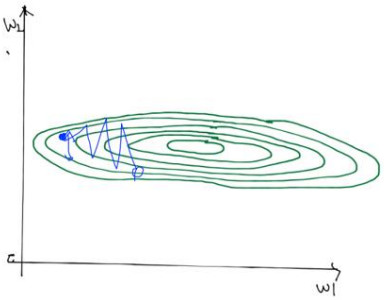
뉴럴네트워크

무인이동체공학 17011882 김 우 혁

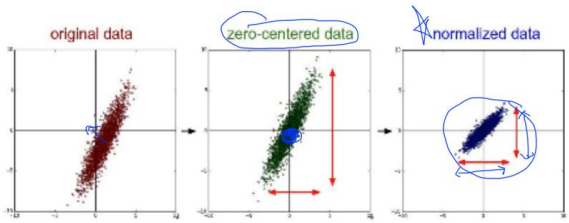
- Learning Rate
  - 작을 경우: 소요 시간 多, 'local minimum'에 빠질 가능성 多 ➔ scheduler 사용!!
  - 클 경우: 발산 가능성 多

- 데이터 전처리 필요!!

x1	x2	y
1	9000	A
2	-5000	A
4	-2000	B
6	8000	B
9	9000	C



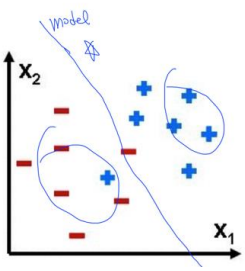
  - Scale 차이로 인해 진동 발생! ➔ gradient는 모든 feature에 동일하게 적용되기 때문에 위 그림에서 W_2가 영향을 많이 받음…!



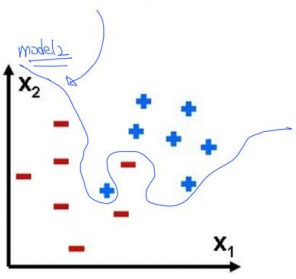
Standardization: $x_j' = \frac{x_j - \mu_j}{\sigma_j}$

- Overfitting: 학습 데이터에 대한 성능은 좋지만, 테스트 데이터에 대한 성능은 떨어짐
  - 일반화 X ➔ 해결책: 학습 데이터 수 증가, 불필요한 feature 수 감소, 규제 사용

Good model



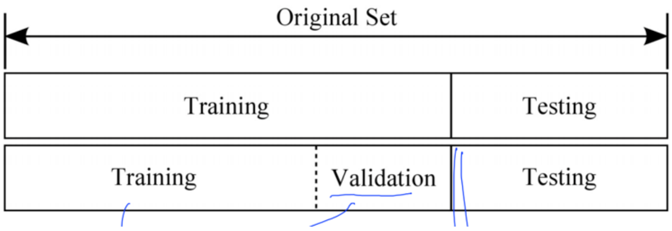
Overfitting model



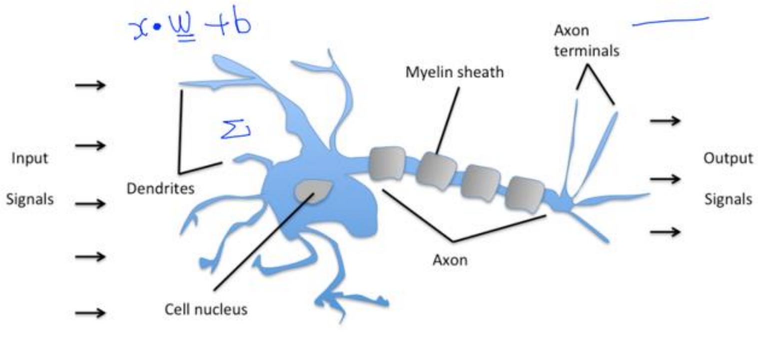
- Regularization: W가 너무 크게 설계되지 않도록!

  - W의 크기가 작은 녀석들은 Regulation term을 이용해 gradient를 크게 하여 상대적으로 크게 갱신

- 학습 데이터와 테스트 데이터를 구분하여 사용!

