forward-backward-kl

June 7, 2020

0.1 KL divergence minimization

In this exercise, we will minimize KL divergence between the true distribution $p(x)$ and our model $q_\theta(x)$ using gradient descent. We will be utilizing Tensorflow Probability (TFP) built on Tensorflow that makes it easy to combine probabilistic models and deep learning. If you haven't already, please go ahead and install the library.

[0]:
```
import numpy as np
import tensorflow.compat.v1 as tf
tf.disable_eager_execution()
import tensorflow_probability as tfp
import matplotlib.pyplot as plt
%matplotlib inline
import subprocess

FIGSIZE = (6, 6)
```

[0]:
```
# Evaluate the probabilities given the samples
def prob_map(p, lim, N):
    x = np.linspace(lim[0], lim[1], N)
    y = np.linspace(lim[2], lim[3], N)
    xy = np.array(np.meshgrid(x, y)).reshape(2, N**2).T
    xy_prob = p.prob(xy.astype(np.float32)).eval()
    x, y = np.meshgrid(x, y)
    return x, y, xy_prob.reshape((N, N))

# Create mean and covariance params for bivariate normal distribution
def gen_gaussian_params(d=2):
    p_mean = np.random.standard_normal((d,)).astype(np.float32)
    p_cov = np.random.standard_normal((d, d)).astype(np.float32)
    p_cov[0, 1] = 0
    p_cov = np.dot(p_cov, p_cov.T)
    return p_mean, p_cov

# Plot helpers
def contour(x, y, xy_prob, lim, cmap=plt.cm.inferno, axis=None):
    if axis is None:
        plt.contour(x, y, xy_prob, np.linspace(0., 1.1*xy_prob.max(), 10),
            zorder=2, extent=lim, linewidths=4, colors='w', alpha=0.5)
```

```
        plt.contour(x, y, xy_prob, np.linspace(0., 1.1*xy_prob.max(), 10),
                zorder=3, extent=lim, linewidths=2, cmap=cmap)
    else:
        axis.contour(x, y, xy_prob, np.linspace(0., 1.1*xy_prob.max(), 10),
            zorder=2, extent=lim, linewidths=4, colors='w', alpha=0.5)
        axis.contour(x, y, xy_prob, np.linspace(0., 1.1*xy_prob.max(), 10),
                zorder=3, extent=lim, linewidths=2, cmap=cmap)

def scatter(sample, axis=None):
    if axis is None:
        plt.scatter(sample[:200, 0], sample[:200, 1], s=25, alpha=0.7, c='k')
    else:
        axis.scatter(sample[:200, 0], sample[:200, 1], s=25, alpha=0.7, c='k')
```

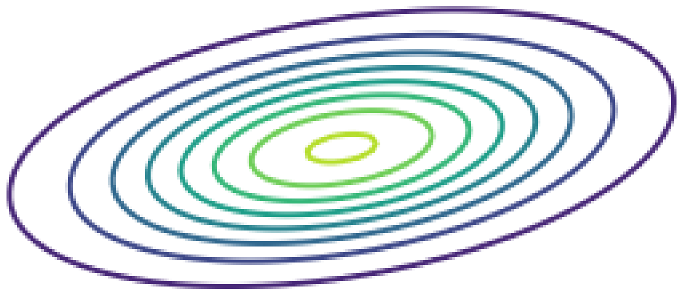
0.1.1 Create a bivariate normal as the true distribution $p(x) = \mathcal{N}(\mu, \Sigma)$

Draw the probability contour plot for the true distribution p(x)

[3]:
```
np.random.seed(555)
tf.reset_default_graph()
sess = tf.InteractiveSession()

p_mean, p_cov = gen_gaussian_params()
p_mean += 0.8
p = tfp.distributions.MultivariateNormalFullCovariance(p_mean, p_cov)

plt.figure(figsize=FIGSIZE)
N = 128
lim = [-2.5, 3.5, -2.5, 3.5]
x, y, xy_prob = prob_map(p, lim, N)
contour(x, y, xy_prob, lim, cmap=plt.cm.viridis)

plt.axis(lim)
plt.axis('off')
plt.show()
```

