Representation_learning_using_Gaussian_Process

April 26, 2019

0.0.1 GPflow

In [ ]:
```
from utils import *
```

In [ ]:
```
data = np.loadtxt('data/DataTrn.txt')
labels_ = np.loadtxt("data/DataTrnLbls.txt")
Y = data
```

In [ ]:
```
Q = 5
M = 20
N = Y.shape[0]
X_mean = gpflow.models.PCA_reduce(Y, Q)
Z = np.random.permutation(X_mean.copy())[:M]

fHmmm = False
if(fHmmm):
    k = (kernels.RBF(3, ARD=True, active_dims=slice(0,3)) +
         kernels.Linear(2, ARD=False, active_dims=slice(3,5)))
else:
    k = (kernels.RBF(3, ARD=True, active_dims=[0,1,2]) +
         kernels.Linear(2, ARD=False, active_dims=[3, 4]))

m = gpflow.models.BayesianGPLVM(X_mean=X_mean, X_var=0.1*np.ones((N, Q)), Y=Y,
                                kern=k, M=M, Z=Z)
m.likelihood.variance = 0.01

opt = gpflow.train.ScipyOptimizer()
m.compile()
opt.minimize(m, maxiter=gpflow.test_util.notebook_niter(10000))
```

In [ ]:
```
m.kern.as_pandas_table()
```

In [ ]:
```
kern = m.kern.kernels[0]
sens = np.sqrt(kern.variance.read_value())/kern.lengthscales.read_value()
print(sens)
```

In [ ]:
```
fig, ax = plt.subplots()
dims = np.arange(len(sens))
```

```
        ax.bar(dims, sens, 0.1, color='y')
        ax.set_xticks(dims)
        ax.set_xlabel('dimension')
        ax.set_title('Sensitivity to latent inputs');
```

```
In [ ]: dim1, dim2 = sens.argsort()[::-1][:2]  # the two dimensions with highest sensitivity
```

```
In [ ]: labels = [l.argmax() for l in labels_]
```

```
In [ ]: XPCAplot = gpflow.models.PCA_reduce(Y, 2)
        GPLVM_X_mean = m.X_mean.read_value()

        f, ax = plt.subplots(1,2, figsize=(10,6))
        colors = cm.rainbow(np.linspace(0, 1, len(np.unique(labels))))

        for i, c in zip(np.unique(labels), colors):
            ax[0].scatter(XPCAplot[labels==i, 0], XPCAplot[labels==i, 1], color=c, label=i)
            ax[0].set_title('PCA')
            ax[1].scatter(GPLVM_X_mean[labels==i, dim1], GPLVM_X_mean[labels==i, dim2], color=
            ax[1].set_title('Bayesian GPLVM')
        plt.savefig("oil_pca_rep.png")
```

```
In [ ]: # reconstruction error based only on the mean predictions
        ((m.predict_y(GPLVM_X_mean[:,:])[0] - Y[:,:])**2).sum(axis = 1).mean()
```

0.0.2 same on MNIST

```
In [ ]: MNIST_train = torchvision.datasets.MNIST(root=root, train=True, transform=trans, downlo
        MNIST_test  = torchvision.datasets.MNIST(root=root, train=False, transform=trans, downl
```

```
In [ ]: plt.imshow(MNIST_train[0][0].reshape(28,28))
```

```
In [ ]: plt.imshow(MNIST_test[0][0].reshape(28,28)), MNIST_test[0][1]
```

```
In [ ]:  MNIST_train[0][1]
```

```
In [ ]: x = []
        labels = []
        for t, l in MNIST_train:
            x.append(t.numpy().flatten())
            labels.append(l)
```

```
In [ ]: x = np.array(x)
        labels = np.array(labels)
```

```
In [ ]: x.shape
```

```
In [ ]: x_5 = None
        l_5 = []
        samples = 20
```

```
        for n in range(5):
            d = x[np.where(labels == n)]
            idxs = np.random.choice(range(d.shape[0]), samples)

            print(d[idxs].shape)
            if x_5 is None:
                x_5 = d[idxs]
            else:
                x_5 = np.vstack((x_5, d[idxs]))
            l_5 = np.hstack((l_5,n* np.ones(samples)))
```

