Neural Networks for Digital Audio

Matthew Wear

26 May 2022

Table of Contents

3. Neural Network training
    1. Forward pass
    2. Loss function
    3. Backward pass

Part III. Neural Network Applications

1. RNNs for speech recognition
2. CNNs for audio classification
3. GANs for audio generation

Conclusion

References

Introduction

An artificial neural network, or neural network (NN), is a statistical model that consists of neurons and connections and is commonly represented as a directed graph. The field of neural networks has become predominant in machine learning with NN research and applications in academics and industry since the mid-twentieth century. With deep learning and improved computational power, neural networks are now at the core of many products and services that people around the world rely on in their daily lives, and the increasing potential of these models can answer scientific questions and become part of useful products for consumers. The purpose of this paper is to introduce the main concepts of neural networks to a general reader, explain how audio data can be used and interpreted with neural networks, and show some applications that combine neural networks and audio data in areas such as speech, music, animal noises, and environmental phenomena, to name a few. The approach of this paper is to formalize the mathematics required to understand a neural network but with an equal emphasis on the design, construction, and ethics of this technology. To what extent will audio applications using neural networks come to define the way we hear and interact with sound, music, and other things?

Part I. Audio Datasets

There are three main paradigms in classical machine learning: supervised, unsupervised, and reinforcement learning. However, as the field continues to grow, new approaches to teaching machines have become more popular. Supervised learning (SL) is the most common and the kind of learning likely found in the first few chapters of an ML textbook. Unsupervised and reinforcement learning techniques are less common but still important machine learning techniques. A few new approaches have roots in training neural networks for digital audio, including self-supervised learning and transfer learning.

1. Supervised Learning

The field of supervised learning deals with algorithms that find models with low loss on input data. SL finds a model $h \in H$, where H is the set of all possible models, which minimizes the loss function L on the training data D. What sets supervised learning apart from unsupervised learning is the inclusion of set output labels Y in training the model with input data X, i.e. a training point is $(x, y) \in D$. The set of labels Y can be a binary, multivariate, or continuous random variable. For the discrete random variable output, the model is a classifier. In the latter, the model is a regression that outputs a real number.

Training, validation, and test sets

One important step in supervised learning is the choice of splitting the data *D* into training and testing data sets. A normal split would be 80% training, 20% test. Here, we will add one more type of set that is useful for fine-tuning neural network parameters, called a validation set. It is common to split the entire dataset *D* into 80% training, 10% validation, and 10% test sets. Now, we will consider the purposes of each type of dataset in solving a supervised learning task.

The training set, sometimes shortened as "xTr" and "yTr" in code for the *X* and *Y* sets reserved for training the model, are used to learn a model *h* using a defined algorithm *A*. The data for a training set, in addition to the validation or test sets, is drawn independently and identically from the same probability distribution $\mathbb{P}$.

The validation set is a small portion of the data that is used to evaluate the performance of various parameters of the model before testing its generalized performance in the real world. When finding a model with lowest loss, a regularizer parameter λ is used to regulate the bias of the model. Bias is a term used to denote how well a model trained on one set of data, i.e., the testing set, performs on unseen data compared to the parameter that has no loss. The overall error of a model can be determined empirically.

$$\varepsilon = (E[\theta_{model} - \theta_{actual}])^2 + Var(\theta_{model})$$

The ε is the model error as the first term of the sum of bias of the model squared and the variance of the model. The process of reducing error as a balance between bias and variance is

known as the bias and variance tradeoff. Reducing this error to satisfy some existing error threshold or preference is the process of empirical risk minimization.

Other types of learning

We turn to unsupervised learning, which concerns the process of learning underlying structure in data, rather than making explicit predictions on data. The key difference between supervised and unsupervised learning is that the data set only consists of the inputs X. One reason for not having labelled data is that manually classifying or accessing the corresponding output of a given input may be expensive or not practical. Two of the most common unsupervised learning techniques are k-means clustering and principal component analysis (PCA). K-means assigns data points to one of k clusters by choosing the cluster that has the nearest cluster centroid using Lloyd's algorithm. After each iteration of recomputed the cluster centroids and assigning points, the set of clusters Z is returned, providing a reasonable inference into the variance of each class or cluster. The second common model, PCA, explains variability in data by finding the k principal components of the data set. The first principal component φ_1 is the vector in which the data points vary the most. Similarly, the second principal component explains the next most direction of variability in the data. The objective of PCA for finding the principal components that make the sample variance of the data as large as possible,

$$\max_{\|\varphi\|} (1/n)\sum_{i=1}^{n}(\varphi^{T}x_i)^2$$

The third paradigm is called reinforcement learning. Without going far into the details of constructing a reinforcement task, this type of learning specifies states that provide a reward from the reward function, and like an unsupervised learning problem, the input data does not require explicit labels. Another type of learning that does not belong to the three above, but like reinforcement learning, is used to make predictions on data without the requiring labels on the input data, is self-supervised learning (SSL). This type of learning has made significant advancements by raising the performance of models assigned to speech-to-text and other audio-based machine learning problems.

An example of this is Facebook AI's wav2vec model, which developed an SSL model for converting text-to-speech that addresses the problem of the costly and time-consuming process of transcribing non-English words for the labels corresponding to a wave file. The model is built in two layers of convolutional neural networks. The first layer encodes the wave form into a 30 ms vector representation, which is just enough time to identify characteristics of certain syllables. The second convolutional model considers the context of these syllables and transforms the input matrix of 30 ms vectors into a 1 second representation, which then selects the most consistent word. In comparison to previous Automatic speech recognition (ASR) models, wav2vec achieved a lower word error rate (WER). An SLL model learns how to predict the correct output, or in this case a word vector, through contrastive or non-contrastive feedback. Contrastive SSL uses positive and negative examples, i.e., those matching or not matching the target, with a loss function that minimizes the distance between positive examples while maximizing the distance between negative examples. On the other hand, non-contrastive SSL uses only positive examples, which may seem to invite a greater rate of false-positives, but this

approach is useful to converge to a local minimum, rather than seeking a non-biased model, i.e., $E(\theta) = \theta_{actual}$, and very low loss.

