LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Page 1

Definition

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.

Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer.

Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. → Major drawback of ML

Definition

Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification.

Deep-learning methods are representation-learning methods with multiple levels of representa- tion, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. With the composition of enough such transformations, very complex functions can be learned.

An image, for example, comes in the form of an array of pixel values, and the learned features in the first layer of representation typically represent the presence or absence of edges at particular orientations and locations in the image. The second layer typically detects motifs by spotting particular arrangements of edges, regardless of small variations in the edge positions. The third layer may assemble motifs into larger combinations that correspond to parts of familiar objects, and subsequent layers would detect objects as combinations of these parts.

motif: pattern on something

In a typical deep-learning system, there may be hundreds of millions of these adjustable weights, and hundreds of millions of labelled examples with which to train the machine.

To properly adjust the weight vector, the learning algorithm com- putes a gradient vector that, for each weight, indicates by what amount the error would increase or decrease if the weight were increased by a tiny amount. The weight vector is then adjusted in the opposite direc- tion to the gradient vector.

The objective function, averaged over all the training examples, can

Page 2

be seen as a kind of hilly landscape in the high-dimensional space of weight values. The negative gradient vector indicates the direction of steepest descent in this landscape, taking it closer to a minimum, where the output error is low on average.

In practice, most practitioners use a procedure called stochastic gradient descent (SGD).

It is called stochastic because each small set of examples gives a noisy estimate of the average gradient over all examples.

After training, the performance of the system is measured on a different set of examples called a test set. This serves to test the generalization ability of the machine — its ability to produce sensible answers on new inputs that it has never seen during training.

Page 3

shallow classifiers require a good feature extractor that solves the selectivity–invariance dilemma

To make classifiers more powerful, one can use generic non-linear features, as with kernel methods[redacted], but generic features such as those arising with the Gaussian kernel do not allow the learner to general- ize well far from the training examples[redacted]. The conventional option is to hand design good feature extractors, which requires a consider- able amount of engineering skill and domain expertise. But this can all be avoided if good features can be learned automatically using a general-purpose learning procedure. This is the key advantage of deep learning.

Definition

A deep-learning architecture is a multilayer stack of simple mod- ules, all (or most) of which are subject to learning, and many of which compute non-linear input–output mappings.

Each module in the stack transforms its input to increase both the selectivity and the invariance of the representation.

With multiple non-linear layers, say a depth of 5 to 20, a system can implement extremely intricate func- tions of its inputs that are simultaneously sensitive to minute details

and insensitive to large irrelevant variations

As long as the modules are relatively smooth functions of their inputs and of their internal weights, one can compute gradients using the backpropagation procedure.

the derivative (or gradi- ent) of the objective with respect to the input of a module can be computed by working backwards from the gradient with respect to the output of that module (or the input of the subsequent module)

To go from one layer to the next, a set of units compute a weighted sum of their inputs from the previous layer and pass the result through a non-linear function. At present, the most popular non-linear function is the rectified linear unit (ReLU), which is simply the half-wave rectifier $f(z) = \max(z, 0)$. In past decades, neural nets used smoother non-linearities, such as $\tanh(z)$ or $1/(1 + \exp(-z))$, but the ReLU typically learns much faster in networks with many layers, allowing training of a deep supervised network without unsupervised pre-training

Definition

Units that are not in the input or output layer are conventionally called hidden units.

The hidden layers can be seen as distorting the input in a non-linear way so that categories become linearly separable by the last layer

In practice, poor local minima are rarely a problem with large net- works.

Regardless of the initial conditions, the system nearly always reaches solutions of very similar quality.

Selectivity - Invariance Dilemma

→ Selectivity: Distinguish between different inputs.
→ Invariance: Ignore irrelevant variations of the same input.

→ Dilemma:
- → Increasing invariance often reduces selectivity and increasing selectivity often reduces invariance.

Why ReLU learns much faster in networks with many layers?

→ No saturation for +ve inputs
- → For active neurons, gradient pass unchanged through many layers.

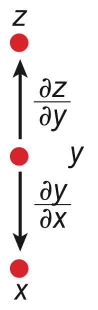
→ Mitigates vanishing gradient problem
- → It keeps gradients large enough for effective learning.

→ Sparse activations
- → Many neurons output exactly 0.
- → Simpler and robust representations → faster optimization.

→ Piecewise linear
- → Easier to optimize than highly curved surfaces.

