CellPress

and/or the extent of branching of N-linked glycans attached to plasma fibrinogen, possibly leading to a bleeding disorder.

Why are proteins glycosylated? Numerous roles have been assigned to protein glycosylation. At the level of the individual protein, glycosylation is directly related to protein folding, solubility, stability and activity, protection from proteases and subcellular targeting. For instance, in the ER, the chaperones calnexin and calreticulin follow the folding status of nascent polypeptides by assessing the composition of N-linked glycans decorating such proteins. Proper protein glycosylation is also important for the formation of protein complexes, for modulating protein–protein interactions and for the correct assembly of higher-order protein structures. At the cellular level, protein glycosylation is important for both transient and sustained cell–cell and cell–matrix recognition events and other interactions. This is exemplified by the impact of altered N-glycosylation on the affinity of antibodies for Fc receptors. Indeed, the enormous diversity that exists in terms of glycan composition and structure lends itself to the high specificity needed for such interactions.

At the same time, various pathogens can exploit surface-exposed glycans to attack target cells, either by binding to such moieties as part of a cellular entry strategy, or by relying on mimicry, whereby host-like glycans are presented with the aim of circumventing target cell defenses. In one striking example of how pathogens use protein glycosylation for attack, enteropathogenic *Escherichia coli* injects the host cell with an enzyme that modifies the glycosylation profile of host defense proteins, thereby preventing them from acting. Indeed, bacterial protein glycosylation is associated with virulence.

Archaea present yet another physiological role for protein glycosylation, with a modified glycosylation profile being seen in response to changes in the surroundings, such as changes in salinity or temperature, a strategy that may contribute to the ability of members of this domain to thrive in some of the most extreme environments on Earth.

Are there human diseases associated with improper protein glycosylation? Congenital disorders of glycosylation (CDGs) are a series of conditions in which mutations affect different components of the protein glycosylation pathway. For instance, CDGs caused by mutations in almost every N-glycosylation pathway gene have been described. While a complete loss of N-glycosylation is lethal, CDG patients can present an array of clinical symptoms, including retarded growth, incomplete brain development, muscle weakness, and abnormal endocrine and liver function. The most common of these maladies is CDG-Ia, in which the enzyme phosphomannomutase 2 is affected. Patients show developmental and motor deficits, hypotonia, dysmorphia, failure to thrive, liver dysfunction, coagulopathy, and abnormal endocrinology. Likewise, genetic maladies in which O-glycosylation is compromised are also known. In several types of muscular dystrophy, O-glycosylation of alpha-dystroglycan, responsible for binding to the extracellular matrix in skeletal muscle, is perturbed. Altered protein glycosylation is, moreover, considered to be a hallmark of cancer. In malignant cells, proteins can present reduced or enhanced glycan levels, incomplete, shortened or augmented glycans, and, less frequently, novel glycans. Indeed, the protein glycosylation profile of a cancer cell can change as the disease progresses, possibly encouraging tumor growth and invasiveness.

Where can I find out more?

Koomey, J.M., and Eichler, J. (2017). Sweet new roles for protein glycosylation in prokaryotes. Trends Microbiol. *25*, 662–672.

Ng, B.G., and Freeze, H.H. (2018). Perspectives on glycosylation and its congenital disorders. Trends Genet. *34*, 466–476.

Schäffer, C., and Messner, P. (2017). Emerging facets of prokaryotic glycosylation. FEMS Microbiol. Rev. *41*, 49–91.

Varki, A. (2017). Biological roles of glycans. Glycobiology *27*, 3–49.

Varki A., Cummings, R.D., Esko, J.D., Stanley, P., Hart, G.W., Aebi, M., Darvill, A.G., Kinoshita, T., Packer, N.H., Prestegard, J.H., *et al*. (2017). The Essentials of Glycobiology. (Cold Spring Harbor, NY: Cold Spring Harbor Press.)

Department of Life Sciences, Ben Gurion University of the Negev, Beersheva 84105, Israel.
E-mail: jeichler@bgu.ac.il

Primer

Neural network models and deep learning

Nikolaus Kriegeskorte[1,2,3,4,*] and Tal Golan[4,*]

Originally inspired by neurobiology, deep neural network models have become a powerful tool of machine learning and artificial intelligence. They can approximate functions and dynamics by learning from examples. Here we give a brief introduction to neural network models and deep learning for biologists. We introduce feedforward and recurrent networks and explain the expressive power of this modeling framework and the backpropagation algorithm for setting the parameters. Finally, we consider how deep neural network models might help us understand brain computation.

Neural network models of brain function

Brain function can be modeled at many different levels of abstraction. At one extreme, neuroscientists model single neurons and their dynamics in great biological detail. At the other extreme, cognitive scientists model brain information processing with algorithms that make no reference to biological components. In between these extremes lies a model class that has come to be called *artificial neural network*.

A biological neuron receives multiple signals through the synapses contacting its dendrites and sends a single stream of action potentials out through its axon. The conversion of a complex pattern of inputs into a simple decision (to spike or not to spike) suggested to early theorists that each neuron performs an elementary cognitive function: it reduces complexity by categorizing its input patterns. Inspired by this intuition, artificial neural network models are composed of *units* that combine multiple inputs and produce a single output.

The most common type of unit computes a weighted sum of the inputs and transforms the result nonlinearly.

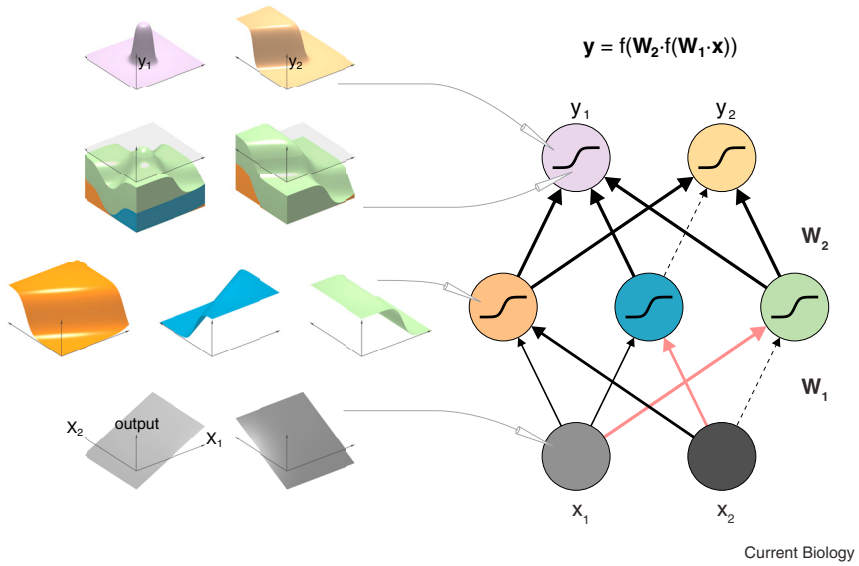


Figure 1. Function approximation by a feedforward neural network.
A feedforward neural network with two input units (bottom), three hidden units (middle), and two output units (top). The input patterns form a two-dimensional space. The hidden and output units here use a sigmoid (logistic) activation function. Surface plots on the left show the activation of each unit as a function of the input pattern (horizontal plane spanned by inputs x_1 and x_2). For the output units, the preactivations are shown below the output activations. For each unit, the weights (arrow thickness) and signs (black, positive; red, negative) of the incoming connections control the orientation and slope of the activation function. The output units combine the nonlinear ramps computed by the hidden units. Given enough hidden units, a network of this type can approximate any continuous function to arbitrary precision.

The weighted sum can be interpreted as comparing the pattern of inputs to a reference pattern of weights, with the weights corresponding to the strengths of the incoming connections. The weighted sum is called the *preactivation*. The strength of the preactivation reflects the overall strength of the inputs and, more importantly, the match between the input pattern and the weight pattern. For a given input strength (measured as the sum of squared intensities), the preactivation will be maximal if the input pattern exactly matches the weight pattern (up to a scaling factor).