Another type of learning is transfer learning, which is another approach that has gained in popularity with some prediction problems with audio. Broadly speaking, transfer learning is the storage of knowledge of one problem and its application to a different but perhaps related problem. In machine learning, transfer learning is applied in both the training of models using synthetic or randomly generated data with real-life observational data and the application of one existing and fully-trained model to solve a similar problem.

2. Preprocessing data

Sound is a vibration in space. In physics, sound is a longitudinal wave with a vibration that travels parallel to the direction of the wave, and the displacement of the medium from changes in pressure occur in the same or opposite direction of the wave propagation. Sound waves create compression and rarefaction which are the increase and decrease of the density of the medium, respectively. The typical human range of hearing in terms of frequency ranges from twenty to twenty-thousand Hertz (Hz). Hertz is the number of cycles per second, called the frequency of a sound wave. Hearing constraints result from the complex biology of the auditory system and evolutionary processes. The auditory system is made up of the outer, middle, and inner ear, the primary auditory cortex (PAC), and other regions in the brain. The PAC transforms acoustic waves received into the ear to a variety of filters, including time and frequency decompositions, and sends these filtered signals to the corresponding voxel, a region of neurons,

in the brain that processes the encoded electrical signals that enable a person to enjoy music, engage in verbal communication, or understand sonic cues from their environment.

The goal of this section is to identify the basic attributes of digital audio and understand how auditory information is interpreted by a computer for machine learning tasks.

Digital audio

Digital audio is a discrete signal, which is a sample-based representation of a continuous signal in space-time as a sound wave. Determining the number of samples per second and the number of bits per sample are key factors in the tradeoff of accurately approximating the original signal versus the costs of computation for processing and space for storing the signal in memory.

A sound wave has three primary components: phase, amplitude, and frequency. Phase measures the location of a single cycle in either degrees or radians, amplitude measures the maximum magnitude or displacement of cycle from its resting equilibrium position often in terms of decibels, and frequency measures the time of a single cycle in Hertz (cycles per second). Another helpful metric is period, which is the time of a vibration to complete one cycle. Therefore, period T is the reciprocal of frequency f, $T = 1/f$. In the case of sound, we are measuring temporal occurrences, or the number of cycles or occurrences of an event over time, rather than spatial frequency which is measured in unit distance.

The human ear comprehends the perceived loudness of sound in a non-linear way. Sound or acoustic pressure is measured in pascals (Pa). The quietest sound that can be heard is

approximately 20μPa while the loudest sound heard before experiencing pain is 60Pa. Since this range requires a unit scale of 10^{13} divisions, a logarithmic scale is used instead, which provides the following equation for deriving a decibel level in terms of a reference sound pressure level.[1] The reference sound pressure is used because loudness, like other acoustic properties, is a root-mean-square (RMS) quantity where the mean of acoustic time-varying measurements is often 0. The formula the sound pressure level (SPL) L_p in decibels (dB) for a given pressure p is

$$L_p = 10\log_{10}(\langle p^2\rangle/p_{\text{ref}}^2) = 10\log_{10}\langle p^2\rangle - 10\log_{10}(p_{\text{ref}}^2)$$

Where $\langle p^2\rangle$ is the mean-square sound pressure for a location r and time t, given by

$$\langle p^2(r, t)\rangle = \lim_{T\to\infty}(1/\mathrm{T})\int_0^{\mathrm{T}} p(r, t)p(r, t)\mathrm{d}t$$

Which provides the root-mean-square (RMS) sound pressure,

$$p_{\text{RMS}} = \sqrt{\langle p^2(r, t)\rangle}$$

Since p_{ref}^2 is assumed to be at equilibrium at 2×10^{-5} N/m^2 or 20 μPa,

$$L_p = 10\log_{10}\langle p^2\rangle = 10\log_{10}\langle p^2\rangle + 94 \quad (\text{dB re } 20\mu\text{Pa})$$

[1] *Foundations of Vibroacoustics*, Ch. 1 (Hansen 2018)

The propagation speed of sound (m/s), or phase speed, in a solid or fluid is a function of the stiffness D and density ρ of the medium,

$$c = \sqrt{D/\rho}$$

For gases, such as atmosphere,

$$c = \sqrt{D/\rho} = \sqrt{\gamma P_s/\rho}$$

Since $D = \gamma P_s$, where γ is the ratio of specific heats and P_s is the static pressure.

$$c = \sqrt{\gamma RT/M}$$

Where R is the universal gas constant, T is temperature in Kevin, and M is molecular weight. To derive the common equation for speed of sound in air, we use the following values,

$$R = 8.314 \text{ J mol}^{-1} \text{ K}^{-1}$$

$$M_{air} = 0.029 \text{ kg/mol}$$

$$\gamma_{\text{air}} = 1.40$$

In terms of temperature and an air density of 1.206 kg/m^3,

$$c = 331 + 0.6T \qquad \text{(m/s)}$$

For a temperature of 20°C, c = 343 m/s, or approximately one mile in 4.7 seconds.

Now that we have the basic principles of acoustics, the translation of sound waves into a discrete signal can be examined more closely with frequency or spectral analysis, which is the field of separating a time-varying single into its various frequency components.

