Lecture notes

An introduction to deep learning for astronomy

1 Abstract

These notes are a complement to the lectures on deep learning that were given on November 2018 at the IAC XXX Winter School in Tenerife (Spain). The course consisted of 4 lectures of 1hr each, that were meant to give an introductory description of the state-of-the art deep learning techniques with a special emphasis on astrophysical applications. The purpose of these Lecture Notes is not to provide a complete description of the topics that were covered but instead to give some guidelines on how to use the online material (slides and videos). The lectures also do not enter in the whole mathematical details of the algorithms. It is closer to a global description from the user perspective. Some additional references are also provided.
Since deep learning is a very popular topic, we emphasize that there is an incredibly large amount of free online material that any interested reader should consult. An excellent review of deep learning techniques is presented in the book by Goodfellow, et al. [2015].

2 A brief introduction to classical machine learning algorithms

The first lecture provides a quick introduction to classical machine learning. The two big families of machine learning, supervised and unsupervised are presented. Classical supervised learning algorithms require a training set with known labels to be trained and can be divided into classification (objects are put in classes) and regression (the prediction is a continuous variable) algorithms. Classical supervised algorithms, specially kernel algorithms such as Support Vector Machines, are extensively covered in the lectures by Biehl.

Through a standard color-color plot used in astronomy (Slides 33 to 38), we introduce the general equation of supervised machine learning:

$$f_{\vec{W}}(\vec{x}) = \vec{y} \tag{1}$$

A supervised machine learning algorithm therefore maps a feature vector $\vec{x}$ into a previously defined label vector $\vec{y}$ through a network function f_W parametrized by some parameters (or weights) W. In the following slides, we broadly describe the three main ingredients of any supervised algorithm which are **a network function, a loss function and an optimization algorithm.**

The loss function is the quantity that is minimized in the optimization process. An example of loss function is a quadratic difference that simply measures the quadratic difference between predicted and true values (Slide 60).

The optimization (or minimization) algorithm is the technique used to minimize the loss. As detailed in the following lectures, neural networks use the gradient descent. A special emphasis is given to the concept of empirical risk (Slides 61-64). By construction, the loss is minimized over the finite number of examples that constitute the training set. All supervised machine learning algorithms assume that the training set is representative of all examples and therefore contain a strong implicit prior. It can provoke unexpected behaviors if the trained algorithms are then applied to a different dataset. This is particularly critical for astronomy applications where all datasets are heavily affected by selection effects. To mitigate the effect, any supervised machine learning algorithm needs to be tested on a completely independent dataset of the one used for training. A training of a machine learning algorithm is therefore divided into two main phases, training and testing.

The network function defines the type of *classical* machine learning algorithm (Slide 69; the function used to map the features into the labels) Supervised Learning algorithms are simplistically divided

in three main groups: decision trees that include random forests (covered by Baron), kernel algorithms among which SVMs are the most popular (covered by Biehl) and artificial neural networks which are the seed for the deep learning and are covered in these series of lectures. Some indicative elements on which algorithm to choose depending on the purpose are given in Slide 70. However there are is not an obvious way to decide which algorithm to use. It depends on the purpose and the nature of the input data.

Unsupervised algorithms do not need a training set with known labels but, instead, group data according to some similarity metric. Clustering algorithms and also more recently generative models are part of the unsupervised machine learning approaches. These are extensively covered in the lectures by Baron. In the last part of these lectures we briefly introduce deep generative models (section 6).

3 *Shallow* Artificial Neural Networks (slides)

3.1 From the perceptron to an Artificial Neural Network

We now focus on Artificial Neural Networks (ANNs). ANNs are one of the oldest machine learning algorithms. The first implementation is called the Rosenblatt Perceptron Rosenblatt [1958] and was implemented as a dedicated hardware machine back in 1957. We refer the reader to Biehl's Lecture Notes and slides for a detailed historical introduction. The basic unit of an ANN is the perceptron which is at first order based on how biological neurons are modeled. Namely a perception has several inputs which are linearly combined with some multiplicative weights and then processed through a non linear activation function. The activation function behaves similarly to a step function, i.e. if the inputs are above a specific threshold the output of the activation will be large. If not, it will be close to zero (the detailed equations can be found on slide 77). The main purpose of the activation function is to introduce the non-linear behavior in the network function.

