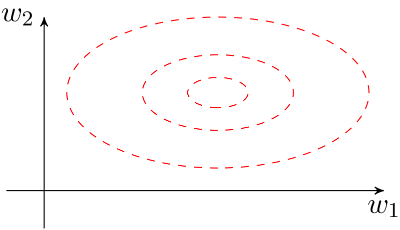
- *in **Example A** empirical risk's curvature is similar in all directions*
- *in **Example B** empirical risk's curvature varies from one direction to another*

For instance, if we assume NN has only two weights to train, the counters of empirical loss can look as below

Example A

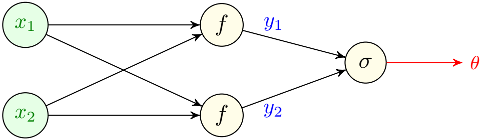


Example B

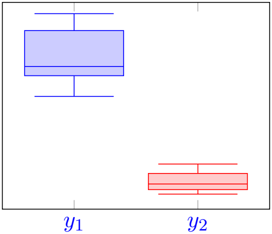


Standardization: *Different Layers*

Such behavior is *not specific* to the *input layer: we could also see the same behavior* at *hidden layers*



For instance in our example, even with properly-scaled inputs, features may evolve differently through training: after multiple iterations we end up with



Standardization via *Normalization*

The solution for *any layer* is to *shift and scale every sample input such that*

it becomes *zero-mean and unit-variance*

Then, we work with the standardized input

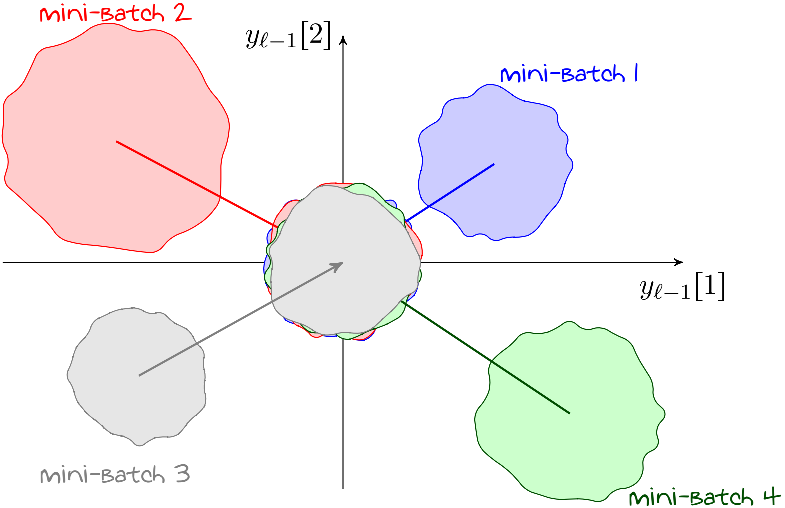
Say we are at layer ℓ: a sample input is $\mathbf{y}_{\ell-1,b}$, so we compute

$$\bar{\mathbf{y}}_{\ell-1,b} = \frac{\mathbf{y}_{\ell-1,b} - \mathbb{E}\left\{\mathbf{y}_{\ell-1}\right\}}{\sqrt{\mathbb{Var}\left\{\mathbf{y}_{\ell-1}\right\}}}$$

and then perform all further operations on $\bar{\mathbf{y}}_{\ell-1,b}$

Visualizing *Normalization*

We can visualize *normalization* as below *for two-dimensional inputs*



Standardization via *Normalization*

$$\bar{\mathbf{y}}_{\ell-1,b} = \frac{\mathbf{y}_{\ell-1,b} - \mathbb{E}\left\{\mathbf{y}_{\ell-1}\right\}}{\sqrt{\mathbb{Var}\left\{\mathbf{y}_{\ell-1}\right\}}}$$

+ *How do we compute mean and variance? We don't know data distribution!*
– Well, as always: *we can approximate them from the dataset*

Say our dataset has B data-points: we approximate mean and variance as

$$\mathbb{E}\left\{\mathbf{y}_{\ell-1}\right\} \approx \frac{1}{B}\sum_{b=1}^{B} \mathbf{y}_{\ell-1,b} \equiv \boldsymbol{\mu}_{\ell-1}$$

$$\mathbb{Var}\left\{\mathbf{y}_{\ell-1}\right\} \approx \frac{1}{B}\sum_{b=1}^{B} \left(\mathbf{y}_{\ell-1,b} - \boldsymbol{\mu}_{\ell-1}\right)^2 \equiv \boldsymbol{\sigma}_{\ell-1}^2$$

Let's check this idea for each layer!

