Introduction and Empirical Evaluation of Parametric ReLU

이찬희

목차

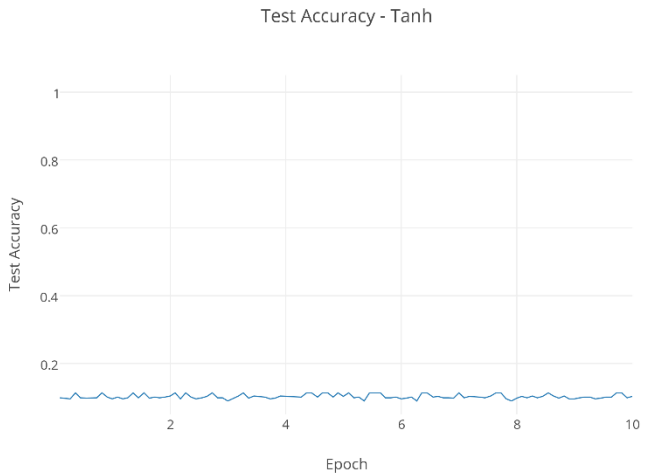
- Vanishing Gradient Problem
- PReLU Activation Function
- 실험 결과

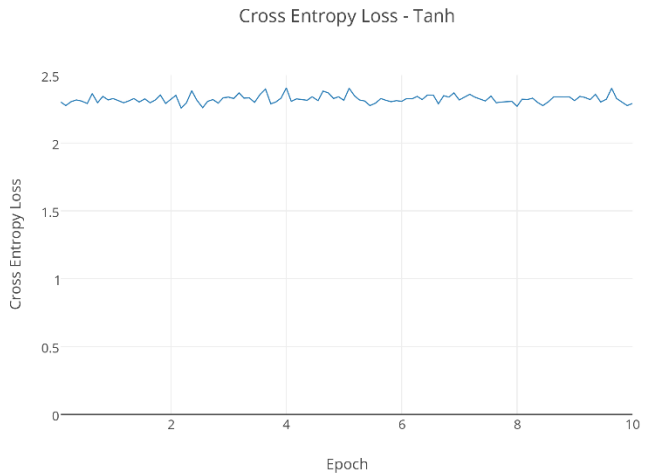
Deep Learning의 강점

- 경험적 증거
  - 저층에서 고층으로 갈수록 이전 층보다 점진적으로 고차원적인 특징들을 추출
  - State-of-the-art 모델들은 대부분 deep structure 사용
    - PReLU Net: 22 layers CNN, 4.94% error on ImageNet (He at al., 2015)
    - ResNet: 152 layers CNN, 3.57% error on ImageNet (He at al., 2016)
- 이론적 증거
  - Single layer perceptron은 XOR gate 학습이 불가능
  - 모든 양의 정수 k에 대하여, $\Theta(k^3)$개의 layer, 각 layer마다 $\Theta(1)$의 node, $\Theta(1)$개의 parameter를 가지는 Neural Network(NN; 신경망)은 학습할 수 있지만, $O(k)$개의 layer, 각 layer마다 $O(2^k)$의 node를 가진 NN으로는 학습할 수 없는 데이터가 존재 (Telgarsky, 2016)

Deep Learning의 걸림돌

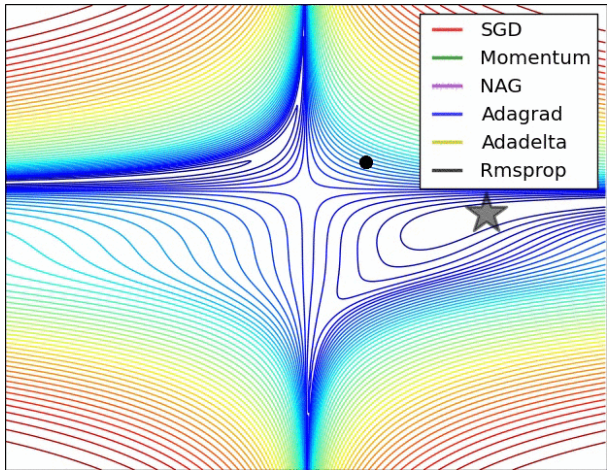
- Vanishing/exploding gradient problem
  - NN을 훈련 시키는데 사용하는 gradient descent 계열 알고리즘에서 비롯되는 문제
  - NN의 layer가 많아질수록 저층 layer의 parameter update가 너무 느려지거나(vanishing) 너무 급격해져서(exploding) 전체 NN이 수렴하지 못함