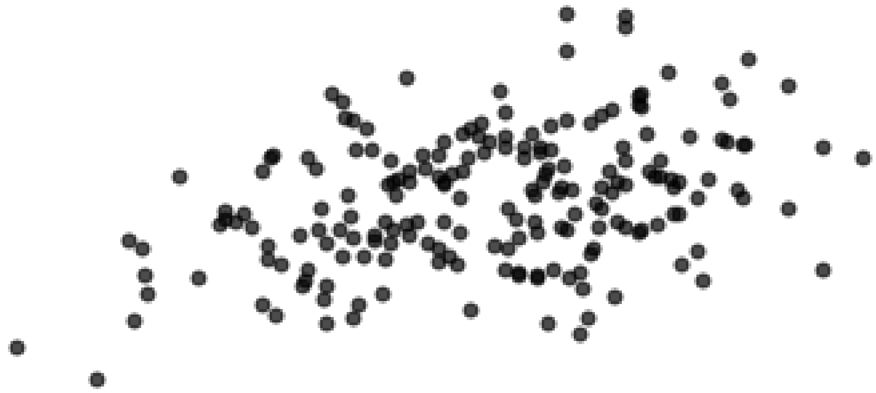
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow_probability/python/distributions/distribution.py:332:
MultivariateNormalFullCovariance.__init__ (from
tensorflow_probability.python.distributions.mvn_full_covariance) is deprecated
and will be removed after 2019-12-01.
Instructions for updating:
`MultivariateNormalFullCovariance` is deprecated, use
`MultivariateNormalTriL(loc=loc,
scale_tril=tf.linalg.cholesky(covariance_matrix))` instead.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/linalg/linear_operator_lower_triangular.py:158:
calling LinearOperator.__init__ (from
```

```
tensorflow.python.ops.linalg.linear_operator) with graph_parents is deprecated
and will be removed in a future version.
Instructions for updating:
Do not pass `graph_parents`.  They will  no longer be used.
```



Draw samples from the true distribution $p(x)$ and visualize

[0]:
```
n_sample = 1000
sample = p.sample(n_sample).eval()

