

메모리, 계산 효율적 딥러닝

Beta-Bernoulli Dropout

Jaehong Yoon

MLAI Lab.

20. 07. 30

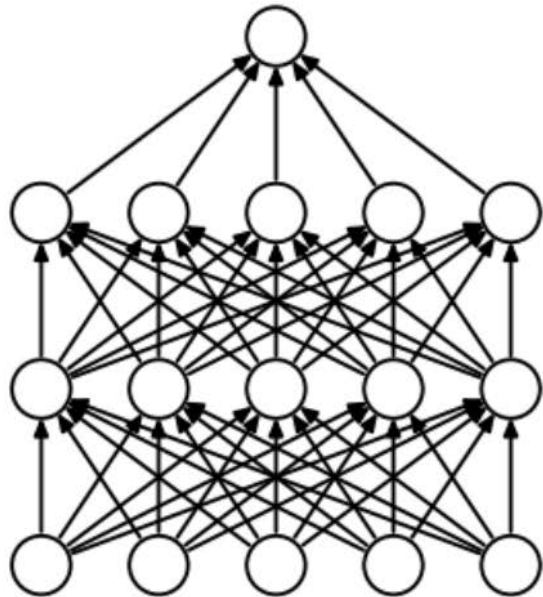
Contents

- Beta-Bernoulli Dropout 이란?
- Backgrounds
- 실습

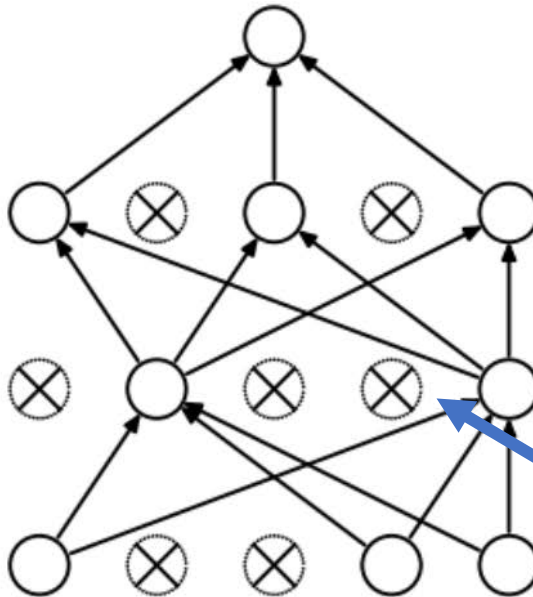
Beta-Bernoulli Dropout

Dropout

랜덤하게 네트워크의 노드를 드랍시켜 오버피팅을 방지하고 학습을 일반화 시킴.



(a) Standard Neural Net



(b) After applying dropout.

$$z_k \sim \text{Ber}(\pi)$$

$$\begin{cases} z_k = 1, & \pi \\ z_k = 0, & 1 - \pi \end{cases}$$

$$W_k * Z_k$$

node dropout mask

Bernoulli Distribution

- 베르누이 분포
- 1회 experiment의 outcome이 성공(1) 혹은 실패(0)의 경우만 가지는 확률 변수 X 의 결과를 모델링한 분포

$$X \sim \text{Bernoulli}(p)$$

$$P(X = 0) = p, P(X = 1) = q = (1 - p)$$

$$P(X = x) = f(x; p) = p^x (1 - p)^{1-x}$$

Beta Distribution

- 두 매개변수 α 와 β 에 따라 $[0,1]$ 구간에서 정의되는 연속 확률 분포들
- 확률 밀도 함수 (PDF)로 정의

✓ (Recap.) PDF ($f(x)$)

✓ 확률 변수의 분포를 나타내는 함수로, 확률 밀도 함수 $f(x)$ 와 구간 $[a,b]$ 에 대해서 확률 변수 X 가 구간에 포함될 확률은

$$P(a \leq X \leq b) = \int_a^b f(x)dx$$

Beta Distribution

Beta 분포의 확률 밀도 함수 (pdf)

두 개의 파라미터, $\alpha > 0$, $\beta > 0$ 를 갖는 분포

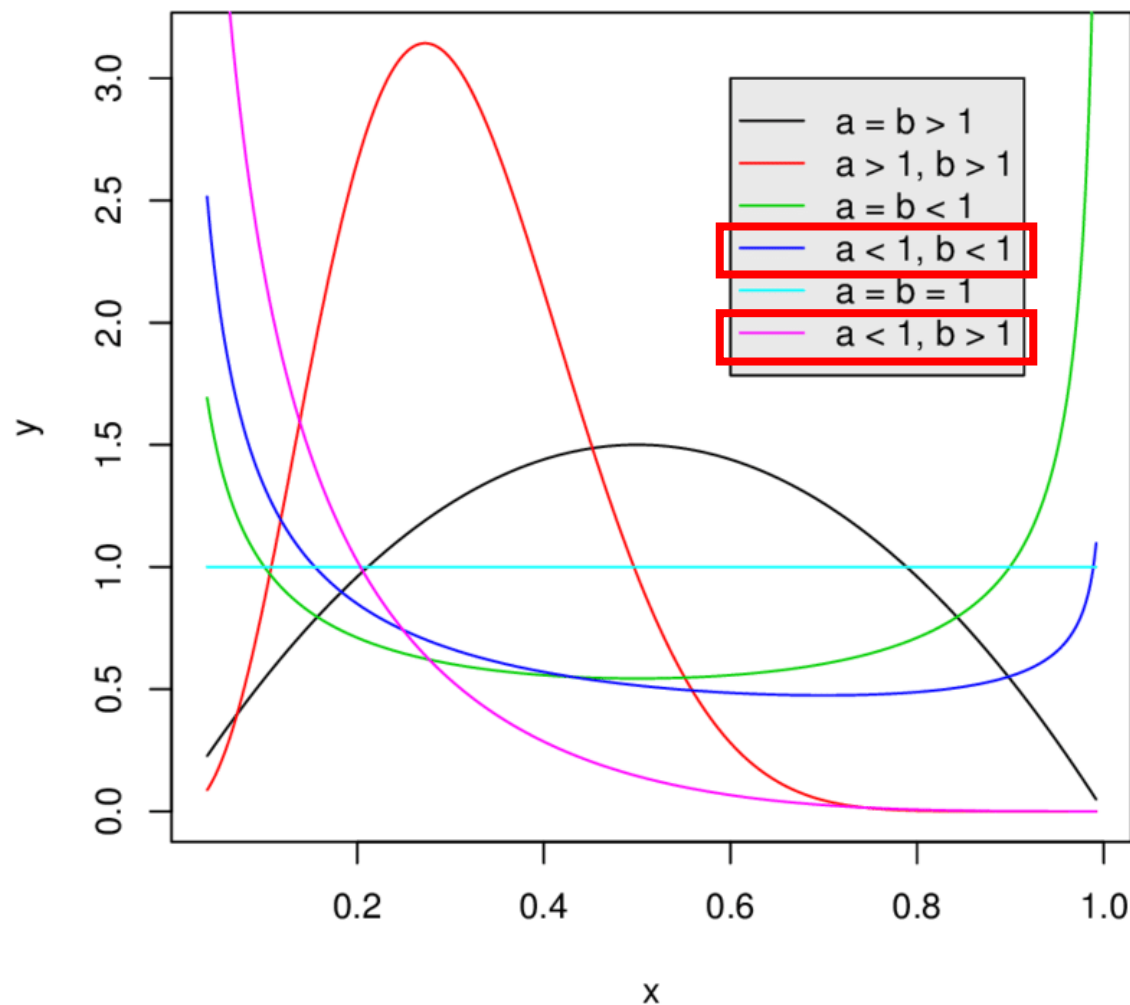
$$f(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1} du}$$