Normalization: *Input Layer*

With input layer, i.e., $\ell = 1$, this approximation works well: *we apply normalization just once and work with normalized data from then on, i.e.,*

$$\bar{\boldsymbol{x}}_b = \frac{\boldsymbol{x}_b - \boldsymbol{\mu}_0}{\boldsymbol{\sigma}_0}$$

for $\boldsymbol{\mu}_0$ and $\boldsymbol{\sigma}_0$ that are computed from the dataset as

$$\boldsymbol{\mu}_0 = \frac{1}{B} \sum_{b=1}^{B} \boldsymbol{x}_b \qquad \boldsymbol{\sigma}_0 = \sqrt{\frac{1}{B} \sum_{b=1}^{B} \left(\boldsymbol{x}_b - \boldsymbol{\mu}_0\right)^2}$$

Let's define the operator $\mathcal{U}\left(\cdot\right)$ as the *standardizer*, *i.e.*,

$$\bar{\boldsymbol{x}}_b = \mathcal{U}\left(\boldsymbol{x}_b | \mathbb{D}\right)$$

where $\mathbb{D}$ is the training dataset

Forward Propagation with *Input Normalization*

```
ForwardProp():
```
1: Initiate with $\mathbf{y}_0 = \mathcal{U}\left(\boldsymbol{x}_b | \mathbb{D}\right)$
2: **for** $\ell = 0, \ldots, L$ **do**
3: Add $y_\ell[0] = 1$ and determine $\mathbf{z}_{\ell+1} = \mathbf{W}_{\ell+1}\mathbf{y}_\ell$ `# forward affine`
4: Determine $\mathbf{y}_{\ell+1} = f_{\ell+1}(\mathbf{z}_{\ell+1})$ `# forward activation`
5: **end for**
6: **for** $\ell = 1, \ldots, L+1$ **do**
7: Return $\mathbf{y}_\ell$ and $\mathbf{z}_\ell$
8: **end for**

+ *Shall we do the same for every layer?*
– Well, we could try, *but we end up with computation complexity close to full-batch training!*

Normalization: *Hidden Layers*

At hidden layers, i.e., $\ell > 1$, *the input depends on the wights and biases of previous layer: for instance, after normalizing input we compute*

$$\mathbf{z}_1 = \mathbf{W}_1 \bar{\mathbf{x}} \rightsquigarrow \mathbf{y}_1 = f_1(\mathbf{z}_1)$$

if we approximate mean and variance with same approach, we should compute

$$\boldsymbol{\mu}_1 = \frac{1}{B} \sum_{b=1}^{B} \mathbf{y}_{1,b} \qquad \boldsymbol{\sigma}_1 = \sqrt{\frac{1}{B} \sum_{b=1}^{B} \left(\mathbf{y}_{1,b} - \boldsymbol{\mu}_1\right)^2}$$

These parameters depend on $\mathbf{W}_1$! This means after each update of weights at the end of each mini-batch, we should repeat this for the entire training dataset!

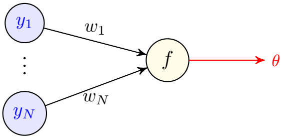
This is not the only issue: *since the normalization of these layers depends on weights, the gradient of loss with respect to weights also changes*

Batch Normalization: *Training with Normalization*

Batch normalization extends the training loop of mini-batch SGD to incorporate also normalization of hidden layers: the idea is simple

- *approximate mean and variance using the mini-batch*
- *compute derivatives by taking normalization into account*

Let's focus first on a *single neuron in a hidden layer*



Say the output neuron has no bias, i.e., θ is given by

$$z = \sum_{n=1}^{N} w_n y_n = \mathbf{w}^\mathsf{T} \mathbf{y} \quad \rightsquigarrow \quad \theta = f(z)$$

