1.1 Deep Learning Hardware: Past, Present, and Future

Yann LeCun, Facebook AI Research and New York University, New York, NY

Abstract
Historically, progress in neural networks and deep learning research has been greatly influenced by the available hardware and software tools. This paper identifies trends in deep learning research that will influence hardware architectures and software platforms of the future.

1 The Past

Modern AI is powered by deep learning (DL), whose origins go back to early experiments with electronic neural nets in the 1950s. DL is based on four simple ideas: (1) complex functions can be efficiently constructed by assembling simple parameterized functional blocks (such as linear operators and point-wise non-linearities) into multi-layer computational graphs; (2) The desired function can be learned from examples by adjusting the parameters; (3) the learning procedure minimizes an objective function through a gradient-based method; (4) the gradient can be computed efficiently and automatically through the back-propagation algorithm (backprop, for short), which is nothing more than a practical application of chain rule to compute the partial derivatives of the objective with respect to all the parameters in the system by propagating signals backwards through the network. The key advantage of DL is that it alleviates the need to hand-design a feature extractor, as would be required with traditional machine learning and pattern-recognition methods. When trained for a particular task, DL systems automatically learn multi-layer hierarchical representations of the data that are suitable for the task. One may wonder why such a simple idea as backprop was not popularized until the late 1980s, and why such a natural idea as DL did not become widely used until the early 2010s, fueling the recent wave of interest in AI.

1.1 The Unreasonable Influence of Hardware and Software Tools on Progress

Several lessons can be drawn from the 60+ year history of neural networks: (1) new ideas seem limited by the available hardware (and software); (2) specialized hardware does not enable new methods; (3) biological inspiration is fruitful but can be a trap. From the 1950s to the 1980s, neural-network models used binary neurons: the McCullloch-Pitts neuron which computes a weighted sum of its inputs and passes it through a sign function. This was largely because multiplications were prohibitively expensive at the time, whether implemented in analog electronics, in digital circuits, or in software. With binary inputs, additions are sufficient to compute the neurons'weighted sums. One reason backprop did not emerge until the late 1980s is that it required the use of neurons with continuous non-linearities (such as sigmoids), which did not become practical until workstation performance approached one million floating-point multiply-accumulate operations per second.

More puzzling, while the basic techniques of DL have been around since the late 1980s, what caused it to lose popularity within the research community in the mid-1990s? (1) the performance of computers at the time; (2) the small number of applications for which collecting large labeled datasets was cost effective; (3) the effort involved in developing flexible neural net simulators; (4) the reluctance of many research institutions at the time to distribute open source software. What sparked its sudden resurgence around 2013? There are four main factors: (1) improved methods; (2) larger datasets with many samples and many categories; (3) Low-cost TFLOPS-class general-purpose GPUs (GPGPUs); (4) open-source libraries with interpreted language front-ends (Torch, Theano, cuda-convNet, Caffe). The first three of these enabled record-breaking results in image recognition and speech recognition, while the last one allowed these results to be easily replicated because they incorporated all the engineering "tricks" necessary to get DL models to work. Arguably, prior to this, computer vision and ML research was limited to what was implementable easily in Matlab.

Interestingly, there were attempts to build dedicated hardware architectures for neural networks before the advent of GPGPUs. But none of them were successful, in part because they lacked flexibility [1, 2] and/or were designed for particular types of neural networks that had no proven practical use [3, 4, and references therein]. What has accounted for the success of GPGPU for DL is their wide availability, generality, programmability, and well-supported software stacks.

The important lesson is that *Hardware capabilities and software tools both motivate and limit the type of ideas that AI researchers will imagine and will allow themselves to pursue. The tools at our disposal fashion our thoughts more than we care to admit*.

In other words, what the AI hardware and software communities will produce over the next few years will shape AI research for the coming decades.

1.2 Neural-Network Hardware of Yesteryear

The history of neural networks is inextricably linked with hardware. The original 1957 Perceptron was a dedicated analog computer whose weights were implemented with motorized potentiometers [5], and the rival model, Adaline, was implemented with electrochemical "memistors" [6]. Early adaptive equalizers based on the Adaline algorithms used relays to represent weights [7]. Interest in neural networks had waned following the publication of Minsky and Papert's book"Perceptrons"in 1969 [8], and for two decades, hardware development for "trainable" systems was confined to adaptive-filter applications.

Then, in the mid 1980s, A second wave of interest in neural networks took off, following work on Hopfield networks, Boltzmann Machines, and the popularization of backprop [9, 10, 11]. In 1985, the Adaptive Systems Research Department was created at Bell Laboratories in Holmdel, NJ, under the leadership of Lawrence D. Jackel. This group, which I joined in 1988, spent a decade developing a series of neural-network chips. The evolution of their technology gives us an idea of the constraints brought about by hardware considerations. For example, an analog vector-matrix multiplication can simply be implemented as a resistor array, with separate rows for positive and negative coefficients. In 1986, the group build a 6×6 micron, 12×12 resistor array using e-beam lithography (see Figure 1.1.1). But it quickly became clear that non-programmability was a major limitation and that line amplifiers, I/O circuitry, and signal conversion would defeat the purpose of manufacturing very small resistors [1]. Next, the group built a 54-neuron mixed analog-digital chip. Each neuron had 54 ternary weights (−1, 0, +1) with analog summing. The chip could perform simple feature extraction on binary images, but its speed was limited by I/O bandwidth [2].

When I joined in late 1988, I developed the first convolutional neural net (ConvNet or CNN for short) and obtained excellent results on handwritten character recognition tasks (zip codes) [12, 13]. The ConvNet architecture was loosely inspired by that of the ventral pathway in the visual cortex and was designed to process data that comes to us in the form of an array (possibly, multidimensional) in which nearby values are correlated: image, video, audio, text, and so on. ConvNets are composed of two main types of stages: convolution stages and pooling stages (see Figure 1.1.2). In a ConvNet designed for image recognition, the input, output, and intermediate layer activations are all 3-dimensional arrays (often called tensors). Each "slice" of the input tensor is a color channel. Each slice of the first layer output (called a feature map) is obtained by performing a discrete convolution of each of the input slices with different convolution kernel (also called filters). The results are added and passed through a half-wave rectification non-linearity (also called a ReLU for Rectified Linear Unit). The coefficient of the convolution kernels, whose size if typically 5×5 or 3×3 are subject to learning. A convolutional filter followed by a ReLU detects a particular motif regardless of its location on the input and produces outputs that are equivariant to shifts (that is when the input shifts, the output shifts accordingly). The subsequent pooling layer reduces the spatial resolution of the feature maps by aggregating values within a neighborhood using a max or L_p norm operation. Pooling windows are stepped by more than one pixel, resulting in a lower-resolution feature map. Pooling makes the representation robust to small shifts in the location of distinctive features.