The perceptron can be considered as the building block of a neural network. Perceptrons can be first stacked vertically to create a layer of neurons. The weight vector takes then the form of a matrix instead of a vector for a single perceptron, and there are as many outputs as neurons vertically stacked (see slide 78 for details). The output of a layer of neurons (i.e. the outputs of all the activation functions) can be then used as input of another layer of neurons (see slide 79). This combinations of horizontal and vertical stacking of perceptrons creates what we know as an ANN. There is no theoretical limit on the number of layers that can be added. The first layer is called the input layer and the last layer the output layer. All layers between the input and the output layers are called hidden layers. The final network function of an ANN with n layers can be written as:

$$p = g_n(W_n g_{n-1}(W_{n-1} ... g_1(W_1 \vec{x_0}))) \tag{2}$$

where p is the output, g_n is the activation function of layer n, W_n are the weights associated to layer n, and $\vec{x_0}$ are the input parameters. The free parameters are the weights (W). The lager the number of layers (and of perceptrons in the layers), the larger number of free parameters has the network function. In principle, a large number of free parameters allows one to cope with more complex situations. This also increases the risk of overfitting as discussed later in the lectures (an example is shown in slide 84).

3.1.1 Activation functions

Activation functions are important because they introduce the non-linear element in neural networks. There are a variety of activation functions (see slide 90) that can be used in ANNs. All of them have in common a non-linear behavior and a threshold above which the output is large (the neuron is activated). The most popular ones (in particular for deep learning applications) are ReLu and Leaky ReLu activations. This is because they have been shown to be more robust to the vanishing and exploding gradient problem in deep networks (see below). The expression for the ReLu activation function is simply:

$$ReLu(x) = max(0, x) \tag{3}$$

If the input value is positive, it simply behaves like the identity function. If it is negative it outputs 0. ReLu activation functions are typically used in hidden layers. The output layers (for classification tasks) are typically activated by *Sigmoid* activation functions:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

The main advantage of the sigmoid activation function is that the output value is always between 0 and 1. It can be directly interpreted as a probability if the distribution in training set is representative of the real distribution. This is typically very useful for classification problems. For multi-class classifications, the *Softmax* activation function is a generalization of the sigmoid function in which the sum of all output values is forced to add up to 1 (see slide 107):

$$softmax(\vec{x}) = \frac{e^{\vec{x}}}{\sum_{i=1}^{n} e^{x_i}} \tag{5}$$

3.1.2 Gradient Descent and Back Propagation algorithm

Equation 2 describes the network function of ANNs. As explained in the previous slides, the free parameters are in the weights. The goal of the optimization (or minimization) algorithm is to find the values of W that minimize the loss function. This requires to minimize an arbitrary complex non-linear function as the one shown in equation 2. Although there are many known approaches to minimize a function, ANNs typically use the simplest available algorithm called Gradient Descent, which simply iteratively follows the slope of maximum variation of the function to minimize (see slide 13 of lecture 2):

$$W_{t+1} = W_t - \lambda_t \nabla f(W_t) \tag{6}$$

where f is the function to minimize with respect to parameters W, λ_t is the learning rate at iteration (or epoch) t, W_t are the values of the weights at iteration t, and W_{t+1} are the values at iteration t+1. This minimization algorithm still requires to compute the gradient (∇f) of an arbitrary complex function. The solution to that problem is the backpropagation algorithm. This algorithm which has remained essentially unchanged since its first publication in 1986 [Rumelhart et al., 1986], reduces the problem of minimization to a local backpropagation of the derivative chain rule at the neuron level. In other words, the gradient of the full network function is never computed. Instead, only local derivatives that depend on the weights only linearly or through the activation functions are computed. The lectures do not provide a formal mathematical derivation of the backpropagation algorithm. Slides 21 to 38 of lecture 2 provide an example of the backpropagation algorithm in a simple neural network with three layers and two neurons per layer. For a more formal explanation, the reader can consult the lectures of Biehl or many of the online ressources (e.g. `https://github.com/mattm/simple-neural-network`)
An important problem of the gradient descent algorithm is that, since it is based on simple first order derivatives, it can easily converge to a local minimum. To avoid this, there are several important hyper-parameters.
One of them encodes the way the weights are initialized in the network before the first iteration (this is detailed in the last part of the lectures).
Another important point are the way the learning rates are updated. Instead of keeping λ at a fixed value for all epochs, it is updated to avoid local minima using the values of the gradient at previous iterations. There are several ways of doing this. Each method has a different name (RMSPROP, ADAM, ADAGRAD) and are considered as additional hyper-parameters of the neural network model (Optimizers in Keras). The equation of the different updating algorithms are given in slides 47 and 48 of lecture 2.
An additional element that helps preventing a convergence to local minima is called Stochastic and Batch Gradient Descent (SGD). The basic idea is that instead of computing the gradient over all the training set and update only once the weights at every epoch, the gradient is computed for every example in the training set (stochastic) or for a batch including a subset of examples. This necessarily increases the computing time since it requires more evaluations of the gradient through the backpropagation algorithm. However, by introducing some stochacity in the process, it is less

