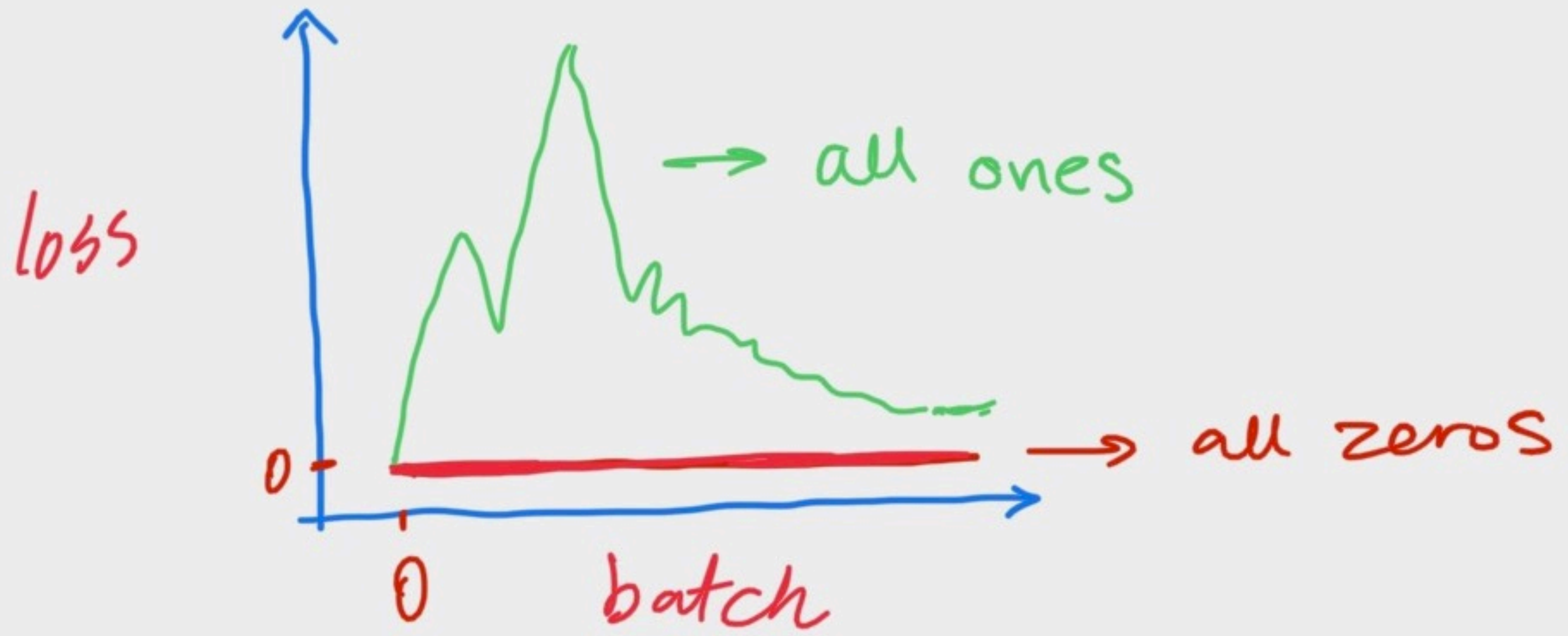
Choosing Initial weights

- All zeros or all ones NOT GOOD

inside the init function of the Module:

```
for m in self.modules():
    if isinstance(m, nn.Linear):
        nn.init.Constant_(m.weight, const_w)
        nn.init.Constant_(m.bias, 0)
```

Comparison: model_zero vs. model-one



} both have low accuracy

- Uniform random weights

e.g., np.random.uniform(-3, 3, [1000])

values between 3, -3

→ generate 1000 numbers

Writing an initialization function:

```
def weight_init_uniform(m):
    classname = m.__class__.__nam__
    if classname.find('Linear') != 1:
        *m.weight.data.uniform_(0.0, 1.0)
        m.bias.data.fill_(0)
```