Multiple alternated stages of convolutions and pooling with increasing numbers of feature maps and decreasing spatial resolutions are stacked, producing outputs that are influenced by a large part of the input image. ConvNet are often trained in a supervised manner using a form of stochastic gradient descent (SGD). An image is shown to the network and the output is computed. The output is compared to the desired output (representing a label for the image) using a loss function. The gradient of this loss with respect to all the filter coefficients in the network is computed through the *back-propagation procedure* which propagates gradient signals *backwards* through a version of the network in which all arrows have been reversed and in which each operator multiplies these signals by its Jacobian matrix. The essential characteristic of ConvNets is their ability to learn hierarchical representations of the signal automatically.

Because of the prevalence of convolutions in image analysis, the next generation of Bell Labs chips, called Net32k, contained 256 neurons, each with 128 ternary weights, capable of 320×10^9 synaptic operations per second [14]. Multiple rows could be combined with power-of-two weights to form 64 neurons with 128 4-bit weights. An important new feature was the presence of shift registers to allow fast convolution with multiple filters, while minimizing access to external memory. Almost simultaneously, another chip called ANNA (Analog Neural-Network Accelerator) was built specifically to run convolutional networks [15]. It contained 64 neurons with 64 weights each. The weights were stored on capacitors and refreshed from an external RAM through a DAC with 6-bit accuracy. The activations were represented digitally by 3 bits. The synapses were essentially multiplying DACs, and the sum was done in analog. The chip ran at 20MHz and was capable of 4×10^9 synaptic operations per second. Just as in Net32k, ANNA had shift registers that allows it to run realistic ConvNet architectures with 130,000 connections in about 1ms, or about 500 times faster than the best workstations at the time, with comparable accuracy on zip code digit recognition [16]. Both Net32k and ANNA came close to being used in commercially deployed applications (postal envelopes and bank-check reading), but were eventually dropped for software implementations on floating-point DSPs. The lessons one can draw from this history is that: (1) bringing exotic fabrication technologies and architectural concepts to market is difficult; (2) system-level performance is considerably more important than raw speed. It took only two years after ANNA for FPGAs to become sufficiently powerful to run ConvNets at the same speed using non-conventional number representations [17].

By the mid-1990s, ConvNet applications had been widely deployed by AT&T and its subsidiaries for reading documents, and by the late 1990s, somewhere between 10% and 20% of all bank checks in the US were automatically read by the ConvNet-based system developed by our team [18]. But by the mid 1990s, interest in neural networks had waned in the machine-learning research community and a second “neural network winter” set in, that was to last over a decade.

The reasons for this winter are somewhat linked with all of hardware, software, and data. Neural networks required a lot of data, but, in the pre-Internet days “large” datasets were available only for a few tasks, such as handwriting and speech recognition. Second, computer hardware was limited. Furthermore, the workhorse of ML research, Unix workstations from SUN or SGI, were typically capable of only 10MFLOPS. Correspondingly, training a simple ConvNet for handwritten character recognition could take weeks. Third, software tools had to be built from the ground up. This was before the pre-eminence of Matlab, Python, and other interactive languages suitable for numerical work. Léon Bottou and I started developing a neural-network simulator with a home-grown Lisp-dialect front end in 1987. This system, called SN, eventually inaugurated the idea of neural networks as computational graphs of standard parameterized modules with automatic differentiation [19], a concept eventually picked up by frameworks like Theano, Torch, Caffe, and now TensorFlow, and PyTorch. This was before it was common-place for companies to release code in open source, and it is not until 2002 that it was open-sourced under the name Lush, as we were leaving AT&T. By then, few people were interested in neural networks.

At the tail end of this second winter, efforts to implement ConvNets on FPGA resumed at my NYU lab with the Xilinx Virtex4-based CNP project in 2008 [20], and the Virtex6-based NeuFlow project in 2010 [21]. NeuFlow revived the idea of dataflow stream processing in which a number of configurable operators can be dynamically assigned different functions, such as convolution, non-linear mapping, pooling/subsampling or arithmetic operations, all interconnected with suitable FIFOs to chain multiple operations without having to write intermediate results to external memory. Performance when running a ConvNet was roughly 150×10^9 operations per second, consuming 10W. Dataflow stream processors have become a popular design for ConvNet accelerators, such as the Eyeriss project [23, 24] which exploits the high level of data reuse in convolution operations.

Around 2003, Geoffrey Hinton (University of Toronto), Yoshua Bengio (University of Montréal) and I (having just left industry and joined NYU), with funding from the Canadian Institute for Advanced Research, started a series of research projects, workshops, and summer schools with a deliberate goal of reviving the interest of the ML community in neural networks. By 2007, enough researchers became interested again, following compelling new results in unsupervised layer-wise learning of very deep networks [25]. That is when the domain was rebranded as Deep Learning. Later, innovations such as using rectifying non-linearities (ReLU) instead of sigmoids and using dropout for regularization enabled purely supervised backprop training of very deep networks (with a dozen layers). By 2009, groups at Microsoft, Google, and IBM, were achieving significant reductions in error rates in speech recognition by replacing acoustic models based on Gaussian Mixture Models with deep networks [26]. Within 18 months, these systems were deployed commercially on Android phones and other platforms. Simultaneously, Collobert and Weston, then at NEC Labs, showed that a type of ConvNet architecture could produce vector representations of words that yielded record-breaking results on various natural language processing tasks [27, 28]. But the NLP community was initially skeptical and deep learning did not become dominant in NLP until quite recently.

Meanwhile, ConvNets were starting to produce record-breaking results on a number of image recognition tasks, such as semantic segmentation [29] and pedestrian detection [30], but the results were largely ignored by the computer vision community. Then in late 2012, Alex Krizhevsky working in Geoffrey Hinton’s lab produced a very efficient implementation of ConvNets on GPUs. He was not the first person to do so (researchers at Microsoft, Stanford, IDSIA, and other places had done it before), but his implementation was efficient for very large and very deep ConvNet, allowing the team to win the ImageNet competition (object recognition in images with 1000 categories, and 1.3M training samples), reducing the top-5 error rate from 25% to 16% [31]. This sent shockwaves through the computer vision community. The team made its code available in open source, and within 2 years, almost everyone in the field was using ConvNets.

