14

Evolving deep neural networks

Risto Miikkulainen[a,b], Jason Liang[b], Elliot Meyerson[b], Aditya Rawal[c], Dan Fink[b], Olivier Francon[b], Bala Raju[d], Hormoz Shahrzad[b], Arshak Navruzyan[e], Nigel Duffy[f], and Babak Hodjat[b]

[a]*DEPARTMENT OF COMPUTER SCIENCE, THE UNIVERSITY OF TEXAS AT AUSTIN, AUSTIN, TX, UNITED STATES* [b]*COGNIZANT AI LABS, SAN FRANCISCO, CA, UNITED STATES* [c]*AMAZON AWS AI LABS, SANTA CLARA, CA, UNITED STATES* [d]*DEIVA AI, PALO ALTO, CA, UNITED STATES* [e]*LAUNCHPAD, INC., KIHEI, HI, UNITED STATES* [f]*CYNCH AI, SAN FRANCISCO, CA, UNITED STATES*

Chapter outlines

1 Introduction

Large databases (i.e., Big Data) and large amounts of computing power have become readily available since the 2000s. As a result, it has become possible to scale up machine learning systems. Interestingly, not only have these systems been successful in such scale-up, but they have become more powerful. Some ideas that did not quite work before, now do, with million times more compute and data. For instance, deep neural networks

Artificial Intelligence in the Age of Neural Networks and Brain Computing. https://doi.org/10.1016/B978-0-323-96104-2.00002-6

(DNNs), that is, convolutional neural networks [1] and recurrent neural networks (in particular, long short-term memory [LSTM] [2]), which have existed since the 1990s, have improved state of the art significantly in computer vision, speech, language processing, and many other areas [3–5].

As DNNs have been scaled up and improved, they have become much more complex. A new challenge has therefore emerged: How to configure such systems? Human engineers can optimize a handful of configuration parameters through experimentation, but DNNs have complex topologies and hundreds of hyperparameters. Moreover, such design choices matter; often success depends on finding the right architecture for the problem. Much of the recent work in deep learning has indeed focused on proposing different hand-designed architectures for new problems [6–11].

The complexity challenge is not unique to neural networks. Software and many other engineered systems have become too complex for humans to optimize fully. As a result, a new way of thinking about such design has started to emerge. In this approach, humans are responsible for the high-level design, and the details are left for computational optimization systems to figure out. For instance, humans write the overall design of a software system, and the parameters and low-level code are optimized automatically [12]; humans write imperfect versions of programs, and evolutionary algorithms are then used to repair them [13]; humans define the space of possible web designs; and evolution is used to find effective ones [14].

This same approach can be applied to the design of DNN architectures. This problem includes three challenges: how to design the components of the architecture, how to put them together into a full network topology, and how to set the hyperparameters for the components and the global design. These three aspects need to be optimized separately for each new task.

This chapter develops an approach for the automatic design of DNNs. It is based on the existing neuroevolution technique of NEAT [15], which has been successful in evolving topologies and weights of relatively small recurrent networks in the past. In this chapter, NEAT is extended to the coevolutionary optimization of components, topologies, and hyperparameters. The fitness of the evolved networks is determined based on how well they can be trained, through gradient descent, to perform the task. The approach is demonstrated in the standard benchmark tasks of object recognition and language modeling, and in a real-world application of captioning images on a magazine website.

The results show that the approach discovers designs that are comparable to the state of the art, and does it automatically without much development effort. The approach is computationally extremely demanding—with more computational power, it is likely to be more effective and possibly surpass human design. Such power is now becoming available in various forms of cloud computing and grid computing, thereby making the evolutionary optimization of neural networks a promising approach for the future.

2 Background and related work

Neuroevolution techniques have been applied successfully to sequential decision tasks for three decades [16–20]. In such tasks, there is no gradient available, so instead of gradient descent, evolution is used to optimize the weights of the neural network. Neuroevolution is a good approach in particular to partially observable Markov decision process (POMDP) problems because of recurrency. It is possible to evolve recurrent connections to allow disambiguating hidden states.

The weights can be optimized using various evolutionary techniques. Genetic algorithms are a natural choice because crossover is a good match with neural networks: they recombine parts of existing neural networks to find better ones. CMA-ES [21], a technique for continuous optimization, works well on optimizing the weights as well because it can capture interactions between them. Other approaches such as SANE, ESP, and CoSyNE evolve partial neural networks and combine them into fully functional networks [22–24]. Further, techniques such as cellular encoding [25] and NEAT [15] have been developed to evolve the topology of the neural network, which is particularly effective in determining the required recurrence. Neuroevolution techniques have been shown to work well in many tasks in control, robotics, constructing intelligent agents for games, and artificial life [19]. However, because of the large number of weights to be optimized, they are generally limited to relatively small networks.

The evolution has been combined with gradient-descent-based learning in several ways, making it possible to utilize much larger networks. These methods are still usually applied to sequential decision tasks, but gradients from a related task (such as prediction of the next sensory inputs) are used to help search. Much of the work is based on utilizing the Baldwin effect, where learning only affects the selection [26]. Computationally, it is possible to utilize Lamarckian evolution as well, that is, encode the learned weight changes back into the genome [25]. However, care must be taken to maintain diversity so that evolution can continue to innovate when all individuals are learning similar behavior.

The evolution of DNNs differs from this prior work in that it is applied to supervised domains where gradients are available, and evolution is used only to optimize the design of the neural network. Deep neuroevolution is thus more closely related to bi-level (or multilevel) optimization techniques [27]. The idea is to use an evolutionary optimization process at a high level to optimize the parameters of a low-level evolutionary optimization process.

Consider, for instance, the problem of controlling a helicopter through aileron, elevator, rudder, and rotor inputs. This is a challenging benchmark from the 2000s for which various reinforcement learning approaches have been developed [28–30]. One of the most successful ones is single-level neuroevolution, where the helicopter is controlled by a neural network that is evolved through genetic algorithms [31]. The eight parameters of the neuroevolution method (such as mutation rate, probability, and amount; crossover

probability; population and elite size) are optimized by hand. It would be difficult to include more parameters because the parameters interact nonlinearly. A large part of the parameter space thus remains unexplored in the single-level neuroevolution approach. However, a bi-level approach, where a high-level evolutionary process is employed to optimize these parameters, can search this space more effectively [32]. With bi-level evolution, the number of parameters optimized could be extended to 15, which resulted in significantly better performance. In this manner, evolution was harnessed in this example task to optimize a system design that was too complex to be optimized by hand.