plt.figure(figsize=FIGSIZE)
scatter(sample)
plt.axis(lim)
plt.axis('off')
plt.show()
```



0.2 Part A. Forward KL minimization

0.2.1 Create the model distribution

Now we will create $q_\theta(x)$ and apply gradient descent to minimize the forward KL divergence

$$\theta^* = \arg\min_\theta D_{KL}(p(x) \parallel q_\theta(x))$$

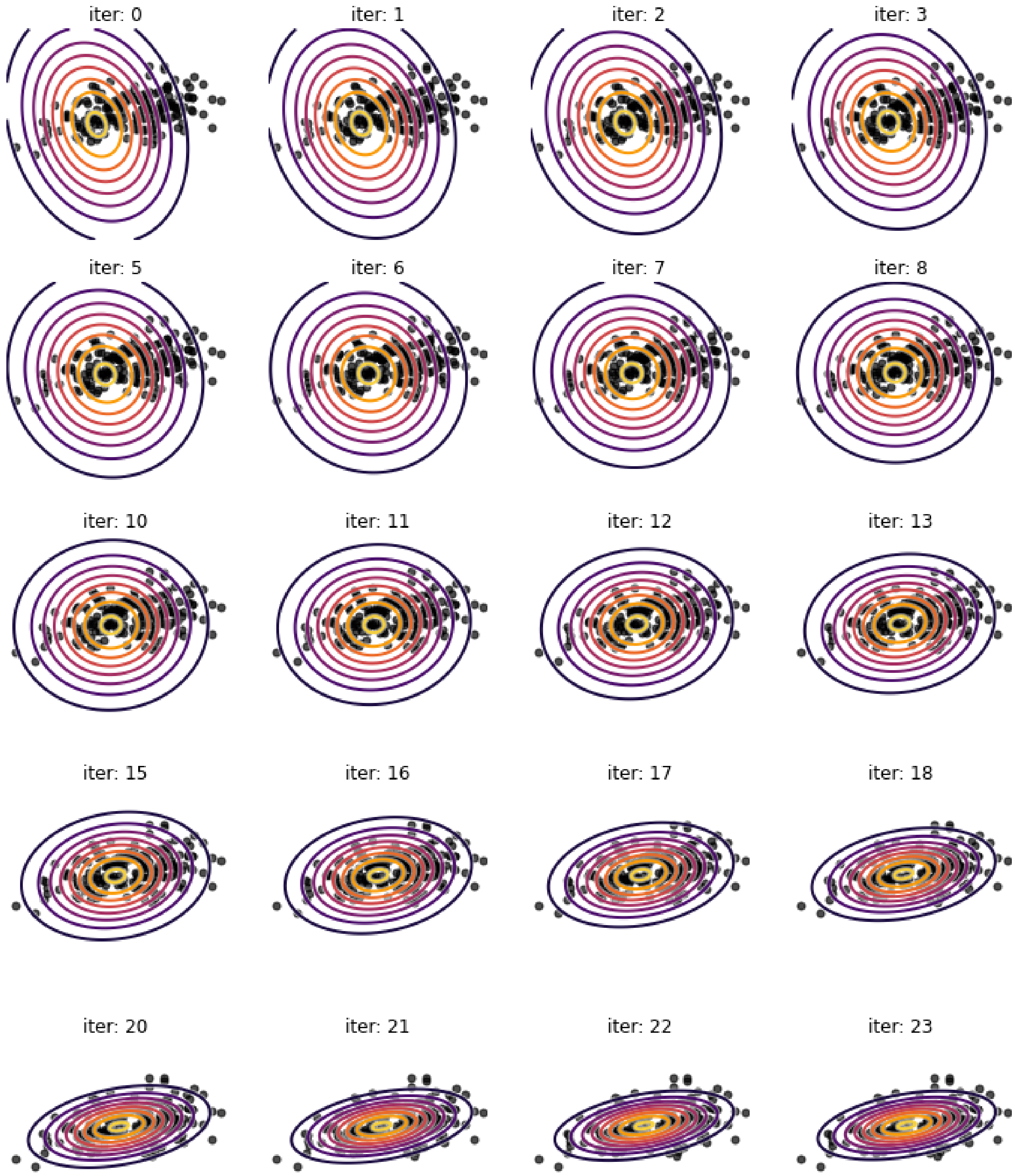
[0]:
```
# Define your model here!
qm = np.random.standard_normal((2,)).astype(np.float32)
qr = np.random.standard_normal((2, 2)).astype(np.float32)

q_mean = tf.Variable(qm) # will convege to true p
q_raw  = tf.Variable(qr)
q_cov = tf.matmul(q_raw, tf.transpose(q_raw))

q = tfp.distributions.MultivariateNormalFullCovariance(q_mean, q_cov)
print(type(p_mean), type(q_mean))
# Define the loss here!
```

```
mu12 = tf.reshape(p_mean - q_mean, [1,2])
q_cov_inv = tf.linalg.inv(q_cov)
sum1 = tf.matmul(tf.matmul(mu12, q_cov_inv), tf.transpose(mu12))
trc = tf.linalg.trace(tf.matmul(q_cov_inv, p.covariance()))

loss = 0.5 * (tf.math.log(tf.linalg.det(q_cov)) + trc + sum1)
train = tf.train.GradientDescentOptimizer(0.1).minimize(loss, var_list =
 ↪[q_mean, q_raw])
sess.run(tf.global_variables_initializer())
```

