COMS30007 - Machine Learning

Unsupervised Learning

Carl Henrik Ek

Week 5

Abstract

In this lab we will again look at the modelling proceedure that we have gone through over the last couple of weeks. However we will do this in a slightly different setting doing something often referred to as *unsupervised learning*. Personally I think this is a bit of a misnomer as it is easy to think that unsupervised means that we do not place any demands on the learning proceedure. This is not the case at all, if it was it wouldn't be possible to learn. Let us take the example of training a dog, supervised learning would be when you throw a ball and you want the dog to figure out that it should fetch it, while unsupervised learning is when you just let the dog out of the house and it can do a little bit as it wants. However, importantly, you still have demands on the behaviour of the dog, just because you let it out of the house and you haven't given it any explicit instructions it doesn't mean it is now OK for the dog to eat your cat. So even though the instructions and demands are more subtle they are still there, otherwise the exercise is pointless.

The aim of unsupervised learning is to observe a set of data $\mathbf{Y}$ and make an assumption that this data have been generated from a latent representation $\mathbf{X}$ through some mapping $f(\cdot)$ such that,

$$\mathbf{y}_i = f(\mathbf{x}_i). \tag{1}$$

Now our task is to recover both the latent representation and the mapping. This seems like a really odd problem at first. A very simple solution to it is to say that the mapping is the identity meaning that the latent and the observed data is the same. So we have to add some form of supervision, some form of direction that tells us what it is that we are looking for. One very general scenario is that we might look for a lower dimensional representation of the data. This is often beneficial as many algorithms scale very badly with dimension so if we can reduce the dimensionality of $\mathbf{Y}$ from say $\mathbb{R}^D$ to $\mathbb{R}^Q$ if $Q < D$ then we can apply our expensive algorithm on the latent representation instead. But there might be lots of other demands as well where for one reason or the other we wish to reorganise our data in a different manner than it was presented.

The important thing is that the problem that we are trying to solve is badly constrained, there are clearly so many possible solutions so that we need to somehow encode our preference towards certain solutions. Now here comes a slightly different interpretations of beliefs namely preference. They actually play a very similar role as what happened during learning in the previous scenarios we worked on was that given two equal solutions under the data we prefered the one we believed in more, now that is just the same as a preference. So another way to think of priors is as preferences over the different hypothesis.

We are going to use the same regression model in this task as the previous two labs, we will have the same likelihood function as we have used for both previous regression tasks. We will then use two different types of priors for the mapping, both the linear and the Gaussian process prior to recover different representations. As we are now starting to build a bit more complicated models several of the computations can no longer be done in closed form. Later in the unit we will see how we can adress these computations in a principled manner. For now you will just have to trust me on some of these. We will use the same Gaussian likelihood as we have used in the last two labs,

$$p(\mathbf{Y}|\mathbf{F}) = \mathcal{N}(\mathbf{F}, \beta^{-1}\mathbf{I}). \tag{2}$$

1 Principle Component Analysis

We will start with a model where we use a linear model for our mapping from the latent to the observed space. We will use the same model as we did in the linear regression case with the only difference being that as we do not know the input we are going to specify a preference/belief over this. Note that throughout this derivation I have switched back to Carl notation rather than the Bishop one that was used during the lecture. First we have a likelihood function which describes how likely observations is under our model,

$$p(\mathbf{Y}|\mathbf{W},\mathbf{X}) = \mathcal{N}(\mathbf{X}\mathbf{W}+\mu, \beta^{-1}\mathbf{I}), \tag{3}$$

where μ is a constant offset. We will then use a Gaussian prior over both the weights $\mathbf{W}$ and the latent coordinates $\mathbf{X}$. This leads to the following joint distribution,

$$p(\mathbf{Y}\mathbf{W},\mathbf{X}) = p(\mathbf{Y}|\mathbf{W},\mathbf{X})p(\mathbf{X})p(\mathbf{W}). \tag{4}$$

Our aim is now to derive the posterior distribution over the unknown parameters $\mathbf{W}$ and $\mathbf{X}$. To do so we need to marginalise out these parameters from the joint defined above. This means we need to compute,

$$p(\mathbf{Y}) = \int p(\mathbf{Y}|\mathbf{W},\mathbf{X})p(\mathbf{X})p(\mathbf{W}). \tag{5}$$

Sadly the following integral is not tractable, we cannot solve it for both $\mathbf{W}$ and $\mathbf{X}$. In order to proceed we are going to integrate out one of the variables and take a point estimate for the other. Now we could either choose $\mathbf{W}$ or $\mathbf{X}$ to optimise for so which one should we choose? Well lets say that we have N data points and we are looking for a Q dimensional latent representation. This means that $\mathbf{W} \in \mathbb{R}^{D\times Q}$ and $\mathbf{X}^{N\times Q}$ assuming that $N > D$ it makes sense to treat as many of the random variables with principle as we can and integrate out $\mathbf{X}$ while we optimise $\mathbf{W}$. Another motivation is that if we are given new data and we want to infer its latent location then we want to have the posterior distribution $p(\mathbf{x}|\mathbf{y})$ and we can only reach this distribution if we marginalise out the latent space.

