Deep learning

5.5. Parameter initialization

François Fleuret

https://fleuret.org/dlc/
Consider the gradient estimation for a standard MLP, as seen in 3.6. “Back-propagation”:

**Forward pass**

\[ x^{(0)} = x, \ \forall l = 1, \ldots, L, \]

\[ s^{(l)} = w^{(l)}x^{(l-1)} + b^{(l)} \]

\[ x^{(l)} = \sigma(s^{(l)}) \]

**Backward pass**

\[
\begin{cases}
\left[ \frac{\partial \ell}{\partial x^{(L)}} \right] & \text{from the definition of } \ell \\
\text{if } l < L, \left[ \frac{\partial \ell}{\partial x^{(l)}} \right] = (w^{(l+1)})^\top \left[ \frac{\partial \ell}{\partial s^{(l+1)}} \right] \\
\left[ \frac{\partial \ell}{\partial w^{(l)}} \right] = \left[ \frac{\partial \ell}{\partial s^{(l)}} \right] \left( x^{(l-1)} \right)^\top \\
\left[ \frac{\partial \ell}{\partial b^{(l)}} \right] = \left[ \frac{\partial \ell}{\partial s^{(l)}} \right].
\end{cases}
\]

\[
\left[ \frac{\partial \ell}{\partial w^{(l)}} \right] = \left[ \frac{\partial \ell}{\partial s^{(l)}} \right] \left( x^{(l-1)} \right)^\top \\
\left[ \frac{\partial \ell}{\partial b^{(l)}} \right] = \left[ \frac{\partial \ell}{\partial s^{(l)}} \right].
\]
We have

\[
\left[ \frac{\partial \ell}{\partial x^{(l)}} \right] = (w^{(l+1)})^\top \left( \sigma'(s^{(l)}) \odot \left[ \frac{\partial \ell}{\partial x^{(l+1)}} \right] \right).
\]

so the gradient “vanishes” exponentially with the depth if the \(w\)s are ill-conditioned or the activations are in the saturating domain of \(\sigma\).
We have
\[
\left[ \frac{\partial \ell}{\partial x^{(l)}} \right] = \left( w^{(l+1)} \right)^\top \left( \sigma'(s^{(l)}) \odot \left[ \frac{\partial \ell}{\partial x^{(l+1)}} \right] \right).
\]

so the gradient “vanishes” exponentially with the depth if the \( w \)s are ill-conditioned or the activations are in the saturating domain of \( \sigma \).

(Glorot and Bengio, 2010)
Weight initialization
The design of the weight initialization aims at controlling

$$
\nabla \left( \frac{\partial \ell}{\partial w_{i,j}^{(l)}} \right) \quad \text{and} \quad \nabla \left( \frac{\partial \ell}{\partial b_i^{(l)}} \right)
$$

so that weights evolve at the same rate across layers during training, and no layer reaches a saturation behavior before others.
We will use that, if $A$ and $B$ are independent

$$\text{V}(AB) = \text{V}(A) \text{V}(B) + \text{V}(A) \text{E}(B)^2 + \text{V}(B) \text{E}(A)^2.$$
We will use that, if $A$ and $B$ are independent

$$\text{Var}(AB) = \text{Var}(A) \text{Var}(B) + \text{Var}(A) \mathbb{E}(B)^2 + \text{Var}(B) \mathbb{E}(A)^2.$$ 

Notation in the coming slides will drop indexes when variances are identical for all activations or parameters in a layer.
In a standard layer

\[ x^{(l)}_i = \sigma \left( \sum_{j=1}^{N_{l-1}} w_{i,j} x^{(l-1)}_j + b_{i}^{(l)} \right) \]

where \( N_l \) is the number of units in layer \( l \), and \( \sigma \) is the activation function.
In a standard layer

\[ x_i^{(l)} = \sigma \left( \sum_{j=1}^{N_{l-1}} w_{i,j} x_j^{(l-1)} + b_i^{(l)} \right) \]

where \( N_l \) is the number of units in layer \( l \), and \( \sigma \) is the activation function.

Assuming \( \sigma'(0) = 1 \), and we are in the linear regime

\[ x_i^{(l)} \approx \sum_{j=1}^{N_{l-1}} w_{i,j} x_j^{(l-1)} + b_i^{(l)} \]
In a standard layer

\[ x^{(l)}_i = \sigma \left( \sum_{j=1}^{N_{l-1}} w^{(l)}_{i,j} x^{(l-1)}_j + b^{(l)}_i \right) \]

where \( N_l \) is the number of units in layer \( l \), and \( \sigma \) is the activation function.

