Autoencoders

Presented by : Melissa Mozifian
July 24, 2017

Overview

1. Definition and Background
2. Family of Autoencoders
3. Applications
4. Autoencoder in dynamic system modeling
5. Beyond Autoencoders, towards Generative Models

Definitions

- An autoencoder is a neural network that is trained to attempt to copy its input to its output
- It has a hidden layer **h** that describes a **code** used to represent the input
- The network has two parts :
  - An encoder function $h = f(x)$
  - A decoder that produces a reconstruction $r = g(h)$
- Designed to be unable to learn to copy perfectly
- It often learns useful properties of the data because the model is forced to prioritize which aspects of the input should be copied

Background

- The idea of autoencoders has been around for decades (LeCun, 1987; Bourlard and Kamp, 1988; Hinton and Zemel,1994)
- Traditionally, used for dimensionality reduction or feature learning
- Recently, theoretical connections between autoencoders and latent variable models have brought autoencoders to the forefront of generative modelling

Undercomplete Autoencoders

- Copying the input to the output may sound useless, but we are typically not interested in the output of the decoder
- Instead, we hope that training the autoencoder will result in **h** taking on useful properties
- One way to obtain useful features from the autoencoder is to constrain **h** to have smaller dimension than **x**
- Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data
- The learning process is described simply as minimizing a loss function

$$L(x, g(f(x))) \tag{1}$$

Regularized Autoencoders

- Autoencoders fail to learn anything useful if the encoder and decoder are given too much capacity
- Similar problem if the hidden code is allowed to have dimension equal to the input and in the **overcomplete** case in which the hidden code has dimensions greater than the input
- The code dimensions and the capacity of the encoder and decoder must be chosen based on the complexity of the distribution of the model
- Instead of limiting the model capacity, regularized autoencoders use a loss function that constrains the model in the same way

Sparse Autoencoders

- An autoencoder whose training criterion involves a sparsity penalty $\Omega(h)$ on the code layer h in addition to the reconstruction error

$$L(x, g(f(x))) + \Omega(h) \tag{2}$$

- $g(h)$ is the decoder output
- $h = f(x)$ the encoder output
- Used to learn features for tasks such as **classification**
- We can think of the penalty $\Omega(h)$ simply as a regularizer term added to a feedforward network
    - Its primary task is to copy the input to the output(unsupervised learning objective) and possibly also perform some supervised task(with a supervised learning objective) that depends on these sparse features

Denoising Autoencoders

- Change the reconstruction error term of the cost function
- A denoising autoencoder or DAE minimizes

$$L(x, g(f(\tilde{x}))) \tag{3}$$

- where $\tilde{x}$ is a corrupted copy of x
- Denoising autoencoders must then undo this corruption rather than simply copying their input

Contractive Autoencoder

- Another strategy for regularizing an autoencoder is to use a penalty Ω as in sparse autoencoders

$$L(x, g(f(x))) + \Omega(h), \tag{4}$$

- But with a different form of Ω

$$\Omega(h) = \lambda || \frac{\partial f(x)}{\partial x} ||_F^2 \tag{5}$$

- The penalty $\Omega(h)$ is the squared Frobenius norm (sum of squared elements) of the Jacobian matrix of partial derivatives associated with the encoder function
- This forces the model to learn a function that does not change much when $\mathbf{x}$ changes slightly
- Because this penalty is applied only at training examples, it forces the autoencoder to learn features that capture information about the training distribution

Deep Autoencoders

- There are many advantages to depth in a feedforward networks
- Because autoencoders are feedforward networks, these advantages also apply to autoencoders
- One major advantage of non-trivial depth is that the universal approximator theorem guarantees that a feedforward neural network with at least one hidden layer can represent an approximation of any function to an arbitrary degree of accuracy, provided that it has enough hidden units
- This means that an autoencoder with a single hidden layer is able to represent the identity function along the domain of the data arbitrarily well
- But the mapping from input to code is shallow!