```
<class 'numpy.ndarray'> <class
'tensorflow.python.ops.resource_variable_ops.ResourceVariable'>
```

Lets run it for 20 iterations and visualize $q_{\theta^{(t)}}(x)$

[0]:
```
fig, axes = plt.subplots(5,4, figsize=[12, 15])

for i in range(5):
    for j in range(4):
        #print(q_raw.eval(session=sess), q_mean.eval(session=sess), q_temp.
 ↪eval(session=sess))
        scatter(sample, axis=axes[i][j])
        x, y, xy_prob = prob_map(q, lim, N)
        contour(x, y, xy_prob, lim, axis=axes[i][j])
        axes[i][j].axis(lim)
        axes[i][j].axis('off')
        axes[i][j].set_title('iter: {}'.format(i*5 + j))
        sess.run(train)

plt.show()
plt.close(fig)

sess.close()
```



0.2.2 Create a mixture of two bivariate normals as the true distribution

$p(x) = \pi_1 \mathcal{N}(\mu_1, \Sigma_1) + \pi_2 \mathcal{N}(\mu_2, \Sigma_2)$ where $\pi_1 + \pi_2 = 1$

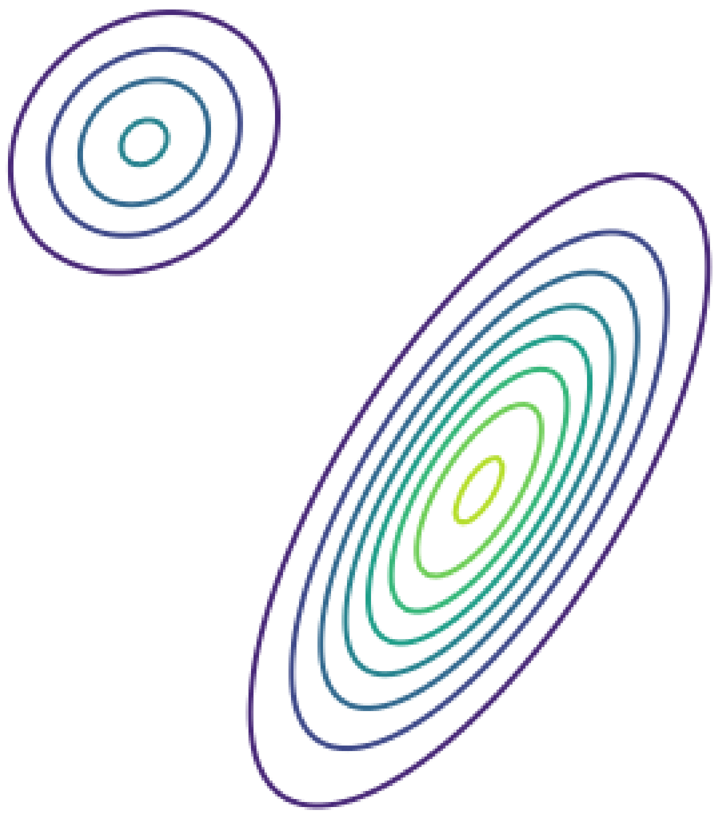
[11]:
```
def build_p(mix=0.8):
    return tfp.distributions.Mixture(
```

```
        cat=tfp.distributions.Categorical(probs=[mix, 1.-mix]),
        components=[
            tfp.distributions.MultivariateNormalFullCovariance(
                loc=[0.5, -0.7], covariance_matrix=[[1., 1.], [1., 2.]]),
            tfp.distributions.MultivariateNormalFullCovariance(
                loc=[-2.5, 2.5], covariance_matrix=[[0.5, 0.1], [0.1, 0.5]]),
        ])

np.random.seed(8)
tf.reset_default_graph()
sess = tf.InteractiveSession()