$$f(x; \alpha, \beta) = \begin{cases} \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, & x \in [0, 1] \\ 0, & otherwise \end{cases}$$

Beta Distribution

다양한 α, β 에 따른 Beta 분포

$$f(x; \alpha, \beta) = \begin{cases} \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, & x \in [0, 1] \\ 0, & \text{otherwise} \end{cases}$$

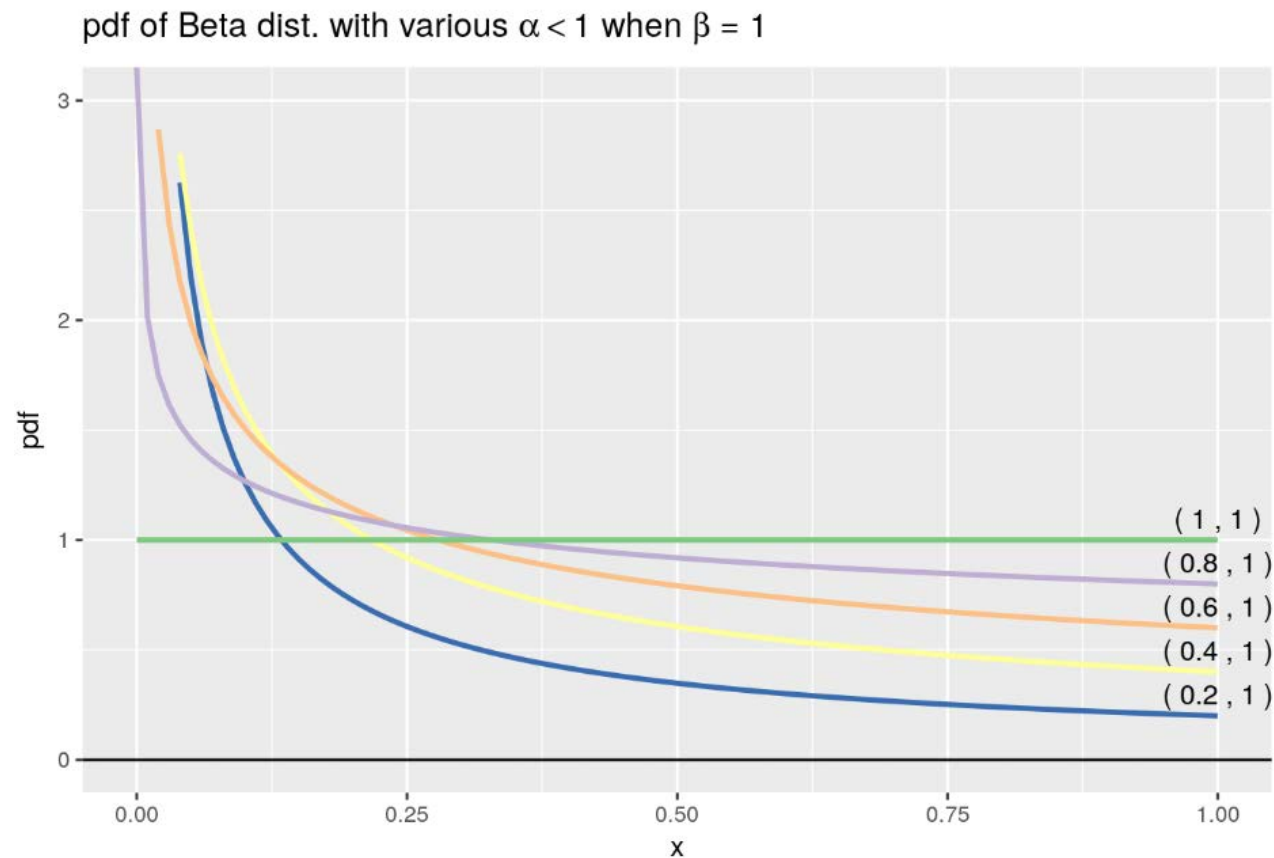


Beta Distribution

다양한 α, β 에 따른 Beta 분포

$$f(x; \alpha, \beta) = \begin{cases} \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, & x \in [0, 1] \\ 0, & \text{otherwise} \end{cases}$$

$\alpha < 1, \beta = 1$
그래프가 왼쪽으로 치우침
(sparsity-inducing prior)



