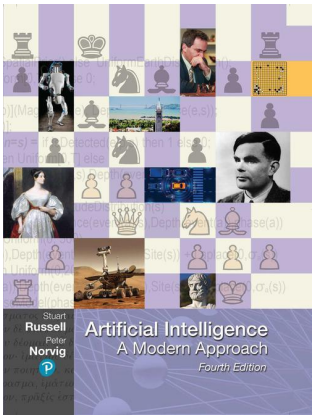
Artificial Intelligence: A Modern Approach

Fourth Edition



Chapter 21

Deep Learning

Outline

- Simple Feedforward Networks
- Computation Graphs for Deep Learning
- Convolutional Networks
- Learning Algorithms
- Generalization
- Recurrent Neural Networks
- Unsupervised Learning and Transfer Learning
- Applications

Simple Feedforward Networks

Feedforward network
- connections only in one direction (input to output)
- directed acyclic graph with designated input and output nodes
- No loops

Recurrent network
- its intermediate or final outputs back into its own inputs.
- signal values within the network form a dynamical system that has internal state or memory

Networks as complex functions
- Each node within a network is called a unit
- calculates the weighted sum of the inputs from predecessor nodes
- applies a nonlinear function to produce its output

$$a_j = g_j\left(\sum_i w_{i,j} a_i\right) \equiv g_j(in_j),$$

- g_j is a nonlinear **activation function,** a_j denotes the output of unit j and $w_{i,j}$ *is* the weight attached to the link from unit i to unit j;

Simple Feedforward Networks

Different activation functions:
- The logistic or sigmoid function

$$\sigma(x) = 1/(1 + e^{-x})$$

- The Rectified Linear Unit, (ReLU) function

$$\text{ReLU}(x) = \max(0, x)$$

- The softplus function, a smooth version of the ReLU function:
  - The derivative of the softplus function is the sigmoid function.

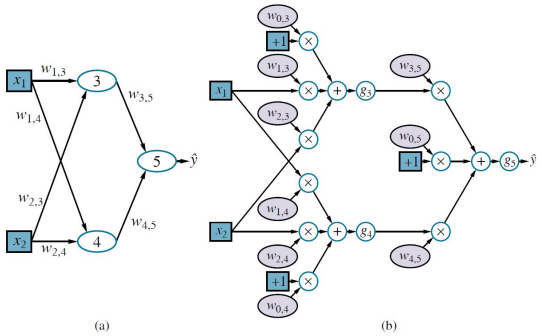
$$\text{softplus}(x) = \log(1 + e^x)$$

- The tanh function:

$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

Note that the range of tanh is $(-1, +1)$. Tanh is a scaled and shifted version of the sigmoid, as $\tanh(x) = 2\sigma(2x) - 1$.

Simple Feedforward Networks



(a) A neural network with two inputs, one hidden layer of two units, and one
output unit. Not shown are the dummy inputs and their associated weights. (b) The network in (a) unpacked into its full computation graph

Simple Feedforward Networks

Gradients and learning
- Using squared loss function, L_2
- Prediction, $\hat{y} = h_{\mathbf{w}}(\mathbf{x})$

- $Loss(h_{\mathbf{w}}) = L_2(y, h_{\mathbf{w}}(\mathbf{x})) = \|y - h_{\mathbf{w}}(\mathbf{x})\|^2 = (y - \hat{y})^2$

back-propagation: the way that the error at the output is passed back through the network

Computation Graphs for Deep Learning

Input encoding

- Input and output nodes of a computational graph are the ones that connect directly to the input data **x** and the output data **y**.
- False is mapped to an input of 0 and true is mapped to 1, or (-1 and +1)
- Numeric attributes used as is
- Categorical attributes with more than two values are usually encoded with one-hot encoding.
  - An attribute with d possible values is represented by d separate input bits.
  - corresponding input bit is set to 1 and other 0.

Computation Graphs for Deep Learning

Output layers and loss functions

Most deep learning applications:

- Common to interpret the output values $\hat{\mathbf{y}}$ as probabilities
- negative log likelihood as the loss function

Minimizing the cross-entropy loss.