Also assume we train via mini-batches with batch size Ω

Recap: *Basic Forward and Backward Pass*

Without batch normalization, we pass forward $\mathbf{y}_\omega$ for every $\omega = 1, \ldots, \Omega$

- *by first computing the affine transform*

$$z_\omega = \mathbf{w}^\mathsf{T} \mathbf{y}_\omega$$

- *then activating as* $\theta_\omega = f(z_\omega)$

Once we get to the output: *we backpropagate by*

- *first computing derivative with respect to* z_ω, *i.e.*,

$$\frac{\mathrm{d}}{\mathrm{d} z_\omega} \mathcal{L} = \left(\frac{\mathrm{d}}{\mathrm{d} \theta_\omega} \mathcal{L} \right) \dot{f}(z_\omega)$$

- *and then tracking back to* $\mathbf{y}_\omega$, *i.e.*,

$$\nabla_{\mathbf{y}_\omega} \mathcal{L} = \mathbf{w} \left(\frac{\mathrm{d}}{\mathrm{d} z_\omega} \mathcal{L} \right)$$

Batch Normalization: *Forward Pass*

With batch normalization *in forward pass, we wait till the whole mini-batch is over: wait till we have $\mathbf{y}_\omega$ for $\omega = 1, \dots, \Omega$. We then*

- *approximate mean and variance as*

$$\boldsymbol{\mu} = \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} \mathbf{y}_\omega \qquad \boldsymbol{\sigma} = \sqrt{\frac{1}{\Omega} \sum_{\omega=1}^{\Omega} \left(\mathbf{y}_\omega - \boldsymbol{\mu}\right)^2}$$

- *normalize $\mathbf{y}_\omega$ for $\omega = 1, \dots, \Omega$ as*

$$\mathbf{u}_\omega = \frac{\mathbf{y}_\omega - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

$\rightarrow$ *scale and shift to a common place*

$$\bar{\mathbf{y}}_\omega = \boldsymbol{\gamma} \odot \mathbf{u}_\omega + \boldsymbol{\beta}$$

- *compute the affine transforms $z_\omega = \mathbf{w}^\mathsf{T} \bar{\mathbf{y}}_\omega$ and activate as $\theta_\omega = f\left(z_\omega\right)$*

Batch Normalization: *Learnable Shift and Scale*

+ *Why do we scale and shift after normalization?*
– This is not guaranteed that center with unit variance is the best place: *we can learn the best place through training!*

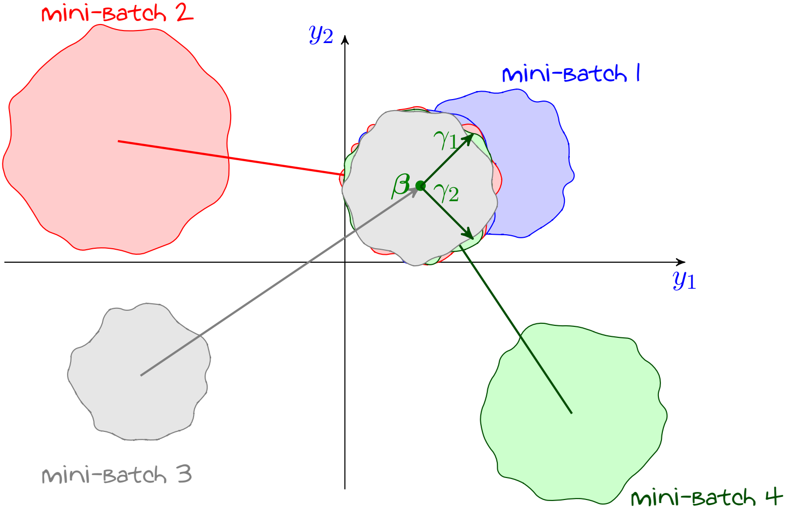
Depending on activation, it might be better *to center all features at another place with some different variances*