Beta-Bernoulli Distribution

$z \sim \text{Ber}(\pi)$ ← Beta 분포로부터 추출

$$\begin{cases} z = 1, & \pi \\ z = 0, & q = 1 - \pi \end{cases}$$

$$\text{Ber}(\pi) = \pi^z (1 - \pi)^{1-z}$$

BB 분포와 Network Sparsification

Latent Feature Model

$$\mathbf{d}_n = f(\mathbf{W}\mathbf{z}_n) = f\left(\sum_{k=1}^K \boxed{z_{n,k}} \boxed{\mathbf{w}_k}\right)$$

binary mask 1 또는 0 Latent feature

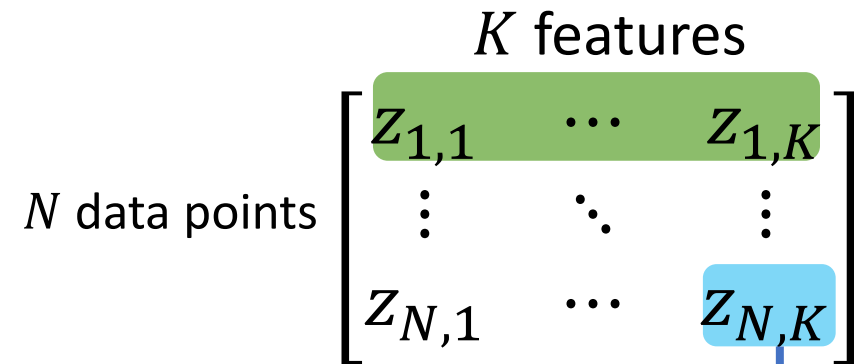
임의의 데이터 \mathbf{d} 는 latent features \mathbf{w}_k 의 조합으로 생성될 수 있다.

BB 분포와 Network Sparsification

Latent Feature Model

$$\mathbf{d}_n = f(\mathbf{W}\mathbf{z}_n) = f\left(\sum_{k=1}^K z_{n,k} \mathbf{w}_k\right)$$

binary matrix $\mathbf{Z} \in \{0, 1\}^{N \times K}$ 를 생성



주어진 데이터 셋에 따라 적응적으로 행렬의 열의 개수를 조정 할 수 있다.

BB 분포와 Network Sparsification

Indian Buffet Process (IBP)

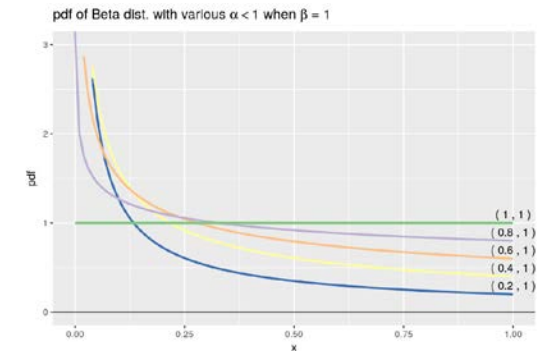
- IBP의 행렬은 BB 분포를 통해 얻어질 수 있다.

$beta(\alpha, \beta)$

$$\begin{bmatrix} z_{1,1} & \cdots & z_{1,k} \\ \vdots & \ddots & \vdots \\ z_{N,1} & \cdots & z_{n,k} \end{bmatrix}$$

$$\pi_k \sim \text{beta}(\alpha/K, 1), \quad z_{n,k} \sim \text{Ber}(\pi_k), \quad K \rightarrow \infty.$$

- BB 분포는 행렬 Z 의 sparsity를 발생**시킨다.
- $K \rightarrow \infty$, $z = 1$ 인 원소의 개수는 $N\alpha$ 로 수렴한다.
- N 은 데이터 개수, α 는 sparsity level을 조정하는 하이퍼파라미터



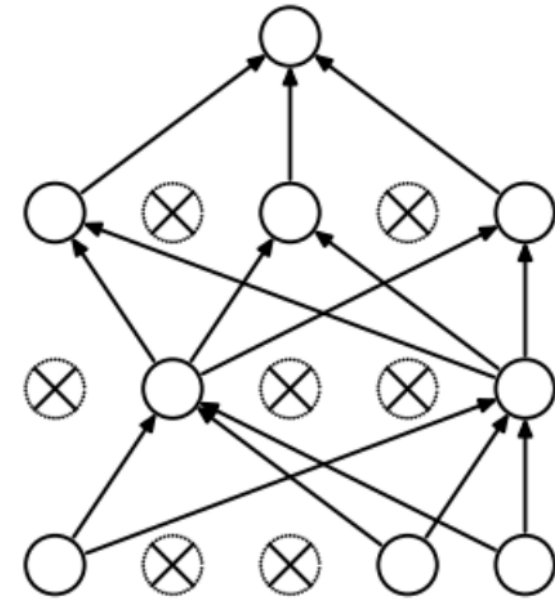
BB 분포와 Network Sparsification

Latent Feature Model

$$\mathbf{d}_n = f(\mathbf{W}\mathbf{z}_n) = f\left(\sum_{k=1}^K \mathbf{z}_{n,k} \mathbf{w}_k\right)$$

dropout mask node
(inputs or Intermediate Feature units)

Structured Pruning




BB 분포를 따르는 IBP를 dropout이 있는 네트워크에 적용시키면, 네트워크를 sparse하게 만들 수 있음.

Variational Inference

사후 확률 (posterior) \propto 사전 확률 (prior) \times 우도 (likelihood)

$$p(\mathbf{W}|\mathcal{D}) \propto p(\mathbf{W}) \times p(\mathcal{D}|\mathbf{W})$$



$$p(\mathbf{y}|\mathbf{f}(\mathbf{x}; \mathbf{W}))$$

데이터를 고려한 후에
자신이 이전에 가지고 있던 사전 확률 분포를 수정하여
사후 확률 분포를 얻음.