p = build_p()
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/client/session.py:1751:
UserWarning: An interactive session is already active. This can cause out-of-
memory errors in some cases. You must explicitly call
`InteractiveSession.close()` to release resources held by the other session(s).
  warnings.warn('An interactive session is already active. This can '
```

Draw the probability contour plot for the true distribution p(x)

[12]:
```
N = 128
lim = [-4, 4, -4, 4]
plt.figure(figsize=FIGSIZE)
x, y, xy_prob = prob_map(p, lim, N)
contour(x, y, xy_prob, lim, cmap=plt.cm.viridis)
plt.axis(lim)
plt.axis('off')
plt.show()
```



Draw samples from the true distribution $p(x)$ and visualize

[13]:
```
n_sample = 1000
sample = p.sample(n_sample).eval()
plt.figure(figsize=FIGSIZE)
scatter(sample)
plt.axis(lim)
plt.axis('off')
plt.show()
```

0.2.3 Create the model distribution

Now we will create $q_\theta(x)$ and apply gradient descent to minimize the forward KL divergence

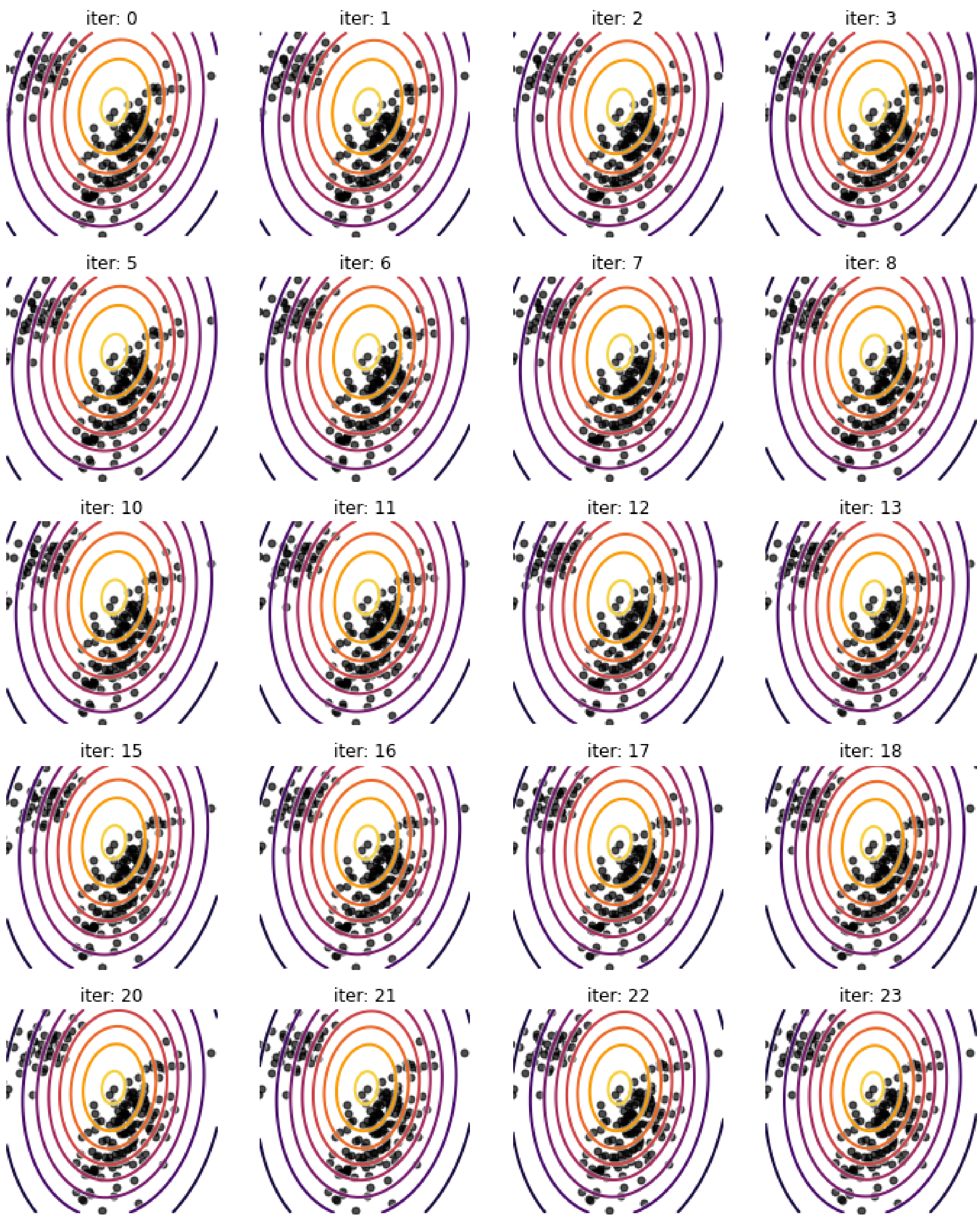
$$\theta^* = \arg\min_\theta D_{KL}(p(x) \parallel q_\theta(x))$$

[0]:
```
# Define your model here!
qm = np.random.standard_normal((2,)).astype(np.float32)
qr = np.random.standard_normal((2, 2)).astype(np.float32)

