CHAPTER III

DEEP LEARNING

Recently, a family of machine learning research under the term *deep learning* has incurred many groundbreaking changes to the world of artificial intelligence, making the long-waited dream of the *artificial general intelligence* (AGI)[1] look not so distant in the future[2]. The impact of deep learning has been so dramatic that many successful applications of deep learning like DeepMind's AlphaGo outplaying the human Go champion and Google's neural machine translation have became familiar to the general public. The core idea of using artificial neural networks to process complex information traces back to the earliest days of computing (Kleene, 1951) but has long been considered less effective than alternative methods, such as support vector machines or probabilistic graphical models. Since around 2012, it has been increasingly shown that neural networks can substantially outperform those other approaches and have much more capability for further improvements, and that the lower performance of neural networks in the past was mostly due to insufficient data, the lack of computational power, and some numerical tricks that had not been employed before. This finding has opened the era of deep learning — a term coined after the fact that neural networks often employ multiple layers of

[1] a loosely defined term referring to human-level intelligence, i.e. an AI system that can solve complex problems in varied and possibly previously-unseen domains with self-understanding and autonomous self-control. (Goertzel & Pennachin, 2007)

[2] Machine learning researchers, according to a large survey by Grace, Salvatier, Dafoe, Zhang, and Evans (2018), believe that there is 50% chance that AI will outperform human in all tasks in the next 45 years.

learned feature transformations — and is continuing to innovate virtually all fields of science and engineering, including, of course, music technology.

This chapter reviews the essential concepts and terminologies of deep learning, from the basic architectures and techniques to the most recent advances in deep generative models. The purpose of this chapter is to present a solid foundation as well as a historical perspective for the deep learning techniques used throughout the subsequent chapters such as autoregressive waveform synthesis models and generative adversarial networks, starting from the simplest concepts. At the same time, this chapter aims to provide a concise and timely overview of the field of deep learning as a whole, rather than explaining only the specific deep learning techniques used in the later chapters.

1 Neural Network Architectures

The key idea of an artificial neural network in the simplest setting is to find an appropriate matrix W to model the relationship between variables $\boldsymbol{x}$ and $\boldsymbol{y}$, represented as real-valued vectors, so that

$$\boldsymbol{y} = \sigma(W\boldsymbol{x}) \tag{3}$$

is a good approximation, where σ is a nonlinear function like the sigmoid or the hyperbolic tangent. This model in Equation 3 is also known as a *perceptron* (Rosenblatt, 1957), one of the first artificial neural networks in history. This computation — a matrix multiplication followed by a nonlinear activation — can be applied multiple times, like

$$\boldsymbol{y} = \sigma(W_3\sigma(W_2\sigma(W_1\boldsymbol{x}))), \tag{4}$$

which gives the model more expressive power, meaning that it can learn more complex relationship in the data that the previous model could not discern, e.g. the XOR problem (Riedmiller, 1994). The model in Equation 4 is called a *multilayer perceptron (MLP)* in a sense that it is a concatenation of perceptrons, and the fact that it contains multiple layers is why these neural networks are called "deep".

A multilayer perceptron is a special case of feedforward neural networks, which refer to any computational graph that does not contain a cycle. A popular model under this category is *convolutional neural networks (CNN)* (LeCun et al., 1995), which uses a convolution (a cross-correlation, to be precise) with fixed-size kernels instead of matrix multiplications. A 2-D convolutional layer takes input arrays $X_c \in \mathbb{R}^{H \times W}$, $c \in \{1, \cdots, C\}$, and produces output arrays $Y_d \in \mathbb{R}^{H \times W}$, $d \in \{1, \cdots, D\}$. The kernels $K_{cd} \in \mathbb{R}^{K_1 \times K_2}$, $c \in \{1, \cdots, C\}$, $d \in \{1, \cdots, D\}$, and the biases $\boldsymbol{b} \in \mathbb{R}^D$ are the parameters to be optimized, and the output is calculated as:

$$Y_d[i,\ j] = \sum_{m=1}^{K_1} \sum_{n=1}^{K_2} \sum_{c=1}^{C} K_{cd}[m,\ n] X_c[i+m,\ j+n] + b_d \tag{5}$$

There are various options to this operation including whether to pad the input or trim the output of the convolution to according to the kernel size, and how much to stride the kernels while moving along the input arrays. The 2-D convolution is suitable for image data, where the initial C can be the 3 RGB channels of color images; this allows the convolutional layer to extract features that are spatially local but across all channels. For similar reasons, 1-D and 3-D convolutions are often used with time-series and video data respectively.

