Model Initialization

Image-to-Image Translation with Conditional Adversarial Nets

Week 5

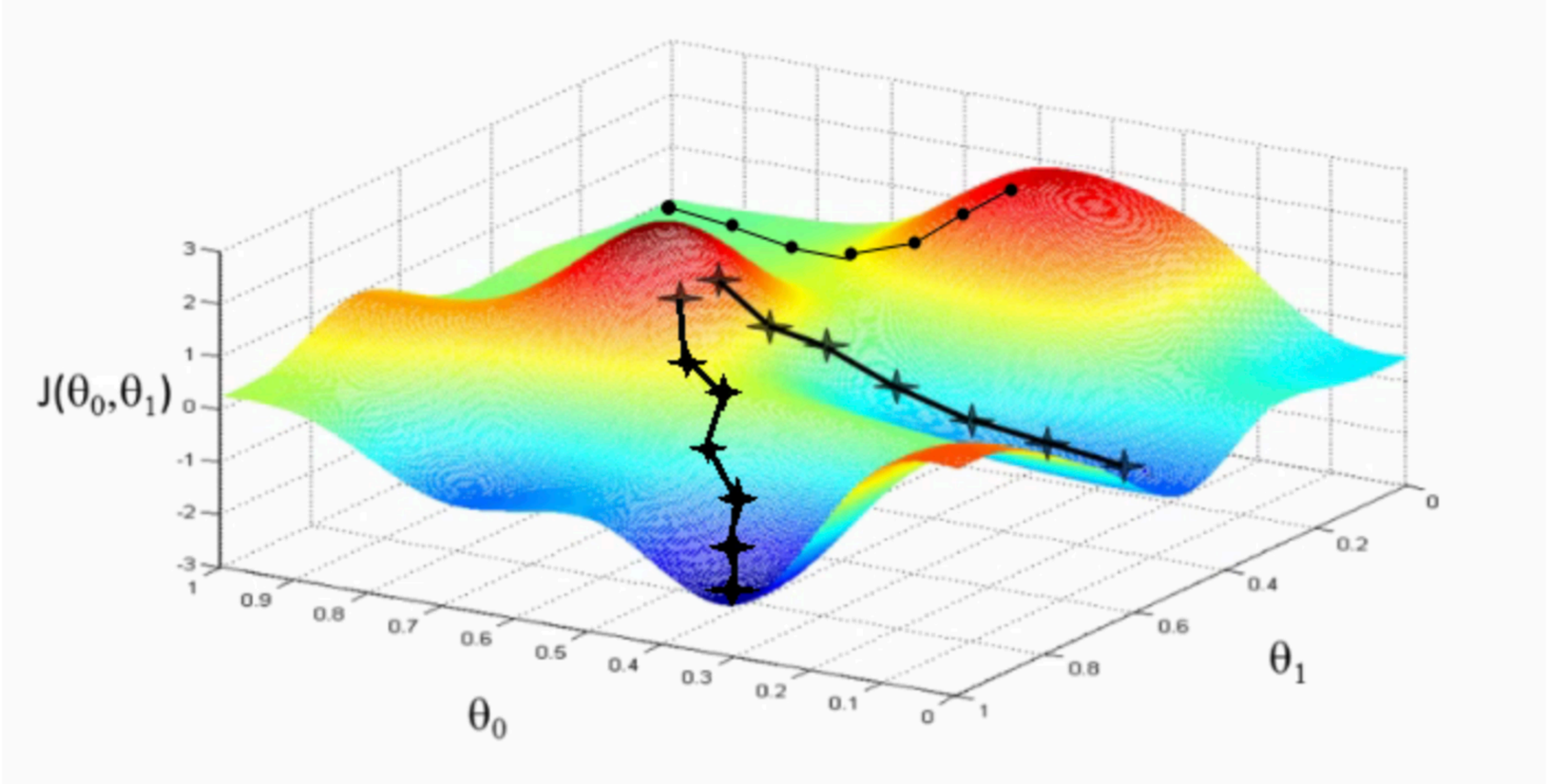
Model Initialization

So… what is it?

- 말 그대로 초기화!
    - Model needs a nice “Starting Point”
- 왜 필요할까?

Model Initialization

So… Why do we have to care about it?



Model Initialization

Types!

- 0.0
- Random Values
  - Range
    - [-0.3, -0.3]
    - [0, 1]
    - [-1, 1]
  - Distribution
    - Gaussian
    - Uniform

Xavier Initialization

Sigmoid / Tanh

- Random Uniform Distribution bounded between

$$a = \text{gain} \times \sqrt{\frac{6}{\text{fan\_in} + \text{fan\_out}}}$$

- Random Normal Distribution bounded between

$$\text{std} = \text{gain} \times \sqrt{\frac{2}{\text{fan\_in} + \text{fan\_out}}}$$

He Initialization(Kaiming)

ReLU

- Random Uniform Distribution bounded between

$$\text{bound} = \text{gain} \times \sqrt{\frac{3}{\text{fan\_mode}}}$$

- Random Normal Distribution bounded between

$$\text{std} = \frac{\text{gain}}{\sqrt{\text{fan\_mode}}}$$

PyTorch - init

torch.nn.init

- torch.nn.init.uniform_(x, lower bound, upper bound)
- torch.nn.init.normal_(x, mean, std)
- torch.nn.init.constant_(x, val)
- torch.nn.init.zeros_(x)
- torch.nn.init.xavier_normal_(x, gain)
- torch.nn.init.xavier_uniform_(x, gain)
- torch.nn.init.kaiming_normal_(x, LeakyReLU slope, fan_mode, nonlinearity)
- torch.nn.init.kaiming_uniform_(x, LeakyReLU slope, fan_mode, nonlinearity)

Pix2Pix Weight Initialization

Gaussian Distribution!

6.2. Training details

Random jitter was applied by resizing the 256×256 input images to 286 × 286 and then randomly cropping back to size 256 × 256.

All networks were trained from scratch. Weights were initialized from a Gaussian distribution with mean 0 and standard deviation 0.02.

Pix2Pix Weight Initialization

So… How to?

`apply(fn)` [SOURCE]

Applies `fn` recursively to every submodule (as returned by `.children()`) as well as self. Typical use includes initializing the parameters of a model (see also torch.nn.init).

Parameters

fn (`Module` -> None) – function to be applied to each submodule

```
>>> @torch.no_grad()
>>> def init_weights(m):
>>>     print(m)
>>>     if type(m) == nn.Linear:
>>>         m.weight.fill_(1.0)
>>>         print(m.weight)
>>> net = nn.Sequential(nn.Linear(2, 2), nn.Linear(2, 2))
>>> net.apply(init_weights)
```