likely to fall in into a local minimum. Slide 56 shows an illustrated example of how batch gradient descent creates some random fluctuations in the objective function. Again the presentation of the batch gradient descent technique in the slides is very qualitative. For a more formal explanation, you can consult the original publications.

3.1.3 From classical to deep learning

The following slides present a general discussion on the typical input data used in *shallow* neural networks. Classical approaches before the emergence of deep learning focussed on efficiently reducing the dimensionality of raw data (pixels, spectral elements) to extract meaningful features (*feature extraction*). These features were expected to correlate with the quantity to be predicted by the learning algorithm. For example, to predict photometric redshifts, the standard approach would be to first measure fluxes of objects in different filters which are then fed to a learning algorithm. This procedure is implicitly assuming that the best quantity to predict photometric redshifts are colors. It represents however a strong dimensionality reduction. All pixels belonging to a given galaxy are reduced to one unique parameter, which is the total flux. In that particular case, all morphological signatures that could potentially contain relevant additional information are ignored. This process of feature extraction has concentrated the efforts of the computer vision community in the past years, before the emergence of deep learning. The alternative, in order to guarantee that all relevant information is effectively used, is to feed a fully connected neural network with raw data (i.e. pixels) and let the network extract the information. This first order approach is however not practical in many cases, because the large number of parameters involved. A 512×512 pixel image contains already 7×10^{10} parameters (slide 77 of lecture 2). But most importantly, a *brutal force* approach ignores all spatial correlations present in the data by treating every pixel or spectral element with an independent series of weights.This is intuitively highly inefficient because images and spectra present strong spatial correlations than can be exploited. Convolutional Neural Networks help addressing these limitations and are therefore the building blocks of deep learning.

4 Convolutional Neural Networks

4.1 Convolutions as neurons

The next slides present the basics of convolutional neural networks (CNNs). As for the previous presentations, we do not follow a formal approach but we focus on a general description. The basic idea behind deep learning is that the processes of learning and feature extraction are done simultaneously and automatically. Consider the general equation for machine learning (Eq. 1) A classical machine learning approach would determine the features $\vec{x}$ after a human engineered feature extraction procedure. The learning process then determines the weights $\vec{W}$ (see previous section). In deep learning both $\vec{x}$ and $\vec{W}$ are determined by the network. This is why deep learning is also called unsupervised features learning (not to mix with unsupervised learning).

In some sense this represents a lost in the degree of control the user has on the features used by the machine. However, it enables significantly better performance in many supervised and classification tasks. The basic tool to automatically extract features from the data is the simple discrete convolution operation, which therefore gives the name to convolutional neural networks. The mathematical expression of the convolution of 2 (2D) functions is (see slide 93 of lecture 2):

$$f(x,y) * w(x,y) = \sum_{k=-\infty}^{k=+\infty} \sum_{l=-\infty}^{l=+\infty} f(k,l) \times w(x-k, y-l) \tag{7}$$

This operation is simply a weighted sum of a kernel or filter function $[w(x,y)]$ over the input data $[f(x,y)]$. Slides 84 to 91 give a simple visual representation of the operation. A key point which is presented in slides 93-95 is that the convolution operation can be also written in the perceptron formalism presented in the previous section. In that case, the weights of the inputs are the values of the kernel (this is what the network learns) and the input parameters are the pixel values $[f(x,y)]$.