Recently, several studies have applied this idea to optimizing DNNs. Originally, due to significant computational demands, they focused on specific parts of the design. For instance, Loshchilov and Hutter [33] used CMA-ES to optimize the hyperparameters of existing DNNs obtaining state-of-the-art results at the time on object recognition. Fernando et al. [34] developed a compositional pattern-producing network (CPPN) [35] to output the weights of an autoencoder neural network. The autoencoder was then trained further through gradient descent, forming gradients for the CPPN training, and its trained weights were then incorporated back into the CPPN genome through Lamarckian adaptation. A related approach was proposed by Zoph and Le [36]: the topology and hyperparameters of a deep network and LSTM network were modified through policy iteration. More recently, Such et al. [37] demonstrated the power of this approach in reinforcement learning tasks, and Real et al. [38] in image classification.

Building on this foundation, a systematic approach to evolving DNNs is developed in this chapter. First, the standard NEAT neuroevolution method is applied to the topology and hyperparameters of convolutional neural networks, and then extended to the evolution of components as well, achieving results comparable to state of the art in the CIFAR-10 image classification benchmark. Second, a similar method is used to evolve the structure of LSTM networks in language modeling, showing that even small innovations in the components can have a significant effect on performance. Third, the approach is used to build a real-world application of captioning images on an online magazine website.

3 Evolution of deep learning architectures

NEAT neuroevolution method [15] is first extended to evolving network topology and hyperparameters of deep neural networks in DeepNEAT, and then further to the coevolution of modules and blueprints for combining them in CoDeepNEAT. The approach is tested in the standard CIFAR-10 benchmark of object recognition, and found to be comparable to the state of the art at the time the experiments were run.

3.1 Extending NEAT to deep networks

DeepNEAT is a most immediate extension of the standard neural network topology-evolution method NEAT to DNN. It follows the same fundamental process as NEAT: first, a population of chromosomes (each represented by a graph) with minimal complexity is

created. Over generations, structure (i.e., nodes and edges) is added to the graph incrementally through mutation. During crossover, historical markings are used to determine how genes of two chromosomes can be lined up. The population is divided into species (i.e., subpopulations) based on a similarity metric. Each species grows proportionally to its fitness and evolution occurs separately in each species.

DeepNEAT differs from NEAT in that each node in the chromosome no longer represents a neuron, but a layer in a DNN. Each node contains a table of real and binary-valued hyperparameters that are mutated through uniform Gaussian distribution and random bitflipping, respectively. These hyperparameters determine the type of a layer (such as a convolutional, fully connected, or recurrent) and the properties of that layer (such as number of neurons, kernel size, and activation function). The edges in the chromosome are no longer marked with weights; instead, they simply indicate how the nodes (layers) are connected together. To construct a DNN from a DeepNEAT chromosome, one simply needs to traverse the chromosome graph, replacing each node with the corresponding layer. The chromosome also contains a set of global hyperparameters applicable to the entire network (such as learning rate, training algorithm, and data preprocessing).

When arbitrary connectivity is allowed between layers, additional complexity is required. If the current layer has multiple parent layers, a merged layer must be applied to the parents in order to ensure that the parent layer's output is the same size as the current layer's input. Typically, this adjustment is done through a concatenation or elementwise sum operation. If the parent layers have mismatched output sizes, all of the parent layers must be downsampled to the parent layer with the smallest output size. The specific method for downsampling is domain dependent. For example, in image classification, a max-pooling layer is inserted after specific parent layers; in image captioning, a fully connected bottleneck layer will serve this function.

During fitness evaluation, each chromosome is converted into a DNN. These DNNs are then trained for a fixed number of epochs. After training, a metric that indicates the network's performance is returned back to DeepNEAT and assigned as fitness to the corresponding chromosome in the population.

While DeepNEAT can be used to evolve DNNs, the resulting structures are often complex and unprincipled. They contrast with typical DNN architectures that utilize the repetition of basic components. DeepNEAT is therefore extended to the evolution of modules and blueprints next.

3.2 Cooperative coevolution of modules and blueprints

Many of the most successful DNNs, such as GoogLeNet and ResNet are composed of modules that are repeated multiple times [5, 6]. These modules often themselves have complicated structure with branching and merging of various layers. Inspired by this observation, a variant of DeepNEAT, called Coevolution DeepNEAT (CoDeepNEAT), is proposed. The algorithm behind CoDeepNEAT is inspired mainly by hierarchical SANE [22] but is also influenced by component-evolution approaches ESP [39] and CoSyNE [24].

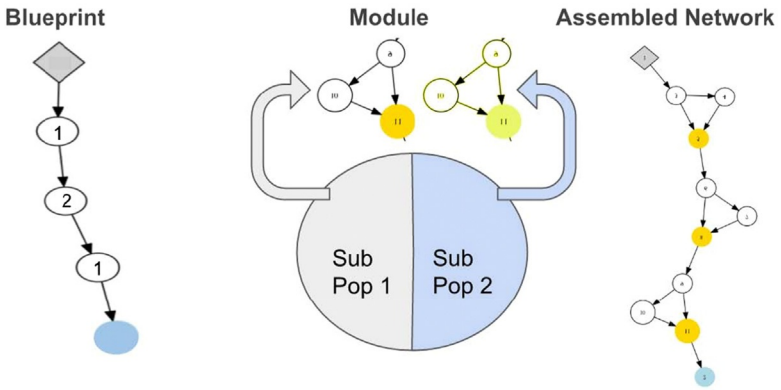

FIG. 1 A visualization of how CoDeepNEAT assembles networks for fitness evaluation. Modules and blueprints are assembled together into a network through the replacement of blueprint nodes with corresponding modules. This approach allows evolving repetitive and deep structures seen in many successful DNNs.

In CoDeepNEAT, two populations of modules and blueprints are evolved separately, using the same methods as described above for DeepNEAT. The blueprint chromosome is a graph where each node contains a pointer to a particular module species. In turn, each module chromosome is a graph that represents a small DNN. Both graphs and the nodes in them evolved. During fitness evaluation, the modules and blueprints are combined to create a larger assembled network (Fig. 1). Each node in the blueprint is replaced with a module is chosen randomly from the species to which that node points. If multiple blueprint nodes point to the same module species, then the same module is used in all of them. The assembled networks are evaluated in a manner similar to DeepNEAT, but the fitnesses of the assembled networks are attributed back to blueprints and modules as the average fitness of all the assembled networks containing that blueprint or module.

