Implementation Tutorial: Bayesian Deep Learning

Hae Beom Lee

MLAI, KAIST

24. Sep. 2019.

Overview

This tutorial is scheduled as follows:

1. **Variational Autoendoer** – 1.5 hour.

2. **MC-dropout** – 1.5 hour.

3. **Concrete-dropout** – (if time allows)

Environments

Prerequisites

- Linux or macOS
- Python >= 2.7
- Tensorflow >= 1.3

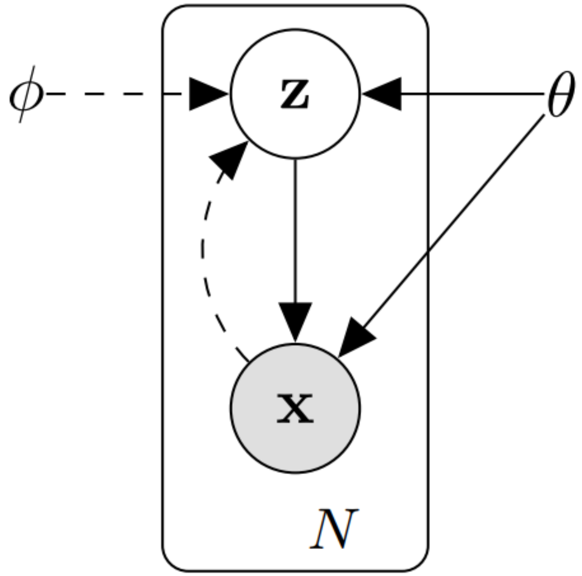
Github Repositories

- VAE: https://github.com/haebeom-lee/vae
- MC-dropout: https://github.com/haebeom-lee/mc-dropout
- Concrete-dropout: https://github.com/haebeom-lee/concrete-dropout

Part I:
Variational Autoencoder
(*VAE*)

Generative process

- For unsupervised learning.
- Graphical model and generative process is as follows.

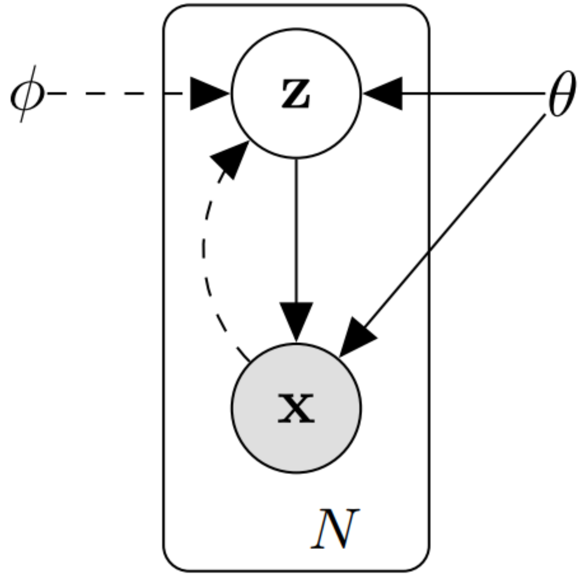


$$
\begin{aligned}
p(X;\theta) &= \prod_{x \in X} p(x;\theta) \\
&= \prod_{x \in X} \int p(x, z; \theta) dz \\
&= \prod_{x \in X} \int p(z) p(x|z;\theta) dz
\end{aligned}
$$

Evidence lower bound

Get the log evidence lower bound → maximization objective.

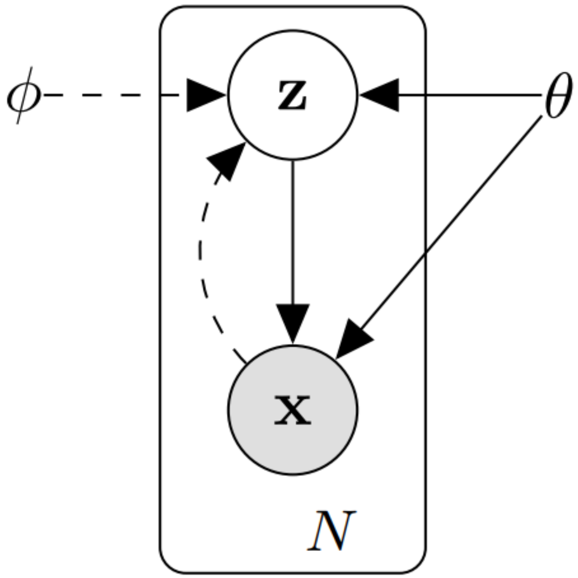
For each datapoint x, we have



$$
\begin{aligned}
\log p(x;\theta) &= \log \int p(x|z;\theta)p(z)dz \\
&= \log \int q(z|x;\phi)\frac{p(x|z;\theta)p(z)}{q(z|x;\phi)}dz \\
&\geq \int q(z|x;\phi)\log\frac{p(x|z;\theta)p(z)}{q(z|x;\phi)}dz \\
&= E_{q(z|x;\phi)}[\log p(x|z;\theta)] - KL[q(z|x;\phi)\|p(z)]
\end{aligned}
$$

Evidence lower bound

Get the log evidence lower bound → maximization objective.



$$\log p(X;\theta)$$

$$\geq \sum_{x \in X} \boxed{E_{q(z|x;\phi)}[\log p(x|z;\theta)]}$$

autoencoder

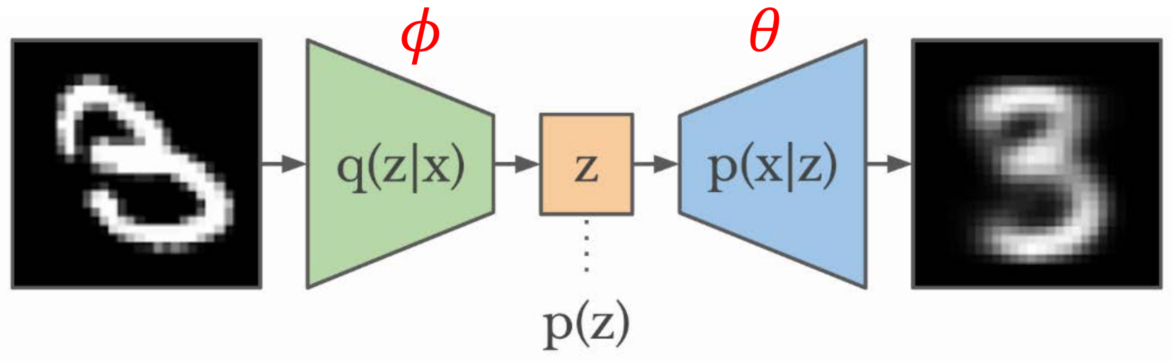
$$\boxed{-KL[q(z|x;\phi)\|p(z)]}$$

KL-divergence (regularizer)

$$= \mathcal{L}(\theta,\phi)$$

Structure of VAE

$$\log p(X;\theta) \geq \sum_{x\in X} E_{q(z|x;\phi)}[\log p(x|z;\theta)] - KL[q(z|x;\phi)\|p(z)]$$



model.py

```python
def encoder(x, zdim, name='encoder', reuse=None):
    x = dense(x, 500, activation=relu, name=name+'/dense1', reuse=reuse)
    x = dense(x, 500, activation=relu, name=name+'/dense2', reuse=reuse)
    mu = dense(x, zdim, name=name+'/mu', reuse=reuse)
    sigma = dense(x, zdim, activation=softplus, name=name+'/sigma', reuse=reuse)
    return mu, sigma
```

```python
def decoder(x, name='decoder', reuse=None):
    x = dense(x, 500, activation=relu, name=name+'/dense1', reuse=reuse)
    x = dense(x, 500, activation=relu, name=name+'/dense2', reuse=reuse)
    x = dense(x, 784, activation=sigmoid, name=name+'/output', reuse=reuse)
    return x
```

Modeling the log likelihood

$$\log p(X;\theta) \geq \sum_{x \in X} E_{q(z|x;\phi)}[\log p(x|z;\theta)] - KL[q(z|x;\phi)\|p(z)]$$

- Bernoulli likelihood for MNIST data (binary)

$$\begin{aligned}
\log p(x|z;\theta) &= \log \prod_d \mathrm{Ber}(x_d; \hat{x}_d(z;\theta)) \\
&= \sum_d \log \hat{x}_d(z;\theta)^{x_d}(1-\hat{x}_d(z;\theta))^{1-x_d} \\
&= \sum_d x_d \log \hat{x}_d(z;\theta) + (1-x_d)\log(1-\hat{x}_d(z;\theta))
\end{aligned}$$

model.py line 21

```
log_likelihood = tf.reduce_sum(x*log(x_hat) + (1-x)*log(1-x_hat), 1)
```

(But actually this is wrong...)