Using convolutional layers results in a fewer number of parameters to learn in each layer than the equivalent multilayer perceptron, allowing deeper models for the same total number of parameters. LeNet (LeCun et al., 1995) for digit

classification is what pioneered the technique of using convolutional layers in neural networks, which has become an essential building block of the majority of deep learning methods. Such models include those that surpassed the human-level accuracy in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014; Szegedy et al., 2015; He et al., 2016). A standard practice of building a CNN is to stack a multiple convolutional layers along with pooling layers to obtain a compact and hierarchical feature repesentation that is also translation-invariant, which is then fed to a multi-layer perceptron as in Equation 4 to produce output. The layers of MLP are called fully connected or dense layers, because unlike the convolutional layers, the weight matrix used in the matrix multiplication associates every pair of the input and output features. *Fully convolutional networks*, which omit the fully connected layers that are typically placed at the last stages of neural networks, do not require a fixed input and output size and are known to perform well for image segmentation (Shelhamer et al., 2017). Using the ability of deep convolutional layers to extract complex semantic information from images, many artistic applications have been developed, such as the transfer of artistic style from one image to another (Gatys et al., 2015), and a captivating transformation of images using neural network weights known as *Deep Dream* (Mahendran & Vedaldi, 2016).

A network with cyclic connections is called a *recurrent neural network* (RNN; Rumelhart, Hinton, & Williams, 1986), and has been successfully applied to modeling sequential data. Because it is hard for a recurrent neural network to propagate long-range dependencies through a chain of recurrent connections, specific recurrent units called *long short-term memory* (LSTM; Hochreiter & Schmidhuber, 1997) and *gated recurrent unit* (GRU; K. Cho et al., 2014) are devised

to resolve the problem and are considered essential for recurrent neural networks. A formulation of recurrent neural network called the sequence-to-sequence model (K. Cho et al., 2014; Sutskever et al., 2014), which can model a mapping from variable-length input to variable-length output, is well known to be very effective for machine translation, and is deployed in production in Google's translation services (Y. Wu et al., 2016). An important technique for building recurrent neural networks is *attention* (Bahdanau et al., 2015), which allows the network to focus on specific parts of a sequence for generating outputs. The attention mechanism is shown to be effective in tasks including not only machine translation as in the original paper, but also in image description generation (Karpathy & Fei-Fei, 2017), speech recognition (Chorowski et al., 2015), and question answering (Sukhbaatar et al., 2015).

Reinforcement learning is a formulation of machine learning where a software agent takes actions in an environment to maximize the reward given according to the actions (Sutton & Barto, 2018). This formulation is inspired by behaviorist psychology and is well-suited for partially-observed environments that involve delayed rewards and require explorations by the agent, such as robotics and games. *Deep Q-Network* (DQN; Mnih et al., 2015) is a neural network model designed for reinforcement learning, which has been successfully applied to automatically playing Atari games (Mnih et al., 2013) and the agent playing the game of Go that surpassed the human level of skill (Silver et al., 2016).

2 Performance Optimization Techniques

The success of deep learning was possible not only because of the architectural design of deeper models and the hardware capable of supporting such models,

but also due to the numerous elaborate techniques and clever tricks that enabled previously impossible performances.

Training a neural network involves optimization of its parameters, e.g. the weights W in Equation 3-4 and the kernels K_{cd} in Equation 5, to minimize a task-specific loss function, which is a mapping from the parameters to a real number that we desire to be as low as possible. Such optimization processes usually require the gradient of the loss function, i.e. the partial derivatives with respect to all of the model's parameters. It is feasible to manually derive the gradient for simple models, but for deep neural networks it is often too complex and error-prone to calculate the derivative by hand. For this reason, a method called backpropagation (Werbos, 1982; Rumelhart, Hinton, & Williams., 1986) was introduced based on the ideas of automatic differentiation (Linnainmaa, 1970) and revived neural network research that had been largely abandoned. The popularization of *backpropagation* in the 1980s partly contributed to the ending of the first AI winter, leading to the first commercially successful application of neural network in optical digit recognition and speech recognition. Backpropagation is still a fundamental element of deep learning, and many deep learning frameworks are capable of automatically calculating gradients using backpropagation when a compute graph is given. This enables the developer to write only the forward calculation and run the backpropagation automatically, greatly improving the productivity.

