KALMAN FILTER POWERED VARIATIONAL AUTOENCODER FOR ACOUSTIC UNIT DISCOVERY

A PREPRINT

Jayanta Mukherjee*
LinkedIn Corporation
Mountain View, CA 94043
jmukherjee@linkedin.com

Bhiksha Raj
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA-15213
bhiksha@cs.cmu.edu

December 8, 2020

ABSTRACT

Structured Variational Autoencoders (SVAE) have been shown to provide efficient neural-network-based approximate inference in the presence of both discrete and continuous latent variables. Inspired by SVAE, we developed a VAE with Kalman Filters to model as latent variables. We applied the resulting Kalman-VAE to the task of acoustic unit discovery in a zero resource scenario. With Kalman Filter in Linear-Gaussian state-space models, the accuracy of the acoustic unit discovery could be significantly improved by reducing the train loss by 48% and root-mean square error by more than 50% by the Kalman-VAE. We demonstrated that Neural networks provide superior modeling with Kalman Filter can perform unsupervised learning task what is well-known in the supervised learning task.

Keywords variational autoencoder, kalman filter, unsupervised learning, acoustic unit discovery

1 Introduction

The Variational Autoencoder are latent variable models which uses deep neural networks to parameterize flexible probability distributions (Deep LVM). An auto-encoder encodes some input into a new and usually more compact representation which can be used to reconstruct the input data again. A VAE makes the assumption that the compact representation follows a probabilistic distribution (usually Gaussian) which makes it possible to sample new points and decode them into new data from a trained variational auto-encoder. There have been lots of work being done using Hidden Markov Model (HMM) as well as using Latent Variable Models (LVM). One striking difference between these two models is that in Hidden Markov Model (HMM) the latent variables are discrete while in LVM the latent variables can be continuous. One more difference is that for LVM both the latent and observed variables follow Gaussian Distribution, while for HMM only the observed variables have to be of Gaussian Distribution.

The Kalman Filter [1] has been popular for estimating dynamic state for multiple decades. The Extended Kalman Filter (EKF) [2] simply linearizes all nonlinear models so that the traditional linear Kalman filter can be applied.

Our contribution is to apply the Extended Kalman Filter to improvise the prediction of the words by moving in a direction which will reduce the distance between the actual word and predicted word. We applied Kalman Filter at every training step of the VAE to reduce the error and training losses by as much as 48% while incurring a maximum overhead of less than 14% of the epoch duration. We used PyTorch [3],[4] Gated Recurrent Unit (GRU) and Recurrent Neural Network (RNN) [5] to model the speech recognition aka Acoustic Unit Discovery. We applied our model to TIMIT [6] and compared against its performance against a Hidden Markov Model (HMM) [7].

*Thanks to my manager Nihit Purwar for his support.

***correspondence:** jmukherjee@linkedin.com

2 Related Work

Bhiksha et. al in their work in 2018 on VAE [8] with Hidden Markov Models (HMMs) as latent models to perform acoustic unit discovery in a zero resource scenario, and showed significant improvement in the accuracy of the acoustic unit discovery. In 2017, Gebhardt et al published their work [9] on the kernel Kalman rule as an improvement over the Kernel Bayes rule. The Extended Kalman Filter being superior in modeling dynamic state, we applied the Extended Kalman Filter with Variational Auto-encoder to bring further improvement in accuracy in acoustic unit discovery.

3 Variational Autoencoder

Variational Autoencoders [10] (VAEs) got popularity in unsupervised learning of complicated distributions. VAEs are appealing as they are built on top of standard function approximators and can be trained with stochastic gradient descent.

For every datapoint Z in the dataset, [11] there is one (or many) settings of the latent variables which causes the model to generate something very similar to Z. Formally, say we have a vector of latent variables x in a high-dimensional space X which we can easily sample according to some probability density function (PDF) P(x) defined over X. Then, say we have a family of deterministic functions f(x; θ), parameterized by a vector θ in some space Θ, where f : X $\times\Theta \rightarrow \chi$. f is deterministic, but if x is random and θ is fixed, then f(x; θ) is a random variable in the space χ . We wish to optimize θ such that we can sample x from p(x) and, with high probability, f(x; θ) will be like the Z's in our dataset.