To take a waveform in the time-domain into the frequency-domain, the Fourier transform may be used to isolate each frequency. For our purposes here, we will primarily look at the Discrete Fourier Transform (DFT) since we are dealing with discrete signals, and will be using an algorithm called the fast Fourier transform (FFT). The result of a Fourier transform is a plot or graph of amplitudes as a function of each frequency.[2]

The forward transform that maps all frequencies over time to a spectrum, $X(f)$. The discrete forward transform, often referred to as the discrete Fourier transform (DFT), is

$$X(f_n) = (1/N)\sum_{k=0}^{N-1} x(t_k)e^{-j2\pi nk/N},\ n = 0, \ldots, N-1$$

The inverse transform (IDFT) obtains the original time-varying signal as a function of the amplitude of each frequency component. The discrete inverse transform is

[2] *Foundations of Vibroacoustics*, Ch. 7 (Hansen 2018)

$$x(t_k) = \sum_{n=0}^{N-1} X(f_n) e^{j2\pi nk/N},\ k = 0, \ldots, N - 1$$

The value k is the value of the sample in the time domain, while n corresponds to the nth frequency component.

Since this formula requires *K* observations in the time domain along with *N* observations in the frequency domain, the time complexity when $N = K$ is $O(N^2)$. As *N* becomes very large, the time requirements grows exponentially. To make the Fourier transform pair more efficient, the Fast Fourier transform is a number of algorithms that all achieve the same DFT matrix in $O(N \log N)$. The algorithm can originally traced to Carl Friedrich Gauss in 1805 where he developed it in part to observe the orbit of astroids, but the method was not published explicitly until 1965 when Cooley and Tukey. In Cooley and Tukey's implementation, the improvement in time complexity is done by making the 1D equation above into a 2D equation and simplifying some of the convolutional multiplications as inner products.

Feature extraction

It is often useful to interpret audio files in ways other than the raw data of discrete samples. Take for instance human sight, in which visual objects supplement our understanding of their sound. In a similar way, inputs to machine learning models can incorporate supplemental data. In the field of transfer learning, we can even consider generative or synthetic data to the

problem at hand. There are many useful attributes of audio, some of which can be easily computed from the wave form in both the time and frequency domains.

The methods of feature extraction have made large contributions toward the field of music information retrieval (MIR) which is the process of categorizing and storing digital music files.

The following low-level features only require the sample values in the time domain. The zero crossing rate (ZCR) is the number of times a signal crossing the 0-line, i.e., from a positive to a negative value, in a given time frame. Usually ZCR and other low-level features are measured with a 1 second time frame. ZCR is useful in applications of distinguishing music and speech, determining the noisiness of a signal, or for approximating the dominant frequency in a signal.

The root mean square (RMS) energy is the square root of the mean of all time samples squared. RMS is often used for measuring the loudness of a time frame or perform higher level tasks, including audio event detection, segmentation, and tempo or beat detection.

The low-energy rate is the number of frames or samples below a certain energy threshold over the total number of frames or samples in the timeframe. In other words, it is the percentage of samples below this threshold measured in RMS energy. The low-energy rate often uses 1 second windows and the mean energy of all windows.

Spectral flux is the squared differences in the frequency distribution between two successive time frames. In this sense, it is a measure of the local rate of change in the spectrum. If there is a high difference in the spectral distribution, the spectral flux is high.

The spectral centroid is the mean frequency of a frequency distribution. It is the balancing point or center of gravity of the spectrum such that the sum of all frequency energies above the spectral centroid equals the sum of all energies below. This measure can be used to infer brightness or the general shape of a spectral distribution.

The spectral rolloff is the 90th percentile of the spectral distribution and can provide a measure of skewness of the spectral shape.

Therefore, the ZCR, RMS, and low-energy rate can be computed from the signal in the time-domain, and the spectral flux, centroid, and rolloff can be found after computing the DFT matrix of the signal in the frequency domain.

In addition to low-level information about an audio file, there are established sets of descriptors, including MPEG-7 and Mel frequency cepstral coefficients (MFCCs). A cepstrum is the inverse Fourier transform of the logarithm of the spectrum. The MPEG-7 set of spectral descriptors from the Moving Pictures Experts Group examines temporal, spectral, timbral and other qualities of an audio file.[3]

MFCCs are the result of applying the Mel scale to the logarithm of the spectrum before using the inverse Fourier transform. The Mel scale, named after the word 'melody,' is a perceptual scale of pitches judged by humans to be equally distant in pitch from one another. One popular formulation for converting a frequency in Hz into a Mel and vice versa is

$$m = 2595 \log_{10}(1 + f/700) = 1127 \ln(1 + f/700) -$$

$$g = 700 (10^{m/2595} - 1) = 700 (e^{m/1127} - 1)$$

[3] *Machine Learning Techniques for Multimedia*, Ch. 11 (Cord & Cunningham 2008)

As a reference, 1000 Mels is equal to 1000 Hz at 40 dB above the human hearing threshold. Often, the discrete cosine transform (DCT) is used instead of the IDFT for calculating MFCCs for computational efficiency improvements and between 5 to 20 MFCCs are used as features of a piece of audio, although the number can be greater. There are many methods for deriving audio features. Some are used for specific applications, including wavelet transforms that can more precisely capture the amplitude of higher frequencies, for example, than the Fourier transform, or beat histograms that measure rhythm or tempo in beats per minute (BPM), which is derived from comparing relative amplitudes of adjacent peaks and other qualities in the time domain of audio.

Part II. Neural Network Theory

The machine has developed an uncanny canniness.

— Norbert Weiner[4]

1. Machine Learning objective

Machine Learning (ML) uses computers to make predictions on data with statistical methods. The input vector, or matrix $X = x_1, x_2, \ldots, x_n,$ is a series of n observations with m dimensions for each observation x_i. The ML model generates an output vector $Y = y_1, y_2, \ldots, y_k$ with k classes.