The preactivation forms the input to the unit's nonlinear activation function. The activation function can be a threshold function (0 for negative, 1 for positive preactivations), indicating whether the match is sufficiently close for the unit to respond. More typically, the activation function is a monotonically increasing function, such as the logistic function (Figure 1) or a rectifying nonlinearity, which outputs the preactivation if it is positive and zero otherwise. These latter activation functions have non-zero derivatives (at least over the positive range of preactivations). As we will see below, non-zero derivatives make it easier to optimize the weights of a network.

The weights can be positive or negative. Inhibition, thus, need not be relayed through a separate set of inhibitory units, and neural network models typically do not respect Dale's law (which states that a neuron performs the same chemical action at all of its synaptic connections to other neurons, regardless of the identity of the target cell). In addition to the weights of the incoming connections, each unit has a bias parameter: the bias is added to the preactivation, enabling the unit to shift its nonlinear activation function horizontally, for example moving the threshold to the left or right. The bias can be understood as a weight for an imaginary additional input that is constantly 1.

Neural networks are universal approximators

Units can be assembled into networks in many different configurations. A single unit can serve as a linear discriminant of its input patterns. A set of units connected to the same set of inputs can detect multiple classes, with each unit implementing a different linear discriminant. For a network to discriminate classes that are not linearly separable in the input signals, we need an intermediate layer between input and output units, called a *hidden layer* (Figure 1).

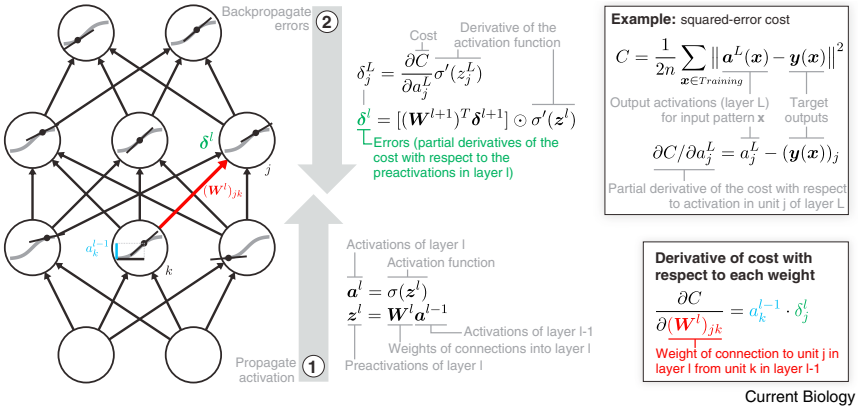
If the units were linear — outputting the weighted sum directly, without passing it through a nonlinear activation function — then the output units reading out the hidden units would compute weighted sums of weighted sums and would, thus, themselves be limited to weighted sums of the inputs. With nonlinear activation functions, a hidden layer makes the network more expressive, enabling it to approximate nonlinear functions of the input, as illustrated in Figure 1.

A feedforward network with a single hidden layer (Figure 1) is a flexible approximator of functions that link the inputs to the desired outputs. Typically, each hidden unit computes a nonlinear ramp, for example sigmoid or rectified linear, over the input space. The ramp rises in the direction in input space that is defined by the vector of incoming weights. By adjusting the weights, we can rotate the ramp in the desired direction. By scaling the weights vector, we can squeeze or stretch the ramp to make it rise more or less steeply. By adjusting the bias, we can shift the ramp forward or backward. Each hidden unit can be independently adjusted in this way.

One level up, in the output layer, we can linearly combine the outputs of the hidden units. As shown in Figure 1, a weighted sum of several nonlinear ramps produces a qualitatively different continuous function over the input space. This is how a hidden layer of linear–nonlinear units enables the approximation of functions very different in shape from the nonlinear activation function that provides the building blocks.

It turns out that we can approximate any continuous function to any desired level of precision by allowing a sufficient number of units in a single hidden layer. To gain an intuition of why this is possible, consider the left output unit (y_1) of the network in Figure 1.

By combining ramps overlapping in a single region of the input space, this unit effectively selects a single compact patch. We could tile the entire input space with sets of hidden units that select different patches in this way. In the output layer, we could then map each patch to any desired output value. As we move from one input region to another, the network would smoothly transition between the different output values. The precision of such an approximation can always be increased by using more hidden units to tile the input space more finely.



Current Biology

Figure 2. The backpropagation algorithm.
Backpropagation is an efficient algorithm for computing how small adjustments to the connection weights affect the cost function that the network is meant to minimize. A feedforward network with two hidden layers is shown as an example. First, the activations are propagated in the feed-forward direction (upward). The activation function (gray sigmoid) is shown in each unit (circle). In the context of a particular input pattern (not shown), the network is in a particular activation state, indicated by the black dots in the units (horizontal axis: preactivation, vertical axis: activation). Second, the derivatives of the cost function (squared-error cost shown on the right) are propagated in reverse (downward). In the context of the present input pattern, the network can be approximated as a linear network (black lines indicating the slope of the activation function). The chain rule defines how the cost (the error) is affected by small changes to the activations, preactivations, and weights. The goal is to compute the partial derivative of the cost with respect to each weight (bottom right). Each weight is then adjusted in proportion to how much its adjustment reduces the cost. The notation roughly follows Nielsen (2015), but we use bold symbols for vectors and matrices.

Deep networks can efficiently capture complex functions

A feedforward neural network is called '*deep*' when it has more than one hidden layer. The term is also used in a graded sense, in which the depth denotes the number of layers. We have seen above that even *shallow* neural networks, with a single hidden layer, are universal function approximators. What, then, is the advantage of *deep* neural networks?

Deep neural networks can re-use the features computed in a given hidden layer in higher hidden layers. This enables a deep neural network to exploit compositional structure in a function, and to approximate many natural functions with fewer weights and units. Whereas a shallow neural network must piece together the function it approximates, like a lookup table (although the pieces overlap and sum), a deep neural network can benefit from its hierarchical structure. A deeper architecture can increase the precision with which a function can be approximated on a fixed budget of parameters and can improve the generalization after learning to new examples.

Deep learning refers to the automatic determination of parameters deep in a network on the basis of experience (data). Neural networks with multiple hidden layers are an old idea and were a popular topic in engineering and cognitive science in the 1980s. Although the advantages of deep architectures were understood in theory, the method did not realize its potential in practice, mainly because of insufficient computing power and data for learning. Shallow machine learning techniques, such as support vector machines, worked better in practice and also lent themselves to more rigorous mathematical analysis. The recent success of deep learning has been driven by a rise in computing power — in particular the advent of graphics processing units, GPUs, specialized hardware for fast matrix–matrix multiplication — and web-scale data sets to learn from. In addition, improved techniques for pretraining, initialization, regularization, and normalization, along with the introduction of rectified linear units, have all helped to boost performance. Recent work has explored a wide variety of feedforward and recurrent network architectures, improving the state-of-the-art in several domains of artificial intelligence and establishing deep learning as a central strand of machine learning in the last few years.

The function that deep neural networks are trained to approximate is often a mapping from input patterns to output patterns, for example classifying natural images according to categories, translating sentences from English to French, or predicting tomorrow's weather from today's measurements. When the cost minimized by training is a measure of the mismatch between the network's outputs and desired outputs (that is, the 'error'), for a training set of example cases, the training is called supervised. When the cost minimized by training does not involve prespecified desired outputs for a set of example inputs, the training is called unsupervised.