- *It seems hard to engineer this point and the variance*
- *We hence introduce a general point* $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}^N$
  - ↳ *We treat these vectors as learnable parameters*
  - ↳ *We update them in addition to weight matrices* $\mathbf{W}_\ell$ *in backward pass*

We denote this operation by

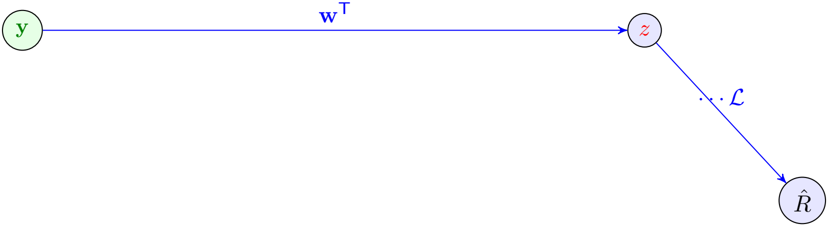
$$\mathcal{B}_\mathcal{N}\left(\mathbf{u}|\boldsymbol{\gamma},\boldsymbol{\beta}\right) = \boldsymbol{\gamma} \odot \mathbf{u} + \boldsymbol{\beta}$$

Batch Normalization: *Learnable Shift and Scale*



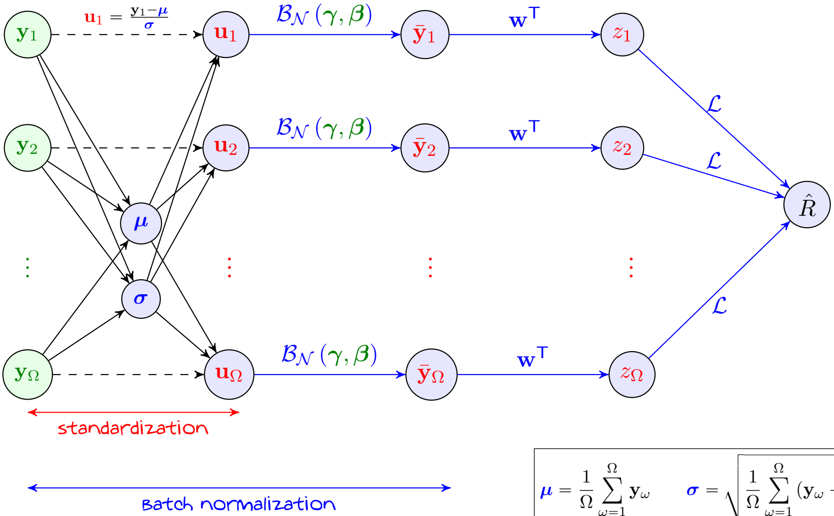
Batch Normalization: *Computation Graph*

In good old days without batch-normalization, we had



But, it gets more complicated now!

Batch Normalization: *Computation Graph*



$$\boldsymbol{\mu} = \frac{1}{\Omega}\sum_{\omega=1}^{\Omega} \mathbf{y}_\omega \qquad \boldsymbol{\sigma} = \sqrt{\frac{1}{\Omega}\sum_{\omega=1}^{\Omega} (\mathbf{y}_\omega - \boldsymbol{\mu})^2}$$

Batch Normalization: *Backpropagation*

Now assume that *forward pass is over for the complete mini-batch*

we could easily get back to normalized variables, i.e., $\bar{\mathbf{y}}_\omega$

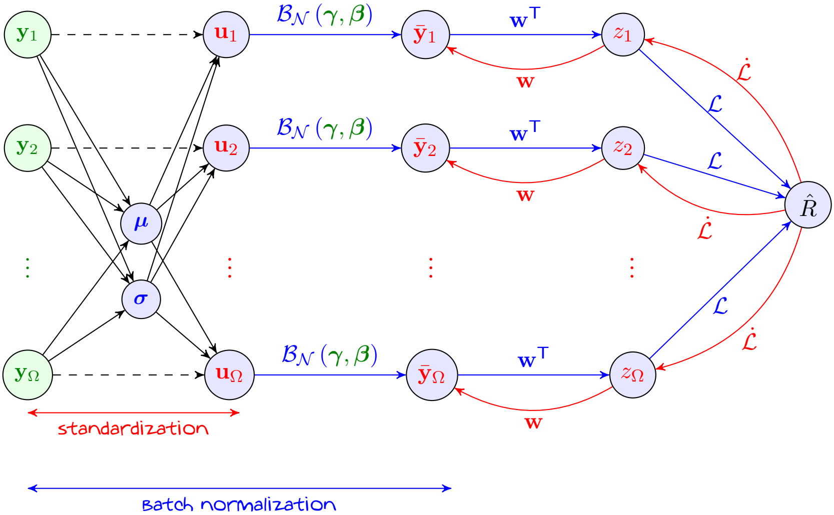
First, we note that *our empirical risk reads*