Once a gradient is known, the standard way of optimizing a neural network is to use a variant of *stochastic gradient descent* (SGD; Robbins & Monro, 1951), where the direction of the gradient descent is determined only based on a mini-batch of training data. Although using only a tiny subset of training data makes the gradient

noisy, in practice, stochastic gradient descent converges faster than batch gradient descent using the same amount of training samples. Adding momentum (Polyak, 1964; Sutskever et al., 2013) in the gradient descent optimizer has shown to be effective for finding the convergence even faster, and many schemes for applying the momentum have been introduced, such as Adagrad (Duchi et al., 2011), RMSprop (G. Hinton, 2012), Adadelta (Zeiler, 2012), and Adam (Kingma & Ba, 2015). While Adam is by far the most popular choice of optimizer, a few modification to Adam's algorithm have been proposed, including Eve (Koushik & Hayashi, 2016) using feedback from the objective function, Nadam (Dozat, 2016) incorporating Nesterov's accelerated gradient descent (Nesterov, 1983), and AMSgrad (Reddi et al., 2018) fixing a failure case of Adam where it does not converge to the optimum even in a simple convex optimization problem.

Historically, the sigmoid and the hyperbolic tangent function have been popular choices for the nonlinearity, but it is surprisingly shown (Nair & Hinton, 2010) that the *rectified linear units* (ReLU),

$$f(x) = \max\{x, 0\}, \tag{6}$$

generally improves the accuracy of deep learning models. It is also known that neural networks with ReLU activations converge faster, and are more robust to the vanishing gradient problem. A number of ReLU variants, including leaky ReLU (Xu et al., 2015), parametric ReLU (PReLU; He, Zhang, Ren, & Sun, 2015), SReLU (Jin et al., 2015), have been devised and shown to be effective in some cases.

As with any other machine learning methods, overfitting is a problem to overcome for deep learning models as well. While directly adding a L1 or L2 regularization term of weights is possible, a few clever tricks for preventing

overfitting have been devised and widely employed, and they are treated as regularization methods in a wider sense. *Dropout* (Srivastava et al., 2014) is a simple yet powerful regularization method that turns off a random subset of activations during the training process. Because the network has to learn how to make accurate predictions using only a random subset of its components, the training becomes more robust and less susceptible to overfitting. *Batch normalization* (Ioffe & Szegedy, 2015) is a method to reduce the covariance shift of activations, by performing normalization for each training mini-batch so that the activations of each layer have zero mean and unit variance. It has also been empirically shown to improve the generalizability of the trained model. Despite being relatively new, dropout and batch normalization are drop-in methods that can be added to most deep architectures with almost no changes to code and yet significantly improve the performance and are thus included almost by default in the majority of newer deep models. *Scaled exponential linear units* (SELU; Klambauer, Unterthiner, Mayr, & Hochreiter, 2017) use a special activation function that induces a self-normalizing property over layers, making the activations have zero mean and unit variance without using batch normalization explicitly.

Additionally, because typical neural networks contain thousands to millions of parameters to train, a proper initialization of the weights prior to training is important. In early days of deep learning, unsupervised pre-training of weights (Bengio et al., 2007; Erhan et al., 2010) was considered necessary, but recently it is shown that a simple random initialization of weights is sufficient with the current computational power of the hardware. A widely practiced way of initializing the weights without unsupervised pre-training is to sample from a Gaussian or uniform

distribution, scaled according to the number of input and output nodes (Glorot & Bengio, 2010; He et al., 2015).

3 Toward Deep Generative Models

Statistical models that describe how data is generated are called *generative models* and provide means of generating samples of data, either by directly modeling the data distribution or through a sampling procedure specified by the model which implicitly defines the probability distribution. This is in contrast with *discriminative models*, which can only predict the labels corresponding to the given data samples.

