

KUBIG 22-1 학기중 스터디

# Dive into DeepLearning

팀장: 10기 조규선

스터디원:

16기 이수찬, 16기 이영노, 16기 임채명,

16기 신인섭, 16기 민윤기, 16기 윤지현, 16기 하예은

16기 유우혁, 16기 임정준, 16기 천원준

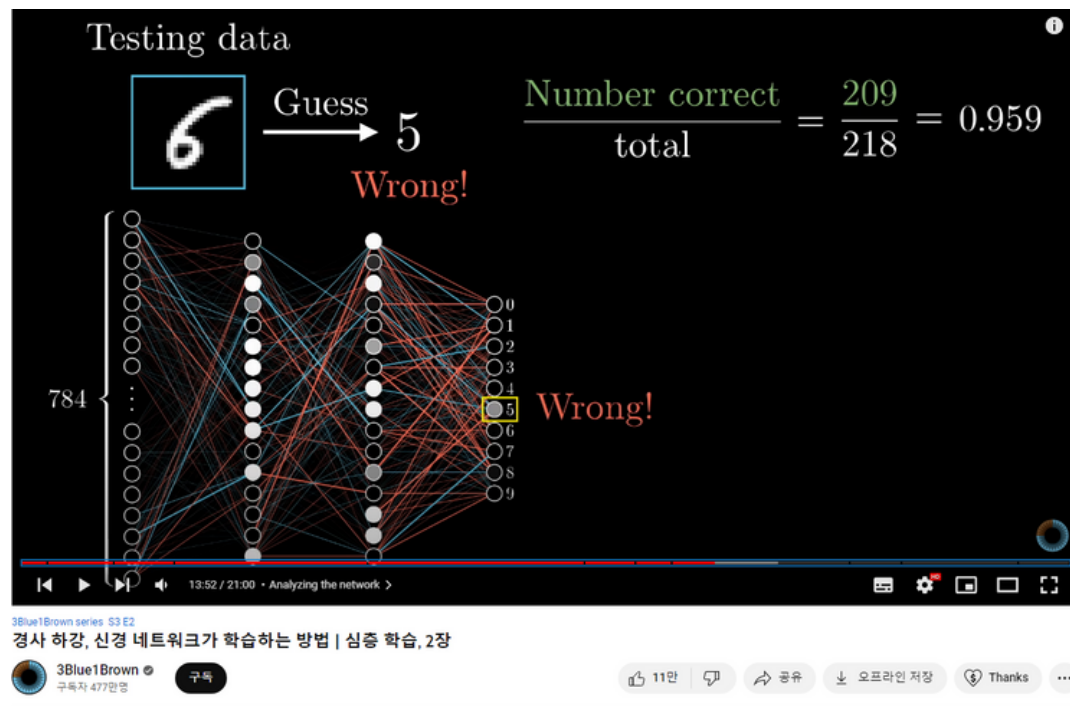
# 진행방식

```
for number, (image, y_true) in enumerate(train_loader):
    number += 1
    # image n * 1 * 28 * 28
    image = image.view(64, 784)
    y_pred = MLP(image)
    loss = criterion(y_pred, y_true)

    predicted = torch.max(y_pred.data, 1)[1]
    batch_corr = (predicted == y_true).sum()
    trn_corr += batch_corr

    # Update parameters
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Print Current Loss
    if batch_size == 0:
        print(f'epoch')
```



머리 빛는 네오

0203: Dataset and Dataloader  
<https://youtu.be/bhf0-PCYeqA>  
0204: Building a Neural Network  
<https://youtu.be/su0pt9YWN3M>  
0205: Training a Neural Network  
<https://youtu.be/bfpftNkl7ZQ>

과제 설명:

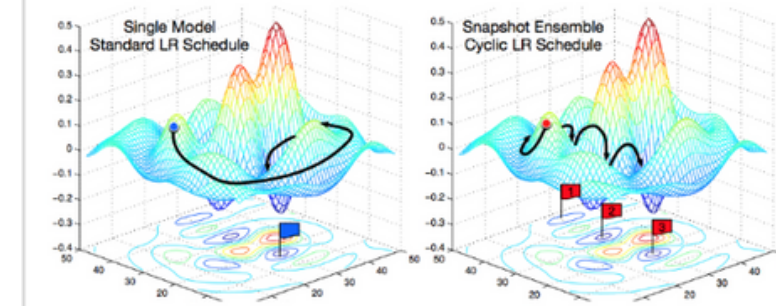
1. Python Classes 에 대해서 정리하여 소개해주세요. <https://docs.python.org/3/tutorial/classes.html>
2. 주요 최적화 알고리즘에 대해서 비교 설명을해주세요: Gradient Descent, Stochastic Gradient Descent, SGD with momentum, Mini-Batch Gradient Descent, AdaGrad, RMSProp, AdaDelta, Adam
3. torch.optim 에서 사용하는 Learning Rate 에 대해서 설명하고, LR.scheduler에 대...

