# Adv Machine Learning Sheet 1

Amol Kapoor

May 2017

## 1 HMMs

1. **Derive the log-likelihood function log(X, Z $|\theta$) by introducing a hidden variable z.**

   The probability of a sequence of states $p(Z)$ is the likelihood of the first state, $p(z_1|\pi)$, followed by the likelihood of the next state given the first state based on transition policies, $p(z_n|z_{n-1}, A)$. Note that the likelihood of observing a state is only dependently on the previous state due to the Markov property. The likelihood of a set observations given a set of states, $p(X|Z)$ is given by the weighted likelihood of an observation in a given state, $p(x_n|z_n, \phi)$. By Bayes, these multiplied together is the joint distribution:

   $$p(Z)p(X|Z) = p(X, Z)$$

   Thus:

   $$p(X, Z|\theta) = p(z_1|\pi)p(x_z|z_1) \prod_{n=2} p(z_n|z_{n-1}, A)p(x_n|z_n, \phi) \tag{1}$$

   And following log rules:

   $$\begin{aligned} log(X, Z|\theta) &= log(p(z_1|\pi)p(x_z|z_1) \prod_{n=2} p(z_n|z_{n-1}, A)p(x_n|z_n, \phi)) \\ &= log(p(z_1|\pi)) + log(p(x_z|z_1)) + log(\prod_{n=2} p(z_n|z_{n-1}, A)) + log(p(x_n|z_n, \phi)) \\ &= log(p(z_1|\pi)) + log(p(x_z|z_1)) + \sum_{n=2} log(p(z_n|z_{n-1}, A)) + log(p(x_n|z_n, \phi)) \end{aligned} \tag{2}$$

   Note that theta was dropped from the equations in the paragraph above, but is still a parameter.

2. **Use the general rules of the EM algo to derive the E and M step update rules for the HMM Max Likelihood solution.**

   **E-Step**

   The lower bound of the maximum likelihood is:

$$L(q, \theta) = \mathbf{E} ln(p(X|\theta)) - KL(q||p)$$

   where:

$$KL(q||p) = -\sum_{q(z)} ln(p(Z|X, \theta)/q(Z))$$

   Therefore, to maximize L, need to minimize KL and select q such that:

$$q(Z) = p(Z|X, \theta^{old})$$

   Further:

$$ln(p(X, Z|\theta) = ln(p(Z|X, \theta)) + ln(p(X|\theta))$$

   Therefore:

$$ln(p(Z|X, \theta^{old})) = ln(p(X|\theta)) - ln(p(X, Z|\theta))$$

   where $p(X|\theta)$ is given by the multiplied probabilities $\prod_i p(x_i|\theta)$ and the log likelihood $ln(p(X, Z|\theta)$ is given above in part one.

   **M-Step**

   Given a new value for q, we can maximize L with respect to $\theta$. Because we selected q such that the KL term is zero, finding the max theta is:

$$\theta^t = \underset{\theta}{\text{argmax}} \quad \mathbf{E}[ln(p(X, Z|\theta))]$$

   The maximization can be done with gradients of each parameter - specifically, setting the gradient to zero with respect to the parameter, and solving for the max point.

   Note that in both cases there is an entropy term, but that was dropped as it is constant in both steps.

3. **Research and explain the forward-backward algorithm in the context of HMMs and how this makes the E step more efficient.**

   The forward-backward algorithm is an algorithm used to find the probability of being in a certain state at a time $z_t$ given a set of observations $X$.

The probability can be given as follows:

$$p(z_t = i|X) = \frac{p(z_t, X)}{p(X)}$$

$$\begin{aligned} p(z_t, X) &= p(X|z_t)p(z_t) \\ &= p(x_0...x_t, x_{t+1}...x_T|z_t)p(z_t) \\ &= p(x_0...x_t|z_t)p(x_{t+1}...x_T|z_t)p(z_t) \\ &= p(x_0...x_t, z_t)p(x_{t+1}...x_T|z_t) \\ &= \alpha(z_t)\beta(z_t) \end{aligned} \tag{3}$$

For the forward pass, we calculate the first term, $\alpha$, which roughly translates as the likelihod of observing a sequence $x_0...x_t$ and ending in state $z_t$; in the backward pass, we calculate the second term, $\beta$, which roughly translates as the likelihood of observing a sequence $x_{t+1}...x_T$ starting from state $z_t$. Both can be computed iteratively.

