Cross-modal Sound Mapping Using Deep Learning

Ohad Fried
Princeton University
Department of Computer Science
ohad@cs.princeton.edu

Rebecca Fiebrink
Princeton University
Department of Computer Science
fiebrink@cs.princeton.edu

ABSTRACT

We present a method for automatic feature extraction and cross-modal mapping using deep learning. Our system uses stacked autoencoders to learn a layered feature representation of the data. Feature vectors from two (or more) different domains are mapped to each other, effectively creating a cross-modal mapping. Our system can either run fully unsupervised, or it can use high-level labeling to fine-tune the mapping according a user's needs. We show several applications for our method, mapping sound to or from images or gestures. We evaluate system performance both in standalone inference tasks and in cross-modal mappings.

Keywords

Deep learning, feature learning, mapping, gestural control

1. INTRODUCTION

In recent years, deep learning has proven to be a very effective machine learning technique. Current deep learning methods achieve good results in tasks such as object recognition and unsupervised learning of hierarchical representations [12], as well as in music-centric tasks such as genre recognition [5] and beat detection [4]. Unsupervised or semi-supervised training of deep networks yields a set of automatically learned features. This work shows how to use such features in cross-modal mappings between sound, visuals and gestures.

The task of mapping gestures to sound and sound to visualizations is critical to computer music creation and performance. Many methods exist to create such mappings, but the feature selection and mapping process still presents practical difficulties. An expert user is required to carefully design feature vectors that represent the data, and to map such feature vectors to output parameters. This work uses automatic feature learning to remove the need for an expert user, making the feature extraction—and optionally the mapping—completely automatic. We use deep learning to translate each input and output vector to a corresponding feature vector (Section 3.1). These feature vectors represent some inherent structure within the data (e.g. features for handwriting can correspond to edges or pen strokes, depending on the depth within the deep network). The next step is to map these feature vectors to each other (Section 3.3), which is easier and more powerful than comparing the original input vectors, since we are working in an automatically-learned, lower-dimensional representation. This paper shows how to construct the deep network, as well as how to implement automatic cross-modal mapping. We demonstrate our approach on different types of data and suggest real-world uses in music.

We begin in Section 2 by discussing previous work that uses deep learning to achieve automatic feature extraction. Section 3 describes in detail the algorithm we developed to extract features and map them to each other. Section 4 outlines various use cases for our method. Section 5 explains how the results were evaluated. Section 6 discusses results, and suggests future directions of research and improvement.

2. RELATED WORK

The idea of deep learning—machine learning using multiple levels of representation—has been around for more than 20 years [10]. Training deep learning systems initially proved to be difficult, but much progress has been achieved in recent years (e.g. [7, 12]). A technical report on the subject [1] gives a good outline of the field. In music informatics, multi-layer architectures have been successfully used for tempo extraction [4], instrument classification [6], emotion detection [14] and more.

The task of meaningful feature extraction has also been tackled in the literature using deep learning. The works that are most similar to this paper are [5] and [8]. These works discuss the virtues of deep learning for automatic feature design, but do not use it for cross-modal mapping—which is the novelty in the current paper.

Automatic feature design and automatic cross-modal mapping, as explained later in this paper, can benefit interactive music systems. Systems such as [3] provide a fast prototyping mechanism for the creation of new interfaces. A shortcoming of such tools is that user expertise is still required to design an appropriate feature representation of the inputs. If the user wishes to control the sound not through synthesis parameters directly, but through "perceptual" or "abstract" control parameters as discussed in [9], he must also design an appropriate parameter abstraction. Further, existing tools cannot appropriately automate mapping creation for applications in which the user does not care much about the nature of the mapping, so long as it allows some set of outputs to be reasonably controllable from some set of inputs. This paper addresses all these problems by providing an automatic method for learning appropriate representations for inputs and outputs, and for optionally automating mappings between them as well.