KL divergence

$$\log p(X;\theta)$$
$$\geq \sum_{x \in X} E_{q(z|x;\phi)}[\log p(x|z;\theta)]$$
$$-KL[q(z|x;\phi) \| p(z)]$$

- between gaussian prior and posterior has a closed form.

$$
\begin{aligned}
&KL[q(z|x;\phi)\|p(z)] \\
&= KL\left[\mathcal{N}(z;\mu(x;\phi),\operatorname{diag}(\sigma(x;\phi)^2))\|\mathcal{N}(z;0,I)\right] \\
&= \sum_j KL\left[\mathcal{N}(z_j;\mu_j,\sigma_j^2)\|\mathcal{N}(z_j;0,1)\right] \\
&= \frac{1}{2}\sum_j(\mu_j^2+\sigma_j^2-\log\sigma_j^2-1)
\end{aligned}
$$

model.py line 22

```
kl = 0.5 * tf.reduce_sum(mu**2 + sigma**2 - log(sigma**2) - 1, 1)
```

Monte-Carlo approximation

$$\log p(X;\theta) \geq \sum_{x\in X} E_{q(z|x;\phi)}[\log p(x|z;\theta)] - KL[q(z|x;\phi)\|p(z)]$$

- Approximate the expectation with MC sampling

$$\mathcal{L}(\theta,\phi) = \sum_{x\in X} E_{q(z|x;\phi)}[\log p(x|z;\theta)] - KL[q(z|x;\phi)\|p(z)]$$

$$\approx \sum_{x\in X} \frac{1}{S}\sum_{s=1}^{S} \log p(x|z^{(s)};\theta) - KL[q(z|x;\phi)\|p(z)], \quad z^{(s)} \sim q(z|x;\phi)$$

By setting S=1, we have

$$\tilde{\mathcal{L}}(\theta,\phi) = \sum_{x\in X} \log p(x|\tilde{z};\theta) - KL[q(z|x;\phi)\|p(z)], \quad \tilde{z} \sim q(z|x;\phi)$$

Reparameterization trick

$$\log p(X;\theta) \geq \sum_{x \in X} E_{q(z|x;\phi)}[\log p(x|z;\theta)] - KL[q(z|x;\phi)\|p(z)]$$

- Reparameterize the likelihood function

$$p(x|\tilde{z};\theta), \quad \tilde{z} \sim q(z|x;\phi)$$

$$p(x|\mu + \sigma \odot \epsilon;\theta), \quad \epsilon \sim \mathcal{N}(0, I)$$

With MC approximation, we have

$$\tilde{\mathcal{L}}(\theta, \phi) = \sum_{x \in X} \log p(x|\mu + \sigma \odot \epsilon;\theta) - KL[q(z|x;\phi)\|p(z)], \quad \epsilon \sim \mathcal{N}(0, I)$$

model.py line 16

```
def autoencoder(x, zdim, training, name='autoencoder', reuse=None):
    mu, sigma = encoder(x, zdim, reuse=reuse)
    z = Normal(mu, sigma).sample() if training else mu
    x_hat = decoder(z, reuse=reuse)
```

Minibatch SGD

- mini-batch size = M

$$\tilde{\mathcal{L}}(\theta, \phi) = \sum_{x \in X} \Big\{ \log p(x|\mu + \sigma \odot \epsilon; \theta) - KL[q(z|x;\phi) \| p(z)] \Big\}, \quad \epsilon \sim \mathcal{N}(0, I)$$

$$\frac{1}{N}\tilde{\mathcal{L}}(\theta, \phi) = \frac{1}{N}\sum_{x \in X} \Big\{ \log p(x|\mu + \sigma \odot \epsilon; \theta) - KL[q(z|x;\phi) \| p(z)] \Big\}, \quad \epsilon \sim \mathcal{N}(0, I)$$

$$\simeq \frac{1}{M}\sum_{x \in B} \Big\{ \log p(x|\mu + \sigma \odot \epsilon; \theta) - KL[q(z|x;\phi) \| p(z)] \Big\}, \quad \epsilon \sim \mathcal{N}(0, I)$$

model.py line 23

```
elbo = tf.reduce_mean(log_likelihood - kl)
```

$$\frac{1}{M}\sum_{x\in B}\left\{\log p(x|\mu+\sigma\odot\epsilon;\theta) - KL[q(z|x;\phi)\|p(z)]\right\}, \quad \epsilon\sim\mathcal{N}(0,I)$$

$$\frac{1}{M}\sum_{x\in B}\left\{\sum_d x_d\log\hat{x}_d + (1-x_d)\log(1-\hat{x}_d) - \frac{1}{2}\sum_j(\mu_j^2+\sigma_j^2-\log\sigma^2-1)\right\}, \quad \epsilon\sim\mathcal{N}(0,I)$$

```
def encoder(x, zdim, name='encoder', reuse=None):
    x = dense(x, 500, activation=relu, name=name+'/dense1', reuse=reuse)
    x = dense(x, 500, activation=relu, name=name+'/dense2', reuse=reuse)
    mu = dense(x, zdim, name=name+'/mu', reuse=reuse)
    sigma = dense(x, zdim, activation=softplus, name=name+'/sigma', reuse=reuse)
    return mu, sigma

def decoder(x, name='decoder', reuse=None):
    x = dense(x, 500, activation=relu, name=name+'/dense1', reuse=reuse)
    x = dense(x, 500, activation=relu, name=name+'/dense2', reuse=reuse)
    x = dense(x, 784, activation=sigmoid, name=name+'/output', reuse=reuse)
    return x

def autoencoder(x, zdim, training, name='autoencoder', reuse=None):
    mu, sigma = encoder(x, zdim, reuse=reuse)
    z = Normal(mu, sigma).sample() if training else mu
    x_hat = decoder(z, reuse=reuse)

    log_likelihood = tf.reduce_sum(x*log(x_hat) + (1-x)*log(1-x_hat), 1)
    kl = 0.5 * tf.reduce_sum(mu**2 + sigma**2 - log(sigma**2) - 1, 1)
    elbo = tf.reduce_mean(log_likelihood - kl)
```

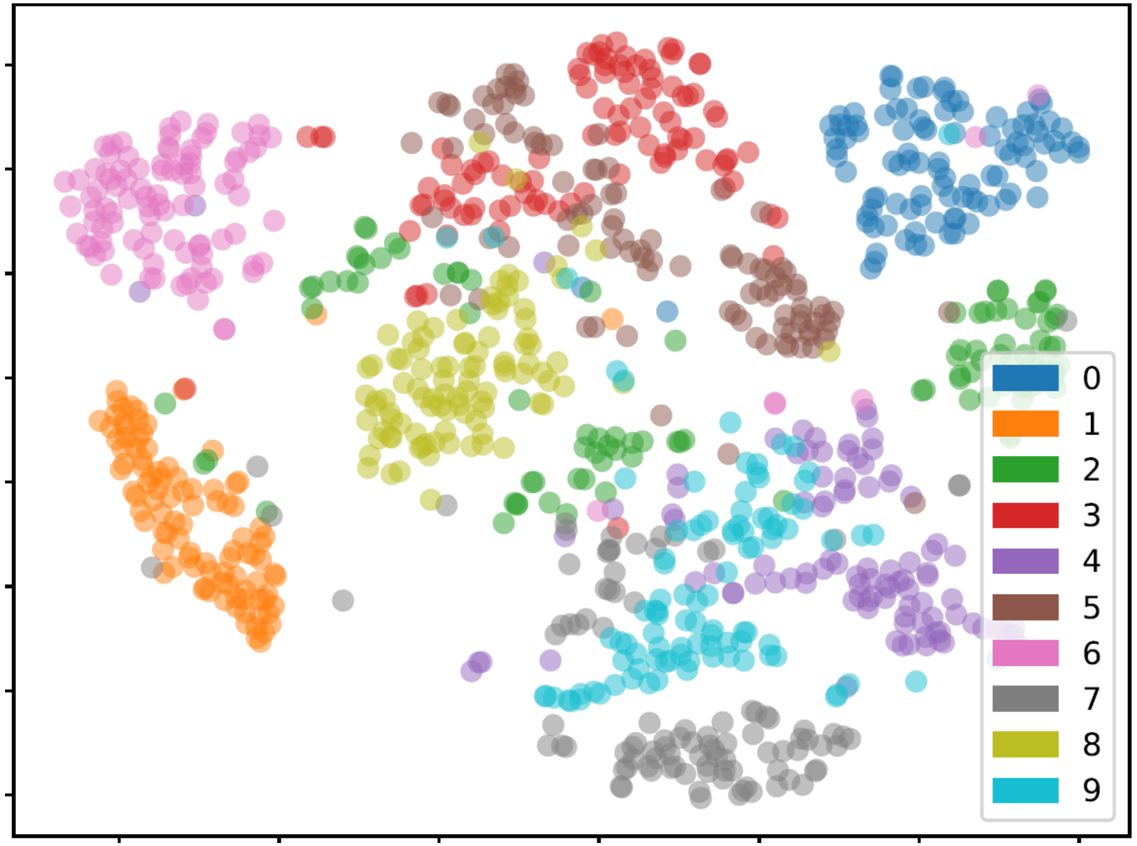
Training

```
hblee@ai1:/st1/hblee/vae/results/run$ vi train.log
```