In [ ]:
```
x_5 = np.array(x_5)
```

In [ ]:
```
Y = x_5.astype(np.float64)
Y.shape
```

In [ ]:
```
del x
```

In [ ]:
```
Q = 5
M = 20
N = Y.shape[0]
X_mean = gpflow.models.PCA_reduce(Y, Q)
Z = np.random.permutation(X_mean.copy())[:M]
kernel_dim = 5

k = kernels.RBF(kernel_dim, ARD=True, active_dims=slice(0,kernel_dim))
m = gpflow.models.BayesianGPLVM(X_mean=X_mean, X_var=0.1*np.ones((N, Q)), Y=Y,
                                kern=k, M=M, Z=Z)
m.likelihood.variance = 0.01

opt = gpflow.train.ScipyOptimizer()
m.compile()
opt.minimize(m, maxiter=gpflow.test_util.notebook_niter(500))
print("done")
```

In [ ]:
```
m
```

In [ ]:
```
m.kern.as_pandas_table()
```

In [ ]:
```
kern = m.kern#.kernels[0]
sens = np.sqrt(kern.variance.read_value())/kern.lengthscales.read_value()
print(sens)
```

In [ ]:
```
fig, ax = plt.subplots()
dims = np.arange(len(sens))
ax.bar(dims, sens, 0.1, color='y')
ax.set_xticks(dims)
ax.set_xlabel('dimension')
ax.set_title('Sensitivity to latent inputs');
plt.savefig('mnist_1l_sens.png')
```

```
In [ ]: dim1, dim2 = sens.argsort()[::-1][:2]

In [ ]: XPCAplot = gpflow.models.PCA_reduce(Y, 2)
        GPLVM_X_mean = m.X_mean.read_value()

        f, ax = plt.subplots(1,2, figsize=(10,6))
        colors = cm.rainbow(np.linspace(0, 1, len(np.unique(l_5))))

        for i, c in zip(np.unique(labels), colors):
            ax[0].scatter(XPCAplot[l_5==i, 0], XPCAplot[l_5==i, 1], color=c, label=i, alpha = 
            ax[0].set_title('PCA')
            ax[1].scatter(GPLVM_X_mean[l_5==i, dim1], GPLVM_X_mean[l_5==i, dim2], color=c, labe
            ax[1].set_title('Bayesian GPLVM')
            f.savefig("mnist_1l_pca_rep.png")

In [ ]: def reconstruct(Q, i):
            print(i)
            y = m.predict_y(GPLVM_X_mean[i].reshape(1,Q))[0]

            plt.imshow(y.reshape(28,28))
            plt.figure()
            plt.imshow(Y[i].reshape(28,28))

        reconstruct(Q, i=np.random.randint(0,100))
```