Since the same kernel is propagated over all the input data, this is equivalent to have a single neuron learning on multiple regions of the input data with a fixed size (kernel size). This simple operation solves the problem of spatial correlations explained in the previous section since the same perceptron is optimized over contiguous regions of a characteristic size. Additionally, the operation is by construction translation invariant . Independently of where in the image a given feature is, the convolution will capture it. The convolution operation also reduces the number of free parameters of the model since the same weights are shared over all the input data. Slides 92 to 102 give some visual examples of the equivalence between convolutions and shared perceptrons. Note that the convolution kernel is not restricted to 2 dimensions. The same operation can be generalized to a third dimension which in astronomy can be images from multiple filters (slide 103).

In the same way that perceptrons of ANNs can be combined vertically to create a layer of neurons, multiple convolutional kernels can also be applied to the same input data. The number of kernels (filters) that are typically applied is called the depth of the layer. A convolutional layer is therefore defined with the number of filters (depth), the 2D size of the filters (X,Y) also known as receptive field and the number of channels which is determined by the number of images in the input. A convolutional layer learns a series of filters on a fixed scale.

4.2 Pooling

Images present correlations at different spatial scales. A given convolutional layer is only sensitive to the receptive field. In order to capture larger scales and also reduce the number of parameters, the so-called pooling operations are usually included in CNNs to further reduce the number of parameters. The simplest and also the most commonly used is *maxpooling*. It simply consists on keeping the pixel with maximum value over a region defined by the user. For example a pooling of size (2,2) will reduce the input image by a factor of 4 and keep only the pixels with maximum response. Slide 125 shows other types of pooling.

A standard convolutional layer is composed of a series of convolutions, followed by an activation function (typically ReLu for deep networks) and some pooling operations. A CNN contains several convolutional layers. Some examples are shown from slides 133 to 137.

4.3 The problems of going too deep

Slides 149 to 173 present some specific problems related to the convergence of deep neural networks. Deep neural networks contain a large number of parameters. One particular problem related to this is that they can easily overfit. Overfitting simply means that the model becomes too specific to the training set and therefore has a poor performance when applied to a test set never seen before. Overfitting is easily identified by inspecting the *learning history* or *learning curve* (Slide 150). The learning history plots the value of the loss in the training and validation samples as a function of the iteration number (*Epoch*). A normal behavior for a training history is a decrease of the loss value at every iteration both for the training and validation samples. Overfitting occurs when the loss decreases for the training sample but stays constant or increases for the validation dataset.

Dropout Hinton, Srivastava, Krizhevsky, Sutskever & Salakhutdinov [2012] is an operation that helps preventing overfitting by randomly removing some units during the training phase (i.e. the weights associated to that particular neuron are not updated on a given epoch). This prevents that a neuron becomes specialized on a given feature vector and helps generalization. Detailed information on how Dropout works can be found in the reference paper cited above.

The vanishing gradient problem is a problem also related to deep networks. Since the gradient is back propagated through the different layers it sometimes can become very small or large and the training becomes unstable or does not converge. Batch Normalization (Slide 171, Ioffe & Szegedy, 2015) helps stabilizing the training and limits the effect of the Vanishing Gradient problem. The values of every batch are normalized (by removing the mean and dividing by the standard deviation) before the activation. This helps keeping reasonable values and it is thus less likely that the gradient

vanishes. As for the Dropout, Batch Normalization is only applied during the training phase.

In order to improve convergence the initialization of weights plays also an important role. One commonly adopted initialization is called He initialization (Slide 167-168). The basic idea is to set the variance of the initial weights inversely proportional to the number of weights. There are other initialization functions which are part of the hyper parameters of the networks.

4.4 Attribution techniques

Deep learning networks achieve unprecedented accuracy in supervised classification and regression tasks. The price to pay is that one looses some control into why the network takes some decisions. Slides 196 to 223 present some basic approaches to better understand the decisions taken by a CNN. These are called *attribution techniques*. We go from the most basic approach to more advanced approaches. The simplest thing to do is to visualize the kernels. This is not very indicative and it is hard to translate into an interpretable decision. We provide some scripts to code it in Keras. Using a similar approach one can also visualize the feature maps (i.e. the response of the different layers to an input image = the convolution of the kernel with the images). More advanced approaches consist on deconvolutions to identify the regions of the image which triggered maximum response in a given layer / filter. Slides 205 to 211 show some examples of the paper by Zeiler et al. [2014]. A link to GitHub repository is also provided. We also briefly present the DeepDream networks which allow one to generate the image of *maximum response* of a given class. More information can be found here: `https://deepdreamgenerator.com/`. We finally overview the attribution technique called Integrated Gradients [Sundararajan, Taly & Yan, 2017]. It is based on an integral of gradients computed on scaled versions of the input image. For Keras implementations of several attribution techniques see: `https://github.com/marcoancona/DeepExplain`.