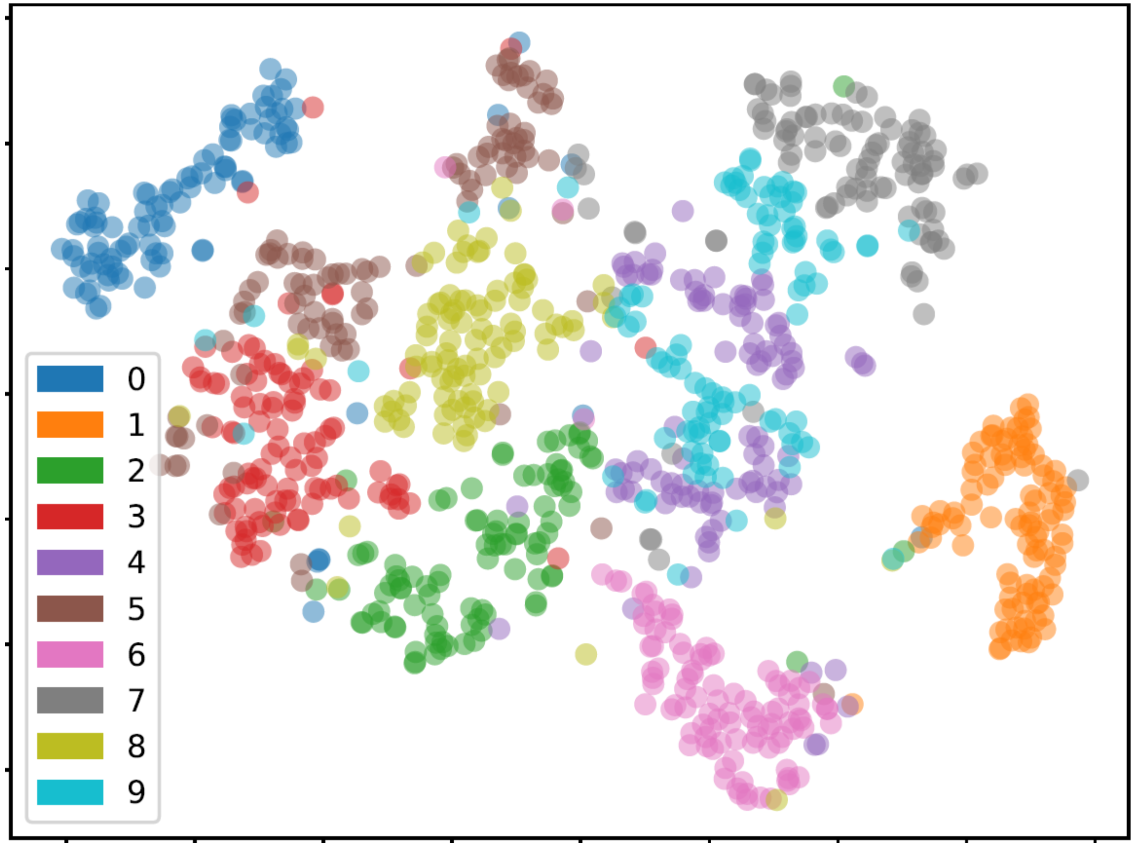
```
Epoch 1 start, learning rate 0.001000
train: epoch 1, (0.535 secs), elbo -400.160945

Epoch 2 start, learning rate 0.001000
train: epoch 2, (0.039 secs), elbo -218.883324

Epoch 3 start, learning rate 0.001000
train: epoch 3, (0.039 secs), elbo -206.140288

Epoch 4 start, learning rate 0.001000
train: epoch 4, (0.040 secs), elbo -202.247569

Epoch 5 start, learning rate 0.001000
train: epoch 5, (0.039 secs), elbo -200.175874

Epoch 6 start, learning rate 0.001000
train: epoch 6, (0.039 secs), elbo -197.606540
```

Learned q(z|x)

- visualize with T-SNE



train



test

Reconstruction

- with train instances

original

reconstructed

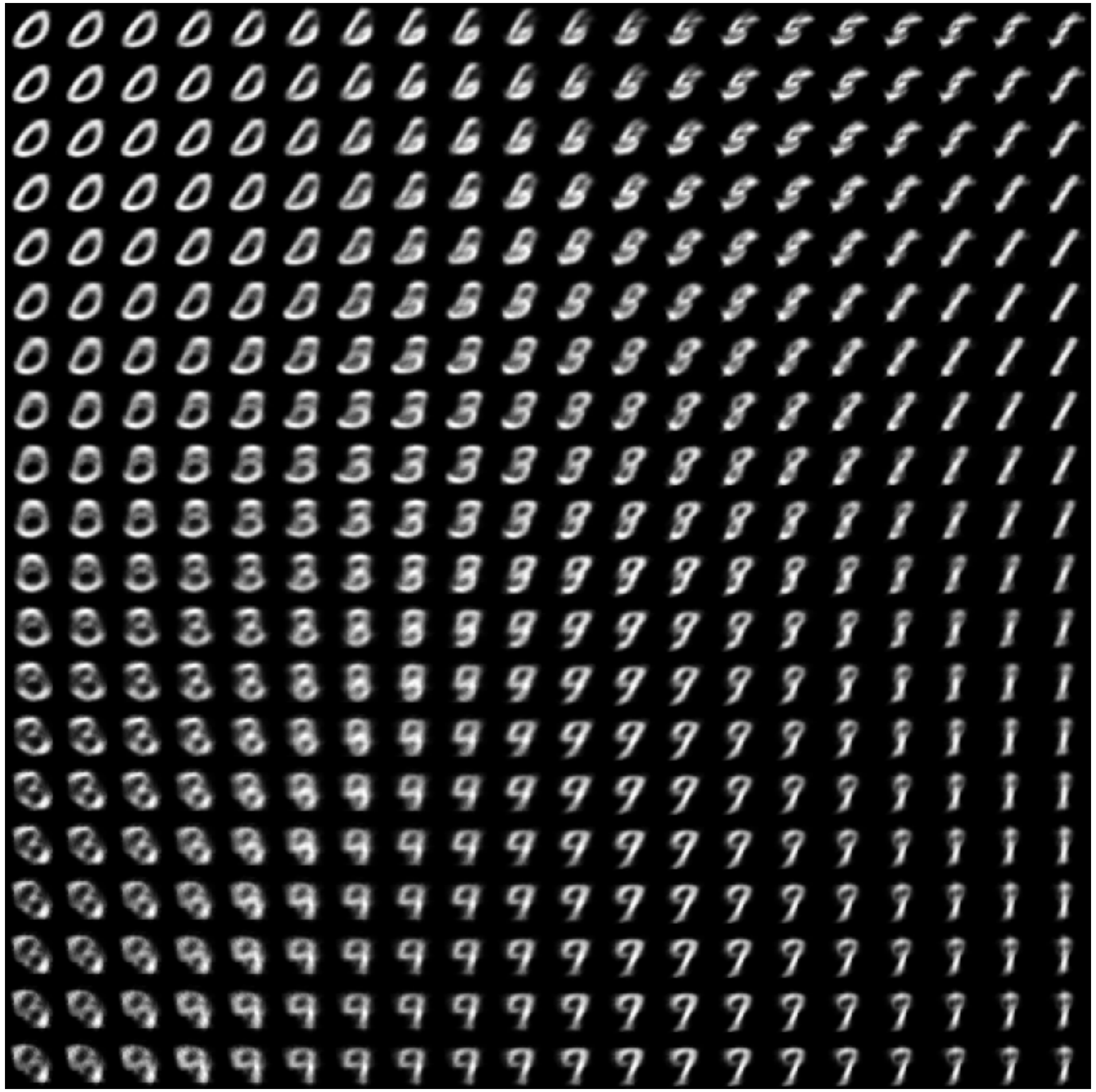
Reconstruction

- with test instances

original

reconstructed

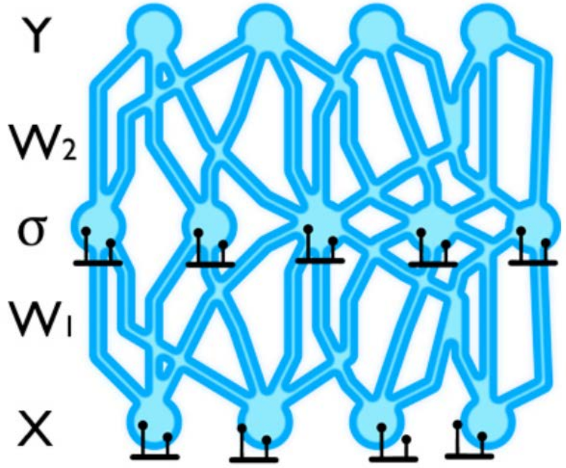
Learned manifold



Part II:
MC-dropout

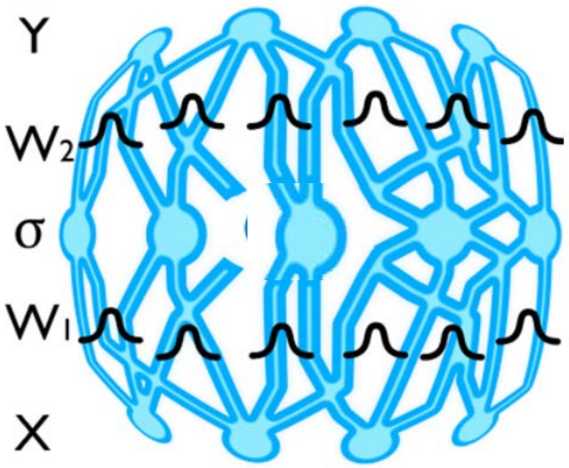
Dropout NN ≈ Bayesian NN

- For supervised learning.
- Dropout NN can be interpreted as a variational approximation of Bayesian NN.



