Day 12

Peer session Daily Presentation

Cifar-18

Batch Normalization
에 대해서 생각해보기

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift(2015)

How Does Batch Normalization Help Optimization? (2018)

why does Batch norm work? / https://www.youtube.com/watch?v=nUUqwaxLnWs

Batch Normalization – EXPLAINED!/ https://www.youtube.com/watch?v=DtEq44FTPM4

Batch norm? 어제 배움

네트워크에서 만들어지는 feature의 평균값과 분산값을 batch 단위로 계산(빼고, 나눠주고)해서 normalize 해주는 기법

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

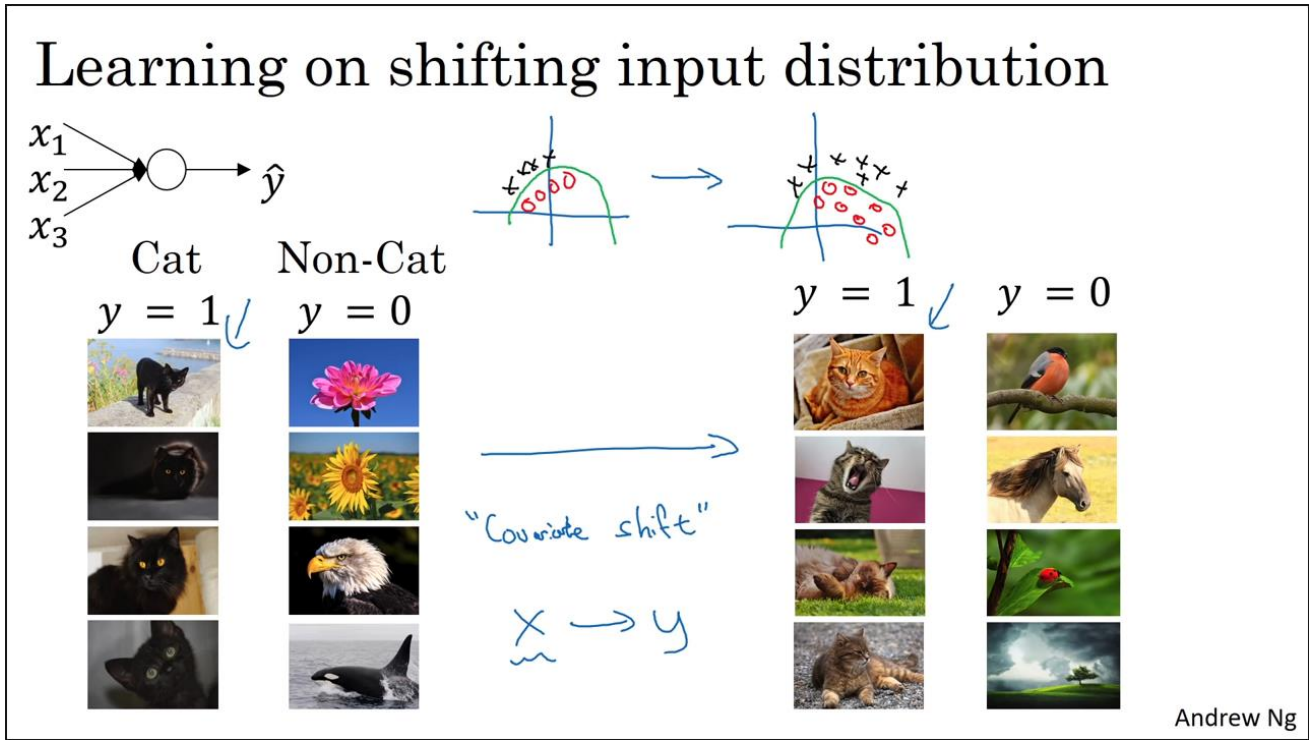
$$\sigma_B^2 = \sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

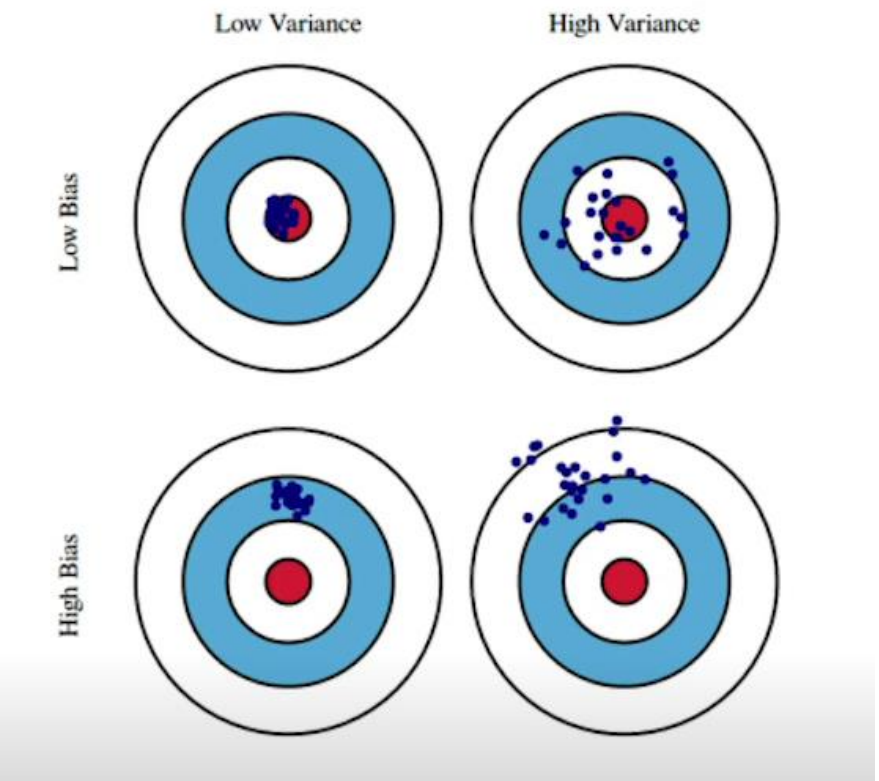
(어제 제꺼 학습정리에서.. ㅎㅎ)

Covariate shift 는 무엇인가?

- covariate = 공변량 / shift = 변화
- 공변량이란? 여러 변수들이 공통적으로 함께 공유하고 있는 변량, 독립변수들이 종속 변수에 얼마나 영향을 주는지에 대한 것



Covariate shift를 줄인 다는 것은?



Bias and Variance Tradeoff

Given $\mathcal{D} = \{(x_i, t_i)\}_{i=1}^N$, where $t = f(x) + \epsilon$ and $\epsilon \sim \mathcal{N}(0, \sigma^2)$

We can derive that what we are minimizing (**cost**) can be decomposed into three different parts: **bias²**, **variance**, and **noise**.

$$\underbrace{\mathbb{E}\left[(t - \hat{f})^2\right]}_{\textbf{cost}} = \mathbb{E}\left[(t - f + f - \hat{f})^2\right]$$

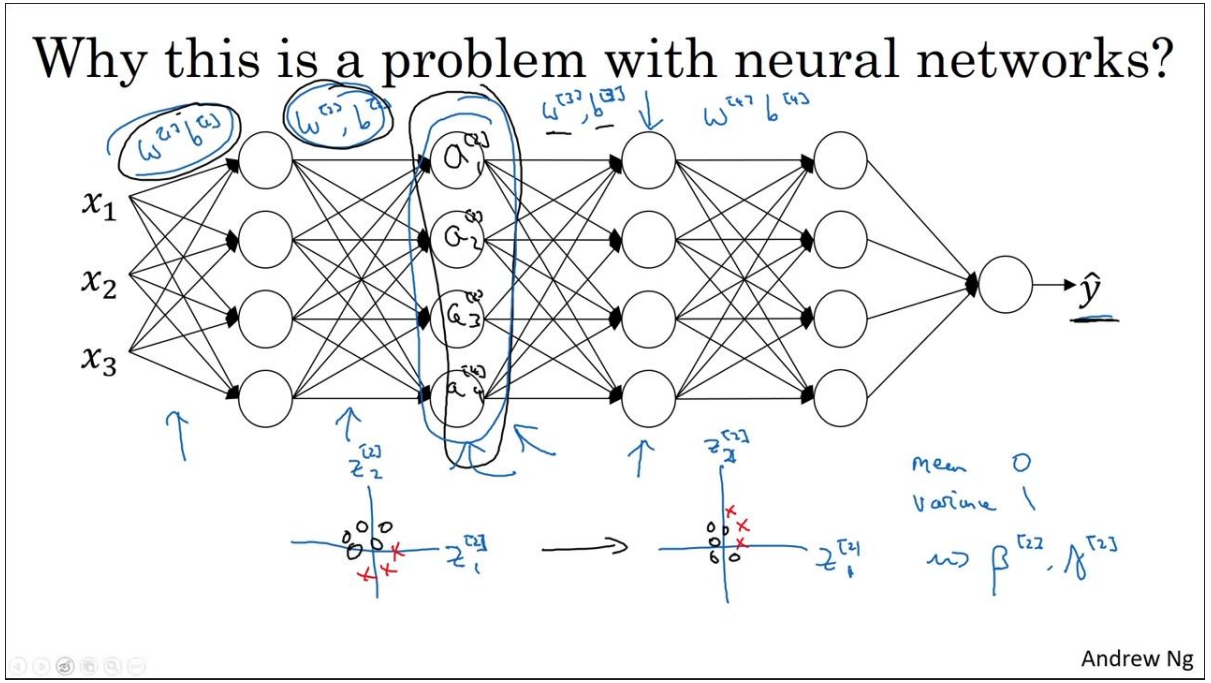
$$= \dots$$

$$= \underbrace{\mathbb{E}\left[(f - \mathbb{E}[\hat{f}]^2)^2\right]}_{\textbf{bias}^2} + \underbrace{\mathbb{E}\left[(\mathbb{E}[\hat{f}] - \hat{f})^2\right]}_{\textbf{variance}} + \underbrace{\mathbb{E}\left[\epsilon\right]}_{\textbf{noise}}$$

bias와 variance를 (강제로) 0과 1에 가까이 맞추고 대신 noise를 일부 증가시킴

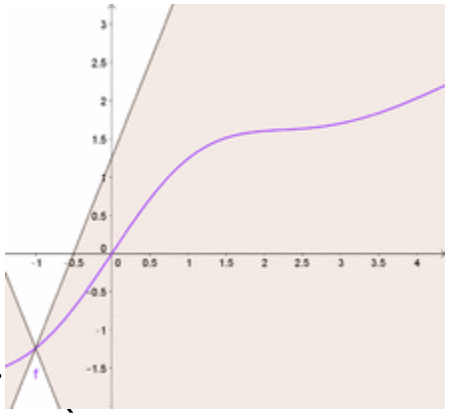
그래서 결론적으로?

- 각 layer 가 학습할 때 이전 layer의 영향을 덜 받게 됨 (= 더 독립적으로 학습하게 됨) => 이래서 학습이 잘되지 않을까? 가 기존의 추측이었음



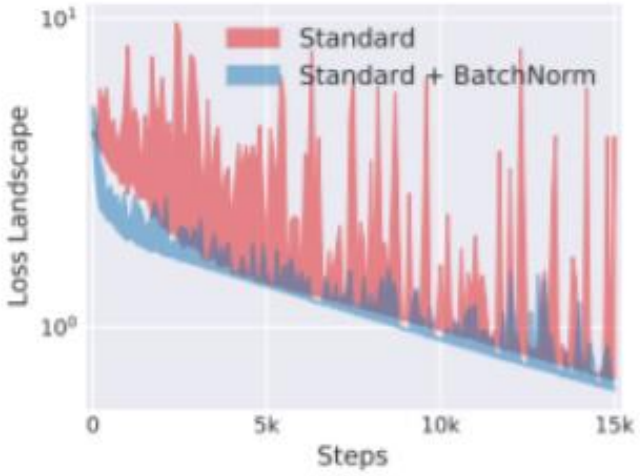
https://en.wikipedia.org/wiki/Lipschitz_continuity

결국 ICS와는 상관이 없었고..