전체보기

1

## 10. CosineAnnealingWarmRestarts

[Exploring Stochastic Gradient Descent with Restarts \(SGDR\)](#) 참고



loss function이 복잡해질때 local minimum을 global minimum이라고 규정할 수 있는 오류가 있음.

이러한 점에서 Gradient Descent < Stochastic Gradient Descent 가 장점을 가짐.

비슷한 motivation으로, local minimum에서 빠져나올 수 있게 하기 위한 방법으로 Stochastic Gradient Descent with 'Restart'를 사용할. 학습률을 다시 restart시켜서 또다른 local minimum을 찾고, 이를 이전 local minimum 들과 비교하여 global minimum을 찾는 정확성을 높여줌.

```
[ ] import torch
import matplotlib.pyplot as plt

model = torch.nn.Linear(2, 1)
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
lr_scheduler = torch.optim.lr_scheduler.CosineAnnealingWarmRestarts(optimizer, T_0=10, T_mult=1, eta_min=0.001, last_epoch=-1)

lrs = []

for i in range(100):
```

## 2. 문제 발생 원인 분석

Backward, grad 함수 정의

backward

Computes the sum of gradients of given tensors with respect to graph leaves.

grad

Computes and returns the sum of gradients of outputs with respect to the inputs.



Gradient를 누적하는 성질을 가짐

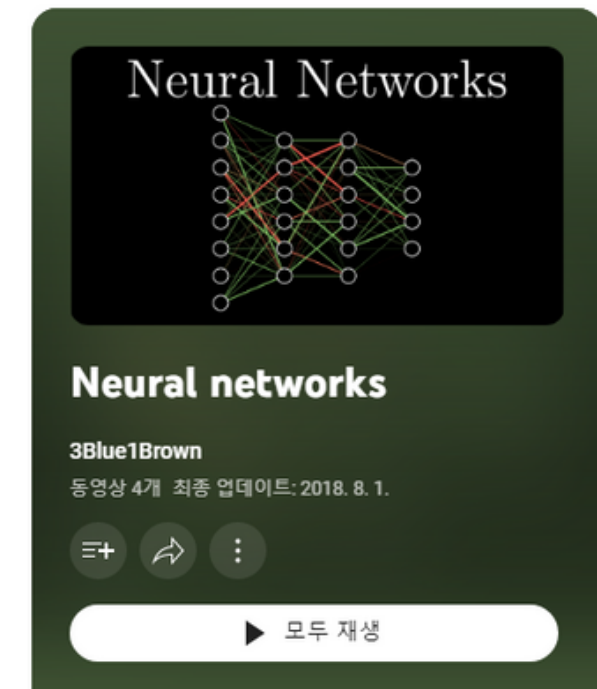
팀장님 제작 영상, 관련영상 시청 후 팀별로 과제 해결, 발표, 피드백 +  
팀장님 추가 설명

# 1,2주차: 딥러닝 개괄

시청 영상:

3Blue1Brown <Neural Networks> 영상 4개  
Introduction to DeepLearning

내용: 머신러닝과 AI의 역사, 머신러닝과 딥러닝 차이,  
뉴럴 네트워크란, NN이 학습하는 방법,  
Stochastic Gradient Descent, Back propagation



Neural Networks

3Blue1Brown

동영상 4개 최종 업데이트: 2018. 8. 1.