We are going to do the computations in a bit of a different way compared to before. First we are going to make use of the fact that the product of two Gaussians is also a Gaussian. So if we multiply our likelihood and our prior we should now again have a Gaussian,

$$p(\mathbf{X},\mathbf{Y}|\mathbf{W}) = p(\mathbf{Y}|\mathbf{W},\mathbf{X})p(\mathbf{X}). \tag{6}$$

Now we know both of the terms on the left hand-side of the expression,

$$\begin{aligned} p(\mathbf{Y}|\mathbf{W},\mathbf{X}) &= \mathcal{N}(\mathbf{X}\mathbf{W}+\mu, \beta^{-1}\mathbf{I}) & (7)\\ p(\mathbf{X}) &= \mathcal{N}(\mathbf{0},\mathbf{I}). & (8)\end{aligned}$$

We will now derive the joint distribution of a pair of points, $\mathbf{z}$ and $\mathbf{x}$ rather than the full data matrix. As we know that the product is a Gaussian we can write up the definition of the terms in this joint distribution,

$$p(\mathbf{y},\mathbf{x}|\mathbf{W}) = \mathcal{N}\left(\begin{bmatrix} \mathbb{E}[\mathbf{y}] \\ \mathbb{E}[\mathbf{x}] \end{bmatrix}, \begin{bmatrix} \mathbb{E}[(\mathbf{y}-\mathbb{E}[\mathbf{y}])(\mathbf{y}-\mathbb{E}[\mathbf{y}])^{\mathrm{T}}] & \mathbb{E}[(\mathbf{y}-\mathbb{E}[\mathbf{y}])(\mathbf{x}-\mathbb{E}[\mathbf{x}])^{\mathrm{T}}] \\ \mathbb{E}[(\mathbf{x}-\mathbb{E}[\mathbf{x}])(\mathbf{y}-\mathbb{E}[\mathbf{y}])^{\mathrm{T}}] & \mathbb{E}[(\mathbf{x}-\mathbb{E}[\mathbf{x}])(\mathbf{x}-\mathbb{E}[\mathbf{x}])^{\mathrm{T}}] \end{bmatrix}\right). \tag{9}$$

We will now walk through each of the expectations in turn to specify the joint distribution above. Once we have the joint we can use the by know well know identities to get the marginal and the posterior distribution that we are looking for. Let us start with the means,

$$\begin{aligned} \mathbb{E}[\mathbf{x}] &= \mathbf{0} & (10)\\ \mathbb{E}[\mathbf{y}] &= \mathbb{E}[\mathbf{W}\mathbf{x}+\mu+\epsilon] = \mathbb{E}[\mathbf{W}\mathbf{x}] + \mathbb{E}[\mu] + \mathbb{E}[\epsilon] & (11)\\ &= \mathbf{W}\mathbb{E}[\mathbf{x}] + \mathbb{E}[\mu] + \mathbb{E}[\epsilon] = \mathbf{0}\mathbf{W} + \mu + \mathbf{0} & (12)\\ &= \mu & (13)\end{aligned}$$

The mean of the latent variable is easy as it come directly through its definition. The one for the observed data comes through the model of the observed data so we first need to rewrite it in terms of the model. Now when we have the means we can move on to deriving the elements of the covariance matrix,

$$\begin{aligned}
\mathbb{E}[(\mathbf{x}-\mathbb{E}[\mathbf{x}])(\mathbf{y}-\mathbb{E}[\mathbf{y}])^{\mathrm{T}}] &= \mathbb{E}[(\mathbf{x}-0)(\mathbf{y}-\mu)^{\mathrm{T}}] && (14)\\
&= \mathbb{E}[\mathbf{x}(\mathbf{W}\mathbf{x}+\mu+\epsilon-\mu)^{\mathrm{T}}] && (15)\\
&= \mathbb{E}[\mathbf{x}(\mathbf{W}\mathbf{x}+\epsilon)^{\mathrm{T}}] = \mathbb{E}[\mathbf{x}\left(\mathbf{W}\mathbf{x}\right)^{\mathrm{T}}+\mathbf{x}\epsilon^{\mathrm{T}}] && (16)\\
&= \mathbb{E}[\mathbf{x}\mathbf{x}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}]+\mathbb{E}[\mathbf{x}]\mathbb{E}[\epsilon] = \mathbb{E}[(\mathbf{x}-\mathbf{0})(\mathbf{x}-\mathbf{0})^{\mathrm{T}}]\mathbf{W}^{\mathrm{T}}+0\cdot 0 && (17)\\
&= \mathbf{I}\mathbf{W}^{\mathrm{T}} = \mathbf{W}^{\mathrm{T}}. && (18)
\end{aligned}$$

We can then derive the final element of the joint distribution the variance of the observed data,