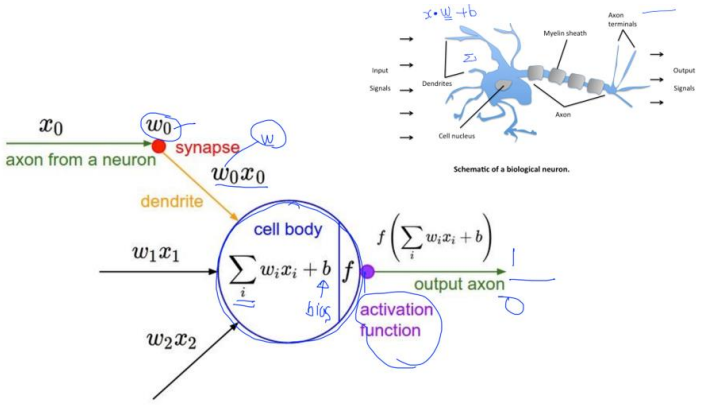


- Perceptron(= 뉴런) → 선형 분류기, Feed forward Network



Schematic of a biological neuron.

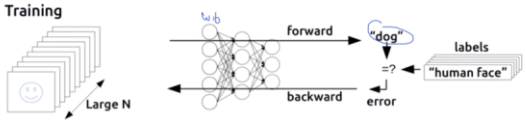
- 활성화 함수



- Backpropagation (역전파): 역방향으로 오차를 전파시키면서 각층의 가중치를 업데이트하고 최적의 학습 결과를 찾아가는 방법 → 문제점: Gradient Vanishing

History of MLP training

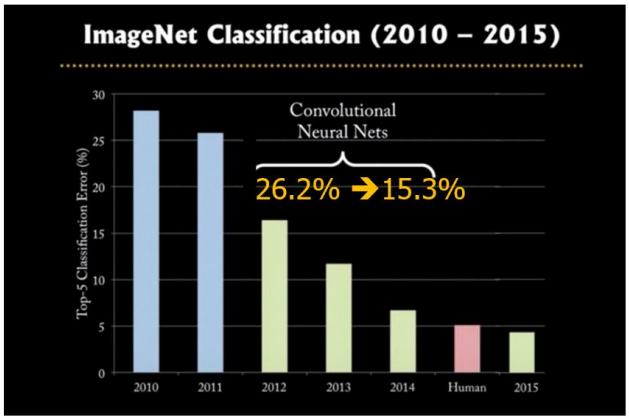
• Backpropagation (1974, 1982 by Paul Werbos, 1986 by Hinton)



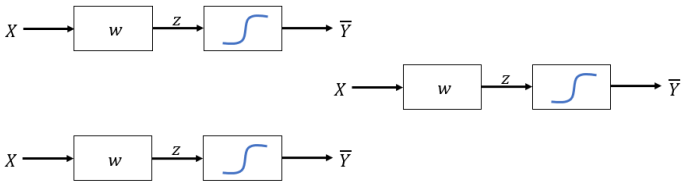
Re-discovery: Backpropagation by Hinton

이후로, XOR보다 더 복잡한 문제를 풀기 시작함

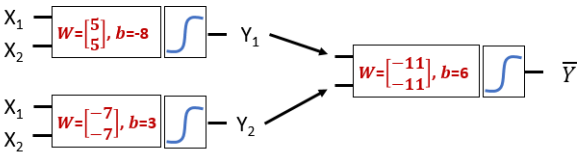
- Deep Learning → 3대 거장: Hinton, Bengio, Lecun
  - Weights, Bias 가 적절하게 초기화되는 것 + dropout 통해 깊게 쌓을 수 있는 것 → 중요!!
  - ImageNet Classification: Computer Vision 연구자들이 모여서 1000개의 클래스를 맞추는 대회 → 2010~15 획기적인 성능 향상 ← Convolutional Neural Nets