CoDeepNEAT can evolve repetitive modular structure efficiently. Furthermore, because small mutations in the modules and blueprints often lead to large changes in the assembled network structure, CoDeepNEAT can explore more diverse and deeper architectures than DeepNEAT. An example application to the CIFAR-10 domain is presented next.

3.3 Evolving DNNs in the CIFAR-10 benchmark

In this experiment, CoDeepNEAT was used to evolve the topology of a convolutional neural network (CNN) to maximize its classification performance on the CIFAR-10 dataset, a common image classification benchmark. The dataset consists of 50,000 training images and 10,000 testing images. The images consist of 32×32 color pixels and belong to 1 of 10 classes. For comparison, the neural network layer types were restricted to those used by Snoek et al. [40] in their Bayesian optimization of CNN hyperparameters. Also following Snoek et al., data augmentation consisted of converting the images from RGB to HSV color space, adding random perturbations, distortions, and crops, and converting them back to RGB color space.

CoDeepNEAT was initialized with population of 25 blueprints and 45 modules. From these two populations, 100 CNNs were assembled for fitness evaluation in every generation. Each node in the module chromosome represents a convolutional layer. Its hyperparameters determine the various properties of the layer and whether additional max-pooling or dropout layers are attached (Table 1). In addition, a set of global hyperparameters were evolved for the assembled network. During fitness evaluation, the 50,000 images were split into a training set of 42,500 samples and a validation set of 7500 samples. Since training a DNN is computationally very expensive, each network was trained for eight epochs on the training set. The validation set was then used to determine classification accuracy, that is, the fitness of the network. After 72 generations of evolution, the best network in the population was returned.

After evolution was complete, the best network was trained on all 50,000 training images for 300 epochs, and the classification error measured. This error was 7.3%, comparable to the 6.4% error reported by Snoek et al. [40]. Interestingly, because only limited training could be done during evolution, the best network evolved by CoDeepNEAT trains very fast. While the network of Snoek et al. takes over 30 epochs to reach 20% test error and over 200 epochs to converge, the best network from evolution takes only 12 epochs to reach 20% test error and around 120 epochs to converge. This network utilizes the same modules multiple times, resulting in a deep and repetitive structure typical of many successful DNNs (Fig. 2).

Table 1 Node and global hyperparameters evolved in the CIFAR-10 domain.

Node hyperparameter	**Range**
Number of filters	[32, 256]
Dropout rate	[0, 0.7]
Initial weight scaling	[0, 2.0]
Kernel size	{1, 3}
Max pooling	{True, false}
Global hyperparameter	**Range**
Learning rate	[0.0001, 0.1]
Momentum	[0.68, 0.99]
Hue shift	[0, 45]
Saturation/value shift	[0, 0.5]
Saturation/value scale	[0, 0.5]
Cropped image size	[26, 32]
Spatial scaling	[0, 0.3]
Random horizontal flips	{True, false}
Variance normalization	{True, false}
Nesterov accelerated gradient	{True, false}

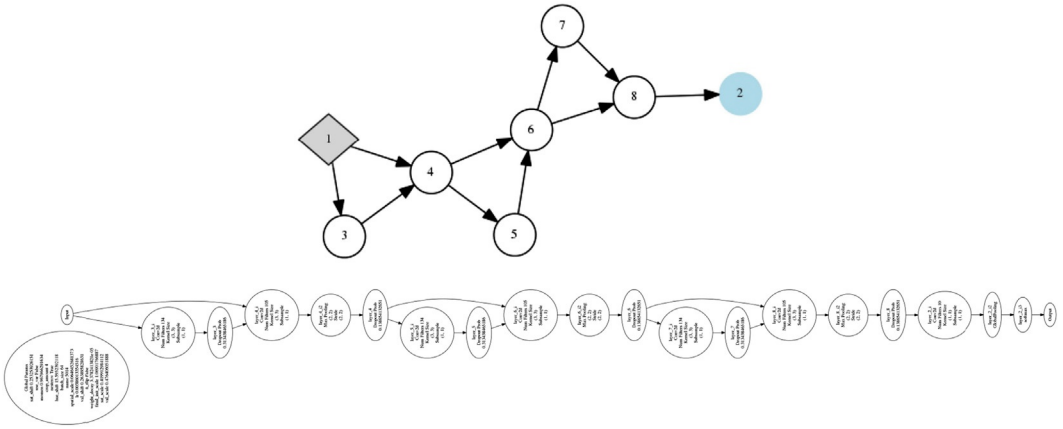

FIG. 2 *Top*: Simplified visualization of the best network evolved by CoDeepNEAT for the CIFAR-10 domain. Node 1 is the input layer, while Node 2 is the output layer. The network has a repetitive structure because its blueprint reuses the same module in multiple places. *Bottom*: A more detailed visualization of the same network.

4 Evolution of LSTM architectures

Recurrent neural networks, in particular those utilizing LSTM nodes, are another powerful approach to DNN. Much of the power comes from the repetition of LSTM modules and the connectivity between them. In this section, CoDeepNEAT is extended with mutations that allow searching for such connectivity, and the approach is evaluated in the standard benchmark task of language modeling.

4.1 Extending CoDeepNEAT to LSTMs

LSTM consists of gated memory cells that can integrate information over longer time scales (as compared to simply using recurrent connections in a neural network). LSTMs have been shown powerful in supervised sequence processing tasks such as speech recognition [41] and machine translation [42].

Recent research on LSTMs has focused in two directions: finding variations of individual LSTM memory unit architecture [36, 43–47], and discovering new ways of stitching LSTM layers into a network [48–50]. Both approaches have improved performance over vanilla LSTMs, with the best recent results achieved through network design. The CoDeepNEAT method incorporates both approaches: neuroevolution searches for both new LSTM units and their connectivity across multiple layers at the same time.

CoDeepNEAT was slightly modified to make it easier to find novel connectivities between LSTM layers. Multiple LSTM layers are flattened into a neural network graph that is then modified by neuroevolution. There are two types of mutations: one enables or disables a connection between LSTM layers, and the other adds or removes skip connections between two LSTM nodes. Recently, skip connections have led to performance improvements in deep neural networks, which suggest that they could be useful for LSTM

networks as well. Thus, neuroevolution modifies both the high-level network topology and the low-level LSTM connections.