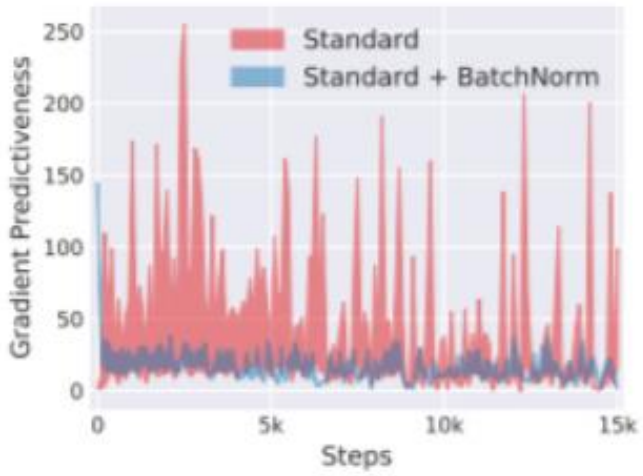


- **How Does Batch Normalization Help Optimization? (2018)**

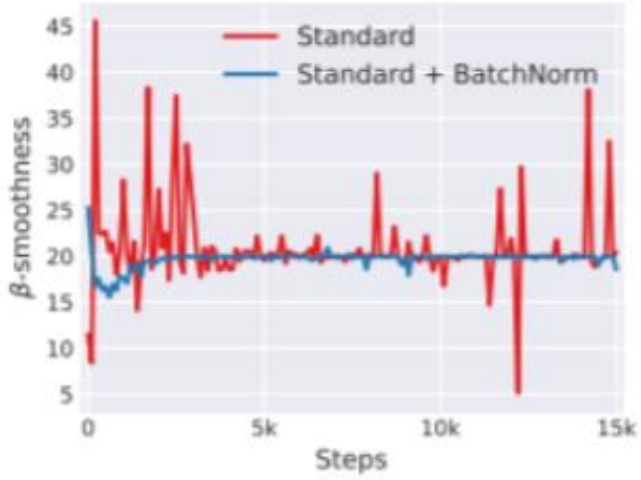
실제로는 **립시츠성(?)**(Lipschitzness)을 개선, beta-smoothness
립시츠 함수 : 두 점 사이의 거리를 일정 비 이상으로 증가시키지 않는 함수 (Uniform continuity의 강력한 형태)



(a) loss landscape

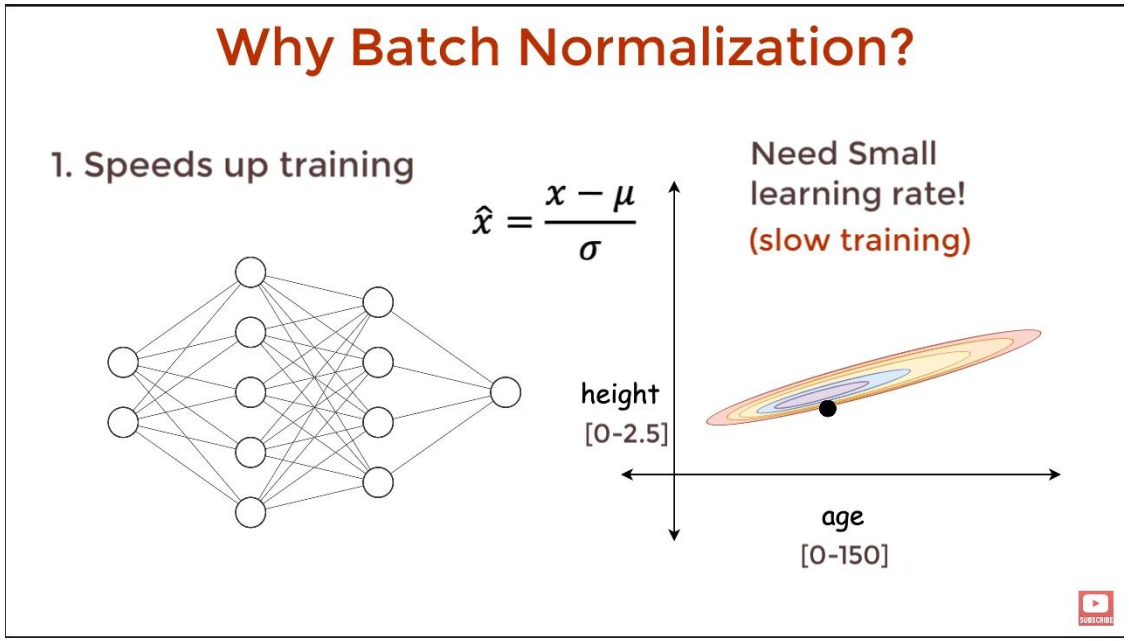


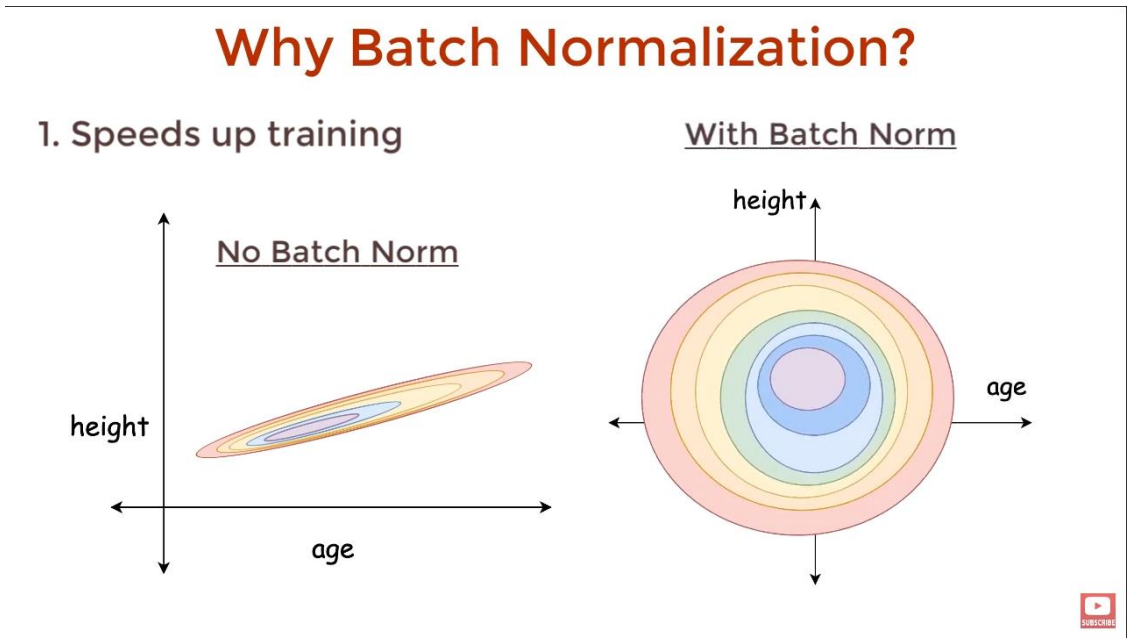
(b) gradient predictiveness



(c) "effective" β-smoothness

그럼 학습 속도랑 무슨상관이죠?

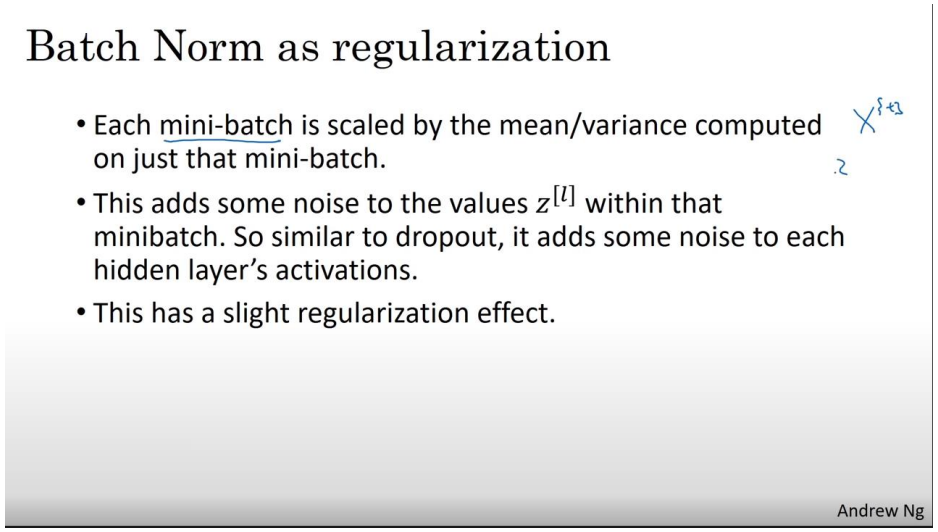




- gradient 변화의 신뢰도& loss 값의 변화량이 안정적임 => learning rate를 높게 설정할 수 있음 => 빨라짐
- 거기다가 어디서 시작하더라도 optimal 값에 이르는 데에 드는 연산 횟수가 비슷해지는 효과도 있음 (allows sub-optimal starts)

Batch Norm의 부수적인 효과

- regularization의 효과도 있음. (drop out과 유사)
- 대신 batch size가 커질 수록 그 효과는 미미해짐 (noise가 커지기 때문 = 곱셈과 뺄셈 연산이 계속 이루어지므로)
- 하지만 Andrew ng 좌 께선 batch norm을 regularization 으로 쓰는 걸 비추. 속도 향상 시키는데 이용하라고 추천



Tensor

PyTorch

- PyTorch는 Numpy, AutoGrad, Function을 지원해줌

- AutoGrad가 되는 이유?

torch.Tensor

- .requires_grad 속성 : bool
- True인 경우 : 해당 tensor에서 이뤄지는 모든 연산들을 추적
- 추적한 연산 기록으로부터 .backward()가 진행

Requires_grad와 .backward()

```
>>> x =torch.ones(2,2)
>>> x
tensor([[1., 1.],
        [1., 1.]])
```

- Tensor requires_grad는 default False

```
>>> x =torch.ones(2,2)
>>> x
tensor([[1., 1.],
        [1., 1.]])
>>> out = x.mean()
>>> out
tensor(1.)
>>> out.backward()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\rlfgm\anaconda3\envs\boostcamp\lib\site-packages\torch\tensor.py", line 195, in backward
    torch.autograd.backward(self, gradient, retain_graph, create_graph)
  File "C:\Users\rlfgm\anaconda3\envs\boostcamp\lib\site-packages\torch\autograd\__init__.py", line 97, in
backward
    Variable._execution_engine.run_backward(
RuntimeError: element 0 of tensors does not require grad and does not have a grad_fn
```

Requires_grad와 .backward()

```
>>> x =torch.ones(2,2, requires_grad = True)
>>> x
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
>>> out = x.mean()
>>> out
tensor(1., grad_fn=<MeanBackward0>)
>>> out.backward()
>>> x.grad
tensor([[0.2500, 0.2500],
        [0.2500, 0.2500]])
```

- x.grad : d(out)/dx

```
>>> x =torch.ones(2,2, requires_grad = True)
>>> x
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
>>> x.grad
>>>
```

- Backward를 진행하지 않았다면 .grad 에는 값이 들어있지 않음

Requires_grad와 .backward()

```
>>> import torch.nn as nn
>>> loss = nn.L1Loss()
>>> input = torch.randn(3,5, requires_grad=True)
>>> input
tensor([[-0.2030, -0.3561, -0.7537,  1.9856, -0.0401],
        [-0.6991, -0.3836,  0.5591,  1.1650, -1.4671],
        [ 0.7387, -0.0524,  0.0712,  1.0169,  0.7298]], requires_grad=True)
>>> target = torch.randn(3,5)
>>> target
tensor([[ 0.0132,  0.3148,  1.9821,  0.4432,  2.0697],
        [ 0.7573,  0.8790, -0.7044, -0.2887,  0.6318],
        [ 0.0243,  0.3464, -2.0329, -0.2538, -0.1000]])
>>> output = loss(input, target)
>>> output
tensor(1.3419, grad_fn=<L1LossBackward>)
>>> output.backward()
>>> input.grad
tensor([[-0.0667, -0.0667, -0.0667,  0.0667, -0.0667],
        [-0.0667, -0.0667,  0.0667,  0.0667, -0.0667],
        [ 0.0667, -0.0667,  0.0667,  0.0667,  0.0667]])
```

- 이때도 requires_grad = True이다.
- Target은 gradient를 추정할 필요 없고 고정된 값이므로 requires_grad를 False로 둔다.