Variational Inference

$$p(y) = \int p(y|\theta)p(\theta)d\theta$$

$$\underbrace{p(\mathbf{y}_*|\mathbf{x}_*, \mathcal{D})}_{\text{예측 기대 값}} = \int \underbrace{p(\mathbf{y}_*|\mathbf{f}(\mathbf{x}_*; \mathbf{W}))}_{\text{예측 값 (prediction)}} \underbrace{p(\mathbf{W}|\mathcal{D})}_{\text{사후 확률 분포 (posterior)}} d\mathbf{W}$$

$p(\mathcal{D})$: intractable \rightarrow approximate $p(\mathbf{W}|\mathcal{D})$ as $q(\mathbf{W}; \phi)$

Variational Inference

$$\text{minimize } D_{\text{KL}}[q(\mathbf{W}; \phi) \| p(\mathbf{W} | \mathcal{D})]$$

\approx maximize ELBO (Evidence lower-bound)

$$\begin{aligned} \mathcal{L}(\phi) = & \sum_{n=1}^N \mathbb{E}_q[\log p(\mathbf{y}_n | \mathbf{f}(\mathbf{x}_n; \mathbf{W}))] && \leftarrow \text{expected log-likelihood} \\ & - D_{\text{KL}}[q(\mathbf{W}; \phi) \| p(\mathbf{W})], && \leftarrow \text{regularization} \end{aligned}$$

Variational Inference

$$\text{minimize } D_{\text{KL}}[q(\mathbf{W}; \boldsymbol{\phi}) || p(\mathbf{W} | \mathcal{D})]$$

$$\begin{aligned} KL(q(\mathbf{w} | \boldsymbol{\phi}) || p(\mathbf{w} | \mathcal{D})) &= \int q(\mathbf{w} | \boldsymbol{\phi}) \log \frac{q(\mathbf{w} | \boldsymbol{\phi})}{p(\mathbf{w} | \mathcal{D})} d\mathbf{w} \\ &= \mathbb{E}_{q(\mathbf{w} | \boldsymbol{\phi})} \log \frac{q(\mathbf{w} | \boldsymbol{\phi})}{p(\mathbf{w} | \mathcal{D})} = \mathbb{E}_{q(\mathbf{w} | \boldsymbol{\phi})} \log \frac{q(\mathbf{w} | \boldsymbol{\phi})}{p(\mathcal{D} | \mathbf{w}) p(\mathbf{w})} p(\mathcal{D}) \end{aligned}$$

$$= \underbrace{KL(q(\mathbf{w} | \boldsymbol{\phi}) || q(\mathbf{w})) - \mathbb{E}_{q(\mathbf{w} | \boldsymbol{\phi})} \log p(\mathcal{D} | \mathbf{w})}_{\text{Minimize it !! (negative evidence lower bound)}} + \log p(\mathcal{D})$$

Minimize it !! (negative evidence lower bound)

Variational Inference

$$\text{minimize } D_{\text{KL}}[q(\mathbf{W}; \phi) \| p(\mathbf{W} | \mathcal{D})]$$

\approx maximize ELBO (Evidence lower-bound)

$$\begin{aligned} \mathcal{L}(\phi) = & \sum_{n=1}^N \mathbb{E}_q[\log p(\mathbf{y}_n | \mathbf{f}(\mathbf{x}_n; \mathbf{W}))] && \leftarrow \text{expected log-likelihood} \\ & - D_{\text{KL}}[q(\mathbf{W}; \phi) \| p(\mathbf{W})], && \leftarrow \text{regularization} \end{aligned}$$

Maximum Likelihood와 cross-entropy

$$\sum_{n=1}^N \mathbb{E}_q[\log p(\mathbf{y}_n | \mathbf{f}(\mathbf{x}_n; \mathbf{W}))] \leftarrow \text{expected log-likelihood}$$

$$\mathcal{L}(\phi) = \sum_{n=1}^N \mathbb{E}_q[\log p(\mathbf{y}_n | \mathbf{f}(\mathbf{x}_n; \mathbf{W}))] - \text{D}_{\text{KL}}[q(\mathbf{W}; \phi) \| p(\mathbf{W})],$$

Maximize log-likelihood = Minimize cross entropy + softmax
 Prediction 확률을 최대화

$$- \sum_x P(x) \log Q(x)$$

label Softmax(logits)

In Cell *Define the functions and utils*

```
def cross_entropy(logits, labels):
    return tf.losses.softmax_cross_entropy(logits=logits, onehot_labels=labels)
```

In Cell *Create models*

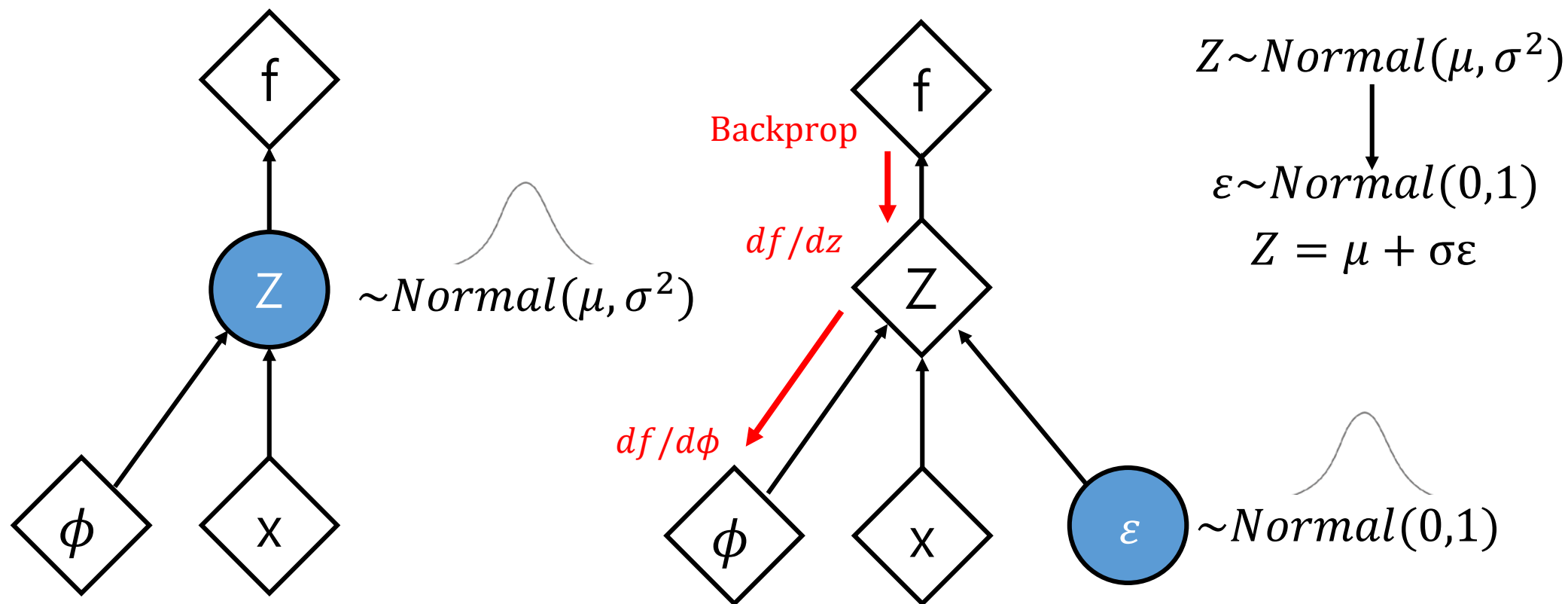
```
net['cent'] = cross_entropy(x, y)
```