- Perceptron → 은닉층 1개 → XOR 문제 해결 불가
  - "MLP" → 은닝층 2개 이상 → Perceptron을 쌓아서 해결

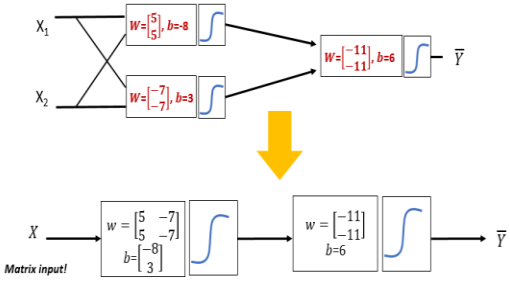


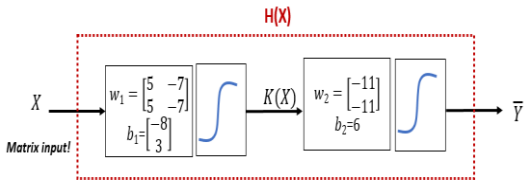
• XOR using NNs



X_1	X_2	Y_1	Y_2	$\overline{Y}$	XOR
0	0	0	1	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	0	0	0

• Forward Propagation을 통한 XOR 문제 풀이 가능 검증 완료



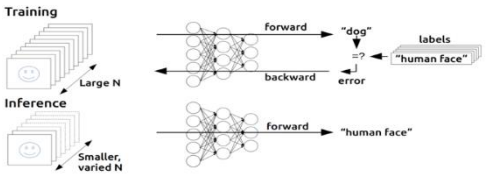


$K(X) = \text{sigmoid}(X \cdot W_1 + b_1)$

$\overline{Y} = H(X) = \text{sigmoid}(K(x) \cdot w_2 + b_2)$

- 그렇다면 학습을 통해 적절한 W1, W2, B1, B2와 같은 파라미터를 어떻게?
  - 우리는 'Gradient Descent'와 같은 방법을 사용했다!
  - 하지만, 'MLP'는 경사하강법을 적용하기에는 너무 복잡 → Backpropagation 이용!

• **Backpropagation** (1974, 1982 by Paul Werbos, 1986 by Hinton)



1) W, b 초기화
2) Forward 계산 & Error 측정
3) Error 값 을 Backward 하면서 w, b 업데이트

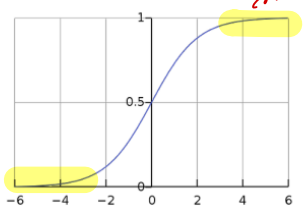
- Problem

  층을 깊게 쌓아서 버전!

  - Neural Network(NN)을 이용 → XOR 문제 해결
  - DNN → Gradient Vanishing 문제 발생
  - 우리의 문제점
    - 적절하지 않은 Weight 초기화 방법 & 활성화 함수 사용

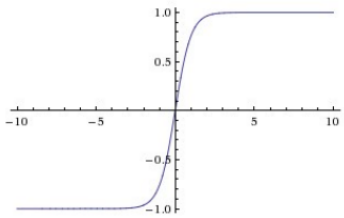
- Activation Function(활성화 함수)
  - 신경학적으로 볼 때, 뉴런 발사의 과정에 해당 → 최종 신호를 다음 뉴런으로 보내줄지 말지 결정하는 역할 ex) sigmoid ➔ 0.5를 기준
  - Step: 입력이 양수 일 때는 1, 음수 일 때는 0의 신호를 보내주는 이진 함수 ➔ 미분 불가능한 함수로 모델 Optimization 과정에 사용이 어려워 신경망의 활성 함수로 사용하지 않음
  - Sigmoid
    - ❑ 단일 퍼셉트론(perceptron)에서 사용했던 활성 함수
    - ❑ 입력을 (0,1) 사이로 정규화(normalization) 함
    - ❑ Backpropagation 단계에서 NN layer 를 거칠 때마다 작은 미분 값이 곱해져, Gradient Vanishing 을 야기함. 여러 개의 Layer를 쌓으면 신경망 학습이 잘 되지 않는 원인
    - ❑ Deep Layer (3개 이상)에서 활성 함수로 사용을 권하지 않음

