KOREATECH
한국기술교육대학교

Advanced Techniques in Deep Learning

November 2023

http://link.koreatech.ac.kr

Diverse Techniques

◈1. Advanced Optimizers
- SGD with Momentum
- Adagrad
- RMSProp
- Adam

◈2. Regularization
- L1 Regularization
- L2 Regularization (Weight decay)

◈3. Dropout

◈4. Normalization
- Batch Normalization
- Layer Normalization

◈5. Data Augmentation

◈6. Learning Rate Decaying

Advanced Optimizers

[읽을거리]
https://velog.io/@yookyungkho/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EC%98%B5%ED%8B%B0%EB%A7%88%EC%9D%B4%EC%A0%80-%EC%A0%95%EB%B3%B5%EA%B8%B0%EB%B6%80%EC%A0%9C-CS231n-Lecture7-Review

Challenges

◈ Mini-batch-based stochastic gradient descent (SGD)

- θ_t: t번 업데이트 된 모델 파라미터
- L_i: i번째 샘플 데이터에 대한 오차 값
  - $i \in B$

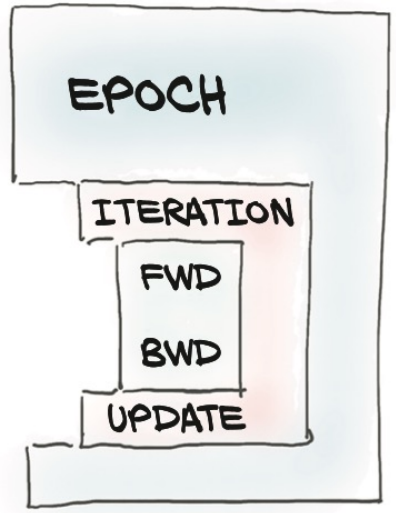
$$L_i(\theta_t)$$

$$\frac{\partial L_B(\theta_t)}{\partial \theta_t} = \sum_{i \in B} \frac{\partial L_i(\theta_t)}{\partial \theta_t}$$

$$\theta_{t+1} = \theta_t - \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}$$

where γ is learning rate

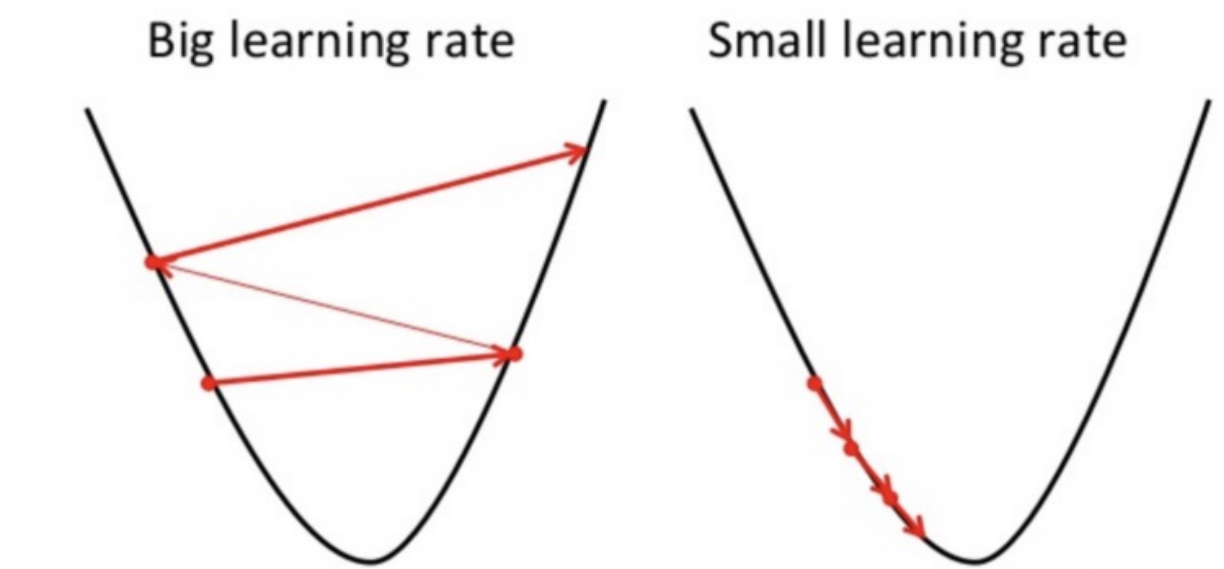
FOR N EPOCHS:
  SPLIT DATASET IN MINIBATCHES
  FOR EVERY MINIBATCH:
    WITH EVERY SAMPLE IN MINIBATCH:
      EVALUATE MODEL (FORWARD)
      COMPUTE LOSS
      ACCUMULATE GRADIENT OF LOSS (BACKWARD)
    UPDATE MODEL WITH ACCUMULATED GRADIENT



Challenges

◈Challenges on mini-batch-based stochastic gradient descent (SGD)

- Choosing a proper learning rate can be difficult
  - A learning rate that is too small → painfully slow convergence
  - A learning rate that is too large → hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge



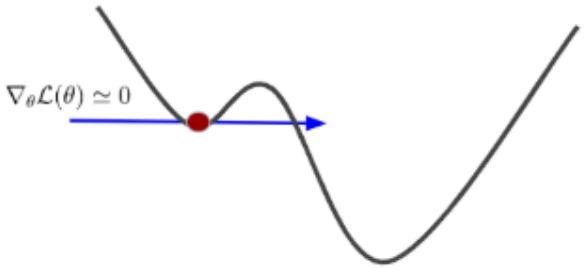
[Source] https://www.ruder.io/optimizing-gradient-descent/

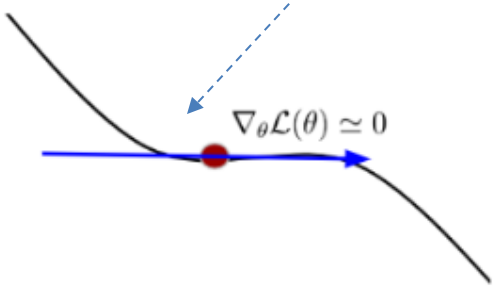
Challenges

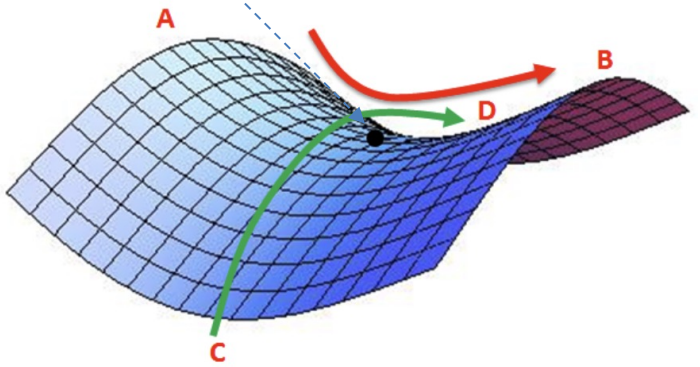
◈Challenges on mini-batch-based stochastic gradient descent (SGD)

- The same learning rate applies to all parameter updates
  - We might not want to update all of weights or biases to the same extent, but perform a larger update on weights or biases related to some features

- Getting trapped in numerous suboptimal local minima
  - The difficulty arises in fact not from local minima but from saddle points, i.e. points where one dimension slopes up and another slopes down.
  - These saddle points are usually surrounded by a plateau of the same error, which makes it notoriously hard for SGD to escape, as the gradient is close to zero in all dimensions







[Source] https://www.ruder.io/optimizing-gradient-descent/

Optimizer Evolution

◈Optimizer Evolution

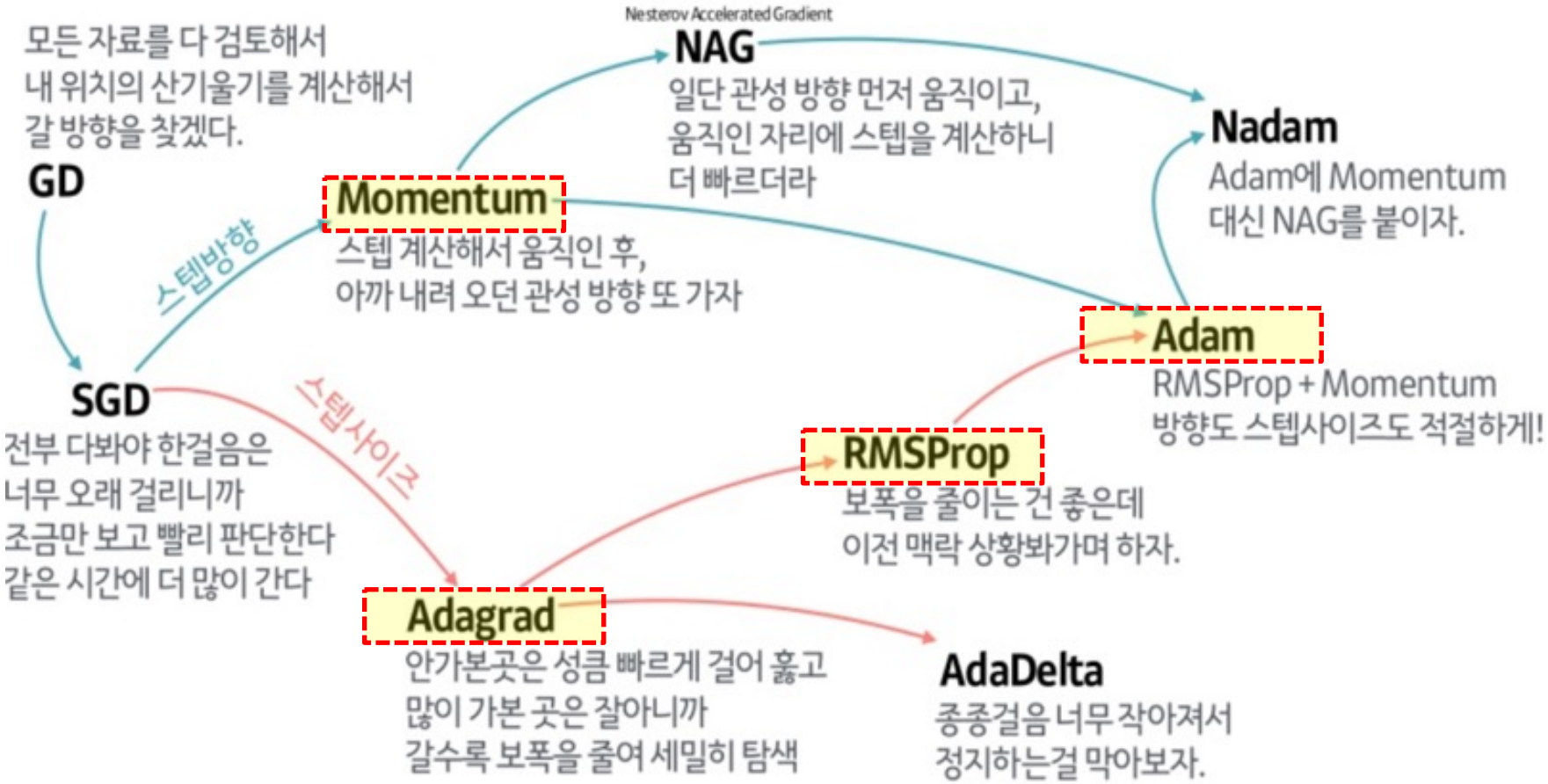
$$\theta_{t+1} = \theta_t - \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}$$ where γ is learning rate