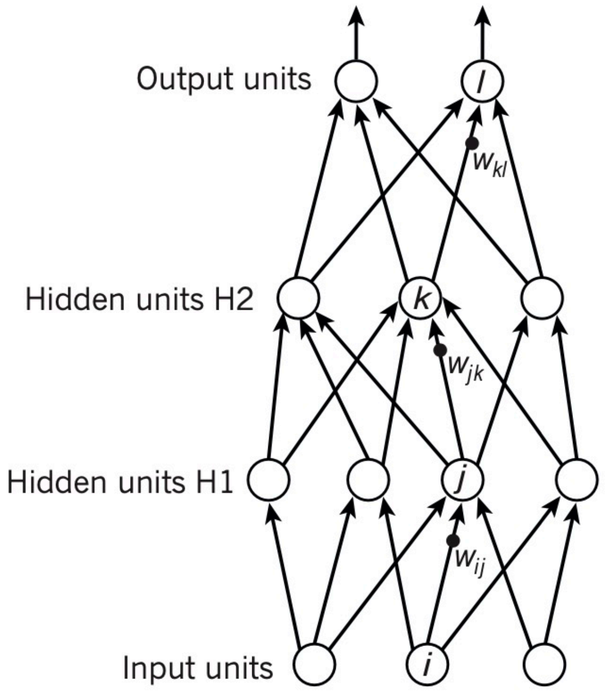
→ Stable forward signal
- → Activations do not shrink into a narrow range.

Chain Rule:



z
$\frac{\partial z}{\partial y}$
y
$\frac{\partial y}{\partial x}$
x

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$



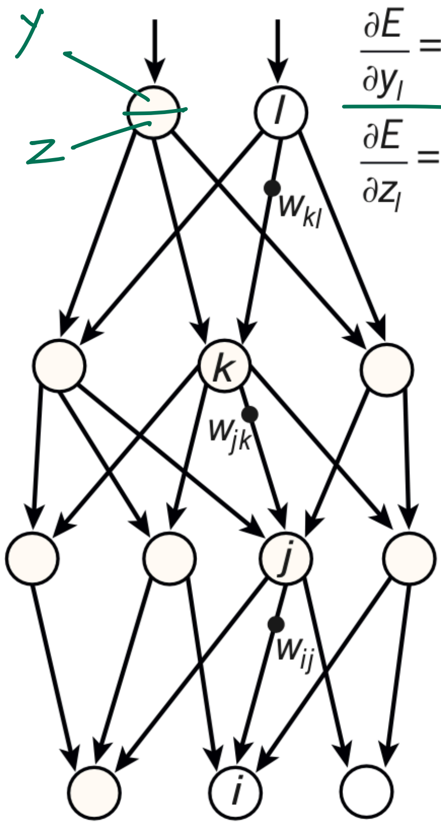
$$y_l = f(z_l)$$
$$z_l = \sum_{k \,\varepsilon\, \text{H2}} w_{kl}\, y_k$$

$$y_k = f(z_k)$$
$$z_k = \sum_{j \,\varepsilon\, \text{H1}} w_{jk}\, y_j$$

$$y_j = f(z_j)$$
$$z_j = \sum_{i \,\varepsilon\, \text{Input}} w_{ij}\, x_i$$

$$W_{ab} \to z_b \to y_b \to E \;\Big|\; \frac{\partial E}{\partial W_{ab}} = \frac{\partial E}{\partial z_b}\frac{\partial z_b}{\partial W_{ab}} \;\Big|\; \frac{\partial z_b}{\partial W_{ab}} = \frac{\partial \sum W_{ab} y_a}{\partial W_{ab}} = y_a$$

$$\frac{\partial E}{\partial W_{ab}} = (\text{Error at destination}) \times (\text{Activation from source})$$



$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$$

$$\frac{\partial E}{\partial W_{kl}} = \frac{\partial E}{\partial z_l} \cdot y_k$$

$$\frac{\partial E}{\partial y_k} = \sum_{l \,\varepsilon\, \text{out}} w_{kl}\frac{\partial E}{\partial z_l}$$

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$$

$$\frac{\partial E}{\partial W_{jk}} = \frac{\partial E}{\partial z_k} \cdot y_j$$

$$\frac{\partial E}{\partial y_j} = \sum_{k \,\varepsilon\, \text{H2}} w_{jk}\frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$$

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial z_j} \cdot x_i$$

Page 4

For smaller data sets, unsupervised pre-training helps to prevent overfitting[redacted], leading to significantly better generalization when the number of labelled examples is small, or in a transfer setting where we have lots of examples for some 'source' tasks but very few for some 'target' tasks.

