Deep learning

5.5. Parameter initialization

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ DE GENÈVE

Consider the gradient estimation for a standard MLP, as seen in 3.6. "Back-propagation":

Forward pass

$$x^{(0)} = x, \quad \forall l = 1, \dots, L, \begin{cases} s^{(l)} = w^{(l)} x^{(l-1)} + b^{(l)} \\ x^{(l)} = \sigma\left(s^{(l)}\right) \end{cases}$$

Backward pass

$$\begin{cases} \left[\frac{\partial \ell}{\partial x^{(L)}}\right] \text{ from the definition of } \ell \\ \text{if } l < L, \left[\frac{\partial \ell}{\partial x^{(l)}}\right] = \left(w^{(l+1)}\right)^\top \left[\frac{\partial \ell}{\partial s^{(l+1)}}\right] \end{cases} \qquad \left[\frac{\partial \ell}{\partial s^{(l)}}\right] = \left[\frac{\partial \ell}{\partial x^{(l)}}\right] \odot \sigma'\left(s^{(l)}\right)$$

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

We have

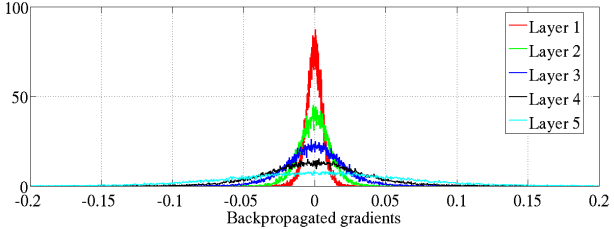
$$\left[\frac{\partial \ell}{\partial x^{(l)}}\right] = \left(w^{(l+1)}\right)^{\top} \left(\sigma'\left(s^{(l)}\right) \odot \left[\frac{\partial \ell}{\partial x^{(l+1)}}\right]\right).$$

so the gradient “vanishes” exponentially with the depth if the ws are ill-conditioned or the activations are in the saturating domain of σ.

We have

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

so the gradient "vanishes" exponentially with the depth if the ws are ill-conditioned or the activations are in the saturating domain of σ.



(Glorot and Bengio, 2010)

Weight initialization

The design of the weight initialization aims at controlling

$$\mathbb{V}\left(\frac{\partial \ell}{\partial w_{i,j}^{(l)}}\right) \text{ and } \mathbb{V}\left(\frac{\partial \ell}{\partial b_{i}^{(l)}}\right)$$

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

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

So in particular, if $\mathbb{E}(A) = \mathbb{E}(B) = 0$, then $\mathbb{V}(AB) = \mathbb{V}(A)\mathbb{V}(B)$.

We will use that, if A and B are independent

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

So in particular, if $\mathbb{E}(A) = \mathbb{E}(B) = 0$, then $\mathbb{V}(AB) = \mathbb{V}(A)\mathbb{V}(B)$.

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

In a standard layer

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

where N_l is the number of units in layer l, and σ is the activation function.

In a standard layer

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

where N_l is the number of units in layer l, and σ is the activation function.

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

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

In a standard layer

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

where N_l is the number of units in layer l, and σ is the activation function.

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

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

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

$$\begin{aligned}\mathbb{V}\left(x_i^{(l)}\right) &\simeq \mathbb{V}\left(\sum_{j=1}^{N_{l-1}} w_{i,j}^{(l)} x_j^{(l-1)}\right)\\ &= \sum_{j=1}^{N_{l-1}} \mathbb{V}\left(w_{i,j}^{(l)}\right) \mathbb{V}\left(x_j^{(l-1)}\right)\end{aligned}$$

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

$$\begin{aligned}\mathbb{V}\left(x_i^{(l)}\right) &\simeq \sum_{j=1}^{N_{l-1}} \mathbb{V}\left(w_{i,j}^{(l)}\right)\mathbb{V}\left(x_j^{(l-1)}\right)\\ &= N_{l-1}\mathbb{V}\left(w^{(l)}\right)\mathbb{V}\left(x^{(l-1)}\right).\end{aligned}$$

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

$$\begin{aligned}\mathbb{V}\left(x_i^{(l)}\right) &\simeq \sum_{j=1}^{N_{l-1}} \mathbb{V}\left(w_{i,j}^{(l)}\right) \mathbb{V}\left(x_j^{(l-1)}\right) \\ &= N_{l-1}\mathbb{V}\left(w^{(l)}\right) \mathbb{V}\left(x^{(l-1)}\right).\end{aligned}$$

So **we have for the variance of the activations:**

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

which leads to a first type of initialization to ensure

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

We can look at the variance of the activations when going though a series of linear layers of various size with a normal weight initialization.

```
s = [ 5, 50, 100, 25, 5 ]
x = torch.randn(1000, s[0])
for n in s[1:]:
    w = torch.randn(x.size(1), n)
    x = x @ w
    print(x.mean(), x.var())
```

prints

```
tensor(0.0045) tensor(5.0118)
tensor(0.0305) tensor(268.1688)
tensor(-0.3304) tensor(22855.8164)
tensor(2.4529) tensor(588037.5625)
```