- 업데이트의 크기(스텝 사이즈)를 올바르게 결정
- 업데이트의 방향(스텝 방향)을 올바르게 결정

[Reference: https://medium.com/@kaitotally/adam-the-birthchild-of-adagrad-and-rmsprop-b5308b24b9cd]

Optimizer Evolution

◈Optimizer Evolution



방향 (gradient 활용)

보폭 (gradient 제곱 활용)

[Reference: https://medium.com/@kaitotally/adam-the-birthchild-of-adagrad-and-rmsprop-b5308b24b9cd]

Momentum

◈SGD with Momentum

– 기존 SGD에 Momentum(관성) 벡터 업데이트 추가

SGD

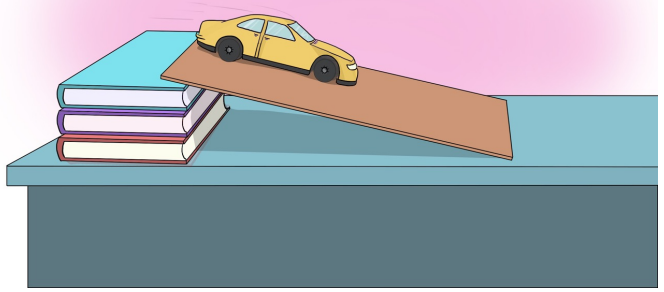
$$\theta_{t+1} = \theta_t - \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad t \geq 0$$

SGD with Momentum

$$v_{t+1} = \mu v_t + \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad v_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - v_{t+1}$$

Momentum 벡터

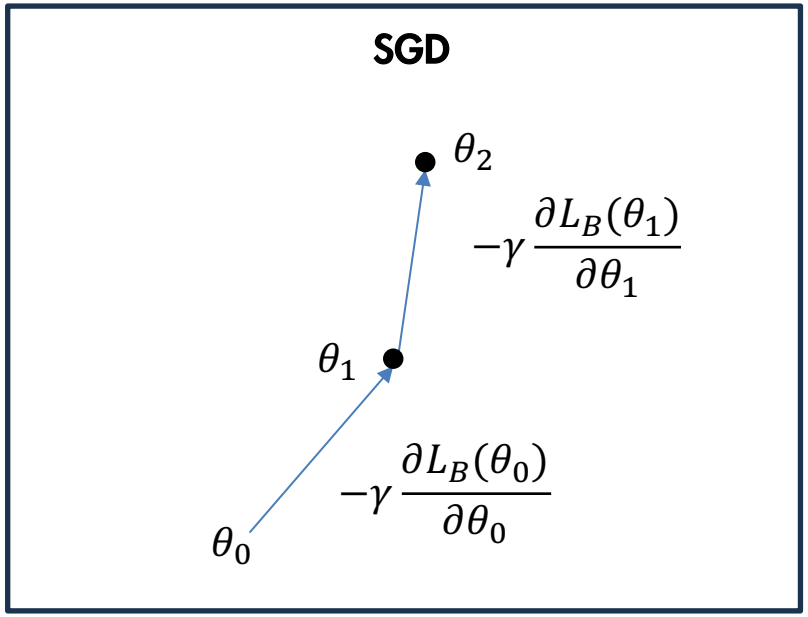


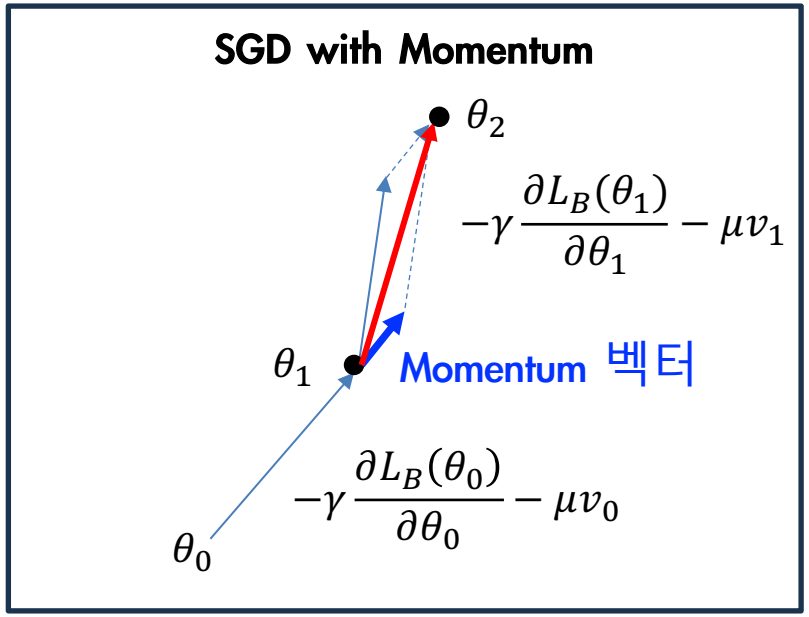
Momentum:
the force that keeps an object moving

Momentum

◈SGD with Momentum

– 기존 SGD에 Momentum(관성) 벡터 업데이트 추가





$$\theta_{t+1} = \theta_t - \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad t \geq 0$$

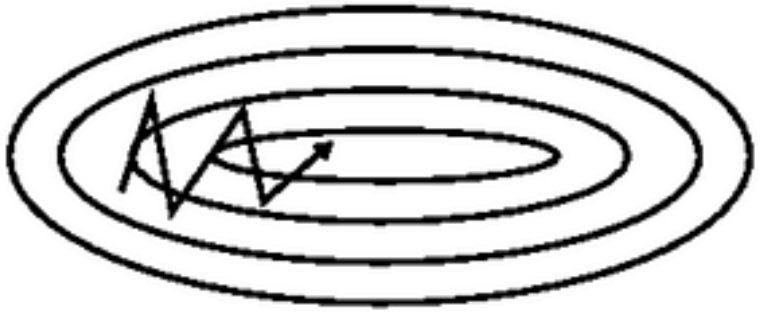
$$v_{t+1} = \mu v_t + \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad v_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - v_{t+1}$$

Momentum

◈SGD with Momentum

(a) SGD without momentum



(b) SGD with momentum

Gradient가 0이라 하더라도,
v값이 더해지면서 멈추지 않고 옆으로 이동할 수 있다!

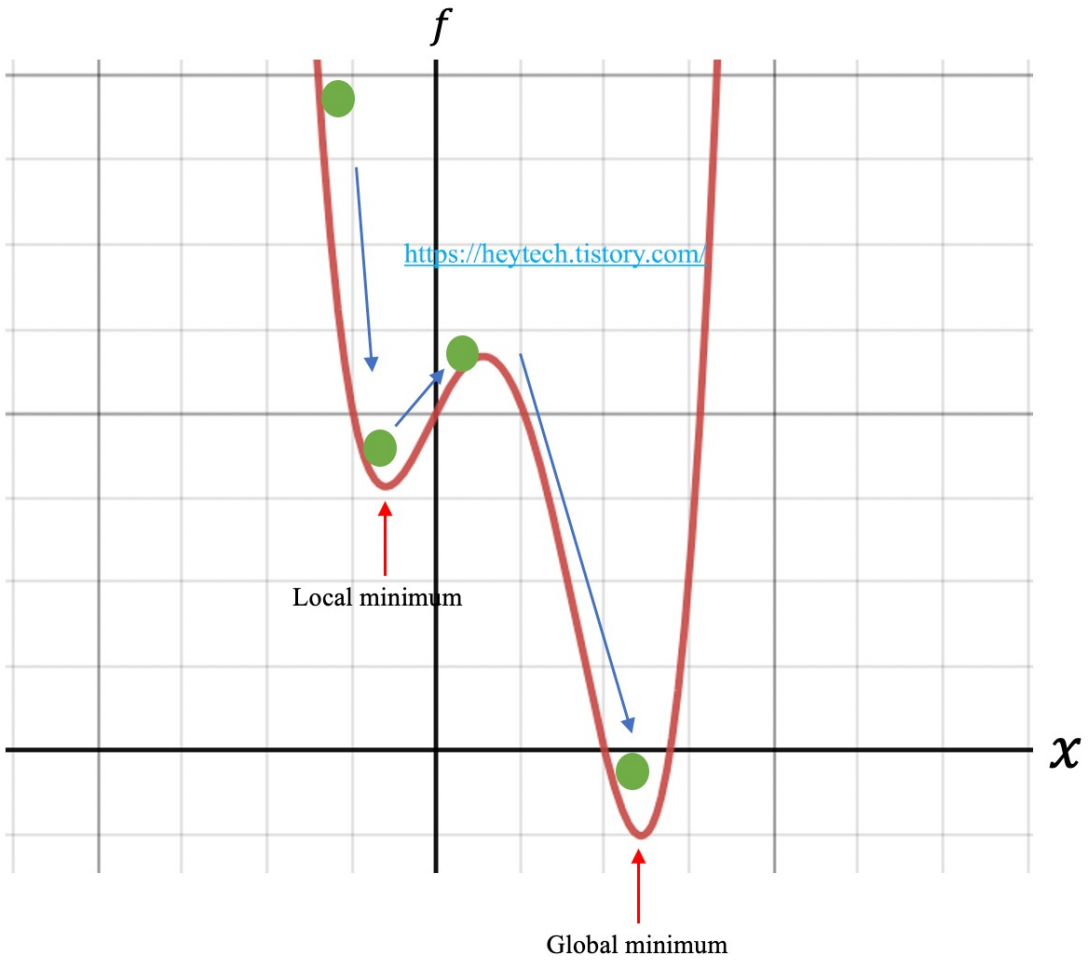
$$\theta_{t+1} = \theta_t - \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad t \geq 0$$

$$v_{t+1} = \mu v_t + \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad v_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - v_{t+1}$$

Momentum

◈SGD with Momentum



Momentum 최적화 기법은 이전에 이동했던 방향을 기억하면서 이전 기울기의 크기(즉, 관성)를 고려하여 약간의 추가 이동을 야기한다.

옆의 그림에서, 좌측에서부터 SGD로 손실 함수를 낮추다가 Local Minimum에 빠질 수 있는 상황에서 Momentum 기법을 사용하면 관성을 고려해 추가로 이동하기 때문에 이를 빠져나갈 수 있는 가능성이 더 커진다.

이후 Global Minimum 지점에 도달했을 때는 추가적인 관성을 받아도 더 올라가기 힘들기 때문에 이 지점이 Global Minimum이 될 수 있다.