The lesson from this experience is that, *The availability of suitable hardware and simple-to-use open-source software is critical to the wide adoption of a class of methods by the community. Good results are essential but not sufficient.*

2 The Need for DL Hardware

Is DL-specific hardware really necessary? The answer is a resounding yes. One interesting property of DL systems is that the larger we make them, the better they seem to work. While this property is true for networks trained with supervised learning, the trend is to rely increasingly on unsupervised, self-supervised, weakly supervised or multi-task learning, for which larger networks perform even better. The demands on DL-specific hardware will undoubtedly increase.

There are five use cases with different hardware requirements: (1) DL research and development; (2) off-line training of DL models for production; inference on servers in data centers; (4) inference on mobile devices and embedded systems; (5) on-line learning on servers and mobile devices.

2.1 DL Research and Development: The requirement is for HPC-type multi-node machines, each hosting multiple GPUs or other flexible/programmable devices with fast 32-bit floating-point performance (FP-32). The communication network must be high bandwidth and low latency to allow for the parallelization of training large models on large datasets. Using FP-32 is necessary because one must be sure that when an experiment fails, it is not because of a lack of numerical accuracy. This use case requires high flexibility, programmability, extensive libraries, and efficient MPI-style communication libraries. Price and power consumption are relatively secondary to performance and flexibility. For most types of DL models, parallelization can be done easily by distributing multiple training samples across processing elements, GPUs, or cluster nodes. However, batching multiple training samples at the level of a GPU is neither desirable, nor always possible. The best architectures are those that can be saturated with the smallest batch of samples.

2.2 Off-Line Training of Well-Understood Models: Once a model architecture has been tested and deployed, the process of retraining it periodically as new data becomes available can be optimized. It is possible to perform routine training on specialized hardware with reduced-precision arithmetics, such as 16-bit float (supported by NVIDIA GPUs and Google's TPU), fixed point, or more exotic number systems. Requirements for low-latency training and high levels of parallelization are less stringent than in R&D scenarios.

2.3 Inference on Servers: Today, much of the computational load of DL systems is spent running neural networks in data centers. However, the volume is growing quickly. For example, Facebook performs 3×10^{14} predictions per day (some of which are run on mobile devices). Many are relatively "simple" neural networks with sparse inputs used for newsfeed and advertisement ranking and for text classification [32]. But a lot of computation goes into larger ConvNets for image, video, and speech understanding, as well as for language translation. Every day, users upload 2 to 3 billion photos on Facebook. Each photo goes through a handful of ConvNets within 2 seconds of being uploaded. A large ConvNet trunk extract features used for generic tagging, objectionable content filtering (nudity and violence), search, and so on. Other ConvNets perform OCR of text in images (to detect hate speech), face detection and recognition, caption generation for the visually impaired, and a few other tasks. As communication services, such as live video, further expand, large-scale spatio-temporal ConvNets are being deployed to perform action recognition [33], speech recognition for subtitling, and language translation [34], all in real time with minimum latency. For this use case, power consumption and cost are important, flexibility and raw performance are secondary, and communication latency is unimportant. The ideal architecture is a specialized DL-inference accelerator sitting in a standard data-center server node. Since much of the computation is spent performing convolution, a convolutional net accelerator working on individual samples (not batches) is ideal. The requirements of the automotive industry for autonomous driving systems are somewhat similar, with considerably more stringent requirements on latency.

2.4 Inference on Mobile and Embedded Devices: The ubiquity of smartphones, and the upcoming availability of self-contained wearable devices for augmented reality (AR) and virtual reality (VR) are putting heavy demands on DL-inference accelerators with very-low power consumption. Real-time tasks require that the DL system be run on the device without the latency of a round-trip to a server. Applications include feature tracking and 3D reconstruction for AR, object segmentation/recognition, OCR in natural scenes, real-time language translation, and speech-based virtual assistants. Beyond mobile and wearables, low-cost DL chips will appear in cameras, appliances, autonomous surveillance and ground maintenance systems, and toys.

2.5 Rethinking Arithmetics: Given the robustness of DL systems to arithmetic imprecision, there is a distinct possibility that using unconventional number representations can improve efficiency for inference on servers and embedded devices. For example, [35] shows that an 8-bit logarithmic number system combined with a Kulisch accumulator leads to significant reduction in power consumption, while yielding negligible degradation in accuracy for a ResNet-50 network trained on ImageNet. Extended to 16 bits, this number system shows significant reduction in both power and silicon area over standard 16 bit floating point formats.

3 Present and Future Deep-Learning Architectures

The word "architecture" in the context of DL designates the graph of functional modules, not the architecture of the underlying hardware. Typical DL architectures are composed of a number of basic modules: multiple convolutions in 1D, 2D, and 3D; linear operators (matrices); linear operators applied to sparse inputs (word embedding lookup tables for NLP); divisive normalization; element-wise functions; pooling/subsampling; element-wise operators; bilinear operators (multiplicative interactions for attention); and so on. Low-level operations are often performed on a batch of multiple samples, simply because parallelization is simple. But there is no algorithmic reason to batch multiple samples. Thus, much of the DL R&D activity goes into designing architectures that are appropriate for a class of problems. Such popular families of architectures include ConvNets, multi-layer LSTM, Transformer Networks, and architectures with "attention" (multiplicative interactions).

3.1 DL Architectures Today

Video, image, and speech recognition, as well as language translation and NLP, use a variety of ConvNet architectures. Figure 1.1.3, from [22], shows the top-1 accuracy on ImageNet of various ConvNet designs as a function of number of operations. In such practical applications, much of the computation is spent performing convolutions in the lower layers. The upper layers are typically less compute bound but more memory bound [24]. The spatio-temporal resolution of layer activations typically diminishes in the upper layers, but the number of channels or feature types typically increases.

In computer-vision applications, the trend is to apply ConvNets to an entire image so as to detect, segment, and recognize objects of any size at any location [37, 38, 39, 40, 62]. In such networks, all the layers are convolutional, though some of the top layers, sometimes called "fully connected, can be viewed as convolutions with a 1×1 kernel.

An increasingly popular class of ConvNet architectures for image segmentation, reconstruction, and object detection are the so-called Feature U-Net, Feature Pyramid Network, RetinaNet, and variants [39, 40, 41]. They can be viewed as a ConvNet encoder topped by a "reverse ConvNet" decoder whose role is to produce an image at the same resolution as the input (see Figure 1.1.4). They contain skipping connections from each layer of the encoder to the layer of the corresponding resolution in the decoder. The number of applications of this type of architecture is likely to increase in image annotation, autonomous driving, and medical image analysis [42].