3.1 Traditional Generative Models

Classic examples of generative models include *naïve Bayes classifiers* (Maron, 1961) which model a conditional distribution of each feature assuming they are conditionally independent given the label and use Bayes' theorem to predict the labels. *Gaussian mixture models* (GMM; Everitt & Hand, 1981) approximates the data distribution with a mixture of multivariate gaussian distributions.

While these simple models work effectively to a certain degree with a well-crafted set of features, it is desirable to have generative models that can capture more intricate geometry of the data distribution. *Probabilistic graphical models* (PGM; Pearl, 1988) specify the structural dependencies between random variables using graphs, with nodes representing random variables and connections between them representing their dependencies. The graphs can have directed or undirected connections to formulate the joint probability distributions of the variables. *Hidden Markov models* (HMM; Rabiner, 1989) and *latent Dirichlet*

allocation (LDA; Blei, Ng, & Jordan, 2003) are special cases of directed probabilistic graphical models, also called *Bayesian networks*, and are widely used for sequence modeling and topic modeling, respectively. Undirected graphical models, also called *Markov random fields* (MRF; Kindermann & Snell, 1980), have many applications in image processing, typically by having connections between nodes corresponding to adjacent pixels. Undirected graphical models where every pair of nodes has a connection are called *Boltzmann machines* (G. E. Hinton et al., 1984) and are capable of learning internal representations of data.

3.2 Early Deep Generative Models and Autoregressive Models

Restricted Boltzmann machines (RBM; Smolensky, 1986) are simplified variants of Boltzmann machines consisting of two layers of nodes with only interlayer connections, and unlike Boltzmann machines, there exists a relatively efficient algorithm (Carreira-Perpiñán & Hinton, 2005) for training restricted Boltzmann machines. *Deep belief networks* (DBN; G. Hinton et al., 2006) are composed of multiple layers of restricted Boltzmann machines, which can be trained using greedy layer-wise optimization, and are capable of classifying hand-written digits as well as conditionally generating them. Although deep belief networks are one of the first successful deep learning methods and gave many architectural and algorithmic insights to the development of deep learning in the subsequent years, the algorithms for training DBNs are not as scalable as those for other discriminative models like stochastic gradient descent, and eventually faded away in favor of the discriminative models that runs more effectively with a larger scale of data.

An alternative method to generate data samples using a neural network is

to produce one element (i.e. one audio sample or one pixel) at a time by feeding the previous elements to the network. This approach is called an autoregressive model, and many architectures based on this idea including NADE (Larochelle & Murray, 2011), DARN (Gregor et al., 2014), RIDE (Theis & Bethge, 2015), DRAW (Gregor et al., 2015), PixelCNN/PixelRNN (van den Oord, Kalchbrenner, & Kavukcuoglu, 2016), SampleRNN (Mehri et al., 2017), WaveNet (van den Oord, Dieleman, et al., 2016), and WaveRNN (Kalchbrenner et al., 2018) are proposed and shown to be capable of generating image and audio samples. However, in addition to being inevitably slow having to repetitively run the model for every element, a drawback of autoregressive models is the difficulty of interpreting the representation, because the autoregressive model only encodes the local dependency of one sample on the adjacent elements and does not provide a compact latent representation corresponding to the global structure.

3.3 Variational Autoencoders

A straightforward method to obtain a compact latent representation from unlabeled data is to build an encoder that transforms the input data into a smaller latent dimension, followed by a decoder that maps it back to the original data. This architecture is called an *autoencoder* (Bengio, 2009), and being a deep extension to principal component analysis (PCA), it is capable of learning a nonlinear mapping for dimensionality reduction. The autoencoder architecture are shown to be effective at encoding the latent representation of data, through a few successful variants including sparse autoencoder (Ng, 2011) which produces a sparse representation of the input data, denoising autoencoder (P. Vincent et al., 2008) which is capable of reducing noise or recovering a redacted portion of an

image, and contractive autoencoder (Rifai et al., 2011) which adds a regularization term to make the model robust to slight variations of input values.