Assuming \( \sigma'(0) = 1 \), and we are in the linear regime

\[ x^{(l)}_i \approx \sum_{j=1}^{N_{l-1}} w^{(l)}_{i,j} x^{(l-1)}_j + b^{(l)}_i. \]

From which, if both the \( w^{(l)} \)s and \( x^{(l-1)} \)s are centered, and biases set to zero:

\[ \mathbb{V} \left( x^{(l)}_i \right) \approx \mathbb{V} \left( \sum_{j=1}^{N_{l-1}} w^{(l)}_{i,j} x^{(l-1)}_j \right) \]

\[ = \sum_{j=1}^{N_{l-1}} \mathbb{V} \left( w^{(l)}_{i,j} \right) \mathbb{V} \left( x^{(l-1)}_j \right) \]

and the \( x^{(l)} \)s are centered.
So if the $w_{i,j}^{(l)}$ are sampled i.i.d in each layer, and all the activations have same variance, then

$$
V(x_i^{(l)}) \simeq \sum_{j=1}^{N_{l-1}} V(w_{i,j}^{(l)}) V(x_j^{(l-1)})
$$

$$
= N_{l-1} V(w^{(l)}) V(x^{(l-1)}).
$$
So if the $w_{i,j}^{(l)}$ are sampled i.i.d in each layer, and all the activations have same variance, then

$$
V(x_i^{(l)}) \simeq \sum_{j=1}^{N_{l-1}} V(w_{i,j}^{(l)}) V(x_j^{(l-1)})
= N_{l-1} V(w^{(l)}) V(x^{(l-1)}).
$$

So we have for the variance of the activations:

$$
V(x^{(l)}) \simeq V(x^{(0)}) \prod_{q=1}^{l} N_{q-1} V(w^{(q)}),
$$

which leads to a first type of initialization to ensure

$$
V(w^{(l)}) = \frac{1}{N_{l-1}}.
$$
The standard PyTorch weight initialization for a linear layer

\[ f : \mathbb{R}^N \rightarrow \mathbb{R}^M \]

is

\[ w_{i,j} \sim \mathcal{U} \left[ -\frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}} \right] \]

hence

\[ \mathbb{V}(w) = \frac{1}{3N}. \]

```python
>>> f = nn.Linear(5, 100000)
>>> f.weight.mean()
tensor(0.0007, grad_fn=<MeanBackward0>)
>>> f.weight.var()
tensor(0.0667, grad_fn=<VarBackward0>)
>>> torch.empty(1000000).uniform_(-1/math.sqrt(5), 1/math.sqrt(5)).var()
tensor(0.0667)
>>> 1./15.
0.06666666666666667
```
We can look at the variance of the gradient w.r.t. the activations. Since

\[
\frac{\partial \ell}{\partial x_i^{(l)}} \simeq \sum_{h=1}^{N_{l+1}} \frac{\partial \ell}{\partial x_h^{(l+1)}} w_{h,i}^{(l+1)}
\]

we get

\[
V\left(\frac{\partial \ell}{\partial x_i^{(l)}}\right) \simeq N_{l+1} V\left(\frac{\partial \ell}{\partial x_h^{(l+1)}}\right) V\left(w^{(l+1)}\right).
\]
We can look at the variance of the gradient w.r.t. the activations. Since
\[
\frac{\partial \ell}{\partial x_i^{(l)}} \simeq \sum_{h=1}^{N_{l+1}} \frac{\partial \ell}{\partial x_h^{(l+1)}} w_{h,i}^{(l+1)}
\]
we get
\[
V\left( \frac{\partial \ell}{\partial x_i^{(l)}} \right) \simeq N_{l+1} V\left( \frac{\partial \ell}{\partial x_i^{(l+1)}} \right) V\left( w^{(l+1)} \right).
\]
So we have for the variance of the gradient w.r.t. the activations:
\[
V\left( \frac{\partial \ell}{\partial x_i^{(l)}} \right) \simeq V\left( \frac{\partial \ell}{\partial x_i^{(l)}} \right) \prod_{q=l+1}^{L} N_q V\left( w^{(q)} \right).
\]
Since
\[ x_i^{(l)} \approx \sum_{j=1}^{N_{i-1}} w_{i,j} x_j^{(l-1)} + b_i^{(l)} \]
we have
\[ \frac{\partial \ell}{\partial w_{i,j}} \approx \frac{\partial \ell}{\partial x_i^{(l)}} x_j^{(l-1)} \]
Since
\[
x_i^{(l)} \simeq \sum_{j=1}^{N_l-1} w_{i,j} x_j^{(l-1)} + b_i^{(l)}
\]
we have
\[
\frac{\partial \ell}{\partial w_{i,j}^{(l)}} \simeq \frac{\partial \ell}{\partial x_i^{(l)}} x_j^{(l-1)}
\]
and we get the variance of the gradient w.r.t. the weights
\[
V\left( \frac{\partial \ell}{\partial w^{(l)}} \right) \simeq V\left( \frac{\partial \ell}{\partial x^{(l)}} \right) V\left( x^{(l-1)} \right)
\]
\[
= V\left( \frac{\partial \ell}{\partial x^{(L)}} \right) \left( \prod_{q=l+1}^{L} N_q V\left( w^{(q)} \right) \right) V\left( x^{(0)} \right) \left( \prod_{q=1}^{l} N_{q-1} V\left( w^{(q)} \right) \right)
\]
\[
= \frac{1}{N_l} N_0 \left( \prod_{q=1}^{L} N_q V\left( w^{(q)} \right) \right) V\left( x^{(0)} \right) V\left( \frac{\partial \ell}{\partial x^{(L)}} \right).
\]