Gradient Descent Algorithms

- 함수 $F(x)$의 local minimum을 찾는 알고리즘
- 흔히, 산을 내려갈 때 현재 지점에서 가장 경사가 가파른 방향으로 이동하는 것으로 비유
- 한 점 $a = (x_1, x_2, \dots, x_n)$에서 시작하여 아래 과정을 일정 횟수 동안 혹은 $F(a)$의 변화가 threshold보다 작아질 때까지 반복
  - 각 x_k $(1 \le k \le n)$에 대하여 $F(x)$의 partial derivative $\frac{dF}{dx_k}$ (gradient) 계산
  - $x_k := x_k - \gamma \frac{dF}{dx_k}$ (γ: learning rate)

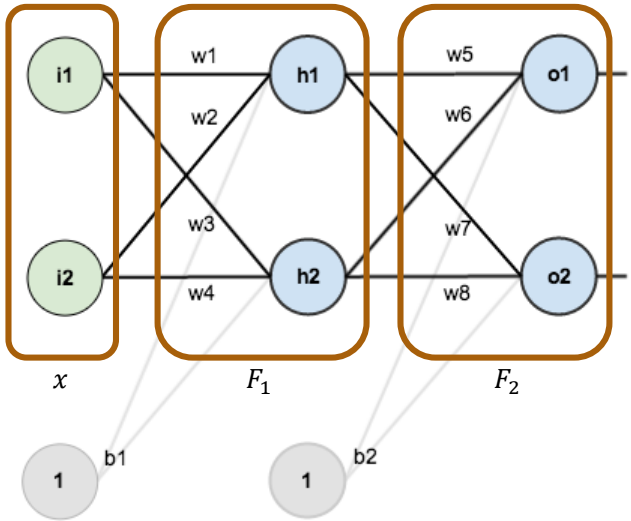


Gradient Descent Algorithms

- Weight matrices W, bias matrices b 등 learnable parameter들의 집합 θ로 이루어진 NN $F_\theta(x)$에 대하여 입력 벡터 x와 정답 y가 주어졌을 때, y와 $F_\theta(x)$의 차이를 수치화하는 loss function $E(F_\theta, x, y)$을 정의
- Gradient descent 알고리즘을 사용하여 $E(F_\theta, x, y)$을 최소화하는 θ를 계산
- Back Propagation: Gradient descent 알고리즘을 사용하기 위하여 θ의 모든 원소들에 대한 $E(F_\theta, x, y)$의 partial derivative(gradient)을 계산하는 과정
- $E(F_\theta, x, y)$이 매우 복잡할 수 있으므로 Chain Rule을 사용하여 단계적으로 gradient를 계산

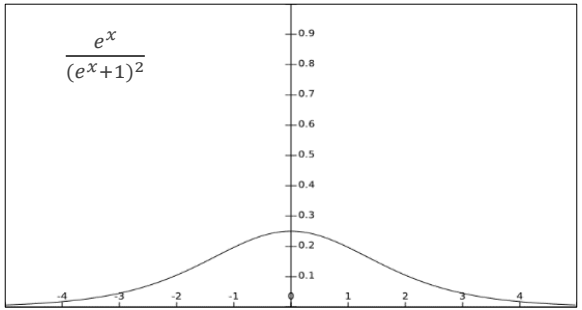
Chain Rule

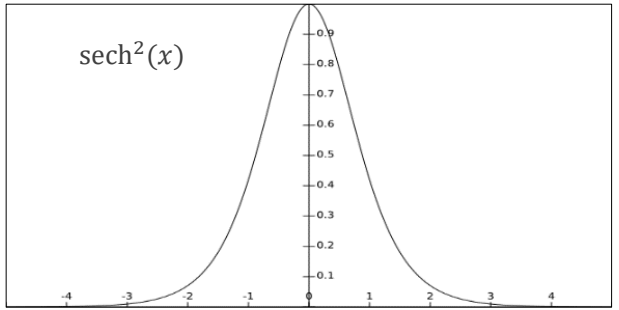
- $F(x) = F_2(F_1(x)) = F_2 \circ F_1(x), \frac{dF}{dx} = \frac{dF_2}{dF_1} \cdot \frac{dF_1}{dx}$
- $F_1(x) = (out_{h1}, out_{h2})$
- $net_{h1} = i1 \times w1 + i2 \times w2 + b1$
- $out_{h1} = \sigma(net_{h1})$, σ: activation function
- $net_{o1} = out_{h1} \times w5 + out_{h2} \times w6 + b2$
- $E_{total} = E(out_{o1}, y1) + E(out_{o2}, y2)$
- ...
- $\frac{dE_{total}}{dw1} = \frac{dE}{dout_{o1}} \cdot \frac{dout_{o1}}{dout_{h1}} \cdot \frac{dout_{h1}}{dw1} + \frac{dE}{dout_{o2}} \cdot \frac{dout_{o2}}{dout_{h1}} \cdot \frac{dout_{h1}}{dw1}$
- $\frac{dout_{o1}}{dout_{h1}} = \frac{dout_{o1}}{dnet_{o1}} \cdot \frac{dnet_{o1}}{dout_{h1}} = \boldsymbol{\frac{d\sigma(net_{o1})}{dnet_{o1}}} \cdot \frac{dnet_{o1}}{dout_{h1}}$