Two examples of unsupervised learning are autoencoders and generative adversarial networks. Autoencoder networks learn to transform input patterns into a compressed latent representation by exploiting inherent statistical structure. Generative adversarial networks operate in the opposite direction, transforming random patterns in a latent representation into novel, synthetic examples of a category, such as fake images of bedrooms. The generator network is trained concurrently with a discriminator network that learns to pick out the generator's fakes among natural examples of the category. The two adversarial networks boost each other's performance by posing increasingly difficult challenges of counterfeiting and detection to each

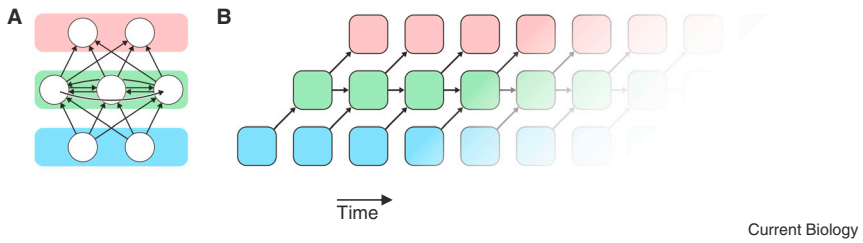


Figure 3. Recurrent neural networks.
(A) A recurrent neural network model with two input units (in blue box), three hidden units (green box), and two output units (pink box). The hidden units here are fully recurrently connected: each sends its output to both other units. The arrows represent scalar weights between particular units. (B) Equivalent feedforward network. Any recurrent neural network can be unfolded along time as a feedforward network. To this end, the units of the recurrent neural network (blue, green, pink sets) are replicated for each time step. The arrows here represent weight matrices between sets of units in the colored boxes. For the equivalence to hold, the feedforward network has to have a depth matching the number of time steps that the recurrent network is meant to run for. Unfolding leads to a representation that is less concise, but easier to understand and often useful in software implementations of recurrent neural networks. Training of the recurrent model by backpropagation through time is equivalent to training of the unfolded model by backpropagation.

other. Deep neural networks can also be trained by reinforcement (deep reinforcement learning), which has led to impressive performance at playing games and robotic control.

Deep learning by backpropagation

Say we want to train a deep neural network model with supervision. How can the connection weights deep in the network be automatically learned? The weights are randomly initialized and then adjusted in many small steps to bring the network closer to the desired behavior. A simple approach would be to consider random perturbations of the weights and to apply them when they improve the behavior. This evolutionary approach is intuitive and has recently shown promise, but it is not usually the most efficient solution. There may be millions of weights, spanning a search space of equal dimension. It takes too long in practice to find directions to move in such a space that improve performance. We could wiggle each weight separately, and determine if behavior improves. Although this would enable us to make progress, adjusting each weight would require running the entire network many times to assess its behavior. Again, progress with this approach is too slow for many practical applications.

In order to enable more efficient learning, neural network models are composed of *differentiable* operations. How a small change to a particular weight affects performance can then be computed as the partial derivative of the error with respect to the weight. For different weights in the same model, the algebraic expressions corresponding to their partial derivatives share many terms, enabling us to efficiently compute the partial derivatives for all weights.

For each input, we first propagate the activation forward through the network, computing the activation states of all the units, including the outputs. We then compare the network's outputs with the desired outputs and compute the cost function to be minimized (for example, the sum of squared errors across output units). For each unit, we then compute how much the cost would drop if the activation changed slightly. This is the *sensitivity* of the cost to a change of activation of each output unit. Mathematically, it is the partial *derivative* of the cost with respect to each activation. We then proceed backwards through the network propagating the cost derivatives (sensitivities) from the activations to the preactivations and through the weights to the activations of the layer below. The sensitivity of the cost to each of these variables depends on the sensitivities of the cost to the variables downstream in the network. *Backpropagating* the derivatives through the network by applying the chain rule provides an efficient algorithm for computing all the partial derivatives.

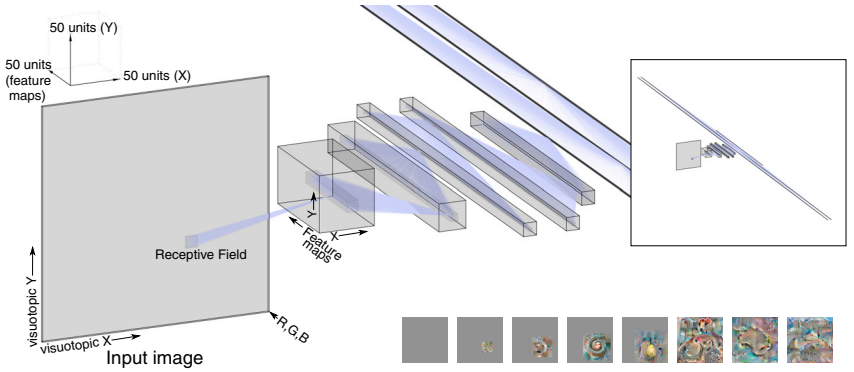
The critical step is computing the partial derivative of the cost with respect to each weight. Consider the weight of a particular connection (red arrow in Figure 2). The connection links a source unit in one layer to a target unit in the next layer. The influence of the weight on the cost for a given input pattern depends on how active the source unit is. If the source unit is off for the present input pattern, then the connection has no signal to transmit and its weight is irrelevant to the output the network produces for the current input. The activation of the source unit is *multiplied* with the weight to determine its contribution to the preactivation of the target unit, so the source activation is one factor determining the influence of the weight on the cost. The other factor is the sensitivity of the cost to the preactivation of the target unit. If the preactivation of the target unit had no influence on the cost, then the weight would have no influence either. The derivative of the cost with respect to the weight is the product of its source unit's activation and its target unit's influence on the cost.

We adjust each weight in the direction that reduces the cost (the error) and by an amount proportional to the derivative of the cost with respect to the weight. This process is called *gradient descent*, because it amounts to moving in the direction in weight space in which the cost declines most steeply. To help our intuition, let us consider two approaches we might take. First, consider the approach of taking a step to reduce the cost *for each individual training example*. Gradient descent will make *minimal and selective adjustments* to reduce the error, which makes sense as we do not want learning from the current example to interfere with what we've learned from other examples. However, our goal is to reduce the *overall error*, which is defined as the sum of the errors across all examples. So second, consider the approach of summing up the error surfaces (or, equivalently, the gradients) across all examples before taking a step. We can still only take a small step, because the error surface is nonlinear and so the gradient will change as we move away from the point about which we linearized the network.

In practice, the best solution is to use small *batches of training examples* to estimate the gradient before taking a

step. Compared to the single-example approach, this gives us a more stable sense of direction. Compared to the full-training-set approach, it greatly reduces the computations required to take a step. Although the full-training-set approach gives exact gradients for the training-set error, it still does not enable us to take large steps, because of the nonlinearity of the error function. Using batches is a good compromise between stability of the gradient estimate and computational cost. Because the gradient estimate depends on the random sample of examples in the current batch, the method is called *stochastic* gradient descent (SGD). Beyond the motivation just given, the stochasticity is thought to contribute also to finding solutions that generalize well beyond the training set.

The cost is not a convex function of the weights, so we might be concerned about getting stuck in local minima. However, the high dimensionality of weight space turns out to be a blessing (not a curse) for gradient descent: there are many directions to escape in, making it unlikely that we will ever find ourselves trapped, with the error surface rising in all directions. In practice, it is saddle points (where the gradient vanishes) that pose a greater challenge than local minima. Moreover, the cost function typically has many symmetries, with any given set of weights having many computationally equivalent twins (that is, the model computes the same overall function for different parameter settings). As a result, although our solution may be one local minimum among many, it may not be a poor local minimum: It may be one of many similarly good solutions.



Current Biology

Figure 4. Deep convolutional feedforward neural networks.
The general structure of Alexnet, a convolutional deep neural network architecture which had a critical role in bringing deep neural networks into the spotlight. Unlike the visualization in the original report on this model, here the tensors' dimensions are drawn to scale, so it is easier to appreciate how the convolutional deep neural network gradually transforms the input image from a spatial to semantic representation. For sake of simplicity, we did not visualize the pooling operations, as well as the splitting of some of these layers between two GPUs. The leftmost box is the input image (a tensor of the dimensions 227×227×3, where 227 is the length of the square input-image edges and three is the number of color components). It is transformed by convolution into the first layer (second box from the left), a tensor with smaller spatial dimensions (55x55) but a larger number of feature maps (96). Each feature map in this tensor is produced by a convolution of the original image with a particular 11×11×3 filter. Therefore, the preactivation of each unit in this layer is a linear combination of one rectangular receptive field in the image. The boundaries of such a receptive field are visualized as a small box within the image tensor. In the next, second layer, the representation is even more spatially smaller (27×27) but richer with respect to the number of feature maps (256). Note that from here and onwards, each feature is not a linear combination of pixels but a linear combination of the previous layer's features. The sixth layer (see the small overview inset at the top-right) combines all feature maps and locations of the fifth layer to yield 4096 different scalar units, each with its own unrestricted input weights vector. The final eighth layer has 1000 units, one for each output class. The eight images on the bottom were produced by gradually modifying random noise images so excite particular units in each of the eight layers. The rightmost image was optimized to activate the output neuron related to the class 'Mosque'. Importantly, these are only local solutions to the activation-maximization problem. Alternative activation-maximizing images may be produced by using different starting conditions or optimization heuristics.