Requires_grad와 .backward()

```
y_pred_adam = model_adam.forward(batch_x)
loss_adam = loss(y_pred_adam,batch_y)
optm_adam.zero_grad()
loss_adam.backward()
optm_adam.step()
```

- Parameter들도 다 requires_grad가 True인 tensor이다.

```
[docs]class Parameter(torch.Tensor):
```

CLASS `torch.nn.parameter.Parameter` [SOURCE]

Parameters

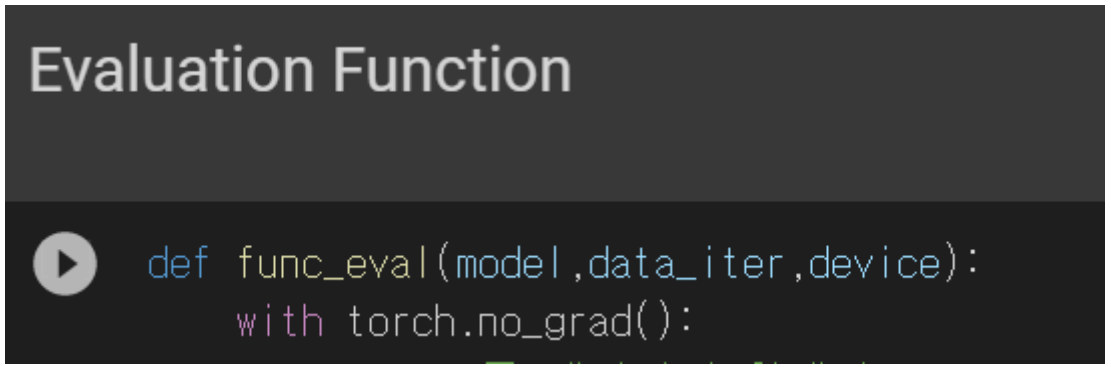
- **data** (*Tensor*) – parameter tensor.
- **requires_grad** (*bool, optional*) – if the parameter requires gradient. See Excluding subgraphs from backward for more details. Default: *True*

With torch.no_grad()

- requires_grad가 True면 해당 tensor의 연산을 모두 추적

- With torch.no_grad()로 코드 블록을 감싸면 해당 코드 블록 내에서의 연산에 대해서는 requires_grad가 True인 tensor들이 추적을 하지 않는다.
- 또한 메모리를 사용하는 것도 방지해 메모리에 대해서도 효율적

- Evaluation 할 때 사용



Tensor의 copy

> • WARNING
>
> `torch.tensor()` always copies `data`. If you have a Tensor `data` and just want to change its `requires_grad` flag, use `requires_grad_()` or `detach()` to avoid a copy. If you have a numpy array and want to avoid a copy, use `torch.as_tensor()`.

- Torch.tensor()는 항상 data를 copy한다.
- Tensor에서 data를 가지고 있고 requires_grad만 바꾸고 싶다면 requires_grad_() 혹은 copy를 피하고 싶다면 detach()를 사용해라
- numpy array를 가지고 있고 copy를 피하고 싶다면 torch.as_tensor()를 사용해라

Tensor의 copy

- Tensor에서 data를 가지고 있고 requires_grad만 바꾸고 싶다면 requires_grad_() 혹은 copy를 피하고 싶다면 detach()를 사용해라

```
>>> a = torch.tensor([[1.,1.], [1.,1.]])
>>> a
tensor([[1., 1.],
        [1., 1.]])
>>> id(a)
2665054750656
```

requires_grad_()

해당 tensor의 requires_grad를 True로 만듬

id가 동일

```
>>> b = a.requires_grad_()
>>> b
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
>>> id(b)
2665054750656
```

Tensor의 copy

- Tensor에서 data를 가지고 있고 requires_grad만 바꾸고 싶다면 requires_grad_() 혹은 copy를 피하고 싶다면 detach()를 사용해라

```
>>> a
tensor([[1., 1.],
        [1., 1.]])
>>> id(a)
2665063330560
```

```
>>> c = a.detach()
>>> c
tensor([[1., 1.],
        [1., 1.]])
>>> id(c)
2665063330176
```

detach()

Tensor를 copy하지 않고 requires_grad가 False인 애를 반환

```
>>> b
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
>>> id(b)
2665054750656
>>> d = b.detach()
>>> d
tensor([[1., 1.],
        [1., 1.]])
>>> id(d)
2665054697600
```

Tensor의 copy

- Tensor에서 data를 가지고 있고 requires_grad만 바꾸고 싶다면 requires_grad_() 혹은 copy를 피하고 싶다면 detach()를 사용해라

```
>>> a
tensor([[1., 1.],
        [1., 1.]])
>>> e = a.detach().requires_grad_()
>>> e
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
>>> id(a)
2665063330560
>>> id(e)
2665063330624
```

detach().requires_grad_()

Copy는 싫은데 requires_grad가 True였으면 좋겠다

Tensor의 copy

- Torch.tensor()는 항상 data를 copy한다.
  방금 전 사용한 함수들 외에 다른 것들을 사용해 기존의 tensor로 부터 다른 tensor를 만들면
  두 tensor의 data는 공유된다.

```
>>> a
tensor([[1., 1.],
        [1., 1.]])
>>> id(a)
2665063330560
```

```
>>> c = a.detach()
>>> c
tensor([[1., 1.],
        [1., 1.]])
>>> id(c)
2665063330176
```

```
>>> a[0][0]=2.
>>> a
tensor([[2., 1.],
        [1., 1.]])
>>> c
tensor([[2., 1.],
        [1., 1.]])
```

Tensor의 id는 달라도 내부의 data를 공유

Tensor의 copy

- Torch.tensor()는 항상 data를 copy한다.
  data공유가 싫으면 .clone()을 사용

```
>>> a
tensor([[2., 1.],
        [1., 1.]])
>>> id(a)
2665063330560
>>> f = a.clone()
>>> f
tensor([[2., 1.],
        [1., 1.]])
>>> id(f)
2665063330752
```

```
>>> a[0][0]=1.
>>> a
tensor([[1., 1.],
        [1., 1.]])
>>> f
tensor([[2., 1.],
        [1., 1.]])
```

Data를 공유하지 않기 때문에 a의 data를 바꿔도 f에는 영향이 없다.

Tensor의 copy

- Torch.tensor()는 항상 data를 copy한다.
  data공유가 싫으면 .clone()을 사용

```
>>> a
tensor([[2., 1.],
        [1., 1.]])
>>> id(a)
2665063330560
>>> f = a.clone()
>>> f
tensor([[2., 1.],
        [1., 1.]])
>>> id(f)
2665063330752
```

```
>>> a[0][0]=1.
>>> a
tensor([[1., 1.],
        [1., 1.]])
>>> f
tensor([[2., 1.],
        [1., 1.]])
```

Data를 공유하지 않기 때문에 a의 data를 바꿔도 f에는 영향이 없다.

Tensor의 copy

- numpy array를 가지고 있고 copy를 피하고 싶다면 torch.as_tensor()를 사용해라

```
>>> x = np.array([[1.,1.],[1.,1.]])
>>> x
array([[1., 1.],
       [1., 1.]])
>>> id(x)
2665063280432
>>> g = torch.as_tensor(x)
>>> g
tensor([[1., 1.],
        [1., 1.]], dtype=torch.float64)
>>> id(g)
2667079067648
>>> h = torch.as_tensor(x)
>>> h
tensor([[1., 1.],
        [1., 1.]], dtype=torch.float64)
>>> id(h)
2665063331072
```

```
>>> g[0][0]=2.
>>> x
array([[2., 1.],
       [1., 1.]])
>>> h
tensor([[2., 1.],
        [1., 1.]], dtype=torch.float64)
```

Tensor의 copy를 피하는 거지
Data의 공유는 동일

Reference

- Tensor의 backward
- https://tutorials.pytorch.kr/beginner/blitz/autograd_tutorial.html

- Tensor의 pytorch docs
- https://pytorch.org/docs/stable/tensors.html

nn.linear, nn.Sequential에서 시작한 발표

Linear Layers

Linear Layers

`nn.Identity`	A placeholder identity operator that is argument-insensitive.
`nn.Linear`	Applies a linear transformation to the incoming data: $y = xA^T + b$
`nn.Bilinear`	Applies a bilinear transformation to the incoming data: $y = x_1^T A x_2 + b$

nn.linear

- 참..
- 그..
- 역시나
- Module 상속
- 합니다

```
class Linear(Module):
    def __init__(self, in_features: int, out_features: int, bias:
bool = True) -> None:
        super(Linear, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(out_features,
in_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self) -> None:
        init.kaiming_uniform_(self.weight, a=math.sqrt(5))
        if self.bias is not None:
            fan_in, _ =
init._calculate_fan_in_and_fan_out(self.weight)
            bound = 1 / math.sqrt(fan_in)
            init.uniform_(self.bias, -bound, bound)

    def forward(self, input: Tensor) -> Tensor:
        return F.linear(input, self.weight, self.bias)

    def extra_repr(self) -> str:
        return 'in_features={}, out_features={}, bias={}'.format(
            self.in_features, self.out_features, self.bias is not
None
        )
```

Parameter

- A kind of Tensor that is to be considered a module parameter

Parameters are `Tensor` subclasses, that have a very special property when used with `Module` s - when they're assigned as Module attributes they are automatically added to the list of its parameters, and will appear e.g. in `parameters()` iterator. Assigning a Tensor doesn't have such effect. This is because one might want to cache some temporary state, like last hidden state of the RNN, in the model. If there was no such class as `Parameter` , these temporaries would get registered too.

- When they're assigned as Module!!

Parameter

- 이건 또..
- 다 Tensor로
- 이어지는데..

- Tensor는
- .. 다른분이?

```
class Parameter(torch.Tensor):
    def __new__(cls, data=None, requires_grad=True):
        if data is None:
            data = torch.Tensor()
        return torch.Tensor._make_subclass(cls, data, requires_grad)

    def __deepcopy__(self, memo):
        if id(self) in memo:
            return memo[id(self)]
        else:
            result = type(self)
(self.data.clone(memory_format=torch.preserve_format),
self.requires_grad)
            memo[id(self)] = result
            return result

    def __repr__(self):
        return 'Parameter containing:\n' + super(Parameter,
self).__repr__()

    def __reduce_ex__(self, proto):
        # See Note [Don't serialize hooks]
        return (
            torch._utils._rebuild_parameter,
            (self.data, self.requires_grad, OrderedDict())
        )

    __torch_function__ = _disabled_torch_function_impl
```

F.linear

```
[docs]def linear(input, weight, bias=None):
    # type: (Tensor, Tensor, Optional[Tensor]) -> Tensor
    r"""
    Applies a linear transformation to the incoming data: :math:`y =
xA^T + b`.

    tens_ops = (input, weight)
    if not torch.jit.is_scripting():
        if any([type(t) is not Tensor for t in tens_ops]) and
has_torch_function(tens_ops):
            return handle_torch_function(linear, tens_ops, input,
weight, bias=bias)
    if input.dim() == 2 and bias is not None:
        # fused op is marginally faster
        ret = torch.addmm(bias, input, weight.t())
    else:
        output = input.matmul(weight.t())
        if bias is not None:
            output += bias
        ret = output
    return ret
```

Performs a matrix multiplication of the matrices `mat1` and `mat2`. The matrix `input` is added to the final result.

What is PyTorch JIT?

PyTorch JIT is an optimized compiler for PyTorch programs.

1. It is a lightweight threadsafe interpreter
2. Supports easy to write custom transformations
3. It's not just for inference as it has auto diff support

https://discuss.pytorch.org/t/torch-jit-is-scripting-and-handle-torch-function/88451

Addmm vs matmul

TORCH.ADDMM

`torch.addmm(`*`input`*`,` *`mat1`*`,` *`mat2`*`,` *`*`*`,` *`beta=1`*`,` *`alpha=1`*`,` *`out=None`*`)` → Tensor

Performs a matrix multiplication of the matrices `mat1` and `mat2`. The matrix `input` is added to the final result.

`torch.matmul(`*`input`*`,` *`other`*`,` *`*`*`,` *`out=None`*`)` → Tensor

Matrix product of two tensors.

