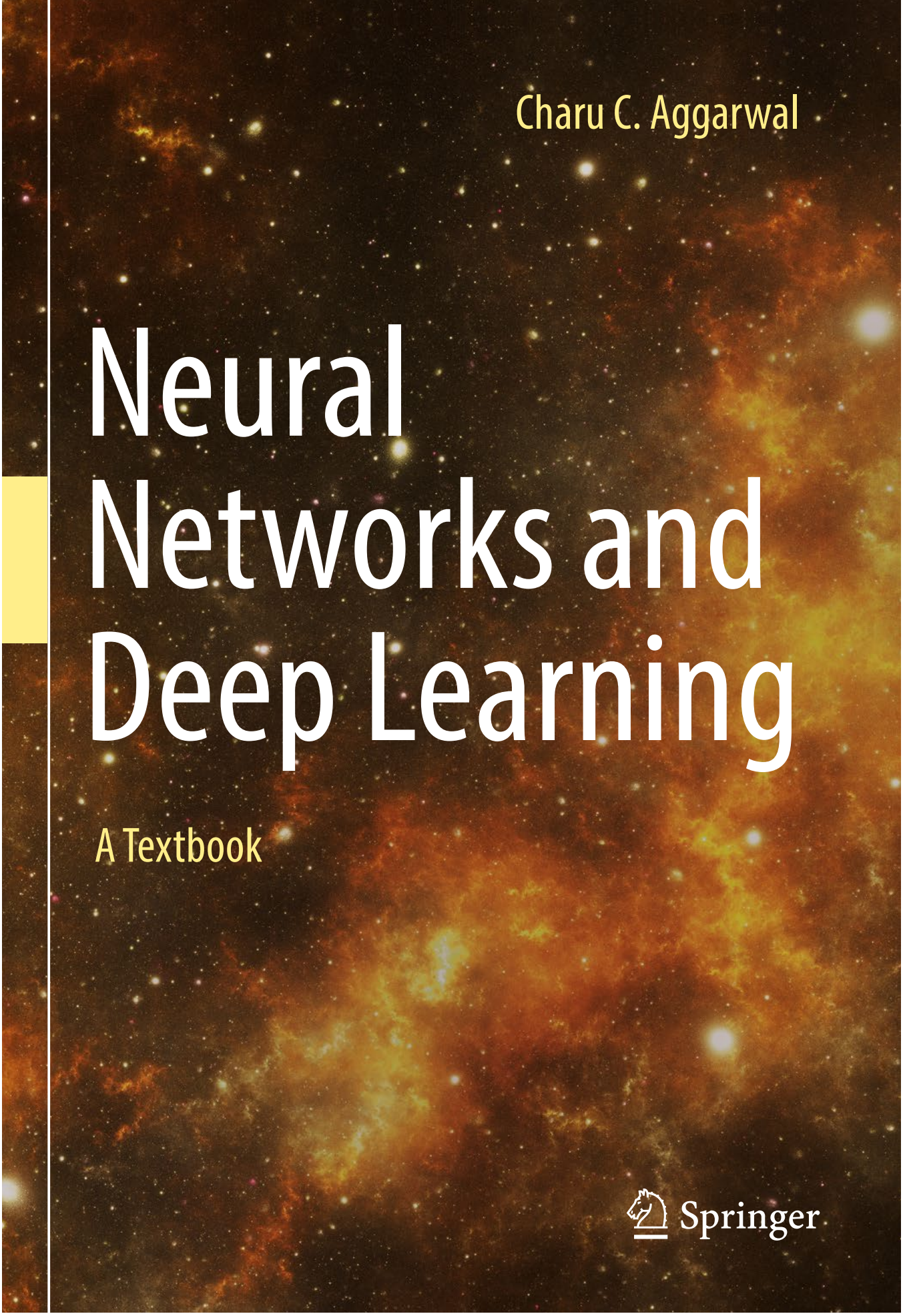
Charu C. Aggarwal
Neural Networks and Deep Learning
A Textbook
Springer

By plugging in the partial derivatives of the loss with respect to the mean and variance in Equation 3.65, we get a full recursion for $\frac{\partial L}{\partial v_i^{(r)}}$ (value before batch-normalization layer) in terms of $\frac{\partial L}{\partial a_i^{(r)}}$ (value *after* the batch normalization layer). This provides a full view of the backpropagation of the loss through the batch-normalization layer corresponding to the BN node. The other aspects of backpropagation remain similar to the traditional case. Batch normalization enables faster inference because it prevents problems such as the exploding and vanishing gradient (which cause slow learning).

A natural question about batch normalization arises during inference (prediction) time. Since the transformation parameters μ_i and σ_i depend on the batch, how should one compute them during testing when a *single* test instance is available? In this case, the values of μ_i are σ_i are computed up front using the *entire* population (of training data), and then treated as constants during testing time. One can also keep an exponentially weighted average of these values during training. Therefore, the normalization is a simple linear transformation during inference.

An interesting property of batch normalization is that *it also acts as a regularizer*. Note that the same data point can cause somewhat different updates depending on which batch it is included in. One can view this effect as a kind of noise added to the update process. Regularization is often achieved by adding a small amount of noise to the training data. It has been experimentally observed that regularization methods like *Dropout* (cf. Section 4.5.4 of Chapter 4) do not seem to improve performance when batch normalization is used [184], although there is not a complete agreement on this point. A variant of batch normalization, known as *layer normalization*, is known to work well with recurrent networks. This approach is discussed in Section 7.3.1 of Chapter 7.

3.7 Practical Tricks for Acceleration and Compression

Neural network learning algorithms can be extremely expensive, both in terms of the number of parameters in the model and the amount of data that needs to be processed. There are several strategies that are used to accelerate and compress the underlying implementations. Some of the common strategies are as follows:

1. *GPU-acceleration:* Graphics Processor Units (GPUs) have historically been used for rendering video games with intensive graphics because of their efficiency in settings where repeated matrix operations (e.g., on graphics pixels) are required. It was eventually realized by the machine learning community (and GPU hardware companies) that such repetitive operations are also used in settings like neural networks, in which matrix operations are extensively used. Even the use of a single GPU can significantly speed up implementation because of its high memory bandwidth and multithreading within its multicore architecture.

2. *Parallel implementations:* One can parallelize the implementations of neural networks by using multiple GPUs or CPUs. Either the neural network model or the data can be partitioned across different processors. These implementations are referred to as *model-parallel* and *data-parallel* implementations.

3. *Algorithmic tricks for model compression during deployment:* A key point about the practical use of neural networks is that they have different computational requirements during training and deployment. While it is acceptable to train a model for a week with a large amount of memory, the final deployment might be performed on a mobile

phone, which is highly constrained both in terms of memory and computational power. Therefore, numerous tricks are used for model compression during testing time. This type of compression often results in better cache performance and efficiency as well.

In the following, we will discuss some of these acceleration and compression techniques.

3.7.1 GPU Acceleration

GPUs were originally developed for rendering graphics on screens with the use of lists of 3-dimensional coordinates. Therefore, graphics cards were inherently designed to perform many matrix multiplications in parallel to render the graphics rapidly. GPU processors have evolved significantly, moving well beyond their original functionality of graphics rendering. Like graphics applications, neural-network implementations require large matrix multiplications, which is inherently suited to the GPU setting. In a traditional neural network, each forward propagation is a multiplication of a matrix and vector, whereas in a convolutional neural network, two matrices are multiplied. When a mini-batch approach is used, activations become matrices (instead of vectors) in a traditional neural network. Therefore, forward propagations require matrix multiplications. A similar result is true for backpropagation, during which two matrices are multiplied frequently to propagate the derivatives backwards. In other words, most of the intensive computations involve vector, matrix, and tensor operations. Even a single GPU is good at parallelizing these operations in its different cores with multithreading [203], in which some groups of threads sharing the same code are executed concurrently. This principle is referred to as *Single Instruction Multiple Threads (SIMT)*. Although CPUs also support short-vector data parallelization via *Single Instruction Multiple Data (SIMD)* instructions, the degree of parallelism is much lower as compared to the GPU. There are different trade-offs when using GPUs as compared to traditional CPUs. GPUs are very good at repetitive operations, but they have difficulty at performing branching operations like *if-then* statements. Most of the intensive operations in neural network learning are repetitive matrix multiplications across different training instances, and therefore this setting is suited to the GPU. Although the clock speed of a single instruction in the GPU is slower than the traditional CPU, the parallelization is so much greater in the GPU that huge advantages are gained.

GPU threads are grouped into small units called *warps*. Each thread in the warp shares the same code in each cycle, and this restriction enables a concurrent execution of the threads. The implementation needs to be carefully tailored to reduce the use of memory bandwidth. This is done by *coalescing* the memory reads and writes from different threads, so that a single memory transaction can be used to read and write values from different threads. Consider a common operation like matrix multiplication in neural network settings. The matrices are multiplied by making each thread responsible for computing a single entry in the product matrix. For example, consider a situation in which a 100×50 matrix is multiplied with a 50×200 matrix. In such a case, a total of $100 \times 200 = 20000$ threads would be launched in order to compute the entries of the matrix. These threads will typically be partitioned into multiple warps, each of which is highly parallelized. Therefore, speedups are achieved. A discussion of matrix multiplication on GPUs is provided in [203].

With high amounts of parallelization, memory bandwidth is often the primary limiting factor. Memory bandwidth refers to the speed at which the processor can access the relevant parameters from their stored locations in memory. GPUs have a high degree of parallelism and high memory bandwidth as compared to traditional CPUs. Note that if one cannot access the relevant parameters from memory fast enough, then faster execution does not

help the speed of computation. In such cases, the memory transfer cannot keep up with the speed of the processor whether working with the CPU or the GPU, and the CPU/GPU cores will idle. GPUs have different trade-offs between cache access, computation, and memory access. CPUs have much larger caches than GPUs and they rely on the caches to store an intermediate result, such as the result of multiplying two numbers. Accessing a computed value from a cache is much faster than multiplying them again, which is where the CPU has an advantage over the GPU. However, this advantage is neutralized in neural network settings, where the sizes of the parameter matrices and activations are often too large to fit in the CPU cache. Even though the CPU cache is larger than that of the GPU, it is not large enough to handle the scale at which neural-network operations are performed. In such cases, one has to rely on high memory bandwidth, which is where the GPU has an advantage over the CPU. Furthermore, it is often faster to perform the same computation again rather than accessing it from memory, when working with the GPU (assuming that the result is unavailable in a cache). Therefore, GPU implementations are done somewhat differently from traditional CPU implementations. Furthermore, the advantage gained can be sensitive to the choice of neural network architecture, as the memory bandwidth requirements and multi-threading gains of different architectures can be different.

At first sight, it might seem from the above example that the use of a GPU requires a lot of low-level programming, and it would indeed be a challenge to create custom GPU code for each neural architecture. With this problem in mind, companies like NVIDIA have modularized the interface between the programmer and the GPU implementation. The key point is that the speeding of primitives like matrix multiplication and convolution can be hidden from the user by providing a library of neural network operations that perform these faster operations behind the scenes. The GPU library is tightly integrated with deep learning frameworks like Caffe or Torch to take advantage of the accelerated operations on the GPU. A specific example of such a library is the *NVIDIA CUDA Deep Neural Network Library* [643], which is referred to in short as *cuDNN*. CUDA is a parallel computing platform and programming model that works with CUDA-enabled GPU processors. However, it provides an abstraction and a programming interface that is easy to use with relatively limited rewriting of code. The cuDNN library can be integrated with multiple deep learning frameworks such as Caffe, TensorFlow, Theano, and Torch. The changes required to convert the training code of a particular neural network from its CPU version to a GPU version are often small. For example, in Torch, the CUDA Torch package is incorporated at the beginning of the code, and various data structures (like tensors) are initialized as CUDA tensors (instead of regular tensors). With these types of modest modifications, virtually the same code can run on a GPU instead of a CPU in Torch. A similar situation holds true in other deep learning frameworks. This type of approach shields the developers from the low-level performance tuning required in GPU frameworks, because the primitives in the library already have the code that takes care of all the low-level details of parallelization on the GPU.

3.7.2 Parallel and Distributed Implementations

It is possible to make training even faster by using multiple CPUs or GPUs. Since it is more common to use multiple GPUs, we focus on this setting. Parallelism is not a simple matter when working with GPUs because there are overheads associated with the communication between different processors. The delay caused by these overheads has recently been reduced with specialized network cards for GPU-to-GPU transfer. Furthermore, algorithmic tricks like using 8-bit approximations of the gradients [98] can help in speeding up the

communication. There are several ways in which one can partition the work across different processors, namely hyperparameter parallelism, model parallelism, and data parallelism. These methods are discussed below.

Hyperparameter Parallelism

The simplest possible way to achieve parallelism in the training process without much overhead is to train neural networks with different parameter settings on different processors. No communication is required across different executions, and therefore wasteful overhead is avoided. As discussed earlier in this chapter, runs with suboptimal hyperparameters are often terminated long before running them to completion. Nevertheless, a small number of different runs with optimized parameters are often used in order to create an ensemble of models. The training of different ensemble components can be performed independently on different processors.