To make this notion precise mathematically, we are aiming maximize the probability of each Z in the training set under the entire generative process, according to:

$$\mathrm{P(Z)} = \int \mathrm{P(Z|x;}\theta\mathrm{)P(x)dx} \tag{1}$$

Here, $\mathrm{f(x;}\theta\mathrm{)}$ has been replaced by a distribution $\mathrm{P(Z|x;}\theta\mathrm{)}$, which allows us to make the dependence of Z on x explicit by using the law of total probability. Based on the principle of "Maximum Likelihood" if the model is likely to produce training set samples, then it is also likely to produce similar samples, and unlikely to produce dissimilar ones. Given an observation z and a likelihood function $\mathrm{p(z_j|x)}$, the **Maximum Likelihood** estimator - ML finds the value of latent variable x which maximises the likelihood function $\mathcal{L} \triangleq \mathrm{p(z|x)}$.

$$\hat{\mathrm{x}}_{\mathrm{ml}} = \underset{\mathrm{x}}{\mathrm{argmax}}\ \mathrm{p(z|x)} \tag{2}$$

In VAEs, the choice of this output distribution is Gaussian, i.e., $\mathrm{p(Z|x;}\theta\mathrm{)} = \mathcal{N}(\mathrm{Z|f(x;}\theta\mathrm{)}, \sigma^2\mathrm{I})$. That is, it has mean $\mathrm{f(x;}\theta\mathrm{)}$ and covariance equal to the identity matrix I times some scalar σ. This replacement is necessary to formalize the intuition that some x needs to result in samples that are merely like Z. In general, and particularly early in training, our model will not produce outputs that are identical to any particular Z. By having a Gaussian distribution, we can use gradient descent (or any other optimization technique) to increase P(X) by making $\mathrm{f(x;}\theta\mathrm{)}$ approach Z for some x, i.e., gradually making the training data more likely under the generative model.

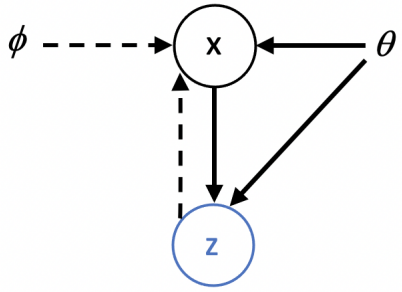


Figure 1: The type of directed graphical model under consideration. Solid lines denote the generative model $\mathrm{p}_\theta\mathrm{(x)p}_\theta\mathrm{(z|x)}$, dashed lines denote the variational approximation $\mathrm{q}_\phi\mathrm{(x|z)}$ to the intractable posterior $\mathrm{p}_\theta\mathrm{(x|z)}$.

VAEs approximately maximize Equation 1, according to the model shown in Figure 1. They are called "Autoencoders" only because the final training objective that derives from this setup does have an encoder and a decoder, and resembles

a traditional autoencoder. We need a new function Q(Z|x) which can take a value of Z and gives us a distribution over x values that are likely to produce Z.

Kullback-Leibler divergence (KL divergence) between p(x|Z) and Q(x), for some arbitrary Q (which may or may not depend on Z):

$$\mathrm{KL}[\mathrm{Q}(\mathrm{x}) \,||\, \mathrm{p}(\mathrm{x}|\mathrm{Z})] = \mathrm{E}_{\mathrm{x}\sim\mathrm{Q}}[\log \mathrm{Q}(\mathrm{x}) - \log \mathrm{p}(\mathrm{x}|\mathrm{Z})] \tag{3}$$

We can get both p(Z) and p(Z|x) into this equation by applying Bayes rule to p(x|Z):

$$\mathrm{KL}[\mathrm{Q}(\mathrm{x}) \,||\, \mathrm{p}(\mathrm{x}|\mathrm{Z})] = \mathrm{E}_{\mathrm{x}\sim\mathrm{Q}}[\log \mathrm{Q}(\mathrm{x}) - \log \mathrm{p}(\mathrm{Z}|\mathrm{x}) - \log \mathrm{p}(\mathrm{x})] + \log \mathrm{p}(\mathrm{Z}) \tag{4}$$

Here, log p(Z) comes out of the expectation because it does not depend on latent variable x. Negating both sides, rearranging, and contracting part of $\mathrm{E}_{\mathrm{x}\sim\mathrm{Q}}$ into a KL-divergence terms yields:

$$\log \mathrm{p}(\mathrm{Z}) - \mathrm{KL}[\mathrm{Q}(\mathrm{x}) \,||\, \mathrm{p}(\mathrm{x}|\mathrm{Z})] = \mathrm{E}_{\mathrm{x}\sim\mathrm{Q}}[\log \mathrm{p}(\mathrm{Z}|\mathrm{x})] - \mathrm{KL}[\mathrm{Q}(\mathrm{x}) \,||\, \mathrm{p}(\mathrm{x})] \tag{5}$$

