EE488 Special Topics in EE
<Deep Learning and AlphaGo>

Sae-Young Chung

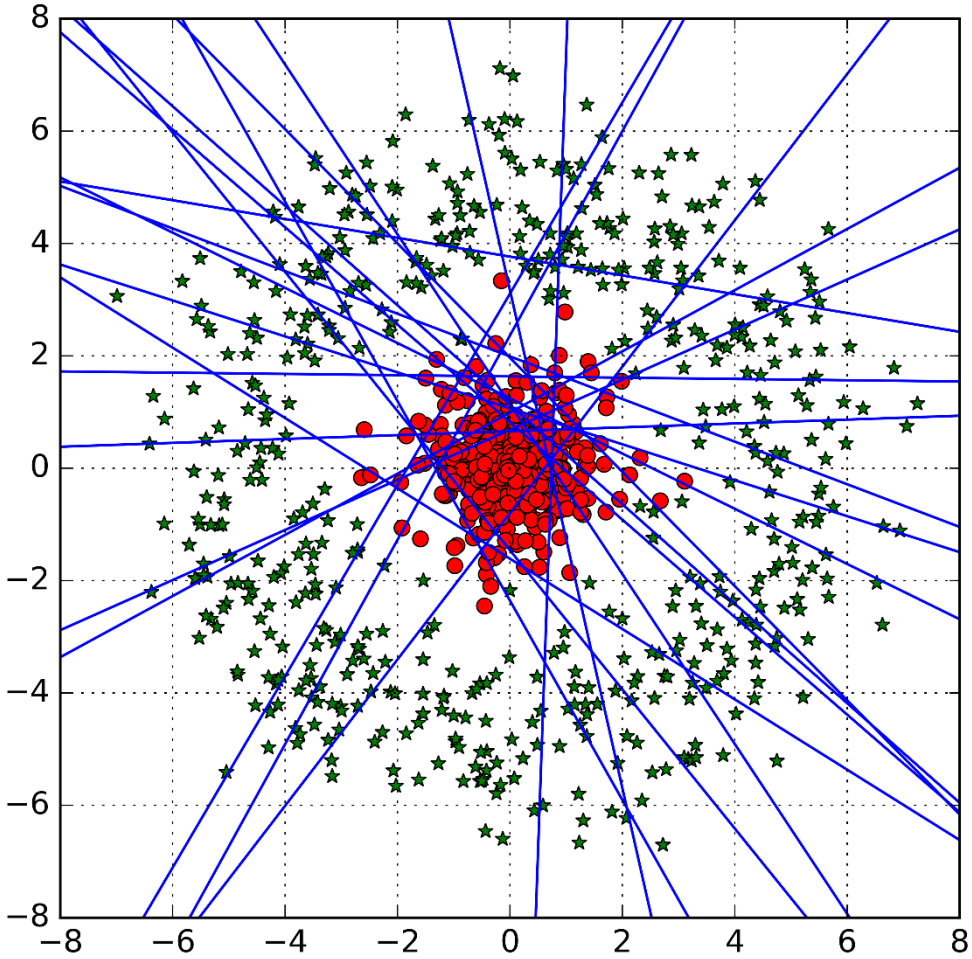
Project #1

November 8, 2017

KAIST

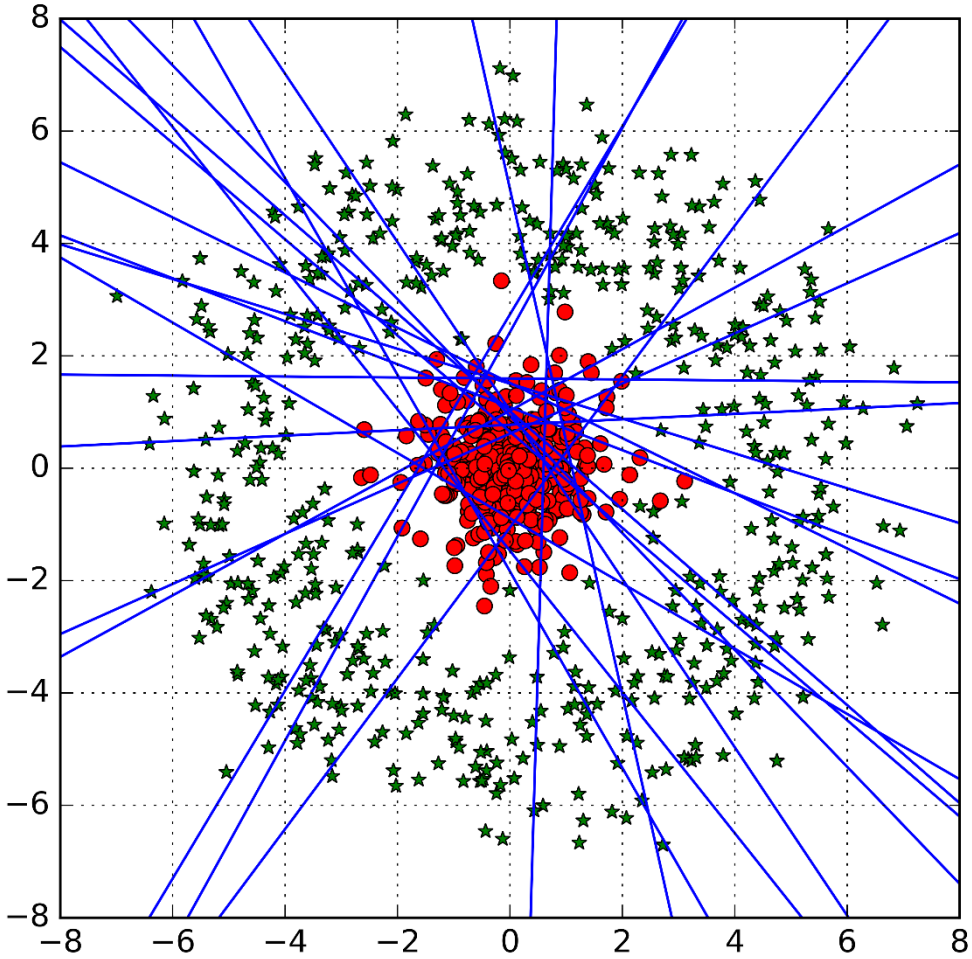
Task 1

Before training



Task 1

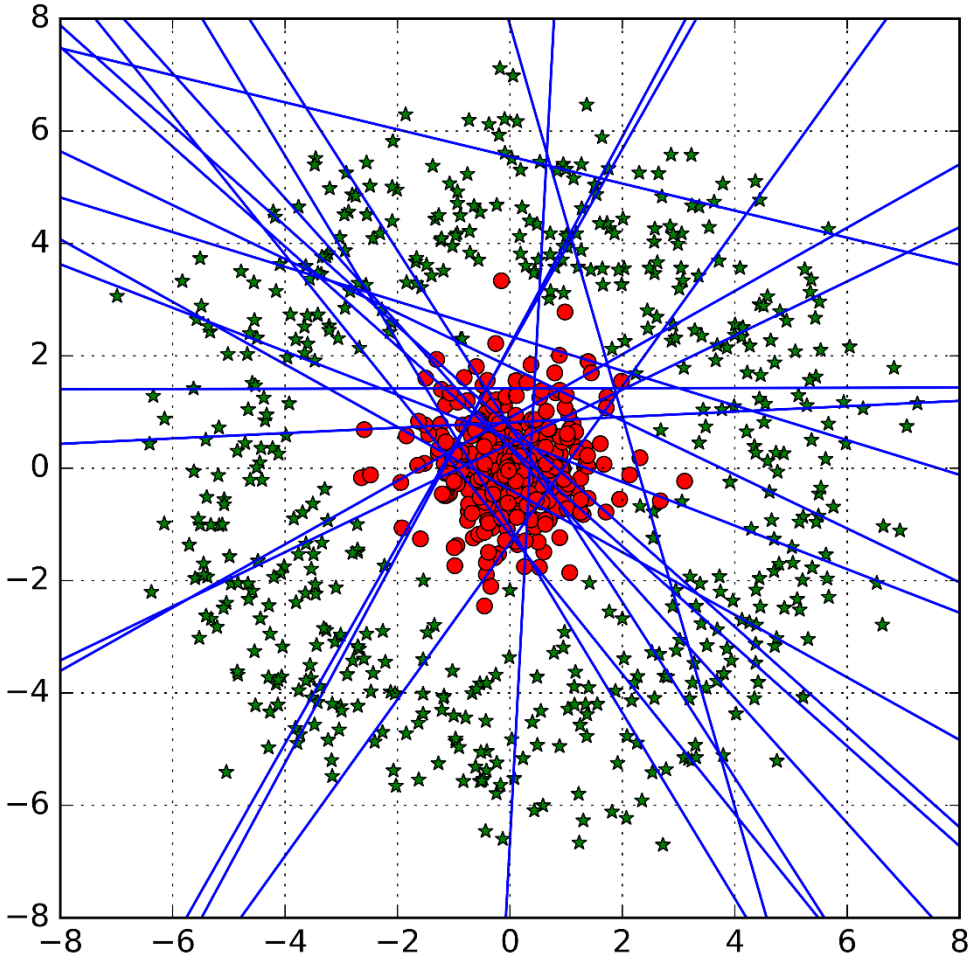
After 10 SGD steps



KAIST

Task 1

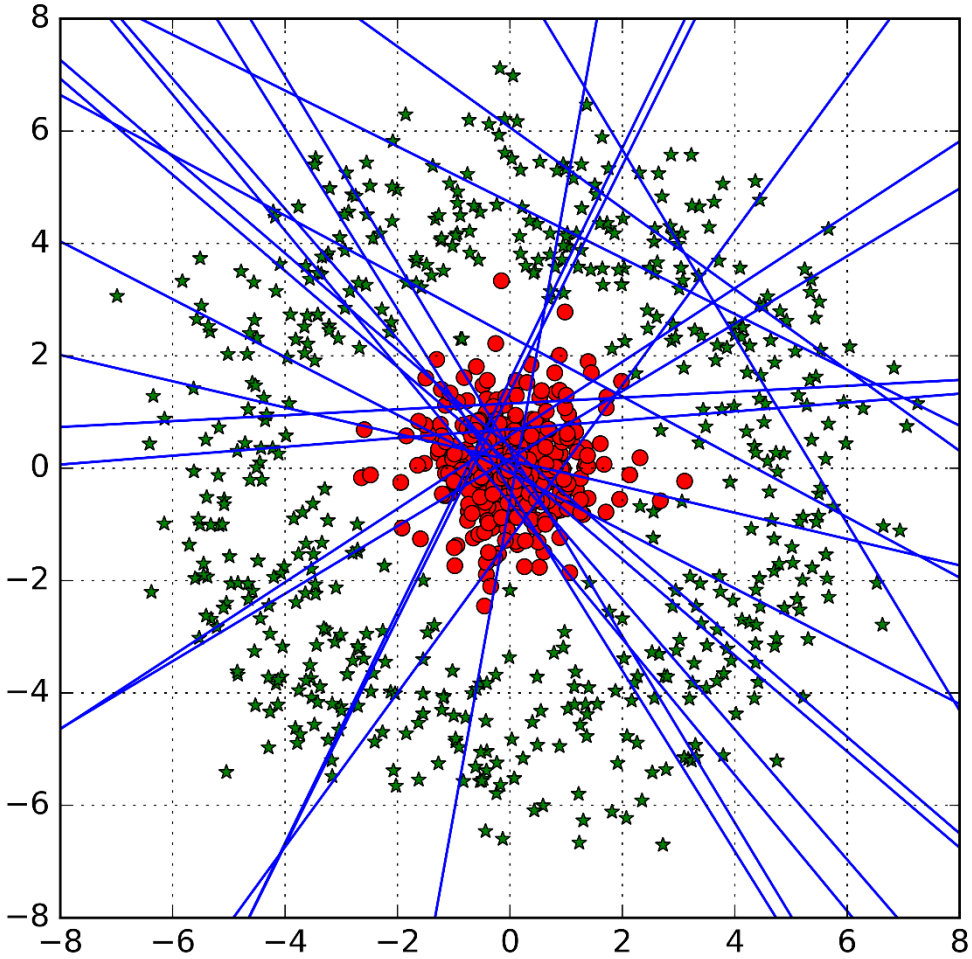
After 25 SGD steps



KAIST