Deep Autoencoders

- A deep autoencoder, with at least one additional hidden layer inside the encoder itself, can approximate any mapping from input to code arbitrarily well, given enough hidden units.
- Experimentally, deep autoencoders yield much better compression than corresponding shallow or linear autoencoders

Training Deep Autoencoders

- A common strategy for training a deep autoencoder is to **greedily** pretrain a deep network by training each layer in turn
- This method trains the parameters of each layer individually while freezing parameters for the remainder of the model
- To produce better results, after this phase of training is complete, fine-tuning using backpropagation can be used to improve the results by tuning the parameters of all layers are changed at the same time

Application of Autoencoders

- Autoencoders succeed at the task of dimensionality reduction
- Lower-dimensional representations can improve performance on many tasks, such as classication
- Models of smaller spaces consume less memory and runtime

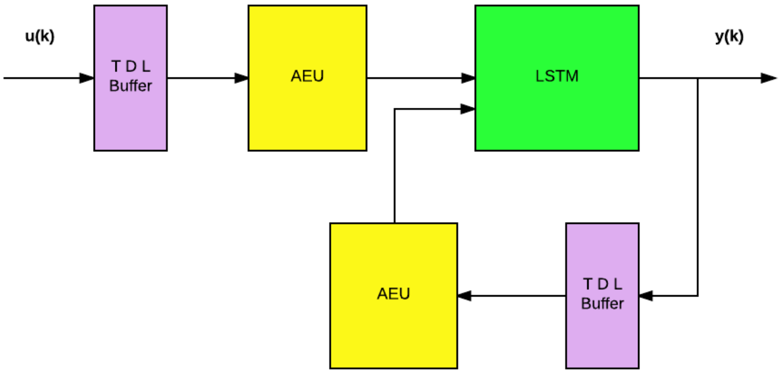
Application of Autoencoders Cont'd

- Another task that benefits even more than usual from dimensionality reduction is information retrieval
  - The task of finding entries in a database that resemble a query entry
- If we train the dimensionality reduction algorithm to produce a code that is low-dimensional and binary, then we can store all database entries in a hash table mapping binary code vectors to entries
- Using a deep autoencoder as a hash-function for finding **approximate** matches

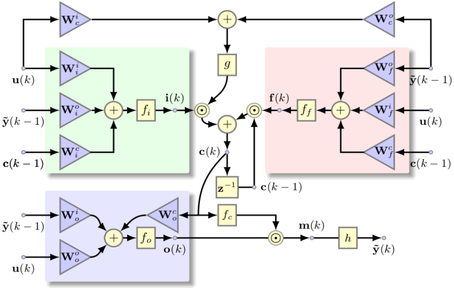
Some terminology :

- **Tapped Delay Line** : A recurrent cell that acts like a **buffer**. The input is a time series signal and the output is a concatenation of the input signal over a given history
- **Teacher Forcing** : Feed ground-truth samples y_t back into the model. These fed back samples force the RNN to stay close to the ground-truth sequence.

Dynamic System Modeling with LSTM & Autoencoders Architecture



Long-Short-Term-Memory (LSTM)

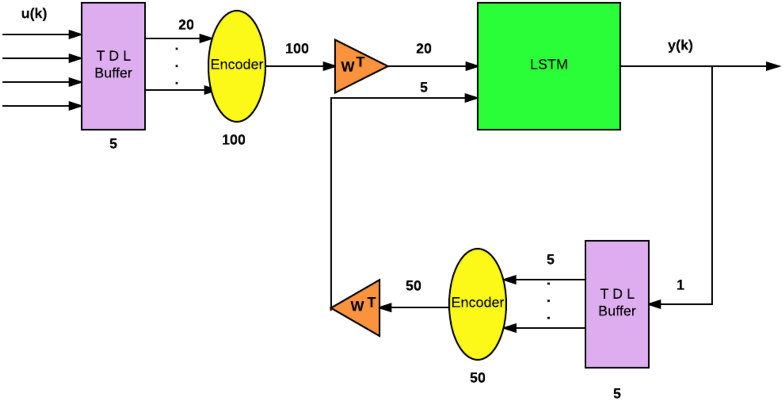


$$\mathbf{i}(k) = f_i\big(\mathbf{W}_i^i * \mathbf{u}(k) + \mathbf{W}_i^o * \tilde{\mathbf{y}}(k-1) + \mathbf{W}_i^c * \mathbf{c}(k-1)\big)$$
$$\mathbf{f}(k) = f_f\big(\mathbf{W}_f^i * \mathbf{u}(k) + \mathbf{W}_f^o * \tilde{\mathbf{y}}(k-1) + \mathbf{W}_f^c * \mathbf{c}(k-1)\big)$$
$$\mathbf{o}(k) = f_o\big(\mathbf{W}_o^i * \mathbf{u}(k) + \mathbf{W}_o^o * \tilde{\mathbf{y}}(k-1) + \mathbf{W}_o^c * \mathbf{c}(k-1)\big)$$
$$\mathbf{c}(k) = \mathbf{i}(k) \odot g\big(\mathbf{W}_c^i * \mathbf{u}(k) + \mathbf{W}_c^o * \tilde{\mathbf{y}}(k-1)\big) + \mathbf{f}(k) \odot \mathbf{c}(k-1)$$
$$\mathbf{m}(k) = \mathbf{c}(k) \odot \mathbf{f}_o(k)$$
$$\tilde{\mathbf{y}}(k) = \mathbf{h}(\mathbf{m}(k-1))$$