The behavior depends on the dimensionality of the tensors as follows:

- If both tensors are 1-dimensional, the dot product (scalar) is returned.
- If both arguments are 2-dimensional, the matrix-matrix product is returned.
- If the first argument is 1-dimensional and the second argument is 2-dimensional, a 1 is prepended to its dimension for the purpose of the matrix multiply. After the matrix multiply, the prepended dimension is removed.
- If the first argument is 2-dimensional and the second argument is 1-dimensional, the matrix-vector product is returned.
- If both arguments are at least 1-dimensional and at least one argument is N-dimensional (where N > 2), then a batched matrix multiply is returned. If the first argument is 1-dimensional, a 1 is prepended to its dimension for the purpose of the batched matrix multiply and removed after. If the second argument is 1-dimensional, a 1 is appended to its dimension for the purpose of the batched matrix multiple and removed after. The non-matrix (i.e. batch) dimensions are broadcasted (and thus must be broadcastable). For example, if `input` is a $(j \times 1 \times n \times m)$ tensor and `other` is a $(k \times m \times p)$ tensor, `out` will be an $(j \times k \times n \times p)$ tensor.

nn.Sequential

Containers

`Module`	Base class for all neural network modules.
`Sequential`	A sequential container.
`ModuleList`	Holds submodules in a list.
`ModuleDict`	Holds submodules in a dictionary.
`ParameterList`	Holds parameters in a list.
`ParameterDict`	Holds parameters in a dictionary.

nn.Sequential

```
hi = nn.Sequential(1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-18-a0c3571d098d> in <module>()
----> 1 hi = nn.Sequential(1)

                                  1 frames
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/module.py in add_module(self, name, module)
    343         if not isinstance(module, Module) and module is not None:
    344             raise TypeError("{} is not a Module subclass".format(
--> 345                 torch.typename(module)))
    346         elif not isinstance(name, torch._six.string_classes):
    347             raise TypeError("module name should be a string. Got {}".format(

TypeError: int is not a Module subclass

SEARCH STACK OVERFLOW
```

```
[19] hi = nn.Sequential(nn.Linear(11,11))
```

CLASS `torch.nn.Sequential(*args: Any)` [SOURCE]

A sequential container. Modules will be added to it in the order they are passed in the constructor. Alternatively, an ordered dict of modules can also be passed in.

To make it easier to understand, here is a small example:

```
# Example of using Sequential
model = nn.Sequential(
          nn.Conv2d(1,20,5),
          nn.ReLU(),
          nn.Conv2d(20,64,5),
          nn.ReLU()
        )

# Example of using Sequential with OrderedDict
model = nn.Sequential(OrderedDict([
          ('conv1', nn.Conv2d(1,20,5)),
          ('relu1', nn.ReLU()),
          ('conv2', nn.Conv2d(20,64,5)),
          ('relu2', nn.ReLU())
        ]))
```

```
[docs]class Sequential(Module):
```

```
@overload
def __init__(self, *args: Module) -> None:
    ...

@overload
def __init__(self, arg: 'OrderedDict[str, Module]') -> None:
    ...

def __init__(self, *args: Any):
    super(Sequential, self).__init__()
    if len(args) == 1 and isinstance(args[0], OrderedDict):
        for key, module in args[0].items():
            self.add_module(key, module)
    else:
        for idx, module in enumerate(args):
            self.add_module(str(idx), module)

def _get_item_by_idx(self, iterator, idx):
    """Get the idx-th item of the iterator"""
    size = len(self)
    idx = operator.index(idx)
    if not -size <= idx < size:
        raise IndexError('index {} is out of range'.format(idx))
    idx %= size
    return next(islice(iterator, idx, None))
```

```
    @_copy_to_script_wrapper
    def __getitem__(self: T, idx) -> T:
        if isinstance(idx, slice):
            return
self.__class__(OrderedDict(list(self._modules.items())[idx]))
        else:
            return self._get_item_by_idx(self._modules.values(),
idx)

    def __setitem__(self, idx: int, module: Module) -> None:
        key = self._get_item_by_idx(self._modules.keys(), idx)
        return setattr(self, key, module)

    def __delitem__(self, idx: Union[slice, int]) -> None:
        if isinstance(idx, slice):
            for key in list(self._modules.keys())[idx]:
                delattr(self, key)
        else:
            key = self._get_item_by_idx(self._modules.keys(), idx)
            delattr(self, key)

    @_copy_to_script_wrapper
    def __len__(self) -> int:
        return len(self._modules)
```

```
    @_copy_to_script_wrapper
    def __dir__(self):
        keys = super(Sequential, self).__dir__()
        keys = [key for key in keys if not key.isdigit()]
        return keys

    @_copy_to_script_wrapper
    def __iter__(self) -> Iterator[Module]:
        return iter(self._modules.values())

    # NB: We can't really type check this function as the type of
input
    # may change dynamically (as is tested in
    # TestScript.test_sequential_intermediary_types).  Cannot
annotate
    # with Any as TorchScript expects a more precise type
    def forward(self, input):
        for module in self:
            input = module(input)
        return input
```

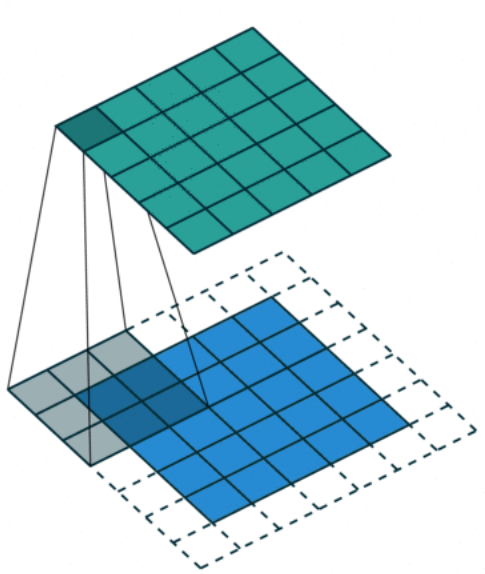
- Len(self) ?? 뭐죠.. 아마도 Sequential.foward일 때 call 하는 객체?
- Sequential._get_item_by_idx도 마찬가지..
- 함수 자체를 호출하는 주체

```
class Module:
    """Base
    def __init__(self):
        """
        Initializes internal Module state, shared by both nn.Module
and ScriptModule.
        """
        torch._C._log_api_usage_once("python.nn_module")

        self.training = True
        self._parameters = OrderedDict()
        self._buffers = OrderedDict()
        self._non_persistent_buffers_set = set()
        self._backward_hooks = OrderedDict()
        self._forward_hooks = OrderedDict()
        self._forward_pre_hooks = OrderedDict()
        self._state_dict_hooks = OrderedDict()
        self._load_state_dict_pre_hooks = OrderedDict()
        self._modules = OrderedDict()
```

Convolution 종류

Convolution

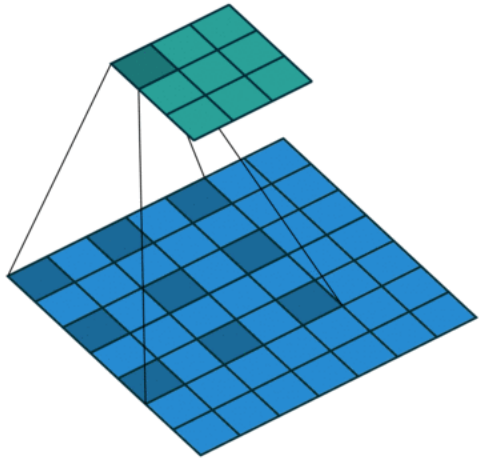


- 충분한 Contextual Information을 확보하기 위해 상대적으로 넓은 Receptive Field 고려할 필요 있음
- 연산량을 경량화시키면서 유의미한 정보를 추출하기 위해 다양한 Convolution 기법 등장

- 파라미터 수: CK^2M
- 연산량: CK^2MHW

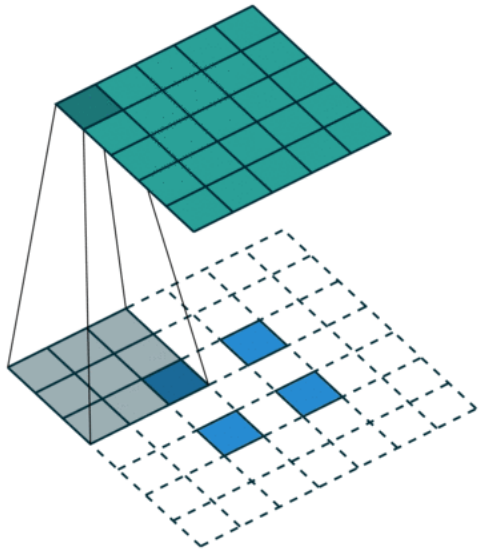
W: input의 width
H: input의 height
C: input의 channel
K: kernel(filter)의 크기
M: output의 channel (filter 개수)

Dilated Convolution (atrous convolution)



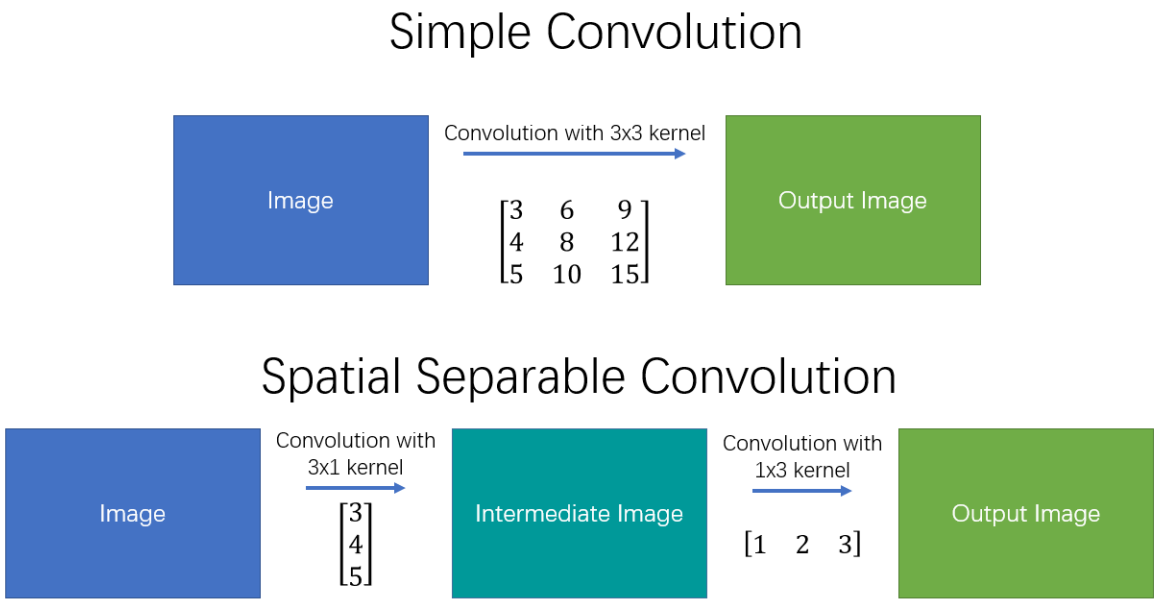
- 기존 컨볼루션 필터가 수용하는 픽셀 사이에 간격을 둔 형태
- Real-time segmentation에서 주로 사용
- 여러 컨볼루션이나 큰 커널을 사용할 여유가 없을 때 사용
- 필터 내부에 zero padding 추가해서 Receptive Field 늘림

Transposed Convolution (Deconvolution)



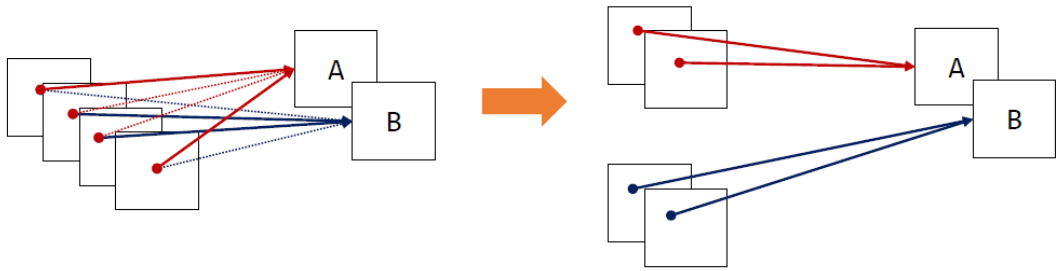
- 압축된 정보를 Up-sampling하여 복원
- 실제 Deconvolution과 공간 해상도는 같지만 수행되는 연산은 다름
- CNN을 사용한 Encoder-Decoder 구조의 Autoencoder
- 이미지 해상도 높이는 Super resolution

Seperable Convolution



- 커널 작업을 여러 단계로 나눠서 곱셈 연산 줄이는 방법
- 계산 복잡성이 줄어들고 네트워크를 더 빠르게 실행할 수 있음
- 예시: Sobel Mask
- 문제점: 모든 커널을 두 개의 작은 커널로 분리할 수는 없음