Task 1

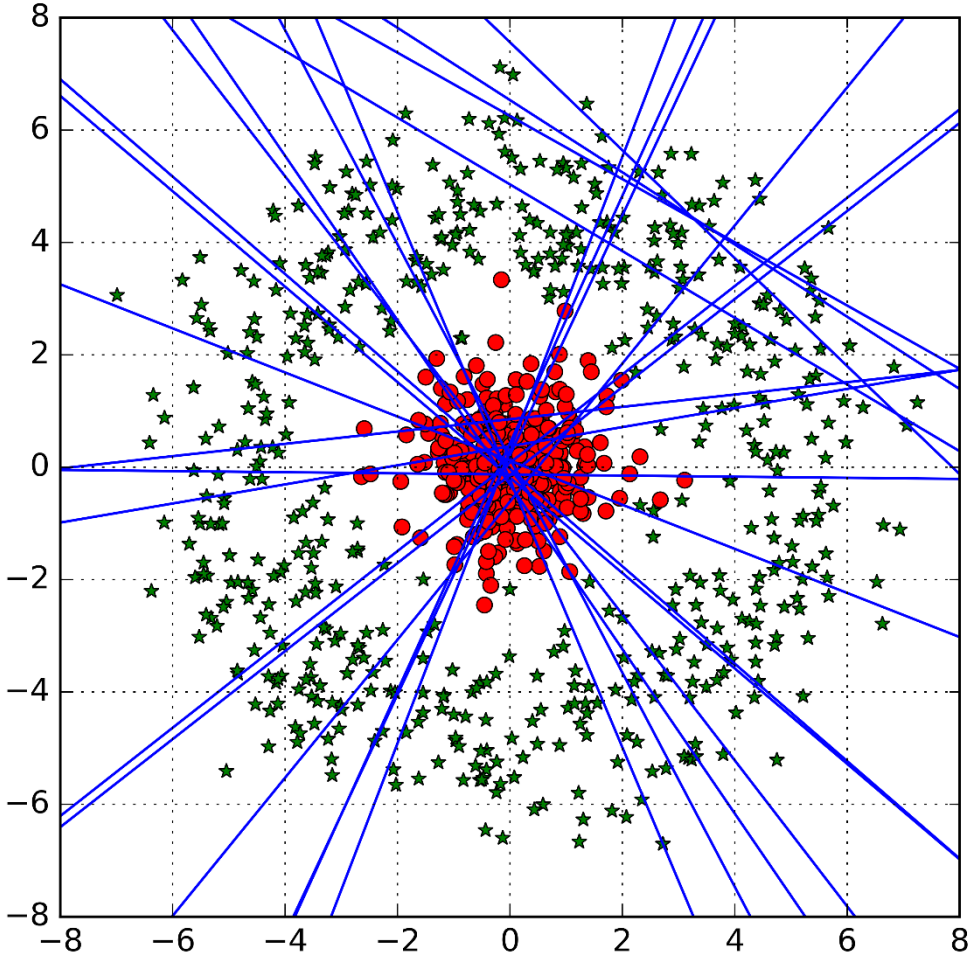
After 50 SGD steps



KAIST

Task 1

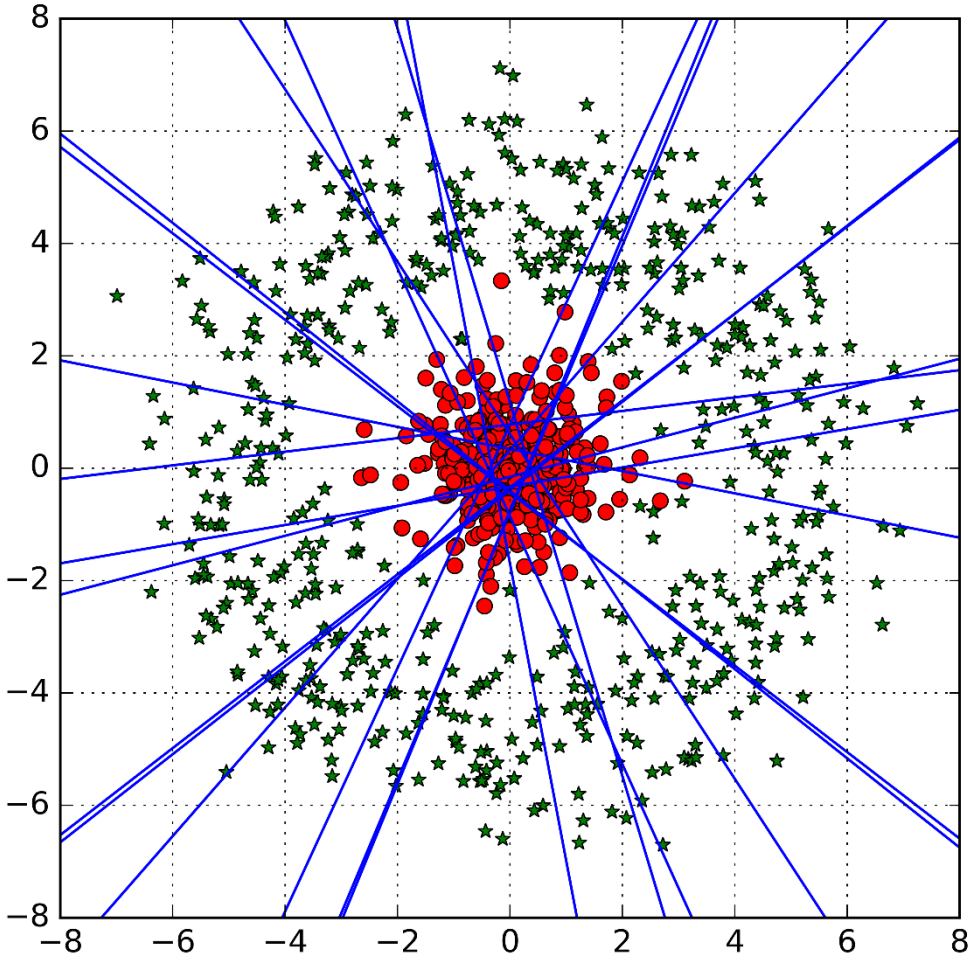
After 100 SGD steps



KAIST

Task 1

After 1000 SGD steps



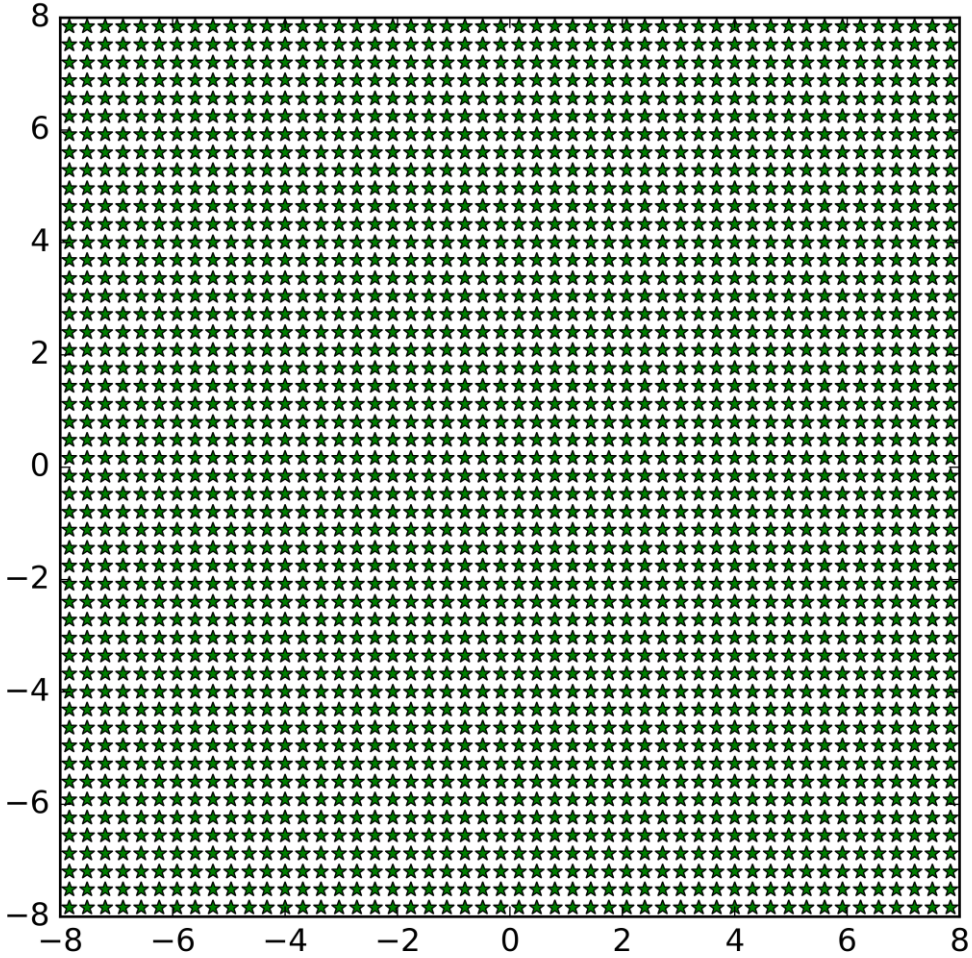
KAIST

Task 1

Before training

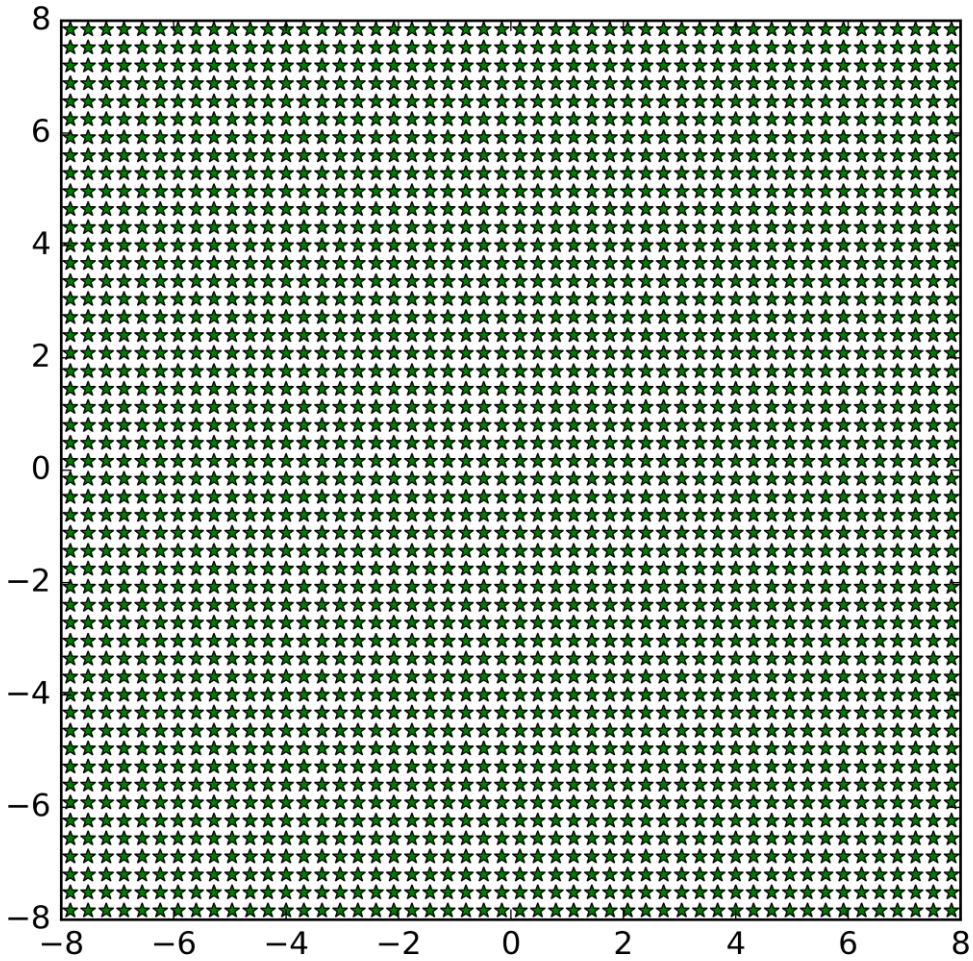
Task 1

After 10 SGD steps



Task 1

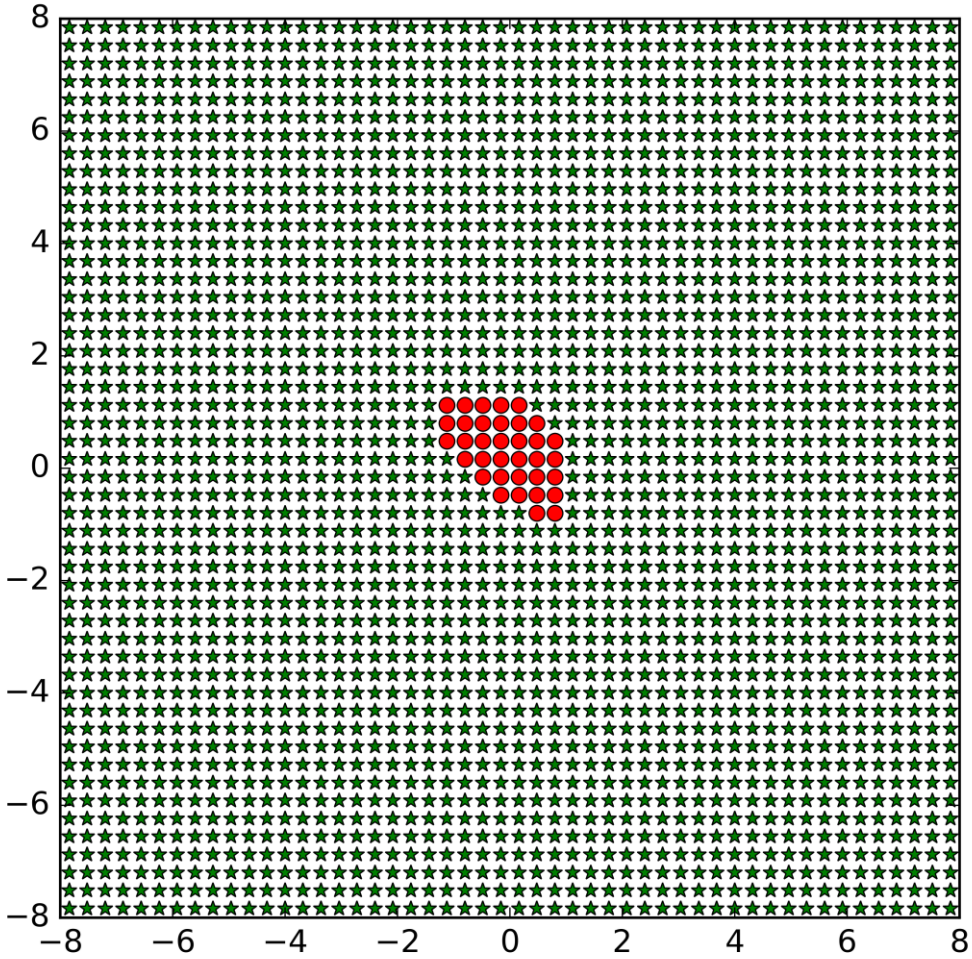