- Cross-entropy: $H(P, Q)$, measure of dissimilarity between two distributions P and Q.[2]

$$H(P, Q) = \mathbf{E}_{\mathbf{z}\sim P(\mathbf{z})}[\log Q(\mathbf{z})] = \int P(\mathbf{z}) \log Q(\mathbf{z}) d\mathbf{z}.$$

- softmax layer
  - outputs a vector of d values given a vector *of* input values $\mathbf{in} = \langle in_1, \ldots, in_d \rangle$

$$\text{softmax}(\mathbf{in})_k = \frac{e^{in_k}}{\sum_{k'=1}^{d} e^{in_{k'}}}.$$

  - outputs a vector of nonnegative numbers that sum to 1.
- **Other output layer:** mixture density layer represents the outputs using a mixture of Gaussian distributions

Computation Graphs for Deep Learning

Hidden layers

Intermediate computations before producing the output ***y***.

Different representation for the input **x**.

Each layer transforms the representation produced by the preceding layer to produce a new representation

In the process of forming all these internal transformations, deep networks often discover meaningful intermediate representations of the data

The hidden layers of neural networks are typically less diverse than the output layers.

Convolutional Networks

Convolutional neural network (CNN) is one that contains spatially local connections

Kernel: A pattern of weights that is replicated across multiple local regions

Convolution: the process of applying the kernel to the pixels of the image

- input vector **x** of size n, corresponding to n pixels in a one-dimensional image, and a vector kernel **k** of size l.
- convolution operation,

$$z_i = \sum_{j=1}^{l} k_j x_{j+i-(l+1)/2}.$$

- kernels centers are separated distance called **stride**, s .
- convolution stops at the edges of the image, but padding the input with extra pixels is possible so that kernel is applied exactly $\lceil n/s \rceil$ times.

Convolutional Networks

Convolution can be viewed as a matrix multiplication

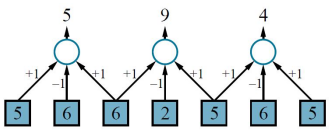
In this weight matrix, the kernel appears in each row, shifted according to the stride relative to the previous row.

$$\begin{pmatrix} +1 & -1 & +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & -1 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 & -1 & +1 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \\ 6 \\ 2 \\ 5 \\ 6 \\ 5 \end{pmatrix} = \begin{pmatrix} 5 \\ 9 \\ 4 \end{pmatrix}.$$

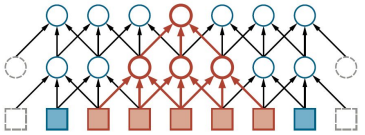
CNNs were inspired originally by models of the visual cortex WHERE the receptive field of a neuron can affect that neuron's activation.

In a CNN, the receptive field of a unit in the first hidden layer is small (size of kernel)

Convolutional Networks



One-dimensional convolution operation with a kernel of size $l = 3$ and a stride $s = 2$.



The first two layers of a CNN for a 1D image with a kernel size $l = 3$ and a stride $s = 1$. Padding is added at the left and right ends in order to keep the hidden layers the same size as the input.

Convolutional Networks

Pooling and downsampling

A pooling layer in a neural network summarizes a set of adjacent units from the preceding layer with a single value

Common forms of pooling:

- **Average-pooling**: computes the average value of its l inputs
  - coarsen the resolution of the image, downsample by a factor of s.
  - $10s$ pixels $=10$ pixels after pooling.
  - Classifier can recognize in the pooled image: facilitates multiscale recognition

- **Max-pooling:** computes the maximum value of its l inputs.
  - Can be used for downsampling
  - logical disjunction, saying that a feature exists somewhere in the unit's receptive field.

Convolutional Networks

Tensor operations in CNNs

tensors, which (in deep learning terminology) are simply multidimensional arrays
of any dimension

Vectors and matrices are one-dimensional and two-dimensional special cases of tensors

a way of keeping track of the "shape" of the data as it progresses through the layers of the network

Computational efficiency of tensor operations: given a description of a network as a sequence of tensor operations, deep learning software package can generate compiled code that is highly optimized for the underlying computational substrate

Are run on GPUs (graphics processing units) or TPUs (tensor processing units), which make available a high degree of parallelism

Convolutional Networks

Residual networks

Residual networks avoid the problem of vanishing gradients in typical deep models

Typical deep models: each layer completely replaces the representation from the preceding layer

key idea of residual networks: a layer should perturb the representation from the previous layer rather than replace it entirely

$$\mathbf{z}^{(i)} = \mathbf{g}_i(\mathbf{z}^{(i-1)} + f(\mathbf{z}^{(i-1)}))$$

$\mathbf{z}^{(i)}$: values of the units in layer i, $\mathbf{g}_i$: the eactivation functions for the residual layer
f as the **residual**, perturbing the default bhavior of passing layer $i-1$ through to layer i.