In Cell *Define loss*

```
loss = net['cent'] + tf.add_n(net['kl'])/float(N) + net['wd']
```

Reparametrization Trick

미분이 안되는 샘플링 된 노드  를 미분 가능하게 바꿔 Back-propagation을 가능하게 한다.



Variational Inference

Mini-batch 단위로 샘플링

$$\sum_{n=1}^N \nabla_{\phi} \mathbb{E}_q[\log p(\mathbf{y}_n | \mathbf{f}(\mathbf{x}_n; \mathbf{W}))]$$
$$\approx \frac{N}{|B|} \sum_{n \in B} \nabla_{\phi} \mathbb{E}_q[\log p(\mathbf{y}_n | \mathbf{f}(\mathbf{x}_n; \mathbf{W}))]$$

Variational Inference

- + Reparametrization trick
- + Mini-batch sampling

$$\nabla_{\phi} \mathcal{L}(\phi) \xrightarrow{\text{update}} \phi$$

$$q(\mathbf{W}; \phi)$$

BBDrop을 VI을 이용하여 어떻게 최적화하는지 한편 살펴봅시다!

VI를 이용한 최적화

사전 정보(prior):

\mathbf{W} 는 Gaussian 분포를 따른다.

$$\mathbf{W} \sim \mathcal{N}(\mathbf{0}, \lambda \mathbf{I}) \quad \boldsymbol{\pi} \sim \prod_{k=1}^K \text{beta}(\pi_k; \alpha/K, 1),$$

$$\mathbf{z}_n | \boldsymbol{\pi} \sim \prod_{k=1}^K \text{Ber}(z_{n,k}; \pi_k), \quad \widetilde{\mathbf{W}}_n = \mathbf{z}_n \otimes \mathbf{W}.$$

BB분포를 따라 z_n 을 뽑으면, 0이 많이 뽑혀서 네트워크가 sparse해짐.

VI를 이용한 최적화

Goal

주어진 데이터 \mathcal{D} 를 관찰 한 후의 사후 확률 분포

$$p(\mathbf{W}, \mathbf{Z}, \pi | \mathcal{D})$$

를 잘 근사한

Variational 분포 구하기

$$q(\mathbf{W}, \mathbf{Z}, \pi | \mathbf{X})$$

VI를 이용한 최적화

$$q(\mathbf{W}, \mathbf{Z}, \boldsymbol{\pi} | \mathbf{X}) = \delta_{\widehat{\mathbf{W}}}(\mathbf{W}) \underbrace{\prod_{k=1}^K q(\pi_k)}_{\pi_k \sim \text{Beta}(\alpha/K, 1) \text{ 근사}} \underbrace{\prod_{n=1}^N \prod_{k=1}^K q(z_{n,k} | \pi_k)}_{z_{n,k} | \pi_k \sim \text{Ber}(\pi_k) \text{ 근사}}$$

V를 이용한 최적화

$$q(\mathbf{W}, \mathbf{Z}, \boldsymbol{\pi} | \mathbf{X}) = \delta_{\hat{\mathbf{W}}}(\mathbf{W}) \prod_{k=1}^K q(\pi_k) \prod_{n=1}^N \prod_{k=1}^K q(z_{n,k} | \pi_k)$$

$q(\pi_k)$ 를 구하기 위해 **Kumaraswamy 분포**² 사용.

$$q(\pi_k; a_k, b_k) = a_k b_k \pi_k^{a_k-1} (1 - \pi_k^{a_k})^{b_k-1}$$

$$\text{비교) } \text{beta}(\alpha, \beta) = \begin{cases} \frac{\pi^{\alpha-1} (1-\pi)^{\beta-1}}{B(\alpha, \beta)}, & \pi \in [0, 1] \\ 0, & \text{otherwise} \end{cases}$$

Beta 분포와 닮았으면서 매개 변수를 구하기 쉽게 바꿀 수 있음 (reparametrizable).