How do we find Y from X? What is the means of evaluation of accuracy between the predicted and actual output? A model $h(x)$ maps an input vector x into an output y.

$$y = h(x)$$

$$Y_K = H(X_P),\ X_P = x_1, x_2, \ldots, x_p \text{ and } X \in \mathbb{R}^{P \times D}$$

After training a model, predictions are outputted for newly observed events in the real world. Classical computing and machine learning and other artificial intelligence (AI). Machine

[4] *God and Golem* (Wiener 1964)

learning begins with data to find models that make predictions from unobserved data. In classical computing, a manual program makes predictions from data. An ML model can be seen as a black box, since the user of the model does not directly observe the inner pieces. However, the logic of a model or neural network can be inferred by the model's architecture and and reaction to inputs.

Classification and regression

Data is either qualitative or quantitative. Qualitative data explains objects in non-numeric terms. For example, the sky is blue, vast, cloudy, etc. A random variable A is a function that inputs an element a in a sample space Ω and outputs a qualitative result b.

$$A(a) = b,\ a \in \Omega$$

For example, we might define a random variable for temperature with above 0°C corresponding to 'freezing' and below 0°C to 'not freezing'. The sample space Ω equals {'freezing', 'not freezing'}. Since the sample space has two classes, predicting the correct label with the model is binary classification. A random variable A that has this property is a Bernoulli random variable. Binary classification can have a sample space of {0, 1} or {−1, 1}. The latter can lead to clean implementations of machine learning algorithms. If we take n measurements, the temperature readings $a_1, a_2, \ldots, a_n$ are a random sample of the random variables $A_1, A_2, \ldots, A_n$. A series of n observations might look like 'not freezing', 'freezing', 'not freezing', etc. Although random variables can transform observations, A_i can be the identical value of observation i, i.e. $A_i = a_i$.

When the predictions of a model are qualitative, the process is called classification. If the data is continuous, then the process of making predictions is called regression.

A continuous variable is bounded within an interval with an infinite number of possible values. For example, when calculating the price of a house, we may be interested in qualities like the number of rooms or bathrooms, which are discrete variables. These discrete values map to a continuous value for price, $y \in R$, where R is the set of real numbers. The price can fall within $(-\infty, \infty)$, or with a minimum and maximum house price of (p_{min}, p_{max}). In other words, the model regresses price on the independent features of a house.

Probabilistic data is a third type, in addition to discrete and continuous, and gives the certainty of belonging to one class or another. In this case, the predictions follow the basic axioms of probability: $P(x_i) \in [0,1]$ and $\sum_{i=1}^{n} P(X) = 1$ for $X = x_1, x_2, \ldots, x_n$.

Statistical learning models

This section introduces basic machine learning models, similarly found in statistical learning, to provide context of the myriad of models in machine learning. We define linear classifier and kernelized linear classifiers for a supervised learning context, which is training an ML model with inputs and labels.

A. Linear regression model with maximum likelihood estimation (MLE)

$y_i \in \mathbb{R}^J$, $y_i = w^T x_i + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2)$

$y_i | x_i \sim N(w^T x_i, \sigma^2) \Rightarrow P(y_i | x_i, w) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i^T w - y_i)^2}{2\sigma^2}}$

$\hat{w}$ with MLE (maximum likelihood estimation)

$$
\begin{aligned}
w &= \operatorname{argmax}_w P(y_1, x_1, y_2, x_2, \ldots, y_n, x_n | w) \\
&= \operatorname{argmax}_w \prod_{i=1}^{n} P(y_i, x_i | w) \\
&= \operatorname{argmax}_w \prod_{i=1}^{n} P(y_i | x_i, w) \underbrace{P(x_i | w)}_{P(x_i) \text{ and constant}} \\
&= \operatorname{argmax}_w \prod_{i=1}^{n} P(y_i | x_i, w) \\
&= \operatorname{argmax}_w \sum_{i=1}^{n} \log(P(y_i | x_i, w)) \\
&= \operatorname{argmax}_w \sum_{i=1}^{n} \left[\underbrace{\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)}_{\text{constant}} + \log\left(e^{-\frac{(x_i^T w - y_i)^2}{2\sigma^2}}\right) \right] \\
&= \operatorname{argmin}_w \frac{1}{n} \sum_{i=1}^{n} (x_i^T w - y_i)^2
\end{aligned}
$$

since we remove constants and change max(-f(x)) to min(f(x)). Multiply by 1/n to get average squared error

B. Linear regression model with maximum a priori (MAP)

$\hat{w}$ from maximum a priori (MAP) estimation

Assumptions:

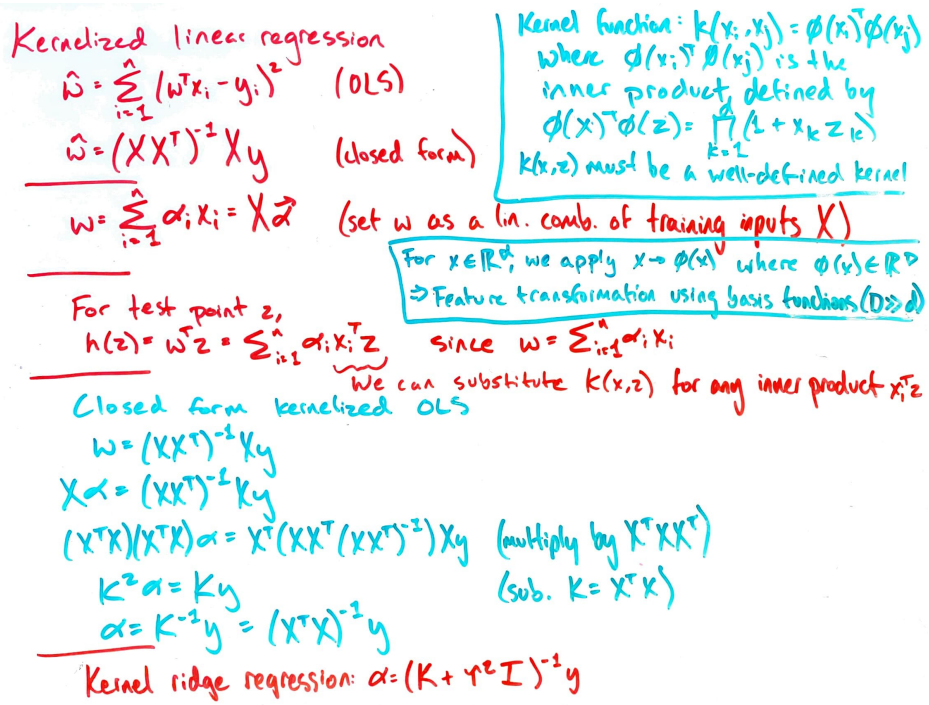
$y_i \in \mathbb{R}$, $y_i = w^T x_i + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2) \Rightarrow y_i | x_i \sim N(w^T x_i, \sigma^2)$

