Choosing Initial weights

- All zeros or all ones NOT GOOD
  inside the init function of the model:
  ```python
  for m in self.modules():
      if isinstance(m, nn.Linear):
          nn.init.constant_(m.weight, const_w)
          nn.init.constant_(m.bias, theta)
  ```
  Comparison: model_zero vs. model_one

- Uniform random weights
  e.g., np.random.uniform([-3, 3], [1000])
  values between -3, 3
  generate 1000 numbers
Writing an initialization function:

```python
def weight_init_uniform(m):
    class_name = m._class_name_
    if class_name.find('Linear') != 1:
        m.weight.data.uniform_(0.0, 1.0)
        m.bias.data.fill_(0)
```

for every linear layer in a model...

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 
      
      \[ [-y, +y] \quad \text{with} \quad y = \frac{1}{\sqrt{n}} \]

#inputs of the neuron
- Normal Distribution

\[ \text{Normal distribution, e.g., np.random.normal(0, 1, [1000])} \]

Typically, \( \text{mean} = \theta \) and \( \text{std dev} = \frac{1}{\sqrt{n}} \).

In weight init function, we do the following:

```python
def weight_init_normal(m):
    classname = m._class.__name__
    if classname.find('Linear') != -1:
        n = m.in_features
        y = 1.0 / np.sqrt(n)
        m.weight.data.normal_(\theta, y)
        m.bias.data.fill_(\theta)
```

- Higher accuracy using \([-y, y]\)
* 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.