The objective is to construct a Q which does depend on Z, and in particular, one which makes KL[Q(x) || p(x|z)] small:

$$\log \mathrm{p}(\mathrm{Z}) - \mathrm{KL}[\mathrm{Q}(\mathrm{x}|\mathrm{Z}) \,||\, \mathrm{p}(\mathrm{x}|\mathrm{Z})] = \mathrm{E}_{\mathrm{x}\sim\mathrm{Q}}[\log \mathrm{p}(\mathrm{Z}|\mathrm{x})] - \mathrm{KL}[\mathrm{Q}(\mathrm{x}|\mathrm{Z}) \,||\, \mathrm{p}(\mathrm{x})] \tag{6}$$

the left hand side of Equation 6 has the quantity we want to maximize: log P(Z) (plus an error term, which makes Q produce x's that can reproduce a given Z; this term will become small if Q is high-capacity). The right hand side is something we can optimize by stochastic gradient descent given the right choice of Q (although it may not be obvious yet how). Note that the framework—in particular, the right hand side of Equation 6 has suddenly taken a form which looks like an autoencoder, since Q is "encoding" Z into x, and P is "decoding" it to reconstruct Z. Maximizing log p(Z) while simultaneously minimizing KL[Q(x|Z) || p(x|Z)]. p(x|Z) is not something we can compute analytically: it describes the values of x that are likely to give rise to a sample like Z under our model in Figure 1. However, the second term on the left is pulling Q(x|z) to match p(x|Z). Assuming we use an arbitrarily high-capacity model for Q(x|z), then Q(x|z) will hopefully actually match p(x|Z), in which case this KL divergence term will be zero, and we will be directly optimizing log p(Z).

Assume that $\mathrm{Q}(\mathrm{x}|\mathrm{Z}) = \mathcal{N}(\mathrm{x}|\mu(\mathrm{Z};\vartheta),(\mathrm{Z};\vartheta))$, where μ and Σ are arbitrary deterministic functions with parameters ϑ that can be learned from data. The last term KL[Q(x|Z) ||p(x)] is now a KL Divergence between two multivariate Gaussian distributions, which can be computed in closed form as:

$$\mathrm{KL}[\mathcal{N}(\mu_0,\Sigma_0)||\mathcal{N}(\mu_1,\Sigma_1)] = \frac{1}{2}(\mathrm{tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1-\mu_0)^{\mathrm{T}}\Sigma_1^{-1}(\mu_1-\mu_0) - \mathrm{k} + \log(\frac{\det\Sigma_1}{\det\Sigma_0})) \tag{7}$$

Where, k is the dimensionality of the distribution. In our case, this simplifies to:

$$\mathrm{KL}[\mathcal{N}(\mu(\mathrm{Z}),\Sigma(\mathrm{Z}))||\mathcal{N}(0,\mathrm{I})] = \frac{1}{2}(\mathrm{tr}(\Sigma(\mathrm{Z})) + (\mu(\mathrm{Z}))^{\mathrm{T}}(\mu(\mathrm{Z})) - \mathrm{k} - \log(\det(\Sigma(\mathrm{Z})))) \tag{8}$$

The first term in Equation 6 is a bit more tricky. We could use sampling to estimate $\mathrm{E}_{\mathrm{x}\sim\mathrm{Q}}[\log \mathrm{p}(\mathrm{Z}|\mathrm{x})]$, but getting a good estimate would require passing many samples of x through f , which would be expensive. Hence, as is standard in stochastic gradient descent, we take one sample of x and treat P(Z|x) for that x as an approximation of $\mathrm{E}_{\mathrm{x}\sim\mathrm{Q}}[\log \mathrm{p}(\mathrm{Z}|\mathrm{x})]$.

The full equation to optimize is:

$$\mathrm{E}_{\mathrm{Z}\sim\mathrm{KL}}[\log \mathrm{p}(\mathrm{Z}) - \mathrm{KL}[\mathrm{Q}(\mathrm{x}|\mathrm{Z}) \,||\, \mathrm{p}(\mathrm{x}|\mathrm{Z})] = \mathrm{E}_{\mathrm{Z}\sim\mathrm{KL}}[\mathrm{E}_{\mathrm{x}\sim\mathrm{Q}}[\log \mathrm{p}(\mathrm{Z}|\mathrm{x})] - \mathrm{KL}[\mathrm{Q}(\mathrm{x}|\mathrm{Z}) \,||\, \mathrm{p}(\mathrm{x})]] \tag{9}$$