    가운데가 0에 가까워짐.

    $$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

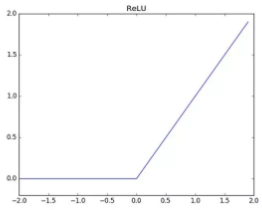
  - Tanh
    - ❑ Sigmoid를 보완하고자 제안된 활성 함수
    - ❑ 입력을 (-1, 1)사이의 값으로 정규화(normalization) 함
    - ❑ Sigmoid 보다 tanh 함수가 전반적으로 성능이 좋음
    - ❑ 여전히 Gradient Vanishing 문제는 발생함 (Sigmoid 보다는 덜 발생함)

    $$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- ReLU (Rectified Linear Unit)

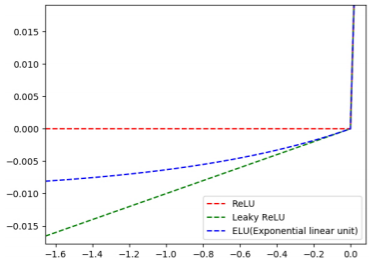
❑ 현재 가장 인기있는 활성화 함수
❑ 양수에서 Linear Function과 같으며 음수는 0을 출력하는 함수
❑ 미분 값을 0 또는 1의 값을 가지기 때문에 Gradient Vanishing 문제가 발생하지 않음
❑ Linear Function과 같은 문제는 발생하지 않으며, 엄연히 Non-Linear함수 이므로 Layer를 deep하게 쌓을 수 있음.
❑ exp() 함수를 실행하지 않아 sigmoid함수나 tanh함수보다 6배 정도 빠르게 학습이 진행됨



$Relu(x)=max(0,x)$

- Leaky ReLU

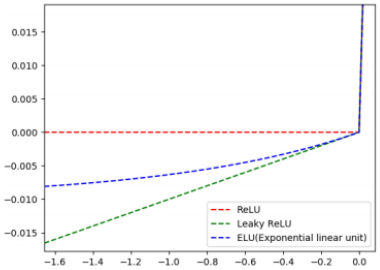
❑ Leaky ReLU는 "dying ReLU" 현상을 해결하기 위해 제시된 함수
❑ ReLU는 x<0인 경우 함수 값이 0이지만, Leaky ReLU는 작은 기울기를 부여함
❑ 보통 작은 기울기는 0.01을 사용함
❑ Leaky ReLU로 성능향상이 발생했다는 보고가 있으나 항상 그렇지는 않음



$Leaky\ Relu(x)=max(0.01*x,x)$

- ELU

❑ ReLU의 threshold를 -1로 낮춘 함수를 exp^x를 이용하여 근사한 것
❑ dying ReLU 문제를 해결함
❑ 출력 값이 거의 zero-centered에 가까움
❑ 하지만 ReLU, Leaky ReLU와 달리 exp()를 계산해야하는 비용이 비쌈



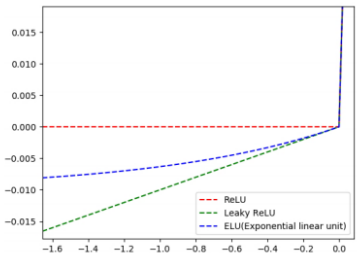
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

- Maxout

❑ 이 함수는 ReLU와 Leaky ReLU를 일반화 한 것. ReLU와 Leaky ReLU는 이 함수의 특수한 경우

❑ Maxout은 ReLU가 갖고 있는 장점을 모두 가지며, dying ReLU 문제도 해결

❑ ReLU 함수와 달리 한 뉴런에 대해 파라미터가 두배이기 때문에 전체 파라미터가 증가한다는 단점이 있음



$$f(x) = max(w_1^T x + b_1 + w_1^T x + b_1)$$

⇨ 적용 순서: **Maxout > ELU, Leaky ReLU >= ReLU > tanh >= sigmoid**

- ■ 가장 먼저 ReLU를 시도
- ■ 다음으로 Leaky ReLU, Maxout, ELU를 시도
- ■ Tanh를 사용해도 되지만 성능이 개선될 확률이 적음
- ■ 앞으로 Deep NN에서는 Sigmoid는 피한다!