$\Rightarrow P(y_i | x_i, w) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i^T w - y_i)^2}{2\sigma^2}}$

Additional assumption: $w \sim N(0, \tau^2) \Rightarrow P(w) = \frac{1}{\sqrt{2\pi\tau^2}} e^{-\frac{w^T w}{2\tau^2}}$

$$
\begin{aligned}
\hat{w} &= \operatorname{argmax}_w P(w | y_1, x_1, y_2, x_2, \ldots, y_n, x_n) \\
&= \operatorname{argmax}_w \frac{P(y_1, x_1, y_2, x_2, \ldots, y_n, x_n | w) P(w)}{P(y_1, x_1, y_2, x_2, \ldots, y_n, x_n)} \\
&\vdots \\
\hat{\theta} &= \operatorname{argmax}_\theta \left[\prod_{i=1}^{n} P(y_i | x_i, \theta) \right] P(\theta) \\
&= \operatorname{argmax}_\theta \sum_{i=1}^{n} \log\left(P(y_i | x_i, \theta)\right) + \log\left(P(\theta)\right) \\
&= \operatorname{argmin}_\theta \frac{1}{2\sigma^2} \sum_{i=1}^{n} \left[(x_i^T \theta - y_i)^2\right] + \frac{1}{2\tau^2} \theta^T \theta \\
&= \operatorname{argmin}_\theta \frac{1}{n} \sum_{i=1}^{n} (x_i^T \theta - y_i)^2 + \lambda \|\theta\| \quad \text{where } \lambda = \frac{\sigma^2}{n\tau^2}
\end{aligned}
$$

C. Kernelized linear regression



Artificial neural networks

Neural networks are most often non-linear combination of linear functions.[5] A useful way to think about a neural net at first glance is through a basic architecture of one hidden layer. A hidden layer or hidden module is a set of hidden units, often referred to as neurons. Each neuron has a wire that connects to some set of inputs. Let's suppose that each input *x_i* in the input vector *X* is connected to all hidden layers. The first step of a hidden unit, or neuron, in the hidden layer is to sum all of the inputs in the previous layer that forms a connection with this hidden unit, creating a linear combination of all connected inputs. We repeat this for every single hidden unit in this layer. Suppose there are M hidden units, then we will create M linear functions. This type of layer is called a fully-connected layer, since each neuron is connected to and receives information from every neuron in the previous layer. This is also referred to as a dense layer.

[5] *The Elements of Statistical Learning*, Ch. 11 (Hastie, Tibshirani & Friedman 2017)

$$z = \sum_{i=1}^{C}(w^T x_i + b)$$

Where C is the total number of inputs connected to neuron z, w is the weight vector for each of the C inputs, and b is the bias vector. When we train the model, we will change the weights to fit our model to become closer to the true function of the data.

When we are referring to a non-linear combination, we are required to transform our data. We could simply use the identity function for this step in the hidden layer, called the activation function. We denote this function by

$$a = \sigma(z)$$

Instead we use a non-linear function to create a more dynamic or better fitting model to our data. Some of the most common activation functions that will be covered are the rectified linear unit (ReLU), the sigmoid, hyperbolic tangent, and a few others.[6] The term a is called the activated neuron and becomes the next input to the next layer. Although this could be another hidden unit somewhere in the neural net, this one-layer feed forward network connects each of the activated neurons to each of the outputs.

The output layer allows us to refine our output into a probability, a real number, or a class.

---

[6] *Deep Learning Illustrated*, Ch. 7 (Krohn 2020)