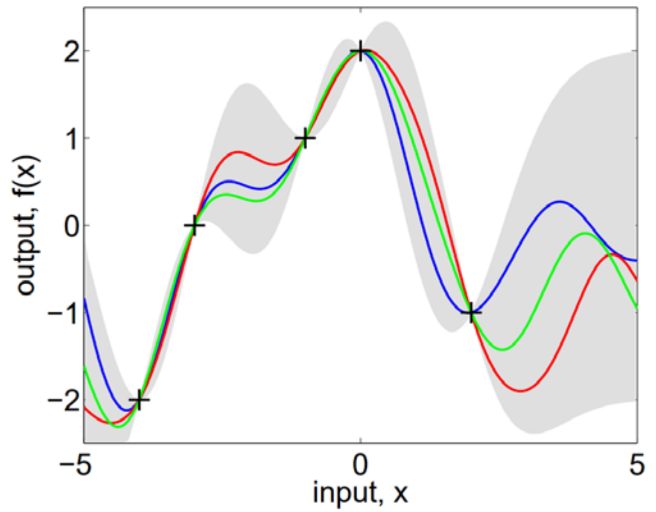
Dropout NN

≈



Bayesian Neural Network (many layers)

≈

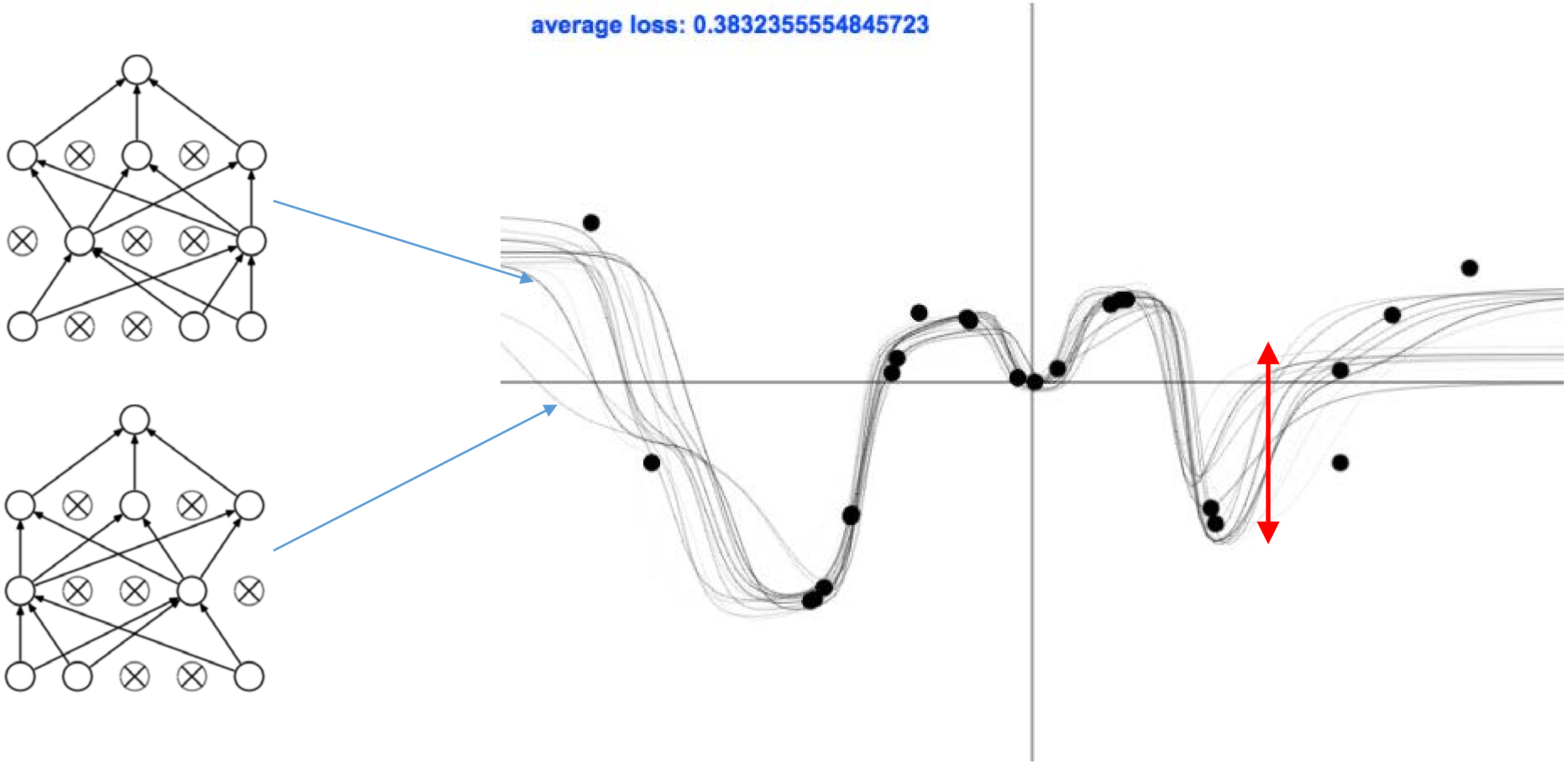


(deep) Gaussian Process

Contribution of [Gal15]

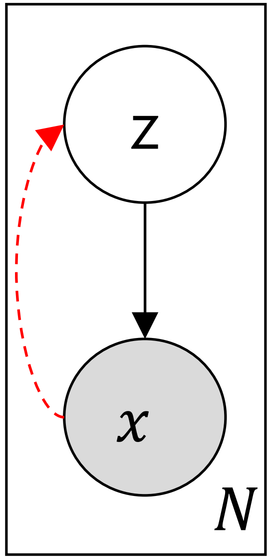
Contribution of [Gal15]

Dropout function draw



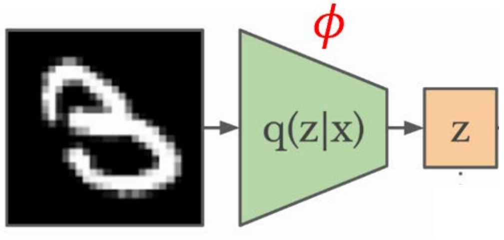
Each solid black line is a function drawn from **dropout sampling**.

Graphical model

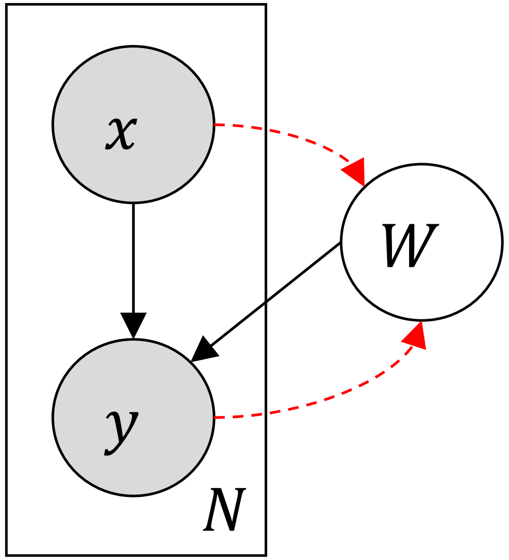


$q(z|x;\phi)$

→ Inference network given x.

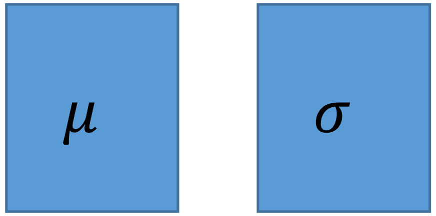


VAE



$q(W;\phi)$

→ Globally updated parameter w.r.t. all D= {(x,y)}.

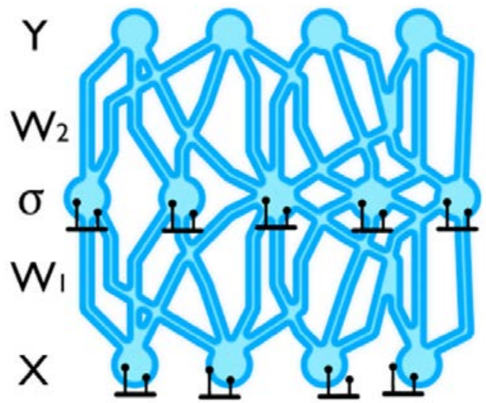


$$q(W;\phi) = \mu + \sigma \odot \epsilon$$

(if q is gaussian.)