Activation Functions

- 주로 sigmoid function과 hyperbolic tangent function을 activation function으로 사용
  - Sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$, $\frac{df}{dx} = \frac{e^x}{(e^x+1)^2} \Rightarrow 0 < \frac{df}{dx} \leq 0.25$
  - Hyperbolic tangent function: $f(x) = \tanh(x)$, $\frac{df}{dx} = \mathrm{sech}^2(x) \Rightarrow 0 < \frac{df}{dx} \leq 1$
- Chain Rule을 사용하여 gradient를 계산하는 과정에서 0과 1 사이의 수를 반복하여 곱하게 됨
- Deep structure의 경우 저층 layer의 gradient가 너무 작아져서 parameter update가 매우 느려짐



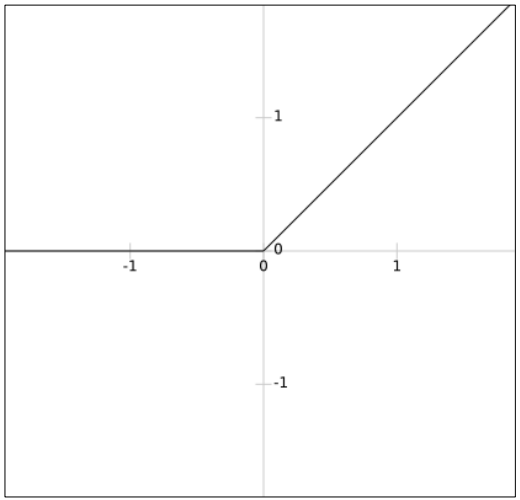


해결책

- NN의 각 layer를 독립적으로 Restricted Boltzmann Machine, Denoising Auto Encoder와 같은 unsupervised method를 사용하여 pre-training한 후, 전체 NN을 supervised method를 사용하여 훈련 (e.g. Deep Belief Network (Hinton et al., 2006))
- Vanishing/exploding gradient effect를 최소화할 수 있는 값으로 각 layer를 초기화 (e.g. Xavier initialization (Glorot et al., 2010))
- 더 빠른 하드웨어를 사용하여 NN을 많은 epoch동안 훈련
- **$f'(x)$가 1인 구간이 있는 activation function 사용**

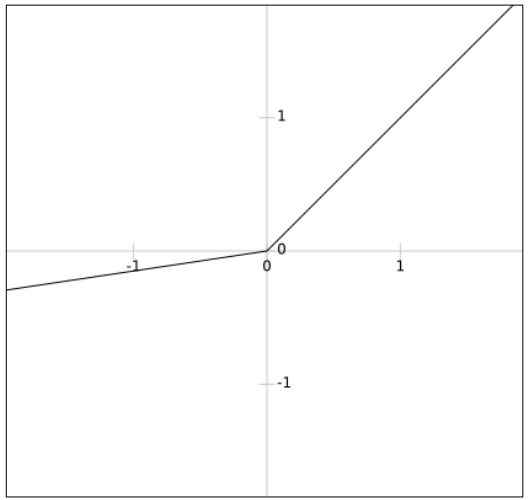
Rectified Linear Units

- Rectified Linear Unit(ReLU) (Nair et al., 2010)
  - $f(x) = \begin{cases} x & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases}$ 혹은 $f(x) = \max(0, x)$
  - $x \geq 0$인 경우 $f'(x) = 1$이므로 deep NN의 저층 layer의 gradient를 계산할 때 gradient가 소실되거나 증폭되는 문제가 완화됨
- ReLU의 단점
  - $x < 0$인 경우 x의 값과 상관 없이 $f(x) = 0$이므로 x의 값이 무시됨
  - $x < 0$인 경우 $f'(x) = 0$이므로 전체 gradient가 모두 0이 됨