Grouped Convolution

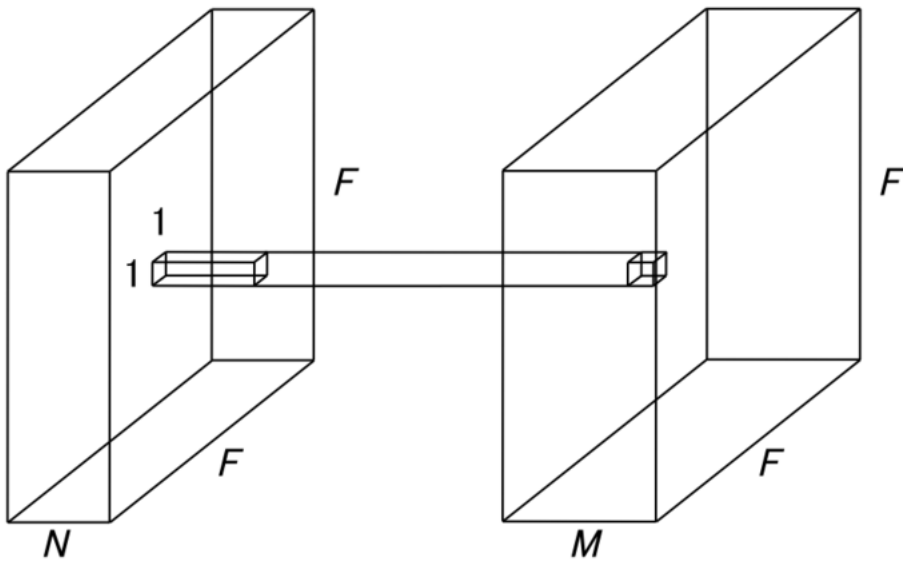


- 입력 값의 채널들을 여러 그룹으로 나누어 독립적으로 컨볼루션 연산 수행
- 병렬 처리에 유리
- 너무 많은 그룹으로 분할하면 오히려 성능이 떨어질 수 있음

- 파라미터 수: CK^2M/g
- 연산량: CK^2MHW/g

g: 그룹 개수

Pointwise Convolution

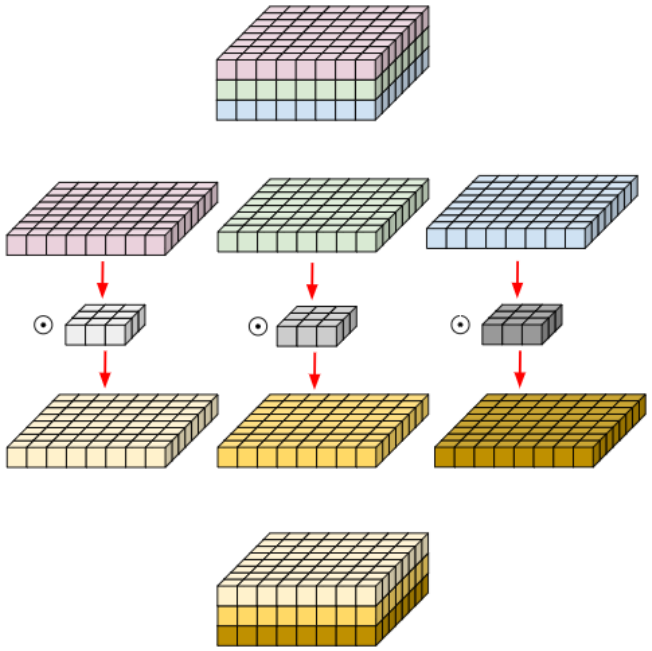


$$PointwiseConv(W, y)_{(i,j)} = \sum_{m}^{M} W_m \cdot y_{(i,j,m)}$$

- 파라미터 수: CM
- 연산량: CMHW

- 채널 방향의 컨볼루션만 진행
- Channel Reduction시에 주로 사용
- 커널 크기가 1X1로 고정
- 연산 속도가 대폭 향상되지만 데이터가 압축되면서 소실되는 문제도 발생
- Inception, Xception, SqueezeNet, MobileNet에 적용

Depthwise Convolution

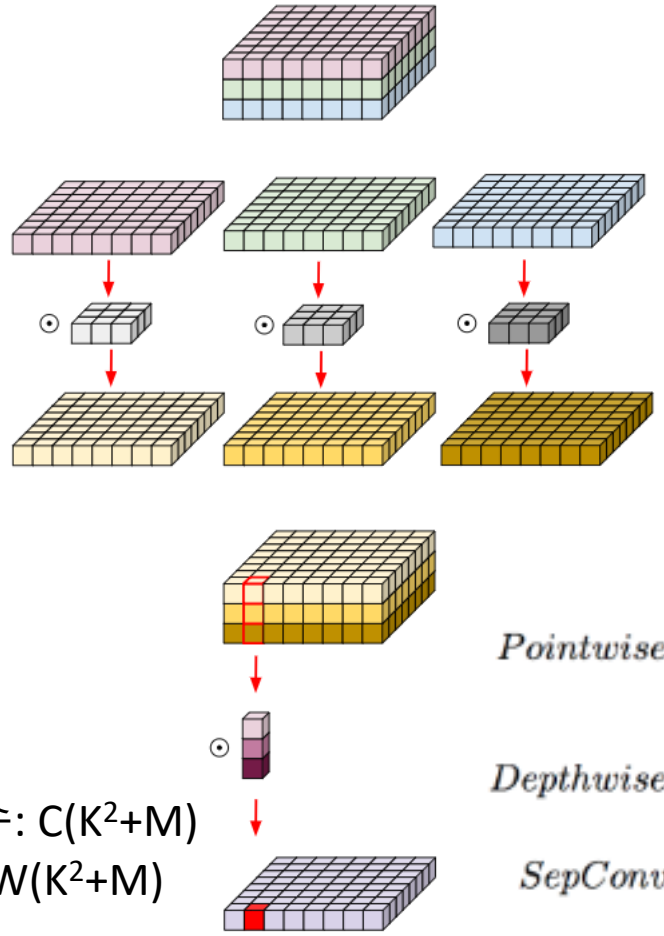


$$DepthwiseConv(W, y)_{(i,j)} = \sum_{k,l}^{K,L} W_{(k,l)} \odot y_{(i+k,j+l)}$$

- 파라미터 수: CK^2
- 연산량: CK^2HW

- 공간 방향의 컨볼루션만 진행
- 각 단일 채널에 대해서만 수행되는 필터 사용
- 입력 및 출력 채널 수 동일

Depthwise Seperable Convolution



- Depthwise + Pointwise
- 채널의 출력값이 하나로 합쳐짐
- Spatial Feature와 Channel-wise Feature를 모두 고려
- MobileNet에서 사용

$$PointwiseConv(W, y)_{(i,j)} = \sum_{m}^{M} W_m \cdot y_{(i,j,m)}$$

$$DepthwiseConv(W, y)_{(i,j)} = \sum_{k,l}^{K,L} W_{(k,l)} \odot y_{(i+k,j+l)}$$

$$SepConv(W_p, W_d, y)_{(i,j)} = PointwiseConv_{(i,j)}(W_p, DepthwiseConv_{(i,j)}(W_d, y))$$

- 파라미터 수: $C(K^2+M)$
- 연산량: $CHW(K^2+M)$

Deformable Convolution

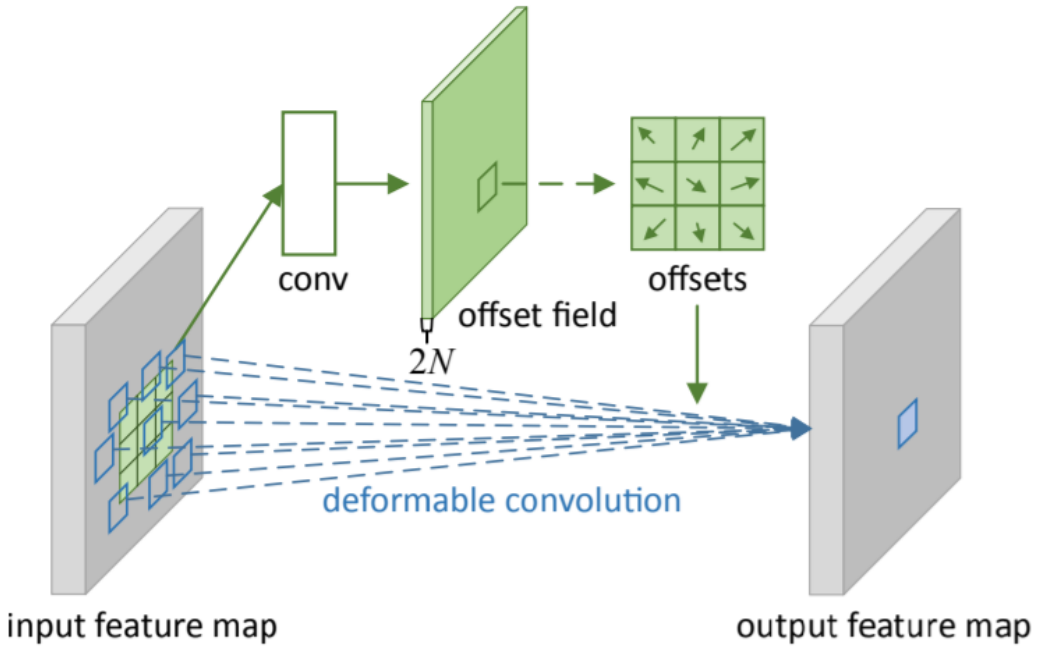


Figure 2: Illustration of 3×3 deformable convolution.

- 기존 CNN에서 사용하는 연산들은 기하학적으로 일정한 패턴을 가정하고 있기 때문에 복잡한 transformation에 유연하게 대처하기 어려움
- sampling grid에 2d offset을 더해 다양한 패턴으로 변화시킬 수 있음

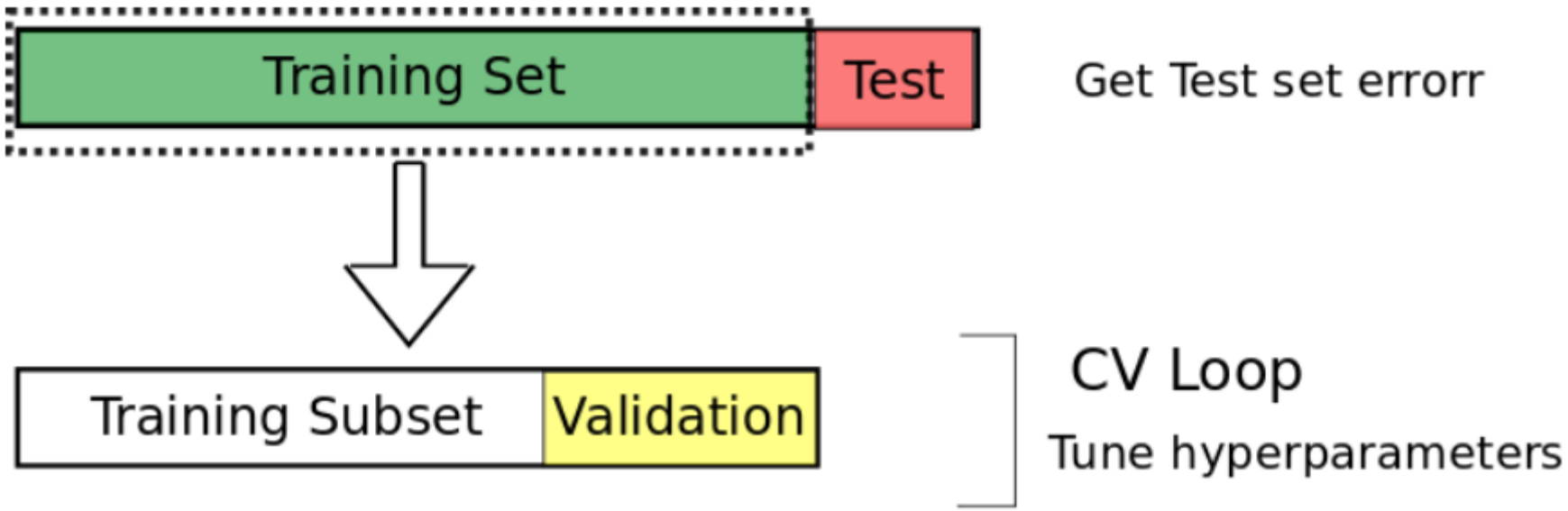
Reference

- Convolution 종류
  - https://eehoeskrap.tistory.com/431
  - https://sotudy.tistory.com/10
- Tensorflow, PyTorch 코드
  - https://underflow101.tistory.com/38 (Conv2D)
  - https://underflow101.tistory.com/40 (Conv2D 내부구조)
  - https://underflow101.tistory.com/42 (Depthwise, Pointwise, Depthwise Seperable)

CV on Time series data

Nested Cross-Validation

Cross-Validation



Time Series 에서는 왜 다른 CV를 사용?