$$\hat{R} = \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} \underbrace{\mathcal{L}\left(\theta_\omega, v_\omega\right)}_{\hat{R}_\omega} = \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} \hat{R}_\omega$$

- *We compute* $\nabla_{z_\omega} \hat{R}_\omega$ *using* $\dot{\mathcal{L}}\left(\cdot\right)$ *and* $\dot{f}\left(\cdot\right)$
- *We compute* $\nabla_{\bar{\mathbf{y}}_\omega} \hat{R}_\omega$ *from* $\nabla_{z_\omega} \hat{R}_\omega$
  - ↳ *We just need to apply standard backward pass of linear transforms*

$$\nabla_{\bar{\mathbf{y}}_\omega} \hat{R}_\omega = \left(\nabla_{z_\omega} \hat{R}_\omega\right) \mathbf{w} = \left(\frac{\mathrm{d}}{\mathrm{d} z_\omega} \hat{R}_\omega\right) \mathbf{w}$$

Batch Normalization: *Backpropagation*



Batch Normalization: *Backpropagation*

By now, *we have* $\nabla_{\bar{\mathbf{y}}_\omega} \hat{R}_\omega$ *for* $\omega = 1, \ldots, \Omega$; next, we move backward

from scaled and shifted variables to standard ones

$\mathcal{B}_\mathcal{N}(\boldsymbol{\gamma}, \boldsymbol{\beta})$ *scales and shifts entry-wise*

$$\begin{bmatrix} \bar{y}_{1,\omega} \\ \vdots \\ \bar{y}_{N,\omega} \end{bmatrix} = \begin{bmatrix} \gamma_1 u_{1,\omega} + \beta_1 \\ \vdots \\ \gamma_N u_{N,\omega} + \beta_N \end{bmatrix}$$

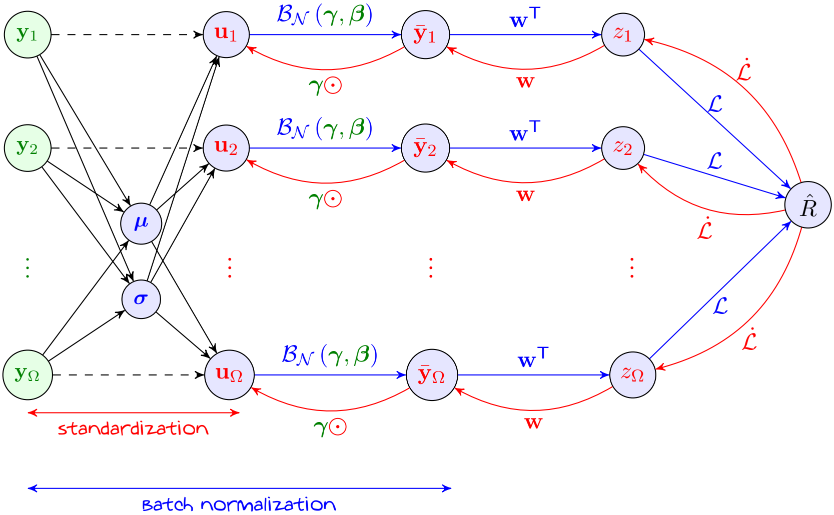
So, we can say

$$\frac{\partial}{\partial u_{i,\omega}} \hat{R}_\omega = \gamma_i \frac{\partial}{\partial \bar{y}_{i,\omega}} \hat{R}_\omega$$