Similar architectures are used in the context of image generation and video prediction [43, 46]. Video prediction is a subject of wide interest because it may allow future systems, such as robots or self-driving cars, to predict what is going to happen in their environment and to plan accordingly [44, 45].

In translation and language understanding, the Transformer Network architecture is increasingly popular [36]. It makes extensive use of multiplicative interactions.

3.2 Architectural Elements of Future DL Systems

We are witnessing an evolution in the types of architecture proposed by DL researchers, which may determine what hardware will be required in the near future. Generally speaking, the evolution is towards more sophisticated network architectures, *dynamic* network architectures that change with each new input in a data-dependent way, inputs and internal states that are not regular tensors, but are graphs whose nodes and edges are annotated with numerical objects (including tensors).

3.2.1 Dynamic Networks, Differentiable Programming: A relatively recent concept in DL is the idea of dynamic networks. Regular DL systems use a static network of parameterized modules. But, in increasingly many applications, the network architecture is dynamic and changes for every new data point. In effect, dynamic DL systems can be seen as the execution trace of a program, with conditionals and loops that are input-dependent. DL frameworks such as PyTorch record a "tape" of this execution trace, which can be played backwards to back-propagate gradients through the program. This method is known as "autograd". The phrase "differentiable programming" designates the process of writing a program with calls to parameterized functions that automatically compute the gradient of the function's output with respect to the parameters, allowing the function to be finalized through learning. Dynamic networks are particularly useful in a variety of applications: for natural-language processing, for data that does not come in the form of a fixed-sized tensor, for systems that need to activate parts of a large network on demand in a data-dependent way (such as the Multi-scale DenseNet architecture [47] shown in Figure 1.1.4) and for "reasoning" networks whose output is another network specifically designed to answer a particular question [48, 49] (see Figure 1.1.5).

3.2.2 Neural Networks on Graphs: One of the most exciting recent developments in DL is neural networks on graphs [50]. Many problems are difficult to represent with fixed-size tensors or variable-length sequences of tensors, but are better represented by graphs whose arcs and nodes are annotated by tensors. This suggests the use of networks of differentiable modules whose inputs and outputs are annotated graphs. The idea goes back to Graph Transformer Networks, built to recognize character strings [18]. But recent incarnations of graph neural networks have been applied to 3D meshes, social networks, gene-regulation networks, and chemical molecules. Convolution operations can easily be defined on irregular graphs: they are defined as diagonal operators in the eigenspace of the graph Laplacian, which is a generalization of the Fourier transform. We foresee an increase in the usage of such networks for a wide variety of applications, which are likely to violate the assumptions of current DL hardware.

3.2.3 Graph Embedding Networks: Increasingly, DL is used for large-scale embedding of knowledge bases. For example, using a large knowledge graph composed of triplets (subject, relation, object), such as ("Barak Obama", "was born in", "Hawaii") one may train a network to rate such triplets or to predict one of the elements from the other two. A special case of this consists in learning a vector for each object and subject, such as a simple scalar-valued operation between the vectors (distance) will predict the presence or absence of a particular relation between the object or subject. These methods, applied on a large scale, are particularly efficient for recommender systems, and can use hyperbolic metric spaces to represent hierarchical categories [51].

3.2.4 Memory-Augmented Networks: To endow DL systems with the ability to reason, they need a short-term memory, to be used as an episodic memory, or a scratchpad/working memory. For example, if a system is to answer questions about a series of events (described as a text), it must be able to store the story in a memory and retrieve the relevant bits to answer a particular question. This led to the memory-network architecture [52, 53] in which a recurrent neural nework is augmented by what amounts to a differentiable associative memory circuit (see Figure 1.1.5). This associative memory can be quite large and requires finding the nearest neighbors to a key vector very efficiently. As DL systems are increasingly used for high-level cognitive tasks, such memory modules will become commonplace and very large, requiring hardware support.

3.2.5 Complex Inference and Search: Most of today's DL systems simply produce an output given an input. But complex reasoning requires that the output variable actually be an *input* to a scoring network whose scalar output (akin to energy) indicates the incompatibility between the input and an output proposal. An inference procedure must search for the output value that minimizes the energy. This type of model is called an *energy-based model* [54]. If the energy-minimizing inference procedure is gradient-based, inference hardware will need to support back-propagation.

3.2.6 Sparse Activations: As the size of DL systems grows, it is likely that the modules' activations will become increasingly sparse, with only a subset of variables of a subset of modules being activated at any one time. This is akin to how the brain represents information: on average, neurons in the brain are at 2% of their maximum activation, and most neurons are quiet most of the time, which is good for power dissipation. Examples of explicitly sparse networks already exist (for processing volumetric imaging data [56]).

3.2.7 Overall:

New architectural concepts such as dynamic networks, graph data, associative-memory structures, and inference-through-minimization procedures are likely to affect the type of hardware architectures that will be required in the future.

4 The Revolution will not be Supervised

With all the hype around the new AI and DL, the way machines learn today is vastly less efficient than the way humans and animals learn. Almost all practical applications of DL use supervised learning (SL), in which the system is fed the desired output during training, with a tiny minority using reinforcement learning (RL). Most humans are capable of learning to drive a car in about 30 hours of training without ever causing accidents. In contrast, current model-free RL methods would likely require millions of hours of practice, with numerous accidents, for an autonomous car to learn to drive. This is not a problem in easy-to-simulate fully-observable environments with discrete state, such as the game of go or chess. But, it does not work in the real world! Obviously, our current learning paradigms are missing a key ingredient.

One hypothesis is that this missing ingredient is self-supervised learning. The bulk of learning in humans and animals is self-supervised: we learn enormous amounts of background knowledge about how the world works by observation in the first days, weeks, and months of life. In particular, we learn intuitive physics and the properties of the physical world. By the age of 9 months, babies understand object permanence, stability, animate vs inanimate objects, stability, gravity, inertia, and so on. The ability to predict what is going to happen in the world is what allows us to learn to drive without causing accidents: our world model allows us to anticipate the consequences of our actions, to maintain the car on the road, and to avoid disasters.

The idea of self-supervised learning is to train a machine to predict any subset of its input from other subsets (with a possible overlap between the subsets). For example, given a 6-frame video clip, one could train a DL system to predict the last two frames from the first four.