Module

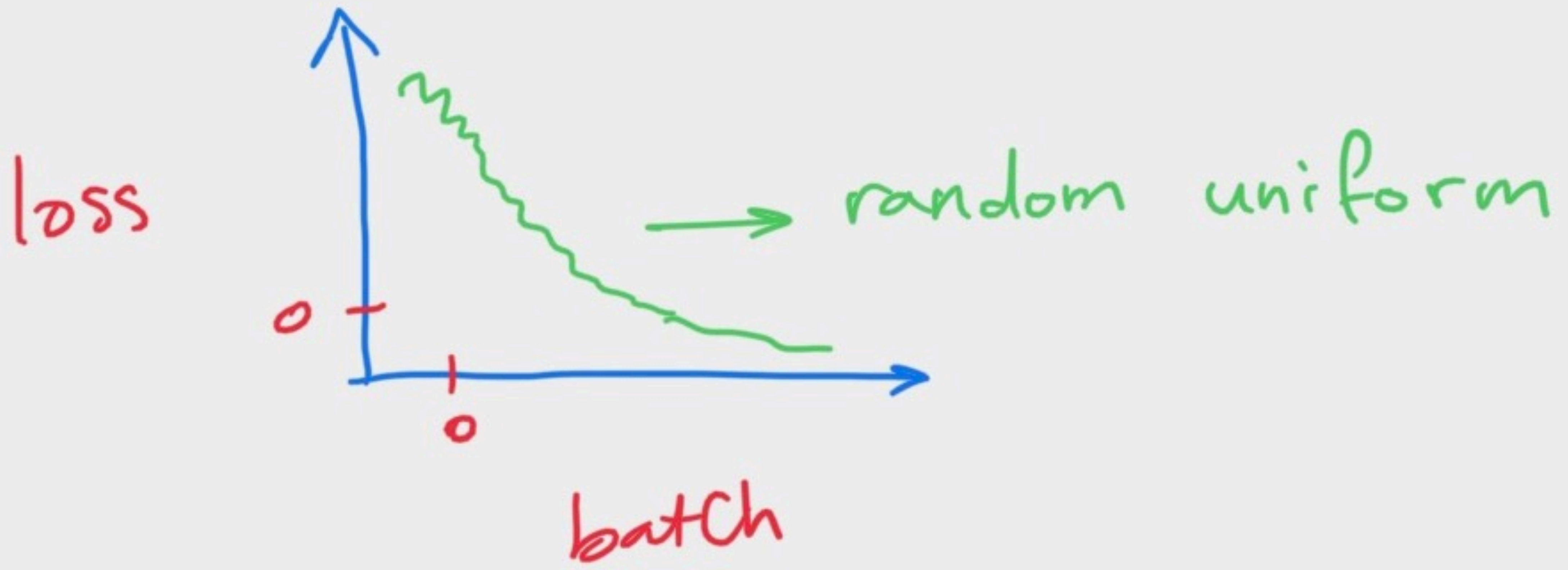
for every Linear layer in a model...

still not the best

set bias to zero

uniform distribution for weights

```
model = Net()
model.apply(weight_init_uniform)
```