$$\begin{aligned}
\mathbb{E}[(\mathbf{y}-\mathbb{E}[\mathbf{y}])(\mathbf{y}-\mathbb{E}[\mathbf{y}])^{\mathrm{T}}] &= \mathbb{E}[(\mathbf{W}\mathbf{x}+\mu+\epsilon-\mu)(\mathbf{W}\mathbf{x}+\mu+\epsilon-\mu)^{\mathrm{T}}] && (19)\\
&= \mathbb{E}[(\mathbf{W}\mathbf{x}+\epsilon)(\mathbf{W}\mathbf{x}+\epsilon)^{\mathrm{T}}] = \mathbb{E}[\mathbf{W}\mathbf{x}\left(\mathbf{W}\mathbf{x}\right)^{\mathrm{T}}+\mathbf{W}\mathbf{x}\epsilon^{\mathrm{T}}+\epsilon\mathbf{W}\mathbf{x}^{\mathrm{T}}+\epsilon\epsilon^{\mathrm{T}}] && (20)\\
&= \mathbb{E}[\mathbf{W}\mathbf{x}\mathbf{x}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}]+\mathbb{E}[\mathbf{W}\mathbf{x}\epsilon^{\mathrm{T}}]+\mathbb{E}[\epsilon(\mathbf{W}\mathbf{x})^{\mathrm{T}}]+\mathbb{E}[\epsilon\epsilon^{\mathrm{T}}] && (21)\\
&= \mathbf{W}\mathbb{E}[\mathbf{x}\mathbf{x}^{\mathrm{T}}]\mathbf{W}^{\mathrm{T}}+\mathbf{W}\mathbb{E}[\mathbf{x}]\mathbb{E}[\epsilon]+\mathbb{E}[\epsilon]\mathbb{E}[\mathbf{x}^{\mathrm{T}}]\mathbf{W}^{\mathrm{T}}+\mathbb{E}[(\epsilon-0)(\epsilon-0)^{\mathrm{T}}] && (22)\\
&= \mathbf{W}\mathbf{I}\mathbf{W}^{\mathrm{T}}+\mathbf{W}0+0\mathbf{W}^{\mathrm{T}}+\sigma^2\mathbf{I} && (23)\\
&= \mathbf{W}\mathbf{W}^{\mathrm{T}}+\sigma^2\mathbf{I}. && (24)
\end{aligned}$$

Now we have all the elements and can write up the final joint distribution as,

$$p(\mathbf{y},\mathbf{x}|\mathbf{W}) = \mathcal{N}\left(\begin{bmatrix}\mu\\ \mathbf{0}\end{bmatrix},\begin{bmatrix}\mathbf{W}\mathbf{W}^{\mathrm{T}}+\sigma^2\mathbf{I} & \mathbf{W}\\ \mathbf{W}^{\mathrm{T}} & \mathbf{I}\end{bmatrix}\right), \tag{25}$$

from which we can derive the marginal over the observed data and the conditional distribution over the latent space using the Gaussian identities as,

$$\begin{aligned}
p(\mathbf{y}|\mathbf{W}) &= \mathcal{N}(\mu,\mathbf{W}\mathbf{W}^{\mathrm{T}}+\sigma^2\mathbf{I}) && (26)\\
p(\mathbf{x}|\mathbf{y},\mathbf{W}) &= \mathcal{N}(\mathbf{W}^{\mathrm{T}}\left(\mathbf{W}\mathbf{W}^{\mathrm{T}}+\sigma^2\mathbf{I}\right)^{-1}(\mathbf{y}-\mu),\mathbf{I}-\mathbf{W}^{\mathrm{T}}(\mathbf{W}\mathbf{W}^{\mathrm{T}}+\sigma^2\mathbf{I})^{-1}\mathbf{W}). && (27)
\end{aligned}$$

Now we have everything that we need in order to proceed. Our first goal is to take a point estimate of the weights of the mapping. Now there are several ways that we can do this, one option would be to take the derivatives of the marginal distribution and maximise this for the data. However, in this case it turns out that there is actually a closed form solution for the weights in the limit when $\beta \rightarrow \infty$. We will cheat a little bit and use this as a solution. We can get this by solving a simple eigenvalue problem using the code below.

Code

```python
def MLW(Y,q):
    v,W = np.linalg.eig(np.cov(Y.T))
    idx = np.argsort(np.real(v))[::-1][:q]
    return np.real(W[:,idx])
```

Once we have the $\mathbf{W}$ that maximises the marginal we can use this in the posterior distribution and recover the latent space. Now we have everything that we need and it is time to generate some data and test the approach. We will generate a simple spiral in two dimensions and then we will draw a random mapping that embedds this in $\mathbb{R}^{101}$.

[1]the code below is a bit messy as I had to save and load the data, so you have to do more than just copy paste the code

Code

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from tempfile import TemporaryFile

# maximum likelihood solution to W
def MLW(x,q):
    v,w = np.linalg.eig(np.cov(x.T))
    idx = np.argsort(np.real(v))[::-1][:q]
    return np.real(w[:,idx])

# posterior distribution of latent variable
def posterior(w, x, mu_x, beta):
    A = np.linalg.inv(w.dot(w.T)+1/beta*np.eye(w.shape[0]))
    mu = w.T.dot(A.dot(x-mu_x))
    varSigma = np.eye(w.shape[1]) - w.T.dot(A.dot(w))

    return mu, varSigma

# Generate a spiral
t = np.linspace(0,3*np.pi,100)
x = np.zeros((t.shape[0],2))
x[:,0] = t*np.sin(t)
x[:,1] = t*np.cos(t)

# pick a random matrix that maps to Y
w = np.random.randn(10,2)
y = x.dot(w.T)
y += np.random.randn(y.shape[0],y.shape[1])
mu_y = np.mean(y,axis=0)