The backward pass is hence for $\omega = 1, \ldots, \Omega$ *is*

$$\nabla_{\mathbf{u}_\omega} \hat{R}_\omega = \boldsymbol{\gamma} \odot \nabla_{\bar{\mathbf{y}}_\omega} \hat{R}_\omega$$

Batch Normalization: *Backpropagation*



Batch Normalization: *Backpropagation*

At this point, we can also *compute the gradients with respect to* γ *and* β

> *For* $\omega = 1, \ldots, \Omega$, *we have*
>
> $$\nabla_{\boldsymbol{\gamma}} \hat{R}_\omega = \mathbf{u}_\omega \odot \nabla_{\bar{\mathbf{y}}_\omega} \hat{R}_\omega$$
> $$\nabla_{\boldsymbol{\beta}} \hat{R}_\omega = \nabla_{\bar{\mathbf{y}}_\omega} \hat{R}_\omega$$

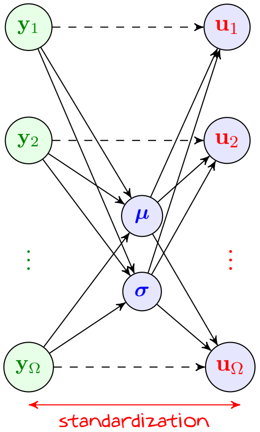
In our optimizer, we then update γ and β using $\nabla_{\boldsymbol{\gamma}} \hat{R}_\omega$ and $\nabla_{\boldsymbol{\beta}} \hat{R}_\omega$: *for instance with standard SGD we have*

$$\boldsymbol{\gamma}^{(t+1)} = \boldsymbol{\gamma}^{(t)} - \frac{\eta}{\Omega} \sum_{\omega=1}^{\Omega} \nabla_{\boldsymbol{\gamma}} \hat{R}_\omega$$

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \frac{\eta}{\Omega} \sum_{\omega=1}^{\Omega} \nabla_{\boldsymbol{\beta}} \hat{R}_\omega$$

Batch Normalization: *Backpropagation*

The most challenging block is standardization: we open expand everything



Let's write again forward pass

$$\mathbf{u}_\omega = \frac{\mathbf{y}_\omega - \boldsymbol{\mu}\left(\mathbf{y}_1, \ldots, \mathbf{y}_\Omega\right)}{\boldsymbol{\sigma}\left(\mathbf{y}_1, \ldots, \mathbf{y}_\Omega\right)}$$

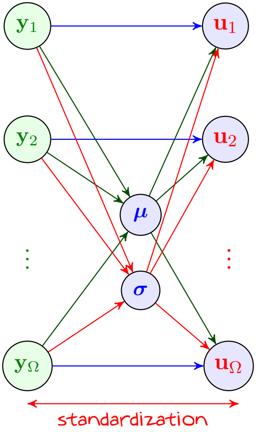
Here, $\hat{R}_\omega$ depends on all branches, i.e.,

$$\hat{R}_\omega\left(\mathbf{y}_1, \ldots, \mathbf{y}_\Omega\right)$$

We know how to use chain rule

$$\begin{aligned}\nabla_{\mathbf{y}_\tau} \hat{R}_\omega &= \sum_{\lambda=1}^{\Omega} \nabla_{\mathbf{u}_\lambda} \hat{R}_\omega \circ \nabla_{\mathbf{y}_\tau} \mathbf{u}_\lambda \\ &= \nabla_{\mathbf{u}_\omega} \hat{R}_\omega \circ \nabla_{\mathbf{y}_\tau} \mathbf{u}_\omega\end{aligned}$$

Batch Normalization: *Backpropagation*



All operations are entry-wise, i.e.,

$$u_{i,\omega} = \frac{y_{i,\omega} - \mu_i\left(y_{i,1}, \ldots, y_{i,\Omega}\right)}{\sigma_i\left(y_{i,1}, \ldots, y_{i,\Omega}\right)}$$

So, it's safe to think of $\mathbf{u}_\omega$ *and* $\mathbf{y}_\tau$ *as scalars*

$$\frac{\partial \hat{R}_\omega}{\partial y_{i,\tau}} = \frac{\partial \hat{R}_\omega}{\partial u_{i,\omega}} \frac{\partial u_{i,\omega}}{\partial y_{i,\tau}}$$