Does not depend on $l$. 

François Fleuret

Deep learning / 5.5. Parameter initialization
Similarly, since

\[ x_i^{(l)} \simeq \sum_{j=1}^{N_{l-1}} w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)} \]

we have

\[ \frac{\partial \ell}{\partial b_i^{(l)}} \simeq \frac{\partial \ell}{\partial x_i^{(l)}} \]
Similarly, since

\[ x_i^{(l)} \simeq \sum_{j=1}^{N_{l-1}} w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)} \]

we have

\[ \frac{\partial \ell}{\partial b_i^{(l)}} \simeq \frac{\partial \ell}{\partial x_i^{(l)}} \]

so we get the variance of the gradient w.r.t. the biases

\[ \text{V} \left( \frac{\partial \ell}{\partial b^{(l)}} \right) \simeq \text{V} \left( \frac{\partial \ell}{\partial x^{(l)}} \right) . \]
Finally, since the ratio of the variance of the gradients w.r.t. weights in two different layers does not depend on the weights, there is no exponential behavior to mitigate.

To control the variance of activations, we need

$$\mathbb{V}(w^{(l)}) = \frac{1}{N_{l-1}},$$

and to control the variance of the gradient w.r.t. activations, and through it the variance of the gradient w.r.t. the biases

$$\mathbb{V}(w^{(l)}) = \frac{1}{N_{l}}.$$
Finally, since the ratio of the variance of the gradients w.r.t. weights in two different layers does not depend on the weights, there is no exponential behavior to mitigate.

To control the variance of activations, we need

\[ \mathbb{V}(w^{(l)}) = \frac{1}{N_{l-1}}, \]

and to control the variance of the gradient w.r.t. activations, and through it the variance of the gradient w.r.t. the biases

\[ \mathbb{V}(w^{(l)}) = \frac{1}{N_l}. \]

From which we get as a compromise the “Xavier initialization”

\[ \mathbb{V}(w^{(l)}) = \frac{1}{\frac{N_{l-1} + N_l}{2}} = \frac{2}{N_{l-1} + N_l}. \]

(Glorot and Bengio, 2010)
In `torch/nn/init.py`

```python
def xavier_normal_(tensor, gain = 1):
    fan_in, fan_out = _calculate_fan_in_and_fan_out(tensor)
    std = gain * math.sqrt(2.0 / (fan_in + fan_out))
    with torch.no_grad():
        return tensor.normal_(0, std)
```
4.2.2 Gradient Propagation Study

To empirically validate the above theoretical ideas, we have plotted some normalized histograms of activation values, weight gradients and of the back-propagated gradients at initialization with the two different initialization methods. The results displayed (Figures 6, 7 and 8) are from experiments on Shapeset-3×2, but qualitatively similar results were obtained with the other datasets.

We monitor the singular values of the Jacobian matrix as:

\[
J_i = \frac{\partial z_{i+1}}{\partial z_i} 
\]

When consecutive layers have the same dimension, the average singular value corresponds to the average ratio of infinitesimal volumes mapped from \(z_i\) to \(z_{i+1}\), as well as to the ratio of average activation variance going from \(z_i\) to \(z_{i+1}\). With our normalized initialization, this ratio is around 0.8 whereas with the standard initialization, it drops down to 0.5.

Figure 6: Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.