A neural network with one linear layer combined with on linear layer:

$$f(\mathbf{z}) = \mathbf{V}\mathbf{g}(\mathbf{W}\mathbf{z})$$

where $\mathbf{W}$ and $\mathbf{V}$ are learned weight matrices with the usual bias weights added.

Learning Algorithms

Computing gradients in computation graphs

Gradient of the loss function could be computed by back-propagating error information from the output layer of the network to the hidden layers

The back-propagation process passes messages back along each link in the network

The messages are all partial derivatives of the loss L.

$$\partial L/\partial h = \partial L/\partial h_j + \partial L/\partial h_k$$

Weight-sharing is handled by treating each shared weight as a single node with multiple outgoing arcs in the computation graph

Learning Algorithms

Computing gradients in computation graphs

The gradient for the shared weight is the sum of the gradient contributions from each place it is used in the network

cost is linear in the number of nodes in the computation graph

all of the gradient computations can be prepared in symbolic form in advance and compiled into very efficient code for each node in the graph.

drawback of back-propagation:

- requires storing most of the intermediate values that were computed during forward propagation in order to calculate gradients in the backward pass

Learning Algorithms

Batch normalization

- improves the rate of convergence of SGD
- rescaling the values generated at the internal layers of the network from the examples within each minibatch

$$\hat{z}_i = \gamma \frac{z_i - \mu}{\sqrt{\epsilon + \sigma^2}} + \beta,$$

where μ is the mean value of z across the minibatch, σ is the standard deviation of $z_1, \ldots, z_m$, ϵ is a small constant added to prevent division by zero, and γ and β are learned parameters.

Batch normalization standardizes the mean and variance of the values, as determined by the values of β and γ.

Without batch normalization

- information can get lost if a layer's weights are too small
- the standard deviation at that layer decays to near zero

Generalization

Choosing a network architecture

Some neural network architectures are explicitly designed to generalize well on particular types of data

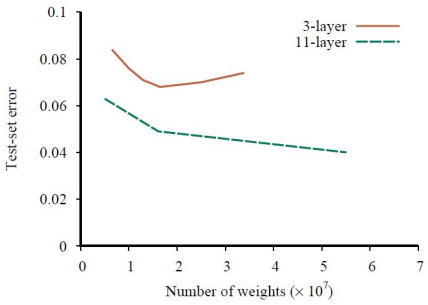
When comparing two networks with similar numbers of weights, the deeper network usually gives better generalization performance.

Deep learning systems perform better than any other pure machine learning approaches for high dimensional inputs (images, video, speech signals, etc)

Deep learning models lack the compositional and quantificational expressive power

May also produce unintuitive errors. Tend to produce input–output mappings that are discontinuous

Generalization



Test-set error as a function of layer width (as measured by total number of weights) for three-layer and eleven-layer convolutional networks. The data come from early versions of Google's system for transcribing addresses in photos taken by Street View cars

Generalization

Neural architecture search

- neural architecture search to explore the state space of possible network architectures.
- Evolutionary algorithms: recombination (joining parts of two networks together) and mutation (adding or removing a layer or changing a parameter value)
- Hill climbing with mutation operations
- major challenge is estimating the value of a candidate network
- train one big network, search for subgraphs of the network that perform better
- heuristic evaluation function

Generalization

Weight decay

- weight decay similar to regularization
- adding a penalty to loss function

$$\lambda \sum_{i,j} W_{i,j}^2$$

- λ strength of the penalty, sum over all of the weights in the networ
- beneficial effect of weight decay is that it implements a form of maximum a posteriori (MAP) learning

$$\begin{aligned} h_{\text{MAP}} &= \operatorname*{argmax}_{\mathbf{w}} P(\mathbf{y}\,|\,\mathbf{X},\mathbf{W})P(\mathbf{W}) \\ &= \operatorname*{argmin}_{\mathbf{w}} [-\log P(\mathbf{y}\,|\,\mathbf{X},\mathbf{W}) - \log P(\mathbf{W})]\,. \end{aligned}$$