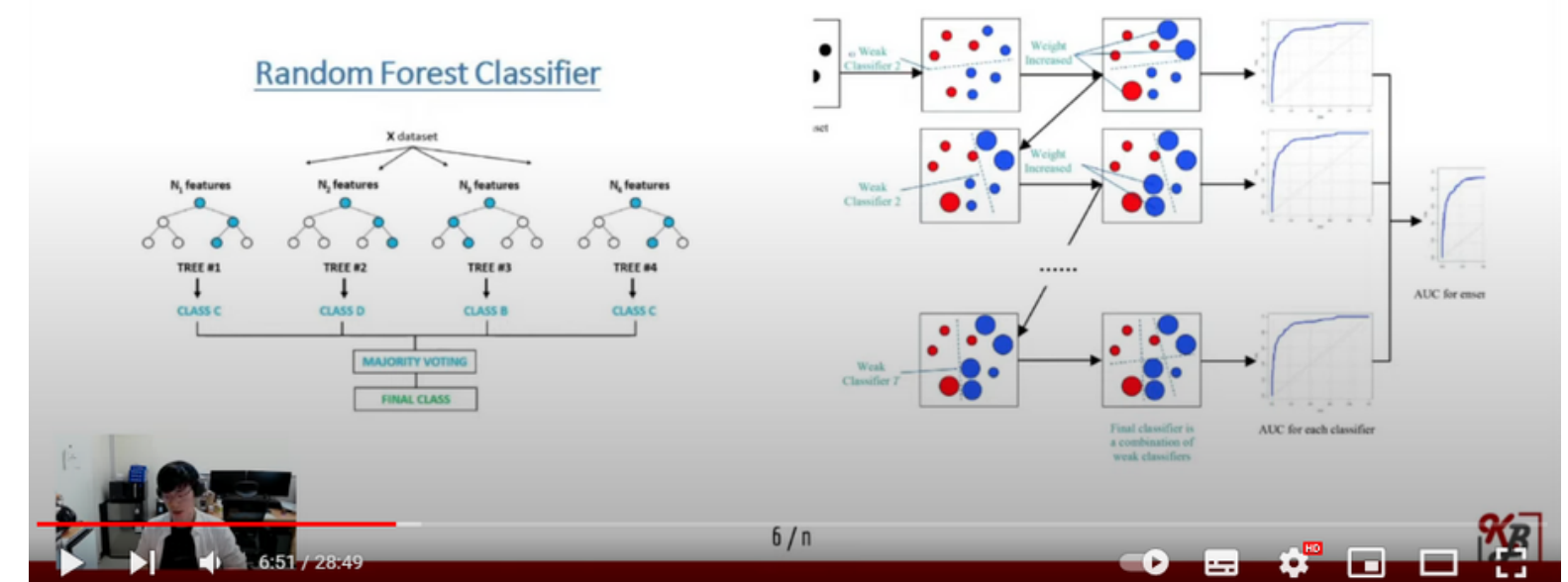
모두 재생

시즌 3 ▾

- 1 Neural Networks 신경망이란 무엇인가? | 1장. 딥러닝에 관하여 3Blue1Brown 19:13
- 2 How machines learn 경사 하강, 신경 네트워크가 학습하는 방법 | 심층 학습, 2장 3Blue1Brown 21:01
- 3 Backpropagation What is backpropagation really doing? | Chapter 3, Deep learning 3Blue1Brown 13:54
- 4 Backpropagation 역전파 미적분 | 심층 학습, 4장 3Blue1Brown 지금 재생 중

## Brief History of Machine Learning and Artificial Intelligence

### 4) Decision Tree, Random Forest, GBM (2010년~)





# 3주차: PYTORCH 개괄

시청 영상:

0201: Tensor Manipulation

0202: CUDA and Autograd

DeepLearning Zero to all-Tensor Manipulation

내용: PyTorch tensor manipulation, 여러 Operators  
chain rule, autograd(auto differentiation)

💡 torch.mm(), torch.bmm(), torch.mul(), torch.matmul(), \* operator , @ operator 쓰임새 비교

@[https://colab.research.google.com/drive/1nPgWwW\\_lviXVela9bgVYdnfplvNA5J5Q?usp=sharing](https://colab.research.google.com/drive/1nPgWwW_lviXVela9bgVYdnfplvNA5J5Q?usp=sharing)

torch.mm()

```
# torch.mm(input, mat2, *, out=None)
# Performs a matrix multiplication of the matrices input and mat2.
# input, mat2는 2-D tensor 즉, 행렬이어야 함.

mat1 = torch.randn(2, 3) # n x m
mat2 = torch.randn(3, 3) # m x p

torch.mm(mat1, mat2) # n x p
```