0.0.3 Deep Model Implementation

```
In [ ]: dgp = DeepGPLVM(kernel_dims = [10,5], n_layers = 2, max_iters=300, latent_dims = [20, 5
        Y = x_5.astype(np.float64)
        dgp.train(Y)

In [ ]: sens = dgp.get_sensitivities()[0]
        dim1, dim2 = sens.argsort()[::-1][:2]
        XPCAplot = gpflow.models.PCA_reduce(Y, 2)
        dgp_mean = dgp.means

        f, ax = plt.subplots(1,2, figsize=(10,6))
        colors = cm.rainbow(np.linspace(0, 1, len(np.unique(l_5))))

        for i, c in zip(np.unique(labels), colors):
            ax[0].scatter(XPCAplot[l_5==i, 0], XPCAplot[l_5==i, 1], color=c, label=i, alpha = 
            ax[0].set_title('PCA')
            ax[1].scatter(dgp_mean[l_5==i, dim1], dgp_mean[l_5==i, dim2], color=c, label=i, alp
            ax[1].set_title('deep Bayesian GPLVM')
            f.savefig("mnist_2l_pca_rep.png")

In [ ]: # GPLVM_X_mean = m.X_mean.read_value()
        # recon = dgp.reconstruct(GPLVM_X_mean[6].reshape(1,Q))
        i = np.random.randint(0,100)
```

```
        recon = dgp.reconstruct(i)
        plt.figure()
        plt.imshow(recon.reshape(28,28))
        plt.savefig("test1.png")
        plt.figure()
        plt.imshow(Y[i].reshape(28,28))

In [ ]: # from google.colab import files
        # files.download("test.png")

In [ ]: idxs = np.random.randint(0, Y.shape[0],10)#np.random.permutation(dgp.means)[:10]

        # random_sample = dgp.means[idxs]

        plt.figure()
        for i, idx in enumerate(idxs,1):
        #     rec= m.reconstruct(idx)
            rec = dgp.reconstruct(idx)
            plt.subplot(2,5,i)
        #     print(rec.shape)
            plt.imshow(rec.flatten().reshape(28, 28))
            plt.axis("off")
        plt.savefig("mnist_2l_recon.png")

        plt.figure()
        for i, y in enumerate(Y[idxs],1):
            plt.subplot(2,5,i)
            plt.axis("off")
            plt.imshow(y.flatten().reshape(28, 28))

        plt.savefig("mnist_orig.png")
        plt.figure()
        for i, idx in enumerate(idxs, 1):
            plt.subplot(2,5,i)
            plt.axis("off")
            plt.imshow(dgp.means[idx].flatten().reshape(1,5))
        plt.savefig("mnist_repn_2l.png")

In [ ]: plt.figure()
        for i, idx in enumerate(idxs,1):
            rec= m.predict_y(GPLVM_X_mean[idx].reshape(1,5))[0]
            plt.subplot(2,5,i)
            plt.axis("off")
            plt.imshow(rec.flatten().reshape(28, 28))
        plt.savefig("mnist_1l_recon.png")
        plt.figure()
        for i, y in enumerate(Y[idxs],1):
            plt.subplot(2,5,i)
```

```
            plt.imshow(y.flatten().reshape(28, 28))


        plt.figure()
        for i, idx in enumerate(idxs, 1):
            plt.subplot(2,5,i)
            plt.axis("off")
            plt.imshow(GPLVM_X_mean[idx].flatten().reshape(1,5))
        plt.savefig("mnist_repn_1l.png")
```

```
In [ ]: dgp.reconstructon_error(Y)
```

```
In [ ]: test = np.random.randn(dgp.latent_dims[-1]).reshape(1, dgp.latent_dims[-1])
        # for model in reversed(dgp.models):
        #     x_recon = model.predict_y(test)[0]
        #     x_mean = x_recon
        recon = dgp.reconstruct_from_input(test)
        plt.imshow(recon.flatten().reshape(28,28))
```

```
In [ ]: kern = dgp.models[1].kern#.kernels[0]
        sens = np.sqrt(kern.variance.read_value())/kern.lengthscales.read_value()
        print(sens)
        fig, ax = plt.subplots()
        dims = np.arange(len(sens))
        ax.bar(dims, sens, 0.2, color='y')
        ax.set_xticks(dims)
        ax.set_xlabel('dimension')
        ax.set_title('Sensitivity to latent inputs');
        plt.savefig("mnist_2l_latent_sens.png")
```

```
In [ ]: # dgp.get_sensitivities(plot = True)
```

```
In [ ]: layers = [1, 2]
        dim_list = [[[5], [10], [20]], [[10, 5], [10, 10], [20, 10]]]
        inducing_pts = [[10, 20, 40], [[20, 20], [20, 10], [10, 10]]]
```