Recurrent neural networks are universal approximators of dynamical systems

So far we have considered feedforward networks, whose directed connections do not form cycles. Units can also be configured in recurrent neural networks (RNNs), where activity is propagated in cycles, as is the case in brains. This enables a network to recycle its limited computational resources over time and perform a deeper sequence of nonlinear transformations. As a result, RNNs can perform more complex computations than would be possible with a single feedforward sweep through the same number of units and connections.

For a given state space, a suitable RNN can map each state to any desired successor state. RNNs, therefore, are universal approximators of dynamical systems. They provide a universal language for modeling dynamics, and one whose components could plausibly be implemented with biological neurons.

Much like feedforward neural networks, RNNs can be trained by backpropagation. However, backpropagation must proceed through the cycles in reverse. This process is called backpropagation through time. An intuitive way to understand an RNN and backpropagation through time is to 'unfold' the RNN into an equivalent feedforward network (Figure 3). Each layer of the feedforward network represents a timestep of the RNN. The units and weights of the RNN are replicated for each layer of the feedforward network. The feedforward network, thus, shares the same set of weights across its layers (the weights of the recurrent network).

For tasks that operate on independent observations (for example, classifying still images), the recycling of weights can enable an RNN to perform better than a feedforward network with the same number of parameters. However, RNNs really shine in tasks that operate on streams of dependent observations. Because RNNs can maintain an internal state (memory) over time and produce dynamics, they lend themselves to tasks that require temporal patterns to be recognized or generated. These include the perception speech and video, cognitive tasks that require maintaining

representations of hidden states of the agent (such as goals) or the environment (such as currently hidden objects), linguistic tasks like the translation of text from one language into another, and control tasks at the level of planning and selecting actions, as well as at the level of motor control during execution of an action under feedback from the senses.

Deep neural networks provide abstract process models of biological neural networks

Cognitive models capture aspects of brain information processing, but do not speak to its biological implementation. Detailed biological models can capture the dynamics of action potentials and the spatiotemporal dynamics of signal propagation in dendrites and axons. However, they have only had limited success in explaining how these processes contribute to cognition. Deep neural network models, as discussed here, strike a balance, explaining feats of perception, cognition, and motor control in terms of networks of units that are highly abstracted, but could plausibly be implemented with biological neurons.

For engineers, artificial deep neural networks are a powerful tool of machine learning. For neuroscientists, these models offer a way of specifying mechanistic hypotheses on how cognitive functions may be carried out by brains. Deep neural networks provide a powerful language for expressing information-processing functions. In certain domains, they already meet or surpass human-level performance (for example, visual object recognition and board games) while relying exclusively on operations that are biologically plausible.

Neural network models in engineering have taken inspiration from brains, far beyond the general notion that computations involve a network of units, each of which nonlinearly combines multiple inputs to compute a single output. For example, convolutional neural networks, the dominant technology in computer vision, use a deep hierarchy of retinotopic layers whose units have restricted receptive fields. The networks are convolutional in that weight templates are automatically shared across image locations (rendering the computation of a feature map's preactivations equivalent to a convolution of the input with the weight template). Although the convolutional aspect may not capture an innate characteristic of the primate visual system, it does represent an idealization of the final product of development and learning in primates, where qualitatively similar features are extracted all over retinotopic maps at early stages of processing. Across layers, these networks transform a visuospatial representation of the image into a semantic representation of its contents, successively reducing the spatial detail of the maps and increasing the number of semantic dimensions (Figure 4).

The fact that a neural network model was inspired by some abstract features of biology and that it matches overall human or animal performance at a task does not make it a good model of how the human or animal brain performs the task. However, we can compare neural network models to brains in terms of detailed patterns of behavior, such as errors and reaction times for particular stimuli. Moreover, we can compare the internal representations in neural networks to those in brains.

In the 'white-box' approach, we evaluate a model by looking at its internal representations. Neural network models form the basis for predicting representations in different brain regions for a particular set of stimuli. One approach is called *encoding models*. In encoding models, the brain activity pattern in some functional region is predicted using a linear transformation of the representation in some layer of the model. In another approach, called *representational similarity analysis*, each representation in brain and model is characterized by a representational dissimilarity matrix. Models are evaluated according to their ability to explain the representational dissimilarities across pairs of stimuli. A third approach is *pattern component modeling*, where representations are characterized by the second moment of the activity profiles.

Recent results from the domain of visual object recognition indicate that deep convolutional neural networks are the best available model of how the primate brain achieves rapid recognition at a glance, although they do not explain all of the explainable variance in neuronal responses.

In the 'black-box' approach, we evaluate a model on the basis of its behavior. We can reject models for failing to explain detailed patterns of behavior. This has already helped reveal some limitations of convolutional neural networks, which appear to behave differently from humans under noisy conditions and to show different patterns of failures across exemplars.

Deep neural networks bridge the gap between neurobiology and cognitive function, providing an exciting framework for modeling brain information processing. Theories of how the brain computes can now be subjected to rigorous tests by simulation. Our theories, and the models that implement them, will evolve as we learn to explain the rich measurements of brain activity and behavior provided by modern technologies in animals and humans.

FURTHER READING

Dayan, P., and Abbott, L.F. (2001). Chapter 7.4, Recurrent neural networks. In Theoretical Neuroscience (Cambridge, MA: MIT Press).

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning (MIT Press).

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

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. Nature *521*, 436–444.

Kietzmann, T., McClure, P., and Kriegeskorte, N. (2019). Deep neural networks in computational neuroscience. In Oxford Research Encyclopedia of Neuroscience. https://doi.org/10.1093/acrefore/9780190264086.013.46

Kriegeskorte, N. (2015). Deep neural networks: a new framework for modeling biological vision and brain information processing. Annu. Rev. Vis. Sci. *1*, 417–446.

Nielsen, M.A. (2015). Neural Networks and Deep Learning (Determination Press).

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural Netw. *61*, 85–117.

Storrs, K.R. and Kriegeskorte, N. (2019). Deep learning for cognitive neuroscience. In The Cognitive Neurosciences (6th Edition), M. Gazzaniga, ed. (Boston: MIT Press).

Yamins, D.L.K., and DiCarlo, J.J. (2016). Using goal-driven deep learning models to understand sensory cortex. Nat. Neurosci. *19*, 356–365.

[1]Department of Psychology, [2]Department of Neuroscience, [3]Department of Electrical Engineering, [4]Zuckerman Mind Brain Behavior Institute, Columbia University, New York, NY 10027, USA.
*E-mail: n.kriegeskorte@columbia.edu (N.K.), tal.golan@columbia.edu (T.G.)