- usual cross-entropy loss; the second term prefers weights that are likely under a prior distribution

Generalization

Dropout

intervene to reduce the test-set error of a network—at the cost of making it harder to fit the training set:

- introducing noise at training time, the model is forced to become robust to noise
- Hidden units trained with dropout useful & compatible with many other possible sets of other hidden units that may or may not be included in the full model.
- dropout approximates the creation of a large ensemble of thinned networks
- applied to later layers in a deep network forces the final decision to be made robustly by paying attention to all of the abstract features of the example

dropout forces the model to learn multiple, robust explanations for each input

Recurrent Neural Networks

Recurrent neural networks (RNNs) allow cycles in the computation graph

each cycle has a delay

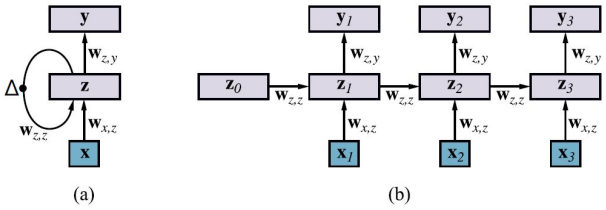
- units may take as input a value computed from their own output at an earlier step
- RNN has an internal state/memory
- RNNs add expressive power compared to feedforward networks,

Training a basic RNN

- input layer $\mathbf{x}$, a hidden layer $\mathbf{z}$ with recurrent connections, and an output layer $\mathbf{y}$,
- $\mathbf{g}_z$ and $\mathbf{g}_y$ denote the activation functions for the hidden and output layers, respectively.

$$\begin{aligned} \mathbf{z}_t &= f_{\mathbf{W}}(\mathbf{z}_{t-1},\mathbf{x}_t) = \mathbf{g}_z(\mathbf{W}_{z,z}\mathbf{z}_{t-1} + \mathbf{W}_{x,z}\mathbf{x}_t) \equiv \mathbf{g}_z(\mathbf{in}_{z,t}) \\ \hat{\mathbf{y}}_t &= \mathbf{g}_y(\mathbf{W}_{z,y}\mathbf{z}_t) \equiv \mathbf{g}_y(\mathbf{in}_{y,t})\,, \end{aligned}$$

Recurrent Neural Networks



(a) Schematic diagram of a basic RNN where the hidden layer **z** has recurrent connections; the Δ symbol indicates a delay.

(b) The same network unrolled over three time steps to create a feedforward network. Note that the weights are shared across all time steps.

Recurrent Neural Networks

Now the gradient for the hidden unit z_t can be obtained from the previous time step as follows:

$$\frac{\partial z_t}{\partial w_{z,z}} = \frac{\partial}{\partial w_{z,z}} g_z(in_{z,t}) = g_z'(in_{z,t}) \frac{\partial}{\partial w_{z,z}} in_{z,t} = g_z'(in_{z,t}) \frac{\partial}{\partial w_{z,z}} \left(w_{z,z} z_{t-1} + w_{x,z} x_t + w_{0,z}\right)$$
$$= g_z'(in_{z,t}) \left(z_{t-1} + w_{z,z} \frac{\partial z_{t-1}}{\partial w_{z,z}}\right),$$

Gradient expression is recursive

Backpropagation through time:
- The contribution to the gradient from time step t is calculated using the contribution from time step $t - 1$.
- The total run time for computing the gradient will be linear in the size of the network

For sigmoids, tanhs, and ReLUs, g^t 1, so our simple RNN will certainly suffer from the **vanishing gradient problem**

if $w_{z,z} > 1$, we may experience the **exploding gradient** problem.

Recurrent Neural Networks

Long short-term memory RNNs

Has a long-term memory component, **memory cell** (**c**) copied from time step to time step
Has gating units:
- vectors control the flow of information
- elementwise multiplication of the corresponding information vector

Types of gating units:
- **Forget gate f** determines if each element of the memory cell is remembered
- **Input gate i** determines if each element of the memory cell is updated additively
- **Output gate o** determines if each element of the memory cell is transferred to the short-term memory **z**