Why should SSL be more efficient than either RL or SL? In RL, the system produces an output (often an action or sequence of actions) and gets in return a single scalar value representing the "reward" for this action. Learning a complex task in this scenario requires a very large number of trials, and a large number of errors. While the process works fine for fully-observable games (such as chess and go) where millions of trials can be generated through self-play, it is largely impractical in the real world. A model-free RL system would require millions of hours of driving and numerous crashes to train a car to drive itself. The number of trials required is large because the feedback from the environment is information-poor. In SL, the system is given the correct answer, generally in the form of a target output vector. While this is less information-poor than in RL, it still requires a lot of training samples to capture the essence of the problem. On the other hand, SSL asks the machine to predict a large amount of information in the form of a high-dimensional signal (such as a whole video frame). More complex models with more parameters can be learned with a given number of samples or trials. The main difficulty is that predicting the future of a video is not achievable exactly because the world is not entirely predictable. If one uses a least-square criterion to train a video predictor, the resulting predictions are blurry frames: an average of all the possible futures. To make sharp predictions, one must have a set of latent variables that, when passed through a predictor, parameterize the set of plausible predictions. One technique used to train such models is Generative Adversarial Networks (GAN) [59], which for training uses two networks simultaneously: a *generator* that makes predictions using observations and a source of random vectors drawn from a known distribution, with a *discriminator* whose role is to produce a scalar energy indicating whether a generated prediction is plausible or not. The discriminator is trained to distinguish real data (low energy) from generated predictions (high energy). The generator trains itself to produce predictions that the discriminator cannot tell are fake. To do so, the generator uses the gradient of the discriminator's output energy with respect to its input to compute how to modify its predictions, and thereby modify its parameters. Variations of GANs have produced stunning results in image generation [61, 62]. Other latent-variable generative models, such as Variational Auto-Encoders [60] and regularized latent variable models [45] have also produced good results.

One hope is that training a system to predict videos will allow it to discover much of the hidden regularities, geometry, and physics of the world, such as the fact that the scenery changes in particular ways as the camera moves, and that certain objects occlude others and can move independently. Such predictions can be done in pixel space [43, 44], or in higher-level representations (such as instance segmentation maps obtained by a pre-trained system [46]).

The use of predictive models that not only predict the evolution of the environment, but also predict the consequences of actions, is key to reducing the number of trials a system needs to learn a skill. I predict that self-supervised latent-variable predictive models will be the centerpiece of intelligent systems based on model-predictive control and model-based reinforcement learning for such applications as robotic grasping and manipulation [44] and autonomous driving [45]. Figure 1.1.6 shows a latent-variable predictive model that predicts a visual representation of the surroundings of a car. This kind of model can be used to predict multiple scenarios of how surrounding cars are going to move, and to plan a driving policy accordingly.

If self-supervised learning eventually allows machines to learn vast amounts of background knowledge about how the world works through observation, one may hypothesize that some form of *machine common sense* could emerge! One form of common sense is our ability to fill in the blanks, using our knowledge of the structure and constraints of the world.

Future DL systems will largely be trained using a form of self-supervised learning. These systems will be much larger than they are today, because the amount of data with which they can be trained (e.g. raw video) is essentially unlimited. Such systems will eventually be trained to acquire vast amounts of background knowledge so as to acquire a form of common sense. New high-performance hardware will be required to enable such progress.

5 Requirements for Future DL Hardware and Software

5.1 How Will DL Software Evolve?

Clearly, what is needed is a software framework for *differentiable programming* that is both interactive, flexible, dynamic, and efficient. Although frameworks such as PyTorch, TensorFlow, and others are moving in that direction, the main obstacle is that people love Python, largely because of its gigantic set of libraries. But Python is very slow and memory hungry. It is often impractical to develop high-volume applications or embedded applications that rely on Python at runtime. However, for static compute graphs, there is no issue: one can export the graph to adhere to a standard format, such ONNX (Open Neural Net Exchange), and use one of the numerous ONNX-compliant backends. On the other hand, for dynamic networks, there are two main options: One is to provide a compiler for a sufficiently large subset of Python that can produce Python-independent executables for DL (such as Torch.Jit in the recently-released PyTorch-1.0 [64]). This may also require an auxiliary domain-specific language to specify low-level numerical operations (on tensors and graphs) such as Tensor Comprehensions [55]; A second option is to design a suitable compilable language from scratch. It would have to be interactive and dynamic, have safe parallelism, and use type inference as much as possible, perhaps something resembling Julia or Skip [63] with good support for scientific computing. However, dedicated user's desire to access the vast repository of Python libraries will limit its potential adoption.

5.2 Hardware for Training

One problem is that sparsity, architecture dynamicity, and modules that manipulate non-tensor data (graphs), break the assumption that one can perform computation on batches of identically-sized samples. Unfortunately, with current hardware, batching is what allows us to reduce most low-level neural network operations to matrix products, and thereby to reduce the memory access-to-computation ratio. Thus, we will need new hardware architectures that can function efficiently with a batch size of one. As well, handling sparse structured data is another requirement. Increasingly, input data will come to us in a variety of forms, beyond tensors, such as graphs annotated with tensors and symbols.

Down the line, one can imagine architectures and learning algorithms that favor sparse activations in the network. When most units are off most of the time, it may become advantageous to make our hardware *event driven*, so that only the units that are activated consume resources. Such sparse networks, such as Submanifold Sparse ConvNets (implemented in software) have been shown to be very effective for processing sparse data, such as 3D scenes, which are represented by voxel arrays that are largely empty [56]. Sparse activation is one of the features that makes the brain so power-efficient.

5.3 Hardware for Inference

While demand for data-center and cloud-based inference will grow, future DL applications will increasingly run on mobile phones, wearables, home appppliances, vehicles, IoT devices, and robots. Applications in augmented and virtual reality and telepresence will require extremely low-power ASICs for DL inference for such things as real-time/low-latency object tracking, 3D re-construction, instance labeling, facial reconstruction, predictive compression and display.

In the short and medium term, the bulk of the computation will be convolutions. Since batching is out of the question, hardware will have to exploit the regularities of convolutions instead of being mere matrix-product engines.

Ultimately, the solution to power constraints may well be the exploitation of sparse activations, perhaps using event-based computation. In any case, it may exploit the use of exotic number representations (the 8-bit logarithmic representation of [35]).

6 The Long Term Outlook

In the long run, could we see a return to analog implementations? Perhaps programmable resistor technology will become sufficiently compact, reliable, durable, and configurable for DL applications. But since this would require one unmovable physical memory cell per parameter in the network, only activations could be circulated (assuming they are converted to digital representation), and hardware multiplexing would be limited to sections that share weights (as in the ANNA chip). It is very unclear whether analog implementations provide any power dissipation advantages over digital, and current evidence seems to point in the opposite direction.