torch.bmm() - 행렬곱 동시에 병렬로 진행

[Reference]  
pytorch.github.io

```
def backward(
    tensors: TensorOrTensors,
    grad_tensors: Optional[TensorOrTensors] = None,
    retain_graph: Optional[bool] = None,
    create_graph: bool = False,
    grad_variables: Optional[TensorOrTensors] = None,
    inputs: Optional[TensorOrTensors] = None,
) -> None:
    """
    Backward pass (backward) of the autograd engine.
    """
    if torch._C._are_function_transforms_active():
        raise RuntimeError(
            "backward() called inside a function transform. This is not "
            "supported, please use torch.func.grad or torch.func.grad_catch "
            "or call backward() outside of function transforms."
        )
    if grad_variables is not None:
        warnings.warn("grad_variables is deprecated. Use 'grad_tensors' instead.")
    if grad_tensors is None:
        grad_tensors = grad_variables
    else:
        raise RuntimeError("grad_tensors and 'grad_variables' (deprecated) "
            "arguments both passed to backward(). Please only "
            "use 'grad_tensors'")
    if inputs is not None and len(inputs) == 0:
        raise RuntimeError("inputs argument to backward() cannot be empty.")
    tensors = (tensors,) if isinstance(tensors, torch.Tensor) else tuple(tensors)
    inputs = (inputs,) if isinstance(inputs, torch.Tensor) else tuple(
        inputs) if inputs is not None else tuple()
    grad_tensors = _tensor_or_tensors_to_tuple(grad_tensors, len(tensors))
    grad_tensors = _make_grads(tensors, grad_tensors, is_grads_batched=False)
    if retain_graph is None:
        retain_graph = create_graph
    # The reason we repeat same the comment below is that
    # some Python versions print out the first line of a multi-line function
    # calls in the traceback and some print out the last line
    Variable._execution_engine.run_backward(  # Calls into the C++ engine to run the backward pass
        tensors, grad_tensors, inputs, retain_graph, create_graph, allow_unreachable=True, accumulate_grad=True)  # Calls into the C++ engine to run the backward pass
```

def backward() Source Code  
Reading

Final Line에  
accumulate\_grad=True로  
Variable.\_execution\_engine.run\_  
backward() 함수에 인자로 전달  
그러나, backward 함수에는 이에  
대한 input 값이 없어서 torch  
package 사용 시 backward 알고리  
즘 자체의 수정은 어려움

9 / n

올고있는 어피치

과제 설명

1. torch.mm(), torch.bmm(), torch.mul(), torch.matmul(), \* operator , @ operator 의 쓰임새를 비교하여라.
2. PyTorch Automatic Differentiation에서 사용하는 chain rule에 대해 설명하고, non-scalar 값에 대한 미분은 어떻게 해야 하는지에 대해 설명하라. (D2L 설명 기반)
3. backward()를 하여도 .grad가 초기화되지 않아 여러번 backward()하면 grad가 쌓인다는 문제가 왜 발생하고, 어떻게 해결하여야 하는지에 대해 설명하라.

오후 3:19

올고있는 어피치

10~15분 발표를 각각 준비해오시면 됩니다.

오후 3:20

# 4,5주차: 기본적인 모델 구현

시청 영상:

0203: Dataset and Dataloader

0204: Building a Neural Network

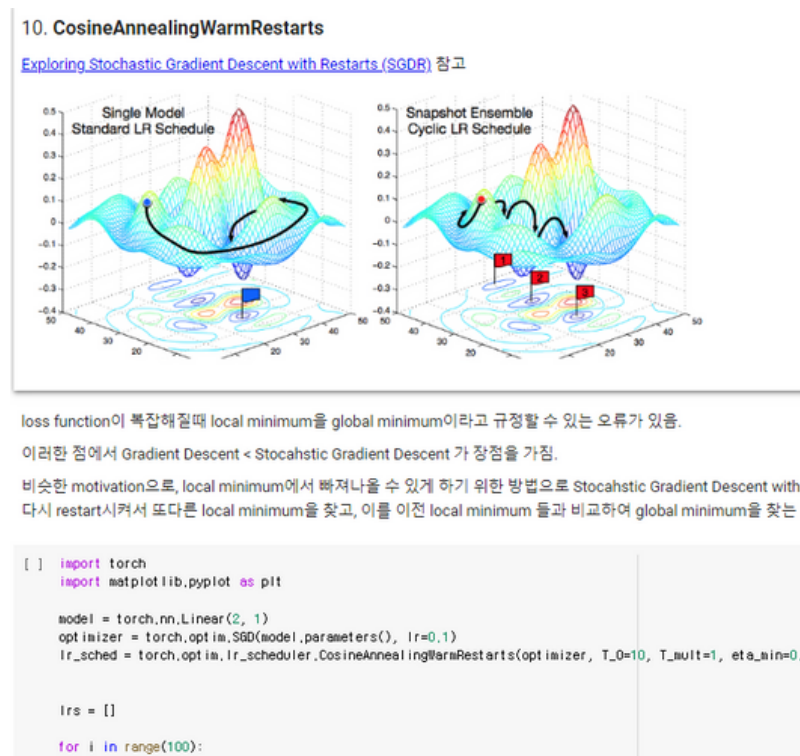
0205: Training a Neural Network

내용: python classes, dataloader 사용이유,  
NN 코드 구현, NN 학습 코드 구현,  
최적화 알고리즘, learning rate, lr scheduling

머리 빛는 네오

0203: Dataset and Dataloader  
<https://youtu.be/bhf0-PCYegA>  
0204: Building a Neural Network  
<https://youtu.be/su0pt9YWN3M>  
0205: Training a Neural Network  
<https://youtu.be/bfpftNkl7ZQ>

과제 설명:  
1. Python Classes 에 대해서 정리하여 소개  
해주세요. <https://docs.python.org/3/tutorial/classes.html>  
2. 주요 최적화 알고리즘에 대해서 비교 설명을  
해주세요: Gradient Descent, Stochastic  
Gradient Descent, SGD with  
momentum, Mini-Batch Gradient  
Descent, AdaGrad, RMSProp, AdaDelta,  
Adam  
3. torch.optim 에서 사용하는 Learning  
Rate 에 대해서 설명하고, LR.scheduler에  
대...



- Generators
- 본 발표에서는 Generator 개념보다는 **Generator** 표현식에 집중 !
- For문을 argument 내부에 삽입하여, Method 내에서 함수가 실행할 수 있게 하여, for문을 외부 Scope에서 번거롭게 쓰지 않아도, 해당 함수의 직관적인 iterating이 가능

## 5. Practical Analysis

```
class LinearRegression(MultiOutputMixin, RegressorMixin, LinearModel):

    def __init__(
        self,
        *,
        fit_intercept=True,
        normalize="deprecated",
        copy_X=True,
        n_jobs=None,
        positive=False,
    ):
        self.fit_intercept = fit_intercept
        self.normalize = normalize
        self.copy_X = copy_X
        self.n_jobs = n_jobs
        self.positive = positive

    def fit(self, X, y, sample_weight=None):
        _normalize = _deprecate_normalize(self.normalize, default=False, estimator_name=self.__class__.__name__)
        n_jobs = self.n_jobs
        accept_sparse = False if self.positive else ["csr", "csc", "coo"]
        X, y = self._validate_data(X, y, accept_sparse=accept_sparse, y_numeric=True, multi_output=True)
        sample_weight = _check_sample_weight(sample_weight, X, dtype=X.dtype, only_non_negative=True)
        X, y, X_offset, y_offset, X_scale = _preprocess_data(X, y, fit_intercept=self.fit_intercept,
                                                             normalize=_normalize,
                                                             sample_weight=sample_weight)

        # Sample weight can be implemented via a simple rescaling.
        X, y, sample_weight_sqrt = _rescale_data(X, y, sample_weight)

        if self.positive:
            if y.ndim < 2:
```

[Citation] [https://github.com/scikit-learn/scikit-learn/blob/36958fb24/sklearn/linear\\_model/\\_base.py#L529](https://github.com/scikit-learn/scikit-learn/blob/36958fb24/sklearn/linear_model/_base.py#L529)

검색 🔍

Class LinearRegression

- MultiOutputMixin, RegressorMixin, LinearModel 을 상속받은 Multiple Inherited Class
- Class가 호출되면 `__init__` function이 자동적으로 호출: intercept, normalize, n\_jobs 등을 정의
- 일반적으로 LinearRegression을 train시킬 때,

`lr=LinearRegression()`

`lr.fit(x_train, y_train)`

으로 많이 활용하는데, 이때 fit 함수는 본 Class의 Method이며, lr이라는 인스턴스에서 클래스 내의 fit 메소드를 호출한 것이 상기된 예제