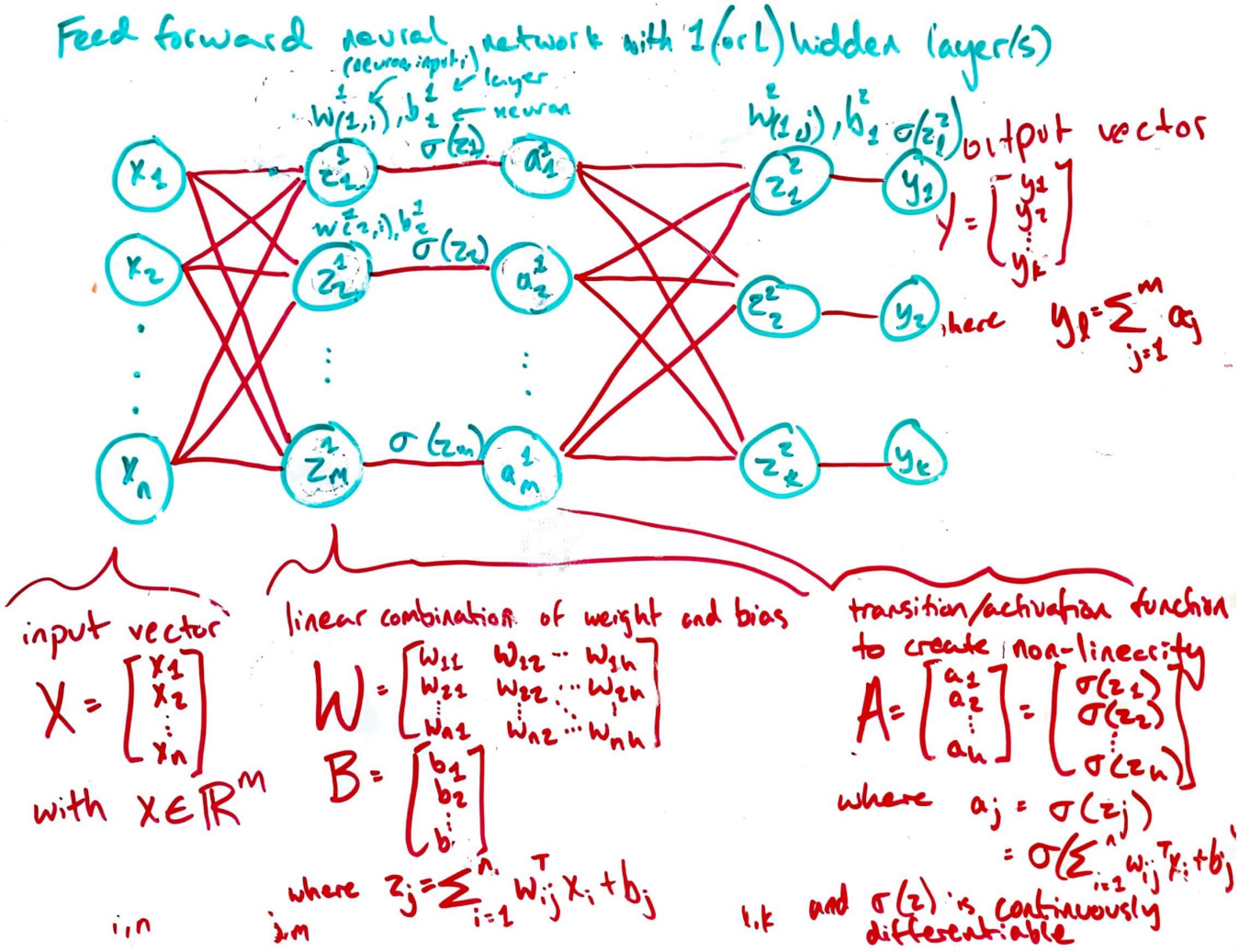


Diagram 1. 2-layer fully-connected neural network

2. Neural network components

Input and output layers

Input and output matrices, *X* and *Y,* respectively*:* $X_P = [x_1, x_2, \ldots, x_p]$, $Y_K = [y_1, y_2, \ldots, y_k]$

Hidden layers

Weight matrix *W* of a layer for a fully-connected neural network of *n* inputs (e.g., Diagram 1) and bias vector *B* of *n* linear combinations:

$W_l \in \mathbb{R}^{n \times n}$

$B_l \in \mathbb{R}^n$,

Activation functions

Activation function on input *z*: $\sigma(z)$ *where* $z = h(a) = \sum w^T x + b$

A. Identity: $\sigma(z) = z$ and $\sigma'(z) = 1$

B. Binary step: $\sigma(z) = \{0 \text{ if } z > 0; 1 \text{ if } z >= 0\}$ and $\sigma'(z) = 0$

C. Sigmoid (or logistic or soft step): $\sigma(z) = 1 / (1 + e^{-z})$ and $\sigma'(z) = f(z)\ f(1 - f(z))$

D. Tanh: $\sigma(z) = (e^z - e^{-z}) / (e^z + e^{-z}) = (e^{2z} - 1) / (e^{2z} + 1)$ and $\sigma'(z) = 1 - \sigma(z)^2 = 1 - \tanh^2(z)$

E. Rectified Linear Unit (ReLU): $\sigma'(z) = \max(0, z)$ and $\sigma'(z) = \{0 \text{ if } z < 0; 1 \text{ if } z > 0; \text{na if } z = 0\}$

F. Softmax: $S = \sigma(z) = e^z / (\sum_{j=1}^{K} e^{z_j})$, where K is the number of output classes. DS is the Jacobian matrix of partial derivatives, where $D_j S_i = \{ S_i(1 - S_j)) \text{ if } i = j; -S_i S_j \text{ if } i \neq j \}$

Model architectures

A. Deep neural network

A deep neural network (DNN) is a neural network that has many, typically at least twenty, consecutive hidden layers. It is not uncommon for a deep network to contain around a 100 layers or more to map more abstract representations of the input data.

B. Recurrent neural network

A recurrent neural network (RNN) has weights that connect a neuron to neurons in both previous and subsequent hidden layers. A RNN processes each element of an input sequence, in which the hidden units have an internal state vector that contains information about the previous elements of the input sequence.

C. Convolutional neural network

A convolutional neural network (CNN) uses a kernel window that performs a convolution on a set of inputs, making this type of architecture useful for data of multiple dimensions such as audio, images, or video.[7] The weights of the kernel window for each hidden unit are trained and the representation of the output of the hidden unit that usually passes through a batch-norm to

[7] *Understanding deep convolutional networks* (Mallat 2016)

normalize the current batch of training data and then a rectified linear unit (ReLU) as the activation function. With high dimensional data like images and audio, a pooling layer is added after the convolutional layer to reduce the dimensionality of the data. For example, a patch of values might be reduced to its max or mean value.

3. Neural network training

Forward pass

The forward pass of an input is the estimation, while the backward pass of an input is the correction of a neural network. The forward pass is therefore the prediction z_i of the present state of the model $h(x)$ on the input x_i during training.

$$h(x_i) = z_i$$

The algorithm used to implement a forward pass is the following:

procedure ForwardPass(x)
    $z_0 = x$
    for l:1:L
        $a_l = W_l z_{l-1} + b_l$
        $z_l = \sigma_l(a_l)$
    end for
return z_l
end procedure

Loss function

The loss function calculates the error of the model or how far off the predicted z was from the actual label y for the input x. The formulation of total loss of an error over n training inputs,

$$\sum_{i=1}^{n} L(h(x_i), y_i)$$

We can find average loss by dividing by the number of training inputs n. Some common loss functions for regression are the following. Each have their own advantages or disadvantages, such as the differentiability or penalty on training data with high error.