ReLU의 변형

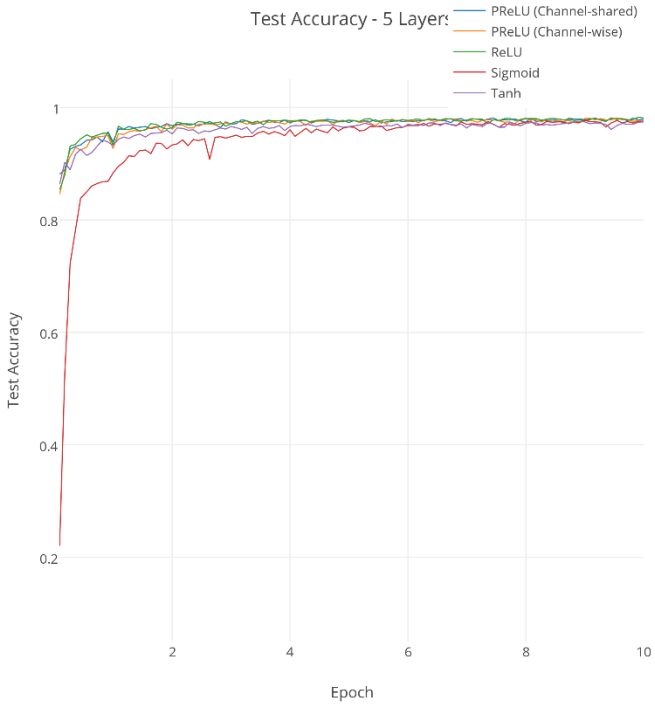
- Leaky ReLU (Maas et al., 2013)
  - $f(x) = \begin{cases} x & if\ x \geq 0 \\ 0.01x & if\ x < 0 \end{cases}$ 혹은 $f(x) = \max(0, x) + 0.01\min(0, x)$
  - ReLU에서 $f(x) = 0$인 부분에서 문제가 발생하므로 $x < 0$인 부분에 작은 경사를 둠
- Parametric ReLU (He at al., 2015)
  - $f(x_i) = \begin{cases} x_i & if\ x \geq 0 \\ a_i x_i & if\ x < 0 \end{cases}$ 혹은 $f(x_i) = \max(0, x_i) + a_i\min(0, x_i)$
  - $x < 0$인 부분의 경사를 trainable parameter로 설정하여 training data에 맞게 activation function의 형태가 변하도록 함
  - Channel-wise: layer의 각 node별로 독립적인 a_i값 사용
  - Channel-shared: 동일한 layer 내의 모든 node가 공통의 a_i값 사용

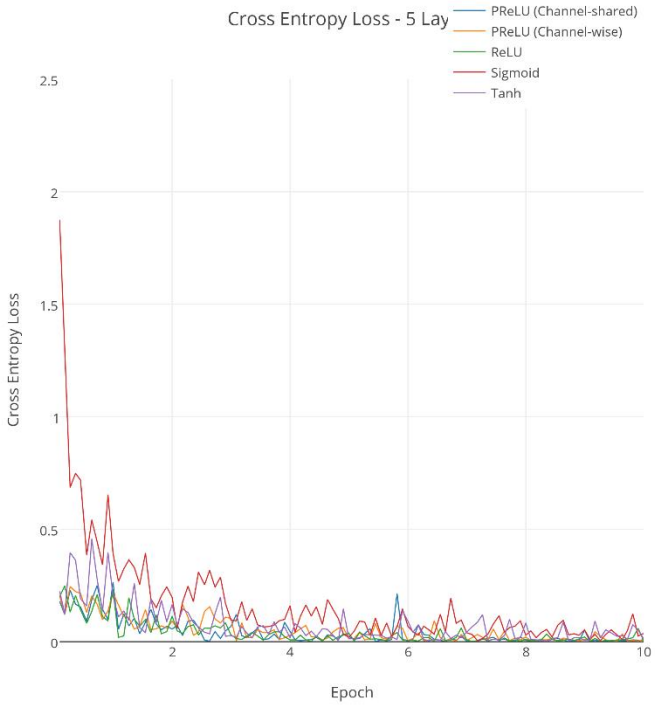


Hyper-parameters

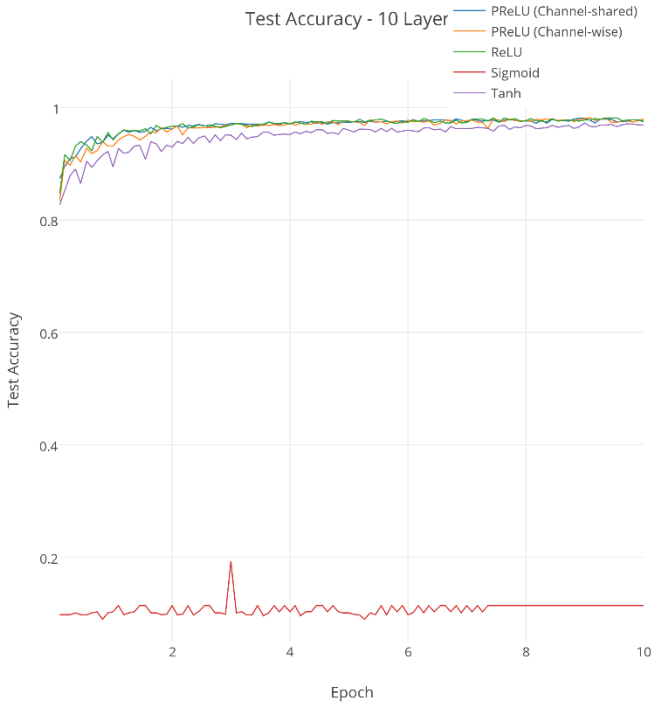
- Training Data: MNIST
- Model: Multilayer Perceptron
- Number of Hidden Layers: 5, 10, 15, 20, 25, 30
- Hidden Layer Size: 500
- Optimization Algorithm: Adam Optimizer (Kingma et al., 2015)
- Training Duration: 10 epoch
- Activation Functions: PReLU (Channel-shared), PReLU (Channel-wise), ReLU, Tanh, Sigmoid
- Implementation: TensorFlow