Model Parallelism

Model parallelism is particularly useful when a single model is too large to fit on a GPU. In such a case, the hidden layer is divided across the different GPUs. The different GPUs work on exactly the same batch of training points, although different GPUs compute different parts of the activations and the gradients. Each GPU only contains the portion of the weight matrix that are multiplied with the hidden activations present in the GPU. However, it would still need to communicate the results of its activations to the other GPUs. Similarly, it would need to receive the derivatives with respect to the hidden units in other GPUs in order to compute the gradients of the weights between its hidden units and those of other GPUs. This is achieved with the use of inter-connections across GPUs, and the computations across these interconnections add to the overhead. In some cases, these interconnections are dropped in a subset of the layers in order to reduce the communication overhead (although the resulting model would not quite be the same as the sequential version). Model parallelism is not helpful in cases where the number of parameters in the neural network is small, and should only be used for large networks. A good practical example of model parallelism is the design of *AlexNet*, which is a convolutional neural network (cf. Section 8.4.1 of Chapter 8). A sequential version of *AlexNet* and a GPU-partitioned version of *AlexNet* are both shown in Figure 8.9 of Chapter 8. Note that the sequential version in Figure 8.9 is not exactly equivalent to the GPU-partitioned version because the interconnections between GPUs have been dropped in some of the layers. A discussion of model parallelism may be found in [74].

Data Parallelism

Data parallelism works best when the model is small enough to fit on each GPU, but the amount of training data is large. In these cases, the parameters are shared across the different GPUs and the goal of the updates is to use the different processors with different training points for faster updates. The problem is that perfect synchronization of the updates can slow down the process, because locking mechanisms would need to be used to synchronize the updates. The key point is that each processor would have to wait for the others to make their updates. As a result, the slowest processor creates a bottleneck. A method that uses *asynchronous* stochastic gradient descent was proposed in [91]. The basic idea is to use a parameter server in order to share the parameters across different GPU processors. The updates are performed without using any locking mechanism. In other words, each GPU can read the shared parameters at any time, perform the computation, and write the

parameters to the parameter server without worrying about locks. In this case, inefficiency would still be caused by one GPU processor overwriting the progress made by another, but there would be no waiting times for writes. As a result, the overall progress would still be faster than with a synchronized mechanism. Distributed asynchronous gradient descent is quite popular as a strategy for parallelism in large-scale industrial settings.

Exploiting the Trade-Offs for Hybrid Parallelism

It is evident from the above discussion that model parallelism is well suited to models with a large parameter footprint, whereas data parallelism is well suited to smaller models. It turns out that one can combine the two types of parallelism over different parts of the network. In certain types of convolutional neural networks that have fully connected layers, the vast majority of parameters occur in the fully connected layers, whereas more computations are performed in the earlier layers. In these cases, it makes sense to use data parallelism for the early part of the network, and model parallelism for the later part of the network. This type of approach is referred to as *hybrid parallelism*. A discussion of this type of approach may be found in [254].

3.7.3 Algorithmic Tricks for Model Compression

Training a neural network and deploying it typically have different requirements in terms of memory and efficiency requirements. While it may be acceptable to require a week to train a neural network to recognize faces in images, the end user might wish to use the trained neural network to recognize a face within a matter of a few seconds. Furthermore, the model might be deployed on a mobile device with little memory and computational availability. In such cases, it is crucial to be able to use the trained model efficiently, and also use it with a limited amount of storage. Efficiency is generally not a problem at deployment time, because the prediction of a test instance often requires straightforward matrix multiplications over a few layers. On the other hand, storage requirements are often a problem because of the large number of parameters in multilayer networks. There are several tricks that are used for model compression in such cases. In most of the cases, a larger trained neural network is modified so that it requires less space by approximating some parts of the model. In addition, some efficiency improvements can also be realized at prediction time by model compression because of better cache performance and fewer operations, although this is not the primary goal. Interestingly, this approximation might occasionally *improve* accuracy on out-of-sample predictions because of regularization effects, especially if the original model is unnecessarily large compared to the training data size.

Sparsifying Weights in Training

The links in a neural network are associated with weights. If the absolute value of a particular weight is small, then the model is not strongly influenced by that weight. Such weights can be dropped, and the neural network can be fine-tuned starting with the current weights on links that have not yet been dropped. The level of sparsification will depend on the weight threshold at which links are dropped. By choosing a larger threshold at which weights are dropped, the size of the model will reduce significantly. In such cases, it is particularly important to fine-tune the values of the retained weights with further epochs of training. One can also encourage the dropping of links by using L_1-regularization, which will be discussed in Chapter 4. When L_1-regularization is used during training, many of

the weights will have zero values anyway because of the natural mathematical properties of this form of regularization. However, it has been shown in [169] that L_2-regularization has the advantage of higher accuracy. Therefore, the work in [169] uses L_2-regularization and prunes the weights that are below a particular threshold.

Further enhancements were reported in [168], where the approach was combined with Huffman coding and quantization for compression. The goal of quantization is to reduce the number of bits representing each connection. This approach reduced the storage required by *AlexNet* [255] by a factor of 35, or from about 240MB to 6.9MB, with no loss of accuracy. It is now possible as a result of this reduction to fit the model into an on-chip SRAM cache rather than off-chip DRAM memory; this also provide a beneficial effect on prediction times.

Leveraging Redundancies in Weights

It was shown in [94] that the vast majority of the weights in a neural network are redundant. In other words, for any $m \times n$ weight matrix W between a pair of layers with m_1 and m_2 units respectively, one can express this weight matrix as $W \approx UV^T$, where U and V are of sizes $m_1 \times k$ and $m_2 \times k$, respectively. Furthermore, it is assumed that $k \ll \min\{m_1, m_2\}$. This phenomenon occurs because of several peculiarities in the training process. For example, the features and weights in a neural network tend to *co-adapt* because of different parts of the network training at different rates. Therefore, the faster parts of the network often adapt to the slower parts. As a result, there is a lot of redundancy in the network both in terms of the features and the weights, and the full expressivity of the network is never utilized. In such a case, one can replace the pair of layers (containing weight matrix W) with three layers of size m_1, k, and m_2. The weight matrices between the first pair of layers is U and the weight matrix between the second pair of layers is V^T. Even though the new matrix is deeper, it is better regularized as long as $W - UV^T$ only contains noise. Furthermore, the matrices U and V require $(m_1 + m_2) \cdot k$ parameters, which is less than the number of parameters in W as long as k is less than half the harmonic mean of m_1 and m_2:

$$\frac{\text{Parameters in } W}{\text{Parameters in } U, V} = \frac{m_1 \cdot m_2}{k(m_1 + m_2)} = \frac{\text{HARMONIC-MEAN}(m_1, m_2)}{2k}$$

As shown in [94], more than 95% of the parameters in the neural network are redundant, and therefore a low value of the rank k suffices for approximation.

An important point is that the replacement of W with U and V must be done *after* completion of the learning of W. For example, if we replaced the pair of layers corresponding to W with the three layers containing the two weight matrices U and V^T and trained from scratch, good results may not be obtained. This is because co-adaptation will occur again during training, and the resulting matrices U and V will have a rank even lower than k. As a result, under-fitting might occur.

Finally, one can compress even further by realizing that both U and V need not be learned because they are redundant with respect to each other. For any rank-k matrix U, one can learn V so that the product UV^T is the same value. Therefore, the work in [94] provides methods to fix U, and then learn V instead.

Hash-Based Compression

One can reduce the number of parameters to be stored by forcing randomly chosen entries of the weight matrix to take on shared values of the parameters. The random choice is achieved with the application of a hash function on the entry position (i, j) in the matrix.

For example, imagine a situation where we have a weight matrix of size 100×100 with 10^4 entries. In such a case, one can hash each weight to a value in the range $\{1, \ldots 1000\}$ to create 1000 groups. Each of these groups will contain an average of 10 connections that will share weights. Backpropagation can handle shared weights using the approach discussed in Section 3.2.9. This approach requires a space requirement of only 1000 for the matrix, which is 10% of the original space requirement. Note that one could instead use a matrix of size 100×10 to achieve the same compression, but the key point is that using shared weights does not hurt the expressivity of the model as much as would reducing the size of the weight matrix *a priori*. More details of this approach are discussed in [66].

Leveraging Mimic Models

Some interesting results in [13, 55] show that it is possible to significantly compress a model by creating a new training data set from a trained model, which is easier to model. This "easier" training data can be used to train a much smaller network without significant loss of accuracy. This smaller model is referred to as a *mimic model*. The following steps are used to create the mimic model:

1. A model is created on the original training data. This model might be very large, and potentially even created out of an ensemble of different models, further increasing the number of parameters; it would not be appropriate to use in space-constrained settings. It is assumed that the model outputs softmax probabilities of the different classes. This model is also referred to as the teacher model.

2. New training data is created by passing unlabeled examples through the trained network. The targets in the newly created training data are set to the softmax probability outputs of the trained model on the unlabeled examples. Since unlabeled data is often copious, it is possible to create a lot of training data in this way. It is noteworthy that the new training data contains soft (probabilistic) targets rather than the discrete targets in the original training data, which significantly contributes to the creation of the compressed model.

3. A much smaller and shallower network is trained using the new training data (with artificially generated labels). The original training data is not used at all. This much smaller and shallower network, which is referred to as the mimic or *student* model, is what is deployed in space-constrained settings. It can be shown that the accuracy of the mimic model does not substantially degrade from the model trained over the original neural network, even though it is much smaller in size.

A natural question arises as to why the mimic model should perform as well as the original model, even though it is much smaller in size both in terms of the depth as well as the number of parameters. Trying to construct a shallow model on the original data cannot match the accuracy of either the shallow model or the mimic model. A number of possible reasons have been hypothesized for the superior performance of the mimic model [13]:

1. If there are errors in the original training data because of mislabeling, it causes unnecessary complexity in the trained model. These types of errors are largely removed in the new training data.

2. If there are complex regions of the decision space, the teacher model simplifies them by providing softer labels in terms of probabilities. Complexity is washed away by filtering targets through the teacher model.

3. The original training data contains targets with 0/1 values, whereas the newly created training contains soft targets, which are more informative. This is particularly useful in one-hot encoded multilabel targets, where there are clear correlations across different classes.

4. The original targets might depend on inputs that are not available in the training data. On the other hand, the teacher-created labels depend on only the available inputs. This makes the model simpler to learn and washes away unexplained complexity. Unexplained complexity often leads to unnecessary parameters and depth.

One can view some of the above benefits as a kind of regularization effect. The results in [13] are stimulating, because they show that deep networks are not *theoretically* necessary, although the regularization effect of depth is practically necessary when working with the original training data. The mimic model enjoys the benefits of this regularization effect by using the artificially created targets instead of depth.

3.8 Summary

This chapter discusses the problem of training deep neural networks. We revisit the backpropagation algorithm in detail along with its challenges. The vanishing and the exploding gradient problems are introduced along with the challenges associated with varying sensitivity of the loss function to different optimization variables. Certain types of activation functions like ReLU are less sensitive to this problem. However, the use of the ReLU can sometimes lead to dead neurons, if one is not careful about the learning rate. The type of gradient descent used to accelerate learning is also important for more efficient executions. Modified stochastic gradient-descent methods include the use of Nesterov momentum, AdaGrad, AdaDelta, RMSProp, and Adam. All these methods encourage gradient-steps that accelerate the learning process.

Numerous methods have been introduced for addressing the problem of cliffs with the use of second-order optimization methods. In particular, Hessian-free optimization is seen as an effective approach for handling many of the underlying optimization issues. An exciting method that has been used recently to improve learning rates is the use of batch normalization. Batch normalization transforms the data layer by layer in order to ensure that the scaling of different variables is done in an optimum way. The use of batch normalization has become extremely common in different types of deep networks. Numerous methods have been proposed for accelerating and compressing neural network algorithms. Acceleration is often achieved via hardware improvements, whereas compression is achieved with algorithmic tricks.

3.9 Bibliographic Notes

The original idea of backpropagation was based on idea of differentiation of composition of functions as developed in control theory [54, 237] under the ambit of *automatic differentiation*. The adaptation of these methods to neural networks was proposed by Paul Werbos in his PhD thesis in 1974 [524], although a more modern form of the algorithm was proposed by Rumelhart *et al.* in 1986 [408]. A discussion of the history of the backpropagation algorithm may be found in the book by Paul Werbos [525].

A discussion of algorithms for hyperparameter optimization in neural networks and other machine learning algorithms may be found in [36, 38, 490]. The random search method for

hyperparameter optimization is discussed in [37]. The use of *Bayesian optimization* for hyperparameter tuning is discussed in [42, 306, 458]. Numerous libraries are available for Bayesian tuning such as *Hyperopt* [614], *Spearmint* [616], and *SMAC* [615].