A. Squared Loss: $L = (h(x_i) - y_i)^2$

B. Absolute Loss: $L = |h(x_i) - y_i|$

C. Huber Loss: $L = \{ 1/2(h(x_i) - y_i)^2 \text{ if } |h(x_i) - y_i| < \delta; \delta|h(x_i) - y_i| - \delta/2, \text{ otherwise } \}$

D. Log-cosh Loss: $L = \log(\cosh(h(x_i) - y_i))$, where $\cosh(x) = (e^x + e^{-x})/2$

Backward pass

The second step in training a neural network after finding the predicted value z of an input is to adjust the model to better fit the data, according to the gradient of the loss function. In other words, since the objection is to minimize the loss function with respect to the model parameters w, a method called gradient descent tweaks the parameter toward a local, or preferably, global minima of the loss function. Gradient descent (GD) is the essential element in any neural network training, and many versions of this algorithm are used in practice, including

Adam and Adagrad.[8] GD works by calculating the gradient, i.e. slope, of the loss function and then takes a step in the opposite direction of the highest gradient, or down the slope toward lower model error. A common version is called Stochastic Gradient Descent (SGD) which introduces a small amount of randomness into the approximation of the gradient, and although imprecise at a given step, SGD may find a better minima, since the noise in the gradient may avoid falling into a worse local minima that regular gradient descent cannot escape from.

The algorithm for the backward pass works by calculating gradients using the chain rule from the last hidden layer to the first layer of a neural network.

procedure BackwardPass(x)
    $\delta_L = \partial L / \partial z_L \odot \sigma_L'(a_L)$
    for l:L:-1:1
        $w_l = w_l - \alpha \cdot \delta_l \cdot z^{T}_{l-1}$
        $b_l = b_l - \alpha \cdot \delta_l$
        $\delta_{l-1} = \sigma_{l-1}'(a_{l-1}) \odot (w_l^{T} \cdot \delta_l)$
    end for
end procedure

[8] D*eep Learning Illustrated*, Ch. 8 (Krohn 2020)

Part III. Neural Network Applications

1. RNNs for speech recognition

Attempts to solve speech recognition started decades ago primarily with hand-engineered machines to match words to their sounds. In the 1960s, IBM Research introduced the Shoebox to process speech for simple math calculations. Now, the prevalence of voice assistants and translation apps, and text-to-speech applications for transcribing audio files for processing, storage, and captioning, brings advancements in using neural network architectures for speech recognition. An architecture by researchers at Google reached state-of-the-art performance on a benchmark data set called TIMIT for predicting phonemes, which are the units of distinguishable sounds that make up words.

Graves, Mohamed, and Hinton, developed an architecture that combined existing deep learning methods when approaching the phoneme recognition problem.[9] A recurrent neural network is useful when dealing with sequential data such as speech or text, since it takes into consideration the previous context when evaluating the current element as the input of the model. However, conventional RNNs are unidirectional, and to use future context of the sequence, the authors opted for a bidirectional recurrent neural network (BRNN). BRNNs process the data in two separate hidden layers for the previous and future context of the sequence, which are both fed into the same output layer.

---

[9] *Speech recognition with deep recurrent neural networks* (Graves, Mohamed & Hinton 2013)

Long short-term memory (LSTM) networks are a version of RNNs with improved memory. This type of network has memory cells for storing and finding significant information in the outer reaches of the sequence. Another technique that was employed by the authors was the use of multiple RNNs stacked together, where the output of one RNN becomes the input for the next RNN. As with other implementations of deep learning, using more layers or neural networks on top of one another can provide higher representations of the objects that are classified, which is particularly useful for abstracting momentary acoustic data into a full string of phonemes that make a word.

As input, the audio data was encoded as an input vector of $D = 123$ dimensions with 40 Mel frequency cepstral coefficients (MFCCs), RMS energy, and the vectors' first and second derivatives over time. The input data was normalized such that each of the d features had a mean of 0 and unit variance, i.e., $\mu_j = 0$ and $\sigma_j = 1$ for $j \in 1, \ldots, D$.

The network uses a method called Connectionist Temporal Classification (CTC). CTC has a softmax layer that defines a probability distribution over all of the K classes with $P(k \mid t)$, i.e., phonemes, in addition to the empty set $\varnothing$. The most likely class or non-result is the output of the network.

Through experimental results on various RNN architectures, the authors found that a deep bidirectional LSTM RNN network with CTC end-to-end training was had the highest accuracy on the TIMIT test set. To avoid overfitting the model, the network added a Gaussian weight noise of $\sigma = 0.075$. For the back propagation, the model learned weights through SGD

with a learning rate of 10^{-4} and a momentum of 0.9 with a random initialization of values uniformly chosen in [−0.1, 0.1].

2. CNNs for audio classification

Although recurrent neural networks have been predominantly encouraged for audio signals, because of their ability to store information from previous or future contexts, convolutional neural networks are used extensively for audio-based machine learning tasks. CNNs originally came into focus with Le Cun et al. paper that demonstrated the effectiveness of CNNs for accurately transcribing handwritten digits from the MNIST data base.[10] The model was then tested in practice to automatically sort letters for the United States Postal Service through handwritten zip-code recognition.