q_mean = tf.Variable(qm) # will convege to true p
q_raw  = tf.Variable(qr)
q_cov = tf.matmul(q_raw, tf.transpose(q_raw))
#print(a.dtype,b,q_mean)
q = tfp.distributions.MultivariateNormalFullCovariance(q_mean, q_cov)

#print(sample.shape, p_mean.shape, p_cov.shape)

# Define the loss here!
```

```
loss = -tfp.monte_carlo.expectation(f = (lambda x : q.log_prob(x)), samples =␣
 ↪sample)
#logp = None
train = tf.train.GradientDescentOptimizer(0.1).minimize(loss, var_list =␣
 ↪[q_mean, q_raw])
sess.run(tf.global_variables_initializer())
```

Lets run it for 20 iterations and visualize $q_{\theta^{(t)}}(x)$

[15]:
```
fig, axes = plt.subplots(5,4, figsize=[12, 15])

for i in range(5):
    for j in range(4):
        scatter(sample, axis=axes[i][j])
        x, y, xy_prob = prob_map(q, lim, N)
        contour(x, y, xy_prob, lim, axis=axes[i][j])
        axes[i][j].axis(lim)
        axes[i][j].axis('off')
        axes[i][j].set_title('iter: {}'.format(i*5 + j))
        sess.run(train)

plt.show()
plt.close(fig)