Bayesian NN

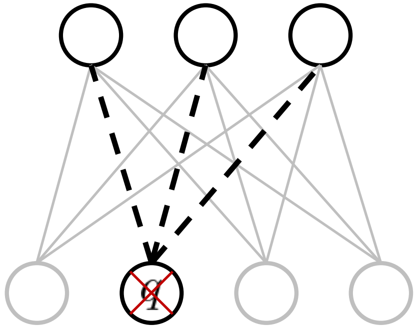
Dropout variational distribution



$$q(W_1, W_2) = q(W_1)q(W_2) \qquad q(W) = \prod_{q=1}^{Q} q(w_q) \qquad q(w_q) = \bar{p}\,0 + (1-\bar{p})m_q$$

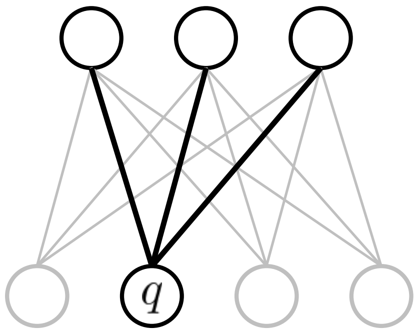
$q(w_q) \quad = \quad \bar{p} \ \times$

$(Q = 4)$



0

$+ \ (1-\bar{p}) \ \times$

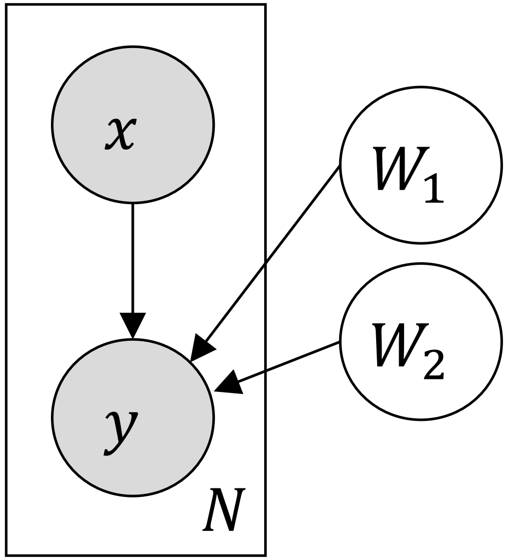


m_q

model.py

```
def network(x, y, keep_prob, training, name='network', reuse=None):
    x = dense(x, 500, activation=relu, name=name+'/dense1', reuse=reuse)
    x = dropout(x, keep_prob)
    x = dense(x, 500, activation=relu, name=name+'/dense2', reuse=reuse)
    x = dropout(x, keep_prob)
    x = dense(x, 10, name=name+'/logit', reuse=reuse)
```

Deriving ELBO



$$\log p(Y|X)$$

$$= \log \iint p(Y|X, W_1, W_2)p(W_1)p(W_2)$$

$$= \log \iint q(W_1)q(W_2)p(Y|X, W_1, W_2)\frac{p(W_1)p(W_2)}{q(W_1)q(W_2)}$$

$$\geq \iint q(W_1)q(W_2) \log \frac{p(Y|X, W_1, W_2)p(W_1)p(W_2)}{q(W_1)q(W_2)}$$

$$= \iint q(W_1)q(W_2) \log p(Y|X, W_1, W_2) + \int q(W_1) \log \frac{p(W_1)}{q(W_1)} + \int q(W_2) \log \frac{p(W_2)}{q(W_2)}$$

Evaluating the KL term

$$ELBO = \underbrace{E_{q(W_1)q(W_2)}[\log p(Y|X, W_1, W_2)]}_{\text{data likelihood}} - \underbrace{\sum_{d=1}^{2} KL[q(W_d)\|p(W_d)]}_{\text{regularization}}$$

$$KL[q(W)\|p(W)] = \sum_{q=1}^{Q} KL[q(w_q)\|p(w_q)] \approx \frac{(1-\bar{p})l^2}{2}\|M\|^2 + C$$

$$= \lambda\|M\|^2 + C$$

See https://arxiv.org/pdf/1506.02157.pdf for the derivation.

model.py line 13

```
net['wd'] = weight_decay(1e-4, var_list=net['weights'])
```

Evaluating the likelihood term

$$ELBO = \underbrace{E_{q(W_1)q(W_2)}[\log p(Y|X, W_1, W_2)]}_{\text{data likelihood}} - \underbrace{\sum_{d=1}^{2} KL[q(W_d)\|p(W_d)]}_{\text{regularization}}$$

$$\sum_{i=1}^{N} E_{q(W_1)q(W_2)}\left[\log p(y_i|x_i, W_1, W_2)\right]$$

$$= \sum_{i=1}^{N} E_{q(\epsilon_1)q(\epsilon_2)}\left[\log p(y_i|x_i, \epsilon_1, \epsilon_2; M_1, M_2)\right] \qquad \epsilon_1, \epsilon_2 \sim Ber(\bar{p})$$

dropout masks

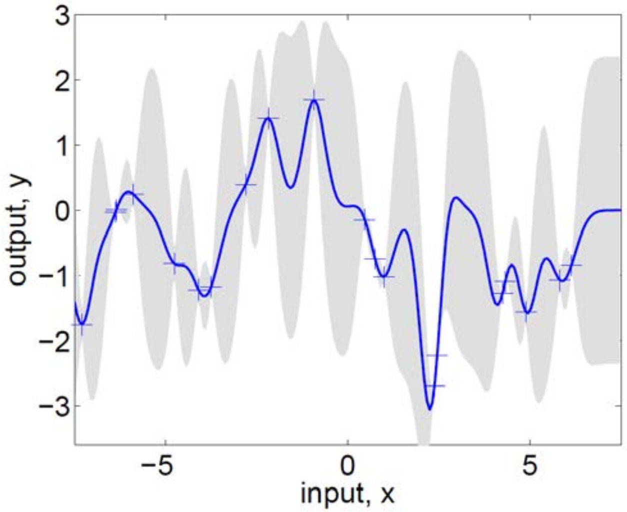
$$\approx \sum_{i=1}^{N} \log p(y_i|x_i, \tilde{\epsilon}_1, \tilde{\epsilon}_2; M_1, M_2)$$

The role of weight decay

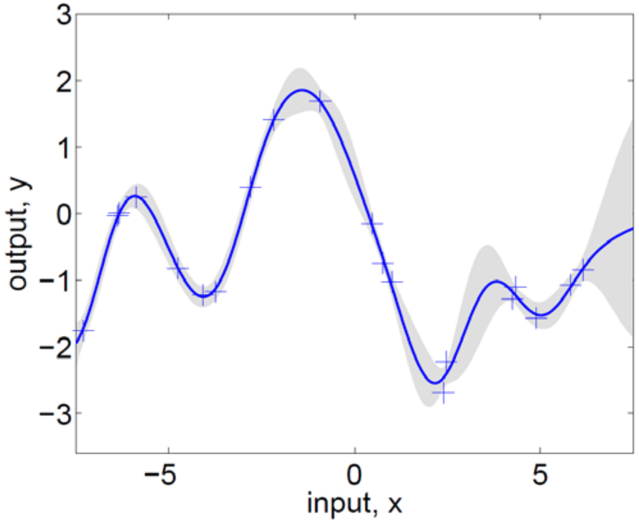
$$p(w_q) \sim \mathcal{N}(0, l^{-2} I)$$

$$J = \frac{1}{N} \sum_{i=1}^{N} - \log p(y_i | x_i, \tilde{\epsilon}_1, \tilde{\epsilon}_2; M_1, M_2) + \frac{(1-\bar{p}) l^2}{2N} \left(\|M_1\|_F^2 + \|M_2\|_F^2 \right)$$

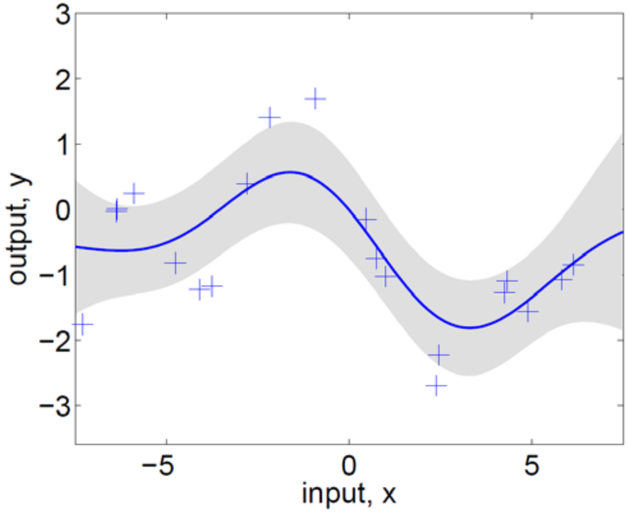
$= \lambda$: weight decay parameter
(tunable with cross validation)



too small λ
(too weak prior)



good λ



too large λ
(too strong prior)