# get maximim likelihood solution of W
w = MLW(y,2)

# compute predictions for latent space
xpred = np.zeros(x.shape)
varSigma = []
for i in range(0, y.shape[0]):
    xpred[i,:], varSigma = posterior(w, y[i,:], mu_y, 1/2)

np.save('05tmp1.npy', xpred)
np.save('05tmp2.npy', varSigma)
```

When we have got the code working to describe the joint distribution we can now use the posterior and map back all the training data-points to the latent space and look at how their distribution looks like.

Code

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

xpred = np.load('05tmp1.npy')
varSigma = np.load('05tmp2.npy')

# generate density
N = 300
x1 = np.linspace(np.min(xpred[:,0]), np.max(xpred[:,0]), N)
x2 = np.linspace(np.min(xpred[:,1]), np.max(xpred[:,1]), N)
x1p, x2p = np.meshgrid(x1, x2)
pos = np.vstack((x1p.flatten(), x2p.flatten())).T

# compute posterior
Z = np.zeros((N,N))
for i in range(0, xpred.shape[0]):
    pdf = multivariate_normal(xpred[i,:].flatten(), varSigma)
    Z += pdf.pdf(pos).reshape(N,N)

# plot figuresy
fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot(121)
ax.scatter(xpred[:,0], xpred[:,1])
ax.set_xticks([])
ax.set_yticks([])
ax = fig.add_subplot(122)
ax.imshow(Z,cmap='hot')
ax.set_ylim(ax.get_ylim()[::-1])
ax.set_xticks([])
ax.set_yticks([])

#IGNORE THIS
plt.tight_layout()
plt.savefig(path, transparent=True)
return path
```

Now as you can see the data looks similar to what we expected, we do recover the spiral as we expected. Now as it turns out we could have actually gotten any rotation of the spiral as the marginal distribution over $\mathbf{W}$ is actually invariant to a rotation. To see this, you can rotate the matrix,

$$\tilde{\mathbf{W}} = \mathbf{W}\mathbf{R} \quad (28)$$

$$\tilde{\mathbf{W}}\tilde{\mathbf{W}}^{\mathrm{T}} = \mathbf{W}\mathbf{R}(\mathbf{W}\mathbf{R})^{\mathrm{T}} \quad (29)$$

$$\mathbf{W}\underbrace{\mathbf{R}\mathbf{R}^{\mathrm{T}}}_{\mathbf{I}}\mathbf{W}^{\mathrm{T}} = \mathbf{W}\mathbf{W}^{\mathrm{T}}. \quad (30)$$

Because a rotation matrix is an orthonormal matrix $\mathbf{R}^{\mathrm{T}} = \mathbf{R}^{-1}$. Later in the unit when we have seen approximate inference we can go back to this model and solve it in a more principled manner. If you already now want to understand in more detail how you solve this in a more principled manner you can read [1] Chapter 12.2.

A different way to solve the maximisation is to take a gradient based approach. This means that we will

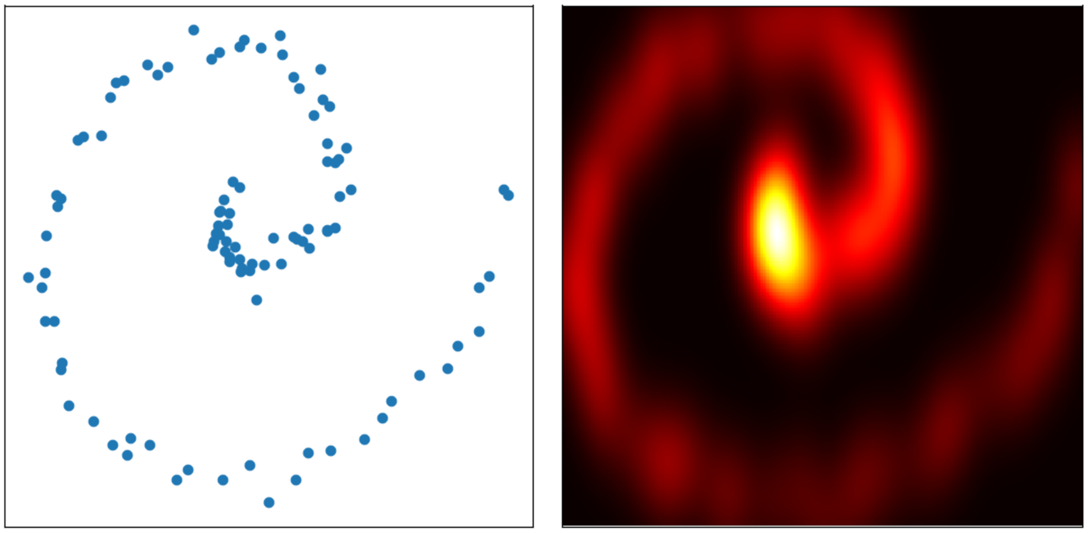

Figure 1: The figure above shows the PCA solution to a 2D sprial embedded in $\mathbb{R}^{10}$. The right plot shows the posterior distribution of each individual point in the training data-set super-imposed on each other.

maximise the marginal distribtuion $p(\mathbf{Y}|\mathbf{W})$ using an iterative gradient based method.