4.3 Back-propagated Gradients During Learning

The dynamic of learning in such networks is complex and we would like to develop better tools to analyze and track it. In particular, we cannot use simple variance calculations in our theoretical analysis because the weights values are not anymore independent of the activation values and the linearity hypothesis is also violated.

As first noted by Bradley (2009), we observe (Figure 7) that at the beginning of training, after the standard initialization (eq. 1), the variance of the back-propagated gradients gets smaller as it is propagated downwards. However we find that this trend is reversed very quickly during learning. Using our normalized initialization we do not see such decreasing back-propagated gradients (bottom of Figure 7).

Figure 7: Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized (bottom) initialization. Top: 0-peak decreases for higher layers.

What was initially really surprising is that even when the back-propagated gradients become smaller (standard initialization), the variance of the weights gradients is roughly constant across layers, as shown on Figure 8. However, this is explained by our theoretical analysis above (eq. 14). Interestingly, as shown in Figure 9, these observations on the weight gradient of standard and normalized initialization change during training (here for a tanh network). Indeed, whereas the gradients have initially roughly the same magnitude, they diverge from each other (with larger gradients in the lower layers) as training progresses, especially with the standard initialization. Note that this might be one of the advantages of the normalized initialization, since having gradients of very different magnitudes at different layers may yield to ill-conditioning and slower training.

Finally, we observe that the softsign networks share similarities with the tanh networks with normalized initialization, as can be seen by comparing the evolution of activations in both cases (resp. Figure 3-bottom and Figure 10).

5 Error Curves and Conclusions

The final consideration that we care for is the success of training with different strategies, and this is best illustrated with error curves showing the evolution of test error as training progresses and asymptotes. Figure 11 shows such curves with online training on Shapeset-3×2, while Table 1 gives final test error for all the datasets studied (Shapeset-3×2, MNIST, CIFAR-10, and Small-ImageNet). As a baseline, we optimized RBF SVM models on one hundred thousand Shapeset examples and obtained 59.47% test error, while on the same set we obtained 50.47% with a depth five hyperbolic tangent network with normalized initialization.

These results illustrate the effect of the choice of activation and initialization. As a reference we include in Figure 254 (Glorot and Bengio, 2010).
The weights can also be scaled to account for the activation functions. E.g. ReLU impacts the forward and backward pass as if the weights had half their variances, which motivates multiplying them by $\sqrt{2}$ (He et al., 2015).
The weights can also be scaled to account for the activation functions. E.g. ReLU impacts the forward and backward pass as if the weights had half their variances, which motivates multiplying them by $\sqrt{2}$ (He et al., 2015).

The same type of reasoning can be applied to other activation functions.

In `torch/nn/init.py`

```python
def calculate_gain(nonlinearity, param=None):
    linear_fns = ['linear', 'conv1d', 'conv2d', 'conv3d',
                  'conv_transpose1d', 'conv_transpose2d', 'conv_transpose3d']
    if nonlinearity in linear_fns or nonlinearity == 'sigmoid':
        return 1
    elif nonlinearity == 'tanh':
        return 5.0 / 3
    elif nonlinearity == 'relu':
        return math.sqrt(2.0)
    /.../
Data normalization
The analysis for the weight initialization relies on keeping the activation variance constant.

For this to be true, not only the variance has to remained unchanged through layers, but it has to be correct for the input too.

\[ V(x^{(0)}) = 1. \]
The analysis for the weight initialization relies on keeping the activation variance constant.

For this to be true, not only the variance has to remained unchanged through layers, but it has to be correct for the input too.

\[ \mathbb{V}(x^{(0)}) = 1. \]

This can be done in several ways. Under the assumption that all the input components share the same statistics, we can do

```python
mu, std = train_input.mean(), train_input.std()
train_input.sub_(mu).div_(std)
test_input.sub_(mu).div_(std)
```
The analysis for the weight initialization relies on keeping the activation variance constant.

For this to be true, not only the variance has to remained unchanged through layers, but it has to be correct for the input too.

\[ \mathbb{V}(x^{(0)}) = 1. \]

This can be done in several ways. Under the assumption that all the input components share the same statistics, we can do

```python
mu, std = train_input.mean(), train_input.std()
train_input.sub_(mu).div_(std)
test_input.sub_(mu).div_(std)
```

Thanks to the magic of broadcasting we can normalize component-wise with

```python
mu, std = train_input.mean(0), train_input.std(0)
train_input.sub_(mu).div_(std)
test_input.sub_(mu).div_(std)
```
To go one step further, some techniques initialize the weights explicitly so that the empirical moments of the activations are as desired.

As such, they take into account the statistics of the network activation induced by the statistics of the data.
The end
References