3. METHOD

Our method uses a neural network topology called stacked autoencoders. An autoencoder is a type of artificial neural network with one hidden layer. In training, weights are adjusted to most accurately reproduce the input values at the output nodes. It is thus an unsupervised method that achieves dimensionality reduction. A stacked autoencoder is an extension to autoencoders, in which several hidden layers are trained sequentially. A stacked autoencoder was chosen here since it automatically learns a meaningful multi-layer representation of the data, which we can effectively use for cross-modal mappings, as we discuss below. We assume basic knowledge of neural networks and autoencoders. For a literature overview on the subject see [13]. Also, good online tutorials are available.[1]

3.1 Stacked autoencoders

In order to create a deep architecture, we can take several autoencoders and stack them on top of each other. The result is a greedy algorithm that learns the weights for each layer iteratively. Past work has shown that higher-level autoencoders can capture higher-level details, such as elaborate pen strokes for the MNIST digit classification task (Section 5.1).

The procedure for building a stacked autoencoder is illustrated in Figure 1. We start by training an autoencoder. Its input is the raw data or some basic manipulation of it (e.g. image patches for images, DFT values for sound). We then discard the output layer and use the input and hidden layers as a way to translate original inputs to level-1 features. We then repeat the process, feeding level-1 features to an autoencoder to produce level-2 features. The process can continue for as many levels as we want.

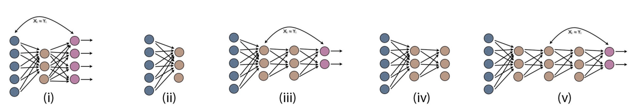

Figure 1: Building a stacked autoencoder. (i) An autoencoder. (ii) After training, the output layer is removed. Inputs are passed through the hidden layer to create compressed representations of the input. (iii) The new representations are inputs for another autoencoder. (iv)–(v) The process continues for arbitrarily many layers.

The output of the entire stacked autoencoder process is a neural network that, given some input, produces a set of features (node activations) which "capture the essence" of the input. This definition is less than formal, and we will elaborate on this point in the following sections.

3.2 Fine-tuning

Up until this point our process was completely unsupervised. We can leave it unsupervised, or add some supervised refinements, depending on the data and the goal. If the data is unlabeled, the process must stay completely unsupervised. On the other hand, consider a task of matching images with music, where both images and music are tagged (this is very common in everyday life, with the prevalence of ID3 tags for music and geotagged images). In such a case we might wish to guide our cross-modal mapping (section 3.3) to match, for example, music from some region with images from the same region. Audio and images may have missing or inaccurate tags, however, so we can further fine-tune our network to solve an inference task. For the examples above, we can train the neural network to classify images according to location. This step is performed by adding another classification layer on top of the stacked autoencoder. After achieving dimensionality reduction in an unsupervised manner (as explained in Section 3.1), we add a softmax classifier on top of the stacked autoencoder and train the combined network on the labeled input examples, which tunes the network for cross-modal mapping according to our labels. The results of such a process are presented in Section 5.

[1]http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial.

3.3 Cross-modal mapping

Having automatically generated features from data, it is time to link together two different types of data. The basic idea is simple: we take two or more sets of data (for example audio recordings and images) and train a stacked autoencoder for each. The stacked autoencoders might have the same structure but will learn different network parameters. At this point we select some subset of the nodes from each network to be our matching nodes. Reasonable choices are the entire set, some specific layer, or a combination of a few layers. For example, if we want to force our matching to be dependent on the last layer (the classification) we will add it to the matching nodes. If we would like the matching to capture some low-dimensional representation but allow deviation from the end classification, we can use one of the middle layers. In all examples presented in this paper, the middle hidden layers were used for matching.