이처럼 Momentum 최적화 기법은 Local Minimum에 빠지는 경우를 대처할 수 있다는 특징이 있다.

SGD & Momentum with PyTorch

◈SGD with Momentum

SGD

$$\theta_{t+1} = \theta_t - \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad t \geq 0$$

```
import torch.optim as optim

optimizer = optim.SGD(
  model.parameters(), lr=0.001
)
loss = loss_fn(model(input), target)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

SGD with Momentum

$$v_{t+1} = \mu v_t + \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad v_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - v_{t+1}$$

```
import torch.optim as optim

optimizer = optim.SGD(
  model.parameters(), lr=0.001, momentum=0.9
)
loss = loss_fn(model(input), target)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

Adagrad

◈ Adagrad: Adaptive Gradient Descent Method

SGD

$$\theta_{t+1} = \theta_t - \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad t \geq 0$$

Adagrad

Running sum of the squared gradient

Element-wise Multiplication

$$g_{t+1} = g_t + \frac{\partial L_B(\theta_t)}{\partial \theta_t} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - \gamma \frac{1}{\sqrt{g_{t+1}}} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}$$

Element-wise Square Root

– Motivation

- Should all parameters share the same learning rate?

– Adagrad provides a "parameter-specific adaptive learning rate"

- It adapts the learning rate for each parameter individually

Adagrad

◈Features of Adagrad

SGD

$$\theta_{t+1} = \theta_t - \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad t \geq 0$$

Adagrad

Running sum of the squared gradient

$$g_{t+1} = g_t + \frac{\partial L_B(\theta_t)}{\partial \theta_t} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - \gamma \frac{1}{\sqrt{g_{t+1}}} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}$$

ADAPTIVE learning rate
for
each parameter

- 1) Individual Learning Rates
  - The learning rate will be lower for parameters with a high gradient
  - The learning rate will be higher for parameters with a low gradient
- 2) Learning Rate Scaling
  - It scales down the learning rate based on the historical behavior of the gradients
  - Parameters that have large gradients will have smaller effective learning rates, while those with small gradients will have larger effective learning rates
- This allows the algorithm to converge more quickly for sparse data

Adagrad

◈Features of Adagrad

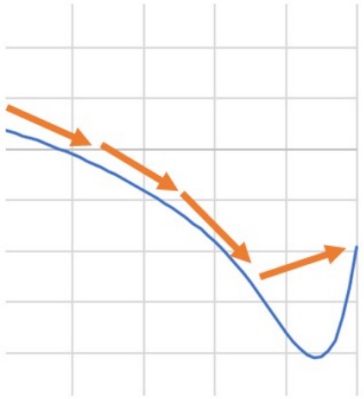
Running sum of the squared gradient

Adagrad

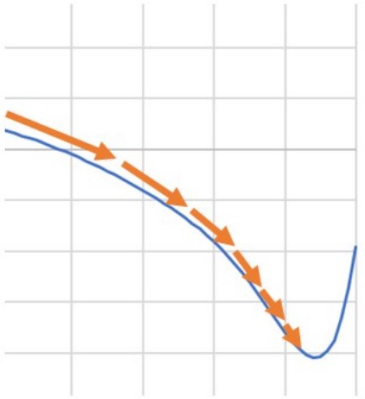
$$g_{t+1} = g_t + \frac{\partial L_B(\theta_t)}{\partial \theta_t} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - \gamma \frac{1}{\sqrt{g_{t+1}}} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}$$

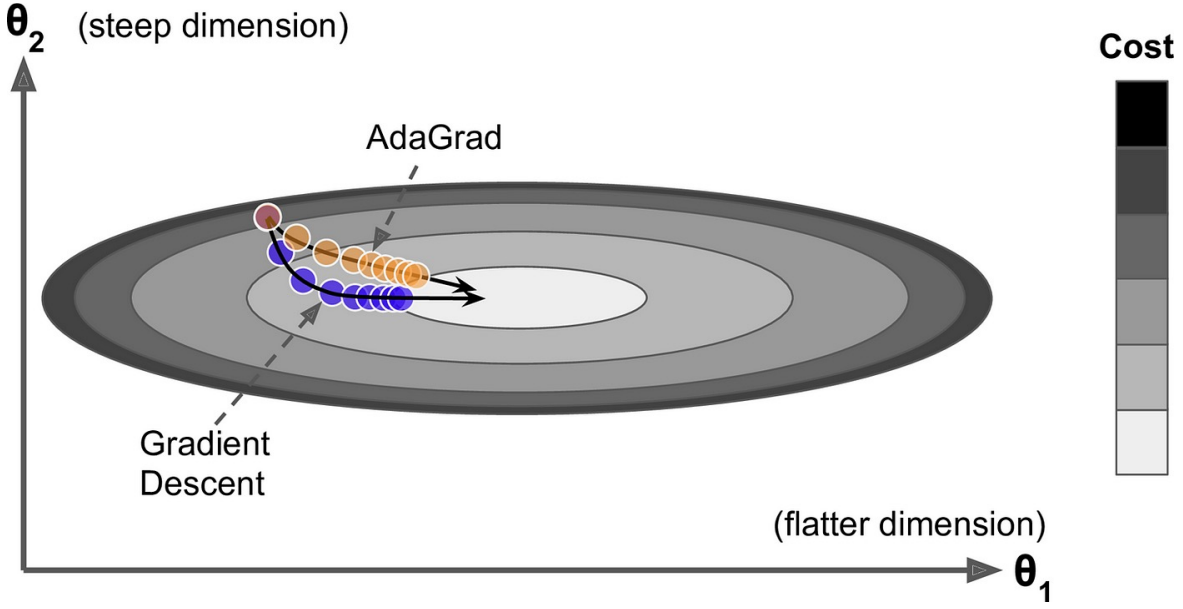
ADAPTIVE learning rate
for
each parameter



SGD



AdaGrad



RMSProp

◈RMSProp: Root Mean Square Propagation

Adagrad

$$g_{t+1} = g_t + \frac{\partial L_B(\theta_t)}{\partial \theta_t} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - \gamma \frac{1}{\sqrt{g_{t+1}}} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}$$

RMSProp

Decay Rate

$$g_{t+1} = \alpha g_t + (1-\alpha) \frac{\partial L_B(\theta_t)}{\partial \theta_t} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - \gamma \frac{1}{\sqrt{g_{t+1}}} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}$$

- Adagrad uses an "**ever-increasing** running sum of the squared gradient"
- Disadvantages of Adagrad
  - the running sum can become quite large, leading to extremely small learning rates
    - ➢ Rapid and abrupt learning rate decaying
  - Learning slows down rapidly near the slope of 0, increasing the risk of falling into local minima or saddle points

RMSProp

◈RMSProp: Root Mean Square Propagation

Adagrad

$$g_{t+1} = g_t + \frac{\partial L_B(\theta_t)}{\partial \theta_t} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - \gamma \frac{1}{\sqrt{g_{t+1}}} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}$$

RMSProp

Decay Rate

$$g_{t+1} = \alpha g_t + (1-\alpha) \frac{\partial L_B(\theta_t)}{\partial \theta_t} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - \gamma \frac{1}{\sqrt{g_{t+1}}} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}$$

– RMSProp's improvement over Adagrad

- Exponential Moving Average (EMA) of the squared gradient is used
  - ➢ a.k.a. exponentially decaying average
- Decay rate (default: 0.99) is used
  - ➢ a.k.a Averaging rate
- RMSProp overcomes the Adagrad's problem by being less aggressive on the learning rate decay

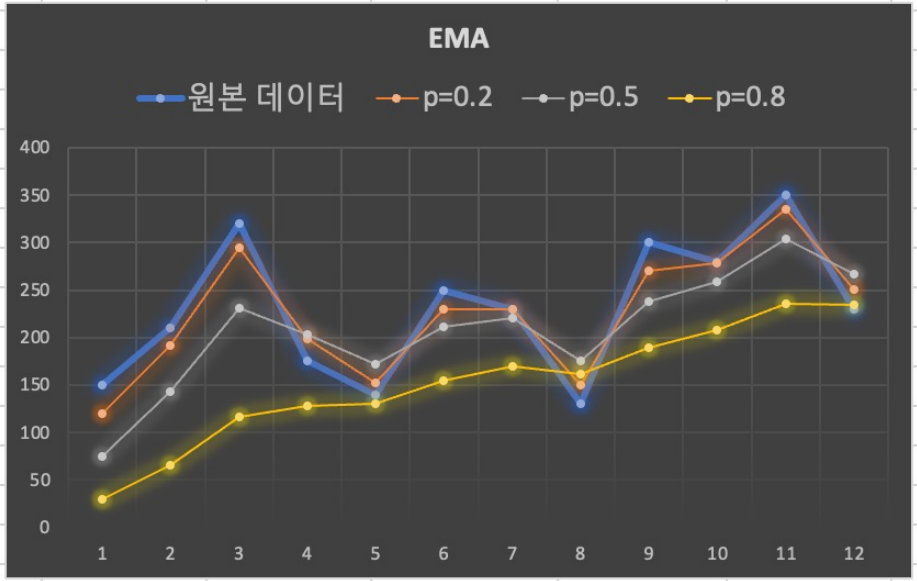
RMSProp

◈RMSProp: Root Mean Square Propagation

– Exponential Moving Average (EMA) (= Exponential Smoothing)

$$g_{t+1} = \alpha g_t + (1-\alpha) l_t, \qquad g_0 = 0, t \geq 0$$

	p	0.2		0.5		0.8	
t	l_t	g_t	g_t+1	g_t	g_t+1	g_t	g_t+1
0	150	0	120.0	0.0	75.0	0.0	30.0
1	210	120.0	192.0	75.0	142.5	30.0	66.0
2	320	192.0	294.4	142.5	231.3	66.0	116.8
3	175	294.4	198.9	231.3	203.1	116.8	128.4
4	140	198.9	151.8	203.1	171.6	128.4	130.8
5	250	151.8	230.4	171.6	210.8	130.8	154.6
6	230	230.4	230.1	210.8	220.4	154.6	169.7
7	130	230.1	150.0	220.4	175.2	169.7	161.7
8	300	150.0	270.0	175.2	237.6	161.7	189.4
9	280	270.0	278.0	237.6	258.8	189.4	207.5
10	350	278.0	335.6	258.8	304.4	207.5	236.0
11	230	335.6	251.1	304.4	267.2	236.0	234.8



Adagrad & RMSProp with PyTorch

◈ Adagrad & RMSProp

Adagrad

$$g_{t+1} = g_t + \frac{\partial L_B(\theta_t)}{\partial \theta_t} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - \gamma \frac{1}{\sqrt{g_{t+1}}} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}$$

```
import torch.optim as optim

optimizer = optim.Adagrad(
  model.parameters(), lr=0.001
)
loss = loss_fn(model(input), target)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

RMSProp

$$g_{t+1} = \alpha g_t + (1 - \alpha) \frac{\partial L_B(\theta_t)}{\partial \theta_t} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - \gamma \frac{1}{\sqrt{g_{t+1}}} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}$$