A number of authors have been advocating architectures with spiking neurons. Unfortunately, the performance of spiking neuron circuits seems considerably inferior to that of traditional digital architectures for realistic ConvNet-type networks [57]. Current learning algorithms do not take advantage of the peculiarities of spiking networks, and no spiking-neuron learning algorithms has been shown to come close to the accuracy of backprop with continuous representations.

The important trends discussed in this paper include: (1) more self-supervised learning, resulting in larger network architectures; (2) dynamic network resulting from *differentiable programs* whose architecture changes for each new sample; (3) the need for hardware that is efficient for batch-size 1, implying the end of reliance of matrix products as the lowest-level operator; (3) exotic number representation for inference on low-power hardware; (4) very large networks with very sparse activations, that new architectures could exploit for power reduction; (5) new operators such as fast K-nearest neighbors for (differentiable) associative-memory modules; (6) networks that manipulate annotated graphs instead of tensors. However, chances are that the bulk of the computation in future DL systems will still consist primarily of convolutions.

References

[1] L.D. Jackel, R.E. Howard, H.P. Graf, B. Straughn, J.S. Denker, "Artificial Neural Networks for Computing". Journal of Vacuum Science & Technology B: Microelectronics Processing and Phenomena, 4(1), pp. 61-63, 1986.

[2] H. Graf, P. de Vegvar, "A CMOS Associative Memory Chip Based on Neural Networks", *ISSCC*, pp. 304- 305, 1987.

[3] G. Indiveri, et al., "Neuromorphic Silicon Neuron Circuits", *Frontiers in Neuroscience*, 5, p. 73, 2011.

[4] S.B. Furber, F. Galluppi, S. Temple, L. Plana, "The Spinnaker Project", Proceedings of the IEEE, 102(5), pp. 652-665, 2014.

[5] F. Rosenblatt, "The Perceptron, A Perceiving and Recognizing Automaton (Project Para)". Cornell Aeronautical Laboratory, 1957.

[6] B. Widrow, W.H. Pierce, J.B. Angell. "Birth, Life, and Death in Microelectronic Systems", IRE Trans. Mil. Electron., 1051(3), pp. 191- 201, 1961.

[7] R.W. Lucky, "Automatic Equalization for Digital Communication", Bell System Technical Journal, 44(4), pp. 547-588, 1965.

[8] M. Minsky, S.A. Papert, "Perceptrons: An Introduction to Computational Geometry". MIT press, 1969.

[9] J.J. Hopfield. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities". Proceedings of the National Academy of Sciences, 79(8), pp. 2554-2558, 1982.

[10] G.E. Hinton, T.J. Sejnowski, "Optimal Perceptual Inference", Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp. 448-453, June 1983.

[11] D.E. Rumelhart, G.E. Hinton, R.J. Williams, "Learning Representations by Back-Propagating Errors. Nature, 323(6088), pp. 533, 1986.

[12] Y. LeCun, B.E. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition", Neural Computation, 1(4), pp. 541-551, 1989.

[13] Y. LeCun, B.E. Boser, J.S. Denker, D. Henderson, R.E. Howard, W.E. Hubbard, L.D. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network", NIPS, pp. 396-404, 1989.

[14] H.P. Graf, R. Janow, D. Henderson, R. Lee, "Reconfigurable Neural Net chip with 32K Connections", Advances in Neural Information Processing Systems, pp. 1032-1038, 1991.

[15] B.E. Boser, E. Sackinger, J. Bromley, Y. Le Cun, L.D. Jackel, "An Analog Neural Network Processor with Programmable Topology". IEEE Journal of Solid-State Circuits, 26(12), pp. 2017-2025, 1991.

[16] E. Sackinger, B.E. Boser, J. Bromley, Y. LeCun, L.D. Jackel, "Application of the ANNA Neural Network Chip to High-Speed Character Recognition", IEEE Transactions on Neural Networks, 3(3), pp. 498-505, 1992.

[17] J. Cloutier, E. Cosatto, S. Pigeon, F.R. Boyer, P.Y. Simard, "VIP: An FPGA-Based Processor for Image Processing and Neural Networks", Proc. of Int. Conf. Microelectronics for Neural Networks, pp. 330-336, 1996.

[18] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-Based Learning Applied to Document Recognition", Proceedings of the IEEE, 86(11), pp. 2278-2324, 1998.

[19] L. Bottou, P. Gallinari, "A Framework for the Cooperation of Learning Algorithms", Advances in Neural Information Processing Systems, pp. 781-788, 1991.

[20] C. Farabet, C. Poulet, Y. LeCun, "An FPGA-Based Stream Processor for Embedded Real-Time Vision with Convolutional Networks, ICCV Workshops, pp. 878-885, September 2009.

[21] C. Farabet, Y. LeCun, K. Kavukcuoglu, E. Culurciello, B. Martini, P. Akselrod, S. Talay, "Large-Scale FPGA-Based Convolutional Networks", R. Bekkerman, M. Bilenko, J. Langford (Eds.), "Scaling up Machine Learning: Parallel and Distributed Approaches", pp. 399-419, Cambridge University Press, 2011.

[22] A. Canziani, A. Paszke, E. Culurciello, "An Analysis of Deep Neural Network Models for Practical Applications, arxiv: 1605.07678, 2017.

[23] Y.H. Chen, T. Krishna, J.S. Emer, 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.

[24] Y.H. Chen, J. Emer, V. Sze, "Eyeriss v2: A Flexible and High-Performance Accelerator for Emerging Deep Neural Networks". arXiv:1807.07928, 2018.

[25] G.E. Hinton, R.R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks", Science, 313(5786), pp. 504-507, 2006.

[26] G.E. Hinton, L. Deng, D. Yu, G.E. Dahl, A.R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The shared views of four research groups", IEEE Signal Processing Magazine, 29(6), pp. 82-97, 2012.

[27] R. Collobert, J. Weston, "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. ICML, pp. 160-167, 2008.

[28] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, "Natural Language Processing (almost) from Scratch", Journal of Machine Learning Research, pp. 2493-2537, August 2011.

[29] C. Farabet, C., Couprie, L. Najman, Y. LeCun, "Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers", ICML arXiv:1202.2160, 2012.