1 frey faces

```
In [ ]: ! wget "http://www.cs.nyu.edu/~roweis/data/frey_rawface.mat" > frey.mat
```

```
In [ ]: ! ls
```

```
In [ ]: from scipy import io as spio
        data = spio.loadmat("frey_rawface.mat")
        faces = data['ff'].T
        faces = faces.astype(np.float32)/255
        faces.shape
```

```
In [ ]: for i in range(100):
            plt.subplot(10,10,i+1)
            plt.axis("off")
            plt.imshow(faces[i, :].reshape(28, 20))

In [ ]: idxs = np.random.randint(0, faces.shape[0], 200)
        subset = faces[idxs, :]

In [ ]: dgp_f = DeepGPLVM(kernel_dims = [5, 3], n_layers = 2, max_iters=50, latent_dims = [10,
        Y = subset.astype(np.float64)
        dgp_f.train(Y)

In [ ]: idxs = np.random.randint(0, Y.shape[0],10)#np.random.permutation(dgp.means)[:10]

        # random_sample = dgp.means[idxs]

        plt.figure()
        for i, idx in enumerate(idxs,1):
        #     rec= m.reconstruct(idx)
            rec = dgp_f.reconstruct(idx)
            plt.subplot(2,5,i)
        #     print(rec.shape)
            plt.axis("off")
            plt.imshow(rec.flatten().reshape(28, 20))
        plt.savefig("frey_recon_2l.png")

        plt.figure()
        for i, y in enumerate(Y[idxs],1):
            plt.subplot(2,5,i)
            plt.axis("off")
            plt.imshow(y.flatten().reshape(28, 20))
        plt.savefig("frey_orig_2l.png")


        plt.figure()
        for i, idx in enumerate(idxs, 1):
            plt.subplot(2,5,i)
            plt.axis("off")
            plt.imshow(dgp_f.means[idx].flatten().reshape(3, 1))
        plt.savefig("frey_2l_rep.png")

In [ ]: dgp_f.reconstructon_error(Y)

In [ ]: steps = np.arange(-3, 3, 0.5)
        for i in range(3):
            plt.figure()
            for j, s in enumerate(steps):

                test = np.zeros((1,3))
```

```
            test[0][i] += s
            recon = dgp_f.reconstruct_from_input(test)
            plt.subplot(1, 12, j + 1 )
            plt.axis("off")
            plt.imshow(recon.reshape(28, 20))
            plt.savefig("frey_feature_{0}_trend_2l.png".format(i))
```

In [ ]:
```
dgp_f3 = DeepGPLVM(kernel_dims = [5, 5, 3], n_layers = 3, max_iters=50, latent_dims =
Y = subset.astype(np.float64)
dgp_f3.train(Y)
```

In [ ]:
```
idxs = np.random.randint(0, Y.shape[0],10)#np.random.permutation(dgp.means)[:10]

# random_sample = dgp.means[idxs]

plt.figure()
for i, idx in enumerate(idxs,1):
#     rec= m.reconstruct(idx)
    rec = dgp_f.reconstruct(idx)
    plt.subplot(2,5,i)
#     print(rec.shape)
    plt.axis("off")
    plt.imshow(rec.flatten().reshape(28, 20))
plt.savefig("frey_3l_recon.png")
plt.figure()
for i, y in enumerate(Y[idxs],1):
    plt.subplot(2,5,i)
    plt.axis("off")
    plt.imshow(y.flatten().reshape(28, 20))

plt.savefig("frey_orig_3l.png")
plt.figure()
for i, idx in enumerate(idxs, 1):
    plt.subplot(2,5,i)
    plt.axis("off")
    plt.imshow(dgp_f.means[idx].flatten().reshape(3, 1))
plt.savefig("frey_3l_rep.png")
```