The rule that the initial weights should depend on both the fan-in and fan-out of a node in proportion to $\sqrt{2/(r_{in} + r_{out})}$ is based on [140]. The analysis of initialization methods for rectifier neural networks is provided in [183]. Evaluations and analysis of the effect of feature preprocessing on neural network learning may be found in [278, 532]. The use of rectifier linear units for addressing some of the training challenges is discussed in [141].

Nesterov's algorithm for gradient descent may be found in [353]. The delta-bar-delta method was proposed by [217]. The AdaGrad algorithm was proposed in [108]. The RMSProp algorithm is discussed in [194]. Another adaptive algorithm using stochastic gradient descent, which is *AdaDelta*, is discussed in [553]. This algorithms shares some similarities with second-order methods, and in particular to the method in [429]. The Adam algorithm, which is a further enhancement along this line of ideas, is discussed in [241]. The practical importance of initialization and momentum in deep learning is discussed in [478]. Beyond the use of the stochastic gradient method, the use of coordinate descent has been proposed [273]. The strategy of *Polyak averaging* is discussed in [380].

Several of the challenges associated with the vanishing and exploding gradient problems are discussed in [140, 205, 368]. Ideas for parameter initialization that avoid some of these problems are discussed in [140]. The gradient clipping rule was discussed by Mikolov in his PhD thesis [324]. A discussion of the gradient clipping method in the context of recurrent neural networks is provided in [368]. The ReLU activation function was introduced in [167], and several of its interesting properties are explored in [141, 221].

A description of several second-order gradient optimization methods (such as the Newton method) is provided in [41, 545, 300]. The basic principles of the conjugate gradient method have been described in several classical books and papers [41, 189, 443], and the work in [313, 314] discusses applications to neural networks. The work in [316] leverages a Kronecker-factored curvature matrix for fast gradient descent. Another way of approximating the Newton method is the quasi-Newton method [273, 300], with the simplest approximation being a diagonal Hessian [24]. The acronym BFGS stands for the Broyden-Fletcher-Goldfarb-Shanno algorithm. A variant known as limited memory BFGS or L-BFGS [273, 300] does not require as much memory. Another popular second-order method is the Levenberg–Marquardt algorithm. This approach is, however, defined for squared loss functions and cannot be used with many forms of cross-entropy or log-losses that are common in neural networks. Overviews of the approach may be found in [133, 300]. General discussions of different types of nonlinear programming methods are provided in [23, 39].

The stability of neural networks to local minima is discussed in [88, 426]. Batch normalization methods were introduced recently in [214]. A method that uses whitening for batch normalization is discussed in [96], although the approach seems not to be practical. Batch normalization requires some minor adjustments for recurrent networks [81], although a more effective approach for recurrent networks is that of *layer normalization* [14]. In this method (cf. Section 7.3.1), a single training case is used for normalizing all units in a layer, rather than using mini-batch normalization of a single unit. The approach is useful for recurrent networks. An analogous notion to batch normalization is that of weight normalization [419], in which the magnitudes and directions of the weight vectors are decoupled during the learning process. Related training tricks are discussed in [362].

A broader discussion of accelerating machine learning algorithms with GPUs may be found in [644]. Various types of parallelization tricks for GPUs are discussed in [74, 91, 254], and specific discussions on convolutional neural networks are provided in [541]. Model

compression with regularization is discussed in [168, 169]. A related model compression method is proposed in [213]. The use of mimic models for compression is discussed in [55, 13]. A related approach is discussed in [202]. The leveraging of parameter redundancy for compressing neural networks is discussed in [94]. The compression of neural networks with the hashing trick is discussed in [66].

3.9.1 Software Resources

All the training algorithms discussed in this chapter are supported by numerous deep learning frameworks like *Caffe* [571], *Torch* [572], *Theano* [573], and *TensorFlow* [574]. Extensions of *Caffe* to Python and MATLAB are available. All these frameworks provide a variety of training algorithms that are discussed in this chapter. Options for batch normalization are available as separate layers in these frameworks. Several software libraries are available for Bayesian optimization of hyperparameters. These libraries include *Hyperopt* [614], *Spearmint* [616], and *SMAC* [615]. Although these are designed for smaller machine learning problems, they can still be used in some cases. Pointers to the NVIDIA cuDNN may be found in [643]. The different frameworks supported by cuDNN are discussed in [645].

3.10 Exercises

1. Consider the following recurrence:

$$(x_{t+1}, y_{t+1}) = (f(x_t, y_t), g(x_t, y_t)) \tag{3.66}$$

Here, $f()$ and $g()$ are multivariate functions.

(a) Derive an expression for $\frac{\partial x_{t+2}}{\partial x_t}$ in terms of only x_t and y_t.

(b) Can you draw an architecture of a neural network corresponding to the above recursion for t varying from 1 to 5? Assume that the neurons can compute any function you want.

2. Consider a two-input neuron that multiplies its two inputs x_1 and x_2 to obtain the output o. Let L be the loss function that is computed at o. Suppose that you know that $\frac{\partial L}{\partial o} = 5$, $x_1 = 2$, and $x_2 = 3$. Compute the values of $\frac{\partial L}{\partial x_1}$ and $\frac{\partial L}{\partial x_2}$.

3. Consider a neural network with three layers including an input layer. The first (input) layer has four inputs x_1, x_2, x_3, and x_4. The second layer has six hidden units corresponding to all pairwise multiplications. The output node o simply adds the values in the six hidden units. Let L be the loss at the output node. Suppose that you know that $\frac{\partial L}{\partial o} = 2$, and $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, and $x_4 = 4$. Compute $\frac{\partial L}{\partial x_i}$ for each i.

4. How does your answer to the previous question change when the output o is computed as a maximum of its six inputs rather than its sum?

5. The chapter discusses (cf. Table 3.1) how one can perform a backpropagation of an arbitrary function by using the multiplication with the Jacobian matrix. Discuss why one must be careful in using this matrix-centric approach.[Hint: Compute the Jacobian with respect to sigmoid function]

Bibliography

[1] D. Ackley, G. Hinton, and T. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1), pp. 147–169, 1985.

[2] C. Aggarwal. Data classification: Algorithms and applications, *CRC Press*, 2014.

[3] C. Aggarwal. Data mining: The textbook. *Springer*, 2015.

[4] C. Aggarwal. Recommender systems: The textbook. *Springer*, 2016.

[5] C. Aggarwal. Outlier analysis. *Springer*, 2017.

[6] C. Aggarwal. Machine learning for text. *Springer*, 2018.

[7] R. Ahuja, T. Magnanti, and J. Orlin. Network flows: Theory, algorithms, and applications. *Prentice Hall*, 1993.

[8] E. Aljalbout, V. Golkov, Y. Siddiqui, and D. Cremers. Clustering with deep learning: Taxonomy and new methods. *arXiv:1801.07648*, 2018.
https://arxiv.org/abs/1801.07648

[9] R. Al-Rfou, B. Perozzi, and S. Skiena. Polyglot: Distributed word representations for multilingual nlp. *arXiv:1307.1662*, 2013.
https://arxiv.org/abs/1307.1662

[10] D. Amodei *at al.* Concrete problems in AI safety. *arXiv:1606.06565*, 2016.
https://arxiv.org/abs/1606.06565

[11] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. *arXiv:1701.04862*, 2017.
https://arxiv.org/abs/1701.04862

[12] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv:1701.07875*, 2017.
https://arxiv.org/abs/1701.07875

[13] J. Ba and R. Caruana. Do deep nets really need to be deep? *NIPS Conference*, pp. 2654–2662, 2014.

C. C. Aggarwal, *Neural Networks and Deep Learning*,
https://doi.org/10.1007/978-3-319-94463-0

[14] J. Ba, J. Kiros, and G. Hinton. Layer normalization. *arXiv:1607.06450*, 2016. https://arxiv.org/abs/1607.06450

[15] J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. *arXiv: 1412.7755*, 2014. https://arxiv.org/abs/1412.7755

[16] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky. Neural codes for image retrieval. *arXiv:1404.1777*, 2014. https://arxiv.org/abs/1404.1777

[17] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt. Sequential deep learning for human action recognition. *International Workshop on Human Behavior Understanding*, pp. 29–39, 2011.

[18] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015. Also *arXiv:1409.0473*, 2014. https://arxiv.org/abs/1409.0473

[19] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv:1611.02167*, 2016. https://arxiv.org/abs/1611.02167

[20] P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri. Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15(11), pp. 937–946, 1999.

[21] N. Ballas, L. Yao, C. Pal, and A. Courville. Delving deeper into convolutional networks for learning video representations. *arXiv:1511.06432*, 2015. https://arxiv.org/abs/1511.06432

[22] J. Baxter, A. Tridgell, and L. Weaver. Knightcap: a chess program that learns by combining td (lambda) with game-tree search. *arXiv cs/9901002*, 1999.

[23] M. Bazaraa, H. Sherali, and C. Shetty. Nonlinear programming: theory and algorithms. *John Wiley and Sons*, 2013.

[24] S. Becker, and Y. LeCun. Improving the convergence of back-propagation learning with second order methods. *Proceedings of the 1988 connectionist models summer school*, pp. 29–37, 1988.

[25] M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47, pp. 253–279, 2013.

[26] R. E. Bellman. Dynamic Programming. *Princeton University Press*, 1957.

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

[28] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE TPAMI*, 35(8), pp. 1798–1828, 2013.

[29] Y. Bengio and O. Delalleau. Justifying and generalizing contrastive divergence. *Neural Computation*, 21(6), pp. 1601–1621, 2009.

[30] Y. Bengio and O. Delalleau. On the expressive power of deep architectures. *Algorithmic Learning Theory*, pp. 18–36, 2011.

[31] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. *NIPS Conference*, 19, 153, 2007.

[32] Y. Bengio, N. Le Roux, P. Vincent, O. Delalleau, and P. Marcotte. Convex neural networks. *NIPS Conference*, pp. 123–130, 2005.

[33] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. *ICML Conference*, 2009.

[34] Y. Bengio, L. Yao, G. Alain, and P. Vincent. Generalized denoising auto-encoders as generative models. *NIPS Conference*, pp. 899–907, 2013.

[35] J. Bergstra *et al.* Theano: A CPU and GPU math compiler in Python. *Python in Science Conference*, 2010.

[36] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl. Algorithms for hyper-parameter optimization. *NIPS Conference*, pp. 2546–2554, 2011.

[37] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13, pp. 281–305, 2012.

[38] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *ICML Confererence*, pp. 115–123, 2013.

[39] D. Bertsekas. Nonlinear programming *Athena Scientific*, 1999.

[40] C. M. Bishop. Pattern recognition and machine learning. *Springer*, 2007.

[41] C. M. Bishop. Neural networks for pattern recognition. *Oxford University Press*, 1995.

[42] C. M. Bishop. Bayesian Techniques. Chapter 10 in "Neural Networks for Pattern Recognition," pp. 385–439, 1995.

[43] C. M Bishop. Improving the generalization properties of radial basis function neural networks. *Neural Computation*, 3(4), pp. 579–588, 1991.

[44] C. M. Bishop. Training with noise is equivalent to Tikhonov regularization. *Neural computation*, 7(1),pp. 108–116, 1995.

[45] C. M. Bishop, M. Svensen, and C. K. Williams. GTM: A principled alternative to the self-organizing map. *NIPS Conference*, pp. 354–360, 1997.

[46] M. Bojarski *et al.* End to end learning for self-driving cars. *arXiv:1604.07316*, 2016. https://arxiv.org/abs/1604.07316

[47] M. Bojarski *et al.* Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car. *arXiv:1704.07911*, 2017. https://arxiv.org/abs/1704.07911

[48] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4), pp. 291–294, 1988.

[49] L. Breiman. Random forests. *Journal Machine Learning archive*, 45(1), pp. 5–32, 2001.

[50] L. Breiman. Bagging predictors. *Machine Learning*, 24(2), pp. 123–140, 1996.

[51] D. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2, pp. 321–355, 1988.

[52] C. Browne *et al.* A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), pp. 1–43, 2012.

[53] T. Brox and J. Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE TPAMI*, 33(3), pp. 500–513, 2011.

[54] A. Bryson. A gradient method for optimizing multi-stage allocation processes. *Harvard University Symposium on Digital Computers and their Applications*, 1961.

[55] C. Bucilu, R. Caruana, and A. Niculescu-Mizil. Model compression. *ACM KDD Conference*, pp. 535–541, 2006.

[56] P. Bühlmann and B. Yu. Analyzing bagging. *Annals of Statistics*, pp. 927–961, 2002.

[57] M. Buhmann. Radial Basis Functions: Theory and implementations. *Cambridge University Press*, 2003.

[58] Y. Burda, R. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. *arXiv:1509.00519*, 2015.
https://arxiv.org/abs/1509.00519

[59] N. Butko and J. Movellan. I-POMDP: An infomax model of eye movement. *IEEE International Conference on Development and Learning*, pp. 139–144, 2008.

[60] Y. Cao, Y. Chen, and D. Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1), 54–66, 2015.