```
import torch.optim as optim

optimizer = optim.RMSProp(
  model.parameters(), lr=0.001, alpha=0.99
)
loss = loss_fn(model(input), target)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

Adam

◈ Adam (=SGD with Momentum + RMSProp)

SGD with Momentum

$$v_{t+1} = \mu v_t + \gamma \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad v_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - v_{t+1}$$

RMSProp

$$g_{t+1} = \alpha g_t + (1-\alpha) \frac{\partial L_B(\theta_t)}{\partial \theta_t} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0, t \geq 0$$

$$\theta_{t+1} = \theta_t - \gamma \frac{1}{\sqrt{g_{t+1}}} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}$$

Adam

$$v_{t+1} = \beta_1 v_t + (1-\beta_1) \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad v_0 = 0, t \geq 0$$

$$g_{t+1} = \beta_2 g_t + (1-\beta_2) \frac{\partial L_B(\theta_t)}{\partial \theta_t} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_1^{t+1}}$$

$$\hat{g}_{t+1} = \frac{g_{t+1}}{1-\beta_2^{t+1}}$$

$$\theta_{t+1} = \theta_t - \gamma \frac{1}{\sqrt{\hat{g}_{t+1}}} \cdot \hat{v}_{t+1}$$

Adam

◈ Adam (=SGD with Momentum + RMSProp)

Adam

$$v_{t+1} = \beta_1 v_t + (1-\beta_1)\frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad v_0 = 0, t \geq 0$$

$$g_{t+1} = \beta_2 g_t + (1-\beta_2)\frac{\partial L_B(\theta_t)}{\partial \theta_t} \cdot \frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0$$

v_{t+1} and g_{t+1} tend to be more biased towards 0, since β_1 and β_2 are usually close to 1

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_1^{t+1}}$$

$$\hat{g}_{t+1} = \frac{g_{t+1}}{1-\beta_2^{t+1}}$$

Bias-correction (편향수정)

$$\theta_{t+1} = \theta_t - \gamma \frac{1}{\sqrt{\hat{g}_{t+1}}} \cdot \hat{v}_{t+1}$$

Adam with PyTorch

◈ Adam (=SGD with Momentum + RMSProp)

Adam

$$v_{t+1} = \beta_1 v_t + (1-\beta_1)\frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad v_0 = 0, t \geq 0$$

$$g_{t+1} = \beta_2 g_t + (1-\beta_2)\frac{\partial L_B(\theta_t)}{\partial \theta_t}\cdot\frac{\partial L_B(\theta_t)}{\partial \theta_t}, \qquad g_0 = 0$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_1^{t+1}}$$

$$\hat{g}_{t+1} = \frac{g_{t+1}}{1-\beta_2^{t+1}}$$

$$\theta_{t+1} = \theta_t - \gamma\frac{1}{\sqrt{\hat{g}_{t+1}}}\cdot\hat{v}_{t+1}$$

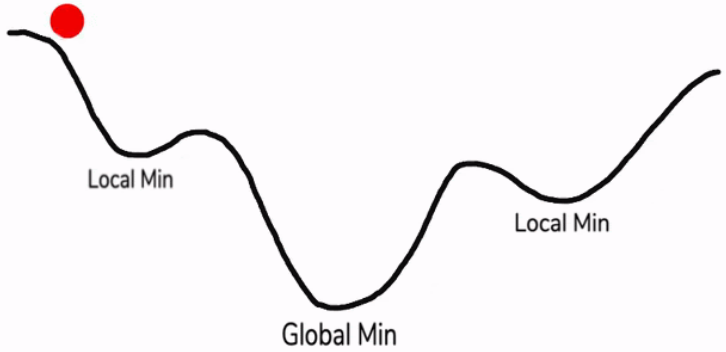
```python
import torch.optim as optim

optimizer = optim.Adam(
  model.parameters(),
  lr=0.001,
  betas=(0.9, 0.999)
)
loss = loss_fn(model(input), target)
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

Optimizer Evolution

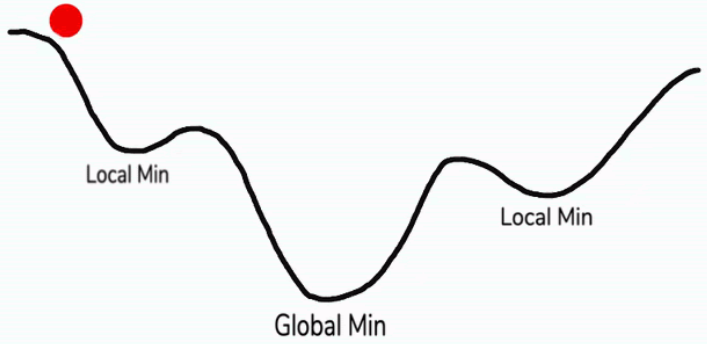
[Optimizer Animation]

https://medium.com/@kaitotally/adam-the-birthchild-of-adagrad-and-rmsprop-b5308b24b9cd

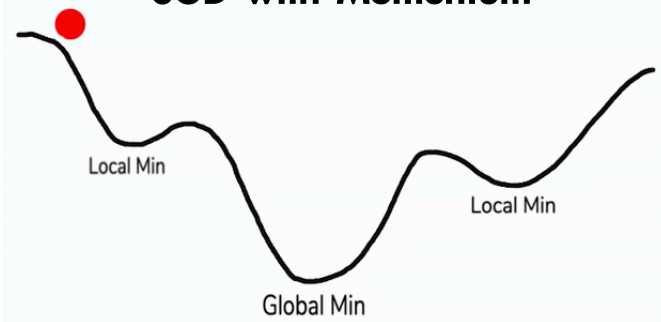
SGD - Stuck in local minima



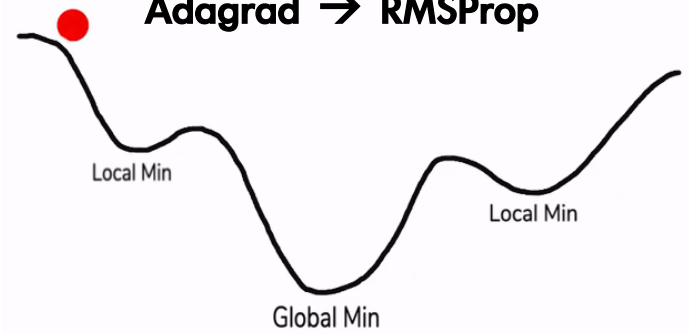
SGD - Oscillation



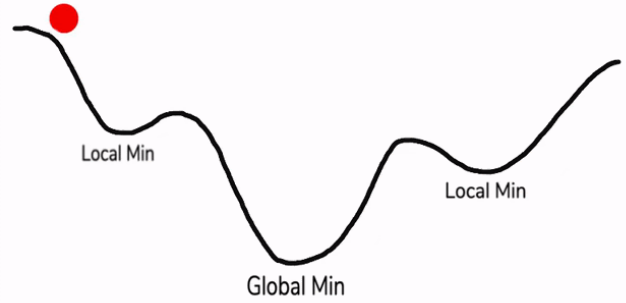
SGD with Momentum



Adagrad → RMSProp



Adam



link_dl/_01_code/_08_diverse_techniques/a_arg_parser.py

Diverse Optimizers with PyTorch

◈Argparser Extension with "optimizer" argument

```
import argparse
def get_parser():
  ...
  parser.add_argument(
    "-o", "--optimizer", type=int, default=0,
    help="Optimizers (0: SGD, 1: Momentum, 2: RMSProp, 4: Adam, default: 0)”
  )
  parser.add_argument(
    "-w", "--weight_decay", type=float, default=0.0, help="Weight decay (float, default: 0.0)”
  )
  parser.add_argument(
    "--dropout", action=argparse.BooleanOptionalAction, default=False, help="Dropout: True or False"
  )
  parser.add_argument(
    "-n", "--normalization", type=int, default=0,
    help="Normalization (0: No Normalization, 1: BatchNorm, 2: LayerNorm, default: 0)”
  )
  return parser
```

link_dl/_01_code/_08_diverse_techniques/b_cifar10_train_cnn_with_diverse_optimizers.py

Diverse Optimizers with PyTorch

◈Diverse Optimizers (1/4)

```
import torch
from torch import optim
from datetime import datetime
import os
import wandb
from pathlib import Path

BASE_PATH = str(Path(__file__).resolve().parent.parent.parent) # BASE_PATH: /Users/yhhan/git/link_dl
import sys
sys.path.append(BASE_PATH)

CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")
if not os.path.isdir(CHECKPOINT_FILE_PATH):
  os.makedirs(os.path.join(CURRENT_FILE_PATH, "checkpoints"))

import sys
sys.path.append(BASE_PATH)
```

link_dl/_01_code/_08_diverse_techniques/b_cifar10_train_cnn_with_diverse_optimizers.py

Diverse Optimizers with PyTorch

◈Diverse Optimizers (2/4)

```
from _01_code._06_fcn_best_practice.c_trainer import ClassificationTrainer
from _01_code._06_fcn_best_practice.h_cifar10_train_fcn import get_cifar10_data
from _01_code._07_cnn.c_cifar10_train_cnn import get_cnn_model
from _01_code._08_diverse_techniques.a_arg_parser import get_parser

def main(args):
  config = {
    'epochs': args.epochs,
    'batch_size': args.batch_size,
    'validation_intervals': args.validation_intervals,
    'print_epochs': args.print_epochs,
    'learning_rate': args.learning_rate,
    'early_stop_patience': args.early_stop_patience
  }

  optimizer_names = ["SGD", "Momentum", "RMSProp", "Adam"]
  run_time_str = datetime.now().astimezone().strftime('%Y-%m-%d_%H-%M-%S')
  name = "{0}_{1}".format(optimizer_names[args.optimizer], run_time_str)
```

link_dl/_01_code/_08_diverse_techniques/b_cifar10_train_cnn_with_diverse_optimizers.py

Diverse Optimizers with PyTorch

◈Diverse Optimizers (3/4)

```
def main(args):
  ...

  project_name = "cnn_cifar10_with_diverse_optimizers”
  wandb.init(
    mode="online" if args.wandb else "disabled",
    project=project_name,
    notes="cifar10 experiment with cnn and diverse optimizers",
    tags=["cnn", "cifar10", "diverse_optimizers"],
    name=name,
    config=config
  )
  device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

  train_data_loader, validation_data_loader, cifar10_transforms = get_cifar10_data(flatten=False)
  model = get_cnn_model()
  model.to(device)
  wandb.watch(model)
```

link_dl/_01_code/_08_diverse_techniques/b_cifar10_train_cnn_with_diverse_optimizers.py

Diverse Optimizers with PyTorch

◈Diverse Optimizers (4/4)