In each generation, a population of these network graphs (i.e., blueprints), consisting of LSTM variants (i.e., modules with possible skip connections), is created. The individual networks are then trained and tested with the supervised data of the task. The experimental setup and the language modeling task are described next.

4.2 Evolving DNNs in the language modeling benchmark

One standard benchmark task for the LSTM network is language modeling, that is, predicting the next word in a large text corpus. The benchmark utilizes the Penn Tree Bank (PTB) dataset [51], which consists of 929,000 training words, 73,000 validation words, and 82,000 test words. It has 10,000 words in its vocabulary.

A population of 50 LSTM networks was initialized with uniformly random initial connection weights within $[-0.05, 0.05]$. Each network consisted of two recurrent layers (vanilla LSTM or its variants) with 650 hidden nodes in each layer. The network was unrolled in time up to 35 steps. The hidden states were initialized to zero. The final hidden states of the current minibatch were used as the initial hidden state of the subsequent minibatch (successive minibatches sequentially traverse the training set). The size of each minibatch was 20. For fitness evaluation, each network was trained for 39 epochs. A learning rate decay of 0.8 was applied at the end of every six epochs; the dropout rate was 0.5. The gradients were clipped if their maximum norm (normalized by minibatch size) exceeded 5. Training a single network took about 200 min on a GeForce GTX 980 GPU card.

After 25 generations of neuroevolution, the best network improved the performance of the PTB dataset by 5% (test-perplexity score 78) as compared to the vanilla LSTM [52]. As shown in Fig. 3, this LSTM variant consists of a feedback skip connection between the memory cells of two LSTM layers. This result is interesting because it is similar to a recent hand-designed architecture that also outperforms vanilla LSTM [48].

The initial results thus demonstrate that CoDeepNEAT with just two LSTM-specific mutations can automatically discover improved LSTM variants. It is likely that expanding the search space with more mutation types and layer and connection types would lead to further improvements.

5 Application case study: Image captioning for the blind

In a real-world case study, the vision and language capabilities of CoDeepNEAT were combined to build a real-time online image-captioning system. In this application, CoDeepNEAT searches for architectures that learn to integrate image and text representations to produce captions that blind users can access through existing screen readers. This application was implemented for a major online magazine website. Evolved networks were trained with the open-source MSCOCO image-captioning dataset [53], along with a new dataset collected for this website.

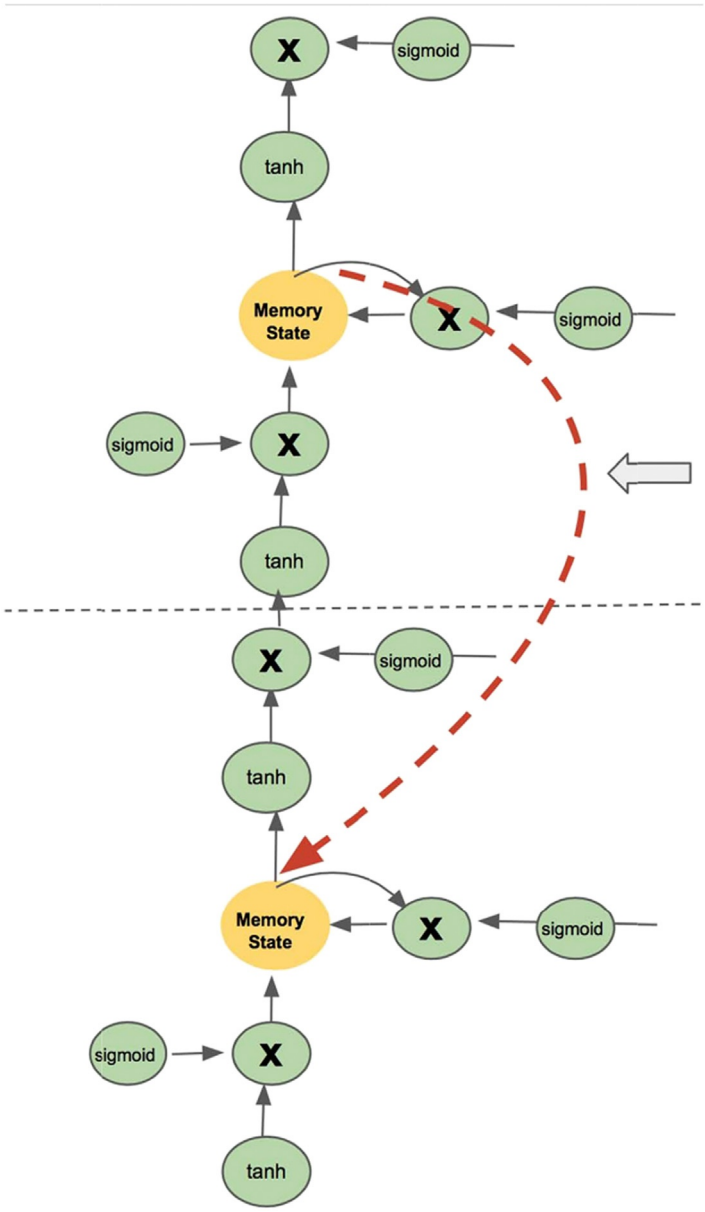

FIG. 3 The best-performing LSTM variant after 25 generations of neuroevolution. It includes a novel skip connection between the two memory cells, resulting in a 5% improvement over the vanilla LSTM baseline. Such improvements are difficult to discover by hand; CoDeepNEAT with LSTM-specific mutation searches for them automatically.

5.1 Evolving DNNs for image captioning

Deep learning has recently provided state-of-the-art performance in image captioning, and several diverse architectures have been suggested [54–58]. The input to an image-captioning system is a raw image, and the output is a text caption intended to describe the contents of the image. In deep learning approaches, a convolutional network is usually used to process the image, and recurrent units, often LSTMs, to generate coherent sentences with long-range dependencies.

As is common in existing approaches, the evolved system uses a pretrained ImageNet model [5] to produce initial image embeddings. The evolved network takes an image embedding as input, along with a one-hot text input. As usual, in training the text input contains the previous word of the ground-truth caption; in inference, it contains the previous word generated by the model [54, 56].