ニューラルネットワークモデルとディープラーニング
Neural network models and deep learning

Nikolaus Kriegeskorte　　Tal Golan

Current Biology 29, R225–R240, April 1, 2019

神経生物学から着想を得たディープニューラルネットワークモデルは、機械学習や人工知能の強力なツールとなっている。ニューラルネットワークモデルは例から学習することで関数やダイナミクスを近似することができる。ここでは生物学者のためにニューラルネットワークモデルと深層学習について簡単に紹介する。フィードフォワードネットワークとリカレントネットワークを紹介しこのモデリングフレームワークの表現力とパラメータを設定するためのバックプロパゲーションアルゴリズムについて説明する。最後にディープニューラルネットワークモデルが脳の計算を理解するのにどのように役立つかを考察する。

1　脳機能のニューラルネットワークモデル

脳の機能はさまざまな抽象度でモデル化することができる。神経科学者は単一のニューロンとそのダイナミクスを生物学的に詳細にモデル化する。一方認知科学者は生物学的な要素を全く考慮しないアルゴリズムによって脳の情報処理をモデル化する。この両極端の間に「人工ニューラルネットワーク」と呼ばれるモデルクラスがある。

生物のニューロンは樹状突起に接触しているシナプスから複数の信号を受け取り軸索を介して単一の活動電位の流れを送り出す。複雑な入力パターンが単純な判断（スパイクするかしないか）に変換されることから初期の理論家はニューロンが初歩的な認知機能を果たしていると考えた。この直感に基づいてニューラルネットワークモデルは複数の入力を組み合わせて単一の出力を生成するユニットで構成されている。

最も一般的なタイプのユニットは入力の加重和を計算しその結果を非線形に変換する。加重和とは入力のパターンを入力される接続の強さに応じた重みを持つ基準パターンと比較することだと解釈できる。この加重和は前活性化値と呼ばれる。前活性化値の強さは入力の全体的な強さとさらに重要なことには入力パターンと重みパターンの一致を反映している。与えられた入力強度（強度の二乗和として測定される）に対して入力パターンが荷重パターンと正確に一致する（拡大縮小係数まで）場合、前活性化は最大となる。

前活性化はユニットの非線形活性化関数への入力を形成する。活性化関数はユニットが反応するのに十分なほどマッチが近づいているかどうかを示すしきい値関数 (負の事前活性化は 0, 正の事前活性化は 1) とすることができます。より一般的には活性化関数はロジスティック関数 (図1) のような単調増加関数, または, 正の場合は前活性化を出力し, そうでない場合はゼロを出力する整流線形関数である。この後者の活性化関数は (少なくとも事前活性化の正の範囲では) 非ゼロの導関数を持つ。後述するように非ゼロの導関数はネットワークの重みを最適化することを容易にする。

重みは正でも負でも構わない。またニューラルネットワークモデルではデイルの法則 (ニューロンは標的細胞の種類にかかわらず他のニューロンとのシナプス結合のすべてにおいて同じ化学作用を行う) を考慮する必要はない。各ユニットは入力される接続の重みに加えてバイアスパラメータを持つ。バイアスは前活性化に追加されユニットが非線形活性化関数を水平方向にシフトすることを可能にする。例えばしきい値を左右に移動させることができる。バイアスは常に 1 である架空の追加入力に対する重みとして理解することができる。

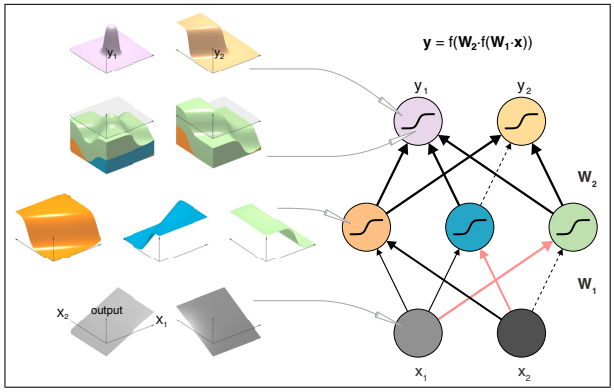


図1　フィードフォワード・ニューラル・ネットワークによる関数の近似。 2 つの入力ユニット(下), 3 つの隠れユニット(中), 2 つの出力ユニット(上) を持つフィードフォワードニューラルネットワーク。入力パターンは 2 次元の空間を形成している。隠れユニットと出力ユニットはシグモイド (ロジスティック) 活性化関数を使用している。左の曲面図は各ユニットの活性化を入力パターン（入力 x 1 と x2 で構成される水平面) の関数として表している。出力ユニットでは出力活性化の下に前活性化が示されている。各ユニットでは入力される結合重み (矢印の太さ) と符号 (黒は正, 赤は負) が, 活性化関数の向きと傾きを制御する。出力ユニットは隠れユニットで計算された非線形の傾斜を組み合わせる。この種のネットワークは十分な数の隠れユニットがあれば，任意の連続関数を任意の精度で近似することができる。

2　ニューラルネットワークは万能近似器

ユニットは様々な構成でネットワークに組み入れることができる。 1 つのユニットはその入力パターンの線形判別器として機能する。同じ入力セットに接続されたユニットのセットは各ユニットが異なる線形判別を行うことで複数のクラスを検出することができる。入力信号が線形に分離できないクラスを識別するためには入力ユニットと出力ユニットの間に隠れ層と呼ばれる中間層が必要になる (図1)。

もしユニットが線形で非線形活性化関数を通さずに加重和を直接出力するならば隠れユニットから読み出された出力ユニットは加重和の加重和を計算することになる。それ自体が入力の加重和に限定されることになる。非線形活性化関数の場合隠れ層を設けることでネットワークの表現力が高まり図 1 のように入力の非線形関数を近似することができる。

単一の隠れ層を持つフィードフォワードネットワーク (図1) は入力を目的の出力に結びつける関数の可読性の高い近似器である。一般的に各隠れユニットは入力空間上の非線形ランプ (シグモイドや整流線形ユニットなど) を計算する。このランプは入力される重みのベクトルによって定義される入力空間の方向に向かって上昇する。重みを調整することでランプを所望の方向に回転させることができまる。重みのベクトルをスケーリングすることでランプを絞ったり伸ばしたりしてより急峻に上昇させることができる。バイアスを調整することでランプを前方または後方に移動させることができる。このようにして各隠れユニットを独立して調整することができる。

1 つ上のレベルの出力層では隠れユニットの出力を線形に組み合わせることができる。図 1 に示すように複数の非線形ランプの加重和は入力空間上で質的に異なる連続関数を生成する。このように線形-非線形ユニットの隠れた層はビルディングブロックを提供する非線形活性化関数とは非常に異なる形状の関数の近似を可能にする。

隠れ層に十分な数のユニットを配置することで任意の連続関数を任意の精度で近似できることがわかった。なぜこのようなことが可能なのかを直感的に理解するために図 1 のネットワークの左の出力ユニット (y1) を考える。

このユニットは入力空間の 1 つの領域に重なるランプを組み合わせることで 1 つのコンパクトなパッチを効果的に選択する。このようにして異なるパッチを選択する隠れユニットのセットで入力空間全体をタイル状にすることができる。出力層では各パッチを任意の出力値にマッピングすることができます。ある入力領域から別の入力領域に移

動するとネットワークは異なる出力値の間をスムーズに移行する。このような近似の精度はより多くの隠れユニットを使用して入力空間をより忠実にタイル化することで常に向上させることができる。

3　ディープネットワークは効率良く複雑な関数を掌握

フィードフォワードニューラルネットワークは複数の隠れ層を持つ場合に「深い」と呼ばれる。この用語は深さが層数を表すような段階的な意味でも使われる。上述したように隠れ層が 1 つの浅いニューラルネットワークであっても普遍的な関数近似器であることがわかっている。では深いニューラルネットワークの利点は何であろうか？

ディープニューラルネットワークはある隠れ層で計算された特徴量をより上位の隠れ層で再利用することができる。これによりディープニューラルネットワークは関数の構成構造を利用しより少ない重みとユニットで多くの自然な関数を近似することができる。浅いニューラルネットワークではルックアップテーブルのように近似する関数を断片的に組み合わせる必要がある (ただし断片は重なり合って合計される) 深いニューラルネットワークでは階層的な構造から恩恵を受けることができる。深いアーキテクチャは限られたパラメータ予算で関数を近似する精度を高め新しい例に対する学習後の一般化を向上させることができる。