Sample a single value of Z and a single value of x from the distribution Q(x|Z), and compute the gradient of:

$$\log \mathrm{p}(\mathrm{Z}|\mathrm{x}) - \mathrm{KL}[\mathrm{Q}(\mathrm{x}|\mathrm{Z}) \,||\, \mathrm{p}(\mathrm{x})] \tag{10}$$

We can then average the gradient of this function over arbitrarily many samples of X and z, and the result converges to the gradient of Equation 9. Given $\mu(\mathrm{Z})$ and $\Sigma(\mathrm{Z})$ the mean and covariance of Q(x|Z) - we can sample from $\mathcal{N}(\mu(\mathrm{Z}),\Sigma(\mathrm{Z}))$ by first sampling $\epsilon \sim (0,\mathrm{I})$, then computing $\mathrm{x} = \mu(\mathrm{Z}) + \Sigma^{1/2}(\mathrm{Z}) * \epsilon$. Thus, the equation we actually take the gradient of is:

$$\mathrm{E}_{\mathrm{Z}\sim\mathrm{KL}}[\mathrm{E}_{\epsilon\sim\mathcal{N}(0,\mathrm{I})}[\log \mathrm{p}(\mathrm{Z}|\mathrm{x} = \mu(\mathrm{Z}) + \Sigma^{1/2}(\mathrm{Z}) * \epsilon)] - \mathrm{KL}[\mathrm{Q}(\mathrm{x}|\mathrm{Z}) \,||\, \mathrm{p}(\mathrm{x})]] \tag{11}$$

4 Kalman Filter

The Kalman filter [12] is being popular in predicting next state of dynamic systems. It uses an iterative approach to tune the model to rectify the error.

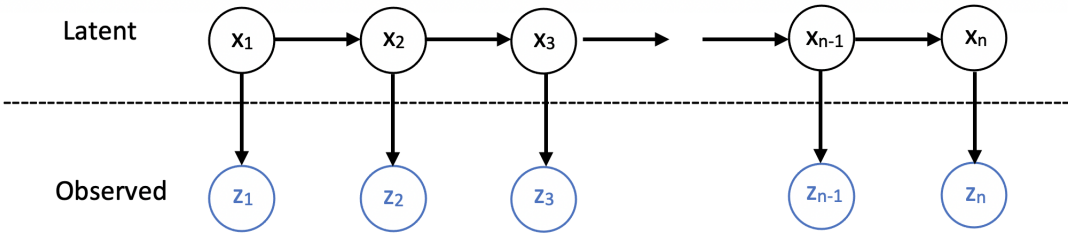


Figure 2: Kalman Filter showing latent and observed variables

The Kalman filter and smoother are based on the following probabilistic model [13].

- Like a discrete-state HMM, the sequence of observations $z_1, z_2, ..., z_n$ is modeled jointly along with a sequence of hidden latent states $x_1, x_2, ..., x_n$. Under assumption:

$$p(z_{1:n}, x_{1:n}) = p(x_1)p(z_1|z_1)\prod_{j=2}^{n} p(x_j|x_{j-1})p(z_j|x_j) \tag{12}$$

- Difference from a discrete-state HMM is that each hidden state x_j is modeled as a continuous random variable in $\mathbb{R}^d$ with a multivariate normal distribution.
- The initial distribution $p(x_1)$, the transition distributions $p(x_j|x_{j-1})$ (a.k.a. the "process model") and the emission distributions $p(z_j|x_j)$ (a.k.a. the "measurement model") are assumed to be

$$\begin{aligned} p(x_1) &= \mathcal{N}(x_1|\mu_0, V_0) \\ p(x_j|x_{j-1}) &= \mathcal{N}(x_j|Fx_{j-1}, Q) \\ p(z_j|x_j) &= \mathcal{N}(z_j|Hx_j, R) \end{aligned} \tag{13}$$

  where
  - $x_j \in \mathbb{R}^d$ (the state of the system at time step j),
  - $z_j \in \mathbb{R}^d$ (the measurements at time step j),
  - $\mu_0 \in \mathbb{R}^d$ is an arbitrary vector (the initial mean, our "best guess" at the initial state),
  - $V_0 \in \mathbb{R}^{d\times d}$ is a symmetric positive definite matrix (the initial covariance matrix, quantifying our uncertainty about the initial state),
  - $F \in \mathbb{R}^{d\times d}$ is an arbitrary matrix (modeling the physics of the process, or a linear approximation thereof),
  - $Q \in \mathbb{R}^{d\times d}$ is a symmetric positive definite matrix (quantifying the noise/error in the process that is not captured by F),
  - $H \in \mathbb{R}^{D\times d}$ is an arbitrary matrix (relating the measurements to the state),
  - $R \in \mathbb{R}^{D\times D}$is a symmetric positive definite matrix (quantifying the noise/error of the measurements).
- The model can easily be extended to handle time-dependence in F, Q, H, and R, by simply replacing them with $F_j, Q_j, H_j,$ andR_j in the expressions above. The Kalman filter and smoother algorithms can easily be modified to handle this generalization.