5 Encoder-Decoders

The last lecture reviews some advanced network configurations. In particular, we focus on Encoder-Decoder networks. Those a particular kind of Fully Convolutional Neural Networks (FCNs) without fully connected layers. The output of these networks are therefore a set of features that can be rearranged as an image. They are commonly used for image to image applications such as for example image segmentation.

Standard CNNs typically reduce the size of the input images through convolutions and pooling operations (see previous sections). In order to generate an output image, the networks need to be able to increase the size. There are several ways of increasing the size. The simplest one would be to replicate the same pixel value multiple times. In the lectures we introduce the concept of transposed convolution through an example. The transposed convolution operation is broadly speaking the inverse of the convolution. We show that convolutions can be re-written in a matrix form and therefore the operation cab be inverted by taking the inverse of the *kernel matrix* (Slides 5 to 9 of last lecture). By combining convolutions with transposed convolutions we can build networks which first reduce the size (Encoder branch) and then increase to output an image with the same size as the input (Decoder Branch). An example is shown in Slide 10.

A popular network configuration used for image segmentation in particular is called the U-net [Ronneberger, Fischer & Brox, 2015]. The U-net is an Encoder-Decoder which in addition has some cross-connections between the Encoding and Decoding part. These connections help the reconstruction and have been shown to significantly increase the quality of segmentations in particular. We then show some applications of U-nets applied to astrophysics for detection of clumps in high redshift galaxies and decomposition of galaxies into bulges and disks.

6 An introduction to Generative Models

The last part of the lectures focusses on a particular type of unsupervised deep learning algorithms called Generative Models. All previous lectures were focused on supervised learning, i.e. a human (or machine) labeled sample is used for training. As explained in the lectures and lecture notes by D. Baron, unsupervised learning is a type of machine learning which does not use labeled training sets.

6.1 VAEs

The same Ecoder-Decoder architecture presented in the previous slides can be used in an unsupervised way. The most straightforward way is to use the same image as input and output. The network then simply tries to reconstruct the image it was given as input. We call such a configuration an Auto-Encoder. This is useful since it can be seen as non-linear dimensionality reduction algorithm (see lectures by D. Baron for more information of dimensionality reduction algorithms). As a matter of fact, the result of the Encoder branch is a low dimensionality representation (Latent Space) of the input data which is then used by the Decoder to reconstruct the data. A pure Auto-Encoder configuration is not tremendously useful though because there are no constraints in the way the data should be represented in the Latent Space. The network can simply learn a random position in the Latent Space for each input image. Variational Auto-Encoders (VAEs) force similar images to cluster together in the Latent Space by introducing some random noise [Kingma & Welling, 2013]. There are a large number of online ressources explaining VAEs (see 7 for an example). The VAE does not learn a fixed position in the Latent Space. Instead, it is trained to learn two numbers: an average position and a dispersion. Therefore, a given input image is not always projected into the exact same position in the Latent Space. There is an uncertainty associated to it given by the dispersion. This forces the network, in order to minimize the reconstruction loss, to project similar objects into nearby positions in the Latent Space which is what one would like for any dimensionality reduction algorithm. In the following slides, we show some applications of VAEs to astrophysics. We show for example how images of high redshift galaxies from the CANDELS survey can be projected into a lower 2D plane using VAEs with a simple quadratic loss. This results in a plane where Spheroidal galaxies cluster in one region of the plane and more disk dominated galaxies tend to be in the opposite part (Slides 33 to 39). In some sense, the VAE finds the classical Hubble Sequence in a fully unsupervised way.

6.2 GANs

The lectures finalize with a brief overview of Generative Adversarial Networks (GANs, Goodfellow, et al., 2014). GANs are a more complex generative model than VAEs. They also the state-of-the-art approach to generate realistic images (e.g. Karras, Laine & Aila, 2018). Instead of having a simple quadratic loss measuring the similarity between input and generated images, a GANs is composed of two neural networks working in an adversarial mode. A first network, called the Generator, is a Decoder CNN which generates an image from a random noise vector. The resulting generated image is fed into a normal CNN (the Discriminator) which is trained as a binary classifier to distinguish between generated and real images. The training is performed in an adversarial way, in the sense that the weights of the two networks are updated one every two epochs. One every two iterations, the weights of the Generator are updated to *fool* the Discriminator (i.e. optimized so that the Discriminator classifies as real, generated images). In the next iteration, the weights of the Generator are frozen and only the Discriminator's weights are updated so that it becomes better in distinguishing generated and real images. GANs have become very popular in the last years and have achieved unprecedented realism in 100% computer generated images [Karras, Laine & Aila, 2018]. GANs are however hard to train.