ディープラーニングとは経験 (データ) に基づいてネットワークの奥深くにあるパラメータを自動的に決定することを指す。複数の隠れ層を持つニューラルネットワークは古くからある考え方で 1980 年代には工学や認知科学の分野で盛んに研究された。理論的にはディープアーキテクチャの利点は理解されていた。だが実際には学習のための計算能力やデータが不足していたことが主な理由で、この手法はその可能性を実現できなかった。サポートベクターマシンのような浅い機械学習手法は実際にはより効果的でありより厳密な数学的分析に適していた。近年のディープラーニングの成功は計算処理能力の向上特に GPU (グラフィックス・プロセッシング・ユニット) の登場と学習対象となるウェブのデータセットの登場による。さらに事前訓練, 初期化, 正則化, 正規化の技術が向上したことや整流線形ユニットが導入されたこともパフォーマンスの向上に貢献している。最近の研究ではフィードフォワードネットワークやリカレントネットワークのアーキテクチャが幅広く検討されている。人工知能のいくつかの分野で最先端の技術を向上させここ数年で深層学習が機械学習の中心的な分野として確立された。

ディープニューラルネットワークが近似するように学習される機能は多くの場合入力パターンから出力パターンへのマッピングである。例えば自然画像をカテゴリーに分類する, 英語からフランス語に文章を翻訳する, 今日の測定値から明日の天気を予測する, などが挙げられる。学習によって最小化される損失関数がネットワークの出力と所望の出力との間の不一致 (すなわち「誤差」) の尺度である場合例題の学習セットに対してその学習は「教師あり」と呼ばれる。訓練によって最小化される損失関数が入力データセットに対する望ましい出力を事前に指定する必要がない場合, その訓練は「教師なし」と呼ばれる。

教師なし学習の 2 つの例として自己符号化器と敵対生成ネットワークがある。自己符号化器は固有の統計的構造を利用して入力パターンを圧縮された潜在的な表現に変換することを学習する。敵対生成ネットワークは逆に潜在表現のランダムなパターンをベッドルームの偽画像などあるカテゴリーの新しい合成例に変換する。生成器ネットワークはカテゴリの自然な例の中から生成器の偽物を選び出すことを学習する識別器ネットワークと同時に学習される。 2 つの敵対的なネットワークは偽造と検出という難しい課題を互いに課すことで互いのパフォーマンスを向上させる。ディープニューラルネットワークは強化学習にも適用でき (深層強化学習), ゲームやロボット制御などで性能を発揮している。

4　誤差逆伝播によるディープラーニング

ディープニューラルネットワークのモデルを監視付きで学習したいとする。ネットワークの深くにある結合係数を自動的に学習するにはどうすればよいのであろうか。結合係数はランダムに初期化され, その後ネットワークを望ま

しい動作に近づけるため, 多くの小さなステップで調整される。単純なアプローチとしては, 重みのランダムな摂動を考慮し, それによって動作が改善されたときに適用するというものがある。この進化論的アプローチは直感的で, 最近では有望視されている。だが, 通常は最も効率的な解決策ではない。何百万もの結合係数が, 同じ次元の探索空間に存在する可能性がある。このような空間の中で, パフォーマンスを向上させるための移動方向を見つけるには, 実際には時間がかかる。それぞれの結合係数を個別に動かして, 動作が改善されるかどうかを判断することもできる。しかし, それぞれの結合係数を調整するには, ネットワーク全体を何度も走らせて挙動を評価する必要がある。またこの方法では多くの実用的なアプリケーションでは遅すぎる。

ニューラルネットワークモデルはより効率的な学習を行うために微分可能な演算で構成される。ある結合係数を少し変えただけで性能にどのような影響が出るかは, その重みに対する誤差の偏微分として計算することができる。同じモデルの異なる結合係数では, 偏微分に対応する数式は多くの項を共有しており, すべての重みの偏微分を効率的に計算することができる。

各入力に対して, まず活性化をネットワークに伝搬させ, 出力を含むすべてのユニットの活性化状態を計算する。次に, ネットワークの出力を目的の出力と比較し, 最小化すべきコスト関数 (たとえば, 出力ユニット間の二乗誤差の合計) を計算する。そして, 各ユニットについて, 活性化を少し変えただけでコストがどれだけ下がるかを計算する。これが各出力ユニットの活性化の変化に対するコストの感度である。数学的には各活性化に対するコストの偏微分となる。次に, コストの微分 (感度) を活性化から前活性化へ, 結合係数から下層の活性化へとネットワークを逆に伝搬させていく。これらの各変数に対するコストの感度は, ネットワークの下流にある変数に対するコストの感度に依存する。合成関数の微分法則を適用して微分をネットワークに逆伝播することで, すべての偏微分を計算するための効率的なアルゴリズムが得られる。

ここで重要なのは, 各結合係数に対するコストの偏微分を計算する。ある結合係数を考えてみよう (図 2 の赤矢印)。この結合係数は, ある層のソースユニットと次の層のターゲットユニットをつなぐものである。与えられた入力パターンに対する重みのコストへの影響は, ソースユニットがどれだけ活性化したがに依存する。ソースユニットが現在の入力パターンに対してオフの場合, 結合係数は送信する信号を持たず, その結合係数はネットワークが現在の入力に対して生成する出力には無関係である。ソースユニットの活性化に結合係数を掛けて, ターゲットユニットの前活性化への貢献度を決定する。ソースの活性化は結合係数のコストへの影響を決定する 1 つの要因となる。もう 1 つの要因は, ターゲットユニットの前活性化に対するコストの感度である。もしターゲットユニットの前活性化がコストに影響を与えないのであれば, 結合係数も影響を与えない。結合係数に関するコストの微分は, ソースユニットの活性化とターゲットユニットのコストへの影響の積である。

各結合係数は, コスト (誤差) を減らす方向に, 結合係数に対するコストの微分に比例した量で調整する。このプロセスは, コストが最も急峻に減少する重み空間の方向に移動することになるので, 勾配降下と呼ばれている。直感的に理解するために, 2 つのアプローチを考えてみよう。まず, 個々の学習例ごとにコストを下げるためのステップを踏むアプローチを考える。勾配降下法では誤差を減らすため最小限の選択的な調整を行う。これは現在の例からの学習が他の例から学んだことに干渉しないようにするためのもので理にかなっている。しかし目標はすべての例での誤差の合計である全体の誤差を減らすことである。そこで次に一歩を踏み出す前に, すべての例の誤差面 (あるいは同等の勾配) を合計するというアプローチを考えてみよう。誤差面は非線形であり, ネットワークを線形化した点から離れると勾配が変化するため, 小さな一歩しか踏み出せない。

実際, 最良の解決策はステップを踏む前に学習例のミニバッチを使って勾配を推定することである。これは，単一例のアプローチと比較してより安定した方向性を得ることができる。完全な訓練データセットアプローチと比較して, ステップを踏むのに必要な計算量を大幅に減らすことができる。完全訓練データセットアプローチでは, 訓練データセットの誤差の正確な勾配が得らる。誤差関数が非線形であるため, 大きなステップを踏むことはできない。ミニバッチを使用することは勾配推定値の安定性と計算コストの間の良い妥協点である。勾配推定値は現在のバッチに含まれる例のランダムサンプルに依存するため, この手法は確率的勾配降下法 (SGD)と呼ばれている。確率的勾配降下法