These are multivariate normal distributions, in other words,

$$\begin{aligned} p(x_j|z_{1:j}) &= N(x_j|\mu_j, V_j) \\ p(x_j|z_{1:n}) &= N(x_j|\hat{\mu}_j, V_j) \end{aligned} \tag{14}$$

for some $\mu_j, V_j, \hat{\mu}_j, \hat{V}_j$. Consequently, we can reformulate the forward and backward algorithms in terms of the parameters $\mu_j, V_j, \hat{\mu}_j, \hat{V}_j$, rather than computing the density at every point.

The Kalman filter is identical to the forward algorithm for discrete-state HMMs, except that it is expressed in terms of μ_j, V_j instead of $s_j(z_j)$ (and the derivation involves an integral instead of a sum).

4.1 Kalman Filter: Forward Algorithm

In the Kalman Filter forward algorithm, we compute $p(x_j|z_{1:j})$ sequentially for j = 1, 2, ..., n in that order.

For j = 1,

$$\begin{aligned} P(x_1|z_1) &\propto p(z_1|x_1)p(x_1) = \mathcal{N}(z_1|Hx_1, R)\mathcal{N}(x_1|\mu_0, V_0) \\ &\propto \mathcal{N}(x_1|(\mu_0 + K_1(x_1 - H\mu_0)), (I - K_1H)V_0) \end{aligned} \tag{15}$$

We define:

$$K_1 = V_0H^T(HV_0H^T + R)^{-1} \tag{16}$$

Therefore, $p(x_1|z_1) = \mathcal{N}(x_1|\mu_1, V_1)$ where,

$$\begin{aligned} \mu_1 &= \mu_0 + K_1(z_1 - H\mu_0) \\ V_1 &= (I - K_1H)V_0 \end{aligned} \tag{17}$$

Generalizing the above equations for step j [13], $p(x_{j-1}|z_{1:j-1}) = \mathcal{N}(x_{j-1}|\mu_{j-1}, V_{j-1})$

$$\begin{aligned} P(x_j|z_{1:j}) \propto p(z_{1:j}|x_j)p(x_1) &= \int p(z_{1:j}, x_{j-1}, x_j)dz_{j-1} \\ &= \int p(z_{1:j}, x_{j-1})p(x_j|x_{j-1})p(x_j|z_j)dx_{j-1} \\ &\propto \int \mathcal{N}(x_{j-1}|\mu_{j-1}, V_{j-1})\mathcal{N}(x_j|Fx_{j-1}, Q)\mathcal{N}(z_j|Hx_j, R)dx_{j-1} \end{aligned} \tag{18}$$

Kalman Gain at step j, $K_j = VH^T(HVH^T + R)^{-1}$

To avoid confusion, and for later reference, let's denote V by P_{j-1}, i.e.,

$$P_{j-1} = FV_{j-1}F^T + Q \tag{19}$$

Therefore, $p(x_j|z_{1:j}) = \mathcal{N}(x_j|\mu_j, V_j)$, where

$$\begin{aligned} \mu_j &= F\mu_{j-1} + K_j(z_j - HF\mu_{j-1}) \\ V_j &= (I - K_jH)P_{j-1} \end{aligned} \tag{20}$$

Putting the above equations altogether:

Algorithm 1: Kalman Filter Forward Algorithm

Input: Observed data $z_1, ..., z_n$ and model parameters μ_0, V_0, F, Q, H, R
Result: $p(x_j|z_{1:j}) = \mathcal{N}(x_j|\mu_j, V_j)\ \forall j = 1, 2, ..., n.$
Initialization:
$K_1 = V_0H^T(HV_0H^T + R)^{-1}$
$\mu_1 = \mu_0 + K_1(z_1 - H\mu_0)$
$V_1 = (I - K_1H)V_0$
$P_1 = FV_1F^T + Q$
for *j = 2, . . . , n* **do**
  $K_j = P_{j-1}H^T(HP_{j-1}H^T + R)^{-1}$
  $\mu_j = F\mu_{j-1} + K_j(z_j - HF\mu_{j-1})$
  $V_j = (I - K_jH)P_{j-1}$
  $P_j = FV_jF^T + Q$
end

Recursive Kalman

$$\hat{u}_{new} = \hat{u}_{old} + K(b_{new} - A_{new}\hat{u}_{old}) \tag{21}$$

where K is the Kalman Gain

4.1.1 State-Space Model

State-Space Model can be user to model the input-output behavior of the controlled system.