sess.close()
```



0.3 Part B. Reverse KL minimization

[16]:
```
tf.reset_default_graph()
sess = tf.InteractiveSession()
p = build_p()
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/client/session.py:1751:
UserWarning: An interactive session is already active. This can cause out-of-
memory errors in some cases. You must explicitly call
`InteractiveSession.close()` to release resources held by the other session(s).
  warnings.warn('An interactive session is already active. This can '
```

0.3.1 Create the model distribution

Now we will create $q_\theta(x)$ and apply gradient descent to minimize the reverse KL divergence

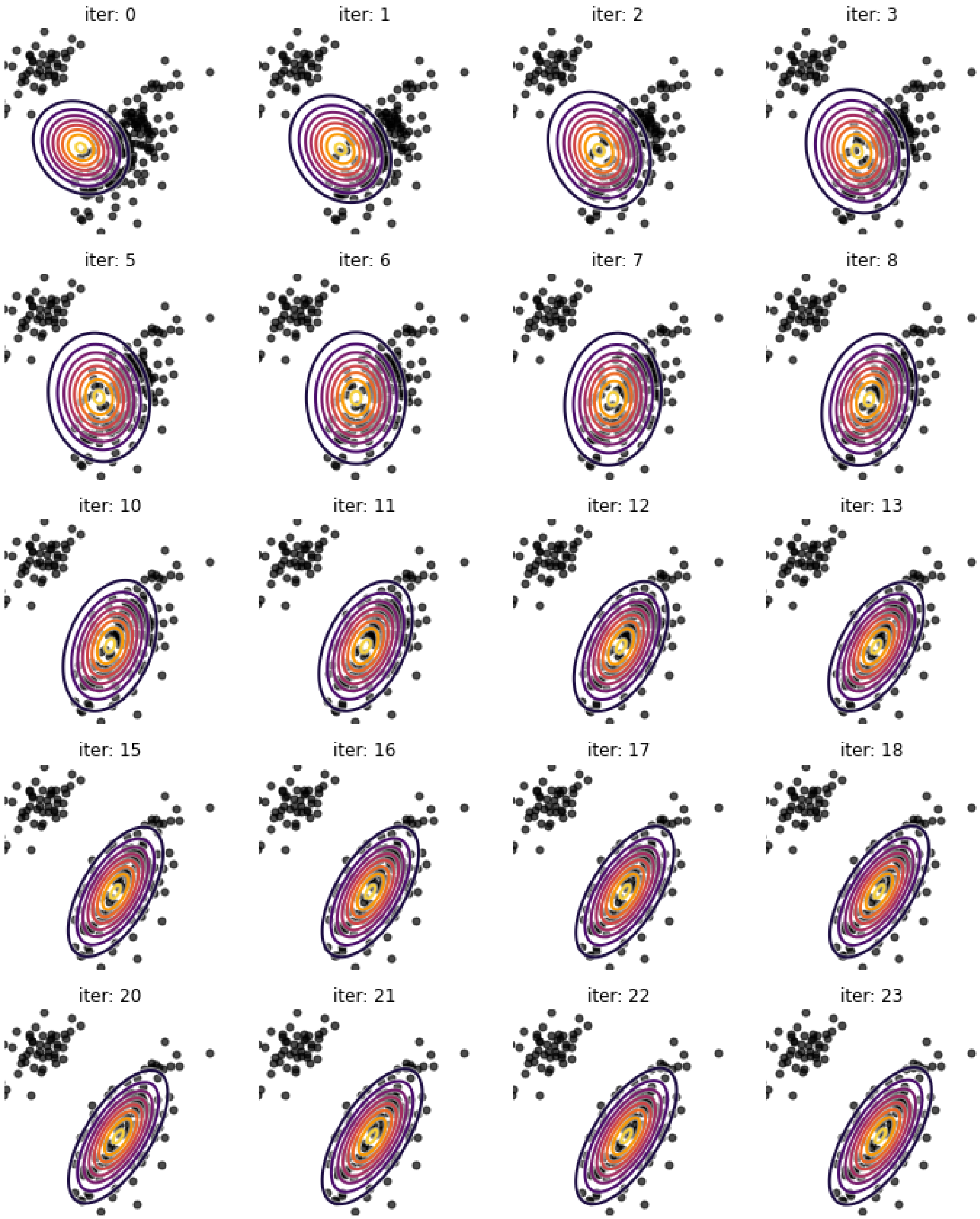
$$\theta^* = \arg\min_\theta D_{KL}(q_\theta(x) \parallel p(x))$$

[0]:
```
# Define your model here!
qm = np.random.standard_normal((2,)).astype(np.float32)
qr = np.random.standard_normal((2, 2)).astype(np.float32)

q_mean = tf.Variable(qm) # will convege to true p
q_raw  = tf.Variable(qr)
q_cov = tf.matmul(q_raw, tf.transpose(q_raw))

## 2 -> p, 1 -> q

q = tfp.distributions.MultivariateNormalFullCovariance(q_mean, q_cov)
loss = tfp.monte_carlo.expectation(
    f=lambda x: -p.log_prob(x) + q.log_prob(x),
    samples = q.sample(1000)
)

# Define the loss here!
train = tf.train.GradientDescentOptimizer(0.1).minimize(loss, var_list =
 ↪[q_mean, q_raw])
sess.run(tf.global_variables_initializer())
```

Lets run it for 20 iterations and visualize $q_{\theta^{(t)}}(x)$

[19]:
```
fig, axes = plt.subplots(5,4, figsize=[12, 15])

for i in range(5):
    for j in range(4):
        scatter(sample, axis=axes[i][j])
        x, y, xy_prob = prob_map(q, lim, N)
        contour(x, y, xy_prob, lim, axis=axes[i][j])
        axes[i][j].axis(lim)
        axes[i][j].axis('off')
        axes[i][j].set_title('iter: {}'.format(i*5 + j))
        sess.run(train)

plt.show()
plt.close(fig)
```

```
sess.close()
```


iter: 0
iter: 1
iter: 2
iter: 3
iter: 5
iter: 6
iter: 7
iter: 8
iter: 10
iter: 11
iter: 12
iter: 13
iter: 15
iter: 16
iter: 17
iter: 18
iter: 20
iter: 21
iter: 22
iter: 23