In the initial CoDeepNEAT population, the image and text inputs are fed to a shared embedding layer, which is densely connected to a softmax output over words. From this simple starting point, CoDeepNEAT evolves architectures that include fully connected layers, LSTM layers, sum layers, concatenation layers, and sets of hyperparameters associated with each layer, along with a set of global hyperparameters (Table 2). In particular, the well-known Show and Tell image-captioning architecture [54] is in this search space, providing a baseline with which evolution results can be compared. These components and the glue that connects them are evolved as described in Section 3.2, with 100 networks trained in each generation. Since there is no single best-accepted metric for evaluating captions, the fitness function is the mean across three metrics (BLEU, METEOR, and CIDEr) [53] normalized by their baseline values. Fitness is computed over a holdout set of 5000 images, that is, 25,000 image-caption pairs.

Table 2 Node and global hyperparameters evolved for the image-captioning case study.

Global hyperparameter	**Range**
Learning rate	[0.0001, 0.1]
Momentum	[0.68, 0.99]
Shared embedding size	[128, 512]
Embedding dropout	[0, 0.7]
LSTM recurrent dropout	{True, false}
Nesterov momentum	{True, false}
Weight initialization	{Glorot normal, He normal}
Node hyperparameter	**Range**
Layer type	{Dense, LSTM}
Merge method	{Sum, Concat}
Layer size	{128, 256}
Layer activation	{ReLU, Linear}
Layer dropout	[0, 0.7]

To keep the computational cost reasonable, during evolution the networks are trained for only six epochs, and only with a random 100,000 image subset of the 500,000 MSCOCO image-caption pairs. As a result, there is evolutionary pressure toward networks that converge quickly. The best-resulting architectures train to near convergence six times faster than the baseline Show and Tell model [54]. After evolution, the optimized learning rate is scaled by one-fifth to compensate for the subsampling.

5.2 Building the application

The images in MSCOCO are chosen to depict "common objects in context." The focus is on a relatively small set of objects and their interactions in a relatively small set of settings. The Internet as a whole, and the online magazine website in particular, contain many images that cannot be classified as "common objects in context." Other types of images from the magazine include staged portraits of people, infographics, cartoons, abstract designs, and *iconic* images, that is, images of one or multiple objects *out of context* such as on a white or patterned background. Therefore, an additional dataset of 17,000 image-caption pairs was constructed for the case study, targeting iconic images in particular. About 4000 images were first scraped from the magazine website, and 1000 of them were identified as iconic. Then, 16,000 images that were visually similar to those 1000 were retrieved automatically from a large image repository. A single ground-truth caption for each of these 17,000 images was generated by human subjects through MightyAI.[a] The holdout set for evaluation consisted of 100 of the original 1000 iconic images, along with 3000 other images.

During evolution, networks were trained and evaluated only on the MSCOCO data. The best architecture from evolution was then trained from scratch on both MSCOCO and MightyAI datasets in an iterative alternating approach: one epoch on MSCOCO, followed by five epochs on MightyAI until the maximum performance was reached on the MightyAI holdout data. Beam search was then used to generate captions from the fully trained models. Performance achieved using the MightyAI data demonstrates the ability of evolved architectures to generalize to domains toward which they were not evolved.

Once the model was fully trained, it was placed on a server where it can be queried with images to caption. A JavaScript snippet was written that a developer can embed in his/her site to automatically query the model to caption all images on a page. This snippet runs in an existing Chrome extension for custom scripts and automatically captions images as the user browses the web. These tools add captions to the "alt" field of images, which screen readers can then read to blind Internet users (Fig. 4).

5.3 Image-captioning results

Trained in parallel on about 100 GPUs, each generation took around 1 h to complete. The most fit architecture was discovered in generation 37 (Fig. 5). Among the components in

[a]See https://mty.ai/computer-vision/.

FIG. 4 An iconic image from an online magazine captioned by an evolved model. The model provides a suitably detailed description without any unnecessary context.

its unique structure are six LSTM layers, four summing merge layers, and several skip connections. A single module consisting of two LSTM layers merged by a sum is repeated three times. There is a path from the input through dense layers to the output that bypasses all LSTM layers, providing the softmax with a more direct view of the current input. The motif of skip connections with a summing merge is similar to residual architectures that are currently popular in deep learning [6, 59]. However, the high-level structure of multiple parallel pathways is an innovation of evolutionary search and is rarely seen in human-designed architectures. During the search, such parallel pathways provide robust performance as the architecture varies, and are thus likely to be discovered. During the performance, they possibly represent multiple parallel hypotheses that are combined to the most likely final answer. This architecture performs better than the hand-tuned baseline [54] when trained on the MSCOCO data alone (Table 3).

However, a more important result is the performance of this network on the magazine website. Because no suitable automatic metrics exist for the types of captions collected for the magazine website (and existing metrics are very noisy when there is only one reference caption), captions generated by the evolved model on all 3100 holdout images were manually evaluated as correct, mostly correct, mostly incorrect, and incorrect (Fig. 6). Fig. 7 shows some examples of good and bad captions for these images.

The model is not perfect, but the results are promising. The evolved network is correct or mostly correct on 63% of iconic images and 31% of all images. There are many known improvements that can be implemented, including ensembling diverse architectures generated by evolution, fine-tuning the ImageNet model, using a more recent ImageNet model, and performing beam search or scheduled sampling during training [60]; preliminary experiments with ensembling alone suggest improvements of about 20%. For this application, it is also important to include methods for automatically evaluation caption quality and filtering captions that would give an incorrect impression to a blind user. However, even without these additions, the results demonstrate that it is now possible to develop practical applications through evolving DNNs.