[61] M. Carreira-Perpinan and G. Hinton. On Contrastive Divergence Learning. *AISTATS*, 10, pp. 33–40, 2005.

[62] S. Chang, W. Han, J. Tang, G. Qi, C. Aggarwal, and T. Huang. Heterogeneous network embedding via deep architectures. *ACM KDD Conference*, pp. 119–128, 2015.

[63] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer. SMOTE: synthetic minority oversampling technique. *Journal of Artificial Intelligence Research*, 16, pp. 321–357, 2002.

[64] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga. Outlier detection with autoencoder ensembles. *SIAM Conference on Data Mining*, 2017.

[65] S. Chen, C. Cowan, and P. Grant. Orthogonal least-squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2), pp. 302–309, 1991.

[66] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. *ICML Confererence*, pp. 2285–2294, 2015.

[67] Y. Chen and M. Zaki. KATE: K-Competitive Autoencoder for Text. *ACM KDD Conference*, 2017.

[68] Y. Chen, T. Krishna, J. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1), pp. 127–138, 2017.

[69] K. Cho, B. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP*, 2014.
https://arxiv.org/pdf/1406.1078.pdf

[70] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. Attention-based models for speech recognition. *NIPS Conference*, pp. 577–585, 2015.

[71] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555*, 2014.
https://arxiv.org/abs/1412.3555

[72] D. Ciresan, U. Meier, L. Gambardella, and J. Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12), pp. 3207–3220, 2010.

[73] C. Clark and A. Storkey. Training deep convolutional neural networks to play go. *ICML Confererence*, pp. 1766–1774, 2015.

[74] A. Coates, B. Huval, T. Wang, D. Wu, A. Ng, and B. Catanzaro. Deep learning with COTS HPC systems. *ICML Confererence*, pp. 1337–1345, 2013.

[75] A. Coates and A. Ng. The importance of encoding versus training with sparse coding and vector quantization. *ICML Confererence*, pp. 921–928, 2011.

[76] A. Coates and A. Ng. Learning feature representations with k-means. *Neural networks: Tricks of the Trade*, Springer, pp. 561–580, 2012.

[77] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. *AAAI Conference*, pp. 215–223, 2011.

[78] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12, pp. 2493–2537, 2011.

[79] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. *ICML Conference*, pp. 160–167, 2008.

[80] J. Connor, R. Martin, and L. Atlas. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, 5(2), pp. 240–254, 1994.

[81] T. Cooijmans, N. Ballas, C. Laurent, C. Gulcehre, and A. Courville. Recurrent batch normalization. *arXiv:1603.09025*, 2016.
https://arxiv.org/abs/1603.09025

[82] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3), pp. 273–297, 1995.

[83] M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. *arXiv:1511.00363*, 2015.
https://arxiv.org/pdf/1511.00363.pdf

[84] T. Cover. Geometrical and statistical properties of systems of linear inequalities with applications to pattern recognition. *IEEE Transactions on Electronic Computers*, pp. 326–334, 1965.

[85] D. Cox and N. Pinto. Beyond simple features: A large-scale feature search approach to unconstrained face recognition. *IEEE International Conference on Automatic Face and Gesture Recognition and Workshops*, pp. 8–15, 2011.

[86] G. Dahl, R. Adams, and H. Larochelle. Training restricted Boltzmann machines on word observations. *arXiv:1202.5695*, 2012.
https://arxiv.org/abs/1202.5695

[87] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *Computer Vision and Pattern Recognition*, pp. 886–893, 2005.

[88] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *NIPS Conference*, pp. 2933–2941, 2014.

[89] N. de Freitas. Machine Learning, University of Oxford (Course Video), 2013. https://www.youtube.com/watch?v=w2OtwL5T1ow&list=PLE6Wd9FREdyJ5lbFl8Uu–GjecvVw66F6

[90] N. de Freitas. Deep Learning, University of Oxford (Course Video), 2015. https://www.youtube.com/watch?v=PlhFWT7vAEw&list=PLjK8ddCbDMphIMSXn-1IjyYpHU3DaUYw

[91] J. Dean *et al.* Large scale distributed deep networks. *NIPS Conference*, 2012.

[92] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *NIPS Conference*, pp. 3844–3852, 2016.

[93] O. Delalleau and Y. Bengio. Shallow vs. deep sum-product networks. *NIPS Conference*, pp. 666–674, 2011.

[94] M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. de Freitas. Predicting parameters in deep learning. *NIPS Conference*, pp. 2148–2156, 2013.

[95] E. Denton, S. Chintala, and R. Fergus. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. *NIPS Conference*, pp. 1466–1494, 2015.

[96] G. Desjardins, K. Simonyan, and R. Pascanu. Natural neural networks. *NIPS Congference*, pp. 2071–2079, 2015.

[97] F. Despagne and D. Massart. Neural networks in multivariate calibration. *Analyst*, 123(11), pp. 157R–178R, 1998.

[98] T. Dettmers. 8-bit approximations for parallelism in deep learning. *arXiv:1511.04561*, 2015. https://arxiv.org/abs/1511.04561

[99] C. Ding, T. Li, and W. Peng. On the equivalence between non-negative matrix factorization and probabilistic latent semantic indexing. *Computational Statistics and Data Analysis*, 52(8), pp. 3913–3927, 2008.

[100] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *IEEE conference on computer vision and pattern recognition*, pp. 2625–2634, 2015.

[101] G. Dorffner. Neural networks for time series processing. *Neural Network World*, 1996.

[102] C. Dos Santos and M. Gatti. Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. *COLING*, pp. 69–78, 2014.

[103] A. Dosovitskiy, J. Tobias Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. *CVPR Conference*, pp. 1538–1546, 2015.

[104] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. *CVPR Conference*, pp. 4829–4837, 2016.

[105] K. Doya. Bifurcations of recurrent neural networks in gradient descent learning. *IEEE Transactions on Neural Networks*, 1, pp. 75–80, 1993.

[106] C. Doersch. Tutorial on variational autoencoders. *arXiv:1606.05908*, 2016. https://arxiv.org/abs/1606.05908

[107] H. Drucker and Y. LeCun. Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, 3(6), pp. 991–997, 1992.

[108] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, pp. 2121–2159, 2011.

[109] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv:1603.07285*, 2016.
https://arxiv.org/abs/1603.07285

[110] A. Elkahky, Y. Song, and X. He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. *WWW Conference*, pp. 278–288, 2015.

[111] J. Elman. Finding structure in time. *Cognitive Science*, 14(2), pp. 179–211, 1990.

[112] J. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48, pp. 781–799, 1993.

[113] D. Erhan, Y. Bengio, A. Courville, P. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning?. *Journal of Machine Learning Research*, 11, pp. 625–660, 2010.

[114] S. Essar *et al.* Convolutional neural networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Science of the United States of America*, 113(41), pp. 11441–11446, 2016.

[115] A. Fader, L. Zettlemoyer, and O. Etzioni. Paraphrase-Driven Learning for Open Question Answering. *ACL*, pp. 1608–1618, 2013.

[116] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE TPAMI*, 28(4), pp. 594–611, 2006.

[117] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE TPAMI*, 32(9), pp. 1627–1645, 2010.

[118] A. Fader, L. Zettlemoyer, and O. Etzioni. Open question answering over curated and extracted knowledge bases. *ACM KDD Conference*, 2014.

[119] A. Fischer and C. Igel. An introduction to restricted Boltzmann machines. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pp. 14–36, 2012.

[120] R. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7: pp. 179–188, 1936.

[121] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9(5), pp. 768–786, 1998.

[122] Y. Freund and R. Schapire. A decision-theoretic generalization of online learning and application to boosting. *Computational Learning Theory*, pp. 23–37, 1995.

[123] Y. Freund and R. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3), pp. 277–296, 1999.

[124] Y. Freund and D. Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. *Technical report*, Santa Cruz, CA, USA, 1994

[125] B. Fritzke. Fast learning with incremental RBF networks. *Neural Processing Letters*, 1(1), pp. 2–5, 1994.

[126] B. Fritzke. A growing neural gas network learns topologies. *NIPS Conference*, pp. 625–632, 1995.

[127] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4), pp. 193–202, 1980.

[128] S. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2), pp. 179–191, 1990.

[129] S. Gallant. Neural network learning and expert systems. *MIT Press*, 1993.

[130] H. Gao, H. Yuan, Z. Wang, and S. Ji. Pixel Deconvolutional Networks. *arXiv:1705.06820*, 2017.
https://arxiv.org/abs/1705.06820

[131] L. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. *NIPS Conference*, pp. 262–270, 2015.

[132] L. Gatys, A. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2414–2423, 2015.

[133] H. Gavin. The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems, 2011.
http://people.duke.edu/~hpgavin/ce281/lm.pdf

[134] P. Gehler, A. Holub, and M. Welling. The Rate Adapting Poisson (RAP) model for information retrieval and object recognition. *ICML Confererence*, 2006.

[135] S. Gelly *et al.* The grand challenge of computer Go: Monte Carlo tree search and extensions. *Communcations of the ACM*, 55, pp. 106–113, 2012.

[136] A. Gersho and R. M. Gray. Vector quantization and signal compression. *Springer Science and Business Media*, 2012.

[137] A. Ghodsi. STAT 946: Topics in Probability and Statistics: Deep Learning, *University of Waterloo*, Fall 2015.
https://www.youtube.com/watch?v=fyAZszlPphs&list=PLehuLRPyt1Hyi78UOkMP–WCGRxGcA9NVOE

[138] W. Gilks, S. Richardson, and D. Spiegelhalter. Markov chain Monte Carlo in practice.*CRC Press*, 1995.

[139] F. Girosi and T. Poggio. Networks and the best approximation property. *Biological Cybernetics*, 63(3), pp. 169–176, 1990.

[140] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *AISTATS*, pp. 249–256, 2010.

[141] X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. *AISTATS*, 15(106), 2011.

[142] P. Glynn. Likelihood ratio gradient estimation: an overview, *Proceedings of the 1987 Winter Simulation Conference*, pp. 366–375, 1987.

[143] Y. Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research (JAIR)*, 57, pp. 345–420, 2016.

[144] C. Goller and A. Küchler. Learning task-dependent distributed representations by backpropagation through structure. *Neural Networks*, 1, pp. 347–352, 1996.

[145] I. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv:1701.00160*, 2016. https://arxiv.org/abs/1701.00160

[146] I. Goodfellow, O. Vinyals, and A. Saxe. Qualitatively characterizing neural network optimization problems. *arXiv:1412.6544*, 2014. [Also appears in *International Conference in Learning Representations*, 2015] https://arxiv.org/abs/1412.6544

[147] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. *MIT Press*, 2016.

[148] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv:1302.4389*, 2013.

[149] I. Goodfellow *et al.* Generative adversarial nets. *NIPS Conference*, 2014.

[150] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. *Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6645–6649, 2013.

[151] A. Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2013. https://arxiv.org/abs/1308.0850

[152] A. Graves. Supervised sequence labelling with recurrent neural networks *Springer*, 2012. http://rd.springer.com/book/10.1007%2F978-3-642-24797-2

[153] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. *ICML Conferer-ence*, pp. 369–376, 2006.

[154] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE TPAMI*, 31(5), pp. 855–868, 2009.

[155] A. Graves and J. Schmidhuber. Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks*, 18(5–6), pp. 602–610, 2005.

[156] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. *NIPS Conference*, pp. 545–552, 2009.

[157] A. Graves and N. Jaitly. Towards End-To-End Speech Recognition with Recurrent Neural Networks. *ICML Conference*, pp. 1764–1772, 2014.

[158] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv:1410.5401*, 2014. https://arxiv.org/abs/1410.5401

[159] A. Graves *et al.* Hybrid computing using a neural network with dynamic external memory. *Nature*, 538.7626, pp. 471–476, 2016.

[160] K. Greff, R. K. Srivastava, J. Koutnik, B. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 2016. http://ieeexplore.ieee.org/abstract/document/7508408/

[161] K. Greff, R. K. Srivastava, and J. Schmidhuber. Highway and residual networks learn unrolled iterative estimation. *arXiv:1612.07771*, 2016. https://arxiv.org/abs/1612.07771

[162] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics*, 42(6), pp. 1291–1307, 2012.

[163] R. Girshick, F. Iandola, T. Darrell, and J. Malik. Deformable part models are convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 437–446, 2015.

[164] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. *ACM KDD Conference*, pp. 855–864, 2016.

[165] X. Guo, S. Singh, H. Lee, R. Lewis, and X. Wang. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. *Advances in NIPS Conference*, pp. 3338–3346, 2014.

[166] M. Gutmann and A. Hyvarinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. *AISTATS*, 1(2), pp. 6, 2010.

[167] R. Hahnloser and H. S. Seung. Permitted and forbidden sets in symmetric threshold-linear networks. *NIPS Conference*, pp. 217–223, 2001.

[168] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. Dally. EIE: Efficient Inference Engine for Compressed Neural Network. *ACM SIGARCH Computer Architecture News*, 44(3), pp. 243–254, 2016.