$$\alpha(z_t) = \sum_{z_{t-1}} \alpha(z_{t-1})p(z_t|z_{t-1})p(x_t|z_t)$$

$$\beta(z_t) = \sum_{z_{t+1}} \beta(z_{t+1})p(x_{t+1}|z_{t+1})p(z_{t+1}|z_t)$$

In the context of HMMs, this can be used in the Baum-Welch algorithm to directly find and update the parameters $\theta$. In order to compute $p(Z|X, \theta^{old})$, we would need to know the transition probabilities $A$ and the parameters $\phi$, both which are likely to be unknown. The forwards-backwards algorithm is a way to estimate their values to convergence, providing a means to calculate $p(Z|X, \theta^{old})$.

# 2 HMM Coding

1. **Run the attached script to create some synthetic data and use the HMM package to train a variational HMM with 100 states.** See attached code in aml_ws2.m, and Figure 1.

2. **What's the most likely sequence of states for the training data? How is this related to the identifiability problem?**

   The most likely sequence is 79, 50, 79, 15, 50, 51, 50, 51, 79, given Figure 1.

   The identifiability property applies to a model (or a model is identifiable) if it is possible to learn the true values of the model's parameters after an infinite number of observations. In other words, a model is identifiable if

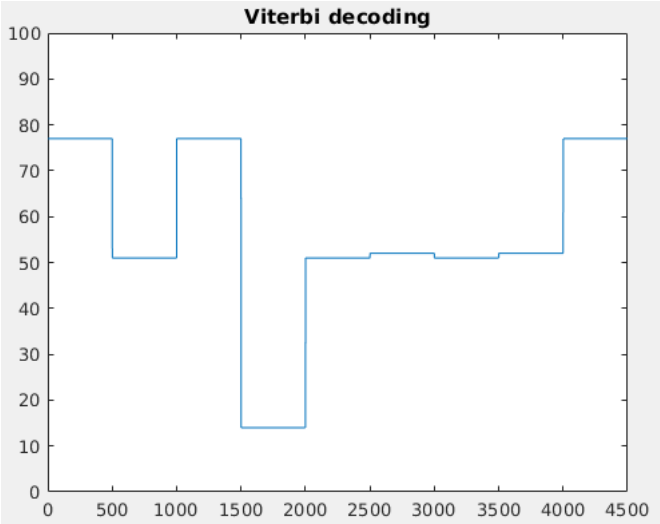


Figure 1: Variational HMM with 100 states.

the input parameters to the model $\theta$ result in a unique output, or is one-to-one. Empirically and analytically the HMM model is not identifiable. Multiple runs of the same code and data result in different state selections, because (analytically) each state is assigned a value based on the arbitrary initialization that is refined with EM trianing. Thus, the same parameters can map to different outcomes and the HMM model for predicting the most likely state is not identifiable.

3. **How many states are actually being used by the HMM after training?**

   The HMM shrinks to use the number of states that is actually required of it - in this case, only 4.

1. **Download and install the Python framework GPy. Then on MATLAB, load the mgdata.dat file to generate 1200 samples of the Mackey-Glass chaotic time series. Separate it into a training set with the first 1000 samples and use the rest as test set. Train a Gaussian Process with an RBF kernel on the training set and produce (mean) forecasts for the test set; then compare with the ground truth.**

   See attached code in mackglass.py and Figure 2. Though the Gaussian Process successfully and accurately predicts the data in the training set, the mean forecasts for the test set are far off from ground truth.

2. **Train a Polynomial Regression model (you're free to choose the hyperparameters!) on your preferred platform (Python or MAT-**