VI를 이용한 최적화

$$q(\mathbf{W}, \mathbf{Z}, \boldsymbol{\pi} | \mathbf{X}) = \delta_{\hat{\mathbf{W}}}(\mathbf{W}) \prod_{k=1}^K q(\pi_k) \prod_{n=1}^N \prod_{k=1}^K q(z_{n,k} | \pi_k)$$

$\pi_k \sim q(\pi_k)$: Kumaraswamy 분포를 따르는 난수 생성기



하지만, 텐서플로우는 Kumaraswamy 분포를 이용한 난수 생성기를 제공 X



Inverse Transform Sampling³를 이용



Kumaraswamy 분포의 누적 분포 함수(CDF)의 역함수에 $u \sim \text{unif}([0,1])$ 를 인풋으로 제공

$$\pi_k(u; a_k, b_k) = \left(1 - u^{\frac{1}{b_k}}\right)^{\frac{1}{a_k}}, \quad u \sim \text{unif}([0, 1])$$

VI를 이용한 최적화

$$q(\mathbf{W}, \mathbf{Z}, \boldsymbol{\pi} | \mathbf{X}) = \delta_{\hat{\mathbf{W}}}(\mathbf{W}) \prod_{k=1}^K q(\pi_k) \prod_{n=1}^N \prod_{k=1}^K q(z_{n,k} | \pi_k)$$

Kumaraswamy 분포의 누적 분포 함수(CDF)의 역함수에 $u \sim \text{unif}([0,1])$ 를 인풋으로 제공

$$\pi_k(u; a_k, b_k) = (1 - u^{\frac{1}{b_k}})^{\frac{1}{a_k}}, \quad u \sim \text{unif}([0, 1])$$

In Cell *Define the Beta-Bernoulli Dropout*

```
pi = (1 - tf.random_uniform([K])** (1/b))**(1/a)
```

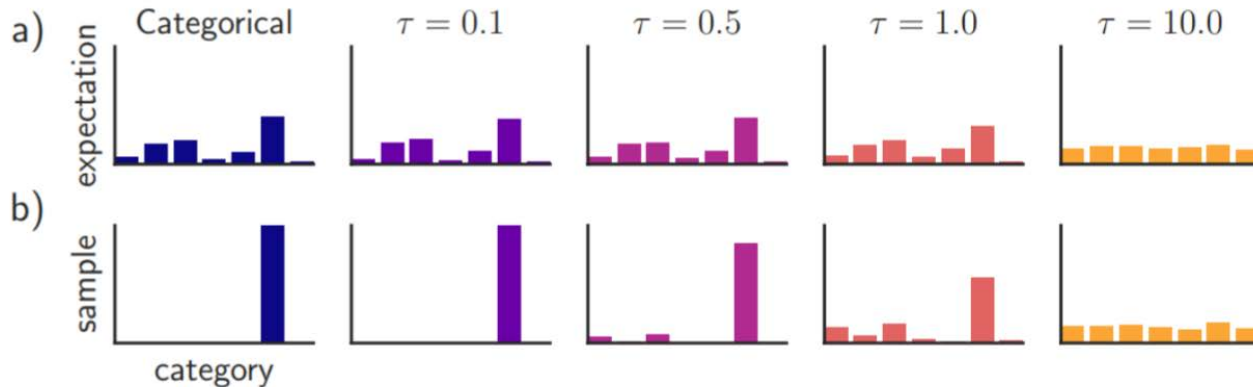
VI를 이용한 최적화

$$q(\mathbf{W}, \mathbf{Z}, \boldsymbol{\pi} | \mathbf{X}) = \delta_{\hat{\mathbf{W}}}(\mathbf{W}) \prod_{k=1}^K q(\pi_k) \prod_{n=1}^N \prod_{k=1}^K q(z_{n,k} | \pi_k)$$

$q(z_{n,k} | \pi_k)$ 는 *continuous relaxation*⁴를 이용하여 *Bernoulli* 분포 근사

$$z_k = \text{sgm} \left(\frac{1}{\tau} \left(\log \frac{\pi_k}{1 - \pi_k} + \log \frac{u}{1 - u} \right) \right) \quad \text{sgm}(x) = \frac{1}{1 + e^{-x}}$$

$u \sim \text{unif}([0, 1])$



$q \rightarrow \text{discrete}, \tau \rightarrow 0$
 $q \rightarrow \text{constant distribution}, \tau \rightarrow \infty$

VI를 이용한 최적화

$$q(\mathbf{W}, \mathbf{Z}, \boldsymbol{\pi} | \mathbf{X}) = \delta_{\hat{\mathbf{W}}}(\mathbf{W}) \prod_{k=1}^K q(\pi_k) \prod_{n=1}^N \prod_{k=1}^K q(z_{n,k} | \pi_k)$$

$q(z_{n,k} | \pi_k)$ 는 *continuous relaxation*⁴를 이용하여 *Bernoulli* 분포 근사

$$z_k = \text{sgm} \left(\frac{1}{\tau} \left(\log \frac{\pi_k}{1 - \pi_k} + \log \frac{u}{1 - u} \right) \right) \quad \text{sgm}(x) = \frac{1}{1 + e^{-x}}$$

$u \sim \text{unif}([0, 1])$

In Cell *Define the Beta-Bernoulli Dropout*

```
z = RelaxedBernoulli(tau, logits=logit(pi)).sample(N)
```

```
return x*z
```

VI를 이용한 최적화

Prior와 variational distribution의 KL-divergence⁵

$$q(\mathbf{W}, \mathbf{Z}, \boldsymbol{\pi} | \mathbf{X}) = \delta_{\hat{\mathbf{W}}}(\mathbf{W}) \prod_{k=1}^K q(\pi_k) \prod_{n=1}^N \prod_{k=1}^K q(z_{n,k} | \pi_k)$$

$$D_{\text{KL}}[q(\mathbf{Z}, \boldsymbol{\pi}) || p(\mathbf{Z}, \boldsymbol{\pi})]$$

$$= \sum_{k=1}^K \left\{ \frac{a_k - \alpha/K}{a_k} \left(-\gamma - \Psi(b_k) - \frac{1}{b_k} \right) + \log \frac{a_k b_k}{\alpha/K} - \frac{b_k - 1}{b_k} \right\},$$

γ : Euler-Mascheroni constant

Ψ : digamma function

Kumaraswamy 분포의 파라미터
 $\pi_k(u; a_k, b_k) = (1 - u^{\frac{1}{b_k}})^{\frac{1}{a_k}}, \quad u \sim \text{unif}([0, 1])$