[169] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural networks. *NIPS Conference*, pp. 1135–1143, 2015.

[170] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE TPAMI*, 12(10), pp. 993–1001, 1990.

[171] M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. *ICML Confererence*, pp. 1225–1234, 2006.

[172] B. Hariharan, P. Arbelaez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. *arXiv:1407.1808*, 2014.
https://arxiv.org/abs/1407.1808

[173] E. Hartman, J. Keeler, and J. Kowalski. Layered neural networks with Gaussian hidden units as universal approximations. *Neural Computation*, 2(2), pp. 210–215, 1990.

[174] H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-Learning. *AAAI Conference*, 2016.

[175] B. Hassibi and D. Stork. Second order derivatives for network pruning: Optimal brain surgeon. *NIPS Conference*, 1993.

[176] D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2), pp. 245–258, 2017.

[177] T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning. *Springer*, 2009.

[178] T. Hastie and R. Tibshirani. Generalized additive models. *CRC Press*, 1990.

[179] T. Hastie, R. Tibshirani, and M. Wainwright. Statistical learning with sparsity: the lasso and generalizations. *CRC Press*, 2015.

[180] M. Havaei *et al.* Brain tumor segmentation with deep neural networks. *Medical Image Analysis*, 35, pp. 18–31, 2017.

[181] S. Hawkins, H. He, G. Williams, and R. Baxter. Outlier detection using replicator neural networks. *International Conference on Data Warehousing and Knowledge Discovery*, pp. 170–180, 2002.

[182] S. Haykin. Neural networks and learning machines. *Pearson*, 2008.

[183] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

[184] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[185] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *European Conference on Computer Vision*, pp. 630–645, 2016.

[186] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. S. Chua. Neural collaborative filtering. *WWW Conference*, pp. 173–182, 2017.

[187] N. Heess *et al.* Emergence of Locomotion Behaviours in Rich Environments. *arXiv:1707.02286*, 2017.
https://arxiv.org/abs/1707.02286
Video 1 at: https://www.youtube.com/watch?v=hx_bgoTF7bs
Video 2 at: https://www.youtube.com/watch?v=gn4nRCC9TwQ&feature=youtu.be

[188] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv:1506.05163*, 2015.
https://arxiv.org/abs/1506.05163

[189] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6), 1952.

[190] G. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40(1–3), pp. 185–234, 1989.

[191] G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8), pp. 1771–1800, 2002.

[192] G. Hinton. To recognize shapes, first learn to generate images. *Progress in Brain Research*, 165, pp. 535–547, 2007.

[193] G. Hinton. A practical guide to training restricted Boltzmann machines. *Momentum*, 9(1), 926, 2010.

[194] G. Hinton. Neural networks for machine learning, *Coursera Video*, 2012.

[195] G. Hinton, P. Dayan, B. Frey, and R. Neal. The wake–sleep algorithm for unsupervised neural networks. *Science*, 268(5214), pp. 1158–1162, 1995.

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

[197] G. Hinton and T. Sejnowski. Learning and relearning in Boltzmann machines. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986.

[198] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313, (5766), pp. 504–507, 2006.

[199] G. Hinton and R. Salakhutdinov. Replicated softmax: an undirected topic model. *NIPS Conference*, pp. 1607–1614, 2009.

[200] G. Hinton and R. Salakhutdinov. A better way to pretrain deep Boltzmann machines. *NIPS Conference*, pp. 2447–2455, 2012.

[201] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, 2012. https://arxiv.org/abs/1207.0580

[202] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *NIPS Workshop*, 2014.

[203] R. Hochberg. Matrix Multiplication with CUDA: A basic introduction to the CUDA programming model. *Unpublished manuscript*, 2012. http://www.shodor.org/media/content/petascale/materials/UPModules/matrixMultiplication/moduleDocument.pdf

[204] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8), pp. 1735–1785, 1997.

[205] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, *A Field Guide to Dynamical Recurrent Neural Networks*, IEEE Press, 2001.

[206] T. Hofmann. Probabilistic latent semantic indexing. *ACM SIGIR Conference*, pp. 50–57, 1999.

[207] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *National Academy of Sciences of the USA*, 79(8), pp. 2554–2558, 1982.

[208] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), pp. 359–366, 1989.

[209] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. *IEEE International Conference on Data Mining*, pp. 263–272, 2008.

[210] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. *European Conference on Computer Vision*, pp. 646–661, 2016.

[211] G. Huang, Z. Liu, K. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv:1608.06993*, 2016. https://arxiv.org/abs/1608.06993

[212] D. Hubel and T. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 124(3), pp. 574–591, 1959.

[213] F. Iandola, S. Han, M. Moskewicz, K. Ashraf, W. Dally, and K. Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv:1602.07360*, 2016. https://arxiv.org/abs/1602.07360

[214] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[215] P. Isola, J. Zhu, T. Zhou, and A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv:1611.07004*, 2016.
https://arxiv.org/abs/1611.07004

[216] M. Iyyer, J. Boyd-Graber, L. Claudino, R. Socher, and H. Daume III. A Neural Network for Factoid Question Answering over Paragraphs. *EMNLP*, 2014.

[217] R. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4), pp. 295–307, 1988.

[218] M. Jaderberg, K. Simonyan, and A. Zisserman. Spatial transformer networks. *NIPS Conference*, pp. 2017–2025, 2015.

[219] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks – with an erratum note. *German National Research Center for Information Technology GMD Technical Report*, 148(34), 13, 2001.

[220] H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304, pp. 78–80, 2004.

[221] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? *International Conference on Computer Vision (ICCV)*, 2009.

[222] S. Ji, W. Xu, M. Yang, and K. Yu. 3D convolutional neural networks for human action recognition. *IEEE TPAMI*, 35(1), pp. 221–231, 2013.

[223] Y. Jia *et al.* Caffe: Convolutional architecture for fast feature embedding. *ACM International Conference on Multimedia*, 2014.

[224] C. Johnson. Logistic matrix factorization for implicit feedback data. *NIPS Conference*, 2014.

[225] J. Johnson, A. Karpathy, and L. Fei-Fei. Densecap: Fully convolutional localization networks for dense captioning. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4565–4574, 2015.

[226] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *European Conference on Computer Vision*, pp. 694–711, 2015.

[227] R. Johnson and T. Zhang. Effective use of word order for text categorization with convolutional neural networks. *arXiv:1412.1058*, 2014.
https://arxiv.org/abs/1412.1058

[228] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. *ICML Confererence*, pp. 2342–2350, 2015.

[229] L. Kaiser and I. Sutskever. Neural GPUs learn algorithms. *arXiv:1511.08228*, 2015.
https://arxiv.org/abs/1511.08228

[230] S. Kakade. A natural policy gradient. *NIPS Conference*, pp. 1057–1063, 2002.

[231] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. *EMNLP*, 3, 39, pp. 413, 2013.

[232] H. Kandel, J. Schwartz, T. Jessell, S. Siegelbaum, and A. Hudspeth. Principles of neural science. *McGraw Hill*, 2012.

[233] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv:1506.02078*, 2015.
https://arxiv.org/abs/1506.02078

[234] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 725–1732, 2014.

[235] A. Karpathy. The unreasonable effectiveness of recurrent neural networks, *Blog post*, 2015.
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[236] A. Karpathy, J. Johnson, and L. Fei-Fei. Stanford University Class CS321n: Convolutional neural networks for visual recognition, 2016.
http://cs231n.github.io/

[237] H. J. Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10), pp. 947–954, 1960.

[238] F. Khan, B. Mutlu, and X. Zhu. How do humans teach: On curriculum learning and teaching dimension. *NIPS Conference*, pp. 1449–1457, 2011.

[239] T. Kietzmann, P. McClure, and N. Kriegeskorte. Deep Neural Networks In Computational Neuroscience. *bioRxiv, 133504*, 2017.
https://www.biorxiv.org/content/early/2017/05/04/133504

[240] Y. Kim. Convolutional neural networks for sentence classification. *arXiv:1408.5882*, 2014.

[241] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.
https://arxiv.org/abs/1412.6980

[242] D. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.
https://arxiv.org/abs/1312.6114

[243] T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907*, 2016.
https://arxiv.org/pdf/1609.02907.pdf

[244] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220, pp. 671–680, 1983.

[245] J. Kivinen and M. Warmuth. The perceptron algorithm vs. winnow: linear vs. logarithmic mistake bounds when few input variables are relevant. *Computational Learning Theory*, pp. 289–296, 1995.

[246] L. Kocsis and C. Szepesvari. Bandit based monte-carlo planning. *ECML Conference*, pp. 282–293, 2006.

[247] R. Kohavi and D. Wolpert. Bias plus variance decomposition for zero-one loss functions. *ICML Conference*, 1996.

[248] T. Kohonen. The self-organizing map. Neurocomputing, 21(1), pp. 1–6, 1998.

[249] T. Kohonen. Self-organization and associative memory. *Springer*, 2012.

[250] T. Kohonen. Self-organizing maps, *Springer*, 2001.

[251] D. Koller and N. Friedman. Probabilistic graphical models: principles and techniques. *MIT Press*, 2009.

[252] E. Kong and T. Dietterich. Error-correcting output coding corrects bias and variance. *ICML Conference*, pp. 313–321, 1995.

[253] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1), 1, 2010.

[254] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv:1404.5997*, 2014.
https://arxiv.org/abs/1404.5997

[255] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS Conference*, pp. 1097–1105. 2012.

[256] M. Kubat. Decision trees can initialize radial-basis function networks. *IEEE Transactions on Neural Networks*, 9(5), pp. 813–821, 1998.

[257] A. Kumar *et al.* Ask me anything: Dynamic memory networks for natural language processing. *ICML Confererence*, 2016.

[258] Y. Koren. Collaborative filtering with temporal dynamics. *ACM KDD Conference*, pp. 447–455, 2009.

[259] M. Lai. Giraffe: Using deep reinforcement learning to play chess. *arXiv:1509.01549*, 2015.

[260] S. Lai, L. Xu, K. Liu, and J. Zhao. Recurrent Convolutional Neural Networks for Text Classification. *AAAI*, pp. 2267–2273, 2015.

[261] B. Lake, T. Ullman, J. Tenenbaum, and S. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, pp. 1–101, 2016.

[262] H. Larochelle. Neural Networks (Course). Universite de Sherbrooke, 2013.
https://www.youtube.com/watch?v=SGZ6BttHMPw&list=PL6Xpj9I5qXYEcOhn7–TqghAJ6NAPrNmUBH

[263] H. Larochelle and Y. Bengio. Classification using discriminative restricted Boltzmann machines. *ICML Conference*, pp. 536–543, 2008.

[264] H. Larochelle, M. Mandel, R. Pascanu, and Y. Bengio. Learning algorithms for the classification restricted Boltzmann machine. *Journal of Machine Learning Research*, 13, pp. 643–669, 2012.

[265] H. Larochelle and I. Murray. The neural autoregressive distribution estimator. *International Conference on Artificial Intelligence and Statistics*, pp. 29–37, 2011.

[266] H. Larochelle and G. E. Hinton. Learning to combine foveal glimpses with a third-order Boltzmann machine. *NIPS Conference*, 2010.

[267] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. *ICML Confererence*, pp. 473–480, 2007.

[268] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv:1605.07648*, 2016.
https://arxiv.org/abs/1605.07648

[269] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1), pp. 98–113, 1997.

[270] Q. Le *et al.* Building high-level features using large scale unsupervised learning. *ICASSP*, 2013.

[271] Q. Le, N. Jaitly, and G. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv:1504.00941*, 2015.
https://arxiv.org/abs/1504.00941

[272] Q. Le and T. Mikolov. Distributed representations of sentences and documents. *ICML Conference*, pp. 1188–196, 2014.

[273] Q. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Ng, On optimization methods for deep learning. *ICML Conference*, pp. 265–272, 2011.

[274] Q. Le, W. Zou, S. Yeung, and A. Ng. Learning hierarchical spatio-temporal features for action recognition with independent subspace analysis. *CVPR Conference*, 2011.

[275] Y. LeCun. Modeles connexionnistes de l'apprentissage. *Doctoral Dissertation*, Universite Paris, 1987.

[276] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361(10), 1995.

[277] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553), pp. 436–444, 2015.

[278] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. in G. Orr and K. Muller (eds.) *Neural Networks: Tricks of the Trade*, Springer, 1998.

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

[280] Y. LeCun, S. Chopra, R. M. Hadsell, M. A. Ranzato, and F.-J. Huang. A tutorial on energy-based learning. *Predicting Structured Data*, MIT Press, pp. 191–246,, 2006.

[281] Y. LeCun, C. Cortes, and C. Burges. The MNIST database of handwritten digits, 1998.
http://yann.lecun.com/exdb/mnist/

[282] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. *NIPS Conference*, pp. 598–605, 1990.

[283] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. *IEEE International Symposium on Circuits and Systems*, pp. 253–256, 2010.