And the same if we scale the weights in $\frac{1}{N_{l-1}}$.

```
s = [ 5, 50, 100, 25, 5 ]
x = torch.randn(1000, s[0])
for n in s[1:]:
    w = torch.randn(x.size(1), n) / math.sqrt(x.size(1))
    x = x @ w
    print(x.mean(), x.var())
```

prints

```
tensor(0.0113) tensor(1.0412)
tensor(3.4459e-05) tensor(1.0622)
tensor(0.0123) tensor(1.1627)
tensor(0.0095) tensor(1.2369)
```

The standard PyTorch weight initialization for a linear layer

$$f : \mathbb{R}^N \to \mathbb{R}^M$$

is

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

hence

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

```
>>> 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

$$\mathbb{V}\left(\frac{\partial \ell}{\partial x^{(l)}}\right) \simeq N_{l+1} \mathbb{V}\left(\frac{\partial \ell}{\partial x^{(l+1)}}\right) \mathbb{V}\left(w^{(l+1)}\right).$$

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

$$\mathbb{V}\left(\frac{\partial \ell}{\partial x^{(l)}}\right) \simeq N_{l+1} \mathbb{V}\left(\frac{\partial \ell}{\partial x^{(l+1)}}\right) \mathbb{V}\left(w^{(l+1)}\right).$$

So **we have for the variance of the gradient w.r.t. the activations:**

$$\mathbb{V}\left(\frac{\partial \ell}{\partial x^{(l)}}\right) \simeq \mathbb{V}\left(\frac{\partial \ell}{\partial x^{(L)}}\right) \prod_{q=l+1}^{L} N_q \mathbb{V}\left(w^{(q)}\right).$$

Since

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

we have

$$\frac{\partial \ell}{\partial w_{i,j}^{(l)}} \simeq \frac{\partial \ell}{\partial x_i^{(l)}} x_j^{(l-1)},$$

Since

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

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**

$$\begin{aligned}\mathbb{V}\left(\frac{\partial \ell}{\partial w^{(l)}}\right) &\simeq \mathbb{V}\left(\frac{\partial \ell}{\partial x^{(l)}}\right)\mathbb{V}\left(x^{(l-1)}\right)\\ &= \mathbb{V}\left(\frac{\partial \ell}{\partial x^{(L)}}\right)\left(\prod_{q=l+1}^{L} N_q \mathbb{V}\left(w^{(q)}\right)\right)\mathbb{V}\left(x^{(0)}\right)\left(\prod_{q=1}^{l} N_{q-1}\mathbb{V}\left(w^{(q)}\right)\right)\\ &= \frac{1}{N_l}\underbrace{N_0\left(\prod_{q=1}^{L} N_q \mathbb{V}\left(w^{(q)}\right)\right)\mathbb{V}\left(x^{(0)}\right)\mathbb{V}\left(\frac{\partial \ell}{\partial x^{(L)}}\right)}_{\text{Does not depend on } l}.\end{aligned}$$

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**

$$\mathbb{V}\left(\frac{\partial \ell}{\partial b^{(l)}}\right) \simeq \mathbb{V}\left(\frac{\partial \ell}{\partial x^{(l)}}\right).$$

Finally:

1. there is no exponential behavior to mitigate for the gradients w.r.t. weights,
2. to control the variance of activations (e.g. avoid the saturating part of the non-linearities), we need

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

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

$$\mathbb{V}\left(w^{(l)}\right) = \frac{1}{N_l}.$$

Finally:

1. there is no exponential behavior to mitigate for the gradients w.r.t. weights,
2. to control the variance of activations (e.g. avoid the saturating part of the non-linearities), we need

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

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

$$\mathbb{V}\left(w^{(l)}\right) = \frac{1}{N_l}.$$

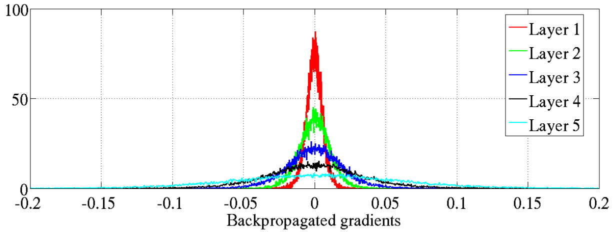
The "Xavier initialization" is a compromise

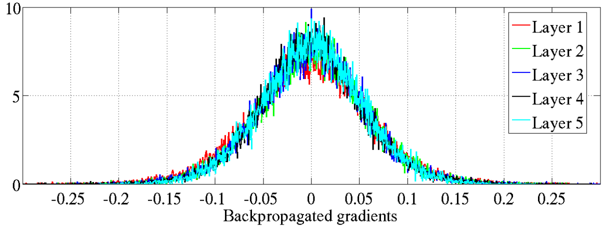
$$\mathbb{V}\left(w^{(l)}\right) = \frac{1}{\frac{N_{l-1}+N_l}{2}} = \frac{2}{N_{l-1}+N_l}.$$

(Glorot and Bengio, 2010)

In `torch/nn/init.py`

```
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)
```





(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`

```
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.

$$\mathbb{V}\left(x^{(0)}\right) = 1.$$

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.

$$\mathbb{V}\left(x^{(0)}\right) = 1.$$

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

```
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.

$$\mathbb{V}\left(x^{(0)}\right) = 1.$$

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

```
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

```
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

X. Glorot and Y. Bengio. **Understanding the difficulty of training deep feedforward neural networks**. In International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.

K. He, X. Zhang, S. Ren, and J. Sun. **Delving deep into rectifiers: Surpassing human-level performance on imagenet classification**. CoRR, abs/1502.01852, 2015.