- Temporal Dependencies
- Arbitrary Choice of Test Set

어떤 CV를 사용해야 할까?

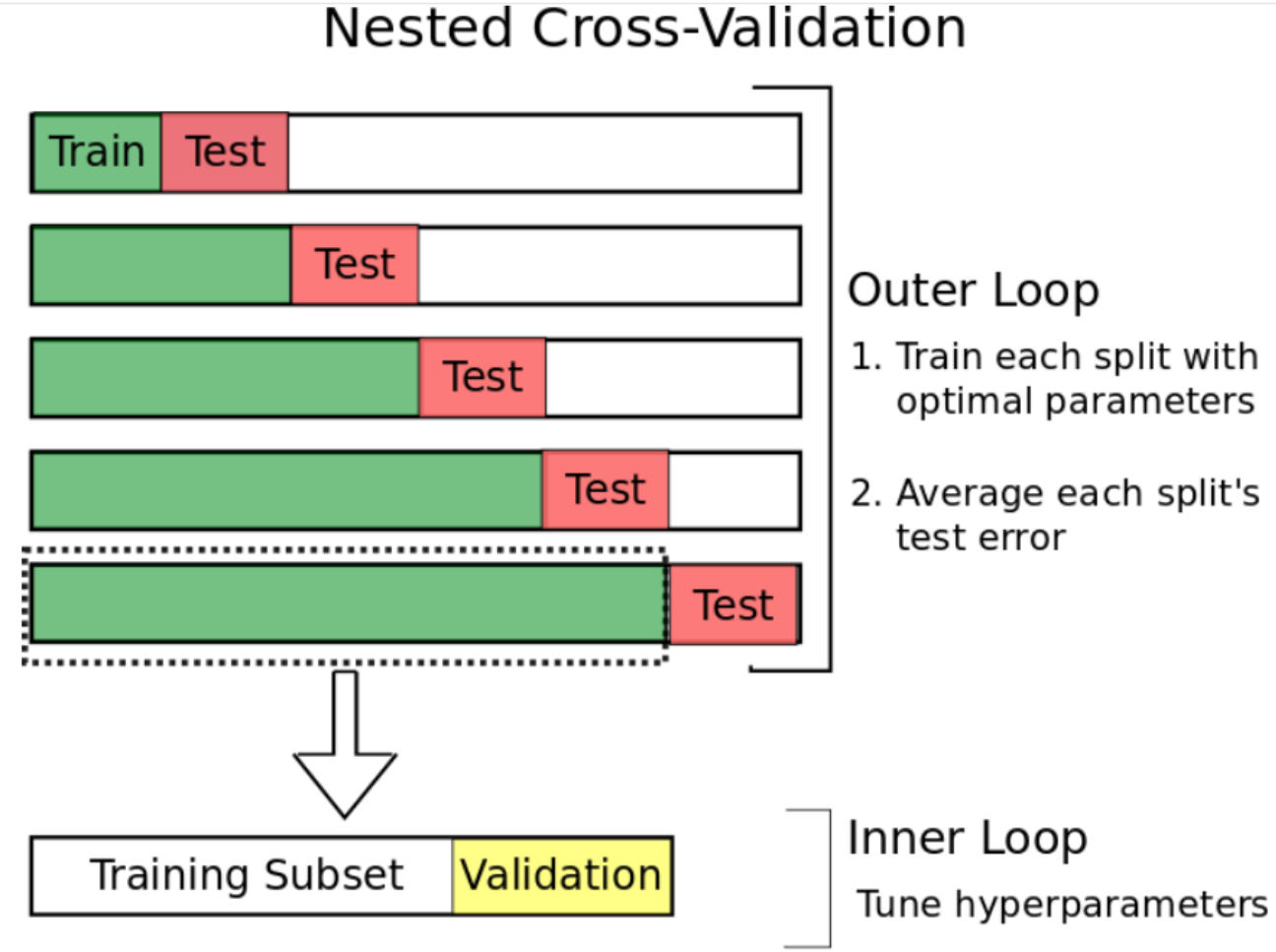


Figure 2: Nested CV Example

- Test나 Validation set은 Train 뒤에 chronologically 나와야 한다.
- Inner loop에서는 hyperparameters를 조정하고 outer loop에서는 inner loop에서 결정한 parameters를 가지고 학습을 하고 test error를 각각 구한다.

1. Predict Second Half

One shortcoming of the Predict Second Half nested cross-validation method is that the arbitrary choice of the hold-out test set can produce biased estimates of prediction error on an independent test set.

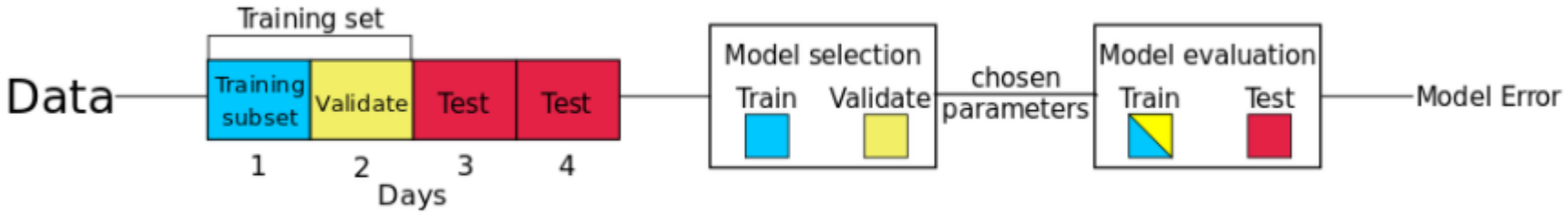


Figure 3: Predict Second Half Nested Cross-Validation

1 train/test split.
Easy to implement
반반 나눠서 앞은 training set 뒤에는 test set.
Train set에서는 다시 validation으로 나눈다.
that the validation set is chronologically subsequent to the training subset.

2. Day Forward-Chaining

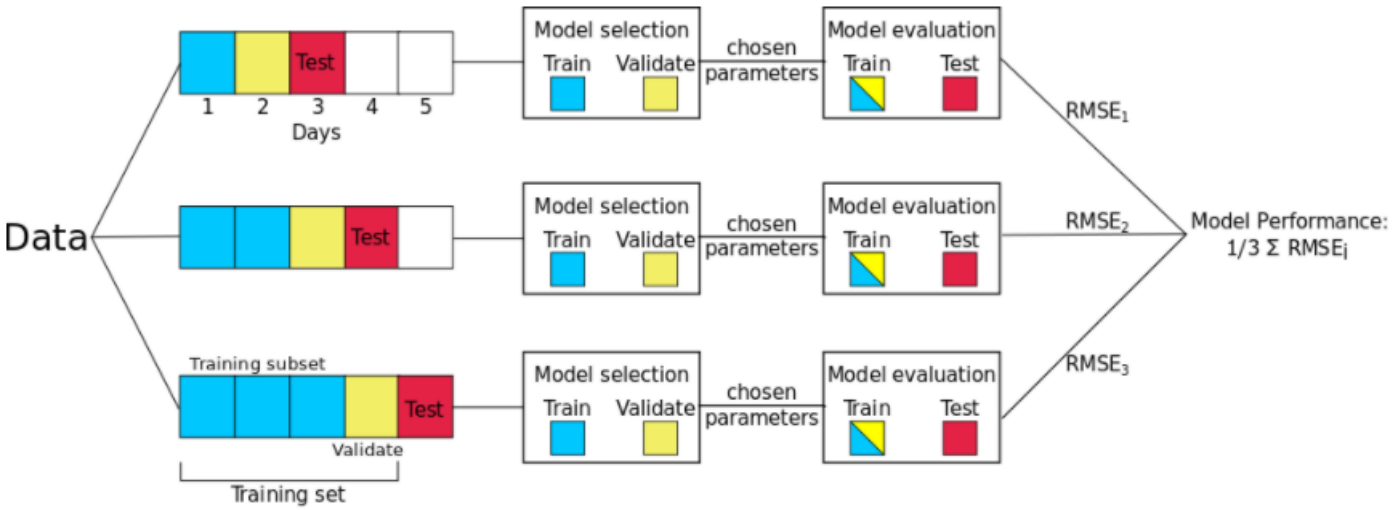


Figure 4: Day Forward-Chaining Nested Cross-Validation

- Using this method, we successively consider each day as the test set and assign all previous data into the training set
- This method produces many different train/test splits and the error on each split is averaged in order to compute a robust estimate of the model error.

Regular & Population-informed

- multiple different time series.

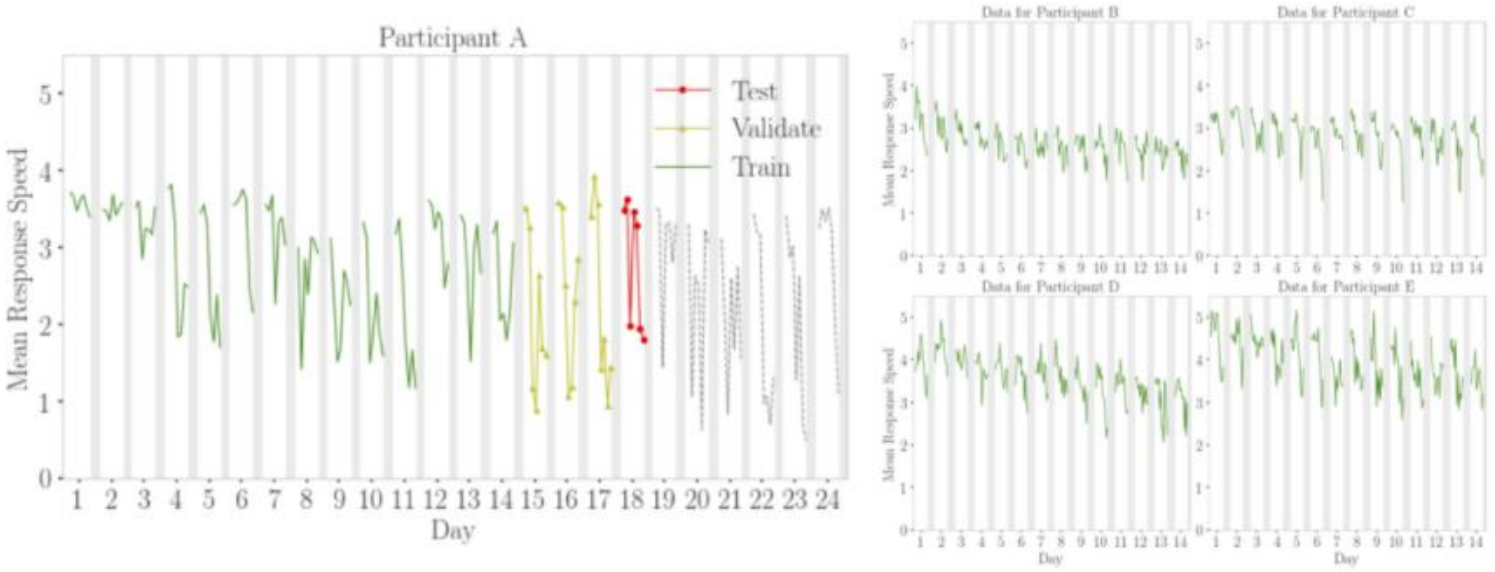


Figure 5: Population-Informed Day Forward-Chaining, where, in addition to Day Forward-Chaining method (left subfigure) for Participant A, we also allow all other participants' data to be in the training set (right subfigure). Note that the grey bars indicate when the participant was sleeping.