```
def main(args):
  ...
  optimizers = [
    optim.SGD(model.parameters(), lr=wandb.config.learning_rate),
    optim.SGD(model.parameters(), lr=wandb.config.learning_rate, momentum=0.9),
    optim.RMSprop(model.parameters(), lr=wandb.config.learning_rate),
    optim.Adam(model.parameters(), lr=wandb.config.learning_rate)
  ]
  print("Optimizer:", optimizers[args.optimizer])

  classification_trainer = ClassificationTrainerNoEarlyStopping(
    project_name, model, optimizers[args.optimizer], train_data_loader, validation_data_loader,
    mnist_transforms, run_time_str, wandb, device, CHECKPOINT_FILE_PATH
  )
  classification_trainer.train_loop()
  wandb.finish()

if __name__ == "__main__":
  parser = get_parser()
  args = parser.parse_args()
  main(args)
```

link_dl/_01_code/_08_diverse_techniques/b_cifar10_train_cnn_with_diverse_optimizers.py

Diverse Optimizers with PyTorch

◈CIFAR10 + CNN + Diverse Optimizers [SGD, Momentum, RMSProp, Adam]

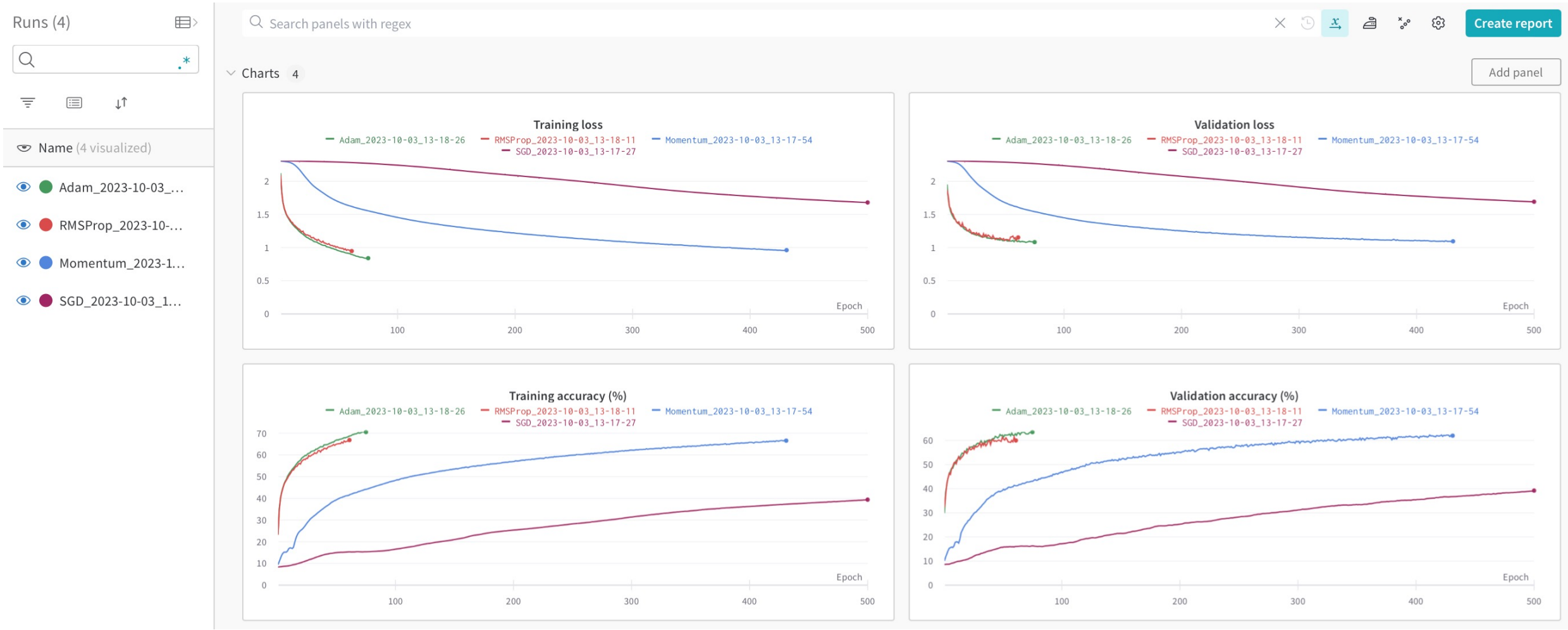
- SGD
  - python _01_code/_08_diverse_techniques/b_cifar10_train_cnn_with_diverse_optimizers.py --wandb -o 0 -v 1
- SGD with Momentum
  - python _01_code/_08_diverse_techniques/b_cifar10_train_cnn_with_diverse_optimizers.py --wandb -o 1 -v 1
- RMSProp
  - python _01_code/_08_diverse_techniques/b_cifar10_train_cnn_with_diverse_optimizers.py --wandb -o 2 -v 1
- Adam
  - python _01_code/_08_diverse_techniques/b_cifar10_train_cnn_with_diverse_optimizers.py --wandb -o 3 -v 1

link_dl/_01_code/_08_diverse_techniques/b_cifar10_train_cnn_with_diverse_optimizers.py

Diverse Optimizers with PyTorch

◈CIFAR10 + CNN + Diverse Optimizers [SGD, Momentum, RMSProp, Adam]

- https://wandb.ai/link-koreatech/cnn_cifar10_with_diverse_optimizers/workspace?workspace=user-link-koreatech

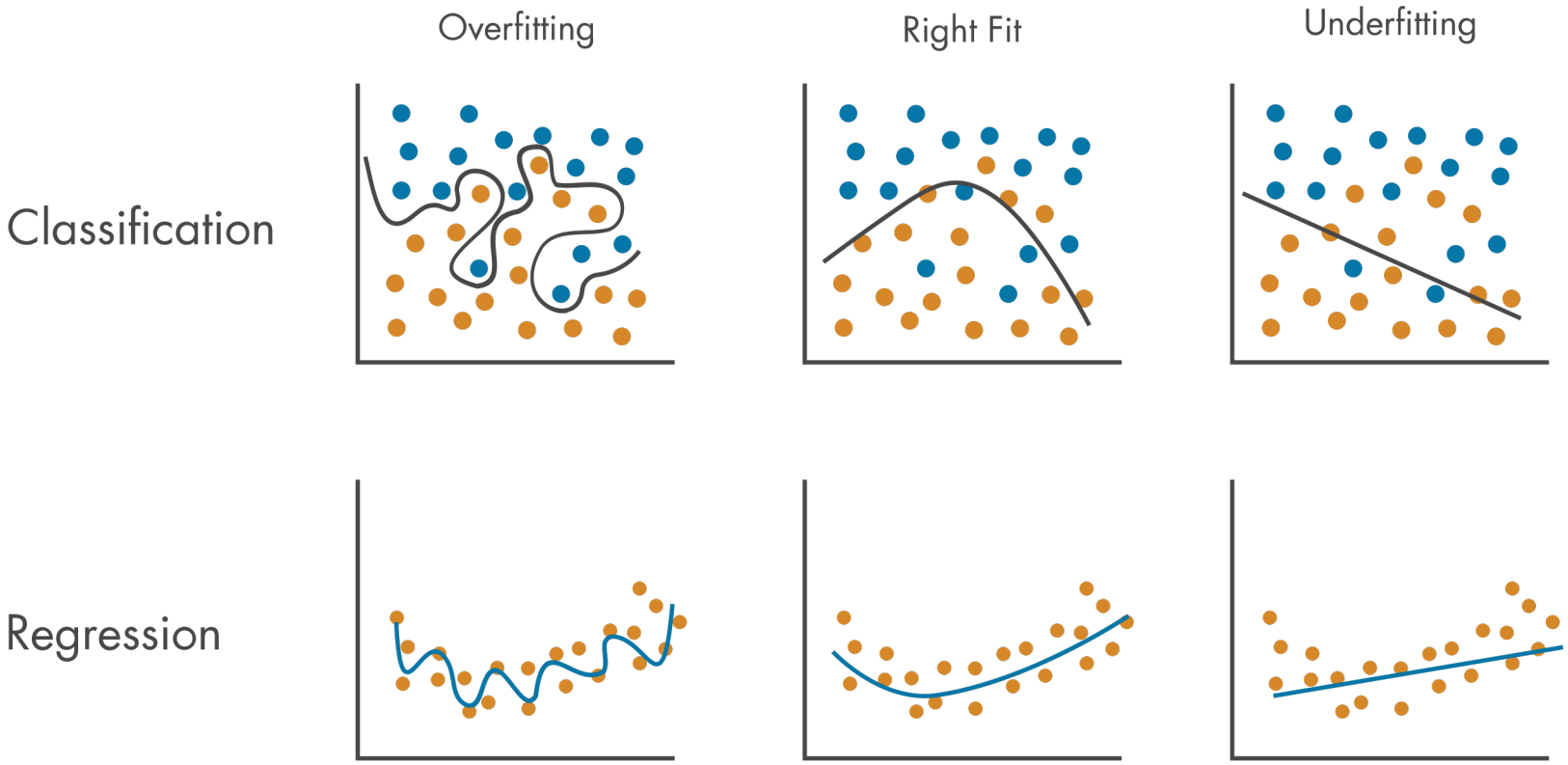


Regularization

Regularization

◈Overfitting

– It occurs when a model learns to fit the training data too closely, capturing noise and irrelevant patterns, which can lead to poor performance on unseen data



[Q] What is noise in the training data?

[A] Humans are prone to making mistakes when collecting data, and data collection instruments may be unreliable, resulting in dataset errors. The errors are referred to as noise. Data noise in machine learning can cause problems since the algorithm interprets the noise as a pattern and can start generalizing from it.

Regularization

◈Regularization (규제화)

- A set of techniques used to prevent overfitting and improve the generalization performance of a model
- Original Loss Functions
  - N: Number of samples
  - K: Number of classes

$$L(\theta) = L(W, b) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_n^k \cdot log\hat{y}_n^k \quad \text{for classification}$$

$$L(\theta) = L(W, b) = \frac{1}{N}\sum_{n=1}^{N} (\hat{y}_n(x_n) - y_n)^2 \quad \text{for regression}$$

- **L1 Regularized Loss functions (L1 Regularization)**

$$L_R(\theta) = L_R(W, b) = L(W, b) + \frac{\lambda}{2}|W|$$

- **L2 Regularized Loss functions (L2 Regularization)**

$$L_R(\theta) = L_R(W, b) = L(W, b) + \frac{\lambda}{2}|W|^2$$

Regularization

◈Regularization

– L1 or L2 Regularization

$$L_R(\theta) = L_R(W, b) = L(W, b) + \frac{\lambda}{2}|W|$$

$$L_R(\theta) = L_R(W, b) = L(W, b) + \frac{\lambda}{2}|W|^2$$

– Why L2 Regularization works?

- It makes the weights of the network smaller
  - ➢ The smallness of weights implies that the regularized network behavior will not change much if there are local noises in the data
  - ➢ This forces the network to learn only features (or patterns) which are seen often across the training set
- It can avoid exploding gradient
  - ➢ This will help keep the weights as small as possible, preventing the weights to grow out of control, and thus avoid exploding gradient.

Regularization

◈ Regularization

– Weight Decay in Gradient Descent Methods

- **Using L2 Regularization**

$$L_R(\theta) = L_R(W,b) = L(W,b) + \frac{\lambda}{2}|W|^2$$

Regularized Loss

$$W = W - \gamma \frac{\partial L_R(W,b)}{\partial W} = W - \gamma \frac{\partial}{\partial W}\left(L(W,b) + \frac{\lambda}{2}|W|^2\right) = W - \gamma \frac{\partial L(W,b)}{\partial W} - \gamma\lambda W$$

$$= \underbrace{(1-\gamma\lambda)W}_{\text{Weight Decay}} - \gamma \frac{\partial L(W,b)}{\partial W}$$

Original Loss

$$W = (1-\gamma\lambda)W - \gamma \frac{\partial L(W,b)}{\partial W}$$

$$b = b - \gamma \frac{\partial L(W,b)}{\partial W}$$

Bias is not regularized. Why?

1) Bias typically require less data as compared to weights to fit accurately
2) Regularizing a bias can introduce a significant amount of underfitting

Regularization with PyTorch

$$W = (1 - \gamma\lambda)W - \gamma \frac{\partial L(W, b)}{\partial W}$$

◈SGD/SGD with Momentum & Weight Decay