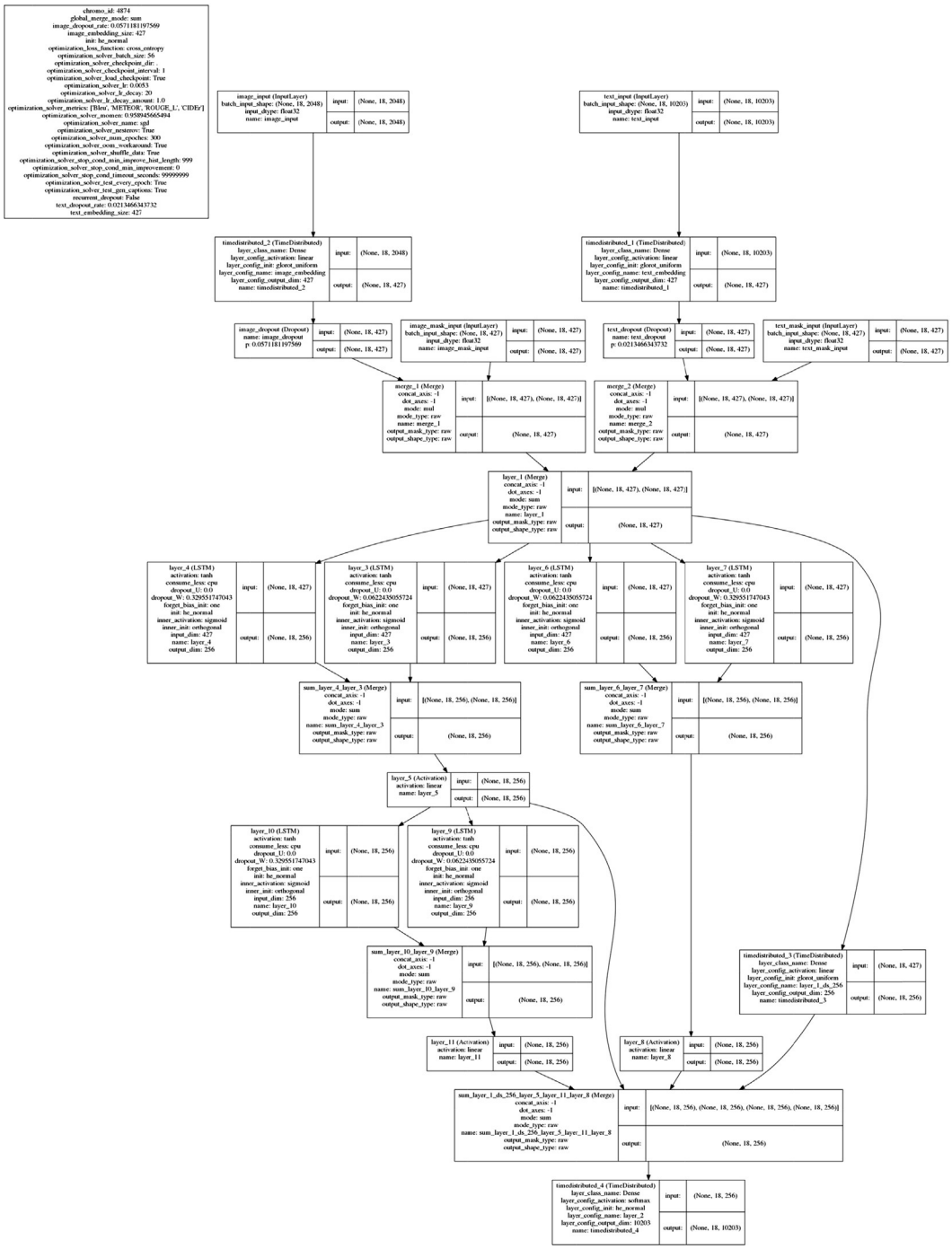

FIG. 5 The most fit architecture found by evolution. It consists of three LSTM-based modules as well as a parallel densely connected pathway. Multiple pathways are robust during search and can represent multiple hypotheses during performance. Such a high-level design is different from most hand-designed architectures and represents an innovation of evolutionary search.

Table 3 The evolved network improves over the hand-designed baseline when trained on MSCOCO alone.

Model	BLEU-4	CIDEr	METEOR
DNGO [40]	26.7	–	–
Baseline [54]	27.7	85.5	23.7
Evolved	**29.1**	**88.0**	**23.8**

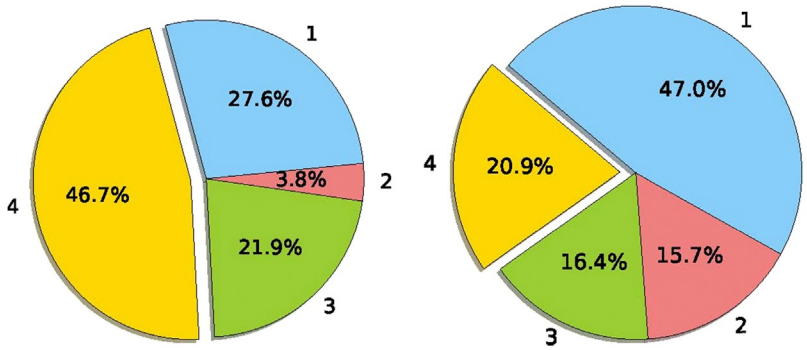

FIG. 6 Results for captions generated by an evolved model for the online magazine images rated from 1 to 4, with 4—correct, 3—mostly correct, 2—mostly incorrect, 1—incorrect. *Left*: On iconic images, the model is able to get about one-half correct. *Right*: On all images, the model gets about one-fifth correct. The superior performance on iconic images shows that it is useful to build supplementary training sets for specific image types.

6 Discussion and future work

The results in this chapter show that the evolutionary approach to optimizing deep neural networks is feasible: The results are comparable to hand-designed architectures in benchmark tasks, and it is possible to build real-world applications based on the approach. It is important to note that the approach has not yet been pushed to its full potential. It takes a couple of days to train each deep neural network on a typical GPU, and over the course of evolution, thousands of them need to be trained. Therefore, the results are limited by the available computational power.

Interestingly, since it was necessary to train networks only partially during evolution, evolution is biased toward discovering fast learners instead of top performers. This is an interesting result on its own: Evolution can be guided with goals other than simply accuracy, including training time, execution time, or memory requirements of the network. On the other hand, if it was possible to train the networks further, it would be possible to identify top performers more reliably, and final performance would likely improve. In order to speed up evolution, other techniques may be developed. For instance, it may be possible to seed the population with various state-of-the-art architectures and modules, instead of having to rediscover them during evolution.