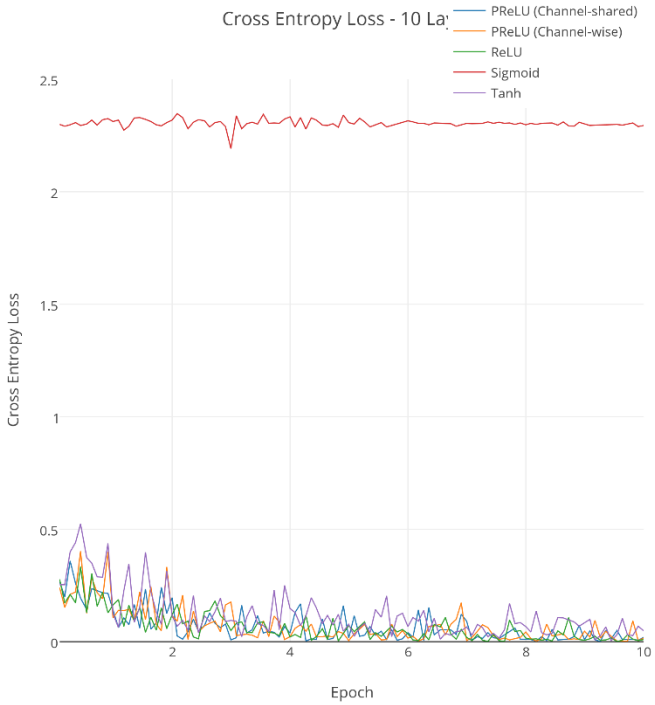
실험 결과 - 5 Layers



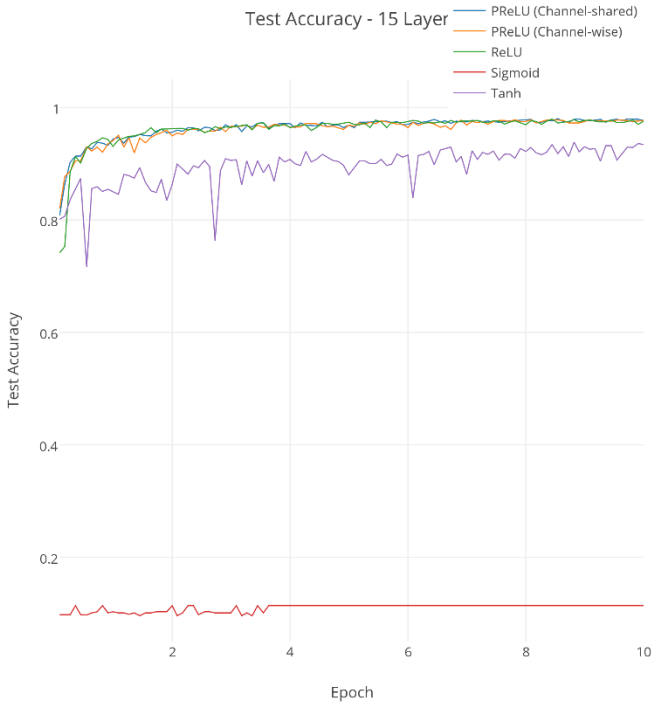


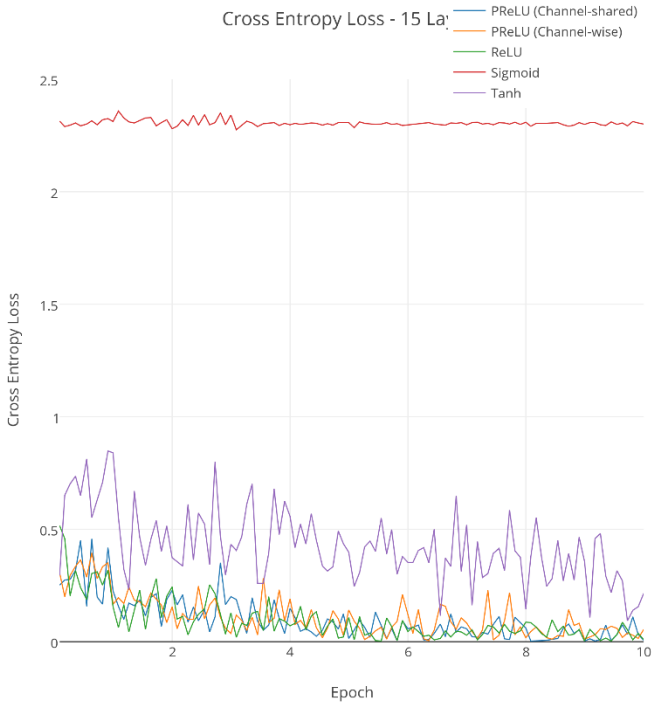
실험 결과 - 10 Layers



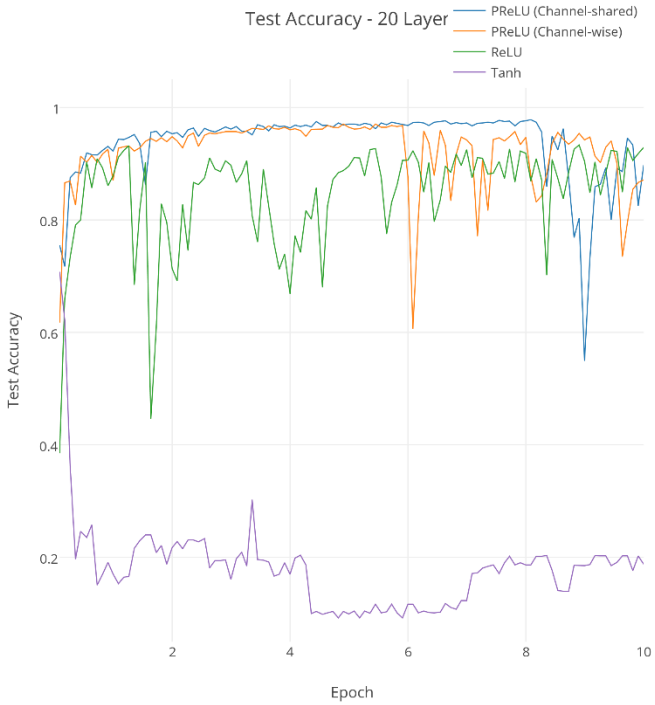


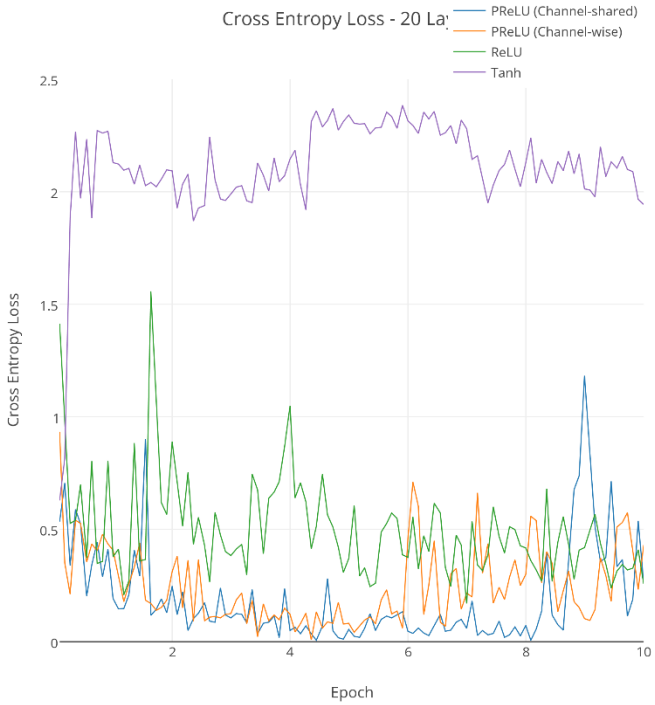
실험 결과 - 15 Layers



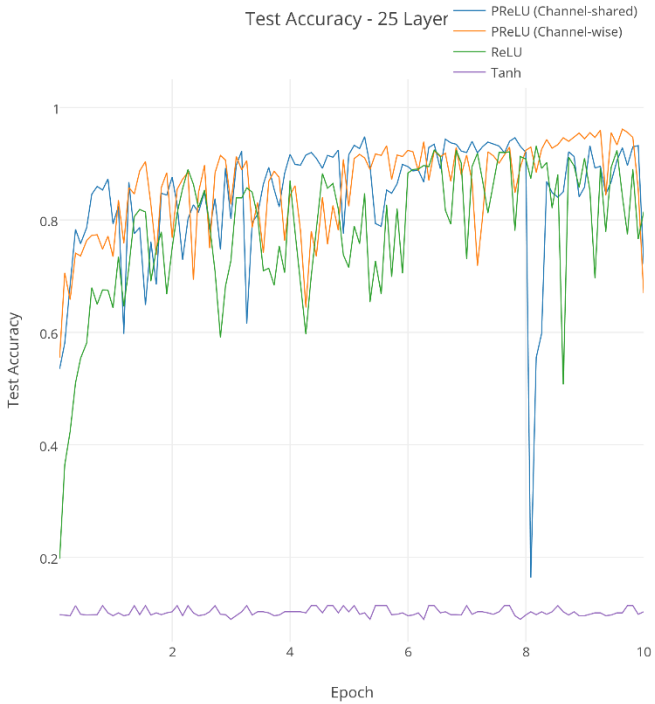


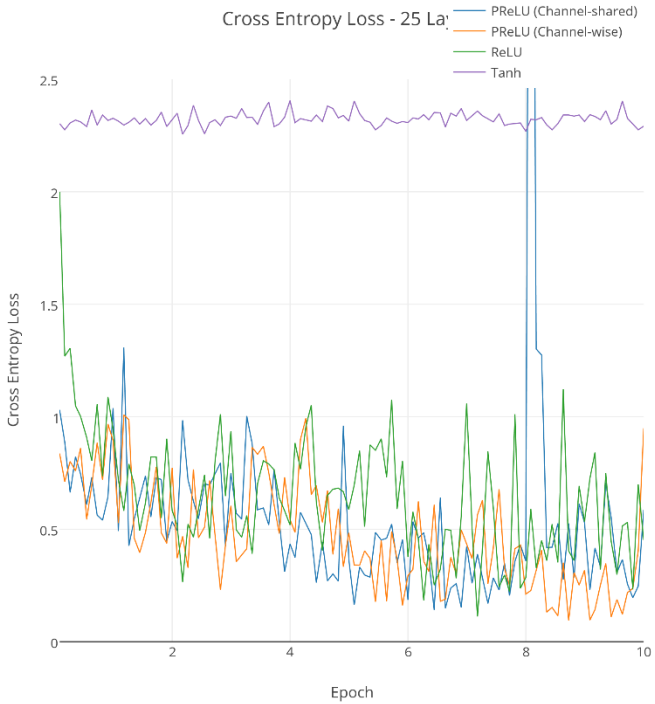
실험 결과 - 20 Layers



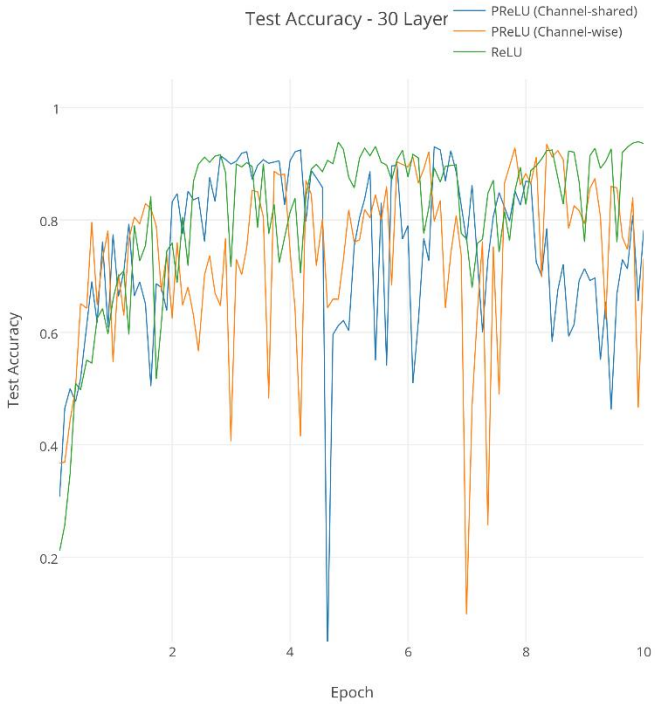


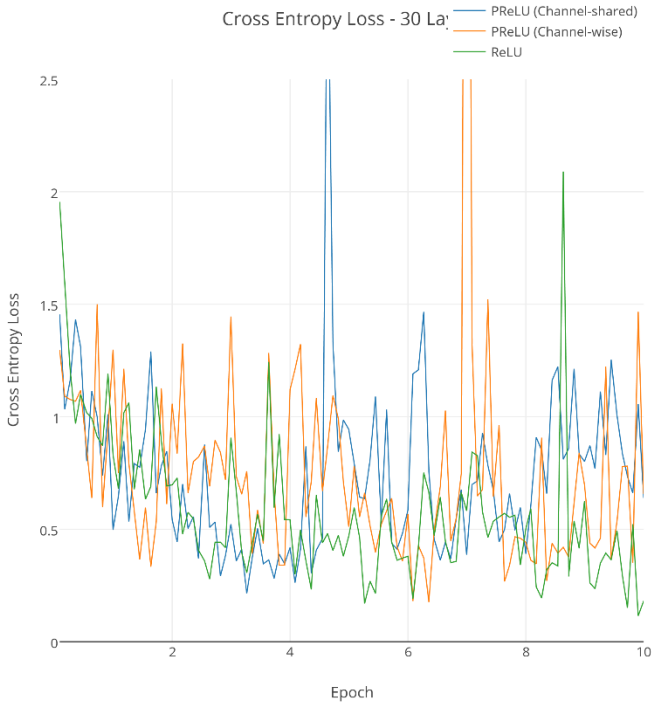