```
import torch.optim as optim
optimizer = optim.SGD(
  model.parameters(), lr=0.001,
 weight_decay=0.001
)
```

```
import torch.optim as optim
optimizer = optim.SGD(
  model.parameters(), lr=0.001, momentum=0.9,
 weight_decay=0.001
)
```

◈Adam/RMSProp & Weight Decay

```
import torch.optim as optim
optimizer = optim.Adagrad(
  model.parameters(), lr=0.001 ,
 weight_decay=0.001
)
```

```
import torch.optim as optim
optimizer = optim.RMSProp(
  model.parameters(), lr=0.001, alpha=0.99 ,
 weight_decay=0.001
)
```

◈Adam & Weight Decay

```
import torch.optim as optim
optimizer = optim.Adam(
  model.parameters(), lr=0.001,
  betas=(0.9, 0.999), weight_decay=0.001
)
```

link_dl/_01_code/_08_diverse_techniques/c_cifar10_train_cnn_with_weight_decay.py

Weight Decay with PyTorch

◈Weight Decay (1/4)

```
import torch
from torch import optim
from datetime import datetime
import os
import wandb
from pathlib import Path

BASE_PATH = str(Path(__file__).resolve().parent.parent.parent) # BASE_PATH: /Users/yhhan/git/link_dl
import sys
sys.path.append(BASE_PATH)

CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")
if not os.path.isdir(CHECKPOINT_FILE_PATH):
  os.makedirs(os.path.join(CURRENT_FILE_PATH, "checkpoints"))

import sys
sys.path.append(BASE_PATH)
```

link_dl/_01_code/_08_diverse_techniques/c_cifar10_train_cnn_with_weight_decay.py

Weight Decay with PyTorch

◈Weight Decay (2/4)

```
from _01_code._06_fcn_best_practice.c_trainer import ClassificationTrainer
from _01_code._06_fcn_best_practice.h_cifar10_train_fcn import get_cifar10_data
from _01_code._07_cnn.c_cifar10_train_cnn import get_cnn_model
from _01_code._08_diverse_techniques.a_arg_parser import get_parser

def main(args):
  config = {
    'epochs': args.epochs,
    'batch_size': args.batch_size,
    'validation_intervals': args.validation_intervals,
    'print_epochs': args.print_epochs,
    'learning_rate': args.learning_rate,
    'early_stop_patience': args.early_stop_patience
    'weight_decay': args.weight_decay
  }

  technique_name = "weight_decay_{0:.3f}_".format(args.weight_decay)
  run_time_str = datetime.now().astimezone().strftime('%Y-%m-%d_%H-%M-%S’)
  name = "{0}_{1}".format(technique_name, run_time_str)
```

link_dl/_01_code/_08_diverse_techniques/c_cifar10_train_cnn_with_weight_decay.py

Weight Decay with PyTorch

◈Weight Decay (3/4)

```
def main(args):
  ...

  project_name = "cnn_cifar10_with_weight_decay”
  wandb.init(
    mode="online" if args.wandb else "disabled",
    project=project_name,
    notes="cifar10 experiment with cnn and weight_decay",
    tags=["cnn", "cifar10", "weight_decay"],
    name=name,
    config=config
  )

  device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
  train_data_loader, validation_data_loader, cifar10_transforms = get_cifer10_data(flatten=False)
  model = get_cnn_model()
  model.to(device)
  wandb.watch(model)
```

link_dl/_01_code/_08_diverse_techniques/c_cifar10_train_cnn_with_weight_decay.py

Weight Decay with PyTorch

◈Weight Decay (4/4)

```
def main(args):
  ...
  optimizer = optim.Adam(
    model.parameters(), lr=wandb.config.learning_rate, weight_decay=args.weight_decay
  )
  classification_trainer = ClassificationTrainer(
    project_name, model, optimizer,
    train_data_loader, validation_data_loader, cifar10_transforms,
    run_time_str, wandb, device, CHECKPOINT_FILE_PATH
  )
  classification_trainer.train_loop()
  wandb.finish()