Training

$$q(y^*|x^*) = \iint p(y^*|x^*, W_1, W_2) q(W_1) q(W_2) \approx \frac{1}{S}\sum_{s=1}^{S} p(y^*|x^*, W_1^{(s)}, W_2^{(s)}), \quad W_1^{(s)}, W_2^{(s)} \sim q$$

./results/run/train.log

```
Epoch 1 start, learning rate 0.001000
train: epoch 1, (0.230 secs), cent 1.979612, wd 0.054315, acc 0.336000

Epoch 2 start, learning rate 0.001000
train: epoch 2, (0.053 secs), cent 1.094547, wd 0.051379, acc 0.686000

Epoch 3 start, learning rate 0.001000
train: epoch 3, (0.048 secs), cent 0.715212, wd 0.049918, acc 0.760000

Epoch 4 start, learning rate 0.001000
train: epoch 4, (0.046 secs), cent 0.523885, wd 0.049172, acc 0.831000

Epoch 5 start, learning rate 0.001000
train: epoch 5, (0.043 secs), cent 0.433346, wd 0.048822, acc 0.862000

Epoch 6 start, learning rate 0.001000
train: epoch 6, (0.040 secs), cent 0.366523, wd 0.048710, acc 0.891000

Epoch 7 start, learning rate 0.001000
train: epoch 7, (0.042 secs), cent 0.276434, wd 0.048749, acc 0.926000

Epoch 8 start, learning rate 0.001000
train: epoch 8, (0.050 secs), cent 0.225711, wd 0.048887, acc 0.936000

Epoch 9 start, learning rate 0.001000
train: epoch 9, (0.046 secs), cent 0.204144, wd 0.049059, acc 0.945000

Epoch 10 start, learning rate 0.001000
train: epoch 10, (0.045 secs), cent 0.164487, wd 0.049250, acc 0.950000
```

Testing

1. Naïve approximation

$$q(y^*|x^*) = \iint p(y^*|x^*, W_1, W_2) q(W_1) q(W_2) \approx p(y^*|x^*, E_{q(W_1)}[W_1], E_{q(W_2)}[W_2])$$

run.py
line 105

```
# naive approximation (expectation inside)
logger = Accumulator('cent', 'wd', 'acc')
logger.accum(sess.run([tnet['cent'], tnet['wd'], tnet['acc']],
        {x:xte, y:yte, keep_prob:1.0}))
```

2. MC sampling

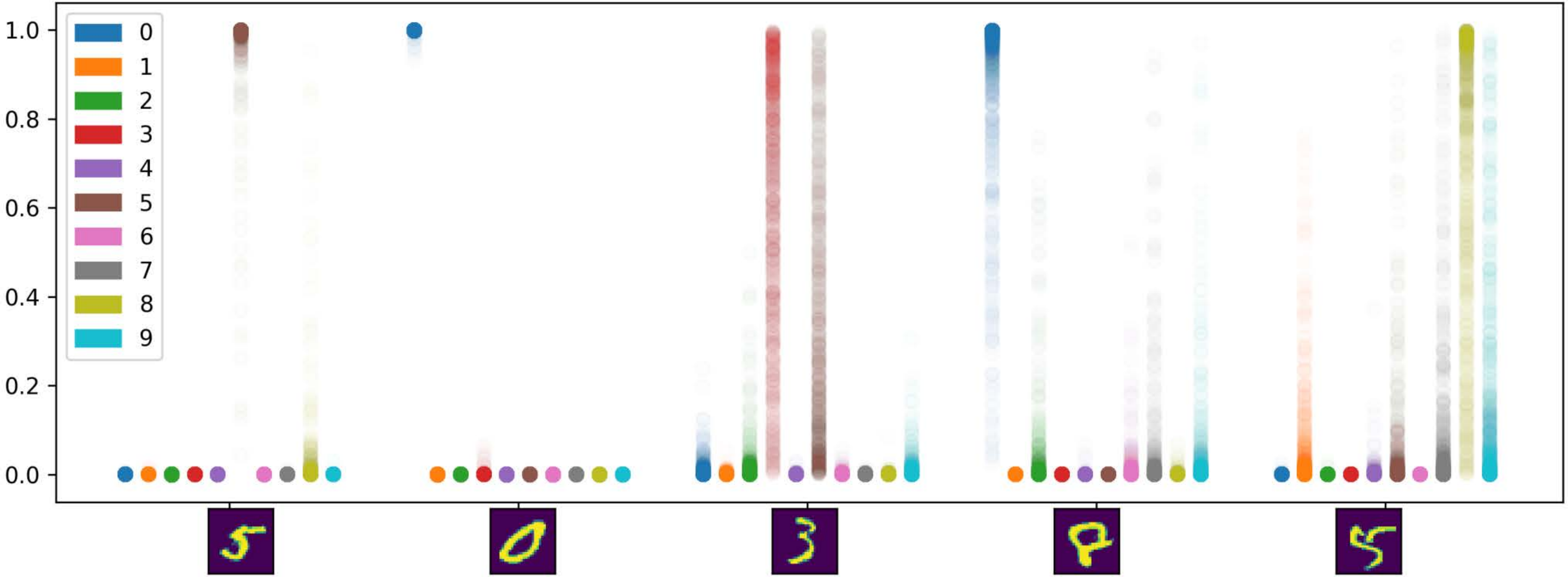
$$q(y^*|x^*) = \iint p(y^*|x^*, W_1, W_2) q(W_1) q(W_2) \approx \frac{1}{S} \sum_{s=1}^{S} p(y^*|x^*, W_1^{(s)}, W_2^{(s)}), \quad W_1^{(s)}, W_2^{(s)} \sim q$$

run.py
line 113

```
# MC sampling (S=100)
p = []
for s in range(100):
    p.append(sess.run(net['pred'],
        {x:xte, y:yte, keep_prob:args.keep_prob}))
p = np.stack(p, axis=0)
p = np.mean(p, axis=0)
```

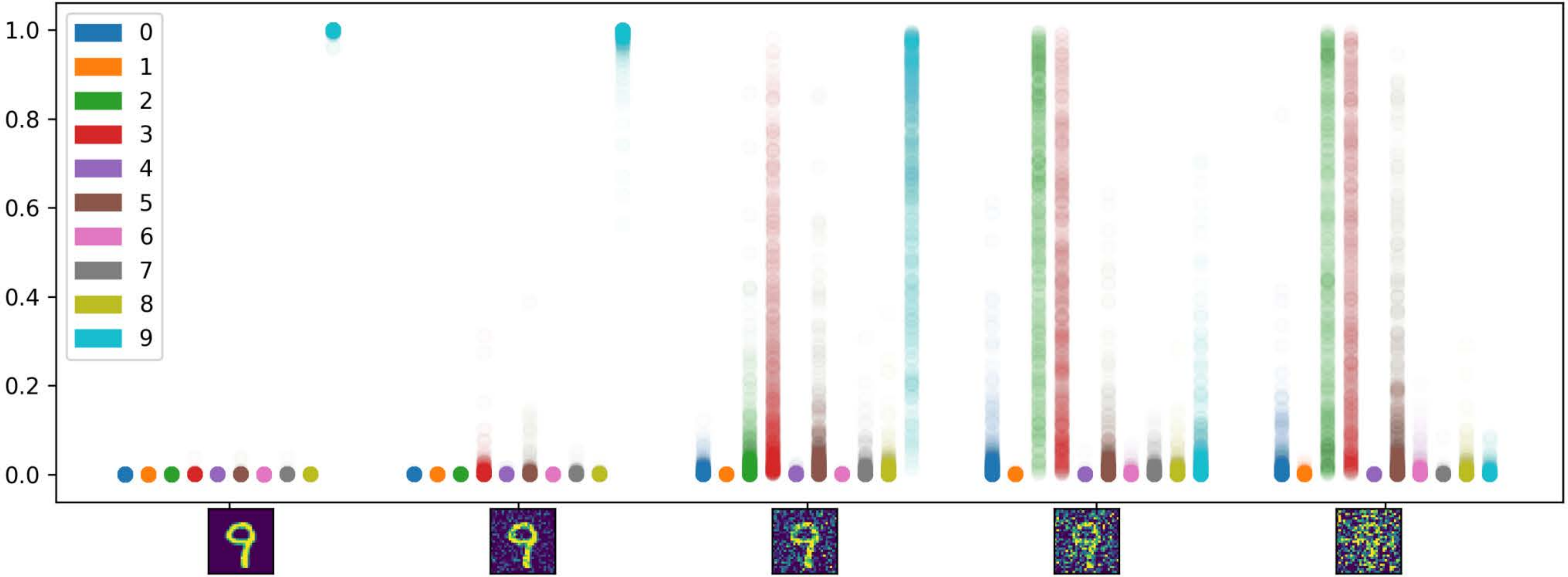
Uncertainty evaluation

Real test examples (MC sampling with S=1000)



Uncertainty evaluation

Synthetic test examples (S=1000)



Part III:
Concrete-dropout

Idea

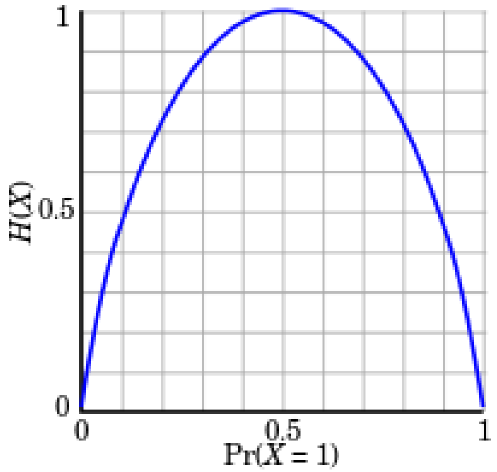
The optimal dropout probability $\bar{p}$ is found with grid-search. Can we learn $\bar{p}$?