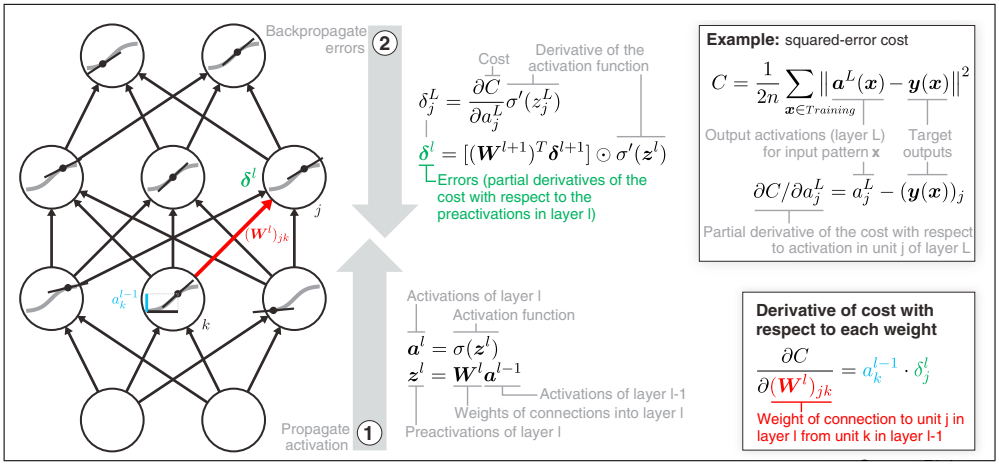


図2　誤差逆伝播アルゴリズム。誤差逆伝播は, 結合係数をわずかに調整することで, ネットワークが最小化しようとしているコスト関数にどのような影響を与えるかを計算するための効率的なアルゴリズムである。例として, 2 つの隠れ層を持つフィードフォワードネットワークを示す。まず, 活性化はフィードフォワード方向 (上向き) に伝搬される。活性化関数 (灰色のシグモイド) は，各ユニット (円) に示されている。特定の入力パターン (図示せず) の文脈では, ネットワークは特定の活性化状態にあり, ユニットの黒い点で示される (横軸: 前活性化, 縦軸:活性化)。. 次に, コスト関数の導関数 (右図の二乗誤差損失) は逆に (下向きに)伝搬される。今回の入力パターンでは, このネットワークは線形ネットワークとして近似される (活性化関数の傾きを示す黒線)。合成関数の微分法則とは, 活性化, 前活性化, 結合係数の小さな変化がコスト (誤差) にどのような影響を与えるかを定義するものである。目的は各重みに対するコストの偏微分を計算することである (右下)。そして各重みはその調整によってコストがどれだけ減少するかに比例して調整される。表記は Nielsen (2015) にほぼ従ったが，ベクトルや行列には太字記号を使っている。

は先に述べた動機に加えて, 訓練データセットを超えて十分に一般化する解を見つけることにも貢献すると考えられている。

コストは重みの凸関数ではないので, 局所最小に陥るのではないかと心配になるかもしれない。しかし, 結合係数空間の高次元性は, 勾配降下法にとって (呪いではなく) 恵みであることがわかる。つまり脱出する方向がたくさんあり, 誤差面がすべての方向に向かって上昇するような, 行き詰まりを感じることはまずない。実際には, 局所最小よりもサドルポイント (勾配が消滅する場所) の方が大きな問題となる。さらにコスト関数には多くの対称性があり, 任意の結合係数セットには計算上同等の双子が多数存在する (つまりモデルは異なるパラメータ設定でも同じ全体的な関数を計算する)。その結果我々の解決策は多くの局所最小の中の 1 つかもしれないが，最適ではない局所最小ではないかもしれず，多くの同様の良い解決策の 1 つかもしれない。

5　リカレントニューラルネットワークは，力学系の普遍近似器

ここまでは有向結合が円環を形成しないフィードフォワードネットワークについて考えてきた。しかし、脳のように活動が周期的に伝播するリカレントニューラルネットワーク (RNN) でもユニットを構成することができる。これによりネットワークは限られた計算資源を時間をかけて再利用し, より深い一連の非線形変換を行うことができる。その結果 RNN は同じ数のユニットと接続を用いて単一のフィードフォワードを通過させる場合よりも複雑な計算を行うことが可能である。

与えられた状態空間に対して適切な RNN は各状態を任意の後継状態にマッピングすることができる。したがってRNN は力学系の普遍的な近似モデルとなる。 RNN は力学系をモデル化するための普遍的な言語であり, その構成要素は生物学的なニューロンで実装することが可能である。

RNN はフィードフォワードニューラルネットワークと同様, 誤差逆伝播法によって学習することができる。しかし

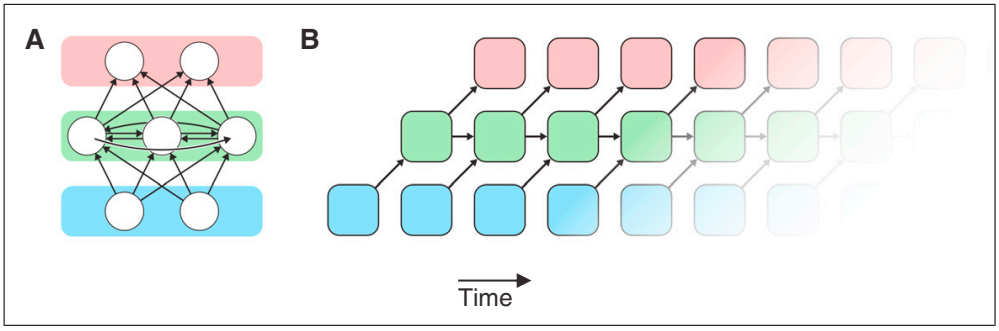


図3　リカレント・ニューラル・ネットワーク。 (A) 2 つの入力ユニット (青), 3 つの隠れユニット(緑), 2 つの出力ユニット (ピンク) からなるリカレントニューラル・ネットワークモデル。隠れユニットは完全にリカレント接続されており各ユニットはその出力を他の両方のユニットに送っている。矢印は特定のユニット間のスカラー結合係数を表している。 (B) フィードフォワードネットワークに相当。 RNN はフィードフォワードネットワークとして時間展開することができる。そのために RNN のユニット (（青, 緑, ピンクのセット）は時間ステップごとに複製される。ここでの矢印は色のついた箱内のユニットのセット間の重み行列を表している。この等価性が成立するためにはフィードフォワードネットワークの深さが RNN が実行される時間ステップ数と一致している必要がある。展開すると表現は簡潔ではなくなるが理解しやすくなる。 RNN のソフトウェア実装に役立つ。誤差逆伝播によるリカレントモデルの学習は誤差逆伝播法によるアンフォールドモデルの学習と同等である。

誤差逆伝播法ではサイクルを逆に進めなければならない。このプロセスは通時バックプロパゲーションと呼ばれている。 RNN とバックプロパゲーションを直感的に理解するには RNN を等価なフィードフォワードネットワークに「展開」することである (図3)。フィードフォワードネットワークの各層は RNN の 1 つの時間ステップを表している。

RNN のユニットと結合係数はフィードフォワードネットワークの各層に複製される。フィードフォワードネットワークは層間で同じ重みのセット (リカレントネットワークの重み) を共有する。

独立した観測データを扱う課題 (例えば静止画の分類) では重みを再利用することで RNN は同じ数のパラメータを持つフィードフォワードネットワークよりも優れた性能を発揮する。 RNN は依存性のある観測データ系列を処理する課題において非常に優れている。 RNN は内部状態 (メモリ) を長期的に維持しダイナミクスを生成できる。このため時間的パターンを認識・生成する必要がある課題に適している。例えば音声や動画の認識エージェントの隠れ状態 (目標など) や環境表現 (現在隠れている対象など) を維持する必要のある認知課題, ある言語から別の言語への翻訳などの言語課題, 行動を計画・選択するレベルの制御課題や感覚からのフィードバックを受けて行動を実行する際の運動制御レベルの制御課題などが挙げられる。

6　ディープニューラルネットワークは生物学的ニューラルネットワークの抽象化したモデル

認知モデルは脳の情報処理の側面を捉えているが, その生物学的な実装については語っていない。詳細な生物学的モデルは活動電位のダイナミクスや樹状突起や軸索における信号伝播の時空間的なダイナミクスを捉えることができる。しかしこれらのプロセスが認知にどのように寄与しているかを説明するには限られた成功しか収めていない。ここで紹介しているディープニューラルネットワークモデルはそのバランスをとっており知覚, 認知, 運動制御などの機能を抽象度の高いユニットのネットワークで説明することができる。

エンジニアにとって人工ディープニューラルネットワークは機械学習の強力なツールとなる。神経科学者にとっては脳がどのようにして認知機能を実現しているのかそのメカニズムの仮説を明確にするためのモデルとなる。ディープニューラルネットワークは情報処理機能を表現するための強力な言語となる。ある領域では生物学的に妥当な操作