if __name__ == "__main__":
  parser = get_parser()
  args = parser.parse_args()
  main(args)
```

link_dl/_01_code/_08_diverse_techniques/c_cifar10_train_cnn_with_weight_decay.py

Weight Decay with PyTorch

◈ CIFAR10 + CNN + Adam Optimizer + Diverse Weight Decays [0.0, 0.001, 0.005, 0.01, 0.02]

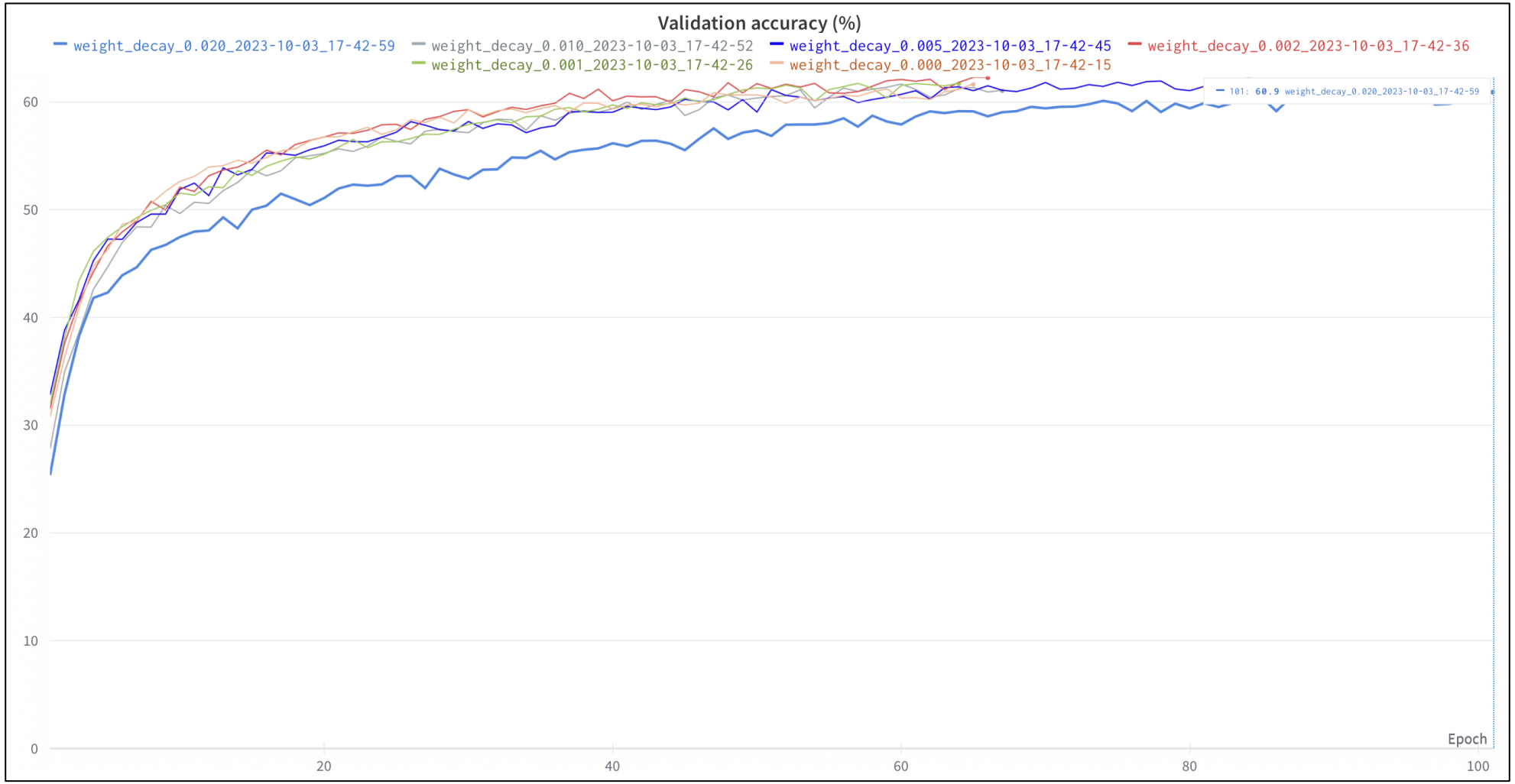
- Weight Decay: 0.0
  - python _01_code/_08_diverse_techniques/c_cifar10_train_cnn_with_weight_decay.py --wandb -v 1 -o 3 -w 0.0
- Weight Decay: 0.001
  - python _01_code/_08_diverse_techniques/c_cifar10_train_cnn_with_weight_decay.py --wandb -v 1 -o 3 -w 0.001
- Weight Decay: 0.002
  - python _01_code/_08_diverse_techniques/c_cifar10_train_cnn_with_weight_decay.py --wandb -v 1 -o 3 -w 0.002
- Weight Decay: 0.005
  - python _01_code/_08_diverse_techniques/c_cifar10_train_cnn_with_weight_decay.py --wandb -v 1 -o 3 -w 0.005
- Weight Decay: 0.01
  - python _01_code/_08_diverse_techniques/c_cifar10_train_cnn_with_weight_decay.py --wandb -v 1 -o 3 -w 0.01
- Weight Decay: 0.02
  - python _01_code/_08_diverse_techniques/c_cifar10_train_cnn_with_weight_decay.py --wandb -v 1 -o 3 -w 0.02

link_dl/_01_code/_08_diverse_techniques/c_cifar10_train_cnn_with_weight_decay.py

Weight Decay with PyTorch

◈ CIFAR10 + CNN + Adam Optimizer + Diverse Weight Decays [0.0, 0.001, 0.005, 0.01, 0.02]

- https://wandb.ai/link-koreatech/cnn_cifar10_with_weight_decay/workspace?workspace=user-link-koreatech



Dropout

Dropout

◈Regularization via Dropout

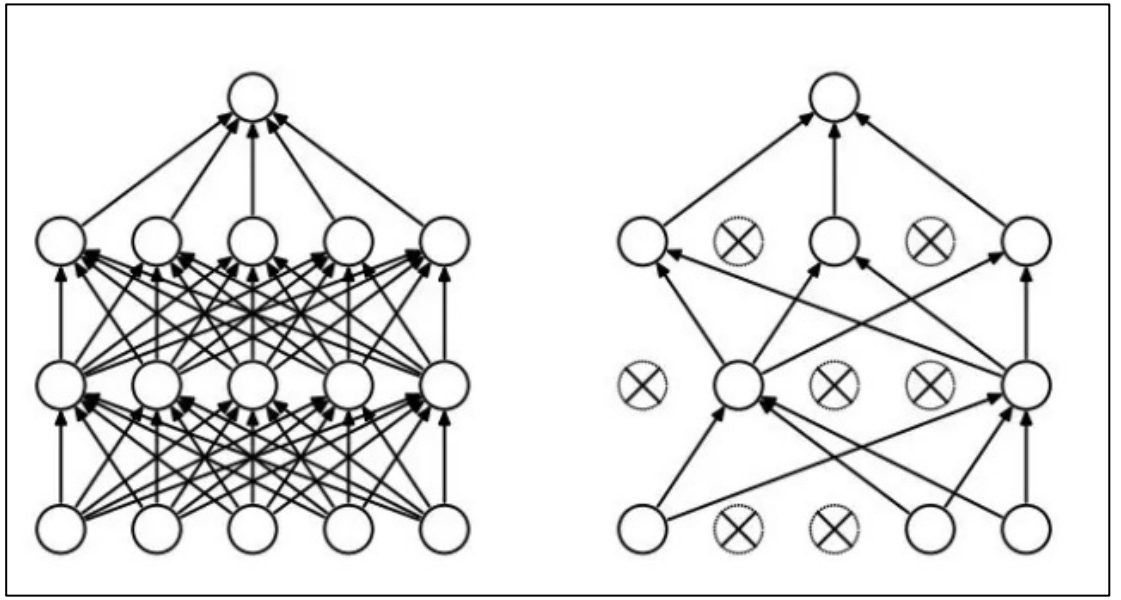
– Randomly drop "hidden and visible units (along with their connections)" during training

- Drop probability: p (independent of each unit, Hyper-parameter)

Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.



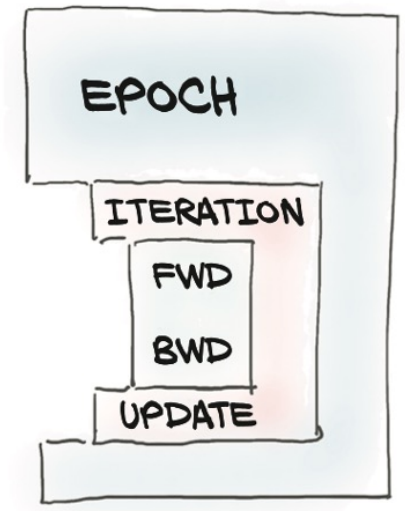
A standard neural net with two hidden layers

A thinned net produced by applying dropout, crossed units have been dropped

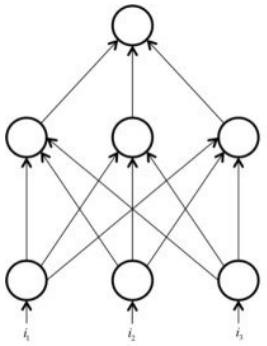
Dropout

Dropout

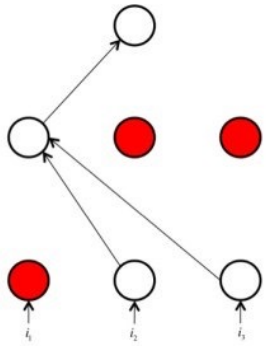
```
FOR N EPOCHS:
  SPLIT DATASET IN MINIBATCHES
  FOR EVERY MINIBATCH:
    WITH EVERY SAMPLE IN MINIBATCH:
      EVALUATE MODEL (FORWARD)
      COMPUTE LOSS
      ACCUMULATE GRADIENT OF LOSS (BACKWARD)
    UPDATE MODEL WITH ACCUMULATED GRADIENT
```



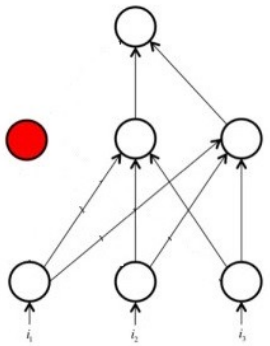
Generate a new network dropped randomly for every new mini-batch



The original network



(b) Random removal of neurons



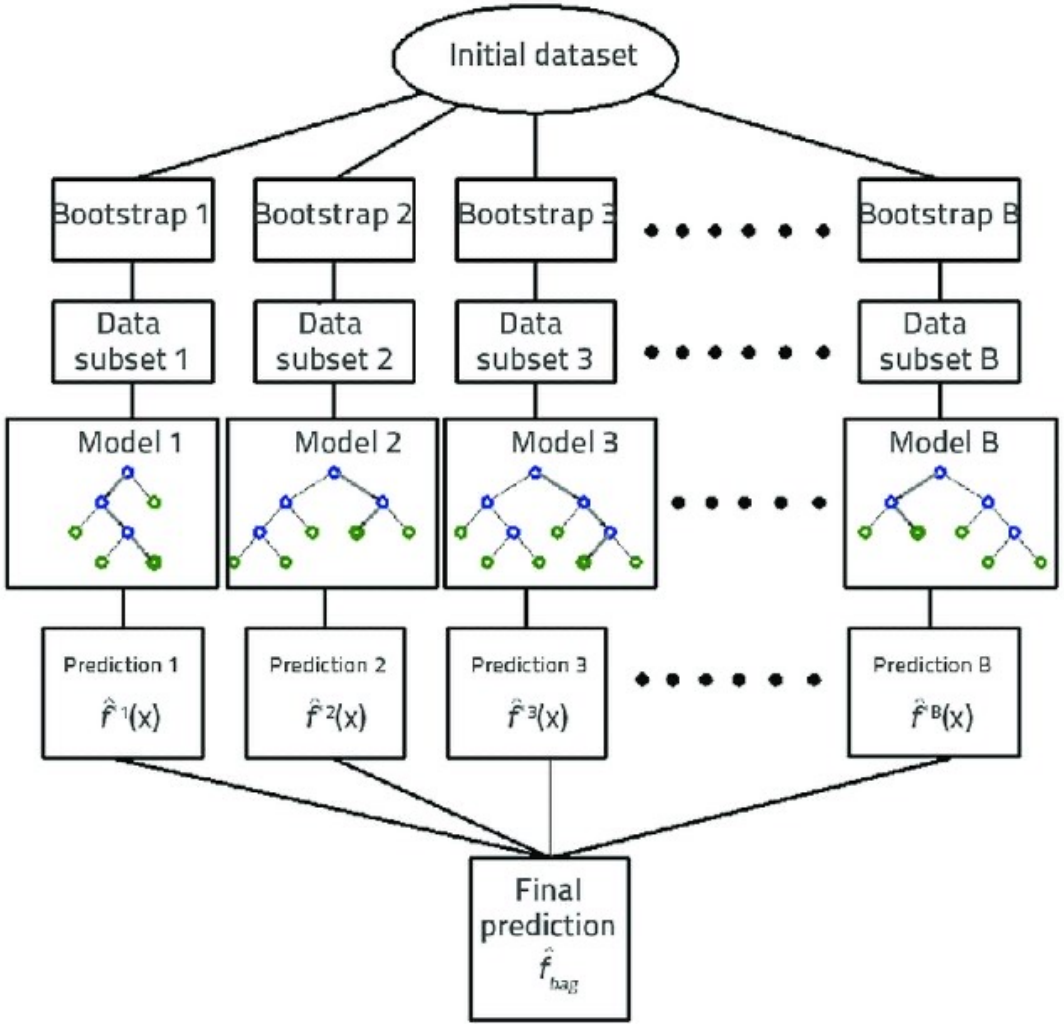
(c) Yet another random removal of neurons for a diffent epoch from that in (a)

Dropout

◈ Advantages of Dropout

– Network Averaging (similar to Bagging)

- Dropout can be thought of as an ensemble method.
- When dropout is applied, it's like training multiple neural networks with different architectures and different dataset, and then averaging the results.
- Each forward pass with dropout corresponds to a different "thinned" version of the original network.
- At test time, all the neurons are used, which is equivalent to taking an ensemble average.



Dropout

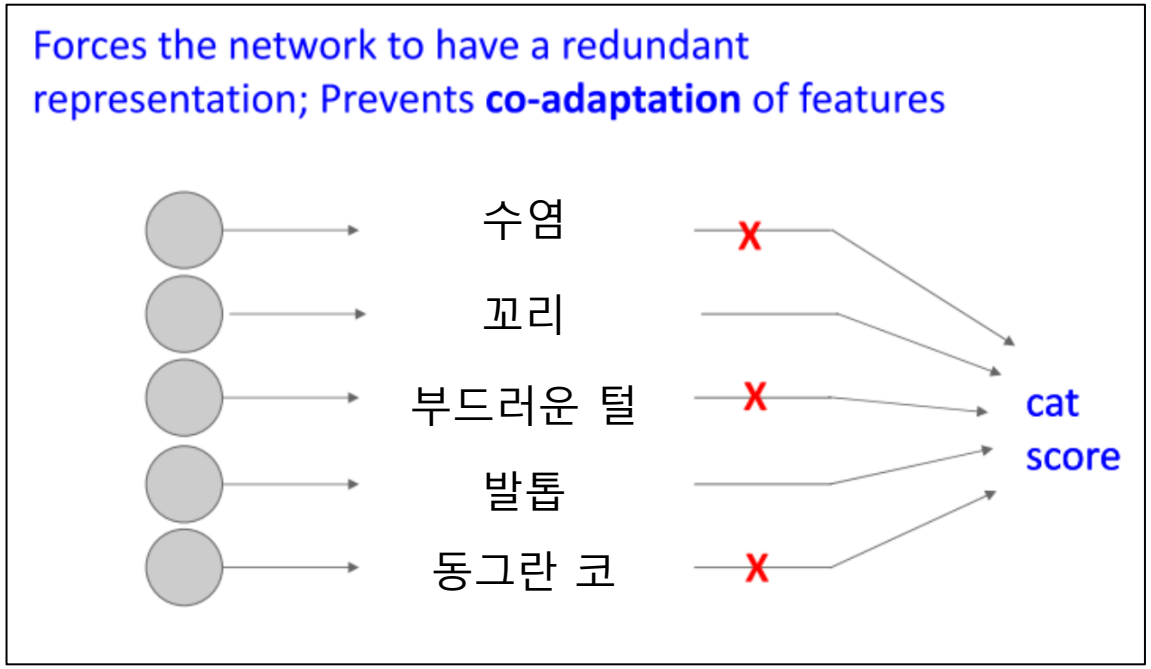
◈ Advantages of Dropout

– Reduced Co-adaptation → Improved Generalization

- Co-adaptation
  - ➢ Some neurons are highly dependent on others
  - ➢ If those independent neurons receive "bad" inputs, then the dependent neurons can be affected as well, and ultimately it can significantly alter the model performance, which is what might happen with overfitting

- By dropout, the model is forced to have neurons that can learn good features (or representations) while not relying on some specific neurons

- Therefore, the resulting model can be more robust to unseen data.



link_dl/_01_code/_08_diverse_techniques/d_my_linear_layer_with_dropout.py

Dropout

◈'dropout' implementation example