If we set $\tau = \omega$*; then, we have*

$$\begin{aligned} \frac{\partial u_{i,\omega}}{\partial y_{i,\omega}} &= \frac{1}{\sigma_i} + \frac{\partial u_{i,\omega}}{\partial \mu_i} \frac{\partial \mu_i}{\partial y_{i,\omega}} + \frac{\partial u_{i,\omega}}{\partial \sigma_i} \frac{\partial \sigma_i}{\partial y_{i,\omega}} \\ &= \frac{1}{\sigma_i} - \frac{1}{\sigma_i} \frac{\partial \mu_i}{\partial y_{i,\omega}} - \frac{y_{i,\omega} - \mu_i}{\sigma_i^2} \frac{\partial \sigma_i}{\partial y_{i,\omega}} \end{aligned}$$

Batch Normalization: *Backpropagation*

Now, let us determine partial derivatives

$$\mu_i = \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} y_{i,\omega}$$

So, we can write

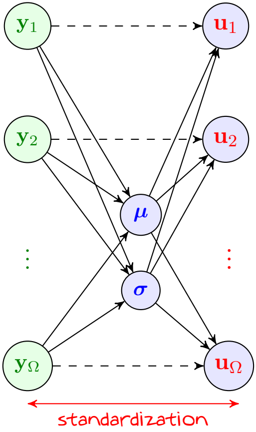
$$\frac{\partial \mu_i}{\partial y_{i,\omega}} = \frac{1}{\Omega}$$

$$\sigma_i = \sqrt{\frac{1}{\Omega} \sum_{\omega=1}^{\Omega} (y_{i,\omega} - \mu_i)^2}$$

Here, we should do it in two steps

$$\begin{aligned} \frac{\partial \sigma_i}{\partial y_{i,\omega}} &= \frac{1}{2\sigma_i} \left(\frac{2}{\Omega} (y_{i,\omega} - \mu_i) \right) + \frac{\partial \sigma_i}{\partial \mu_i} \frac{\partial \mu_i}{\partial y_{i,\omega}} \\ &= \frac{1}{\Omega \sigma_i} (y_{i,\omega} - \mu_i) + \underbrace{\frac{\partial \sigma_i}{\partial \mu_i}}_{0} \frac{\partial \mu_i}{\partial y_{i,\omega}} \\ &= \frac{1}{\Omega \sigma_i} (y_{i,\omega} - \mu_i) \end{aligned}$$

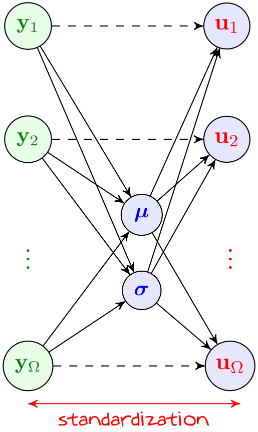
Batch Normalization: *Backpropagation*



Let's replace for the case $\tau = \omega$

$$\frac{\partial u_{i,\tau}}{\partial y_{i,\omega}} = \frac{1}{\sigma_i} + \frac{\partial u_{i,\omega}}{\partial \mu_i}\frac{\partial \mu_i}{\partial y_{i,\omega}} + \frac{\partial u_{i,\omega}}{\partial \sigma_i}\frac{\partial \sigma_i}{\partial y_{i,\omega}}$$
$$= \frac{1}{\sigma_i} - \frac{1}{\sigma_i}\frac{\partial \mu_i}{\partial y_{i,\omega}} - \frac{y_{i,\omega} - \mu_i}{\sigma_i^2}\frac{\partial \sigma_i}{\partial y_{i,\omega}}$$
$$= \frac{1}{\sigma_i} - \frac{1}{\Omega\sigma_i} - \frac{\left(y_{i,\omega} - \mu_i\right)^2}{\Omega\sigma_i^3}$$

Batch Normalization: *Backpropagation*



$$u_{i,\omega} = \frac{y_{i,\omega} - \mu_i\left(y_{i,1}, \ldots, y_{i,\Omega}\right)}{\sigma_i\left(y_{i,1}, \ldots, y_{i,\Omega}\right)}$$