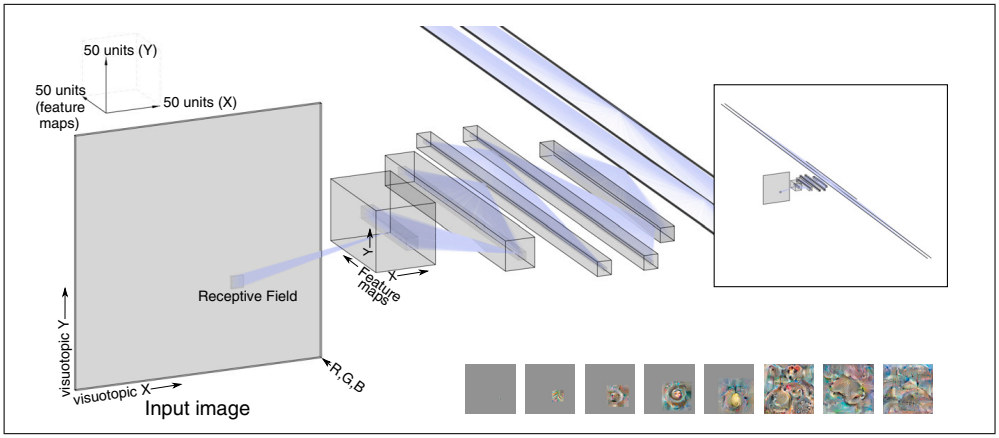


図4　畳み込み型のフィードフォワードニューラルネットワーク。ディープニューラルネットワークが注目されるきっかけとなった畳み込みディープニューラルネットワークのアーキテクチャ「Alexnet」の一般的な構造。このモデルのオリジナルレポートでの視覚化とは異なり, ここではテンソルの次元がスケールに合わせて描かれている。このため畳み込みディープニューラルネットワークが入力画像を空間表現から意味表現へと徐々に変換していく様子を理解しやすくなっている。簡単のため, プーリング操作や, 2 つの GPU に分割された層の一部は表示していない。左のボックスは, 入力画像 (227 × 227 × 3 のテンソル, 227 は正方形の入力画像の縦横の長さ 3 は色成分数)。このテンソルは空間的サイズは 55×55 と小さいが特徴量マップの数が 96 と多い。このテンソルの各特徴マップは原画像と特定の 11×11×3 のフィイルタとの畳み込みによって生成される。そのためこの層の各ユニットの前活性化は画像中の 1 つの長方形の受容野の線形結合である。このような受容野の境界は画像テンソル内の小さなボックスとして視覚化されている。次の第 2 層では空間的にはさらに小さくなる(27×27), しかし, 特徴マップの数は 256 と豊富になる。ここから先の各特徴は画素の線形結合ではなく, 前層の特徴との線形結合であることに注意。第 6 層 (右上図) では第 5 層のすべての特徴マップと位置を組み合わせて 4096 個の異なるスカラーユニットを生成する。各ユニットには制限のない入力重みベクトルがある。最後の第 8 層には出力クラスごとに 1 つずつ合計 1000 個のユニットが配置されている。下段の 8 枚の画像はランダムなノイズ画像を徐々に変化させることで 8 層それぞれの特定のユニットを活性化させたもの。右端画像はクラス「モスク」に関連する出力ニューロンを活性化するように最適化されている。重要なのはこれらの画像は活性化最大化問題の局所解にすぎないということである。異なる開始条件や最適化のヒューリスティックを使用することで別の活性化最大化画像を作成することができる。

のみに頼りながら, 視覚的物体認識やボードゲームなど, すでに人間レベルの性能を満たしているか, それ以上の性能を発揮している。

工学分野におけるニューラルネットワークモデルは「計算には複数の入力を非線形に組み合わせて 1 つの出力を算出するユニットのネットワークが必要である」という一般的な概念をはるかに超えて脳からインスピレーションを得ている。例えばコンピュータビジョンの分野で主流となっている畳み込みニューラルネットワークでは網膜層の深い階層を利用しておりユニットの受容野が制限されている。畳み込みニューラルネットワークでは結合係数のテンプレートが画像上の複数の場所で自動的に共有される (特徴マップの前活性化の計算は入力と重みテンプレートの畳み込みに相当)。畳み込みの方式は霊長類の視覚システムの生来の特徴を捉えていないかもしれないが, 霊長類の発達と学習の最終的な成果を理想化したものであり, 処理の初期段階で網膜上のマップ全体から質的に類似した特徴が抽出される。これらのネットワークは層を越えて画像の視覚空間表現をその内容の意味的表現に変換しマップの空間的な詳細さを徐々に減らし, 意味的な次元の数を増やしていく (図4)。

ニューラルネットワークモデルが生物学の抽象的な特徴から着想を得て, ある課題における人間や動物の全体的な成績と一致したからといって, それが人間や動物の脳がその課題をどのように実行するかの良いモデルになるとは限らない。しかしニューラルネットワークモデルと脳の比較は, 特定の刺激に対する誤りや反応時間などの詳細な行動パターンの観点から行うことができる。さらに, ニューラルネットワークの内部表現と脳の内部表現を比較すること

もできる。

「ホワイトボックス」アプローチではモデルの内部表現を見て評価を行う。ニューラルネットワークモデルは特定の刺激に対する異なる脳領域での表現を予測するための基礎となる。ひとつのアプローチは符号化モデルと呼ばれるものである。符号化モデルではある機能領域の脳活動パターンをモデルのある層にある表現の線形変換を用いて予測する。もう 1 つのアプローチは表現類似性分析と呼ばれる。表現類似性分析では, 脳とモデルの各表現を非類似性行列で特徴づける。モデルは刺激対間の表現の非類似性を説明する能力によって評価される。 3 つ目のアプローチはパターン要素モデリングである。このアプローチの表現は活動プロファイルの 2 番目として特徴付けられる。

視覚的物体認識の分野における最近の研究成果によると, 霊長類の脳が一目で迅速な認識を行う仕組みについては, ディープ畳み込みニューラルネットワークが最良のモデルであることが示されている。しかし，ニューロン応答の説明可能な分散のすべてを説明できるわけではない。

「ブラックボックス」アプローチではモデルの動作に基づいて評価する。挙動の詳細なパターンを説明できないモデルを否定することができる。畳み込みニューラルネットワークはノイズの多い環境下では人間とは異なる振る舞いをすることや, 模範解答の間で異なるパターンの失敗を示すことなど, すでにその限界が明らかになっている。

ディープニューラルネットワークは神経生物学と認知機能の間のギャップを埋め脳の情報処理をモデル化するためのエキサイティングなフレームワークを提供する。脳の計算方法に関する理論はシミュレーションによって厳密に検証することができるようになった。我々の理論とそれを実装したモデルは動物や人間の現代的な技術によって得られた脳の活動や沢山の行動の測定値を説明できるようになるにつれて進化していくだろう。

7　より進んだ文献

- Dayan, P., and Abbott, L.F. (2001). Chapter 7.4, Recurrent neural networks. In Theoretical Neuroscience (Cambridge, MA: MIT Press).
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning (MIT Press).
- Hassabis, D., Kumaran, D., Summerfi eld, C., and Botvinick, M. (2017). Neuroscience-inspired artifi cial intelligence. Neuron 95, 245–258.
- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. Nature 521, 436–444.
- Kietzmann, T., McClure, P., and Kriegeskorte, N. (2019). Deep neural networks in computational neuroscience. In Oxford Research Encyclopedia of Neuroscience. https://doi.org/10.1093/acrefore/9780190264086.013.46
- Kriegeskorte, N. (2015). Deep neural networks: a new framework for modeling biological vision and brain information processing. Annu. Rev. Vis. Sci. 1, 417–446.
- Nielsen, M.A. (2015). Neural Networks and Deep Learning (Determination Press).
- Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural Netw. 61, 85–117.
- Storrs, K.R. and Kriegeskorte, N. (2019). Deep learning for cognitive neuroscience. In The Cognitive Neurosciences (6th Edition), M. Gazzaniga, ed. (Boston: MIT Press).
- Yamins, D.L.K., and DiCarlo, J.J. (2016). Using goal-driven deep learning models to understand sensory cortex. Nat. Neurosci. 19, 356–365.