[30] P. Sermanet, K. Kavukcuoglu, S. Chintala, Y. LeCun, "Pedestrian Detection with Unsupervised Multi-Stage Feature Learning", CVPR pp. 3626-3633, 2013.

[31] A. Krizhevsky, I. Sutskever, G.E. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks", Advances in Neural Information Processing Systems, pp. 1097-1105, 2012.

[32] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, "Bag of Tricks for Efficient Text Classification", Proc 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers. Vol. 2, pp. 427-431, 2017.

[33] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, M. Paluri, "A Closer Look at Spatiotemporal Convolutions for Action Recognition:, Proc. Computer Vision and Pattern Recognition, pp. 6450-6459, 2018.

[34] M. Ott, S. Edunov, D. Grangier, M. Auli, "Scaling Neural Machine Translation". arXiv:1806.00187, 2018.

[35] J. Johnson, "Rethinking Floating Point for Deep Learning", ArXiv:1811.01721, 2018.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, "Attention is all you need" NIPS, pp. 5998-6008, 2017.

[37] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun, "Overfeat: Integrated Recognition, Localization and Detection Using Convolutional Networks". Proc. ICLR, arXiv:1312.6229, 2014.

[38] K. He, G. Gkioxari, P. Dollr, R. Girshick, R. (2017, "Mask R-CNN", Proc. ICCV, pp.2980-2988, October 2017.

[39] T.Y. Lin, P. Dollr, R.B. Girshick, K. He, B. Hariharan, S.J. Belongie, "Feature Pyramid Networks for Object Detection" CVPR, Vol. 1, No. 2, p. 4, 2017.

[40] T.Y. Lin, P. Goyal, R. Girshick, K. He, P. Dollr, "Focal Loss for Dense Object Detection", Proc. ICCV, arXiv:1708.02002, 2017.

[41] O. Ronneberger, P. Fischer, T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", International Conference on Medical Image Computing and Computer-Assisted Intervention", pp. 234-241, October 2015.

[42] P. Jaeger, S. Kohl, S. Bickelhaupt, F. Isensee, T.A.Kuder, H.-P. Schlemmer, K. Maier-Hein, "Retina U-Net: Embarrassingly Simple Exploitation of Segmentation Supervision for Medical Object Detectio", arXiv:1811.08661, 2018.

[43] M. Mathieu, C. Couprie, Y. LeCun, "Deep Multi-Scale Video Prediction Beyond Mean Square Error", ICLR, arXiv:1511.05440, 2016.

[44] C.Finn, I. Goodfellow, S. Levine, "Unsupervised Learning for Physical Interaction Through Video Prediction", Advances in Neural Information Processing Systems, pp. 64-72, 2016.

[45] M. Henaff, A. Canziani, Y. LeCun, "Model-Predictive Policy Learning with Uncertainty Regularization for Driving in Dense Traffic" To appear in 2019.

[46] P. Luc, C. Couprie, Y. LeCun, J. Verbeek, "Predicting Future Instance Segmentations by Forecasting Convolutional Features". ECCV, arXiv:1803.11496, 2018.

[47] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, K.Q. Weinberger, “Multi-Scale Dense Networks for Resource Efficient Image Classification”. ICLR, arXiv:1703.09844, 2018.
[48] J. Johnson, et al, “Inferring and Executing Programs for Visual Reasoning” ICCV, pp. 3008-3017, 2017.
[49] R. Hu, J. Andreas, M. Rohrbach, T. Darrell, K. Saenko, “Learning to Reason: End-to-End Module Networks for Visual Question Answering”, ICCV arxiv:1704.05526, 2017.
[50] M.M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst, “Geometric Deep Learning: Going Beyond Euclidean Data”, IEEE Signal Processing Magazine, 34(4), pp. 18-42, 2017.
[51] M. Nickel, D. Kiela, “Learning Continuous Hierarchies in the Lorentz Model of Hyperbolic Geometry”, arXiv:1806.03417, 2018.
[52] S. Sukhbaatar, J. Weston, R. Fergus, “End-to-End Memory Networks”, Advances in Neural Information Processing Systems, pp. 2440-2448, 2015.
[53] A. Miller, A.Fisch, J. Dodge, A.H. Karimi, A. Bordes, J. Weston, “Key-Value Memory Networks for Directly Reading Documents”, ArXiv:1606.03126, 2016.
[54] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, F. Huang, “A Tutorial on Energy-BasedLlearning.in Bakir et al (Eds), *Predicting Structured Data*, MIT Press, 2006.
[55] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. DeVito, W.S. Moses, S. Verdoolaege, A. Adams, A. Cohen, “Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions”. arXiv:1802.04730, 2018.
[56] B. Graham, M. Engelcke, L. van der Maaten, “3D Semantic Segmentation with Submanifold Sparse Convolutional Networks”, CVPR, pp 18-22, 2018.
[57] C. Farabet, R. Paz, J. Prez-Carrasco, C. Zamarreo, A. Linares-Barranco, Y. LeCun, E. Culurciello, T. Serrano-Gotarredona, B. Linares-Barranco, “Comparison Between Frame-Constrained Fix-Pixel-Value and Frame-Free Spiking-Dynamic-Pixel ConvNets for Visual Processing:, Frontiers in Neuroscience, 6, 32, 2012.
[58] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y Li, A. Bharambe, L. van der Maaten, “Exploring the Limits of Weakly Supervised Pretraining” ECCV, arXiv:1805.00932, 2018.
[59] I. Goodfellow, et al, “Generative Adversarial Nets”, NIPS, pp. 2672-2680, 2014.
[60] D.P. Kingma, M. Welling, “Auto-Encoding Variational Bayes”, ICLR. arXiv:1312.6114, 2014.
[61] T. Karras, T. Aila, S. Laine, J. Lehtinen, “Progressive Growing of Gans for Improved Quality, Stability, and Variation”, ICLR. arXiv:1710.10196, 2018.
[62] A. Brock, J. Donahue, K. Simonyan, “Large Scale Gan Training for High Fidelity Natural Image Synthesis”. arXiv:1809.11096, 2018.
[63] http://github.com/facebookresearch/maskrcnn-benchmark
[64] http://www.skiplang.com
[65] https://pytorch.org

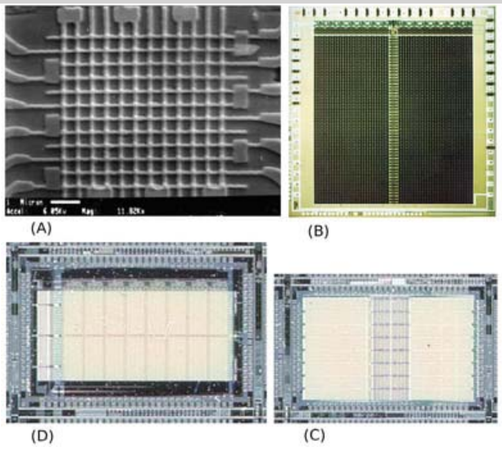

Figure 1.1.1: Early neural network chips from Bell Labs. (A) 1986: 12-resistor array, 6×6 microns [1]; (B) 1987: 54×54 analog array with programmable ternary weights [2]; (C) 1991: Net32K Convolver 256×128 programmable ternary weight array with FIFOs for convolutions [14]; (D) 1991: ANNA ConvNet chip 64×64 array with 6-bit weights and 3-bit activations [15].