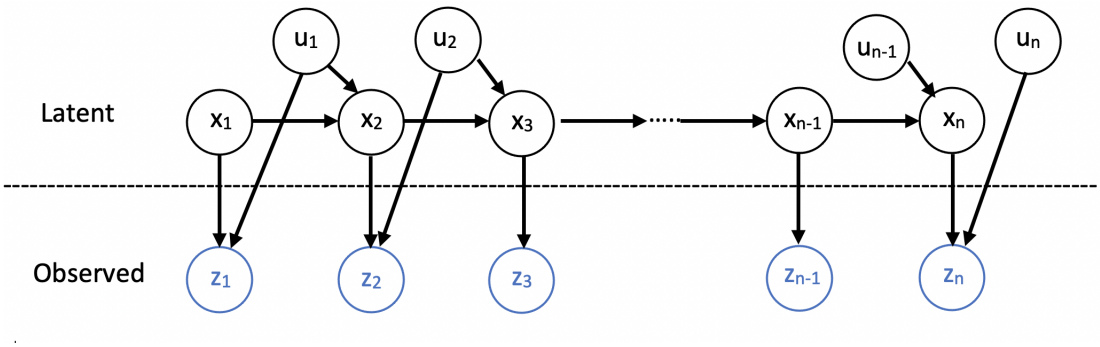


Figure 3: State-Space Model showing latent and observed variables

State Dynamics Equation

$$x_t = Ax_{t-1} + Bu_{t-1} + w_t \tag{22}$$

Output Equation

$$z_t = Cx_t + Du_t + v_t \tag{23}$$

$$\hat{x}_t = \frac{1}{t}\sum_{\tau=1}^{t} z_\tau \tag{24}$$

$$\hat{x}_{t-1} = \frac{1}{t-1}\sum_{\tau=1}^{t-1} z_\tau \tag{25}$$

$$\hat{x}_t = \frac{t-1}{t}\hat{x}_{t-1} + \frac{1}{t}z_t \tag{26}$$

$$\hat{x}_t = \hat{x}_{t-1} + \frac{1}{t}(z_t - \hat{x}_{t-1}) \tag{27}$$

Where $\frac{1}{t}$ represents Kalman Gain K_t.

Forward Recursion based on Bayes Rule

$$\begin{aligned} P(x_t|z_{1:t}) = \int P(x_t, x_{t-1}|z_t, z_{1:t-1})dy_{t-1} &= \int \frac{P(x_t, x_{t-1}, z_t|z_{1:t-1})}{P(z_t|z_{1:t-1})}dy_{t-1} \\ &\propto \int P(x_{t-1}|z_{1:t-1})P(x_t|x_{t-1}, z_{1:t-1})P(z_t|x_t, z_{1:t-1})dy_{t-1} \\ &= \int P(x_{t-1}|z_{1:t-1})P(x_t|x_{t-1})P(z_t|x_t)dy_{t-1} \end{aligned} \tag{28}$$

Notation:

$$\hat{x}_t^\tau \equiv \mathbb{E}[x_t|z_1.....z_\tau] \tag{29}$$

Prediction:

$$\hat{x}_t^{t-1} = A\hat{x}_{t-1}^{t-1} \tag{30}$$

Correction:

$$\hat{x}_t = \hat{x}_t^{t-1} + K_t(z_t - C\hat{x}_t^{t-1}) \tag{31}$$

Where, K_t is the Kalman Gain = $\hat{V}_t^{t-1}C^T(C\hat{V}_t^{t-1}C^T + R)^{-1}$