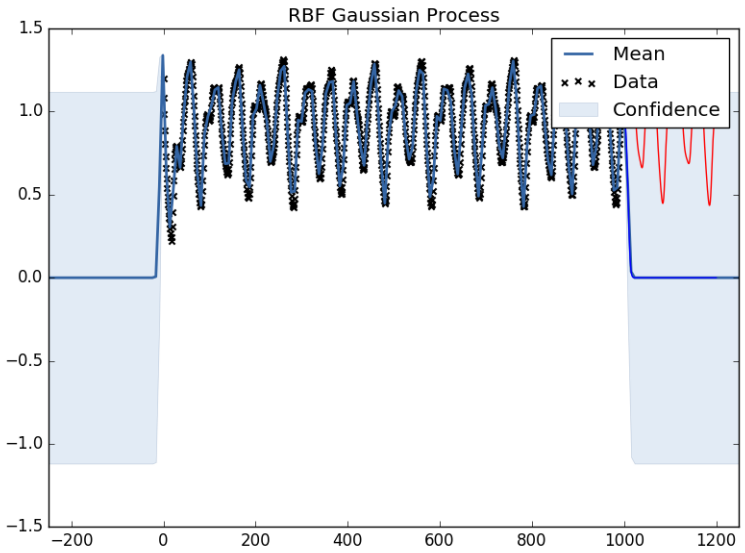


Figure 2: Mackey Glass RBF prediction.

**LAB are convenient!) and produce forecasts for the test set too. Which model did the best?**

See attached code and Figure 3. This model did worse than the previous - although it was able to generate a polynomial based on the training data, that polynomia curved away at the training set (compared to the flatline for the GP) and it was not able to fit the exact training data unless the hyperparameters were set such that the degree polynomial that was generated was extremely large (though anything large enough would freeze my computer).

3. **Research what is an autoregressive model. Train a GP on the Mackey-Glass time series in an autoregressive fashion (choose the lookback window length as you deem appropriate; you can use a starting value of 25 lookback samples) and forecast the next 200. Compare with the other methods in this question.**

   See attached code and Figure 4. The autoregressive model did the best of the previous models. This makes sense - unlike the previous models, the autoregressive model uses the immediately preceding steps to predict the next outcomes, allowing it to get a better predictive power.

   The model that was used was from the python package *statsmodels*. The implementation of the autoregressive model automatically finds the optimal lookback length given a maximum value. This is seen in the documentation **here**. The source can be found **here**

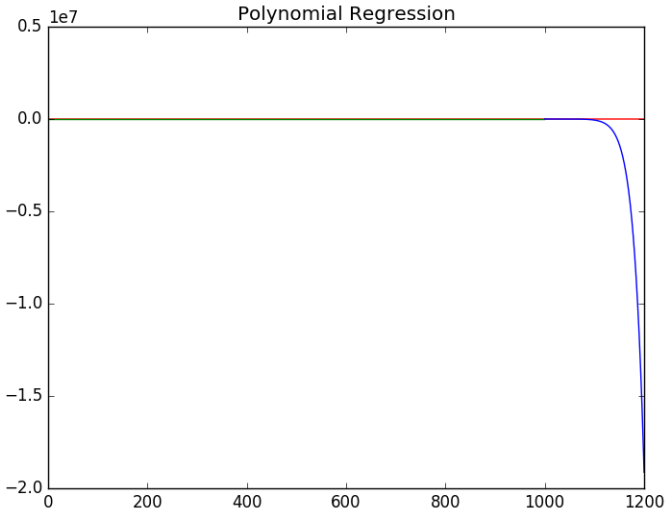


Figure 3: Mackey Glass Polynomial prediction.

In brief, the model decreases from the maximum lag value until the t-stat test has a significance of 95%.

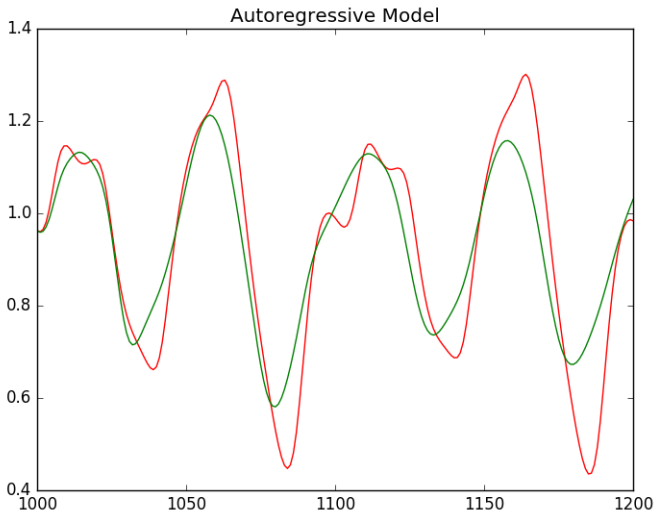


Figure 4: Mackey Glass Autoregressive prediction.