ConvNets are designed to process data that come in the form of multiple arrays

1D for signals and sequences, including language; 2D for images or audio spectrograms; and 3D for video or volumetric images.

There are four key ideas behind ConvNets that take advantage of the properties of natural signals: local connections, shared weights, pooling and the use of many layers.

the role of the convolutional layer is to detect local con- junctions of features from the previous layer

the role of the pooling layer is to merge semantically similar features into one

stride

Neighbouring pooling units take input from patches that are shifted by more than one row or column, thereby reducing the dimension of the representation and creating an invariance to small shifts and dis- tortions

Two or three stages of convolution, non-linearity and pool- ing are stacked, followed by more convolutional and fully-connected layers.

Deep neural networks exploit the property that many natural sig- nals are compositional hierarchies, in which higher-level features are obtained by composing lower-level ones.

The pooling allows representations to vary very little when elements in the previous layer vary in position and appearance.

→ Makes representation robust

→ Throws away exact spatial detail

→ Keeps presence of a feature

$[0.1, 0.9, 0.2, 0.3] \rightarrow 0.9$

$[0.1, 0.2, 0.9, 0.1] \rightarrow 0.9$

Page 5

ConvNets are easily amenable to efficient hardware implemen- tations in chips or field-programmable gate arrays

The hidden layers of a multilayer neural network learn to repre- sent the network's inputs in a way that makes it easy to predict the target outputs.

Page 6

Each word in the context is presented to the network as a one-of-N vector, that is, one component has a value of 1 and the rest are 0.

In the first layer, each word creates a different pattern of activations, or word vectors

other layers of the network learn to convert the input word vec- tors into an output word vector for the predicted next word, which can be used to predict the probability for any word in the vocabulary to appear as the next word

The network learns word vectors that contain many active components each of which can be interpreted as a separate feature of the word

These semantic features were not explicitly present in the input. They were discovered by the learning procedure as a good way of factorizing the structured relationships between the input and output symbols into multiple 'micro-rules'.

Before the introduction of neural language models[redacted], the standard approach to statistical modelling of language did not exploit distrib- uted representations: it was based on counting frequencies of occur- rences of short symbol sequences of length up to N (called N-grams). The number of possible N-grams is on the order of
$V^N \leftarrow$ VN, where V is the vocabulary size

N-grams treat each word as an atomic unit, so they cannot generalize across semantically related sequences of words, whereas neural language valued features, and semantically related words end up close to each models can because they associate each word with a vector of real other in that vector space

RNNs process an input sequence one element at a time, maintaining in their hidden units a 'state vector' that implicitly contains information about the history of all the past elements of the sequence.

RNNs are very powerful dynamic systems, but training them has proved to be problematic because the backpropagated gradients either grow or shrink at each time step, so over many time steps they typically explode or vanish

after reading an English sentence one word at a time, an English 'encoder' network can be trained so that the final state vector of its hidden units is a good representation of the thought expressed by the sentence

This thought vector can then be used as the initial hidden state of (or as extra input to) a jointly trained French 'decoder' network, which outputs a prob- ability distribution for the first word of the French translation

If a particular first word is chosen from this distribution and provided as input to the decoder network it will then output a probability dis- tribution for the second word of the translation and so on until a full stop is chosen

Overall, this process generates sequences of French words according to a probability distribution that depends on the English sentence.

Page 7

RNNs, once unfolded in time [redacted], can be seen as very deep feedforward networks in which all the layers share the same weights. Although their main purpose is to learn long-term dependencies, theoretical and empirical evidence shows that it is difficult to learn to store information for very long

To correct for that, one idea is to augment the network with an explicit memory.

The first proposal of this kind is the long short-term memory (LSTM) networks that use special hidden units, the natural behaviour of which is to remember inputs for a long time

A special unit called the memory cell acts like an accumulator or a gated leaky neuron: it has a connection to itself at the next time step that has a weight of one, so it copies its own real-valued state and accumulates the external signal, but this self-connection is multiplicatively gated by another unit that learns to decide when to clear the content of the memory.

neural Turing machines and mem- ory networks are being used for tasks that would normally require reasoning and symbol manipulation

Neural Turing machines can be taught ‘algorithms’.

Memory networks can be trained to keep track of the state of the world in a setting similar to a text adventure game and after reading a story, they can answer questions that require complex inference

Unsupervised learning [redacted] had a catalytic effect in reviving interest in deep learning, but has since been overshadowed by the successes of purely supervised learning.

Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object.

major progress in artificial intelligence will come about through systems that combine representation learning with complex reasoning

Krizhevsky, A., Sutskever, I. & Hinton, G. ImageNet classification with deep convolutional neural networks. In Proc. Advances in Neural Information Processing Systems 25 1090–1098 (2012). This report was a breakthrough that used convolutional nets to almost halve the error rate for object recognition, and precipitated the rapid adoption of deep learning by the computer vision community.

Hinton, G. et al. Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Processing Magazine 29, 82–97 (2012). This joint paper from the major speech recognition laboratories, summarizing the breakthrough achieved with deep learning on the task of phonetic classification for automatic speech recognition, was the first major industrial application of deep learning.

Page 8

Sutskever, I. Vinyals, O. & Le. Q. V. Sequence to sequence learning with neural networks. In Proc. Advances in Neural Information Processing Systems 27 3104–3112 (2014). This paper showed state-of-the-art machine translation results with the architecture introduced in ref. 72, with a recurrent network trained to read a sentence in one language, produce a semantic representation of its meaning, and generate a translation in another language.

Glorot, X., Bordes, A. & Bengio. Y. Deep sparse rectifier neural networks. In Proc. 14th International Conference on Artificial Intelligence and Statistics 315–323 (2011). This paper showed that supervised training of very deep neural networks is much faster if the hidden layers are composed of ReLU.

Hinton, G. E., Osindero, S. & Teh, Y.-W. A fast learning algorithm for deep belief nets. Neural Comp. 18, 1527–1554 (2006). This paper introduced a novel and effective way of training very deep neural networks by pre-training one hidden layer at a time using the unsupervised learning procedure for restricted Boltzmann machines.

Bengio, Y., Lamblin, P., Popovici, D. & Larochelle, H. Greedy layer-wise training of deep networks. In Proc. Advances in Neural Information Processing Systems 19 153–160 (2006). This report demonstrated that the unsupervised pre-training method introduced in ref. 32 significantly improves performance on test data and generalizes the method to other unsupervised representation-learning techniques, such as auto-encoders.

LeCun, Y. et al. Handwritten digit recognition with a back-propagation network. In Proc. Advances in Neural Information Processing Systems 396–404 (1990). This is the first paper on convolutional networks trained by backpropagation

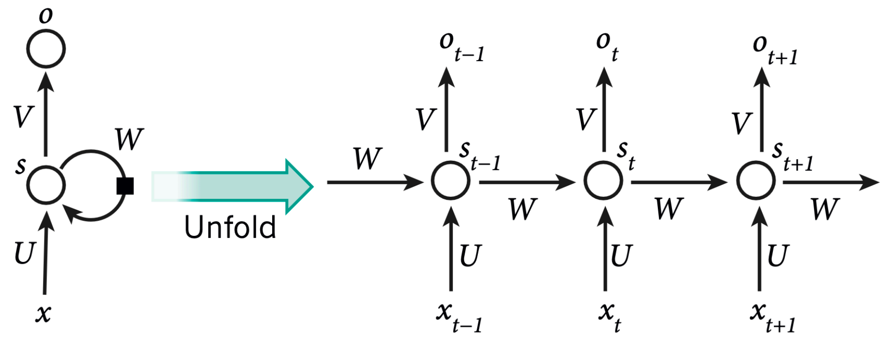
for the task of classifying low-resolution images of handwritten digits.

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. Proc. IEEE 86, 2278–2324 (1998). This overview paper on the principles of end-to-end training of modular systems such as deep neural networks using gradient-based optimization showed how neural networks (and in particular convolutional nets) can be combined with search or inference mechanisms to model complex outputs that are interdependent, such as sequences of characters associated with the content of a document.

Bengio, Y., Ducharme, R. & Vincent, P. A neural probabilistic language model. In Proc. Advances in Neural Information Processing Systems 13 932–938 (2001). This paper introduced neural language models, which learn to convert a word symbol into a word vector or word embedding composed of learned semantic features in order to predict the next word in a sequence.

Hochreiter, S. & Schmidhuber, J. Long short-term memory. Neural Comput. 9, 1735–1780 (1997). This paper introduced LSTM recurrent networks, which have become a crucial ingredient in recent advances with recurrent networks because they are good at learning long-range dependencies.

RNN unfolded in time:



NTM learns how to manipulate memory like algorithms, while memory networks learn how to retrieve and reason over stored information.