In [ ]:
```
steps = np.arange(-3, 3, 0.5)
for i in range(3):
    plt.figure()
    for j, s in enumerate(steps):

        test = np.zeros((1,3))
        test[0][i] += s
        recon = dgp_f.reconstruct_from_input(test)
        plt.subplot(1, 12, j + 1 )
        plt.axis("off")
```

```
            plt.imshow(recon.reshape(28, 20))
            plt.savefig("frey_feature_{0}_trend_3l.png".format(i))
```

```
In [ ]: dgp2 = DeepGPLVM(kernel_dims = [10,5], n_layers = 2, max_iters=100, latent_dims = [10,
        Y = subset.astype(np.float64)
        dgp2.train(Y)
        dgp2.reconstructon_error(Y)
```

```
In [ ]: # random_sample = dgp.means[idxs]

        plt.figure()
        for i, idx in enumerate(idxs,1):
        #     rec= m.reconstruct(idx)
            rec = dgp2.reconstruct(idx)
            plt.subplot(2,5,i)
        #     print(rec.shape)
            plt.imshow(rec.flatten().reshape(28, 20))

        plt.figure()
        for i, y in enumerate(Y[idxs],1):
            plt.subplot(2,5,i)
            plt.imshow(y.flatten().reshape(28, 20))


        plt.figure()
        for i, idx in enumerate(idxs, 1):
            plt.subplot(2,5,i)
            plt.imshow(dgp2.means[idx].flatten().reshape(5, 1))
```

```
In [ ]: steps = np.arange(-3, 3, 0.5)
        for i in range(5):
            plt.figure()
            for j, s in enumerate(steps):

                test = np.zeros((1,5))
                test[0][i] += s
                recon = dgp2.reconstruct_from_input(test)
                plt.subplot(1, 12, j + 1 )
                plt.axis("off")
                plt.imshow(recon.reshape(28, 20))
```

1.0.1 Downstream classification accuracy comparison (MNIST)

```
In [ ]: XPCA = gpflow.models.PCA_reduce(Y, 5)
        results = []
        for i in tqdm(range(500)):
            trn_idxs, test_idxs = get_idxs(x_5.shape[0])
            a1 = classifier_on_data(x_5, l_5, trn_idxs, test_idxs)
```

```
        a2 = classifier_on_data(XPCA, l_5, trn_idxs, test_idxs)
        a3 = classifier_on_data(GPLVM_X_mean, l_5, trn_idxs, test_idxs)
        a4 = classifier_on_data(dgp.means, l_5, trn_idxs, test_idxs)

        results.append([a1, a2, a3, a4])

    results = np.array(results)
```

In [ ]:
```
_ = plt.boxplot(results[:, :])
plt.xticks([1,2,3, 4], ['data', 'PCA', 'GPLVM', 'DGPLVM-2'])
plt.xlabel('Representation')
plt.ylabel('accuracy')
plt.savefig("boxplot_mnist.png")
```

1.0.2 Classification accuracy on oil flow dataset

In [ ]:
```
dgp_oil = DeepGPLVM(kernel_dims = [10,5], n_layers = 2, max_iters=2000, latent_dims =
dgp_oil.train(Y)
```

In [ ]:
```
XPCAplot = gpflow.models.PCA_reduce(Y, 2)
GPLVM_X_mean = m.X_mean.read_value()

f, ax = plt.subplots(1,2, figsize=(10,6))
colors = cm.rainbow(np.linspace(0, 1, len(np.unique(labels))))

for i, c in zip(np.unique(labels), colors):
    ax[0].scatter(XPCAplot[labels==i, 0], XPCAplot[labels==i, 1], color=c, label=i)
    ax[0].set_title('PCA')
    ax[1].scatter(dgp_oil.means[labels==i, dim1], dgp_oil.means[labels==i, dim2], colo
    ax[1].set_title('Bayesian GPLVM')
plt.savefig("oil_pca_rep_2l.png")
```

In [ ]:
```
XPCA = gpflow.models.PCA_reduce(Y, 5)
from tqdm import tqdm
results = []
labels = np.array(labels)
for i in tqdm(range(100)):
    trn_idxs, test_idxs = get_idxs(Y.shape[0])
    a1 = classifier_on_data(Y, labels, trn_idxs, test_idxs)
    a2 = classifier_on_data(XPCA, labels, trn_idxs, test_idxs)
    a3 = classifier_on_data(GPLVM_X_mean, labels, trn_idxs, test_idxs)
    a4 = classifier_on_data(dgp_oil.means, labels, trn_idxs, test_idxs)

    results.append([a1, a2, a3, a4])

results = np.array(results)
```

In [ ]:
```
_ = plt.boxplot(results[:, :-1])
plt.xticks([1,2,3], ['data', 'PCA', 'GPLVM'])
```

```
plt.xlabel('Representation')
plt.ylabel('accuracy')
plt.savefig("boxplot_oil.png")
```