The values of the selected matching nodes (for a given input) become our feature vectors used to represent the input. More precisely, given a set of examples $\{S_i\}$, we can feed each example to our network. Each example will yield a vector of node values $V(S_i)$. We shall mark the selected subset of these values as $V(S_i)_{[indices]}$, where *indices* is the list of matching nodes. We now have a natural way to calculate distances between examples of the two datasets, using some norm value. For example, using the standard l^2-norm the distance between a example S_i from the first dataset and example T_j from the second dataset will be: $\|V(S_i)_{[ind]} - V(T_j)_{[ind]}\|_2$

Using a norm to calculate the distance between $V(S_i)_{[ind]}$ and $V(T_j)_{[ind]}$ works for simple cases where the inputs and two networks are similar. For more complicated inputs we can use yet another neural network to learn a transformation from $V(S_i)_{[ind_S]}$ to $V(T_j)_{[ind_T]}$. The network structure is identical to our autoencoder, with one difference: instead of setting the output goal to be equal to the input, we set it to be $V(T_j)_{[ind_T]}$ while the inputs are $V(S_i)_{[ind_S]}$.

4. APPLICATION

We have presented a general framework for automatic feature extraction and cross-modal mapping. In this section we present possible applications for such a system. The examples given are all related to music and sound, but it is important to note that the method is general and not tightly coupled with audio signals.

4.1 Slideshow-to-music and music-to-slideshow

In this example we consider a user with a large collection of images and a large music library. A common task is to create a slideshow from some subset of the images. Our system can automatically match the slideshow with an appropriate soundtrack. Moreover, we can select different tracks for different parts of the slideshow, for example happy pop music for images from a party vs. soothing music for outdoor scenery images. This task cannot be fully automatic, since a user must choose the type of music she would like to accompany each image class (i.e. some people might prefer blues to accompany outdoor images, and others classical music). In detail, the entire process will be as follows:

1. The user selects k categories for music (e.g. classical,

blues, rock and pop) and for images (e.g. party, cats, outdoor and other). It is important to note here that we do not require the entire dataset to be labeled. The user can label just a few examples or use a pre-labeled database which can be completely disjoint from her own data.

2. The user specifies a matching (classical ⇔ cats, blues ⇔ outdoor, rock ⇔ party and pop ⇔ other).
3. The system is trained to detect the genre categories on songs and the type categories on images, using stacked autoencoders and fine-tuning, as specified in Sections 3.1 and 3.2. Songs and images are translated to feature vectors.
4. For each set of photos we find the best matching music.

The above process will yield a soundtrack to accompany our slideshow. It is important to note that the only input from the user is the high level categories and matching. The intermediate levels of the deep network are trained in an unsupervised manner. Only after all the layers are constructed they are fine-tuned according to the high level categories in order to direct the output towards the user's needs.

Another important observation is that the process is completely reversible. Instead of starting with a slideshow and looking for music to accompany the slideshow, one can start with some song or playlist and look for images to accompany the music (Figure 3). Also, the training can be performed on videos and not just still images. This creates a framework for audio visualization: given some audio signal we find appropriate images or videos to accompany the audio.

4.2 Gesture-to-audio

Our framework can be used to create automatic gesture-to-audio mappings. The system works as follows:

1. The user demonstrates many different gestures. Since many gestures are needed, another option would be for a computer to automatically synthesize gestures according to some model.
2. The system is trained (unsupervised) on the different gestures, as well as on different sound samples or possible outputs from a sound synthesis algorithm. The trained network is used to translate all gesture and audio data to feature vectors.
3. A mapping between gestures and sound feature vectors is created, using a third neural network.
4. For each new gesture the user supplies, we pass the example through the networks and find the closest matching sound. The result is that each input gesture corresponds to an output sound.

The above procedure learns both features and mapping in an unsupervised manner. This emphasizes the strength of unsupervised deep learning. The unsupervised setting means that higher-level properties of the gestures will control higher-level properties of the music, even without the user's involvement in specifying the mapping. The nature of these properties will vary depending on the training sets used by the system. The accompanying video demonstrates both unsupervised and supervised scenarios.

Also note that while the training is a lengthy process, once we have trained the networks, the translation from gesture to sound is instantaneous. This means that the above process can be used as a pre-processing step for a real-time music performance.