After 25 SGD steps



KAIST

Task 1

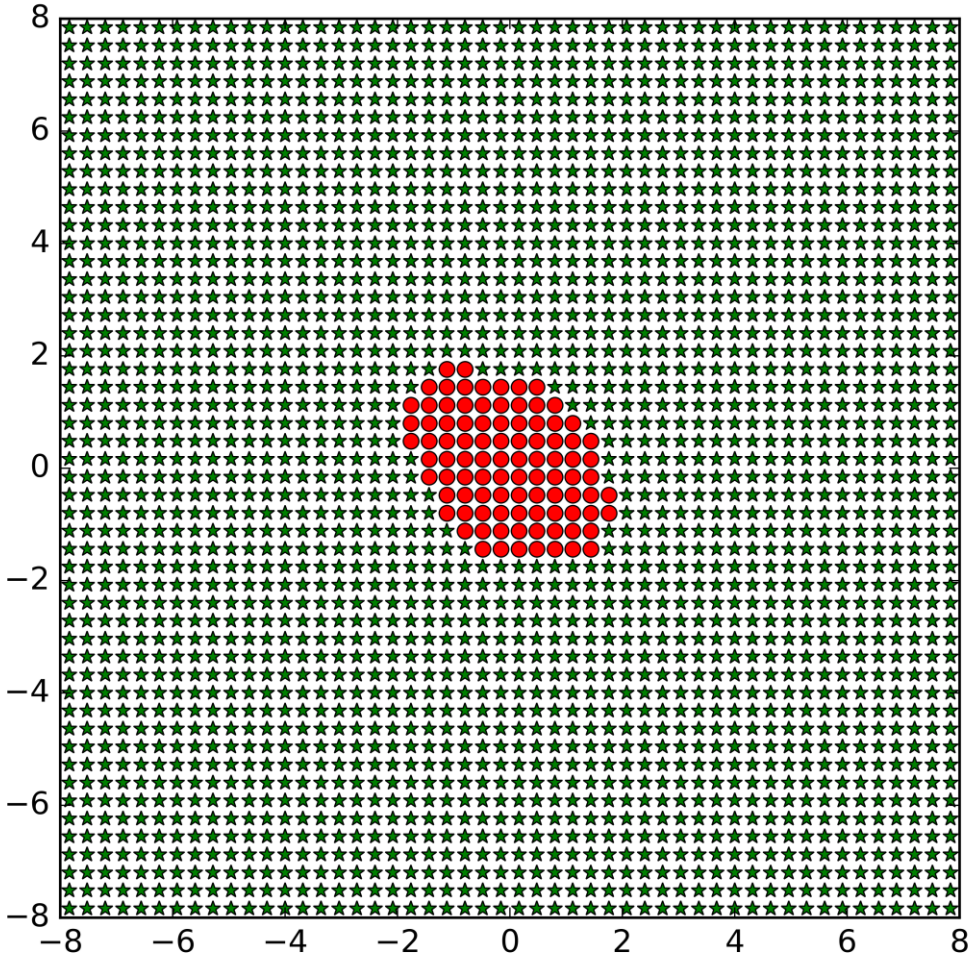
After 50 SGD steps



KAIST

Task 1

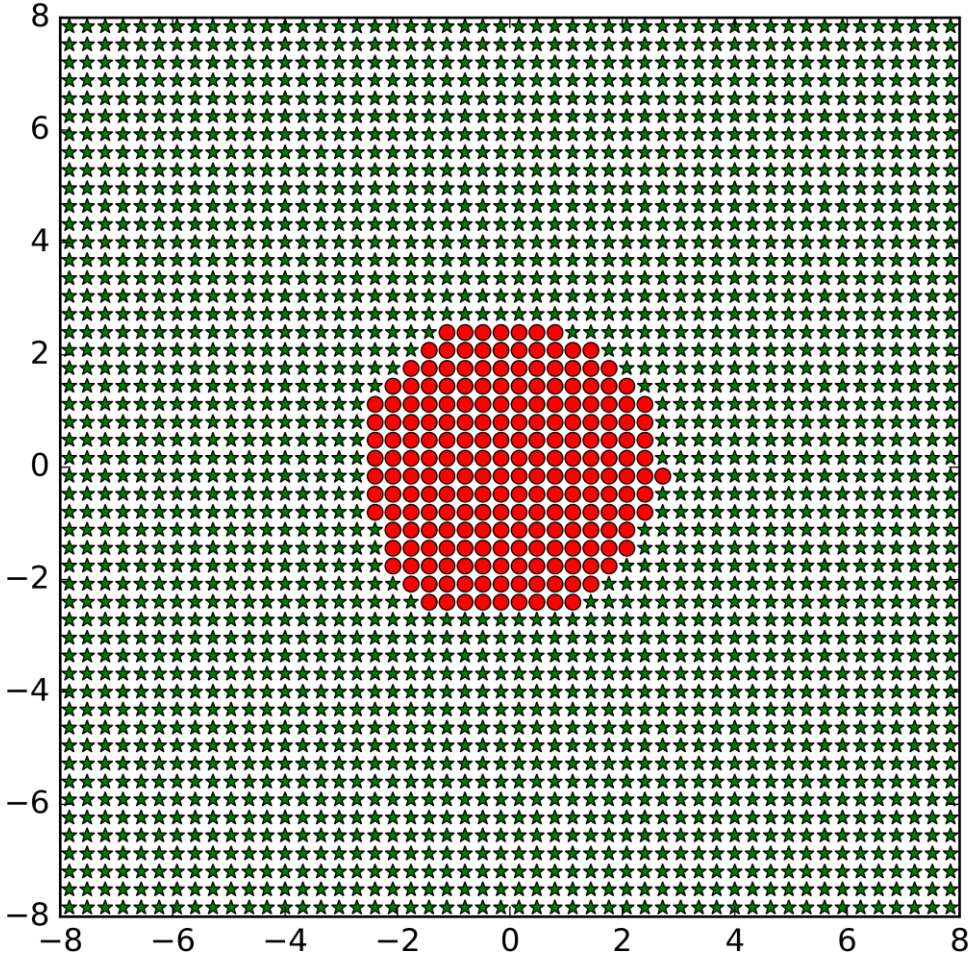
After 100 SGD steps



KAIST

Task 1

After 1000 SGD steps



KAIST

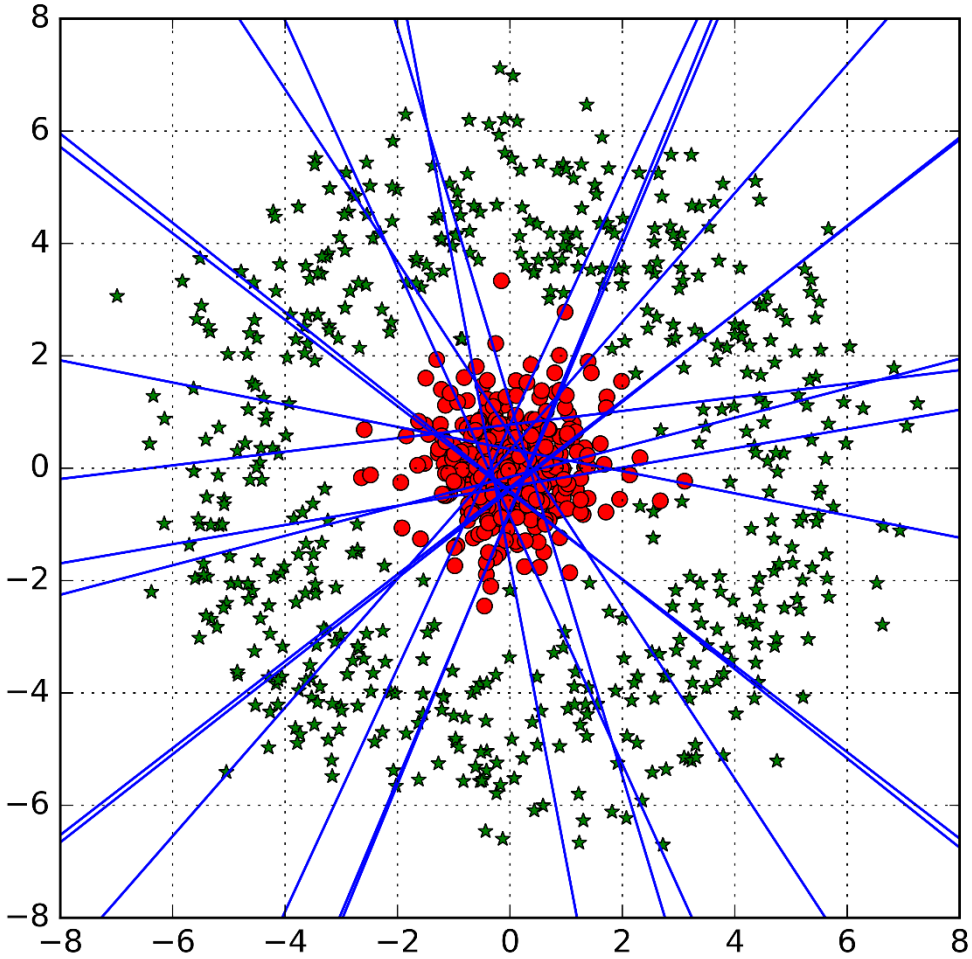
Task 1

Test accuracy: 0.9890

```
W_fc1=[[-0.69107968  0.83289248  0.71666086  0.01113512  0.3567695  -0.04778225