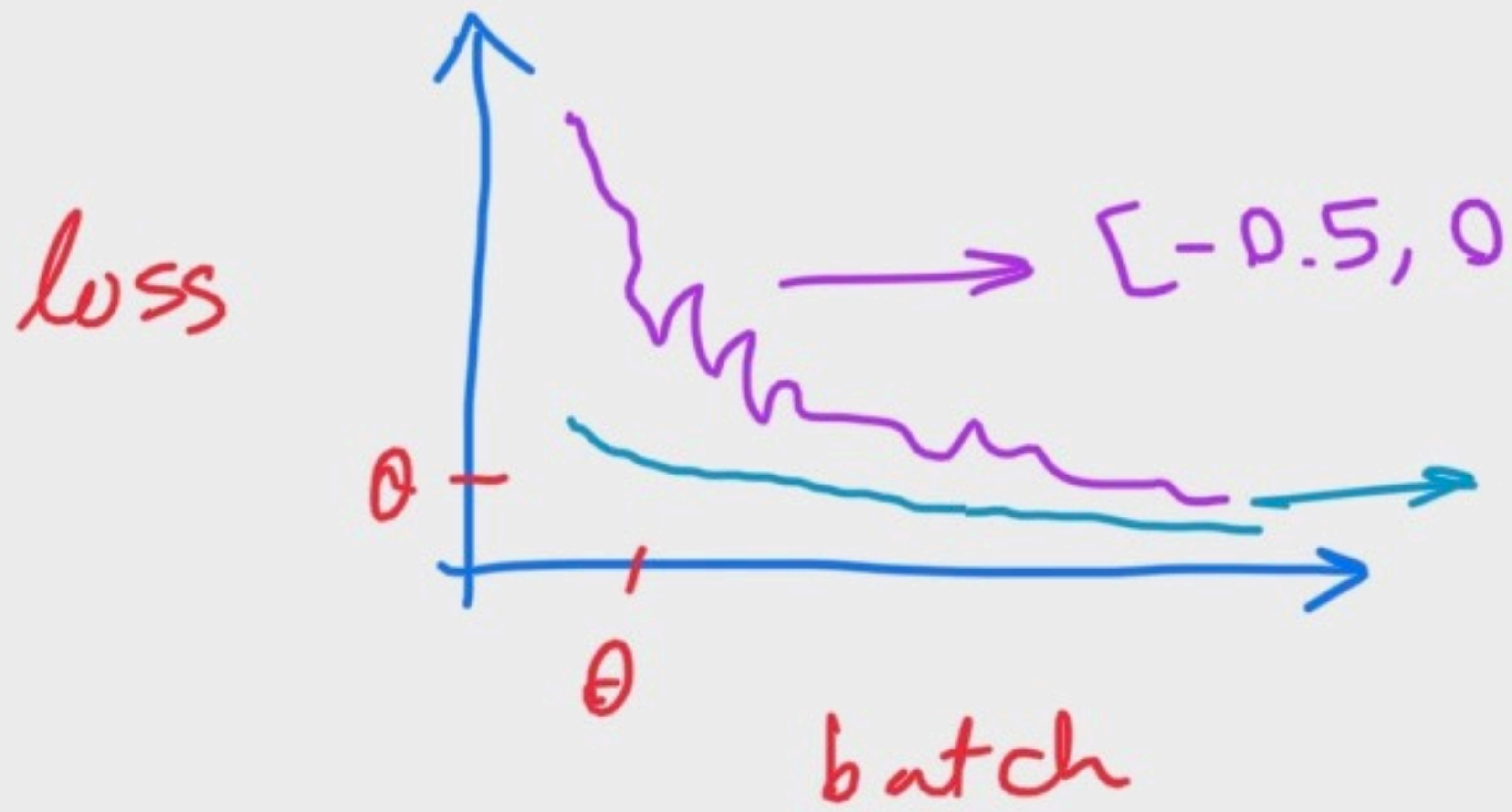
- Relationship between #inputs and uniform range

more input ; smaller range

Rule of thumb :

choose uniform random weights from the range $[-y, +y]$. $y = \frac{1}{\sqrt{n}}$

n → #inputs of the neuron



→ [-0.5, 0.5]