Autoencoders are not generative models in a strict sense, because, while its decoder part can produce data samples from their latent representations, it lacks the ability to randomly sample the points in the latent dimensions that corresponds to the data distribution. *Variational autoencoders* (VAE; Kingma & Welling, 2014) address this problem by restricting the posterior latent distribution to be Gaussian. This is achieved by variational inference, reformulating the evidence lower bound (ELBO) of the data log-likelihood as:

$$\log p(\boldsymbol{x}) \geq \mathcal{L}(p_\theta, q_\phi) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})), \tag{7}$$

where $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ models the encoder and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ models the decoder, and both are parameterized using neural networks which provides a flexible and differentiable family of functions. Note that maximizing $\mathcal{L}$ will maximize the first term of RHS, the log likelihood of the reconstructed data, and minimize the second term, the KL divergence between the encoded data distribution and the Gaussian prior, serving as a regularizer that induces the posterior to be Gaussian. The Gaussian prior gives the KL divergence a closed-form solution making it straightfoward to derive the derivative according to ϕ, whereas the first term contains an expectation over a distribution depending on ϕ, which disallows moving the gradient operator into the expectation and makes the stochastic gradient descent and backpropagation impossible. A reparameterization trick is used to address this issue, by setting $\boldsymbol{z} = g(\epsilon, \boldsymbol{x}) = \boldsymbol{\mu}_\phi(\boldsymbol{x}) + \epsilon \cdot \boldsymbol{\sigma}_\phi(\boldsymbol{x})$ where $\epsilon \sim \mathcal{N}(0, 1)$, which gives:

$$\nabla_\phi \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] = \nabla_\phi \mathbb{E}_\epsilon[\log p_\theta(\boldsymbol{x}|g(\epsilon, \boldsymbol{x}))] = \mathbb{E}_\epsilon[\nabla_\phi \log p_\theta(\boldsymbol{x}|g(\epsilon, \boldsymbol{x}))], \tag{8}$$

making the stochastic gradient ascent and backpropagation on the ELBO possible.

A drawback of variational autoencoders, however, is the blurriness in reconstructed images, that may come from the inexactness of the Gaussian assumption and the variational lower bound used by the model (Doersch, 2016). There have been many attempts to overcome this by allowing more flexible prior distributions (Rezende & Mohamed, 2015) as well as better latent representations (Kingma et al., 2016). VQ-VAE (van den Oord et al., 2017) uses discrete prior and posterior distributions, and is able to generate less blurry images and perform speaker conversion using raw audio. Variational autoencoders are powerful deep generative models with the advantages of having a single objective function to be optimized and thus having a stable training scheme. Despite being one of the most successful types of deep generative models to date, it remains to be seen if variational autoencoders can be extended to become a building block of an automatic music transcription system.

4 Generative Adversarial Networks

Generative adversarial networks (GAN; Goodfellow et al., 2014) are a family of deep generative models that have become extremely popular. Unlike other deep neural network models that use optimization to find the weights minimizing the loss function, GANs try to find a Nash equilibrium between its two components, the generator and discriminator. Given the training data $\boldsymbol{x} \sim p_{\text{data}}$ and the prior of latent vectors $\boldsymbol{z} \sim p_{\boldsymbol{z}}$ which typically is a multivariate Gaussian distribution, GAN performs the following minimax game:

$$\min_G \max_D \Big[\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \log D(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z} \sim p_z} \log \left(1 - D(G(\boldsymbol{z}))\right) \Big], \tag{9}$$

where the generator G learns to transform a noise vector $\boldsymbol{z}$ into a data point that can fool the discriminator as if it is a real data sample, while the discriminator D tries to correctly distinguish the output of generator $G(\boldsymbol{z})$ from the real data $\boldsymbol{x}$. Because the second expectation has a near-zero gradient where $D(G(\boldsymbol{z})) \approx 0$, i.e. the discriminator classifies the generated samples as fake, the authors suggests using a non-saturating loss for training the generator:

$$\max_{G} \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}} \log D(G(\boldsymbol{z})), \tag{10}$$

which maximizes the log-likelihood of the discriminator classifying the generated samples as real.