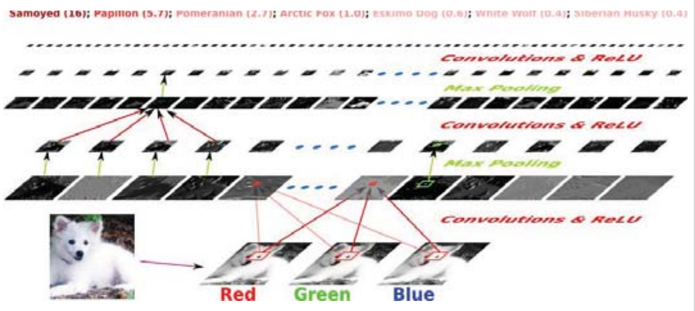
Samoyed (16); Papillon (5.7); Pomeranian (2.7); Arctic Fox (1.0); Eskimo Dog (0.6); White Wolf (0.4); Siberian Husky (0.4)
Convolutions & ReLU
Max Pooling
Convolutions & ReLU
Max Pooling
Convolutions & ReLU
Red
Green
Blue

Figure 1.1.2: An example of Convolutional Network architecture for image recognition. Not all layers are represented [37].

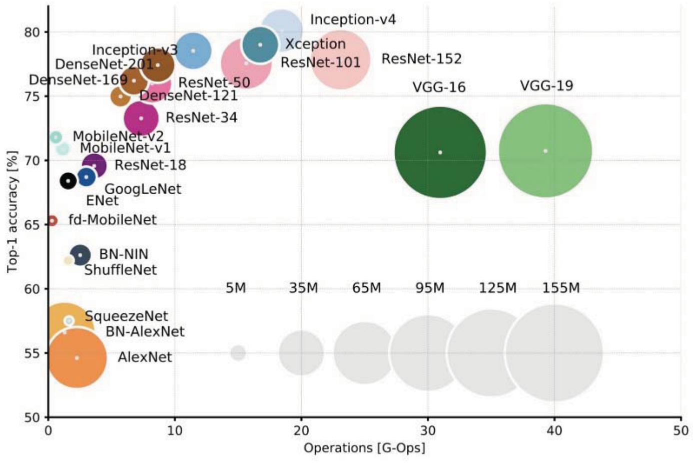

Figure 1.1.3: Top-1 accuracy on ImageNet versus number of operations for one pass of various ConvNet architectures. Circle size indicates the number of parameters [22].

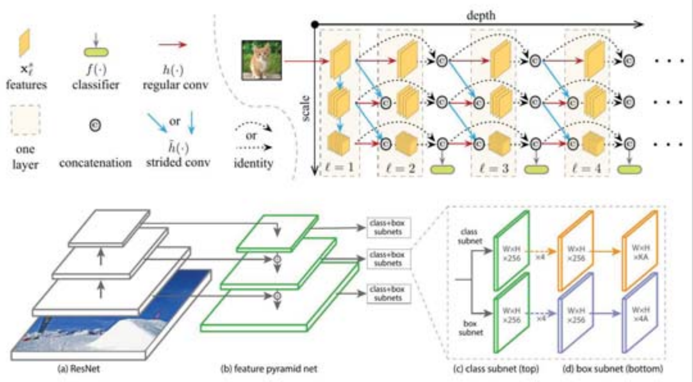

Figure 1.1.4: (Top) Multi-Scale DenseNet with conditional computation for accelerated results [47]. (Bottom) RetinaNet architecture for image semantic segmentation [40].

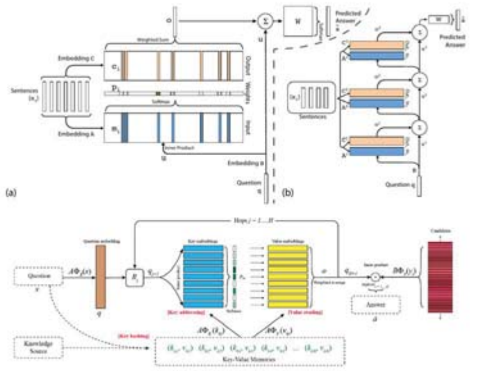

Figure 1.1.5: (Top) Memory Network architecture [52]; (Bottom) Key-Value Memory Network architecture for question answering [53]. Both architectures contain a central processing network connected with a “soft” associative memory circuit that stores facts. The memory module is a “soft” associative memory circuit in which the “address”vector is compared with each key vector through a dot product, producing scalar matching scores. The scores are normalized to sum to one. The output is a linear combination of the stored value vectors, weighted by the normalized scores.

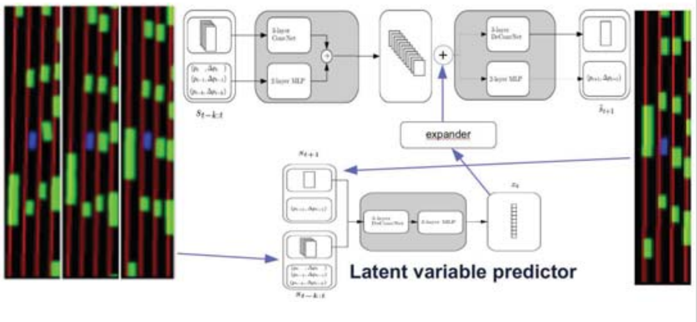

Figure 1.1.6: An example of self-supervised learning. A latent-variable model predicts how surrounding cars will move relative to the ego car (in the center). The model takes a few past frames and predicts the future relative positions of other cars, conditioned on a vector of latent variables. It is trained using data collected from traffic cameras overlooking roads. Different samplings of the latent variable produce different futures. This model can be used to plan or to train an artificial driver to minimize the probability of collision.