[284] H. Lee, C. Ekanadham, and A. Ng. Sparse deep belief net model for visual area V2. *NIPS Conference*, 2008.

[285] H. Lee, R. Grosse, B. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *ICML Conference*, pp. 609–616, 2009.

[286] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39), pp. 1–40, 2016.
Video at: https://sites.google.com/site/visuomotorpolicy/

[287] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. *NIPS Conference*, pp. 2177–2185, 2014.

[288] O. Levy, Y. Goldberg, and I. Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3, pp. 211–225, 2015.

[289] W. Levy and R. Baxter. Energy efficient neural codes. *Neural Computation*, 8(3), pp. 531–543, 1996.

[290] M. Lewis, D. Yarats, Y. Dauphin, D. Parikh, and D. Batra. Deal or No Deal? End-to-End Learning for Negotiation Dialogues. *arXiv:1706.05125*, 2017.
https://arxiv.org/abs/1706.05125

[291] J. Li, W. Monroe, A. Ritter, M. Galley,, J. Gao, and D. Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv:1606.01541*, 2016.
https://arxiv.org/abs/1606.01541

[292] L. Li, W. Chu, J. Langford, and R. Schapire. A contextual-bandit approach to personalized news article recommendation. *WWW Conference*, pp. 661–670, 2010.

[293] Y. Li. Deep reinforcement learning: An overview. *arXiv:1701.07274*, 2017.
https://arxiv.org/abs/1701.07274

[294] Q. Liao, K. Kawaguchi, and T. Poggio. Streaming normalization: Towards simpler and more biologically-plausible normalizations for online and recurrent learning. *arXiv:1610.06160*, 2016.
https://arxiv.org/abs/1610.06160

[295] D. Liben-Nowell, and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7), pp. 1019–1031, 2007.

[296] L.-J. Lin. Reinforcement learning for robots using neural networks. *Technical Report*, DTIC Document, 1993.

[297] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv:1312.4400*, 2013.
https://arxiv.org/abs/1312.4400

[298] Z. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv:1506.00019*, 2015.
https://arxiv.org/abs/1506.00019

[299] J. Lu, J. Yang, D. Batra, and D. Parikh. Hierarchical question-image co-attention for visual question answering. *NIPS Conference*, pp. 289–297, 2016.

[300] D. Luenberger and Y. Ye. Linear and nonlinear programming, *Addison-Wesley*, 1984.

[301] M. Lukosevicius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3), pp. 127–149, 2009.

[302] M. Luong, H. Pham, and C. Manning. Effective approaches to attention-based neural machine translation. *arXiv:1508.04025*, 2015.
https://arxiv.org/abs/1508.04025

[303] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik. Deep neural nets as a method for quantitative structure-activity relationships. *Journal of Chemical Information and Modeling*, 55(2), pp. 263–274, 2015.

[304] W. Maass, T. Natschlager, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11), pp. 2351–2560, 2002.

[305] L. Maaten and G. E. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9, pp. 2579–2605, 2008.

[306] D. J. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3), pp. 448–472, 1992.

[307] C. Maddison, A. Huang, I. Sutskever, and D. Silver. Move evaluation in Go using deep convolutional neural networks. *International Conference on Learning Representations*, 2015.

[308] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5188–5196, 2015.

[309] A. Makhzani and B. Frey. K-sparse autoencoders. *arXiv:1312.5663*, 2013.
https://arxiv.org/abs/1312.5663

[310] A. Makhzani and B. Frey. Winner-take-all autoencoders. *NIPS Conference*, pp. 2791–2799, 2015.

[311] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders. *arXiv:1511.05644*, 2015.
https://arxiv.org/abs/1511.05644

[312] C. Manning and R. Socher. CS224N: Natural language processing with deep learning. *Stanford University School of Engineering*, 2017.
https://www.youtube.com/watch?v=OQQ-W_63UgQ

[313] J. Martens. Deep learning via Hessian-free optimization. *ICML Conference*, pp. 735–742, 2010.

[314] J. Martens and I. Sutskever. Learning recurrent neural networks with hessian-free optimization. *ICML Conference*, pp. 1033–1040, 2011.

[315] J. Martens, I. Sutskever, and K. Swersky. Estimating the hessian by back-propagating curvature. *arXiv:1206.6464*, 2016.
https://arxiv.org/abs/1206.6464

[316] J. Martens and R. Grosse. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. *ICML Conference*, 2015.

[317] T. Martinetz, S. Berkovich, and K. Schulten. 'Neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Network*, 4(4), pp. 558–569, 1993.

[318] J. Masci, U. Meier, D. Ciresan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. *Artificial Neural Networks and Machine Learning*, pp. 52–59, 2011.

[319] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv:1511.054*, 2015.
https://arxiv.org/abs/1511.05440

[320] P. McCullagh and J. Nelder. Generalized linear models *CRC Press*, 1989.

[321] W. S. McCulloch and W. H. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4), pp. 115–133, 1943.

[322] G. McLachlan. Discriminant analysis and statistical pattern recognition *John Wiley & Sons*, 2004.

[323] C. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximations*, 2, pp. 11–22, 1986.

[324] T. Mikolov. Statistical language models based on neural networks. *Ph.D. thesis, Brno University of Technology*, 2012.

[325] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv:1301.3781*, 2013.
https://arxiv.org/abs/1301.3781

[326] T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato. Learning longer memory in recurrent neural networks. *arXiv:1412.7753*, 2014.
https://arxiv.org/abs/1412.7753

[327] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *NIPS Conference*, pp. 3111–3119, 2013.

[328] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur. Recurrent neural network based language model. *Interspeech*, Vol 2, 2010.

[329] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4), pp. 235–312, 1990.
https://wordnet.princeton.edu/

[330] M. Minsky and S. Papert. Perceptrons. An Introduction to Computational Geometry, *MIT Press*, 1969.

[331] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv:1411.1784*, 2014.
https://arxiv.org/abs/1411.1784

[332] A. Mnih and G. Hinton. A scalable hierarchical distributed language model. *NIPS Conference*, pp. 1081–1088, 2009.

[333] A. Mnih and K. Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. *NIPS Conference*, pp. 2265–2273, 2013.

[334] A. Mnih and Y. Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv:1206.6426*, 2012.
https://arxiv.org/abs/1206.6426

[335] V. Mnih *et al.* Human-level control through deep reinforcement learning. *Nature*, 518 (7540), pp. 529–533, 2015.

[336] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv:1312.5602.*, 2013.
https://arxiv.org/abs/1312.5602

[337] V. Mnih *et al.* Asynchronous methods for deep reinforcement learning. *ICML Confererence*, pp. 1928–1937, 2016.

[338] V. Mnih, N. Heess, and A. Graves. Recurrent models of visual attention. *NIPS Conference*, pp. 2204–2212, 2014.

[339] H. Mobahi and J. Fisher. A theoretical analysis of optimization by Gaussian continuation. *AAAI Conference*, 2015.

[340] G. Montufar. Universal approximation depth and errors of narrow belief networks with discrete units. *Neural Computation*, 26(7), pp. 1386–1407, 2014.

[341] G. Montufar and N. Ay. Refinements of universal approximation results for deep belief networks and restricted Boltzmann machines. *Neural Computation*, 23(5), pp. 1306–1319, 2011.

[342] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2), pp. 281–294, 1989.

[343] A. Moore and C. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1), pp. 103–130, 1993.

[344] F. Morin and Y. Bengio. Hierarchical Probabilistic Neural Network Language Model. *AISTATS*, pp. 246–252, 2005.

[345] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley. Deep learning for healthcare: review, opportunities and challenges. *Briefings in Bioinformatics*, pp. 1–11, 2017.

[346] M. Müller, M. Enzenberger, B. Arneson, and R. Segal. Fuego - an open-source framework for board games and Go engine based on Monte-Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2, pp. 259–270, 2010.

[347] M. Musavi, W. Ahmed, K. Chan, K. Faris, and D. Hummels. On the training of radial basis function classifiers. *Neural Networks*, 5(4), pp. 595–603, 1992.

[348] V. Nair and G. Hinton. Rectified linear units improve restricted Boltzmann machines. *ICML Conference*, pp. 807–814, 2010.

[349] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1), pp. 4–27, 1990.

[350] R. M. Neal. Connectionist learning of belief networks. *Artificial intelligence*, 1992.

[351] R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. *Technical Report CRG-TR-93-1*, 1993.

[352] R. M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2), pp. 125–139, 2001.

[353] Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27, pp. 372–376, 1983.

[354] A. Ng. Sparse autoencoder. *CS294A Lecture notes*, 2011.
https://nlp.stanford.edu/~socherr/sparseAutoencoder_2011new.pdf
https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf

[355] A. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. *Uncertainity in Artificial Intelligence*, pp. 406–415, 2000.

[356] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4694–4702, 2015.

[357] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Ng. Multimodal deep learning. *ICML Conference*, pp. 689–696, 2011.

[358] A. Nguyen, A. Dosovitskiy, J. Yosinski, T., Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *NIPS Conference*, pp. 3387–3395, 2016.

[359] J. Nocedal and S. Wright. Numerical optimization. *Springer*, 2006.

[360] S. Nowlan and G. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4), pp. 473–493, 1992.

[361] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1717–1724, 2014.

[362] G. Orr and K.-R. Müller (editors). Neural Networks: Tricks of the Trade, *Springer*, 1998.

[363] M. J. L. Orr. Introduction to radial basis function networks, *University of Edinburgh Technical Report, Centre of Cognitive Science*, 1996.
ftp://ftp.cogsci.ed.ac.uk/pub/mjo/intro.ps.Z

[364] M. Palatucci, D. Pomerleau, G. Hinton, and T. Mitchell. Zero-shot learning with semantic output codes. *NIPS Conference*, pp. 1410–1418, 2009.

[365] J. Park and I. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(1), pp. 246–257, 1991.

[366] J. Park and I. Sandberg. Approximation and radial-basis-function networks. *Neural Computation*, 5(2), pp. 305–316, 1993.

[367] O. Parkhi, A. Vedaldi, and A. Zisserman. Deep Face Recognition. *BMVC*, 1(3), pp. 6, 2015.

[368] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *ICML Conference*, 28, pp. 1310–1318, 2013.

[369] R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2012.

[370] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. *CVPR Conference*, 2016.

[371] J. Pennington, R. Socher, and C. Manning. Glove: Global Vectors for Word Representation. *EMNLP*, pp. 1532–1543, 2014.

[372] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. *ACM KDD Conference*, pp. 701–710.

[373] C. Peterson and J. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1(5), pp. 995–1019, 1987.

[374] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4), pp. 682–697, 2008.

[375] F. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19), 2229, 1987.

[376] E. Polak. Computational methods in optimization: a unified approach. *Academic Press*, 1971.

[377] L. Polanyi and A. Zaenen. Contextual valence shifters. *Computing Attitude and Affect in Text: Theory and Applications*, pp. 1–10, Springer, 2006.

[378] G. Pollastri, D. Przybylski, B. Rost, and P. Baldi. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics*, 47(2), pp. 228–235, 2002.

[379] J. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1), pp. 77–105, 1990.

[380] B. Polyak and A. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4), pp. 838–855, 1992.

[381] D. Pomerleau. ALVINN, an autonomous land vehicle in a neural network. *Technical Report*, Carnegie Mellon University, 1989.

[382] B. Poole, J. Sohl-Dickstein, and S. Ganguli. Analyzing noise in autoencoders and deep networks. *arXiv:1406.1831*, 2014.
https://arxiv.org/abs/1406.1831

[383] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. *Computer Vision Workshops (ICCV Workshops)*, pp. 689–690, 2011.

[384] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv:1511.06434*, 2015.
https://arxiv.org/abs/1511.06434

[385] A. Rahimi and B. Recht. Random features for large-scale kernel machines. *NIPS Conference*, pp. 1177–1184, 2008.

[386] M.' A. Ranzato, Y-L. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. *NIPS Conference*, pp. 1185–1192, 2008.

[387] M.' A. Ranzato, F. J. Huang, Y-L. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. *Computer Vision and Pattern Recognition*, pp. 1–8, 2007.

[388] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko. Semi-supervised learning with ladder networks. *NIPS Conference*, pp. 3546–3554, 2015.

[389] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *European Conference on Computer Vision*, pp. 525–542, 2016.

[390] A. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 806–813, 2014.

[391] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, 2016.

[392] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. *ICML Conference*, pp. 1060–1069, 2016.

[393] S. Reed and N. de Freitas. Neural programmer-interpreters. *arXiv:1511.06279*, 2015.

[394] R. Rehurek and P. Sojka. Software framework for topic modelling with large corpora. *LREC 2010 Workshop on New Challenges for NLP Frameworks*, pp. 45–50, 2010.
https://radimrehurek.com/gensim/index.html

[395] M. Ren, R. Kiros, and R. Zemel. Exploring models and data for image question answering. *NIPS Conference*, pp. 2953–2961, 2015.