4.1 Evolution of the GAN Architecture

The original formulation of GAN uses neural networks, which can only be applied to simple datasets of up to 32×32 images such as MNIST, CIFAR-10, and Toronto Face Dataset. LAPGAN (Denton et al., 2015) is the first GAN formulation to generate 64×64 images, which builds upon a Laplacian pyramid of convolutional layers that conditionally generates an image that is twice larger, gradually building 64×64 images from 4×4 samples. DCGAN (Radford et al., 2015) provides a simpler method of training convolutional GANs by following a list of architectural choices, making adversarial training of 64×64 images possible only using one generator and one discriminator. Together with Improved GAN (Salimans et al., 2016) which proposed now-standard tricks such as feature matching, minibatch discrimination, historical averaging, and one-sided label smoothing to generate 128×128 images, the DCGAN architecture is employed by virtually all subsequent GAN applications. For photo-realistic images synthesis in a higher resolution,

StackGAN (H. Zhang et al., 2017a) uses a multi-stage GAN architecture to synthesize 256×256 images, and progressive growing of GANs (Karras et al., 2018) uses a training scheme that gradually switches to larger GANs to generated photo-realistic images of size 1024×1024. The GAN architecture is also applicable to 3-D (J. Wu et al., 2016) and 1-D (Donahue et al., 2018) synthesis, suitable for generation of video and audio data, respectively.

4.2 The GAN Zoo

GANs are notoriously difficult to train (Arjovsky & Bottou, 2017); its convergence is unstable because of the minimax nature of its formulation, and the generator network may simply memorize and output just a few samples of training data, an undesirable phenomena called mode collapsing. A plethora of variations of GAN have been proposed to mitigate this problem, aiming to stabilize the training and/or improve the perceptual quality of generated samples. A common approach among them is to devise a different loss function or to add a regularization term to the loss function that penalizes mode collapsing.

f-GAN (Nowozin & Cseke, 2016) is a generalization of the original GAN using a family of f-divergences in addition to the original GAN's formulation which uses Jensen-Shannon divergence. Wasserstein GAN (WGAN) (Arjovsky et al., 2017) minimizes the Wasserstein distance between the model and real distribution, and was later extended to WGAN-GP (Gulrajani et al., 2017) which uses a gradient penalty that does not require weight clipping as in Wasserstein GAN. Based on WGAN's observation that Lipschitz continuity of GAN is beneficial, spectral normalization (Miyato et al., 2018) is another technique to impose Lipschitz

continuity that is more stable and provides higher diversity in generated images than gradient penalty.

Least-Square GAN (LSGAN) (Mao et al., 2017) uses least-square losses instead, and also can be trained with gradient penalty (Mao et al., 2018) which achieves a training stability similar to that of WGAN-GP. Energy-Based GAN (EBGAN) (Zhao et al., 2017) views the discriminator as an energy function that puts low energy near the data manifold, and uses an autoencoder to model the discriminator. Boundary Equilibrium GAN (BEGAN) (Berthelot et al., 2017) extends the EBGAN architecture using Wasserstein distance to balance the generator and discriminator during training. While there are many more GAN formulations claiming to be superior than others, a large-scale empirical study (Lucic et al., 2017) suggested that the none of the popular variants actually outperforms the original GAN, provided that the hyperparameters are sufficiently optimized.

4.3 Conditional Generation and Other Applications

It is usually desirable to generate samples according to certain conditions, e.g. generating images for a specific digit. Conditional GAN (cGAN) (Mirza & Osindero, 2014) is an architecture where the generator can use the class label as well as the noise input to produce samples. Auxiliary Classifier GAN (AC-GAN) (Odena et al., 2016) is an extension to cGAN in which the discriminator can also serve as a classifier for categories of data. InfoGAN (Chen et al., 2016) can perform conditional generation in a completely unsupervised manner, i.e. without using class labels during training, by maximizing the mutual information between a subset of the latent variables and the observations.

All of the above architectures use conditional latent components

concatenated to the other feature components, which makes training a conditional GAN with a large number of classes difficult. (Miyato & Koyama, 2018) overcomes this by performing projections on the feature space of the discriminator, successfully demonstrating conditional image synthesis on the 1,000 classes of images of the ILSVRC dataset (Russakovsky et al., 2015).