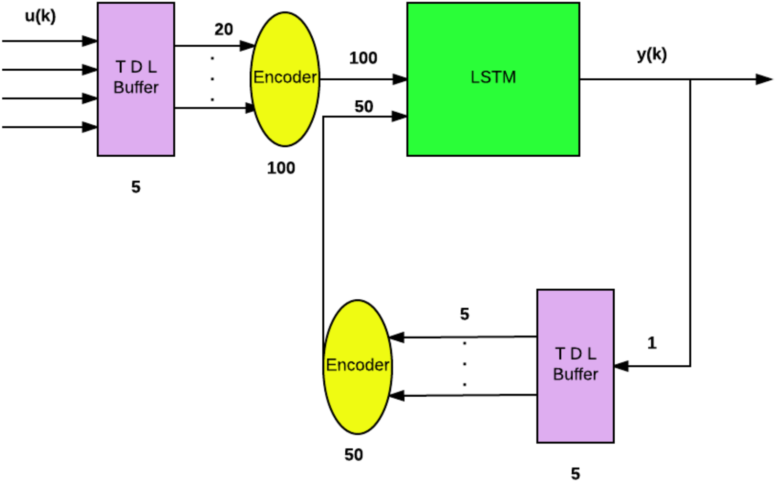
LSTM & Autoencoders Stage-wise Training

- Two-Stage Training
  - Stage 1 : Train the network with the autoencoder as trainable with reconstruction and teacher force
  - Stage 2 : Fine-tune the encoder weights without reconstruction and teacher force. Then freeze the encoder weights and train the LSTM weights