2 Gaussian Process Latent Variable Model

We can also use a GP prior over the mapping f. If we do this it lead to a model called the Gaussian Process Latent Variable Model [2]. This is a really powerful model that has been used in a large range of different settings such as animantion [3], pose estimation [4] and reinforcement learning [5]. Using a GP changes the model slightly as our prior now need to be indexed by the locations that we want to evaluate it at. This means that we have the following joint distribution,

$$p(\mathbf{Y}, \mathbf{X}, \mathbf{F}, \theta) = p(\mathbf{Y}|\mathbf{F})p(\mathbf{F}|\mathbf{X}, \theta)p(\mathbf{X})p(\theta). \tag{31}$$

Now we will do a couple of assumptions to make things a little bit easier. First we will say that given the input the output dimensions are independent. This might sounds strange at first, but think about it again, what it really says is that if I know the input location where I want to query the output of the function the input encodes all the information necessary to decide on the output. This means that we can factorise the prior as this,

$$p(\mathbf{F}|\mathbf{X}, \theta) = \prod_{i=d}^{D} p(\mathbf{f}_d|\mathbf{X}, \theta). \tag{32}$$

The noise assumption that we have done for the likelihood also implies that given the output of the function the data points and the dimensions are independent,

$$p(\mathbf{Y}|\mathbf{F}) = \prod_{i=1}^{N}\prod_{d=1}^{D} p(y_{id}|f_{id}) = \prod_{i=1}^{N}\prod_{d=1}^{D} \mathcal{N}(y_{id}|f_{id}, \beta^{-1}). \tag{33}$$

Now what we want to infer is the latent locations $\mathbf{X}$ and the parameters of the GP namely the parameters of the covariance function θ. This means that we need to solve the following integral,

$$p(\mathbf{Y}) = \int p(\mathbf{Y}, \mathbf{X}, \mathbf{F}, \theta)\mathrm{d}\mathbf{X}\mathrm{d}\mathbf{F}\mathrm{d}\theta = \int p(\mathbf{Y}|\mathbf{F})p(\mathbf{F}|\mathbf{X}, \theta)p(\mathbf{X})p(\theta)\mathrm{d}\mathbf{X}\mathrm{d}\mathbf{F}\mathrm{d}\theta. \tag{34}$$

Sadly it is intractable for us to compute this in closed form. What we can do is integrate out $\mathbf{F}$ but $\mathbf{X}$ and θ are both intractable. In order to proceed we are going to compute the integral over $\mathbf{F}$ first and then we are going to find a maximum-likelihood solution for $\mathbf{X}$. First we can intergrate out the actual function values leading to the marginal likelihood,

$$p(\mathbf{Y}|\mathbf{X},\theta) = \frac{1}{(2\pi)^{\frac{DN}{2}}|K|^{\frac{D}{2}}} e^{-\frac{1}{2}\mathrm{tr}(\mathbf{Y}\mathbf{K}^{-1}\mathbf{Y}^{\mathrm{T}})} \tag{35}$$

Now when we have this distribution we are going to find the latent locations $\mathbf{X}$ that maximises this. This means that we should solve the following optimisation problem,

$$\hat{\mathbf{X}} = \mathrm{argmax}_{\mathbf{X},\theta} p(\mathbf{Y}|\mathbf{X},\theta) = \mathrm{argmin}_{\mathbf{X},\theta} - \log p(\mathbf{Y}|\mathbf{X},\theta) = \mathcal{L}(\mathbf{X}). \tag{36}$$

As no closed form can be found of the following we have to solve this using gradient based methods. Doing so is quite a challenge while we leave this for extra work. The important thing is that you see how similar the two models are.

2.1 Implementation

Now sadly the implementation of the GP-LVM is a bit more tedious to do and we need to think quite a bit more about how to compute things both in more clever ways but also how to implement things in a stable way. Therefore the implementation goes beyond the scope of this lab but I've left the derivation here if you are interested in having a look at this.

Computing the gradients is rather tedious affair so we went ahead and did this for you. If you want to compute them on your own you can use the excellent matrix cookbook [6] or if you are really lazy you can use an automatic differentiation package such as JAX that is capable of computing gradients automatically from `numpy` code[2]. Computing gradients of expressions involving matrices are a little bit more involved compared to scalar operators. Effectively what we can think is that we can compute derivatives of a scalar with a matrix and a matrix with a scalar but a matrix with a matrix is not easily defined. You do **not** need to understand or be able to perform the commputations, they are simply here so that you see that it is not magic and so that you can do them if you want. Also, there might be errors in the derivation so if you spot something strange let me know. In order to work out the gradients I've made heavy use of the matrix cookbook [6]. You can see the gradient calculations in Section 4. Once we have the gradients we can now use a gradient based optimiser that is implemented in `scipy.optimise`. What we need to provide the optimiser with is a function that returns the objective value for a specific parameter setting and a function that returns the gradients. If we have a function $f(\cdot)$ that takes $\mathbf{x}$ as an input we need to implement a structure such as this,

Code

```python
import numpy as np
import scipy as sp
import scipy.optimize as opt

def f(x, *args):
    # return the value of the objective at x
    return val

def dfx(x,*args):
    # return the gradient of the objective at x
    return val

x_star = opt.fmin_cg(f,x0,fprime=dfx, args=args)
```

[2]we will have a look at automatic differentiation in the last optional lab of the unit

The function `f(x, *args)` is the objective function $\mathcal{L}(\mathbf{X})$ and `dfx(x, *args)` is the gradients of the objective. We can use `*args` to pass arguments to the function that we do not optimise. By calling `opt.fmin_cg` the gradients and the objective will be called by the mechanism that is implemented in the optimiser, in order for this to work `x` always needs to be in the form of a vector. This means that we when `f` is called the input is a vector and we need to reshape it to the correct size in order to calculate the objective functions value, same thing is true for the gradient, the returned object needs to be a vector.