VI를 이용한 최적화

Prior와 variational distribution의 KL-divergence⁵

$$q(\mathbf{W}, \mathbf{Z}, \boldsymbol{\pi} | \mathbf{X}) = \delta_{\hat{\mathbf{W}}}(\mathbf{W}) \prod_{k=1}^K q(\pi_k) \prod_{n=1}^N \prod_{k=1}^K q(z_{n,k} | \pi_k)$$

$$\begin{aligned} D_{\text{KL}}[q(\mathbf{Z}, \boldsymbol{\pi}) \| p(\mathbf{Z}, \boldsymbol{\pi})] \\ = \sum_{k=1}^K \left\{ \frac{a_k - \alpha/K}{a_k} \left(-\gamma - \Psi(b_k) - \frac{1}{b_k} \right) \right. \\ \left. + \log \frac{a_k b_k}{\alpha/K} - \frac{b_k - 1}{b_k} \right\}, \end{aligned}$$

In Cell *Define the Beta-Bernoulli Dropout*

```
_digamma = digamma_approx
kl = (a-alpha)/a * (-Euler - _digamma(b) - 1/b) W
      + log(a*b) - log(alpha) - (b-1)/b
```

VI를 이용한 최적화

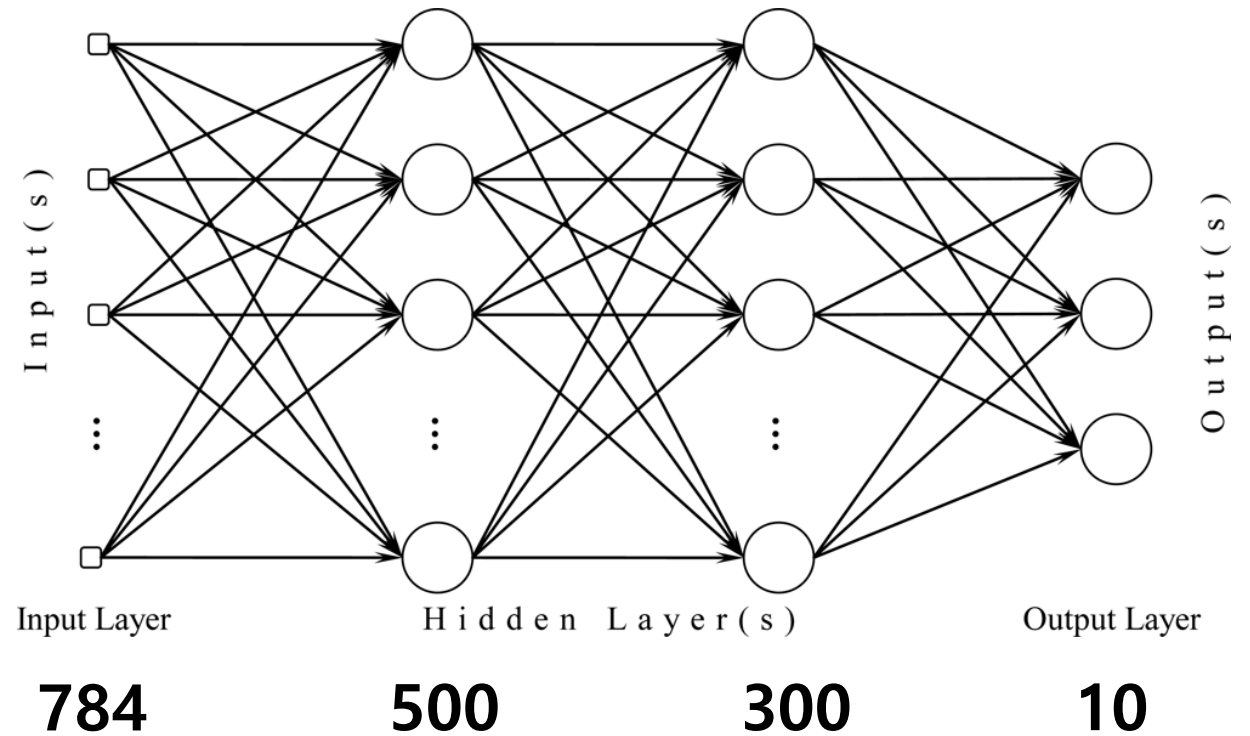
인풋 x_* 에 대한 prediction:

$$\begin{aligned} p(\mathbf{y}_* | \mathbf{x}_*, \mathcal{D}, \mathbf{W}) \\ &= \mathbb{E}_{p(\mathbf{z}_*, \boldsymbol{\pi}, \mathbf{W} | \mathcal{D})} [p(\mathbf{y}_* | \mathbf{f}(\mathbf{x}_*; \mathbf{z}_* \otimes \mathbf{W}))] \\ &\approx \mathbb{E}_{q(\mathbf{z}_*, \boldsymbol{\pi})} [p(\mathbf{y}_* | \mathbf{f}(\mathbf{x}_*; \mathbf{z}_* \otimes \widehat{\mathbf{W}}))], \end{aligned}$$

데이터셋, 네트워크



MNIST



이런

https://github.com/HayeonLee/sparsification_samsung

```
[ ] train()
```

```
↳ Epoch 1 start, learning rate 0.010000  
train: epoch 1, (5.304 secs), cent 0.413326, acc 0.877817  
test: epoch 1, (5.596 secs), cent 0.110209, acc 0.967600  
kl: [4585.8677, 3876.8252, 2495.3503]  
n_active: [523, 469, 297]
```

```
Epoch 2 start, learning rate 0.010000  
train: epoch 2, (4.134 secs), cent 0.164370, acc 0.949483  
test: epoch 2, (4.369 secs), cent 0.083183, acc 0.973700  
kl: [4452.902, 3971.1667, 2566.4827]  
n_active: [511, 467, 297]
```

```
Epoch 3 start, learning rate 0.010000  
train: epoch 3, (4.131 secs), cent 0.125859, acc 0.960067  
test: epoch 3, (4.362 secs), cent 0.067286, acc 0.978200  
kl: [4552.4395, 4077.387, 2617.6377]  
n_active: [503, 465, 296]
```

https://github.com/HayeonLee/sparsification_samsung/blob/1823e44e1231cc56d3cec8c887742c1a5ec2a79c/bbdropout_samsung.ipynb