Significantly more computational resources are likely to become available in the near future. Cloud-based services offer GPU computation at a reasonable cost, and they are

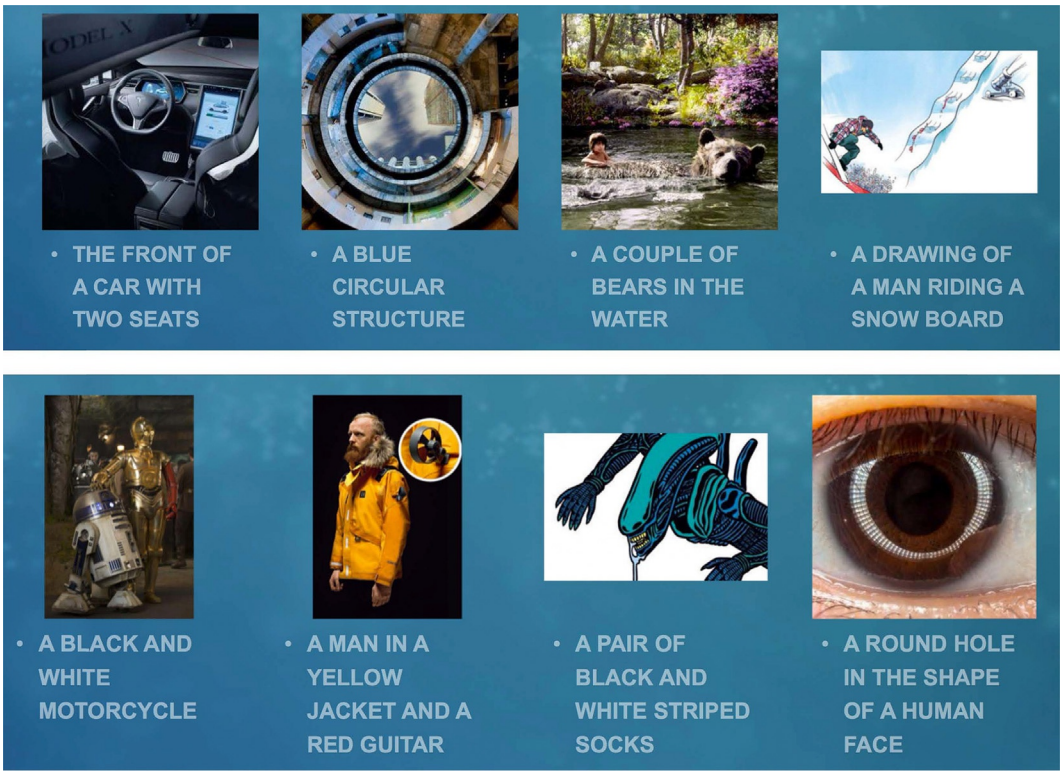

FIG. 7 *Top*: Four good captions. The model is able to abstract about ambiguous images and even describe drawings, along with photos of objects in context. *Bottom*: Four bad captions. When it fails, the output of the model still contains some correct sense of the image.

becoming cheaper. Not many approaches can take advantage of such power, but the evolution of deep learning neural networks can. The search space of different components and topologies can be extended, and more hyperparameters be optimized. While cloud services such as AutoML provide some such functionality and benefit from the increasing computing power as well, they currently represent brute-force approaches that are unlikely to scale as well. Given the results in this chapter, the evolutionary approach is likely to discover designs that are superior to those that can be developed by hand today; it is also likely to make it possible to apply deep learning to a wider array of tasks and applications in the future.

7 Conclusions

Evolutionary optimization makes it possible to design more complex deep learning architectures that can be designed by hand. The topology, components, and hyperparameters of the architecture can all be optimized simultaneously to fit the requirements of the task, resulting in superior performance. Such automated design can make new applications of

deep learning possible in vision, speech, language, and other areas. The approach is computationally demanding and thus offers one way to put the anticipated increases in computing power to good use.

References

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

[2] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.

[3] R. Collobert, J. Weston, A unified architecture for natural language processing: deep neural networks with multitask learning, in: Proceedings of the 25th International Conference on Machine learning, ACM, 2008, pp. 160–167.

[4] A. Graves, A.-R. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2013, pp. 6645–6649.

[5] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proc. of CVPR, 2016, pp. 2818–2826.

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

[7] J.Y. Ng, M.J. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, G. Toderici, Beyond short snippets: deep networks for video classification, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 4694–4702.

[8] Z. Che, S. Purushotham, K. Cho, D. Sontag, Y. Liu, Recurrent neural networks for multivariate time series with missing values, Sci. Rep. 8 (2018) 6085.

[9] G. Huang, Z. Liu, K.Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[10] M. Tan, Q. Le, EfficientNet: rethinking model scaling for convolutional neural networks, in: International Conference on Machine Learning, 2019, pp. 6105–6114.

[11] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, MN, 2019, pp. 4171–4186, https://doi.org/10.18653/v1/N19-1423.

[12] H. Hoos, Programming by optimization, Commun. ACM 55 (2012) 70–80.

[13] C.L. Goues, T. Nguyen, S. Forrest, W. Weimer, GenProg: a generic method for automatic software repair, ACM Trans. Softw. Eng. 38 (2012) 54–72.

[14] R. Miikkulainen, N. Iscoe, A. Shagrin, R. Cordell, S. Nazari, C. Schoolland, M. Brundage, J. Epstein, R. Dean, G. Lamba, Conversion rate optimization through evolutionary computation, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2017), ACM, New York, NY, 2017, pp. 1193–1199.

[15] K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evol. Comput. 10 (2002) 99–127.

[16] D.J. Montana, L. Davis, Training feedforward neural networks using genetic algorithms, in: Proceedings of the 11th International Joint Conference on Artificial Intelligence, 1989, pp. 762–767.

[17] D. Floreano, P. Dürr, C. Mattiussi, Neuroevolution: from architectures to learning, Evol. Intell. 1 (2008) 47–62.

[18] X. Yao, Evolving artificial neural networks, Proc. IEEE 87 (9) (1999) 1423–1447.

[19] J. Lehman, R. Miikkulainen, Neuroevolution, Scholarpedia 8 (2013) 30977.

[20] K.O. Stanley, J. Clune, J. Lehman, R. Miikkulainen, Designing neural networks through evolutionary algorithms, Nat. Mach. Intell. 1 (2019) 24–35.

[21] C. Igel, Neuroevolution for reinforcement learning using evolution strategies, in: Proceedings of the 2003 Congress on Evolutionary Computation, IEEE Press, Piscataway, NJ, 2003, pp. 2588–2595.

[22] D.E. Moriarty, R. Miikkulainen, Forming neural networks through efficient and adaptive coevolution, Evol. Comput. 5 (1997) 373–399.

[23] F. Gomez, R. Miikkulainen, Incremental evolution of complex general behavior, Adapt. Behav. 5 (1997) 317–342.

[24] F. Gomez, J. Schmidhuber, R. Miikkulainen, Accelerated neural evolution through cooperatively coevolved synapses, J. Mach. Learn. Res. 8 (2008) 937–965.