3 Summary

This lab ends the first part of the unit where we have been building models. Think about what you have done, you have created parametrised probabilities using hierarchical distributions. You have tried to describe the generative process of the data using the tools that you have. Once we have done this we have formulated our beliefs in prior distributions, the random variables on top of the chain. Then we used the tools of probability, the sum and the product rule to try and get the conditional distributions of interest so that we could extract the information that we wanted from data. Up till this lab the computations could be done in closed form but as you could see now the more complicated the model of the data becomes the more challenging the computations have become. In this lab we had to circumvent this problem by throwing principle out the window and perform maximum likelihood estimation. This is in general a dangerous path to walk as we no longer have uncertainty in the estimate, simply because we do not have any beliefs in them. Toward the end of the unit we will see how we can perform approximate computation in order to circumvent the intractability of these models. But till then we just have to feel a little bit bad about having done a couple of point estimates.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] Neil D Lawrence. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research*, 6:1783–1816, 2005.

[3] Keith Grochow, Steven L Martin, Aaron Hertzmann, and Zoran Popović. Style-based inverse kinematics. *SIGGRAPH '04: SIGGRAPH 2004 Papers*, August 2004.

[4] Raquel Urtasun, David J Fleet, and P. Fua. 3D people tracking with Gaussian process dynamical models. *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, 1:238–245, 2006.

[5] Steindór Sæmundsson, Katja Hofmann, and Marc Peter Deisenroth. Meta reinforcement learning with latent variable gaussian processes. *CoRR*, 2018.

[6] K. B. Petersen and M. S. Pedersen. The Matrix Cookbook, November 2012. Version 20121115.

4 Gradients Calculations

In this section we will derive the gradients that we can use to find the maximum of the marginal likelihood. The derivations for the linear and for the non-linear case are very similar so the derivation will start by first deriving the linear solution and then from there we will derive the ones for the non-linear case. Remember that the only difference between the two objectives is that for one we integrate out the latent space and optimise the parameters of the mapping while in the other we do the opposite and optimise the latent representation directly.

The first thing that we need to do is to write up the objective function, what we want to do is to find,

$$\hat{\mathbf{W}} = \text{argmax}_{\mathbf{W}} p(\mathbf{Y}|\mathbf{W}). \tag{37}$$

Because we are only interested in the location of the optima not the value of the optima we can transform the objective using any monotonic function. As the probability contains an exponential function we therefore take the logarithm of the expression. Furthermore, it is common practice to minimise rather than maximise, we will therefore seek the minima of the negative logarithm of the objective as,

$$\hat{\mathbf{W}} = \text{argmin}_{\mathbf{W}} - \log\left(p(\mathbf{Y}|\mathbf{W})\right) = \mathcal{L}(\mathbf{W}). \tag{38}$$

If we write up the marginal likelihood of all the data $\mathbf{Y}$ we get something like this,

$$p(\mathbf{Y}|\mathbf{W}) = \prod_{i=1}^{N} p(\mathbf{y}_i|\mathbf{W}) = \prod_{i=1}^{N} \frac{1}{(2\pi)^{\frac{D}{2}}|\mathbf{C}(\mathbf{W})|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{y}_i^{\mathrm{T}}(\mathbf{C}(\mathbf{W}))^{-1}\mathbf{y}_i)} \tag{39}$$

$$= \frac{1}{(2\pi)^{\frac{DN}{2}}|\mathbf{C}(\mathbf{W})|^{\frac{D}{2}}} e^{-\frac{1}{2}\sum_{i=1}^{N}(\mathbf{y}_i^{\mathrm{T}}(\mathbf{C}(\mathbf{W}))^{-1}\mathbf{y}_i)}. \tag{40}$$

this leads to an objective function consisting of three terms which looks like this where we have multiplied both therm with two,

$$\mathcal{L}(\mathbf{W}) = \text{constant} + D\log|\mathbf{C}(\mathbf{W})| + \sum_{i}^{N} \mathbf{y}_i^{\mathrm{T}}(\mathbf{C}(\mathbf{W}))^{-1}\mathbf{y}_i \tag{41}$$

Now we need to compute the derivatives of $\mathbf{C}(\mathbf{W})$ with respect to $\mathbf{W}$. To do this lets first rewrite everything on matrix form so that we can remove the sum from the objective function. First note that,

$$\sum_i \mathbf{x}_i\mathbf{x}_i = \text{tr}\left(\begin{bmatrix} \leftarrow & \mathbf{x}_1 & \rightarrow \\ \leftarrow & \mathbf{x}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{x}_N & \rightarrow \end{bmatrix}\begin{bmatrix} \leftarrow & \mathbf{x}_1 & \rightarrow \\ \leftarrow & \mathbf{x}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{x}_N & \rightarrow \end{bmatrix}^{\mathrm{T}}\right), \tag{42}$$

this means that we can rewrite the objective function as,

$$\mathcal{L}(\mathbf{W}) = \text{constant} + D\log|\mathbf{C}(\mathbf{W})| + \text{tr}\left(\mathbf{Y}(\mathbf{C}(\mathbf{W}))^{-1}\mathbf{Y}^{\mathrm{T}}\right). \tag{43}$$