  -0.5127095   0.25392085  0.47210968 -0.83496195 -0.02550286 -1.02638435
   0.11379732 -0.25934827  0.30825436 -0.03156706 -0.04883416  0.39683303
   0.07302649 -0.2767081 ]
 [-0.13069014  0.25048164 -0.91417444 -0.09201895 -0.14639856 -0.05094288
  -0.66635793  1.28728104  0.30778161 -0.3785173  -0.03107899  0.47389507
  -0.66632491  0.22661941 -0.39856046 -0.03758751 -0.04067416 -0.1669932
  -0.27823114 -0.34845114]]

b_fc1=[-0.22006473 -0.29502028 -0.35706663  0.07127684 -0.11066952  1.31220722
 -0.29072025 -0.43791121 -0.1874508  -0.32270789  0.6020782  -0.36414447
 -0.21950881 -0.07111012 -0.13745987  0.87032992  1.12057614 -0.13288166
 -0.04316865 -0.13902982]

W_fc2=[[  7.12282836e-01],  [  9.09052730e-01],  [  1.19971871e+00],  [ -1.25133537e-03],  [  3.78481567e-01],
 [ -1.31396043e+00],  [  8.70186806e-01],  [  1.38032675e+00],  [  5.75001955e-01],  [  9.74261820e-01],
 [ -5.89137316e-01],  [  1.17575133e+00],  [  7.06265688e-01],  [  2.97493547e-01],  [  4.99691963e-01],
 [ -8.62129629e-01],  [ -1.11455798e+00],  [  4.34579015e-01],  [  2.36930788e-01],  [  4.39590335e-01]]

b_fc2=[-2.24619102]
```