- Weight Initialization
  - Neural Network에서는 weight 선정에 주의 요망
  - W=0이면 Backpropagation시 gradient값이 0되어 Gradient Vanishing 현상이 발생 ➔ 절대 모두 0으로 초기화하지 말 것
  - 노드의 입출력 수에 비례해서 초기값을 결정짓는 방법 제안 ➔ Xavier initialization (교수님께서 주로 사용하시는 방법), He's initialization

Xavier/He initialization

- Makes sure the weights are 'just right', not too small, not too big
- Using number of input (fan_in) and output (fan_out)

```
# Xavier initialization
# Glorot et al. 2010
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)

# He et al. 2015
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

Pytorch 초기화 방법 ➔ https://pytorch.org/docs/stable/nn.init.html

Xavier initialization

`torch.nn.init.xavier_normal_(tensor, gain=1.0)` [SOURCE]

Fills the input *Tensor* with values according to the method described in *Understanding the difficulty of training deep feedforward neural networks* - Glorot, X. & Bengio, Y. (2010), using a normal distribution. The resulting tensor will have values sampled from $\mathcal{N}(0, \text{std}^2)$ where

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

Also known as Glorot initialization.

Parameters

- **tensor** – an n-dimensional *torch.Tensor*
- **gain** – an optional scaling factor

Examples

```
>>> w = torch.empty(3, 5)
>>> nn.init.xavier_normal_(w)
```

He initialization

`torch.nn.init.kaiming_normal_(tensor, a=0, mode='fan_in', nonlinearity='leaky_relu')` [SOURCE]

Fills the input *Tensor* with values according to the method described in *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification* - He, K. et al. (2015), using a normal distribution. The resulting tensor will have values sampled from $\mathcal{N}(0, \text{std}^2)$ where

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

Also known as He initialization.

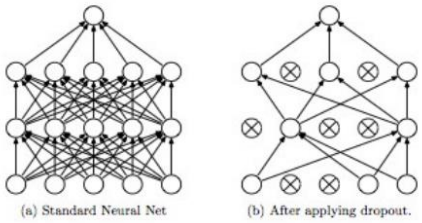
Parameters

- **tensor** – an n-dimensional *torch.Tensor*
- **a** – the negative slope of the rectifier used after this layer (only
- **with 'leaky_relu')** (*used*) –
- **mode** – either `'fan_in'` (default) or `'fan_out'`. Choosing `'fan_in'` preserves the magnitude of the variance of the weights in the forward pass. Choosing `'fan_out'` preserves the magnitudes in the backwards pass.
- **nonlinearity** – the non-linear function (*nn.functional* name), recommended to use only with `'relu'` or `'leaky_relu'` (default).

- **Overfitting:** 너무 과도하게 학습 데이터에 대해 모델을 learning 한 경우
  - 새로운 데이터에 대한 대응력이 없어 모델 학습 의미 상실
  - 해결책: 많은 양의 학습 데이터 + feature 수 감소 + Dropout과 같은 규제 사용

Dropout for overfitting → 과적합 문제 해결

- 훈련 데이터에 대한 복잡한 공동 적응을 방지하여 신경망의 과적 합을 줄이기 위한 Google이 제안한 정규화 기술 (regularization)
- "드롭 아웃"이라는 용어는 신경망에서 유닛을 제거하는 것
- 학습 시에만 적용하고 테스트 시에는 모든 유닛을 사용함



  - 왜 성능향상? Dropout → Ensemble model(집단지성과 유사) 학습과 같은 효과

Examples:

```
>>> m = nn.Dropout(p=0.2)
>>> input = torch.randn(20, 16)
>>> output = m(input)
```

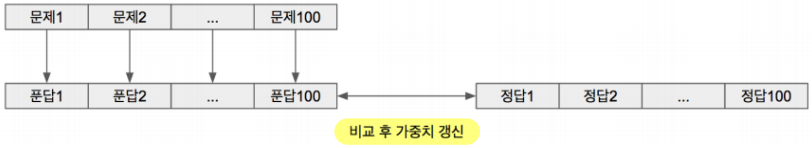
  - Ensemble (앙상블 학습법): 학습 알고리즘들을 따로 쓰는 경우에 비해 더 좋은 예측 성능을 얻기 위해 다수의 학습 알고리즘을 사용하는 방법

- Summary

• DNN 모델 학습을 위한 팁

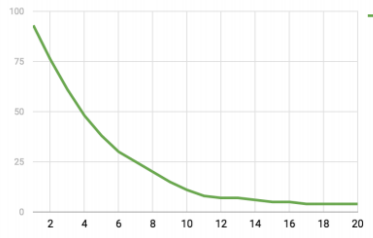
  • 활성 함수를 잘 선택한다.
    • ReLU가 가장 널리 사용된다.
  • 가중치 초기화 방법을 잘 선택한다.
    • Xavier가 가장 널리 사용된다
  • 드랍 아웃을 잘 적용한다.
    • "NN-ReLU-Dropout"을 하나의 블락으로 쌓는다.
  • BN을 잘 적용한다.
    • "NN-ReLU-BN"을 하나의 블락으로 쌓는다.

- Batch_size (배치 사이즈): 몇 문항을 풀고 해답을 맞추는 지를 의미



  - 배치 사이즈가 큰 경우, 기억 용량 곧, GPU 메모리가 엄청 커야 함
  - 배치 사이즈가 작은 경우, 학습은 꼼꼼히 잘되지만 많은 시간 소요!

- Epochs (에포크) → 같은 데이터셋으로 반복적으로 가중치를 갱신하면서 모델이 학습됨



그래프에서 세로축이 100문항 중 틀린 개수이고, 가로축이 모의고사 풀이 반복횟수를 의미함.

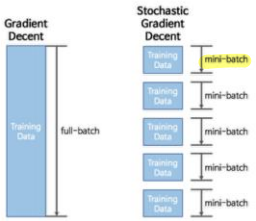
풀이를 반복할수록 틀린 개수가 적어지는 것을 보수 있음.

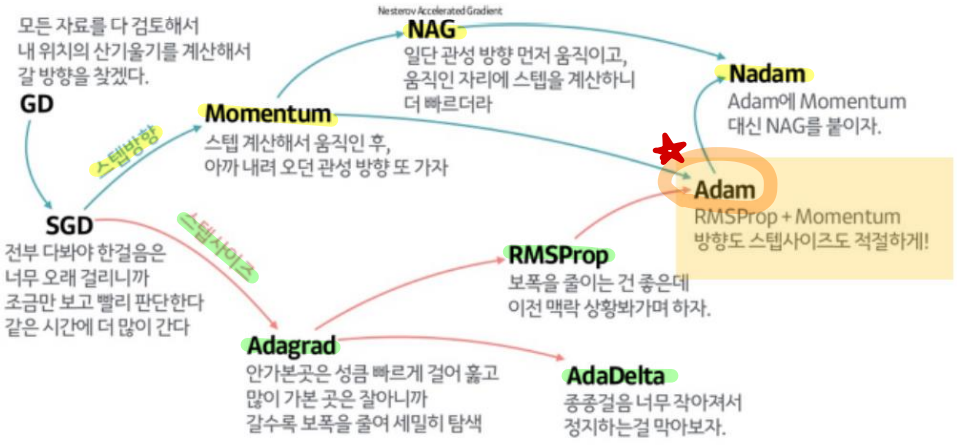
처음에는 틀린 개수가 확 줄어들지만, 반복이 늘어날수록 완만하게 틀린 오답 수가 줄어듬.

우리가 공부할 때도 낮은 점수에서는 공부를 조금하면 점수가 확 오르지만, 높은 점수에서 1~2점 올리는 것이 쉽지 않은 것과 비슷

- Optimizer: 기본

  - Gradient Descent (GD)
    - 학습데이터 전체를 사용하는 최적화
  - Stochastic Gradient Descent (SGD)
    - 학습데이터 일부(mini-batch)를 사용하는 최적화
    - BGD보다 빠르게 수렴
    - SGD를 여러 번 반복할 수록 BGD와 유사한 결과로 수렴

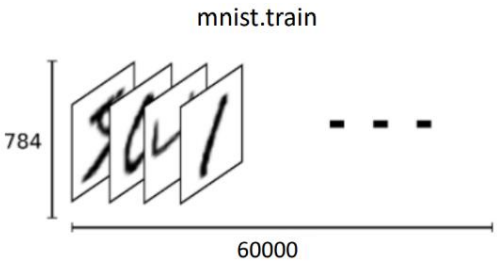




From 하용호

- MNIST Dataset 소개

학습데이터: 60000x784
학습데이터 라벨: 60000x10
테스트데이터: 10000x784