[396] S. Rendle. Factorization machines. *IEEE ICDM Conference*, pp. 995–100, 2010.

[397] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. *ICML Conference*, pp. 833–840, 2011.

[398] S. Rifai, Y. Dauphin, P. Vincent, Y. Bengio, and X. Muller. The manifold tangent classifier. *NIPS Conference*, pp. 2294–2302, 2011.

[399] D. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv:1401.4082*, 2014. https://arxiv.org/abs/1401.4082

[400] R. Rifkin. Everything old is new again: a fresh look at historical approaches in machine learning. *Ph.D. Thesis*, Massachusetts Institute of Technology, 2002.

[401] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5, pp. 101–141, 2004.

[402] V. Romanuke. Parallel Computing Center (Khmelnitskiy, Ukraine) represents an ensemble of 5 convolutional neural networks which performs on MNIST at 0.21 percent error rate. Retrieved 24 November 2016.

[403] B. Romera-Paredes and P. Torr. An embarrassingly simple approach to zero-shot learning. *ICML Confererence*, pp. 2152–2161, 2015.

[404] X. Rong. word2vec parameter learning explained. *arXiv:1411.2738*, 2014. https://arxiv.org/abs/1411.2738

[405] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386, 1958.

[406] D. Ruck, S. Rogers, and M. Kabrisky. Feature selection using a multilayer perceptron. *Journal of Neural Network Computing*, 2(2), pp. 40–88, 1990.

[407] H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE TPAMI*, 20(1), pp. 23–38, 1998.

[408] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323 (6088), pp. 533–536, 1986.

[409] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by back-propagating errors. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pp. 318–362, 1986.

[410] D. Rumelhart, D. Zipser, and J. McClelland. Parallel Distributed Processing, *MIT Press*, pp. 151–193, 1986.

[411] D. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive science*, 9(1), pp. 75–112, 1985.

[412] G. Rummery and M. Niranjan. Online Q-learning using connectionist systems (Vol. 37). *University of Cambridge, Department of Engineering*, 1994.

[413] A. M. Rush, S. Chopra, and J. Weston. A Neural Attention Model for Abstractive Sentence Summarization. *arXiv:1509.00685*, 2015. https://arxiv.org/abs/1509.00685

[414] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann machines for collaborative filtering. *ICML Confererence*, pp. 791–798, 2007.

[415] R. Salakhutdinov and G. Hinton. Semantic Hashing. *SIGIR workshop on Information Retrieval and applications of Graphical Models*, 2007.

[416] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. One shot learning with memory-augmented neural networks. *arXiv: 1605:06065*, 2016. https://www.arxiv.org/pdf/1605.06065.pdf

[417] R. Salakhutdinov and G. Hinton. Deep Boltzmann machines. *Artificial Intelligence and Statistics*, pp. 448–455, 2009.

[418] R. Salakhutdinov and H. Larochelle. Efficient Learning of Deep Boltzmann Machines. *AISTATs*, pp. 693–700, 2010.

[419] T. Salimans and D. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *NIPS Conference*, pp. 901–909, 2016.

[420] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *NIPS Conference*, pp. 2234–2242, 2016.

[421] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3, pp. 210–229, 1959.

[422] T Sanger. Neural network learning control of robot manipulators using gradually increasing task difficulty. *IEEE Transactions on Robotics and Automation*, 10(3), 1994.

[423] H. Sarimveis, A. Alexandridis, and G. Bafas. A fast training algorithm for RBF networks based on subtractive clustering. *Neurocomputing*, 51, pp. 501–505, 2003.

[424] W. Saunders, G. Sastry, A. Stuhlmueller, and O. Evans. Trial without Error: Towards Safe Reinforcement Learning via Human Intervention. *arXiv:1707.05173*, 2017.
https://arxiv.org/abs/1707.05173

[425] A. Saxe, P. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Ng. On random weights and unsupervised feature learning. *ICML Confererence*, pp. 1089–1096, 2011.

[426] A. Saxe, J. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120*, 2013.

[427] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6), pp. 233–242, 1999.

[428] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv:1511.05952*, 2015.
https://arxiv.org/abs/1511.05952

[429] T. Schaul, S. Zhang, and Y. LeCun. No more pesky learning rates. *ICML Confererence*, pp. 343–351, 2013.

[430] B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with Gaussian kernels to radial basis function classifiers. *IEEE Transactions on Signal Processing*, 45(11), pp. 2758–2765, 1997.

[431] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61, pp. 85–117, 2015.

[432] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. *ICML Conference*, 2015.

[433] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *ICLR Conference*, 2016.

[434] M. Schuster and K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11), pp. 2673–2681, 1997.

[435] H. Schwenk and Y. Bengio. Boosting neural networks. *Neural Computation*, 12(8), pp. 1869–1887, 2000.

[436] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. Autorec: Autoencoders meet collaborative filtering. *WWW Conference*, pp. 111–112, 2015.

[437] T. J. Sejnowski. Higher-order Boltzmann machines. *AIP Conference Proceedings*, 15(1), pp. 298–403, 1986.

[438] G. Seni and J. Elder. Ensemble methods in data mining: Improving accuracy through combining predictions. *Morgan and Claypool*, 2010.

[439] I. Serban, A. Sordoni, R. Lowe, L. Charlin, J. Pineau, A. Courville, and Y. Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. *AAAI*, pp. 3295–3301, 2017.

[440] I. Serban, A. Sordoni, Y. Bengio, A. Courville, and J. Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. *AAAI Conference*, pp. 3776–3784, 2016.

[441] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv:1312.6229*, 2013. https://arxiv.org/abs/1312.6229

[442] A. Shashua. On the equivalence between the support vector machine for classification and sparsified Fisher's linear discriminant. *Neural Processing Letters*, 9(2), pp. 129–139, 1999.

[443] J. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. *Technical Report, CMU-CS-94-125*, Carnegie-Mellon University, 1994.

[444] H. Siegelmann and E. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1), pp. 132–150, 1995.

[445] D. Silver *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature*, 529.7587, pp. 484–489, 2016.

[446] D. Silver *et al.* Mastering the game of go without human knowledge. *Nature*, 550.7676, pp. 354–359, 2017.

[447] D. Silver *et al.* Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv*, 2017.
https://arxiv.org/abs/1712.01815

[448] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated subgradient solver for SVM. *Mathematical Programming*, 127(1), pp. 3–30, 2011.

[449] E. Shelhamer, J., Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE TPAMI*, 39(4), pp. 640–651, 2017.

[450] J. Sietsma and R. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1), pp. 67–79, 1991.

[451] B. W. Silverman. Density Estimation for Statistics and Data Analysis. *Chapman and Hall*, 1986.

[452] P. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. *ICDAR*, pp. 958–962, 2003.

[453] H. Simon. The Sciences of the Artificial. *MIT Press*, 1996.

[454] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.
https://arxiv.org/abs/1409.1556

[455] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *NIPS Conference*, pp. 568–584, 2014.

[456] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv:1312.6034*, 2013.

[457] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1: Foundations. pp. 194–281, 1986.

[458] J. Snoek, H. Larochelle, and R. Adams. Practical bayesian optimization of machine learning algorithms. *NIPS Conference*, pp. 2951–2959, 2013.

[459] R. Socher, C. Lin, C. Manning, and A. Ng. Parsing natural scenes and natural language with recursive neural networks. *ICML Confererence*, pp. 129–136, 2011.

[460] R. Socher, J. Pennington, E. Huang, A. Ng, and C. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 151–161, 2011.

[461] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. *Empirical Methods in Natural Language Processing (EMNLP)*, p. 1642, 2013.

[462] Socher, Richard, Milind Ganjoo, Christopher D. Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. *NIPS Conference*, pp. 935–943, 2013.

[463] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. *NIPS Conference*, 2015.

[464] R. Solomonoff. A system for incremental learning based on algorithmic probability. *Sixth Israeli Conference on Artificial Intelligence, Computer Vision and Pattern Recognition*, pp. 515–527, 1994.

[465] Y. Song, A. Elkahky, and X. He. Multi-rate deep learning for temporal recommendation. *ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 909–912, 2016.

[466] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv:1412.6806*, 2014.
https://arxiv.org/abs/1412.6806

[467] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1), pp. 1929–1958, 2014.

[468] N. Srivastava and R. Salakhutdinov. Multimodal learning with deep Boltzmann machines. *NIPS Conference*, pp. 2222–2230, 2012.

[469] N. Srivastava, R. Salakhutdinov, and G. Hinton. Modeling documents with deep Boltzmann machines. *Uncertainty in Artificial Intelligence*, 2013.

[470] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv:1505.00387*, 2015.
https://arxiv.org/abs/1505.00387

[471] A. Storkey. Increasing the capacity of a Hopfield network without sacrificing functionality. *Artificial Neural Networks*, pp. 451–456, 1997.

[472] F. Strub and J. Mary. Collaborative filtering with stacked denoising autoencoders and sparse inputs. *NIPS Workshop on Machine Learning for eCommerce*, 2015.

[473] S. Sukhbaatar, J. Weston, and R. Fergus. End-to-end memory networks. *NIPS Conference*, pp. 2440–2448, 2015.

[474] Y. Sun, D. Liang, X. Wang, and X. Tang. Deepid3: Face recognition with very deep neural networks. *arXiv:1502.00873*, 2013.
https://arxiv.org/abs/1502.00873

[475] Y. Sun, X. Wang, and X. Tang. Deep learning face representation from predicting 10,000 classes. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1891–1898, 2014.

[476] M. Sundermeyer, R. Schluter, and H. Ney. LSTM neural networks for language modeling. *Interspeech*, 2010.

[477] M. Sundermeyer, T. Alkhouli, J. Wuebker, and H. Ney. Translation modeling with bidirectional recurrent neural networks. *EMNLP*, pp. 14–25, 2014.

[478] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. *ICML Confererence*, pp. 1139–1147, 2013.

[479] I. Sutskever and T. Tieleman. On the convergence properties of contrastive divergence. *International Conference on Artificial Intelligence and Statistics*, pp. 789–795, 2010.

[480] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *NIPS Conference*, pp. 3104–3112, 2014.

[481] I. Sutskever and V. Nair. Mimicking Go experts with convolutional neural networks. *International Conference on Artificial Neural Networks*, pp. 101–110, 2008.

[482] R. Sutton. Learning to Predict by the Method of Temporal Differences, *Machine Learning*, 3, pp. 9–44, 1988.

[483] R. Sutton and A. Barto. Reinforcement Learning: An Introduction. *MIT Press*, 1998.

[484] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *NIPS Conference*, pp. 1057–1063, 2000.

[485] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[486] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

[487] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *AAAI Conference*, pp. 4278–4284, 2017.

[488] G. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. *European Conference on Computer Vision*, pp. 140–153, 2010.

[489] G. Taylor, G. Hinton, and S. Roweis. Modeling human motion using binary latent variables. *NIPS Conference*, 2006.

[490] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. *ACM KDD Conference*, pp. 847–855, 2013.

[491] T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. *ICML Conference*, pp. 1064–1071, 2008.

[492] G. Tesauro. Practical issues in temporal difference learning. *Advances in NIPS Conference*, pp. 259–266, 1992.

[493] G. Tesauro. Td-gammon: A self-teaching backgammon program. *Applications of Neural Networks*, Springer, pp. 267–285, 1992.

[494] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3), pp. 58–68, 1995.

[495] Y. Teh and G. Hinton. Rate-coded restricted Boltzmann machines for face recognition. *NIPS Conference*, 2001.

[496] S. Thrun. Learning to play the game of chess *NIPS Conference*, pp. 1069–1076, 1995.

[497] S. Thrun and L. Platt. Learning to learn. *Springer*, 2012.

[498] Y. Tian, Q. Gong, W. Shang, Y. Wu, and L. Zitnick. ELF: An extensive, lightweight and flexible research platform for real-time strategy games. *arXiv:1707.01067*, 2017. https://arxiv.org/abs/1707.01067

[499] A. Tikhonov and V. Arsenin. Solution of ill-posed problems. *Winston and Sons*, 1977.

[500] D. Tran *et al.* Learning spatiotemporal features with 3d convolutional networks. *IEEE International Conference on Computer Vision*, 2015.

[501] R. Uijlings, A. van de Sande, T. Gevers, and M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2), 2013.

[502] H. Valpola. From neural PCA to deep unsupervised learning. *Advances in Independent Component Analysis and Learning Machines*, pp. 143–171, Elsevier, 2015.

[503] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. *ACM International Conference on Multimedia*, pp. 689–692, 2005. http://www.vlfeat.org/matconvnet/

[504] V. Veeriah, N. Zhuang, and G. Qi. Differential recurrent neural networks for action recognition. *IEEE International Conference on Computer Vision*, pp. 4041–4049, 2015.

[505] A. Veit, M. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. *NIPS Conference*, pp. 550–558, 2016.

[506] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. ICML Confererence, pp. 1096–1103, 2008.