Now we have two terms we need to take derivatives of, both of these include the same matrix $\mathbf{C}$ in one form of the other so it will come in handy to take this derivative first,

$$\frac{\partial \mathbf{C}}{\partial \mathbf{W}_{ij}} = \frac{\partial \mathbf{W}\mathbf{W}^{\mathrm{T}}}{\partial \mathbf{W}_{ij}}. \tag{44}$$

We will now use the excellent Matrix Cookbook [6] to find the rules that we need to figure this one out. If you use the version from 2012 URL we can first use Eq. 37 to rewrite the the product,

$$\partial(\mathbf{X}\mathbf{Y}) = (\partial\mathbf{X})\mathbf{Y} + \mathbf{X}(\partial\mathbf{Y}). \tag{45}$$

We can then combine this with Eq. 32 which states,

$$\frac{\partial \mathbf{X}_{kl}}{\partial \mathbf{X}_{ij}} = \delta_{ik}\delta_{lj}, \tag{46}$$

where δ_{ij} is the kronecker delta function,

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases} \tag{47}$$

This leads to the following derivative,

$$\frac{\partial \mathbf{W}\mathbf{W}^{\mathrm{T}}}{\partial \mathbf{W}_{ij}} = \mathbf{W}\frac{\partial \mathbf{W}^{\mathrm{T}}}{\partial \mathbf{W}_{ij}} + \frac{\partial \mathbf{W}}{\partial \mathbf{W}_{ij}}\mathbf{W}^{\mathrm{T}} = \mathbf{W}\mathbf{J}_{ij} + \mathbf{J}_{ji}\mathbf{W}^{\mathrm{T}}, \tag{48}$$

where we $\mathbf{J}_{ij}$ is a matrix who has all zero entries except for $(\mathbf{J}_{ij})_{ij} = 1$.

Now lets strart by tackling the first term, the derivative of the log determinant.

$$\frac{\partial}{\partial \mathbf{W}_{ij}} \log|\mathbf{C}| \tag{49}$$

We will start by using Eq. 43 that states that,

$$\partial \log|\mathbf{X}| = \text{tr}\left(\mathbf{X}^{-1}\partial\mathbf{X}\right). \tag{50}$$

We can now rewrite this as,

$$\frac{\partial}{\partial \mathbf{W}_{ij}} \log|\mathbf{C}| = \text{tr}\left(\mathbf{C}^{-1}\frac{\partial \mathbf{C}}{\partial \mathbf{W}_{ij}}\right). \tag{51}$$

So now we have our first term and we can move on to the second. This term is a derivative of a trace we can now use Eq. 36 which states,

$$\partial\left(\text{tr}(\mathbf{X})\right) = \text{tr}\left(\partial\mathbf{X}\right). \tag{52}$$

This means that our second term becomes,

$$\frac{\partial}{\partial \mathbf{W}_{ij}} \text{tr}\left(\mathbf{Y}(\mathbf{C})^{-1}\mathbf{Y}^{\mathrm{T}}\right) = \text{tr}\left(\frac{\partial}{\partial \mathbf{W}_{ij}}\mathbf{Y}(\mathbf{C})^{-1}\mathbf{Y}^{\mathrm{T}}\right). \tag{53}$$

Now we want to break the quadratic form and we will do this by using the chain-rule to do the derivative,

$$\text{tr}\left(\frac{\partial}{\partial \mathbf{W}_{ij}}\mathbf{Y}(\mathbf{C})^{-1}\mathbf{Y}^{\mathrm{T}}\right) = \text{tr}\left(\frac{\partial}{\partial \mathbf{C}^{-1}}(\mathbf{Y}\mathbf{C}^{-1}\mathbf{Y}^{\mathrm{T}})\frac{\partial \mathbf{C}^{-1}}{\partial \mathbf{W}_{ij}}\right) = \text{tr}\left((\mathbf{Y}\mathbf{Y}^{\mathrm{T}})^{\mathrm{T}}\frac{\partial \mathbf{C}^{-1}}{\partial \mathbf{W}_{ij}}\right). \tag{54}$$

Now we have isolated the derivative and we are nearly there. The last rule we will use is to use the derivative of a matrix inverse Eq. 40,

$$\partial \mathbf{X}^{-1} = -\mathbf{X}^{-1}(\partial\mathbf{X})\mathbf{X}^{-1}. \tag{55}$$

This means we can reach the derivative of the last term as,

$$\text{tr}\left((\mathbf{Y}\mathbf{Y}^{\mathrm{T}})^{\mathrm{T}}\frac{\partial \mathbf{C}^{-1}}{\partial \mathbf{W}_{ij}}\right) = \text{tr}\left(\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\left(-\mathbf{C}^{-1}\frac{\partial \mathbf{C}}{\partial \mathbf{W}_{ij}}\mathbf{C}^{-1}\right)\right), \tag{56}$$

where we know the derivative of the inner-most term as we computed it first.