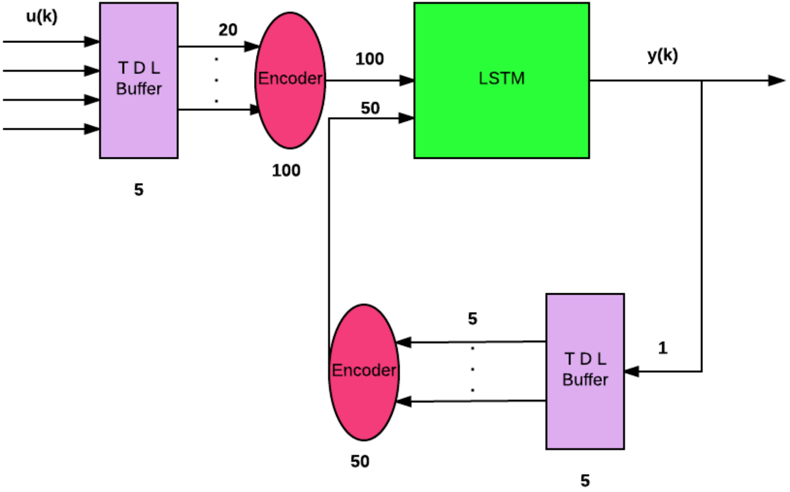
LSTM & Autoencoders Stage 1 Training



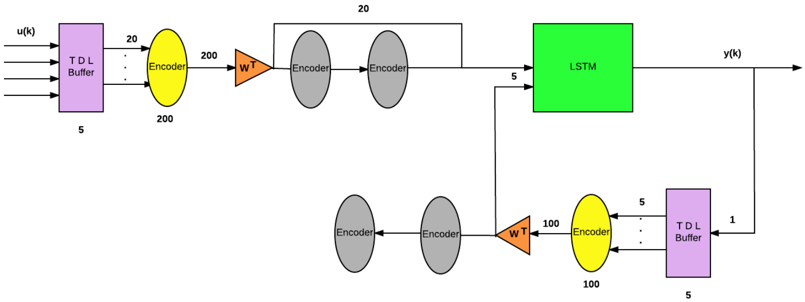
LSTM & Autoencoders Fine-Tuning



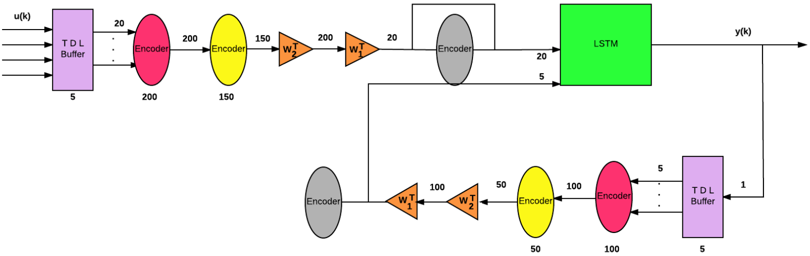
LSTM & Autoencoders Stage 2 Training



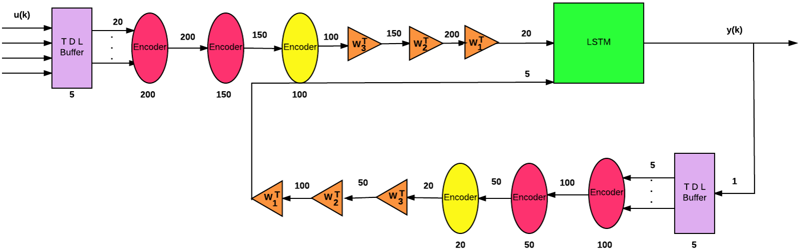
LSTM & Deep Autoencoders Greedy Stage-wise Training



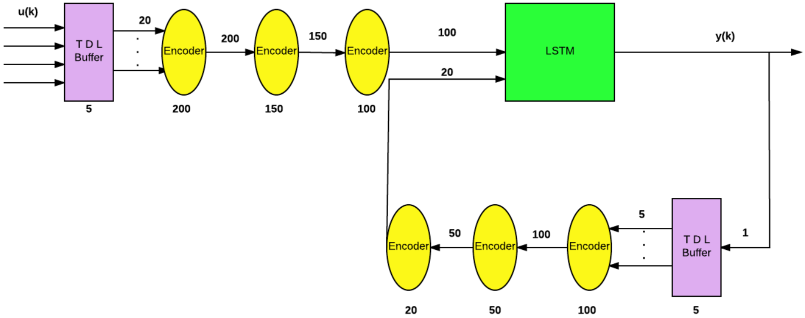
LSTM & Deep Autoencoders Greedy Stage-wise Training



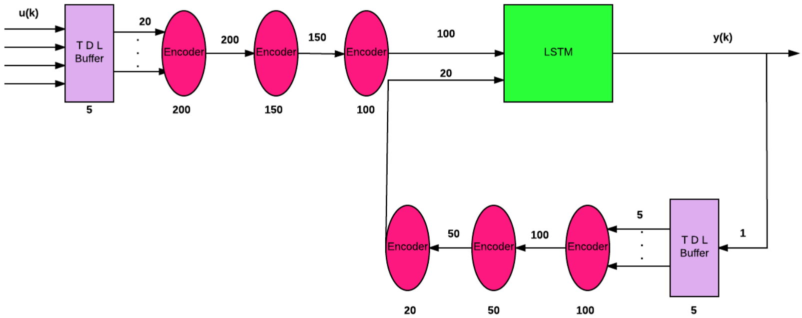
LSTM & Deep Autoencoders Greedy Stage-wise Training



LSTM & Deep Autoencoders Fine-Tuning



LSTM & Deep Autoencoders Stage 2



LSTM & Autoencoders Results

- We are comparing the following :
  - LSTM
  - LSTM TDL
  - LSTM TDL/AEU
  - LSTM TDL/Deep AEU

Generative Models

- It is easy to forget how much we know about the world
  - There is massive amount of information out there either in the physical world of atoms or digital world of bits
- The trick is developing models and algorithms that can analyze and understand this trove of data
- Generative models are one of the most promising approaches towards this goal

Generative Models

- To train a generative model we first collect a large amount of data in some domain
- We then train a model to generate data like it
- This way the models are forced to discover and efficiently internalize the essence of the data in order to generate it
- In the long run these models hold the potential to automatically learn the natural features of a dataset