Prediction Variance:

$$\hat{V}_t^{t-1} = A\hat{V}_{t-1}^{t-1}A^T + Q \tag{32}$$

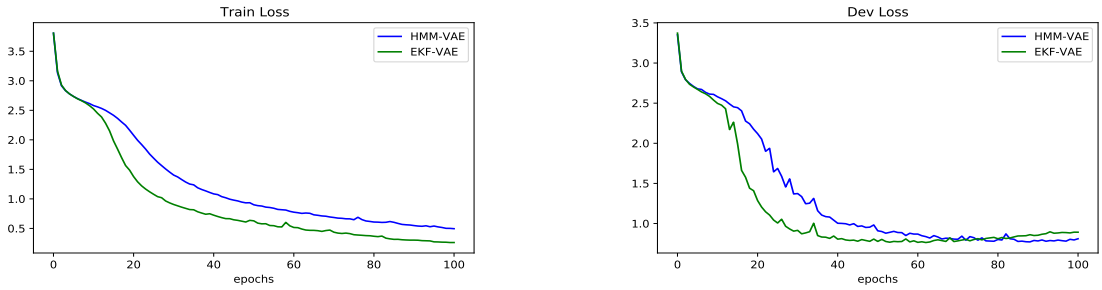
Corrected Variance:

$$\hat{V}_t^t = A\hat{V}_t^{t-1} - K_t C\hat{V}_t^{t-1} \tag{33}$$

Experiments

We build our model using RNN, GRU Cell of PyTorch and developed Encoder and Decoder RNN as part of the VAE model for benchmarking the Hidden Markov Model of VAE. For experimenting Extended Kalman VAE, we added Extended Kalman Filter based prediction in each step of the training to improve the prediction of the dynamic state. We applied model on TIMIT [6] data and compute the error by computing the distance between the actual and predicted data and learn from it.

We achieved more than 47% reduction in training loss at the end of 100 iterations (epochs) and experienced a maximum of 49% reduction in training loss individual epoch as shown in 4 .

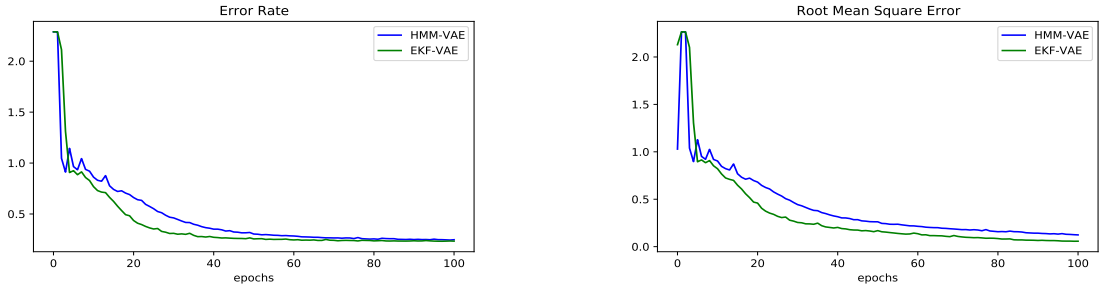


(a) Training Loss (b) Evaluation/Dev Loss

Figure 4: Loss Reduction

As shown in 4, we achieved more than 42% reduction in evaluation (dev) loss in evaluation set at a specific epoch, although in later iterations HMM over-performed by 10%.

The error and root-mean square error 5 reduced by Extended Kalman VAE over HMM based VAE. The error rate reduced up to 37% and root-mean square error reduced by 57% at individual epoch.

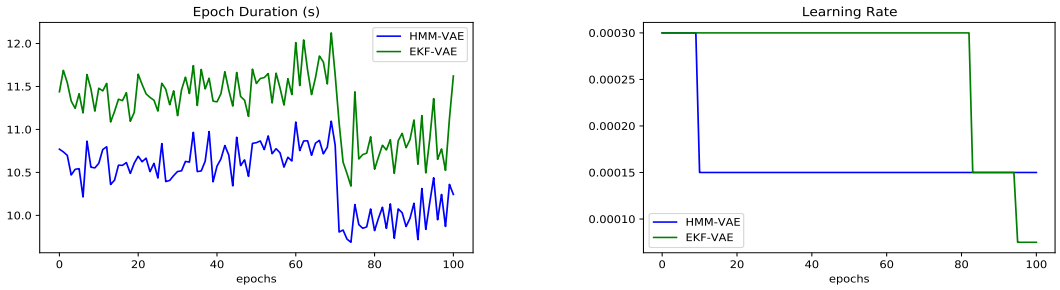


(a) Error Rate (b) Root Mean Square Error

Figure 5: Error Reduction

But, the additional Extended Kalman Filter computation increases the training time at every iteration which resulted in less than 14% increase in epoch duration.

The error for acoustic discovery 7 at the end of the 100 epochs 8 went less than 6% as compared to HMM based VAE with more than 12% error.