If we set $\tau \neq \omega$; then, we have

$$\begin{aligned}\frac{\partial u_{i,\omega}}{\partial y_{i,\tau}} &= 0 + \frac{\partial u_{i,\omega}}{\partial \mu_i}\frac{\partial \mu_i}{\partial y_{i,\tau}} + \frac{\partial u_{i,\omega}}{\partial \sigma_i}\frac{\partial \sigma_i}{\partial y_{i,\tau}} \\ &= -\frac{1}{\sigma_i}\frac{\partial \mu_i}{\partial y_{i,\tau}} - \frac{y_{i,\omega} - \mu_i}{\sigma_i^2}\frac{\partial \sigma_i}{\partial y_{i,\tau}} \\ &= -\frac{1}{\Omega\sigma_i} - \frac{\left(y_{i,\tau} - \mu_i\right)^2}{\Omega\sigma_i^3}\end{aligned}$$

Everything as before

↳ *Just the first term drops*

Batch Normalization: *Backpropagation*

The last piece of derivation is to relate these partial derivatives to gradient of empirical risk determined over the whole mini-batch

$$\nabla_{\mathbf{y}_\tau} \hat{R} = \nabla_{\mathbf{y}_\tau} \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} \hat{R}_\omega = \frac{1}{\Omega} \sum_{\omega=1}^{\Omega} \nabla_{\mathbf{y}_\tau} \hat{R}_\omega$$

From above derivation we have

$$\nabla_{\mathbf{y}_\tau} \hat{R}_\omega = \frac{\mathbb{1}\left\{\tau = \omega\right\}}{\boldsymbol{\sigma}} - \frac{1}{\Omega \boldsymbol{\sigma}} - \frac{\left(\mathbf{y}_\tau - \boldsymbol{\mu}\right)^2}{\Omega \boldsymbol{\sigma}^3}$$

with all operations being entry-wise! We then derive $\nabla_{\mathbf{W}_\ell} \hat{R}$ exactly as in sample-wise backpropagation

Suggestion

Try to write the complete backpropagation with batch normalization

Batch Normalization: *Testing*

+ *Say we trained our NN! Now how we test it for single new point? We do not have any mini-batch anymore!*
– Good point! In practice, we use *moving average*

Throughout training, we compute moving averages $\bar{\boldsymbol{\mu}}_\ell$ and $\bar{\boldsymbol{\sigma}}_\ell$

At each layer ℓ, we start with some initial $\bar{\boldsymbol{\mu}}_\ell$ and $\bar{\boldsymbol{\sigma}}_\ell$ and compute

$$\bar{\boldsymbol{\mu}}_\ell = \alpha \bar{\boldsymbol{\mu}}_\ell + (1 - \alpha)\, \boldsymbol{\mu}_\ell$$
$$\bar{\boldsymbol{\sigma}}_\ell = \alpha \bar{\boldsymbol{\sigma}}_\ell + (1 - \alpha)\, \boldsymbol{\sigma}_\ell$$

after each iteration for some $0 < \alpha < 1$: typically close to 1

We use these values for normalization after we are over with training

Batch Normalization: *Final Points*

Few points that you may observe *in implementation of batch normalization*

- *We usually perturb variance with a small constant for numerical stability*

$$\boldsymbol{\sigma} = \sqrt{\frac{1}{\Omega}\sum_{\omega=1}^{\Omega}\left(\mathbf{y}_\omega - \boldsymbol{\mu}\right)^2 + \epsilon}$$

- *We could normalize before or after activation*
  - ↳ *Here, we did it after activation*
  - ↳ *In the original proposal is was before activation*
  - ↳ *In general, it is not fully known which one is better*
    - ⇝ *We may try both and see which one gives better result*
- *With batch-normalization, we should distinguish training from evaluation*
  - ↳ *We use* `model.train()` *for training*
    - ⇝ *Here, we update the moving averages*
  - ↳ *We use* `model.eval()` *for evaluation*
    - ⇝ *Here, we only use the moving averages*