Generative Models

- Nearly any generative model with latent variables and equipped with an inference procedure (for computing latent representations given input) may be viewed as a particular form of autoencoder
- Two generative modeling approaches that emphasize this connection with autoencoders are :
  - Variational Autoencoder
  - Generative Stochastic Networks
- These models naturally learn high-capacity, overcomplete encodings of the input and do not require regularization
- This is because they were trained to approximately maximize the probability of the training data rather than copying input to the output

Approaches to generative models

- Generative Adversarial Networks (GANs)
    - Pose the training process as a game between two separate networks
        - A generator network and a second discriminative network
    - The discriminative network tries to classify samples as either coming from the true distribution $p(x)$ or the model distribution $p(\hat{x})$
    - Every time the discriminator notices a difference between the two distributions the generator adjusts its parameters slightly to make it go away
    - The discriminator emits a probability value, indicating the probability that $\mathbf{x}$ is a real training example rather than a fake sample drawn from the model
- Variational Autoencoders (VAEs)
    - Allows us to formalize this problem in the framework of probabilistic graphical models where we are maximizing a lower bound on the log likelihood of the data

Variational Autoencoders Motivation

- There are a couple of downsides to using plain GANs
- The data, say a picture, is usually generated off some arbitrary noise
- If we wanted to generate a picture with specific features, there's no way of determining which initial noise values would produce that picture
- A generative adversarial model only discriminates between 'real' and 'fake' images
    - There is no constraint that an image of a cat has to look like a cat?

Variational Autoencoders

- VAE is a directed model that uses learned approximate inference and can be trained purely with gradient-based methods
- To generate a sample from the model, the VAE first draws a sample z from the code distribution $p_{model}(z)$
- The sample is then run through a differentiable generator network $g(z)$
- Finally $\mathbf{x}$ is sampled from a distribution

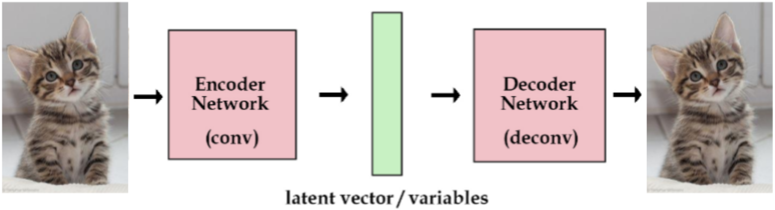
$$p_{model}(x; g(z)) = p_{model}(x|z) \tag{6}$$

- During training, the encoder $q(z|x)$ is used to obtain z and $p_{model}(x|z)$ is then viewed as the decoder

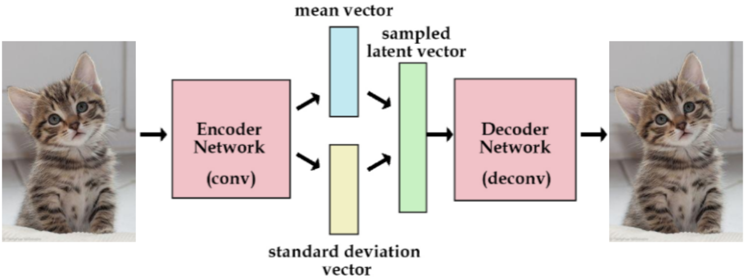
Variational Autoencoders Intuition

- Say we have an autoencoder that takes in some image and encodes it for us
- We add a constraint on the encoding network, that forces it to generate latent vectors that roughly follow a unit gaussian distribution
- To generate new data we then sample a latent vector from the unit gaussian and pass it into the decoder
- In practice, there is a tradeoff between how accurate our network can be and how close its latent variables can match the unit gaussian distribution
    - We let the network decide this itself

Standard Autoencoders



Variational Autoencoders



Variational Autoencoders Loss Function

- For the loss term, we sum up two separate losses
  - The generative loss : Mean squared error that measures how accurately the network reconstructed the images
  - Latent loss: which is the KL divergence that measures how closely the latent variables match a unit gaussian

Variational Autoencoders Extension

- The VAE framework is very straightforward to extend to a wide range of model architectures
- VAEs can be extended to generate sequences by defining variational RNNs (Chung et al, 2015b) by using a recurrent encoder and decoder within the VAE framework

Resources

- http://www.deeplearningbook.org/
  - Chapter 14 Autoencoders
  - Chapter 20 Deep Generative Modesl
    - 20.10.3 Variational Autoencoders
    - 20.10.4 Generative Adversatial Networks
- Code: https://github.com/wavelab/rnn_sysid/tree/autoencoder

Discussion

Questions/Thoughts?