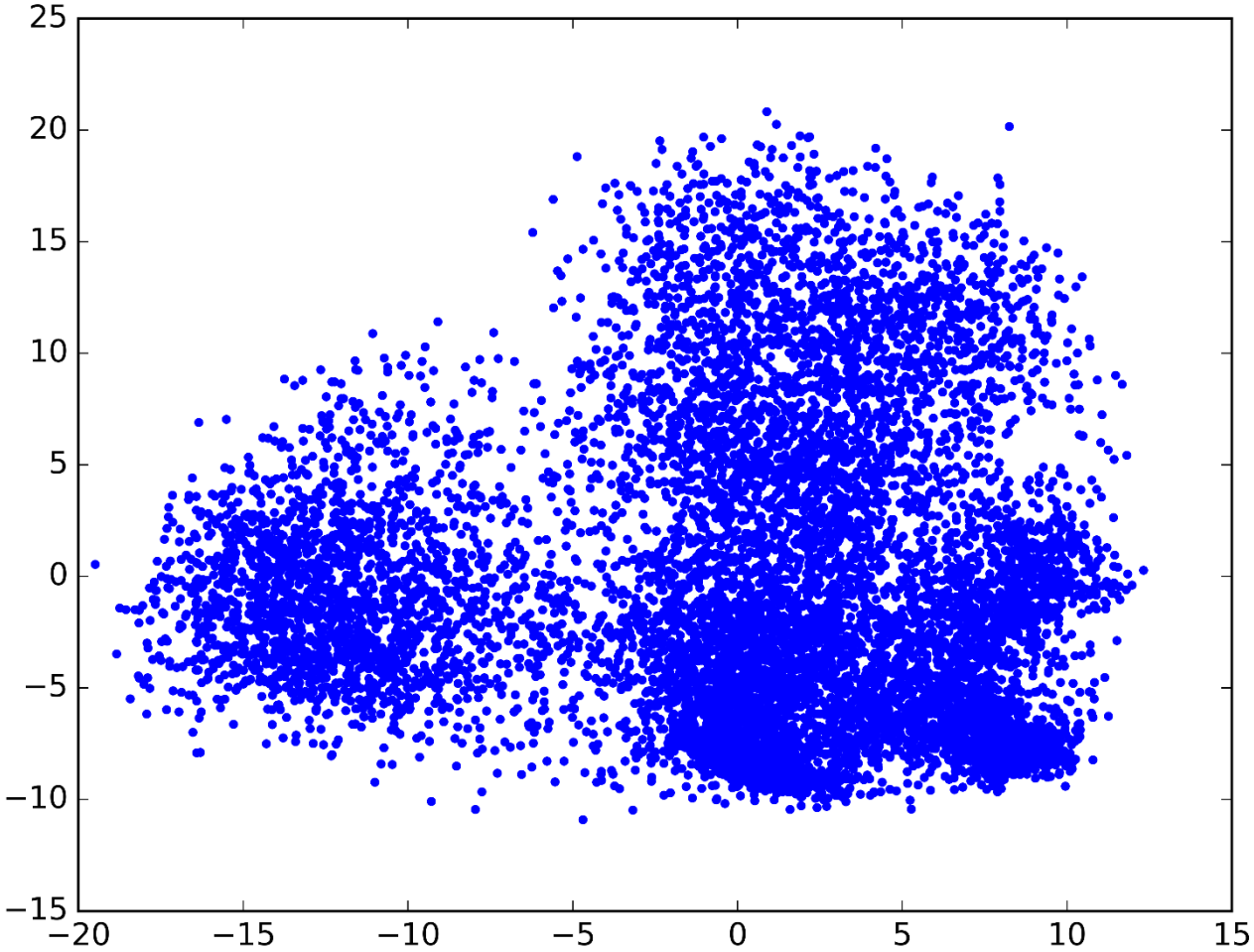
Task 1

After 1000 SGD steps

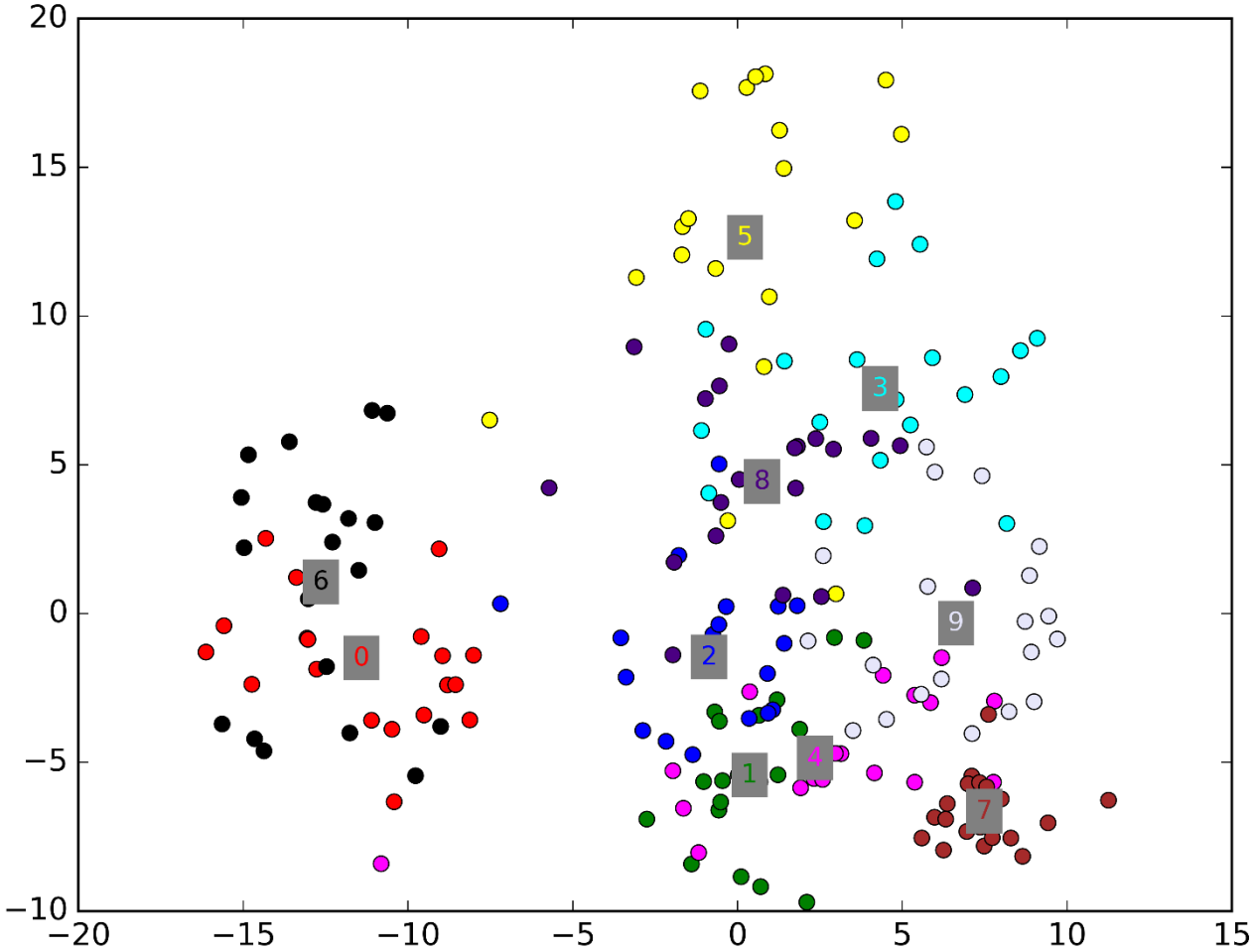


KAIST

Task 3



Task 3



KAIST

reduce_mean vs. reduce_sum

```
# mnist_conv1.py
cross_entropy = -tf.reduce_sum(y_*log(y_hat))
train_step = tf.train.AdamOptimizer(0.001).minimize(cross_entropy)
```

This was 1e-4 in mnist_conv1.py, but 1e-3 makes learning slightly faster. 1e-3 is also the default learning rate of AdamOptimizer

```
# reduce_mean is actually better since it provides normalization over mini-batch size
# Caution) it is still reduce_sum across each row of y_*log(y_hat)
# But, it does not matter if you use AdamOptimizer
#   since it already includes normalization of gradients

cross_entropy = -tf.reduce_mean(tf.reduce_sum(y_*log(y_hat),1))
train_step = tf.train.AdamOptimizer(0.001).minimize(cross_entropy)
```

Almost no difference in learning performance due to normalization of gradients by AdamOptimizer

Adaptive Momentum

- Adam (adaptive momentum)

$$
\begin{aligned}
\mathbf{g} &\leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)}) \\
\mathbf{s} &\leftarrow \rho_1 \mathbf{s} + (1-\rho_1)\mathbf{g} \\
\mathbf{r} &\leftarrow \rho_2 \mathbf{r} + (1-\rho_2)\mathbf{g} \odot \mathbf{g} \\
\hat{\mathbf{s}} &\leftarrow \frac{\mathbf{s}}{1-\rho_1^t} \\
\hat{\mathbf{r}} &\leftarrow \frac{\mathbf{r}}{1-\rho_2^t} \\
\Delta\boldsymbol{\theta} &\leftarrow -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\delta + \hat{\mathbf{r}}}} \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}
\end{aligned}
$$

AdamOptimizer in TensorFlow

```
__init__(
    learning_rate=0.001,
    beta1=0.9,
    beta2=0.999,
    epsilon=1e-08,
