Techniques comparison for music generation

Cristiano Milanese

2600169
B Computer Science
Vrije Universiteit

Joe Hopkinson

#-student number-#
B Computer Science
Vrije Universiteit

Gregorio Bertozzini

#-student number-#
B Computer Science
Vrije Universiteit

Etienne Herbeaux

#-student number-#
B Computer Science
Vrije Universiteit

Razvan Hotanu

#-student number-#
B Computer Science
Vrije Universiteit

March 18, 2020

Abstract

This is the paper's abstract . . .

Outline

Discriminative vs Generative models Introducing machine learning models, the first decision we are called to make is whether the scope of our research focuses on classification or generation. This two concepts tightly relate to their learning process. A discriminative model maps directly into classification problems, shaping the decision boundary between the inputs in order to infer the class of a particular instance, while the generative one takes into consideration the joint probability distribution, namely the likelihood of an input instance to belong to a class rather than another.

Supervised vs Unsupervised learning

Deep Learning After unsuccessfully experiment with with probability classes using Hidden Markov models, we sought for a solution that could solve the problem of error propagation, mitigating the accumulation of imprecision during the substeps of feature learning. Deep Learning helps in this sense as it deploys end-to-end learning, an approach that combines intermediate training, empowerin the network to take into acccount

from Machine Learning to Generative models Moving on from multiplayer perceptron and classical machine learning methods, CNNs adopt convolutional operation at their hidden layer phases to detect patterns from the input data. The main different with a fully connected layer is that we are able to retain the spacial factor or shape of our input. This operation is achieved by defining what, in image processing jergon, is called a *kernel* or *filter*. Taking the dot product of the input with this filter results in new mappings, each definining a important characteristic of the input. Lastly, a fully connected layer is hooked to our model in order to produce a score function and ranks our guessed outputs. What prevented us from using CNN for our project, despite keen to preserve the initial shape of the data, is the lacking ability to deal with sequential data, one of the prelevant feature of songs, read as sequences of notes. **we needed models that are able to perform**

generation from different length sequences Our focus, therefore, moved towards specialized models for sequence analysis, such as Recurrent Neural Nets. These architecture are good at modelling sequences of data, among its application we often encounter natural sequences anaysis such as for audio or text. Common to other machine learning models, RNNs share a structure of input, hidden and output layers connected via weighted functions, but they are equipped with loops at the hidden layers in order to render the information persistent. They use hidden states as memory cells, passing on each of their outputs to the following cell, which vector is then concatenated with the following input, collecting information coming from previous states. As with other models, after each forward pass, the neural network performs backpropagation in order to learn, but under this shape our model is not able to retain previous iteration and we would end up learning only from the last. Backpropagation through Time is a technique applicable to unrolled models, namely the version of what we have built so far with all temporal passes withheld in memory. Yet our solution suffers from the vanishing gradient problem, embedded in the nature of backpropagation as the update method for the various weights of the hidden layers. Its value shrinks until it is so small that it does not contribute to learning anymore. Main consequence of this issue is that vanilla RNN struggle to remember long sequences of input.

Long-Short Term Memory architecture

Is an evolution of vanilla recurrent neural network that solves the issue of vanishing gradient descent, allowing the net to remember longer sequences by forgetting some information along the way. This all process is implemented though *gates*, which filter out parts of the sequences that are not relevant. Three different gates are responsible for regulating information in a cell:

- **input gate** collects data from the current input and compares it with the previous one, applying a sigmoid activation function
- **forget gate** decides which information is to be kept, the sum of the previous (hidden) state and the current state are passed onto a sigmoid activation function, which discriminates
- **output gate** provide the output data for the following layer of the network

Dataframes We retrieved the dataset from http://www.piano-e-competition.com/piano e-competition as the *midi* files are suitable for our project and easily scrapeable from the linked website. Initially our main concerns regarded the *key* of each file

Input formatting Interfacing the data we retrieved with our model starts from the concept of multivariate normal distributions, namely an interpretation of probability distribution which teaches a neural network to produce probabilites. In order to achieve the prepended scope we read our output as parameters of the probability distribution, namely mean value and from the covariance we extract the original variance of the data. It is vital, in order to fit the model to our data, to preserve the **time dependency** of the input when splitting into training and test data. Walk forward validation is also a technique used to cumulatively valdate out test set by appending the test set output to the training data and re-train.

Sequences Dimentionality is defined by the characteristic of the data at any given point in time, for example, if more than one note is played at a specific time step, then we will have to format our input data as multi-dimentional symbolic sequence. Then we have to define our input data either as a single sequence or a set of sequences. Feature extraction would allow to convert them to classical machine learning instances, but we are interested into modelling the data as they are. What is the probability of having this sequence X from the learned ones. Break the sequence into notes and treat them as random variables, the joint distribution over these notes is kinda difficult to collect as the number of combinations of length —X— over a whole song grows exponentially. So what we do is factorization of the data, given the conditional probability of two notes we can draw the joint probability using the chain rule (conditional probability * marginal probability of given note). Using the chain rule of probability helps compute the joint probability of this specific sequence. We then use a language model to predict the following instance to appear thanks to its understanding of the underlying grammatical and semantic forms of the songs. Markov models are a first approach we can take on these sort of data, by making what is called teh **Markov assumption**, which implies a limited memory, so that we can pretend that the current note depends only on the previous (let's say) 2 notes, so our model becomes a *2nd order Markov model.* What we did so far was using bigrams, hence our model is a *1st order Markov model* Loop starting with a small sequence, predict the next word using the proability distribution given by the chain rule and **sample** from all the ones with probability ¿ zero, add the word to the seed and loop again. Multiply probabilites will underflow the result pretty quickly so instead we take the sum of the logarithm of each probability.

EXAMPLE
on a sequence A - B - C - A - D - F - A , we can pretend the note C depends on A and B, and calculate their probabiliy as:

$$p(C|A,B) = \frac{\#('ABC')}{\#('AB')}$$

Laplace Smoothing
We want to add pseudo-observations to our formula in order to correct the estimation and avoid having zero probabilities, the upper formula then becomes:

$$p(C|A,B) = \frac{1 + \#('ABC')}{|V^{order}| + \#('AB')}$$

where the *order* is the numeric value we assigned with the Markov assumption and is meant to retain the sum of probabilities to 1.
why this does not work As we mentioned earlier it is important to be able to encode semantic meaning to our data (what is the semantic meaning of notes??), for the case of words, we would need to implement embedded vectors, which in turn will encode the relation between similar words. These meaning are drawn from the presence of this similar word in the same context.

from categorical to numerical features

investigating loss curves

autoencoders and their variations this is just a filler text **truncated backpropagation through time**
minibatch is used to optimize backpropagation and calculate the backward pass in batches so that is much more efficient

1 Previous work

2 Results

In this section we describe the results.

3 Conclusions

We worked hard, and achieved very little.