실험 결과 - 25 Layers





실험 결과 - 30 Layers



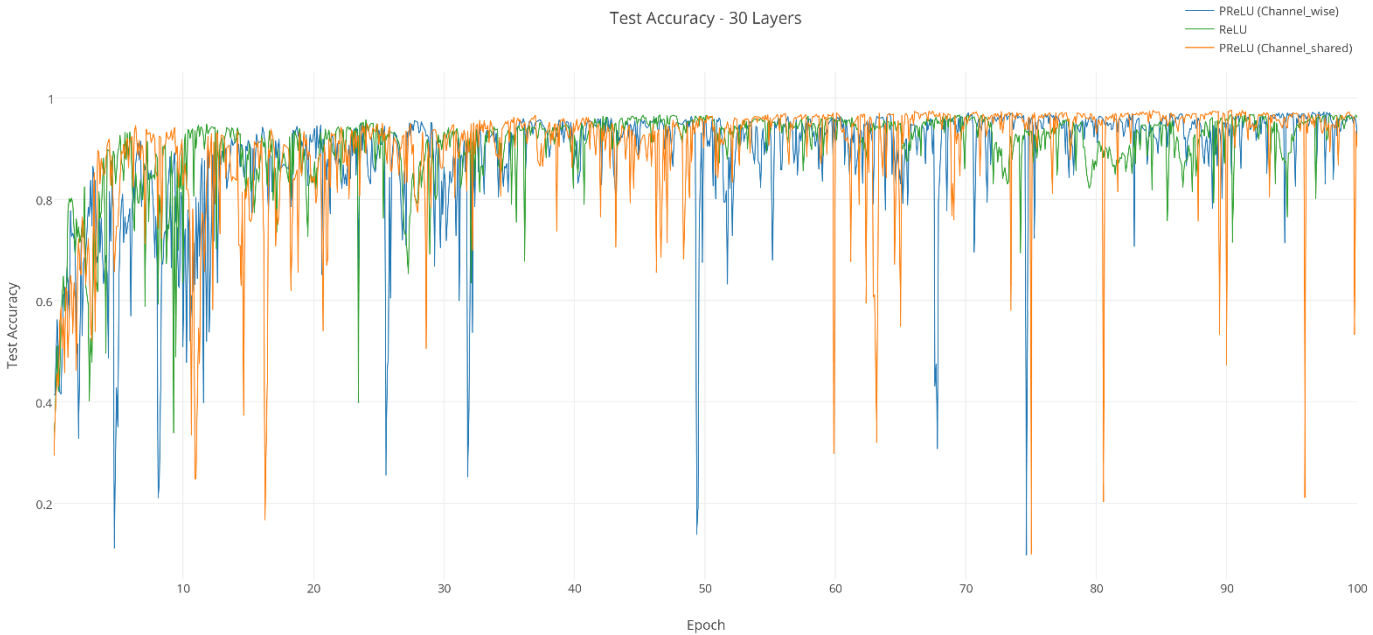


실험 결과 - 종합

	5 Layers	10 Layers	15 Layers	20 Layers	25 Layers	30 Layers
PReLU (Channel-shared)	98.09	98.04	97.95	97.78	94.77	92.96
PReLU (Channel-wise)	98.09	98.16	97.81	97.04	96.13	93.40
ReLU	**98.25**	98.11	97.89	93.33	93.10	93.87
Tanh	97.44	97.21	93.79	70.75	Fail	Fail
Sigmoid	97.69	Fail	Fail	Fail	Fail	Fail

각 실험 별 Test Accuracy 최대값(%). Test Accuracy 20% 미만은 'Fail'로 표시. 같은 layer 수에서의 최대값은 밑줄로, 전체 최대값은 **굵은 글씨**로 표시.

실험 결과 - 30 Layers, 100 Epoch



100 epoch에 걸친 training에 따른 test accuracy의 변화.
Test Accuracy 최대값: 30 Layers PReLU (Channel-wise) 97.23%, 30 Layers PReLU (Channel-shared) **97.56**%, ReLU 96.75%

참고 문헌

- K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In *ICCV*, 2015.
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE CVPR*, 2016.
- M. Telgarsky. Benefits of depth in neural networks. *arXiv: 1602.04485*, 2016.
- G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. In *Neural Computation*, 2006.
- X. Glorot, and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.
- V. Nair, and G. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.
- A. Maas, A. Hannun, and A. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013.
- D. Kingma, and J. Ba. Adam: a method for stochastic optimization. In *ICLR*, 2015.