    use_locking=False,
    name='Adam'
)
```

Initialization:

```
m_0 <- 0 (Initialize initial 1st moment vector)
v_0 <- 0 (Initialize initial 2nd moment vector)
t <- 0 (Initialize timestep)
```

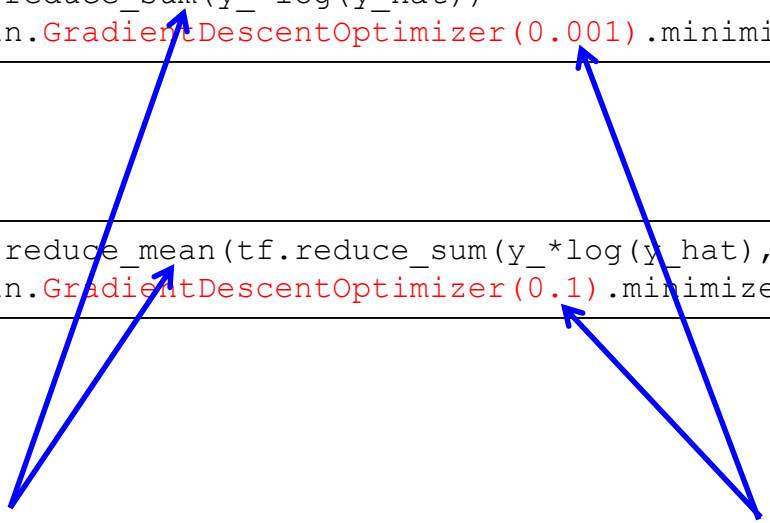
The update rule:

```
t <- t + 1
lr_t <- learning_rate * sqrt(1 - beta2^t) / (1 - beta1^t)

m_t <- beta1 * m_{t-1} + (1 - beta1) * g
v_t <- beta2 * v_{t-1} + (1 - beta2) * g * g
variable <- variable - lr_t * m_t / (sqrt(v_t) + epsilon)
```

reduce_mean vs. reduce_sum

```
cross_entropy = -tf.reduce_sum(y_*log(y_hat))
train_step = tf.train.GradientDescentOptimizer(0.001).minimize(cross_entropy)
```

```
cross_entropy = -tf.reduce_mean(tf.reduce_sum(y_*log(y_hat),1))
train_step = tf.train.GradientDescentOptimizer(0.1).minimize(cross_entropy)
```



However, if you use GradientDescentOptimizer, then there's difference between reduce_sum & reduce_mean

Learning rate needs to be lowered by the mini-batch size if reduce_sum is used

Softmax & Cross Entropy

```
# Not good since you take softmax and then log
# But, do this for TensorFlow version 0.11 (Computers in N5)

y_hat = tf.nn.softmax(tf.matmul(h_fc1, W_fc2) + b_fc2)

cross_entropy = -tf.reduce_mean(tf.reduce_sum(y_*log(y_hat),1))
train_step = tf.train.AdamOptimizer(0.001).minimize(cross_entropy)
```

```
# This is better, but tf.nn.softmax_cross_entropy_with_logits is available only for
# TensorFlow version 1.0 or higher (Computers in Haedong lounge and LG hall)

h_fc2 = tf.matmul(h_fc1, W_fc2) + b_fc2
y_hat = tf.nn.softmax(h_fc2)

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=h_fc2))
train_step = tf.train.AdamOptimizer(0.001).minimize(cross_entropy)
```