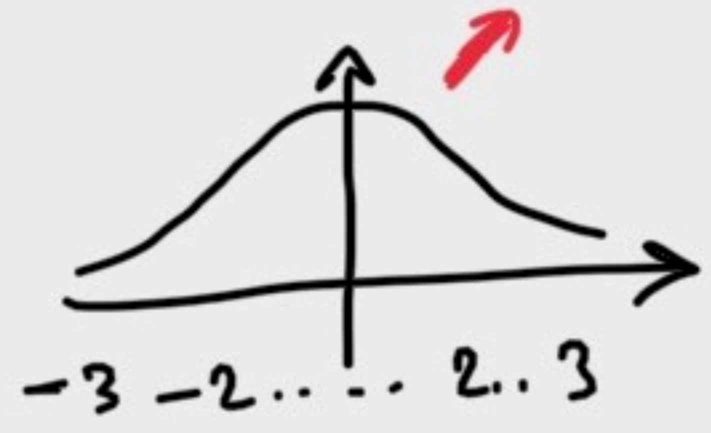
→ using [-y, y] → ends up with higher accuracy

- Normal Distribution

mean

e.g., np.random.normal(0, 1, [1000])

std dev

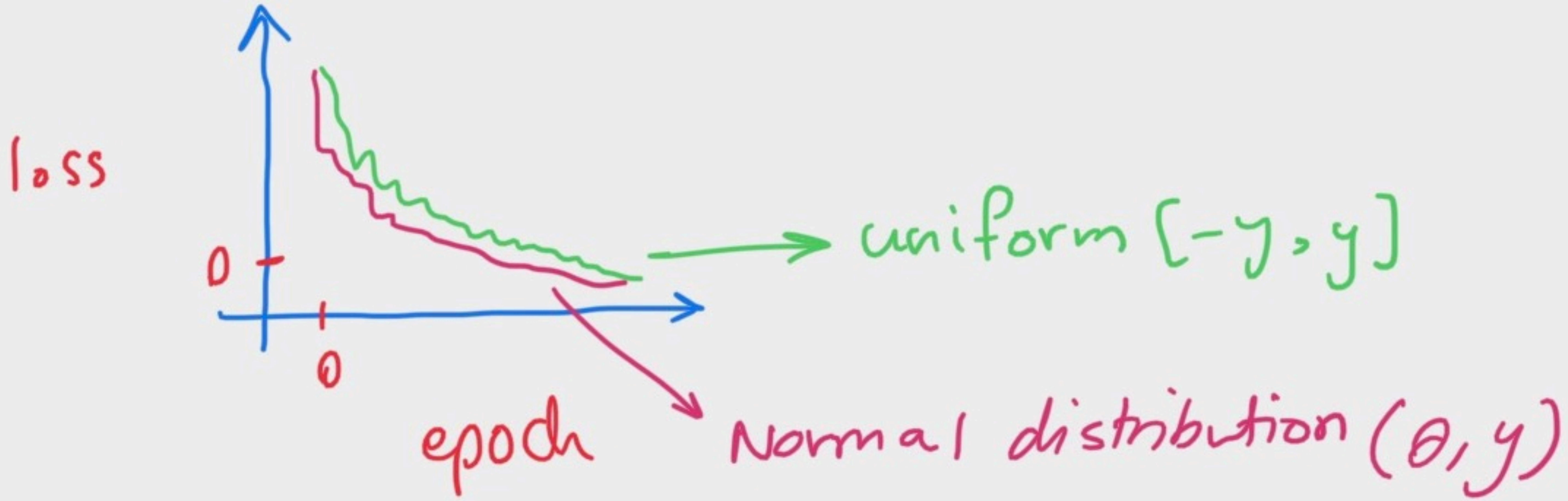


more values near zero

typically, mean = 0 and std dev = $1/\sqrt{n}$.

in weight_init function, we do the following,

```
def weight_init_normal(m):
    classname = m.__class__.__nam__
    if classname.find('Linear') != 1:
        n = m.in_features
        y = 1.0/np.sqrt(n)
      ★ m.weight.data.normal_(0, y)
        m.bias.data.fill_(0)
```



* In PyTorch, the default behavior for initialization of weights is Cool! Actually, it is usually the "uniform" distribution. However, there are Cases where "normal" distribution will be better.