```
import torch
from torch import nn

class MyLinearWithActivationAndDropout(nn.Module):
  def __init__(self, in_features, out_features):
    super().__init__()
    self.weight = nn.Parameter(torch.randn(out_features, in_features))
    self.bias = nn.Parameter(torch.randn(out_features))
    self.activation = nn.Sigmoid()
    self.drop_prob = 0.3

  def forward(self, input, is_train):
    z = input @ self.weight.t() + self.bias
    a = self.activation(z)
    if is_train:   # dropout only if model is trained
      a_masked, a_masked_and_scaled = self.dropout(a)
      return a_masked, a_masked_and_scaled
    else:
      return a
```

link_dl/_01_code/_08_diverse_techniques/d_my_linear_layer_with_dropout.py

Dropout

◈'dropout' implementation example

```
class MyLinearWithActivationAndDropout(nn.Module):
  ...
  def dropout(self, a):
    ...
    # Step 1: initialize matrix r, where its shape is same as the above a tensor
    r = torch.rand_like(a)

    # Step 2: convert entries of D1 to 0 or 1 (using keep_prob as the threshold)
    mask = (r < self.drop_prob).to(torch.int)
    print(mask, "!!!")
    # >>> tensor([[[1, 0, 0, 0, 1, 0, 0],
    # >>>           [0, 0, 0, 0, 0, 0, 0],
    # >>>           [0, 1, 1, 1, 1, 0, 0]], dtype=torch.int32) !!!

    # Step 3: shut down some neurons of A1
    masked_a = mask * a

    # Step 4: scale the value of neurons that haven't been shut down
    a_masked_and_scaled = a_masked / (1 - self.drop_prob)  # It is called 'Inverted Dropout'
    return a_masked, a_masked_and_scaled
```

link_dl/_01_code/_08_diverse_techniques/d_my_linear_layer_with_dropout.py

Dropout

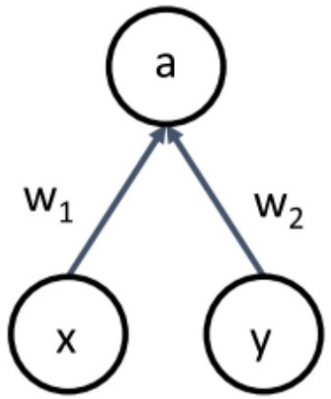
◈'dropout' implementation example

```
if __name__ == "__main__":
  my_linear = MyLinearWithActivationAndDropout(in_features=4, out_features=10)

  batch_input = torch.randn(3, 4)
  batch_output, batch_input_scaled = my_linear(batch_input)

  print("input.shape:", batch_input.shape)                    # >>> torch.Size([3, 4])
  print("output.shape:", batch_output.shape)                  # >>> torch.Size([3, 7])
  print(batch_output, torch.sum(batch_output, dim=0))
  # >>> tensor([[0.0072, 0.0000, 0.0000, 0.0000, 0.1074, 0.0000, 0.0000],
  # >>>         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
  # >>>         [0.0000, 0.9088, 0.7807, 0.5589, 0.5824, 0.0000, 0.0000]], grad_fn=<MulBackward0>)
  # >>> tensor([0.0072, 0.9088, 0.7807, 0.5589, 0.6898, 0.0000, 0.0000], grad_fn=<SumBackward1>)
  print(batch_input_scaled, torch.sum(batch_input_scaled, dim=0))
  # >>> tensor([[0.0103, 0.0000, 0.0000, 0.0000, 0.1534, 0.0000, 0.0000],
  # >>>         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
  # >>>         [0.0000, 1.2983, 1.1152, 0.7984, 0.8320, 0.0000, 0.0000]], grad_fn=<DivBackward0>)
  # >>> tensor([0.0103, 1.2983, 1.1152, 0.7984, 0.9854, 0.0000, 0.0000], grad_fn=<SumBackward1>)
```

Dropout

◈Why Inverted Dropout?

– Consider a single output neuron and the <u>drop probability (p) is 0.4</u>



x	y
x	0
0	y
0	0

At the validation or test time $E[a] = w_1x + w_2y$

At the training time

$$E[a] = \frac{3}{5}\cdot\frac{3}{5}(w_1x + w_2y) + \frac{3}{5}\cdot\frac{2}{5}(w_1x + 0y)$$
$$+\frac{2}{5}\cdot\frac{3}{5}(0x + w_2y) + \frac{2}{5}\cdot\frac{2}{5}(0x + 0y)$$
$$= \frac{9+6}{25}w_1x + \frac{9+6}{25}w_2y = \frac{3}{5}w_1x + \frac{3}{5}w_2y$$

At the training time, with scaling

$$E[a] = \left(\frac{3}{5}w_1x + \frac{3}{5}w_2y\right)/(1-0.4)$$
$$= \left(\frac{3}{5}w_1x + \frac{3}{5}w_2y\right)/\frac{3}{5} = w_1x + w_2y$$

link_dl/_01_code/_08_diverse_techniques/e_cifar10_train_cnn_with_dropout.py

Dropout with PyTorch

◈ Dropout (1/6)

```
import torch
from torch import optim, nn
from datetime import datetime
import os
import wandb
from pathlib import Path

BASE_PATH = str(Path(__file__).resolve().parent.parent.parent) # BASE_PATH: /Users/yhhan/git/link_dl
import sys
sys.path.append(BASE_PATH)

CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")
if not os.path.isdir(CHECKPOINT_FILE_PATH):
  os.makedirs(os.path.join(CURRENT_FILE_PATH, "checkpoints"))

import sys
sys.path.append(BASE_PATH)
```

link_dl/_01_code/_08_diverse_techniques/e_cifar10_train_cnn_with_dropout.py

Dropout with PyTorch

◈ Dropout (2/6)

```
from _01_code._06_fcn_best_practice.c_trainer import ClassificationTrainer
from _01_code._06_fcn_best_practice.h_cifar10_train_fcn import get_cifar10_data
from _01_code._07_cnn.c_cifar10_train_cnn import get_cnn_model
from _01_code._08_diverse_techniques.a_arg_parser import get_parser

def get_cnn_model_with_dropout():
  class MyModel(nn.Module):

    def __init__(self, in_channels, n_output):
      super().__init__()

      self.model = nn.Sequential(
        # 3 x 32 x 32 --> 6 x (32 - 5 + 1) x (32 - 5 + 1) = 6 x 28 x 28
        nn.Conv2d(in_channels=in_channels, out_channels=6, kernel_size=(5, 5), stride=(1, 1)),
        # 6 x 28 x 28 --> 6 x 14 x 14
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.ReLU(),
```

link_dl/_01_code/_08_diverse_techniques/e_cifar10_train_cnn_with_dropout.py

Dropout with PyTorch

◈ Dropout (3/6)

```
def get_cnn_model_with_dropout():
  class MyModel(nn.Module):
    def __init__(self, in_channels, n_output):
      ...
      self.model = nn.Sequential(
        ...
        # 6 x 14 x 14 --> 16 x (14 - 5 + 1) x (14 - 5 + 1) = 16 x 10 x 10
        nn.Conv2d(in_channels=6, out_channels=16, kernel_size=(5, 5), stride=(1, 1)),
        # 16 x 10 x 10 --> 16 x 5 x 5
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.ReLU(),
        nn.Flatten(),
        nn.Dropout(p=0.5),        # p: dropout probability
        nn.Linear(400, 128),
        nn.ReLU(),
        nn.Dropout(p=0.5),        # p: dropout probability
        nn.Linear(84, n_output),
      )
```

link_dl/_01_code/_08_diverse_techniques/e_cifar10_train_cnn_with_dropout.py

Dropout with PyTorch

◈ Dropout (4/6)

```
def get_cnn_model_with_dropout():
  class MyModel(nn.Module):
    ...
    ...

    def forward(self, x):
      x = self.model(x)
      return x

  # 3 * 32 * 32
  my_model = MyModel(in_channels=3, n_output=10)

  return my_model
```

link_dl/_01_code/_08_diverse_techniques/e_cifar10_train_cnn_with_dropout.py

Dropout with PyTorch

◈ Dropout (5/6)

```
class ClassificationTrainer:
  def __init__(
    self, project_name, model, optimizer, train_data_loader, validation_data_loader, transforms,
    run_time_str, wandb, device, checkpoint_file_path
  ):
    ...
    self.model = model
    ...

  def do_train(self):
    self.model.train()
    ...
    return train_loss, train_accuracy

  def do_validation(self):
    self.model.eval()
    ...
    return validation_loss, validation_accuracy
```

link_dl/_01_code/_08_diverse_techniques/e_cifar10_train_cnn_with_dropout.py

Dropout with PyTorch

◈Dropout (6/6)

```
def main(args):
  ...
  if args.dropout:
    model = get_cnn_model_with_dropout()
  else:
    model = get_cnn_model()

  model.to(device)
  wandb.watch(model)
  optimizer = optim.Adam(
    model.parameters(), lr=wandb.config.learning_rate, weight_decay=args.weight_decay
  )
  ...

if __name__ == "__main__":
  parser = get_parser()
  args = parser.parse_args()
  main(args)
```

link_dl/_01_code/_08_diverse_techniques/e_cifar10_train_cnn_with_dropout.py

Dropout with PyTorch

◈CIFAR10 + CNN + Adam Optimizer + Weight Decay 0.002 + [No-Dropout, Dropout]

- Dropout
  - python _01_code/_08_diverse_techniques/e_cifar10_train_cnn_with_dropout.py --wandb -v 1 -o 3 -w 0.002 --dropout

- No Dropout
  - python _01_code/_08_diverse_techniques/e_cifar10_train_cnn_with_dropout.py --wandb -v 1 -o 3 -w 0.002 --no-dropout

link_dl/_01_code/_08_diverse_techniques/e_cifar10_train_cnn_with_dropout.py

Dropout with PyTorch

◈CIFAR10 + CNN + Adam Optimizer + Weight Decay 0.002 + [No-Dropout, Dropout]

- https://wandb.ai/link-koreatech/cnn_cifar10_with_dropout/workspace?workspace=user-link-koreatech