5. EVALUATION AND RESULTS

We will split the evaluation into several sections. We first evaluate the learning algorithm as a standalone component, to make sure that indeed our algorithm is capturing some relevant features of the input data. This evaluation is performed using standard datasets and inference tasks. We will evaluate the results qualitatively (i.e. how reasonable the detected features look) and quantitatively (i.e. how well we do on standard inference tasks). Later we evaluate our method of using the learned features for cross-modal mapping between two different datasets.

Table 1: Inference Results

Database	Expected Random Accuracy	Our Result
MNIST	10.00%	97.76%
STL-10	25.00%	80.19%
GTZAN	10.00%	47.01%

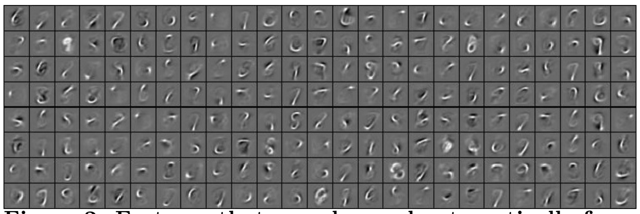
Figure 2: Features that were learned automatically from the MNIST dataset. Detected features behave as specialized edge and stroke detectors.

5.1 Standalone evaluation

Before we present our results, it is important to understand the goal of our learning algorithm. Unlike other works that strive to achieve the best inference result, we simply wish to achieve a good inference result. This good-enough result will yield features which are useful for cross-modal mappings. If we would have tuned our algorithms to achieve the best performance on a specific task, we would have lost the generality required for mapping. That being said, the authors believe that better tuning of algorithm parameters might improve the results presented below.

The MNIST dataset is a well-known benchmark for learning algorithms [11]. The dataset consists of 60,000 training examples and 10,000 test examples of handwritten digits. The task at hand is to recognize the digit (0–9). When running our stacked autoencoders on the dataset, the autoencoders can learn low-level features such as edge detectors and medium-level features such as strokes (Figure 2). This validates the correctness of our unsupervised learning method. For quantitative results of MNIST classification, as well as other datasets discussed later on, see Table 1.

The reduced STL-10 dataset [2] is comprised of 2,000 training and 3,200 test examples, with each example being a 96x96 labeled color image belonging to one of 4 classes: airplane, car, cat or dog. In the object classification task, a random choice is expected to achieve 25% accuracy. Our classifier reached an accuracy of 80.19% (Table 1).

The GTZAN dataset [15] contains 1,000 30-second audio clips, each belonging to a different music genre (blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae or rock), 100 clips for each genre. The dataset was split down the middle to create 500 train examples and 500 test examples. A random choice is expected to achieve 10% accuracy; our genre detector reached an accuracy of 47.01% (Table 1).

5.2 Cross-modal evaluation

For the cross-modal evaluation, we have implemented the audio-to-images and gesture-to-audio mapping schemes. For audio-to-images, files from the GTZAN dataset were matched with images from the STL-10 dataset. Some results can be seen in Figure 3. As can be seen in the figure, our system managed to capture differences in the genres and match them to different types of images. The matching was done by first training on both datasets separately for the tasks of genre recognition and object recognition. We then take the values of the hidden layers of the neural network, and compare between audio and images using the l^2-norm of these values. We only consider 4 genres out of the 10 so that the networks will be similar—each trying to solve an inference task of categorizing into 4 groups.

We chose to map between two standard datasets, GTZAN and STL-10, allowing for a more quantitative standalone evaluation (Section 5.1). A more common everyday use would be to match a personal music library with an image collection, which can be done, as-is, with our current implementation. Also note that the mapping was done using the simplest of methods—by comparing l^2-norms. Using a third neural network for the mapping process makes more sense, as we demonstrate in the following application.