Image-to-image translation models are also considered as a conditional generation problem, where the input image is the condition. Many such models have been developed using GANs with artistically interesting results. `pix2pix` (Isola et al., 2017) is trained on pairs of cross-domain images and learns to translate images between the domains, e.g. satellite images to corresponding maps, day photos to night photos, and edges of images to the original. DiscoGAN and CycleGAN (T. Kim et al., 2017; Zhu et al., 2017) are capable of performing similar tasks, but does not require paired training samples, greatly expanding the ranges of datasets that can be used for training. StarGAN (Y. Choi et al., 2017) learns to translate between more than two domains.

GAN has been successfully applied to many other tasks, including image super-resolution (AffGAN) (Sønderby et al., 2017), text-to-image synthesis (StackGAN) (H. Zhang et al., 2017b, 2017a), text generation (Boundary-seeking GAN, BGAN) (Hjelm et al., 2018), and speech enhancement (SEGAN) (Pascual et al., 2017).

4.4 Evaluation of Generated Samples

Unlike discriminative models that can be evaluated using well-defined metrics, it is not as straightforward to evaluate the performance of generative models (Theis et al., 2016). Structural similarity (SSIM) (Z. Wang et al., 2004) and its multi-scale

extension MS-SSIM (Z. Wang et al., 2003) can measure the similarity between two images using luminance, contrast, and structure information.

Another popular method for evaluating the perceptual quality of generated images is the Inception score (Salimans et al., 2016), based on the image classification model under the same name (Szegedy et al., 2016) which is a 1000-class image classifier trained on the ILSVRC dataset (Russakovsky et al., 2015). Assuming that meaningful images would have a low-entropy conditional label distribution, and a diverse set of generated images would have a high-entropy marginal label distribution, the Inception score can be obtained by feeding generated images to the Inception classifier and calculating:

$$\exp\left(\mathbb{E}_{\boldsymbol{x}}\Big[\mathrm{KL}\left(p(y|\boldsymbol{x})||p(y)\right)\Big]\right). \tag{11}$$

While Inception scores correlate well with human perception, a drawback of this is that the distribution of real data is not considered in the calculation, which is problematic for a metric measuring how realistic the generated samples are.

Fréchet Inception distance (FID) (Heusel et al., 2017) addresses this problem and provides a distance metric between the data distribution and model distribution. FID is defined using the activations of a coding layer of the Inception-v3 model, `pool_3` to be specific, as the Fréchet distance between multivariate Gaussian approximations of the two distributions:

$$d^2 = \|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|^2 + \mathrm{Tr}\left(C_1 + C_2 - 2(C_1C_2)^{1/2}\right), \tag{12}$$

where $\boldsymbol{\mu}_i$ and C_i are the mean vector and the covariance matrix of the coding layer for each distribution.

4.5 Theories on GAN Convergence

The formulation of GAN training and ways to overcome its instability have been studied from various angles. Mohamed and Lakshminarayanan (2017) generalized the GAN objective function to a wider range of implicit generative models, and Fedus et al. (2018) studied the relations of the divergence function and the equilibrium. Arjovsky and Bottou (2017) and Kodali, Abernethy, Hays, and Kira (2017) studied the training dynamics of GANs around the local equilibria and its relation to the mode collapse problem, and Daskalakis, Ilyas, Syrgkanis, and Zeng (2018) used optimistic mirror descent to reduce the cycling behavior in GAN training. Mescheder, Nowozin, and Geiger (2017) observed that the training is locally convergent when all eigenvalues of the Jacobian have negative real-part. Nagarajan and Kolter (2017) showed that with an absolute continuity assumption on the data and generator distributions ensure this case, but Sønderby et al. (2017) confirmed that this assumption does not necessarily hold in general.

Despite these advances in theoretical understanding, many of these studies are inconclusive, and there is not yet a out-of-the-box method that can make a GAN always converge. Unfortunately, the best practice at the moment is to carefully monitor the divergences and the Jacobian to determine if the training is leading to a convergence.

5 Summary

This chapter provided a comprehensive introduction to deep learning and deep generative models, with an emphasis on generative adversarial networks. The fast progress of the research on deep generative models, together with the clear trend toward data-driven MIR research as discussed in Chapter II, further motivates the

utilization of deep learning and especially deep generative models in automatic music transcription research.