(a) Epoch Duration (b) Learning Rate

Figure 6: Performance Comparison between HMM-based and EKF-based VAE

```
Predict:
w uh kcl k ix dcl b eh r l ix s iy dh ax f iy y ao r dcl z pau f r uw dh ax s n ow f l er ix s h#
Ground-truth:
h# w ax kcl k ix dcl b eh r l ix s iy dh ax f iy y ao r dcl z pau th r uw dh ax s n ow f l er ix s h#
```

Figure 7: Extended Kalman Filter based VAE Prediction

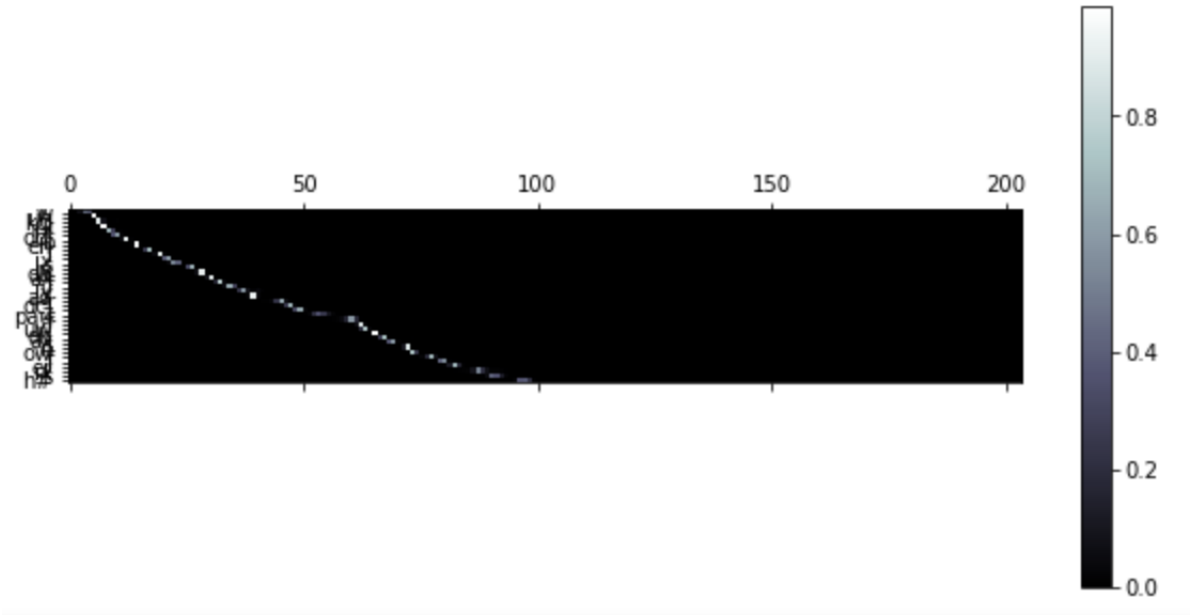


Figure 8: Acoustic Discovery

Conclusions

EKF Powered VAE provide significant improvement over HMM based VAE due to better prediction of the dynamic state by the Kalman filter.

Acknowledgements

Thanks to my wife Mahaswata for her relentless support and buying me NVIDIA RTX2080 based laptop on which all the experiments were performed.

References

[1] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35, 1960. doi: 10.1115/1.3662552. URL http://dx.doi.org/10.1115/1.3662552.

[2] Simon J. Julier and Jeffrey K. Uhlmann. A new extension of the kalman filter to nonlinear systems. pages 182–193, 1997.

[3] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[5] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.*, 45 (11):2673–2681, 1997. URL http://dblp.uni-trier.de/db/journals/tsp/tsp45.html#SchusterP97.

[6] The darpa timit acoustic-phonetic continuous speech corpus.

[7] L. Rabiner and B. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, 1986. doi: 10.1109/MASSP.1986.1165342.

[8] Janek Ebbers, Jahn Heymann, Lukas Drude, Thomas Glarner, Reinhold Haeb-Umbach, and Bhiksha Raj. Hidden markov model variational autoencoder for acoustic unit discovery. In *INTERSPEECH 2017, Stockholm, Schweden*, 2017.

[9] G. H. W. Gebhardt, A. Kupcsik, and G. Neumann. The kernel kalman rule - efficient nonparametric inference with recursive least squares. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[10] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[11] Carl Doersch. Tutorial on variational autoencoders, 2016. URL http://arxiv.org/abs/1606.05908. cite arxiv:1606.05908.

[12] Paul Michael Newman. Ekf based navigation and slam. SLAM Summer School 2006, Oxford, July 2006. Background Material, Notes and Example Code.

[13] Jeffrey W. Miller. Kalman filter and smoother. Lecture Notes on Advanced Stochastic Modeling. Duke University, Durham, NC, 2016.