[507] O. Vinyals, C. Blundell, T. Lillicrap, and D. Wierstra. Matching networks for one-shot learning. *NIPS Conference*, pp. 3530–3638, 2016.

[508] O. Vinyals and Q. Le. A Neural Conversational Model. *arXiv:1506.05869*, 2015. https://arxiv.org/abs/1506.05869

[509] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. *CVPR Conference*, pp. 3156–3164, 2015.

[510] J. Walker, C. Doersch, A. Gupta, and M. Hebert. An uncertain future: Forecasting from static images using variational autoencoders. *European Conference on Computer Vision*, pp. 835–851, 2016.

[511] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. *ICML Conference*, pp. 1058–1066, 2013.

[512] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. *ACM KDD Conference*, pp. 1225–1234, 2016.

[513] H. Wang, N. Wang, and D. Yeung. Collaborative deep learning for recommender systems. *ACM KDD Conference*, pp. 1235–1244, 2015.

[514] L. Wang, Y. Qiao, and X. Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4305–4314, 2015.

[515] S. Wang, C. Aggarwal, and H. Liu. Using a random forest to inspire a neural network and improving on it. *SIAM Conference on Data Mining*, 2017.

[516] S. Wang, C. Aggarwal, and H. Liu. Randomized feature engineering as a fast and accurate alternative to kernel methods. *ACM KDD Conference*, 2017.

[517] T. Wang, D. Wu, A. Coates, and A. Ng. End-to-end text recognition with convolutional neural networks. *International Conference on Pattern Recognition*, pp. 3304–3308, 2012.

[518] X. Wang and A. Gupta. Generative image modeling using style and structure adversarial networks. *ECCV*, 2016.

[519] C. J. H. Watkins. Learning from delayed rewards. *PhD Thesis*, King's College, Cambridge, 1989.

[520] C. J. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3–4), pp. 279–292, 1992.

[521] K. Weinberger, B. Packer, and L. Saul. Nonlinear Dimensionality Reduction by Semidefinite Programming and Kernel Matrix Factorization. *AISTATS*, 2005.

[522] M. Welling, M. Rosen-Zvi, and G. Hinton. Exponential family harmoniums with an application to information retrieval. *NIPS Conference*, pp. 1481–1488, 2005.

[523] A. Wendemuth. Learning the unlearnable. *Journal of Physics A: Math. Gen.*, 28, pp. 5423–5436, 1995.

[524] P. Werbos. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. *PhD thesis, Harvard University*, 1974.

[525] P. Werbos. The roots of backpropagation: from ordered derivatives to neural networks and political forecasting (Vol. 1). *John Wiley and Sons*, 1994.

[526] P. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10), pp. 1550–1560, 1990.

[527] J. Weston, A. Bordes, S. Chopra, A. Rush, B. van Merrienboer, A. Joulin, and T. Mikolov. Towards ai-complete question answering: A set of pre-requisite toy tasks. *arXiv:1502.05698*, 2015.
https://arxiv.org/abs/1502.05698

[528] J. Weston, S. Chopra, and A. Bordes. Memory networks. *ICLR*, 2015.

[529] J. Weston and C. Watkins. Multi-class support vector machines. *Technical Report CSD-TR-98-04*, Department of Computer Science, Royal Holloway, University of London, May, 1998.

[530] D. Wettschereck and T. Dietterich. Improving the performance of radial basis function networks by learning center locations. *NIPS Conference*, pp. 1133–1140, 1992.

[531] B. Widrow and M. Hoff. Adaptive switching circuits. *IRE WESCON Convention Record*, 4(1), pp. 96–104, 1960.

[532] S. Wieseler and H. Ney. A convergence analysis of log-linear training. *NIPS Conference*, pp. 657–665, 2011.

[533] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4), pp. 229–256, 1992.

[534] C. Wu, A. Ahmed, A. Beutel, A. Smola, and H. Jing. Recurrent recommender networks. *ACM International Conference on Web Search and Data Mining*, pp. 495–503, 2017.

[535] Y. Wu, C. DuBois, A. Zheng, and M. Ester. Collaborative denoising auto-encoders for top-n recommender systems. *Web Search and Data Mining*, pp. 153–162, 2016.

[536] Z. Wu. Global continuation for distance geometry problems. *SIAM Journal of Optimization*, 7, pp. 814–836, 1997.

[537] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *arXiv:1611.05431*, 2016.
https://arxiv.org/abs/1611.05431

[538] E. Xing, R. Yan, and A. Hauptmann. Mining associated text and images with dual-wing harmoniums. *Uncertainty in Artificial Intelligence*, 2005.

[539] C. Xiong, S. Merity, and R. Socher. Dynamic memory networks for visual and textual question answering. *ICML Confererence*, pp. 2397–2406, 2016.

[540] K. Xu *et al.* Show, attend, and tell: Neural image caption generation with visual attention. *ICML Confererence*, 2015.

[541] O. Yadan, K. Adams, Y. Taigman, and M. Ranzato. Multi-gpu training of convnets. *arXiv:1312.5853*, 2013.
https://arxiv.org/abs/1312.5853

[542] Z. Yang, X. He, J. Gao, L. Deng, and A. Smola. Stacked attention networks for image question answering. *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 21–29, 2016.

[543] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9), pp. 1423–1447, 1999.

[544] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv:1511.07122*, 2015.
https://arxiv.org/abs/1511.07122

[545] H. Yu and B. Wilamowski. Levenberg–Marquardt training. *Industrial Electronics Handbook*, 5(12), 1, 2011.

[546] L. Yu, W. Zhang, J. Wang, and Y. Yu. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. *AAAI Conference*, pp. 2852–2858, 2017.

[547] W. Yu, W. Cheng, C. Aggarwal, K. Zhang, H. Chen, and Wei Wang. NetWalk: A flexible deep embedding approach for anomaly Detection in dynamic networks, *ACM KDD Conference*, 2018.

[548] W. Yu, C. Zheng, W. Cheng, C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang. Learning deep network representations with adversarially regularized autoencoders. *ACM KDD Conference*, 2018.

[549] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv:1605.07146*, 2016. https://arxiv.org/abs/1605.07146

[550] W. Zaremba and I. Sutskever. Reinforcement learning neural turing machines. *arXiv:1505.00521*, 2015.

[551] W. Zaremba, T. Mikolov, A. Joulin, and R. Fergus. Learning simple algorithms from examples. *ICML Confererence*, pp. 421–429, 2016.

[552] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv:1409.2329*, 2014.

[553] M. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv:1212.5701*, 2012. https://arxiv.org/abs/1212.5701

[554] M. Zeiler, D. Krishnan, G. Taylor, and R. Fergus. Deconvolutional networks. *Computer Vision and Pattern Recognition (CVPR)*, pp. 2528–2535, 2010.

[555] M. Zeiler, G. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. *IEEE International Conference on Computer Vision (ICCV)—*, pp. 2018–2025, 2011.

[556] M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *European Conference on Computer Vision*, Springer, pp. 818–833, 2013.

[557] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *arXiv:1611.03530*. https://arxiv.org/abs/1611.03530

[558] D. Zhang, Z.-H. Zhou, and S. Chen. Non-negative matrix factorization on kernels. *Trends in Artificial Intelligence*, pp. 404–412, 2006.

[559] L. Zhang, C. Aggarwal, and G.-J. Qi. Stock Price Prediction via Discovering Multi-Frequency Trading Patterns. *ACM KDD Conference*, 2017.

[560] S. Zhang, L. Yao, and A. Sun. Deep learning based recommender system: A survey and new perspectives. *arXiv:1707.07435*, 2017. https://arxiv.org/abs/1707.07435

[561] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *NIPS Conference*, pp. 649–657, 2015.

[562] J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *arXiv:1609.03126*, 2016. https://arxiv.org/abs/1609.03126

[563] V. Zhong, C. Xiong, and R. Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv:1709.00103*, 2017. https://arxiv.org/abs/1709.00103

[564] C. Zhou and R. Paffenroth. Anomaly detection with robust deep autoencoders. *ACM KDD Conference*, pp. 665–674, 2017.

[565] M. Zhou, Z. Ding, J. Tang, and D. Yin. Micro Behaviors: A new perspective in e-commerce recommender systems. *WSDM Conference*, 2018.

[566] Z.-H. Zhou. Ensemble methods: Foundations and algorithms. *CRC Press*, 2012.

[567] Z.-H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: many could be better than all. *Artificial Intelligence*, 137(1–2), pp. 239–263, 2002.

[568] C. Zitnick and P. Dollar. Edge Boxes: Locating object proposals from edges. *ECCV*, pp. 391–405, 2014.

[569] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv:1611.01578*, 2016.
https://arxiv.org/abs/1611.01578

[570] https://deeplearning4j.org/

[571] http://caffe.berkeleyvision.org/

[572] http://torch.ch/

[573] http://deeplearning.net/software/theano/

[574] https://www.tensorflow.org/

[575] https://keras.io/

[576] https://lasagne.readthedocs.io/en/latest/

[577] http://www.netflixprize.com/community/topic_1537.html

[578] http://deeplearning.net/tutorial/lstm.html

[579] https://arxiv.org/abs/1609.08144

[580] https://github.com/karpathy/char-rnn

[581] http://www.image-net.org/

[582] http://www.image-net.org/challenges/LSVRC/

[583] https://www.cs.toronto.edu/~kriz/cifar.html

[584] http://code.google.com/p/cuda-convnet/

[585] http://caffe.berkeleyvision.org/gathered/examples/feature_extraction.html

[586] https://github.com/caffe2/caffe2/wiki/Model-Zoo

[587] http://scikit-learn.org/

[588] http://clic.cimec.unitn.it/composes/toolkit/

[589] https://github.com/stanfordnlp/GloVe

[590] https://deeplearning4j.org/

[591] https://code.google.com/archive/p/word2vec/

[592] https://www.tensorflow.org/tutorials/word2vec/

[593] https://github.com/aditya-grover/node2vec

[594] https://www.wikipedia.org/

[595] https://github.com/caglar/autoencoders

[596] https://github.com/y0ast

[597] https://github.com/fastforwardlabs/vae-tf/tree/master

[598] https://science.education.nih.gov/supplements/webversions/BrainAddiction/guide/lesson2-1.html

[599] https://www.ibm.com/us-en/marketplace/deep-learning-platform

[600] https://www.coursera.org/learn/neural-networks

[601] https://archive.ics.uci.edu/ml/datasets.html

[602] http://www.bbc.com/news/technology-35785875

[603] https://deepmind.com/blog/exploring-mysteries-alphago/

[604] http://selfdrivingcars.mit.edu/

[605] http://karpathy.github.io/2016/05/31/rl/

[606] https://github.com/hughperkins/kgsgo-dataset-preprocessor

[607] https://www.wired.com/2016/03/two-moves-alphago-lee-sedol-redefined-future/

[608] https://qz.com/639952/googles-ai-won-the-game-go-by-defying-millennia-of-basic-human-instinct/

[609] http://www.mujoco.org/

[610] https://sites.google.com/site/gaepapersupp/home

[611] https://drive.google.com/file/d/0B9raQzOpizn1TkRIa241ZnBEcjQ/view

[612] https://www.youtube.com/watch?v=1L0TKZQcUtA&list=PLrAXtmErZgOeiKm4sgNOkn–GvNjby9efdf

[613] https://openai.com/

[614] http://jaberg.github.io/hyperopt/

[615] http://www.cs.ubc.ca/labs/beta/Projects/SMAC/

[616] https://github.com/JasperSnoek/spearmint

[617] https://deeplearning4j.org/lstm

[618] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[619] https://www.youtube.com/watch?v=2pWv7GOvuf0

[620] https://gym.openai.com

[621] https://universe.openai.com

[622] https://github.com/facebookresearch/ParlAI

[623] https://github.com/openai/baselines

[624] https://github.com/carpedm20/deep-rl-tensorflow

[625] https://github.com/matthiasplappert/keras-rl

[626] http://apollo.auto/

[627] https://github.com/Element-Research/rnn/blob/master/examples/

[628] https://github.com/lmthang/nmt.matlab

[629] https://github.com/carpedm20/NTM-tensorflow

[630] https://github.com/camigord/Neural-Turing-Machine

[631] https://github.com/SigmaQuan/NTM-Keras

[632] https://github.com/snipsco/ntm-lasagne

[633] https://github.com/kaishengtai/torch-ntm

[634] https://github.com/facebook/MemNN

[635] https://github.com/carpedm20/MemN2N-tensorflow

[636] https://github.com/YerevaNN/Dynamic-memory-networks-in-Theano

[637] https://github.com/carpedm20/DCGAN-tensorflow

[638] https://github.com/carpedm20

[639] https://github.com/jacobgil/keras-dcgan

[640] https://github.com/wiseodd/generative-models

[641] https://github.com/paarthneekhara/text-to-image

[642] http://horatio.cs.nyu.edu/mit/tiny/data/

[643] https://developer.nvidia.com/cudnn

[644] http://www.nvidia.com/object/machine-learning.html

[645] https://developer.nvidia.com/deep-learning-frameworks