테스트

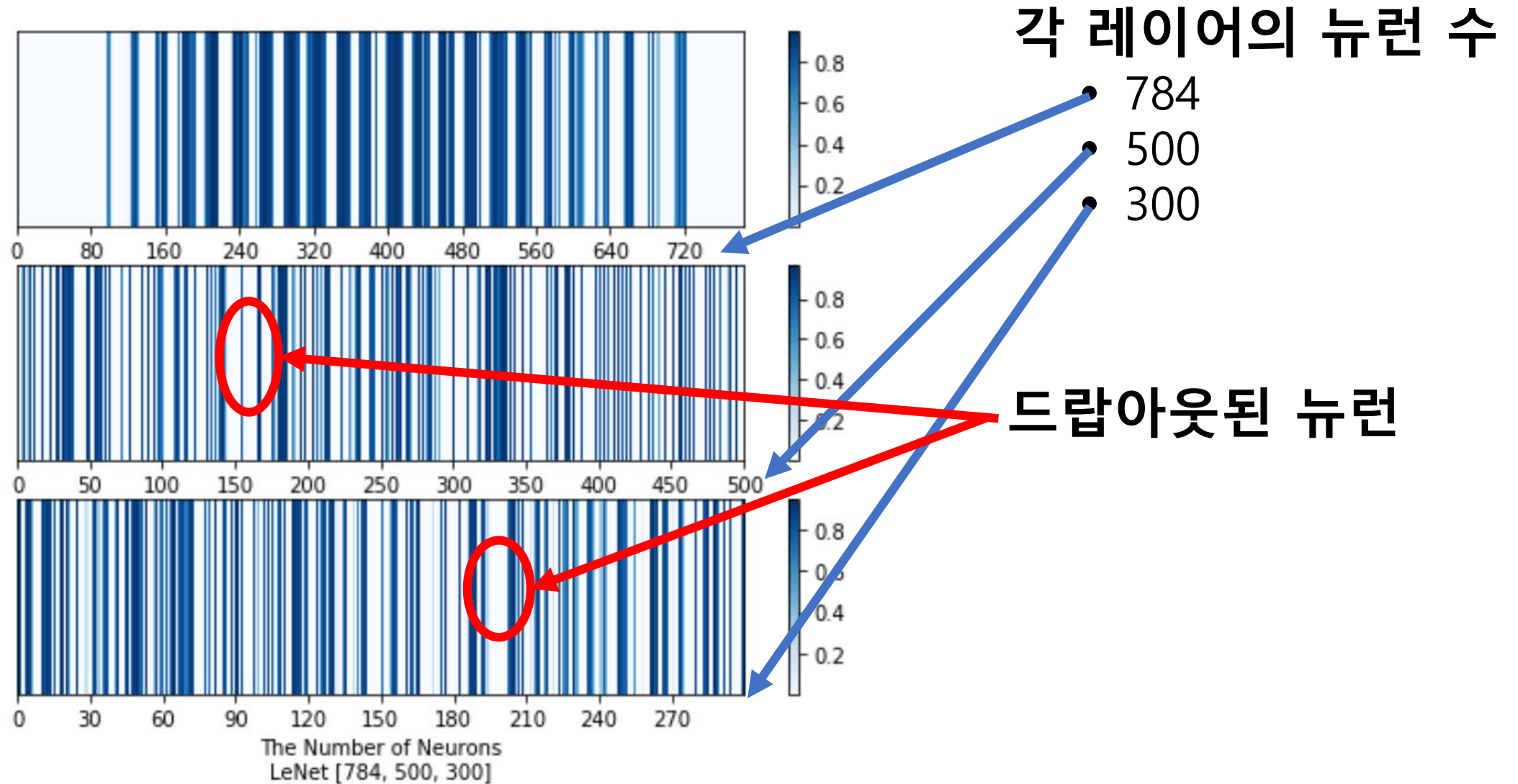
```
[21] def test():
    sess = tf.Session()
    saver = tf.train.Saver(tnet['weights']+tnet['qpi_vars'])
    saver.restore(sess, os.path.join(savedir, 'model'))
    logger = Accumulator('cent', 'acc')
    to_run = [tnet['cent'], tnet['acc']]
    for j in range(n_test_batches):
        bx, by = mnist.test.next_batch(batch_size)
        logger accum(sess.run(to_run, {x:bx, y:by}))
    logger.print_(header='test')

    n_active = sess.run(tnet['n_active'])
    print("The percentage of activated neurons per layer:")
    for na, nl in zip(n_active, [784, 500, 300]):
        print('{} / {} = {:.2f}%'.format(na, nl, float(na)/nl * 100))

test()
```

☞ test: cent 0.041238, acc 0.987600
The percentage of activated neurons per layer
358/784 = 45.66%
209/500 = 41.80%
152/300 = 50.67%

시각화



BBdrop 실습

Setting (1차과정)

접속

MobaXterm 프로그램 실행

```
ssh -X -p 2222 com01@143.248.159.**
```

1번 PC: com01@143.248.159.27

2번 PC: com02@143.248.159.28

4번 PC: com04@143.248.159.3

5번 PC~26번 PC: comN@143.248.159.N

Conda 실행

```
source ~/.bashrc  
conda activate bbdrop  
jupyter notebook --port 9000
```

폴더 이동

```
cd BBDrop
```


Setting (2차과정)

접속

MobaXterm 프로그램 실행

```
ssh -X -p 2222 com0102@143.248.159.\*\*
```

1번 PC: com0102@143.248.159.27

2번 PC: com0202@143.248.159.28

4번 PC: com0402@143.248.159.3

5번 PC~26번 PC: comN02@143.248.159.N

Conda 실행

```
source ~/.bashrc  
conda activate bbdrop  
jupyter notebook --port 9999
```

폴더 이동

```
cd BBDrop
```