Method	Splits	Positives	Negatives
Regular Predict Second Half	1	• Easy to implement • Produces a single model • Computationally inexpensive	• Arbitrary test set choice could produce biased estimate of independent test set error
Population-Informed Predict Second Half	p	• More unbiased estimate of error compared to Regular Predict Second Half • Computational expense on order of number of participants	• Choice of test sets is still fairly arbitrary • Multiple models can make model inspection and interpretation difficult
Regular Day Forward-Chaining	d	• More splits • Can inspect how model fares on different days	• Requires consistent number of days in data for each participant • Multiple models
Population-Informed Day Forward-Chaining	dp	• Most unbiased estimate of error versus other methods • Can inspect how model fares on different days	• May be very computationally expensive • Multiple models

Reference

- Time Series Nested Cross-Validation | by Courtney Cochrane | Towards Data Science

K-fold Cross Validation

Hyperparameters & Validation

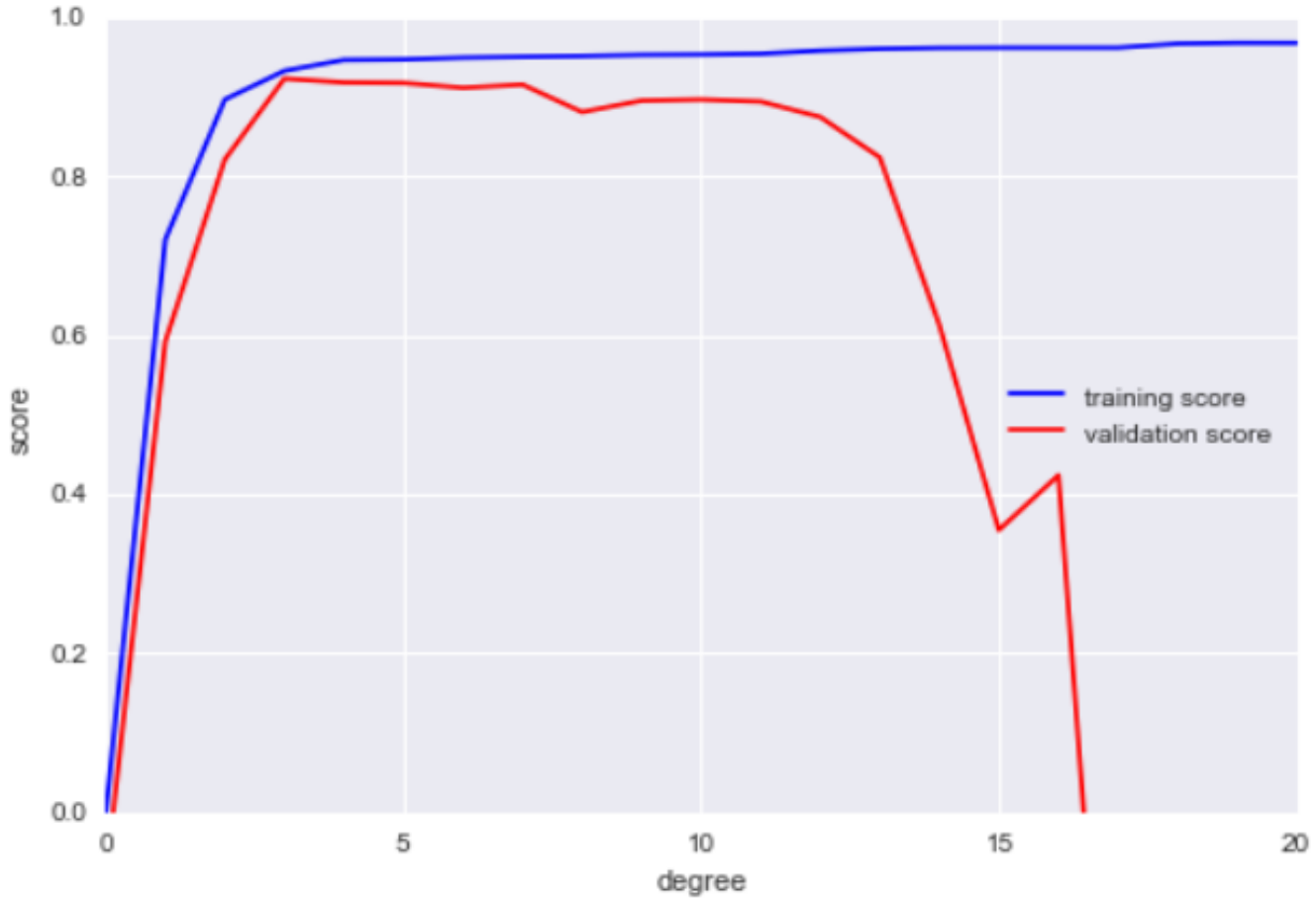
K-fold CV

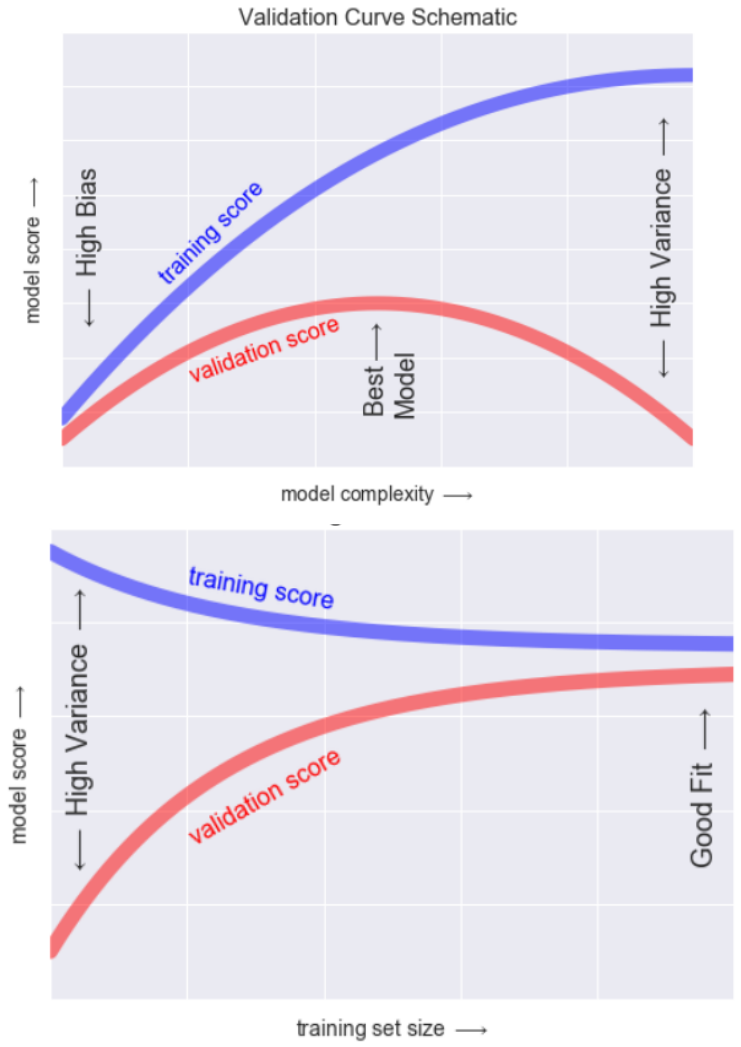
Pytorch usage

Hyperparameters & Validation

- Our model and Our hyperparameters are a good fit to to the data.
- 직관적으로, 주어진 학습 데이터셋에 대해 여러 모델과 하이퍼 파라미터를 설정 후 적용해보고 metric을 비교해보면서 best fit 를 찾아가는 또는 입증하는 과정.
- Hyperparameter가 늘어날 수록 검증 과정의 복잡도는 기하급수적으로 증가한다. (변수 간 독립적인 관계라면 그나마 낫지만 배제하기 힘든 상관관계가 존재할 시 난이도가 급상승)

Hyperparameters & Validation





→ 데이터셋만 확보되면, 사실 만사 오케이

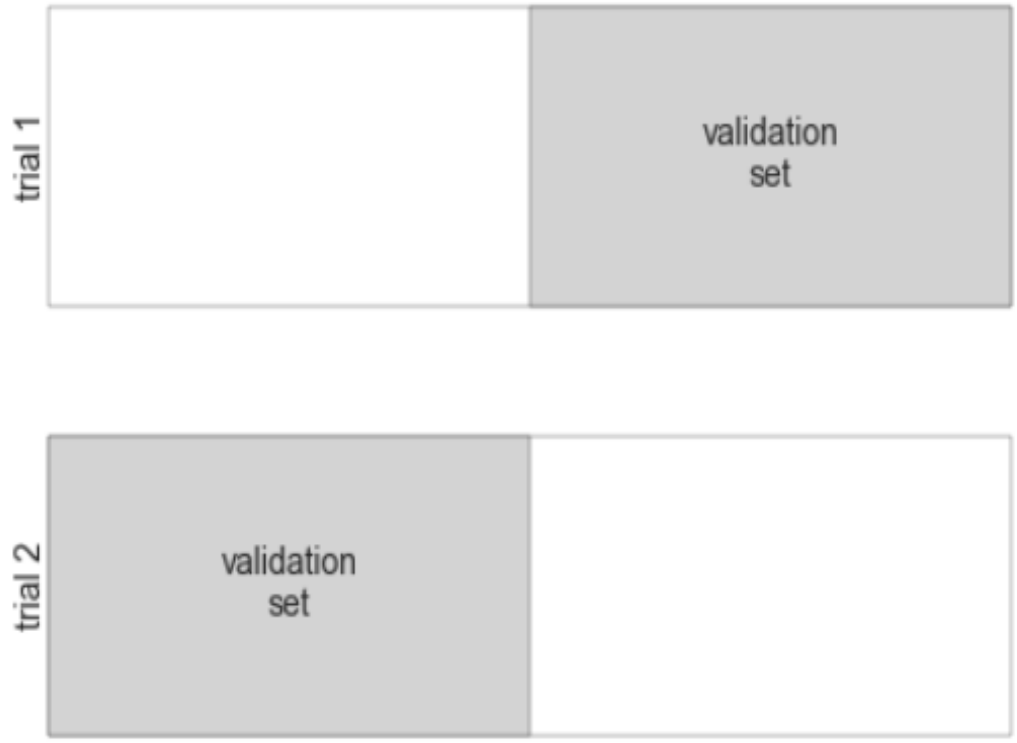
Hyperparameters & Validation

The challenge to validate

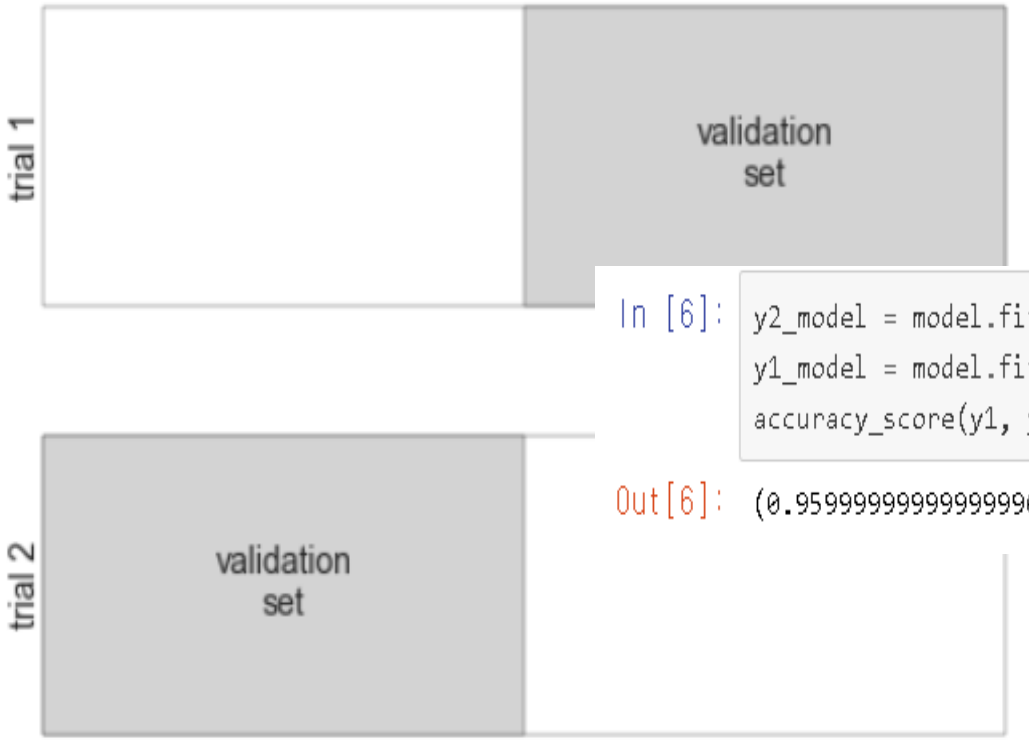
- Whole dataset : KNN을 예로, K=1로 설정하여 accuracy를 측정해보면 100%가 나올 수 있다. 데이터 샘플 자체는 다른 point이나 결국 동일한 데이터셋을 학습하고 검증했기에 'validate'에는 의미가 없다.
- Holdout set : 언급한 문제를 해결하기 위해 우리가 흔히 알고 있는 train / test 으로 데이터를 partition 한다. → train과 test의 적절한 비율 조정을 통한 분리(avoid overfitting-underfitting)는 우리의 귀중한 training set의 손실이 불가피하다.