For each neuron, we have

$$KL[q(w)\|p(w)] = \int q(w)(\log q(w) - \log p(w))\mathrm{d}w$$

$$\approx \frac{(1-\bar{p})l^2}{2}\|m\|^2 - (-\bar{p}\log\bar{p} - (1-\bar{p})\log(1-\bar{p})) + C$$

$$= \frac{(1-\bar{p})l^2}{2}\|m\|^2 - \mathcal{H}(\bar{p}) + C$$



[Gal et al.] Concrete Dropout, NIPS2017

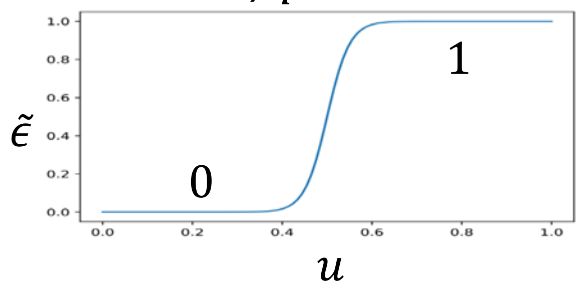
Concrete Relaxation

Reparameterization trick for Bernoulli variables.

$$E_{q(W)}\left[\sum_{i=1}^{N} \log p(y_i|x_i, W)\right] \approx \sum_{i=1}^{N} \log p(y_i|x_i, \tilde{\epsilon}; M), \qquad \tilde{\epsilon} \sim Ber(\bar{p})$$

$$\tilde{\epsilon} = \mathrm{sigm}\left(\frac{1}{t}\left(\log\frac{\bar{p}}{1-\bar{p}} + \log\frac{u}{1-u}\right)\right) \qquad u \sim \mathcal{U}(0,1)$$

ex) $\bar{p} = 0.5$



layer.py

```
RelaxedBernoulli = tf.contrib.distributions.RelaxedBernoulli
```

```
masks = RelaxedBernoulli(0.01, tf.expand_dims(keep_prob, 0) \
        * tf.ones_like(x)).sample()
```

[Gal et al.] Concrete Dropout, NIPS2017

Final objective

For mini-batch SGD.

$$\frac{1}{N}\mathcal{L}(\phi) = \frac{1}{N}\sum_{i=1}^{N} -\log p(y_i|x_i, \tilde{u}; \bar{p}, M) + \frac{1}{N}\sum_q \frac{(1-\bar{p}_q)l^2}{2}\|m_q\|^2 - \frac{1}{N}\sum_q \mathcal{H}(\bar{p}_q)$$

$$\approx \frac{1}{B}\sum_{j=1}^{B} -\log p(y_j|x_j, \tilde{u}; \bar{p}, M) + \frac{1}{N}\sum_q \frac{(1-\bar{p}_q)l^2}{2}\|m_q\|^2 - \frac{1}{N}\sum_q \mathcal{H}(\bar{p}_q)$$

run.py line 56

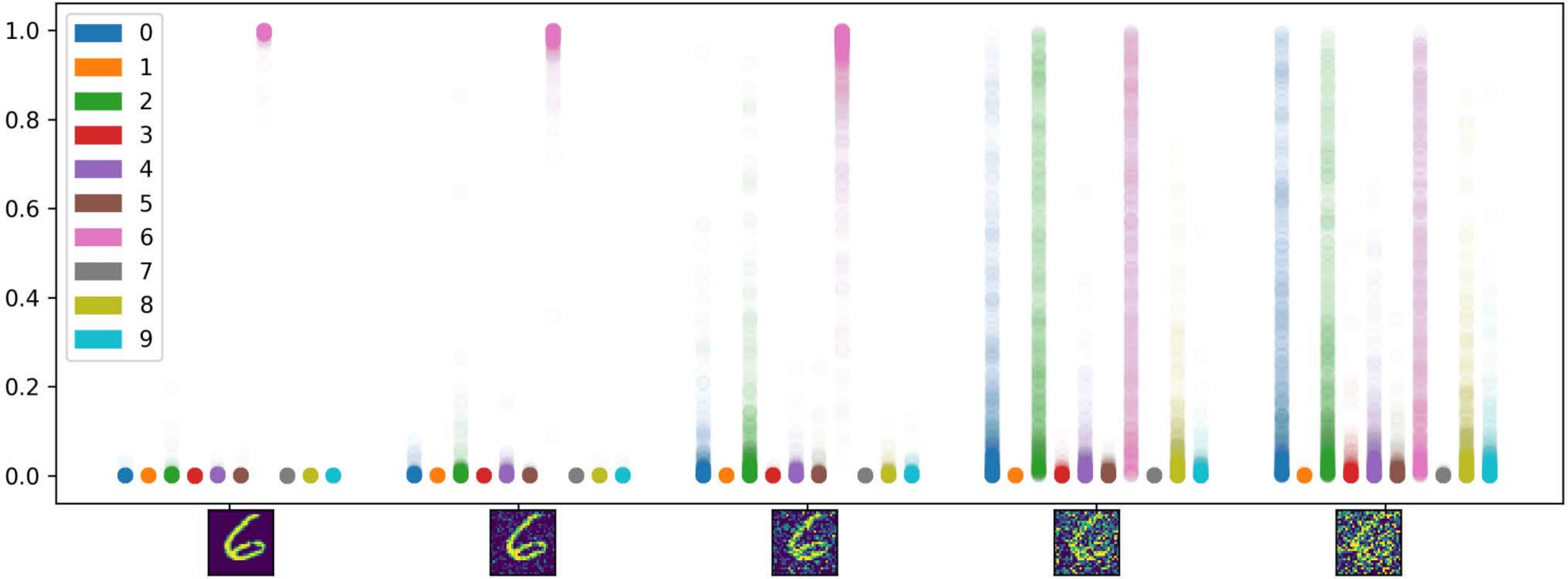
```
loss = net['cent'] + (net['wd'] - net['ent'])/args.N # negative ELBO
```

Thus, the effect of regularization terms is relative to the size of dataset.