So now we have the full derivative, as we can see the dimensionality makes sense, we take the derivative of our objective function which is a scalar with another scalar and therefore expect a scalar back, as both of our terms are traces of matrices this makes sense. This leads to the full derivative,

$$\frac{\partial}{\partial \mathbf{W}_{ij}}\mathcal{L}(\mathbf{W}) = D\text{tr}\left(\mathbf{C}^{-1}\frac{\partial \mathbf{C}}{\partial \mathbf{W}_{ij}}\right) + \text{tr}\left(\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\left(-\mathbf{C}^{-1}\frac{\partial \mathbf{C}}{\partial \mathbf{W}_{ij}}\mathbf{C}^{-1}\right)\right) \tag{57}$$

$$\frac{\partial \mathbf{C}}{\partial \mathbf{W}_{ij}} = \mathbf{W}\mathbf{J}_{ij} + \mathbf{J}_{ji}\mathbf{W}^{\mathrm{T}} \tag{58}$$

Now we want to change the calculations to the non-linear case. The objective function now stays pretty much the same with only two differences. The first one is the order of the dimensions in the marginal likelihood. In the linear regression case the exponent was,

$$\sum_{i}^{N}(\mathbf{Y}_{i,:}\mathbf{C}(\mathbf{W})^{-1}\mathbf{Y}_{i,:}^{\mathrm{T}}), \tag{59}$$

i.e. the covariance was between the dimensions in the data. Now for the non-parametric case it becomes the other way around, the covariance is between the data-points instead,

$$\sum_{d}^{D}(\mathbf{Y}_{:,d}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{Y}_{:,d}), \tag{60}$$

So writing the objective now instead becomes,

$$\mathcal{L}(\mathbf{X}) = \text{constant} + N\log|\mathbf{K}| + \text{tr}\left(\mathbf{Y}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{Y}\right). \tag{61}$$

The second change is that the inner derivative of the covariance matrix is now a function of the latent variables rather than a weight matrix. This means that we can replace $\mathbf{C}(\mathbf{W})$ with $k(\mathbf{X},\mathbf{X}) = \mathbf{K}$ instead. This means that the only alterations that we need to do is to change the inner derivative, and take the transpose of the data matrix, everything else stays the same. Leading to the following derivative,

$$\frac{\partial}{\partial \mathbf{x}_{id}}\mathcal{L}(\mathbf{X}) = N\text{tr}\left(\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \mathbf{x}_{id}}\right) + \text{tr}\left(\mathbf{Y}\mathbf{Y}^{\mathrm{T}}\left(-\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \mathbf{x}_{id}}\mathbf{K}^{-1}\right)\right). \tag{62}$$

Now lets compute the derivative of the kernel matrix.

$$\frac{\partial \mathbf{K}}{\partial x_{id}} = \frac{\partial}{\partial x_{id}}\begin{bmatrix} k_{11} & k_{12} & \dots & k_{1N} \\ k_{21} & k_{22} & \dots & k_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ k_{N1} & k_{N2} & \dots & k_{NN} \end{bmatrix} \tag{63}$$

$$k_{ij} = k(\mathbf{x}_i,\mathbf{x}_j) = \theta_0 e^{-\frac{1}{\theta_1}(\mathbf{x}_i-\mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i-\mathbf{x}_j)} \tag{64}$$

This means that we can compute the derivative for the kernel as,

$$\begin{aligned}
\frac{\partial k_{ij}}{\partial x_{id}} &= \frac{\partial}{\partial x_{id}}\theta_0 e^{-\frac{1}{\theta_1}(\mathbf{x}_i-\mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i-\mathbf{x}_j)} & (65)\\
&= \theta_0 e^{-\frac{1}{\theta_1}(\mathbf{x}_i-\mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i-\mathbf{x}_j)}\frac{\partial}{\partial x_{id}}\left(-\frac{1}{\theta_1}(\mathbf{x}_i-\mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i-\mathbf{x}_j)\right) & (66)\\
&= k_{ij}\cdot\left(-\frac{1}{\theta_1}\right)\frac{\partial}{\partial x_{id}}(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_i - \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j - \mathbf{x}_j^{\mathrm{T}}\mathbf{x}_i + \mathbf{x}_j^{\mathrm{T}}\mathbf{x}_j) & (67)\\
&= -\frac{1}{\theta_1}k_{ij}\cdot 2(x_{id}-x_{jd}) & (68)
\end{aligned}$$

We can also compute the transpose element of the kernel by flipping the indices which means that,

$$\frac{\partial k_{ji}}{\partial x_{id}} = -\frac{1}{\theta_1}k_{ij}\cdot 2(x_{jd}-x_{id}) = -\frac{\partial k_{ij}}{\partial x_{id}}. \tag{69}$$

All the other elements of the kernel matrix does not contain x_{id} which means that they will be zero. Now we can use the regular structure of the derivative to write everything on matrix form as,

$$\frac{\partial k(\mathbf{X},\mathbf{x}_i)}{\partial x_{id}} = \frac{1}{\theta_1}k(\mathbf{X},\mathbf{x}_i)\odot\begin{bmatrix} x_{1d}-x_{id} \\ \vdots \\ x_{Nd}-x_{id}\end{bmatrix}, \tag{70}$$

where $\odot$ is the Hadamar product defined as,

$$(\mathbf{A}\odot\mathbf{B})_{ij} = (\mathbf{A})_{ij}(\mathbf{B})_{ij}. \tag{71}$$

Now we have the gradients for one column of the kernel matrix and can use this result to get the results also for a row by using the symmetry in the derivatives. With that we have the gradients that we need.