Hyperparameters & Validation

- 옆 그림과 같이, 동일한 트레이닝 데이터셋의 일부를 validation Set으로 분리하고 이를 선정하는 방식을 rotation이 되게 하면 어떨까? == holdset의 단점이었던 데이터셋의 효용성의 한계를 어느정도 해결할 수 있다.



K-fold CV




```
In [6]: y2_model = model.fit(X1, y1).predict(X2)
        y1_model = model.fit(X2, y2).predict(X1)
        accuracy_score(y1, y1_model), accuracy_score(y2, y2_model)

Out[6]: (0.9599999999999996, 0.90666666666666662)
```

2-fold Cross Validation

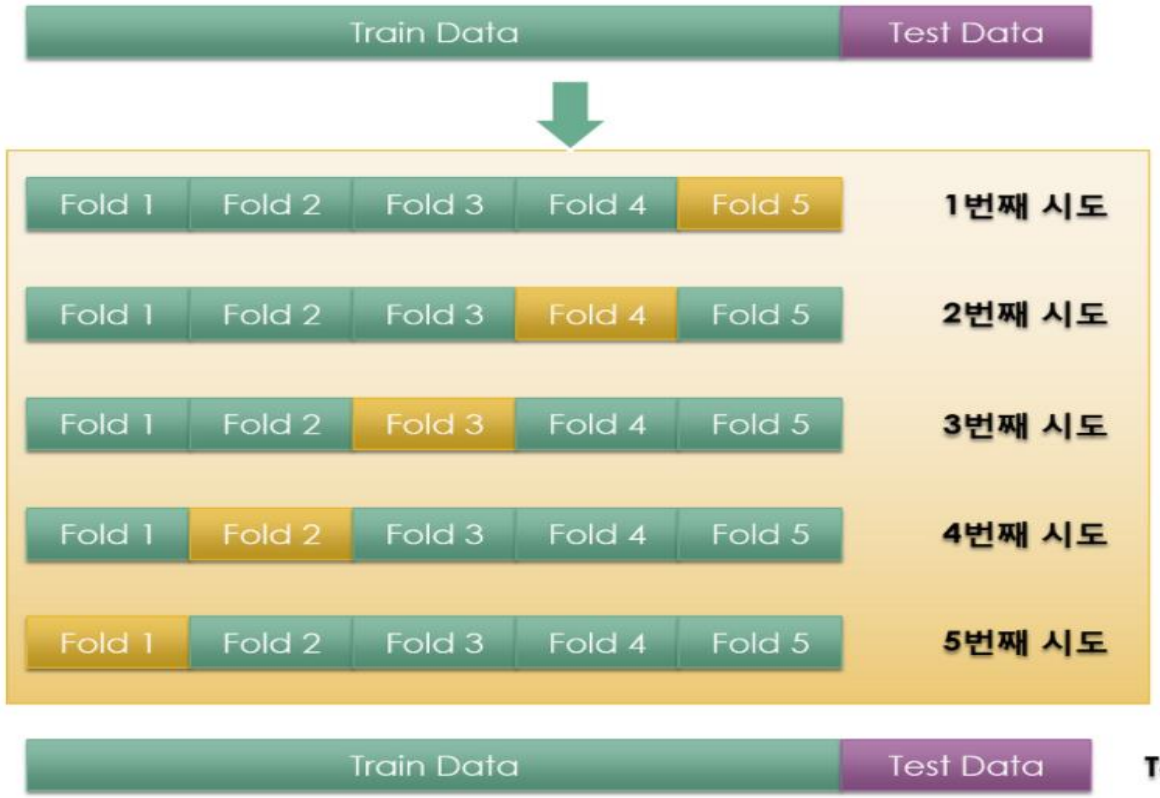
K-fold CV


```
In [7]: from sklearn.cross_validation import cross_val_score
        cross_val_score(model, X, y, cv=5)

Out[7]: array([ 0.96666667,  0.96666667,  0.93333333,  0.93333333,  1.        ])
```

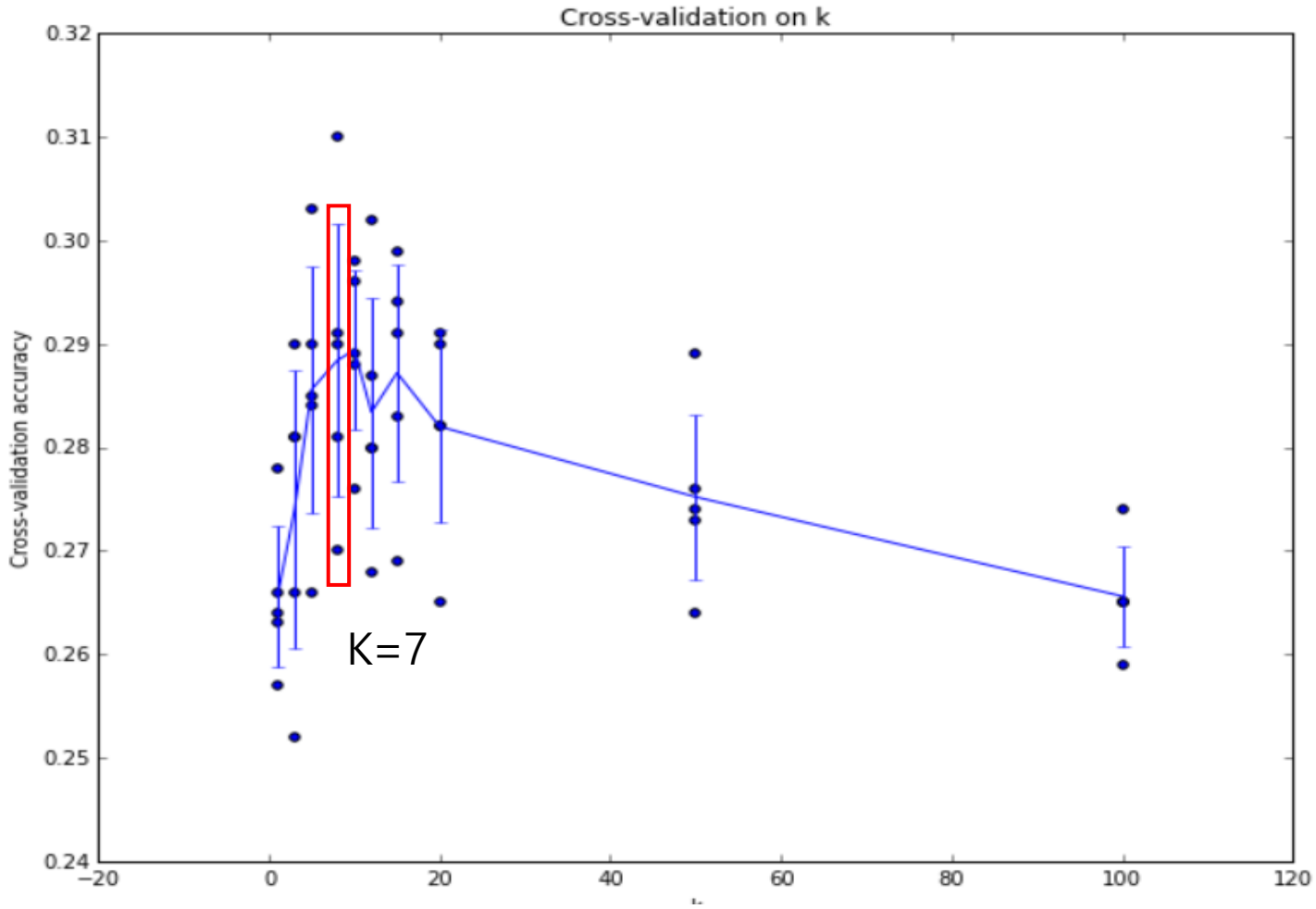
5-fold Cross Validation

K-fold CV



- 1~5번 trial에 각각 사전에 정한 검증 범위와 level에 따라 hyperparameter를 달리 설정.
- 6번째 trial에서, 지금까지 validation 한 hyper parameters의 평균(또는 다른 통계치)을 최종적인 hyper parameter로 설정 후 test data를 가지고 검증한다.

K-fold CV ; KNN Algorithm, Tuning K



Pytorch Usage

Import data

- Import epileptic seizure data from UCI ML repository
- Split train and test data using `random_split()`
- Train logistic regression model with training data and evaluate results with test data

:
```
class SeizureDataset(torch.utils.data.Dataset):
  def __init__(self):
    # import and initialize dataset
    df = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/00388/data.csv")
    df = df[df.columns[1:]]

    self.X = df[df.columns[:-1]].values
    self.Y = df["y"].astype("category").cat.codes.values.astype(np.int32)

  def __getitem__(self, idx):
    # get item by index
    return self.X[idx], self.Y[idx]

  def __len__(self):
    # returns length of data
    return len(self.X)
```

:
```
seizuredataset = SeizureDataset()
```

:
```
NUM_INSTANCES = len(seizuredataset)
TEST_RATIO = 0.3
TEST_SIZE = int(NUM_INSTANCES * 0.3)
TRAIN_SIZE = NUM_INSTANCES - TEST_SIZE

print(NUM_INSTANCES, TRAIN_SIZE, TEST_SIZE)
```

```
11500 8050 3450
```

:
```
train_data, test_data = torch.utils.data.random_split(seizuredataset, (TRAIN_SIZE, TEST_SIZE))

print(len(train_data), len(test_data))
```

```
8050 3450
```

Pytorch Usage

```
!pip install -U skorch
```

```
Collecting skorch
  Downloading https://files.pythonhosted.org/packages/49/61/d0949b994b8e1faa7c0218c45e94034d6ebf1c4fd87e99663eddfe761e95/skorch-0.3.0-py3-none-any.whl (89kB)
    100% |████████████████████████████████| 92kB 4.5MB/s
Requirement already satisfied, skipping upgrade: scipy in /usr/local/lib/python3.6/dist-packages (from skorch) (0.19.1)
Collecting tabulate (from skorch)
  Downloading https://files.pythonhosted.org/packages/12/c2/11d6845db5edf1295bc08b2f488cf5937806586afe42936c3f34c097ebdc/tabulate-0.8.2.tar.gz (45kB)
    100% |████████████████████████████████| 51kB 5.3MB/s
Requirement already satisfied, skipping upgrade: scikit-learn>=0.18 in /usr/local/lib/python3.6/dist-packages (from skorch) (0.19.2)
Requirement already satisfied, skipping upgrade: tqdm in /usr/local/lib/python3.6/dist-packages (from skorch) (4.26.0)
Requirement already satisfied, skipping upgrade: numpy in /usr/local/lib/python3.6/dist-packages (from skorch) (1.14.6)
Building wheels for collected packages: tabulate
  Running setup.py bdist_wheel for tabulate ... - ₩ done
  Stored in directory: /root/.cache/pip/wheels/2a/85/33/2f6da85d5f10614cbe5a625eab3b3aebfdf43e7b857f25f829
Successfully built tabulate
Installing collected packages: tabulate, skorch
Successfully installed skorch-0.3.0 tabulate-0.8.2
```

```
from skorch import NeuralNetClassifier
from sklearn.model_selection import cross_val_score
```

```
# generate skorch high-level classifier and perform 5-fold cross validation using cross_val_score()
logistic = NeuralNetClassifier(model, max_epochs = 10, lr = 1e-2)
scores = cross_val_score(logistic, X_data, y_data, cv = 5, scoring = "accuracy")
```

Reference

- https://jakevdp.github.io/PythonDataScienceHandbook/05.03-hyperparameters-and-model-validation.html , Hyperparameters and Model Validation
- https://cinema4dr12.tistory.com/1275 , 교차검증 (Cross-validation)
- https://github.com/buomsoo-kim/PyTorch-learners-tutorial , pytorch & CV