Since their first publication, several important improvements have been proposed, especially oriented to facilitate convergence. Some examples are DCGANs [Radford, Metz & Chintala, 2015] and WGANs [Arjovsky, Chintala & Bottou, 2017]. In astrophysics, they are still marginally used. In the Slides 47 to 56 we show some applications for deblending overlapping sources and deconvolution of galaxy images. We also discuss some on-going work using WGANs to detect outliers in large surveys. The adversarial training can be used to learn a representation of *normal* data. After training, when

the discriminator is confronted to an anomalous object outlier it should be able to detect it with high confidence.

7 Online Ressources

There is an overwhelming (and rapidly evolving) amount of online ressources on deep learning. Any provided list would be incomplete. I strongly recommend the readers to look online for a more complete description of the topic. These lectures are also based on available ressources (some of the figures shown are taken from these ressources as acknowledged in the first slides).

I also entertain a GitHub repository with updated slides and tutorials: `https://github.com/mhuertascompany/deeplearning4astronomy`

Some useful links (for me) are:

- Deep-Learning. Do it your self! ENS, Paris - `https://www.di.ens.fr/~lelarge/dldiy/`
- Practical Aspects from Deep Learning @ Coursera: `https://www.coursera.org/lecture/deep-neural-network/weight-initialization-for-deep-networks-RwqYe`
- Machine Learning Lectures - Thomas Keck - `https://thomaskeck.github.io/`
- EPFL Deep Learning Course - `https://fleuret.org/ee559/`
- Deep Learning @ Udacity - `https://eu.udacity.com/course/deep-learning--ud730`
- Deep Learning Course Standford - `http://cs231n.github.io/` - Karpathy
- VAEs explained: `https://jaan.io/what-is-variational-autoencoder-vae-tutorial/`

7.1 Other links indicated in the slides

- Slide 11 (L1): `https://www.slideshare.net/LuMa921/deep-learning-the-past-present-and-future-of-artificial-intelligence`
- Slide 71 (L1): `https://www.slideshare.net/mlvlc/l1-state-of-the-art-in-machine-learning`
- Slide 72 (L1): `https://chatbotnewsdaily.com/since-the-initial-standpoint-of-science-technology-and-ai-scientists-following-blaise-pascal-and-804ac13d8151`
- Slide 2 (L2): `https://medium.com/@seanswayze1/using-neural-networks-to-predict-the-2018-midterm-election-e972ccfc74a`
- Slide 22 (L2): `https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/`
- Slide 39 (L2): `http://www.emergentmind.com/neural-network`

References

Arjovsky M., Chintala S., Bottou L., 2017, arXiv e-prints, arXiv:1701.07875
Goodfellow I. J., et al., 2016, DEEP LEARNING ISBN-13: 978-0262035613
Goodfellow I. J., et al., 2014, arXiv e-prints, arXiv:1406.2661
Hinton G. E., Srivastava N., Krizhevsky A., Sutskever I., Salakhutdinov R. R., 2012, arXiv e-prints, arXiv:1207.0580
Ioffe S., Szegedy C., 2015, arXiv e-prints, arXiv:1502.03167
Karras T., Laine S., Aila T., 2018, arXiv e-prints, arXiv:1812.04948
Kingma D. P., Welling M., 2013, arXiv e-prints, arXiv:1312.6114

Radford A., Metz L., Chintala S., 2015, arXiv e-prints, arXiv:1511.06434
Ronneberger O., Fischer P., Brox T., 2015, arXiv e-prints, arXiv:1505.04597
Rumelhart, D. E., Hinton, G. E., & Williams, R. J. 1986, Nature, 323, 533
Rosenblatt 1957, Psychological Review, 65, 6, 1958, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf
Sundararajan M., Taly A., Yan Q., 2017, arXiv e-prints, arXiv:1703.01365
Zeiler et al. 2014, arXiv:1311.2901