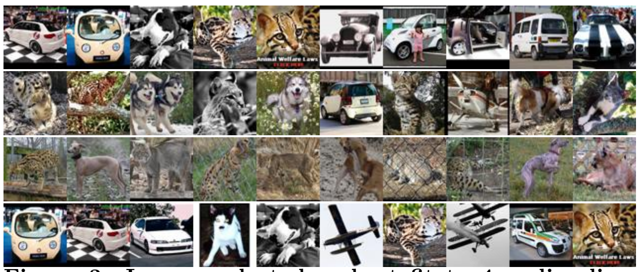
Figure 3: Images selected as best fit to 4 audio clips. Each row contains the best 10 fits for some input file. From top to bottom: blues, country, hip-hop, metal. It can be seen that blues is mostly represented by cars, country by cats and dogs, hip-hop by cats and dogs in a greener setting and metal has no unique distinction.

For a demonstration of gesture-to-music mapping, we chose to map 2D coordinates to the output of an FM synthesizer. A user's free-form mouse strokes were recorded for 7 minutes, converting mouse locations to relative polar coordinates (each location relative to the previous one), creating $\vec{\theta}$ and $\vec{\rho}$ values that are our raw gesture data. On the audio side, 5000 1-sec sound snippets were created using FM-synthesis with randomly selected parameters. Single-sided FFT values of the audio were used as the input representation. We fed each type of data to a 2-layer stacked autoencoder. All values from both hidden layers were used as our feature vectors, and were fed to a third neural network to create gesture-to-music mapping. Gesture-to-music results can be seen and heard in the accompanying video.

It is important to note that during this process no user intervention was required. Random sound snippets are assigned to random gestures, yet our method creates an instrument that "makes sense", since similar strokes yield similar sounds. Greater user control over this type of mapping is also possible, for example by matching different gesture categories with different sound timbres.

6. DISCUSSION AND FUTURE WORK

In this paper we have presented a fully automatic system for cross-modal mapping. The system can be unsupervised or use high-level cues from the user in order to create the mapping. Several different architectural choices were presented and discussed. We suggest teaser applications for our system, knowing that many more are possible.

There are many directions for future work that could be pursued. The method presented in this paper can be applied to many problems, both within the music community and outside of it. Such unexplored applications include gesture-to-speech, gesture-to-visualization and sound-to-haptic mappings. It would be interesting to implement such systems, as well as to evaluate their effectiveness through an extensive user study.

In parallel to creating more applications, the authors feel that the effect of different network architectures and parameters could be further investigated. While this work presents initial research in that direction, the number of possible architectures is enormous. It is highly likely that other deep learning architectures exist that also solve this problem, perhaps in a better way.

7. REFERENCES

[1] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[2] A. Coates, H. Lee, and A. Y. Ng. An analysis of single-layer networks in unsupervised feature learning. *Proc. Intl. Conf. Artificial Intelligence and Statistics*, 2011.

[3] R. Fiebrink, D. Trueman, and P. R. Cook. A metainstrument for interactive, on-the-fly machine learning. *Proc. NIME*, 2009.

[4] P. Grosche and M. Muller. Extracting predominant local pulse information from music recordings. *IEEE Trans. Audio, Speech and Language Processing*, 19(6):1688–1701, 2011.

[5] P. Hamel and D. Eck. Learning features from music audio with deep belief networks. *Proc. ISMIR*, 2010.

[6] P. Hamel, S. Wood, and D. Eck. Automatic identification of instrument classes in polyphonic and poly-instrument audio. *Proc. ISMIR*, 2009.

[7] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[8] E. J. Humphrey, J. P. Bello, and Y. LeCun. Moving beyond feature design: Deep architectures and automatic feature learning in music informatics. *Proc. ISMIR*, 2012.

[9] A. Hunt and M. Wanderley. Mapping performer parameters to synthesis engines. *Organised Sound*, 7(2):97–108, 2002.

[10] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, 1998.

[12] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *Proc. Intl. Conf. Machine Learning*, pages 609–616, 2009.

[13] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach, 3rd edition. *Prentice Hall*, 2009.

[14] E. M. Schmidt, J. Scott, and Y. E. Kim. Feature learning in dynamic environments: modeling the acoustic structure of musical emotion. *Proc. ISMIR*, 2012.

[15] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *Speech and Audio Processing, IEEE Transactions on*, 10(5), 2002.