테스트데이터 라벨: 10000x10



- 다양한 실험

  - Random Init / NN Layer #1 (784=>10) / SGD / CrossEntropy

  ⇨ https://colab.research.google.com/drive/15ArrVdJELyV-PeuATH5poe2uZoHm6Ih8

  - Random Init / NN Layer #1 (784=>10) / Adam / CrossEntropy

  ⇨ https://colab.research.google.com/drive/1bimNZtout48XzSy7VoWbmdpOHQ-SAkHX

  - Random Init / NN Layer #3 (784=>256=>10) / Adam / CrossEntropy

  ⇨ https://colab.research.google.com/drive/1jsP6IkoSZW5roMip53QSPJjiVP6Ej3Zk

  - Xavier Init / NN Layer #3 (784=>256=>10) / Adam / CrossEntropy

  ⇨ https://colab.research.google.com/drive/1CI11CA5otqB7-RsQLKak3LJW3xeCUr9q

  - Xavier Init / DNN Layer #5 (784=>256=>10) / Adam / CrossEntropy

  ⇨ https://colab.research.google.com/drive/1y9qF3D4vbQVhX_dZCgplRuw_EcgW8Sg2

  - Xavier Init / DNN Layer #5 (784=>256=>256=>256=>10) / Adam / CrossEntropy / dropout (0.3)

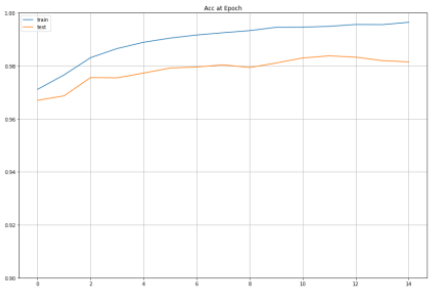
  ⇨ https://colab.research.google.com/drive/15Q5GSAPgsHeR0Y70zaiigHBBo4kwJ86o

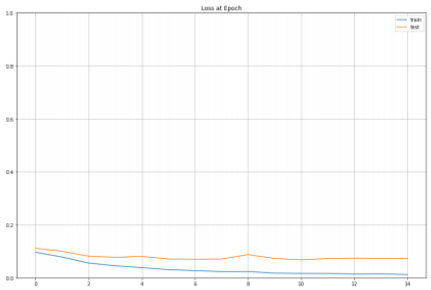
SGD	Adam	NN	Xavier	DNN	Dropout
0.42	0.77	0.94	0.9791	0.9824	0.9776

- 학습 과정 Plot 하기

  - Xavier Init / NN Layer #5 (784=>256=256=>256=>10) / Adam / CrossEntropy / dropout (0.3)

    ⇨ https://colab.research.google.com/drive/1EzvjlRNjMvo4ECnFuxx6ReuXih_vPA8k





Test Accuracy: 0.9815