[25] F. Gruau, D. Whitley, Adding learning to the cellular development of neural networks: evolution and the Baldwin effect, Evol. Comput. 1 (1993) 213–233.

[26] G.E. Hinton, S.J. Nowlan, How learning can guide evolution, Complex Syst. 1 (1987) 495–502.

[27] A. Sinha, P. Malo, P. Xu, K. Deb, A bilevel optimization approach to automated parameter tuning, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2014), Vancouver, BC, Canada, 2014, pp. 847–854.

[28] J. Bagnell, J. Schneider, Autonomous helicopter control using reinforcement learning policy search methods, in: Proceedings of the International Conference on Robotics and Automation 2001, IEEE, 2001.

[29] A.Y. Ng, H.J. Kim, M. Jordan, S. Sastry, Autonomous helicopter flight via reinforcement learning, in: Advances in Neural Information Processing Systems 16, 2004.

[30] P. Abbeel, A. Coates, M. Quigley, A.Y. Ng, An application of reinforcement learning to aerobatic helicopter flight, in: Advances in Neural Information Processing Systems 19, 2007.

[31] R. Koppejan, S. Whiteson, Neuroevolutionary reinforcement learning for generalized control of simulated helicopters, Evol. Intell. 4 (2011) 219–241.

[32] J.Z. Liang, R. Miikkulainen, Evolutionary bilevel optimization for complex control tasks, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2015), July, 2015, pp. 871–878.

[33] I. Loshchilov, F. Hutter, CMA-ES for hyperparameter optimization of deep neural networks, in: International Conference on Learning Representations (ICLR-2016) Workshop Track, 2016.

[34] C. Fernando, D. Banarse, F. Besse, M. Jaderberg, D. Pfau, M. Reynolds, M. Lactot, D. Wierstra, Convolution by evolution: differentiable pattern producing networks, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016), ACM, New York, NY, 2016.

[35] K. Stanley, Compositional pattern producing networks: a novel abstraction of development, Genet. Program Evolvable Mach. 8 (2) (2007) 131–162, https://doi.org/10.1007/s10710-007-9028-8.

[36] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, in: International Conference on Learning Representations, 2017.

[37] F.P. Such, V. Madhavan, E. Conti, J. Lehman, K.O. Stanley, J. Clune, Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning, arXiv:1712.06567 (2017).

[38] E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2019, pp. 4780–4789.

[39] F. Gomez, R. Miikkulainen, Solving non-Markovian control tasks with neuroevolution, in: Proceedings of the 11th International Joint Conference on Artificial Intelligence, Kaufmann, San Francisco, 1999, pp. 1356–1361.

[40] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M.M.A. Patwary, M. Prabhat, R.P. Adams, Scalable Bayesian optimization using deep neural networks, in: Proc. of ICML, 2015, pp. 2171–2180.

[41] A. Graves, N. Jaitly, Towards end-to-end speech recognition with recurrent neural networks, in: Proc. 31st ICML, 2014, pp. 1764–1772.

[42] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, in: ICLR, 2015.

[43] J. Bayer, D. Wierstra, J. Togelius, J. Schmidhuber, Evolving memory cell structures for sequence learning, in: Artificial Neural Networks ICANN, 2009, pp. 755–764.

[44] R. Jozefowicz, W. Zaremba, I. Sutskever, An empirical exploration of recurrent network architectures, in: Proceedings of the 32nd International Conference on Machine Learning, 2015, pp. 2342–2350.

[45] K. Cho, B. van Merrienboer, Ç. Gulcehre, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1724–1734.

[46] K. Greff, R.K. Srivastava, J. Koutník, B.R. Steunebrink, J. Schmidhuber, LSTM: A Search Space Odyssey, IEEE Trans. Neural Netw. Learn. Syst. 28 (2017) 222–232.

[47] A. Rawal, R. Miikkulainen, Discovering gated recurrent neural network architectures, in: H. Iba, N. Noman (Eds.), Deep Neural Evolution—Deep Learning With Evolutionary Computation, Springer, 2020, pp. 233–251.

[48] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Gated feedback recurrent neural networks, in: Proceedings of the 32nd International Conference on Machine Learning, 2015.

[49] N. Kalchbrenner, I. Danihelka, A. Graves, Grid long short-term memory, arXiv:1507.01526 (2015).

[50] J.G. Zilly, R.K. Srivastava, J. Koutník, J. Schmidhuber, Recurrent highway networks, in: Proceedings of the 34th International Conference on Machine Learning, 2017, pp. 6346–6357.

[51] M.P. Marcus, M.A. Marcinkiewicz, B. Santorini, Building a large annotated corpus of English: the Penn Treebank, Comput. Linguist. 19 (2) (1993) 313–330.

[52] W. Zaremba, I. Sutskever, O. Vinyals, Recurrent neural network regularization, arXiv:1409.2329 (2014).

[53] X. Chen, H. Fang, T.Y. Lin, R. Vedantam, S. Gupta, P. Dollar, C.L. Zitnick, Microsoft COCO captions: data collection and evaluation server, arXiv:1504.00325 (2015).

[54] O. Vinyals, A. Toshev, S. Bengio, D. Erhan, Show and tell: a neural image caption generator, in: Proc. of CVPR, 2015, pp. 3156–3164.

[55] K. Xu, J. Ba, R. Kiros, K. Cho, A.C. Courville, R. Salkhutdinov, R.S. Zemel, Y. Bengio, Show, attend and tell: neural image caption generation with visual attention, in: Proc. of ICML, 2015, pp. 77–81.

[56] A. Karpathy, L. Fei-Fei, Deep visual-semantic alignments for generating image descriptions, in: Proc. of CVPR, 2015, pp. 3128–3137.

[57] Q. You, H. Jin, Z. Wang, C. Fang, J. Luo, Image captioning with semantic attention, in: Proc. of CVPR, 2016, pp. 4651–4659.

[58] R. Vedantam, S. Bengio, K. Murphy, D. Parikh, G. Chechik, Context-aware captions from context-agnostic supervision, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[59] C. Szegedy, S. Ioffe, V. Vanhoucke, A. Alemi, Inception-v4, inception-ResNet and the impact of residual connections on learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2016.

[60] O. Vinyals, A. Toshev, S. Bengio, D. Erhan, Show and tell: lessons learned from the 2015 MSCOCO image captioning challenge, Trans. Pattern Anal. Mach. Intell. 39 (2016) 652–663.