With the success of CNNs for images and video, researchers began to apply this architecture, where each hidden unit learns distinctive features that become activated in the form of a feature map. For example, in a deep CNN, the first few layers may recognize specific phonemes that distinguish the training data, and in later layers, the feature maps would generate higher representations such as syllables, words, or phrases. Despite RNNs reputation for being the "default" neural network for audio problems, CNNs are competitive and can outperform RNNs when they are built to store information about the other elements in the sequence.[11]

---

[10] *Handwritten Digit Recognition with a Back-Propagation Network* (Le Cun et al. 1990)

[11] *An empirical evaluation of generic convolutional and recurrent networks for sequence modeling* (Bai, Kolter & Koltun 2018)

Hershey et al. directly modified state-of-the-art image classification CNN models for processing audio.[12] They sought to find whether these models could accurately label audio data in a data base of 100 million Youtube videos, called Youtube-100M. They preprocessed the data into non-overlapping 960 ms frames. The log Mel spectrogram of each frame was computed and separated into 64 Mel-spaced frequency bins.

The model used a cross entropy loss function, commonly used for categorical or multinomial problems, with a sigmoid output function instead of softmax because the classes are not mutually exclusive, i.e., each audio input can belong to multiple classes. The modified image classifiers, including AlexNet, VGG, and Inception V3, labelled audio better than a fully-connected neural network built for audio classification.

3. GANs for audio generation

The field of generative adversarial networks (GANs) is a generative model that pins a generator against a discriminator. The generator attempts to convert random noise into observations that could have reasonably been sampled from the dataset. The discriminator predicts whether the input is a genuine sample from the dataset or the output of the generator.[13] GANs are trained in a cycle of these two models, and the network is built to incorporate information from each, such that the generator finds new methods to fool the discriminator and the discriminator becomes more adept at predicting real versus forged samples.

---

[12] *CNN architectures for large-scale audio classification* (Hershey et al. 2017)

[13] *Generative Deep Learning*, Ch. 4 (Foster 2019)

In a paper that aimed to build an unsupervised model for generating audio samples, Donahue, McAuley, and Puckette created WaveGAN, a GAN to synthesize 1 second audio clips to be used for sound effect generation.[14] Their model provides a solution to Foley artists and musicians who can generate short clips of audio by defining variables in a latent space of audio. At first, the sound can be specified in broad steps, but further details can be provided to create sounds, including drum samples, speech, piano, and bird calls. For example, the authors propose that the model can identify "footsteps" in the latent space but additional latent variables can be provided to generate a "large boot on gravel path" sound.

WaveGAN follows the formulation of other GAN models with a G converting noise or specified variables into an audio file and a discriminator D returning 0 or 1 if the input is predicted as real or generated, respectively. G attempts to minimize the following value function, while D maximizes,

$$V(D, G) = \mathrm{E}_X[\log(D(x))] + \mathrm{E}_Z[\log(1 - D(G(x)))]$$

The construction of WaveGAN resembles the DCGAN model for images that uses transposed convolutions to upsample feature maps into a higher resolution output. Since DCGAN outputs 64x64 pixel images, which equals a total of 4096 samples, the authors added a layer that upsamples the output to 16384 samples, which is approximately one second of audio at 16 kHz. This sample rate is sufficient for the proposed purpose of the model for sound effects and voice command generation.

---

[14] *Adversarial audio synthesis* (Donahue, McAuley & Puckette 2019)

For speech, they test the intelligibility of the output of WaveGAN of spoken digits with human listeners. The authors also measured the Inception score, which is the output of a pre-trained Inception classifier used to determine the diversity and semantic discriminability of generated outputs like audio or images.

The authors similarly trained a model with spectrogram images, called SpecGAN. However, human listeners preferred the sound of wave files generated by WaveGan even though SpecGAN achieved a higher Inception score. The authors intend to expand the WaveGAN model to variable audio lengths and additional methods for specifying conditional labels on the generated images.

Conclusion

Machine learning applications for digital audio provide inferences on physical sounds in nature. By identifying the characteristics of acoustic signals and the method in which these signals relate to vectors of discrete samples readable to a computer, the process of acquiring new information from sound may be attained with neural networks. The interest in relevant research in speech recognition, the classification of audio, and generative audio for sound effects, speech, and music provide a few avenues for which digital audio has become an integral part of field of machine learning. However, other applications and problems exist, particularly in building comprehensive networks for environmental sounds or semantics in speech or music. With more advanced networks, new interfaces and intelligent systems for communication, music, and monitoring of acoustic phenomena may introduce new relationships to sound.

This paper introduced multiple architectures of neural networks and a selection of components for each layer in a network. Deep neural networks give higher representations of sounds, recurrent neural networks can retrieve information from elsewhere in the signal, and convolutional neural networks can develop weights for local kernel windows that accurately identify subtle features in the input signal. The process of training a neural network involves the forward and backward pass, which work off one another to find more accurate model parameters that reduce the amount of error that model experiences on unseen testing data.

References

1. *Foundations of Vibroacoustics*, Ch. 1, 7 (Hansen 2018)
2. *Foundations of Vibroacoustics*, Ch. 7 (Hansen 2018)
3. *Machine Learning Techniques for Multimedia*, Ch. 11 (Cord & Cunningham 2008)
4. *God and Golem* (Wiener 1964)
5. *The Elements of Statistical Learning*, Ch. 11 (Hastie, Tibshirani & Friedman 2017)
6. *Deep Learning Illustrated*, Ch. 7 (Krohn 2020)
7. *Deep Learning* (LeCun, Bengio & Hinton 2015)
8. *Understanding deep convolutional networks* (Mallat 2016)
9. *Deep Learning Illustrated*, Ch. 8 (Krohn 2020)
10. *Speech recognition with deep recurrent neural networks* (Graves, Mohamed & Hinton 2013)
11. *Handwritten Digit Recognition with a Back-Propagation Network* (Le Cun et al. 1990)
12. *CNN architectures for large-scale audio classification* (Hershey et al. 2017)
13. *An empirical evaluation of generic convolutional and recurrent networks for sequence modeling* (Bai, Kolter & Koltun 2018)
14. *Generative Deep Learning,* Ch. 4 (Foster 2019)
15. *Adversarial audio synthesis* (Donahue, McAuley & Puckette 2019)