[Gal et al.] Concrete Dropout, NIPS2017

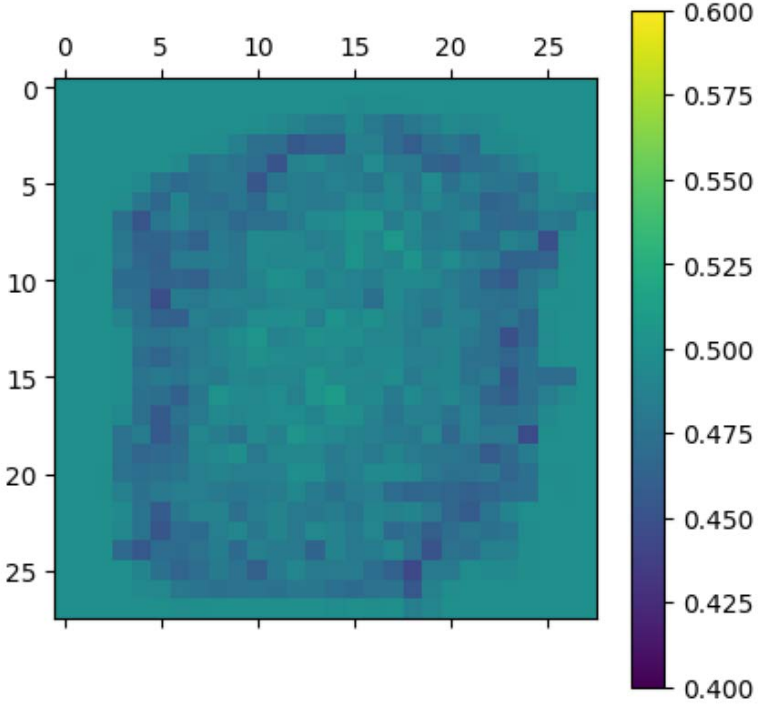
Evaluate uncertainties

Synthetic test examples

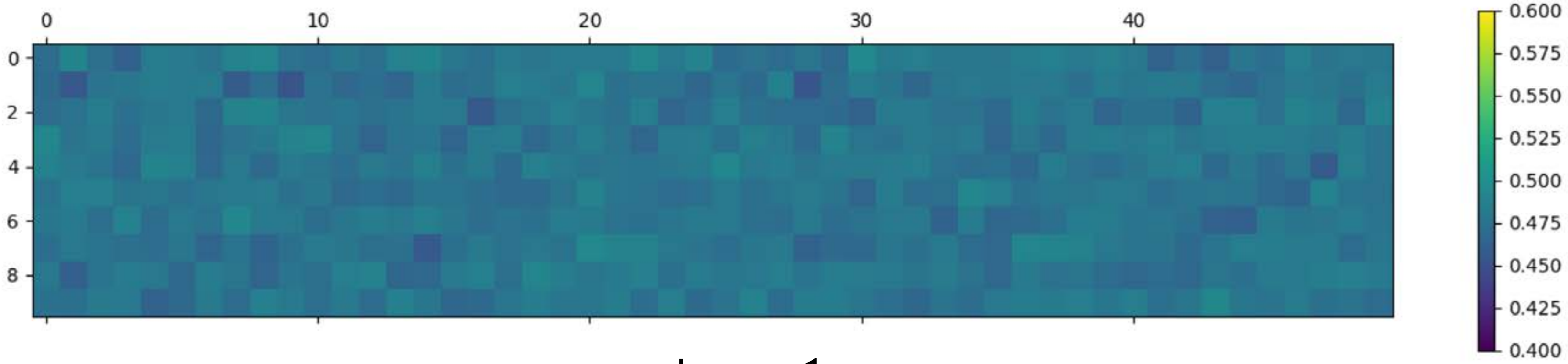


Learned keep probabilities

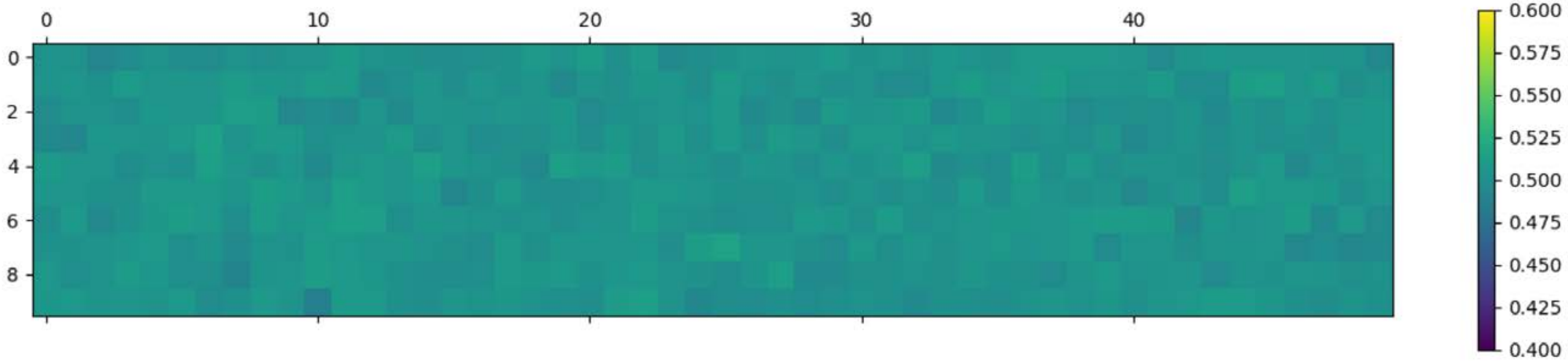
Number of train instances = 1000



input



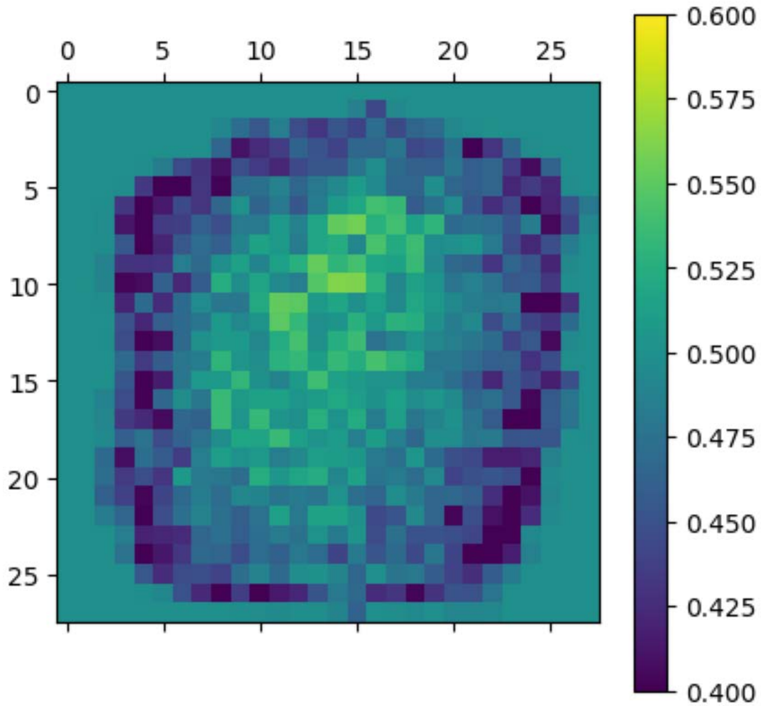
Layer 1



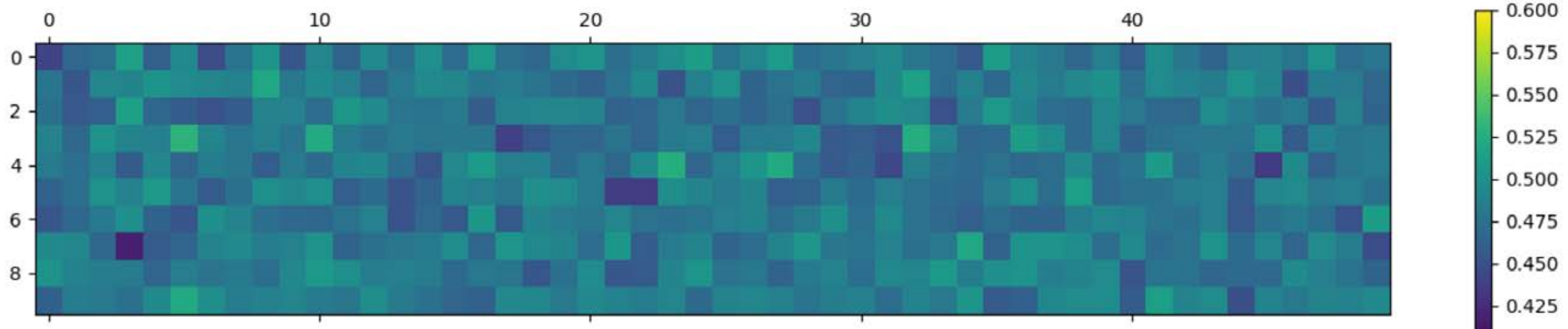
Layer 2

Learned keep probabilities

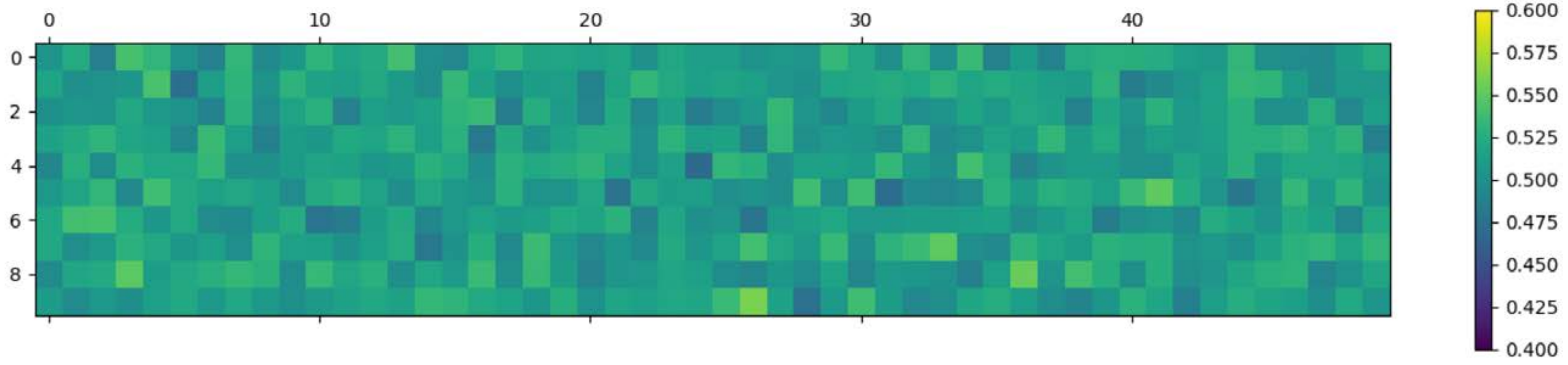
Number of train instances = 10000



input



Layer 1



Layer 2

Test accuracy

	Naïve approx	MC sampling
Base	89.37 ± 0.32	-
MC dropout	90.07 ± 0.44	90.17 ± 0.23
Concrete dropout	$\mathbf{90.23 \pm 0.21}$	$\mathbf{90.20 \pm 0.32}$
Variational dropout	88.77 ± 0.53	89.60 ± 0.56