Gates in LSTM differ from Boolean functions, they have range [0,1] and outputs of a sigmoid

$$\mathbf{f}_t = \sigma(\mathbf{W}_{x,f}\mathbf{x}_t + \mathbf{W}_{z,f}\mathbf{z}_{t-1})$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_{x,i}\mathbf{x}_t + \mathbf{W}_{z,i}\mathbf{z}_{t-1})$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{x,o}\mathbf{x}_t + \mathbf{W}_{z,o}\mathbf{z}_{t-1})$$
$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t + \mathbf{i}_t \odot \tanh(\mathbf{W}_{x,c}\mathbf{x}_t + \mathbf{W}_{z,c}\mathbf{z}_{t-1})$$
$$\mathbf{z}_t = \tanh(\mathbf{c}_t) \odot \mathbf{o}_t,$$

Unsupervised Learning and Transfer Learning

Unsupervised learning

Unsupervised learning algorithms take a training set of unlabeled examples **x**
- **To learn new representations**
- **To learn a generative model**

Probabilistic PCA: A simple generative model
- probabilistic principal components analysis (PPCA)
- **z** chosen from a zero-mean, spherical Gaussian, **x** generated from **z** by applying a weight matrix **W** and adding spherical Gaussian noise:

$$P(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$$
$$P_W(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x}; \mathbf{W}\mathbf{z}, \sigma^2\mathbf{I})$$

The weights **W** (and optionally the noise parameter σ^2) can be learned by maximizing

$$P_W(\mathbf{x}) = \int P_W(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \mathcal{N}(\mathbf{x}; \mathbf{0}, \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I}).$$

can be done by gradient methods or by an efficient iterative EM algorithm

Unsupervised Learning and Transfer Learning

Autoencoders

autoencoder is a model containing two parts:
- an encoder that maps from **x** to a representation $\hat{\mathbf{z}}$
- a decoder that maps from a representation $\hat{\mathbf{z}}$ to observed data **x**.

linear autoencoder, simple encoder where both f and g are linear with a shared weight matrix **W**:

$$\hat{\mathbf{z}} = f(\mathbf{x}) = \mathbf{W}\mathbf{x}$$
$$\mathbf{x} = g(\hat{\mathbf{z}}) = \mathbf{W}^\top\hat{\mathbf{z}}.$$

Variational autoencoder (VAE) more complex can capture more complex kinds of generative models
- use **variational posterior** $Q(\mathbf{z})$, as an approximation
- KL divergence: "as close as possible"

$$D_{KL}(Q(\mathbf{z}) \| P(\mathbf{z} \mid \mathbf{x})) = \int Q(\mathbf{z}) \log \frac{Q(\mathbf{z})}{P(\mathbf{z} \mid \mathbf{x})} d\mathbf{z},$$

- ***variational*** **lower bound** $\mathcal{L}$ (**evidence lower bound**, or **ELBO**)

$$\mathcal{L}(\mathbf{x}, Q) = \log P(\mathbf{x}) - D_{KL}(Q(\mathbf{z}) \| P(\mathbf{z} \mid \mathbf{x}))$$

Unsupervised Learning and Transfer Learning

Variational autoencoders provide a means of performing variational learning in the deep learning setting.

Variational learning involves maximizing $\mathcal{L}$ with respect to the parameters of both P and Q

The decoder $g(\mathbf{z})$ is interpreted defining log P($\mathbf{x} \mid \mathbf{z}$).

Autoregressive model (AR model):
- each element x_i is predicted based on other elements of the vector **x** & has no latent variables
- If **x** fixed size, fully observable and possibly fully connected Bayes net.
- Used in analysis of time series data

Deep autoregressive model:
- linear–Gaussian model is replaced by an arbitrary deep network with a suitable output layer depending on whether x_i is discrete or continuous

Unsupervised Learning and Transfer Learning

Generative adversarial networks

pair of networks that combine to form a generative system
- the **generator**, maps values from **z** to **x**
- the **discriminator**, is a classifier trained to classify inputs **x** as real or fake (generated)

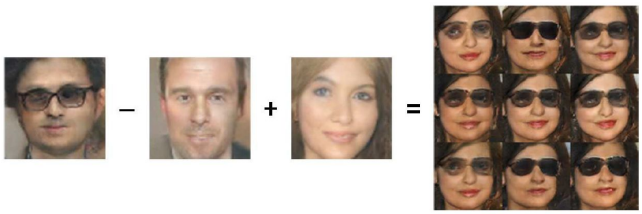
implicit model: samples can be generated but their probabilities are not readily available

generator and the discriminator are trained simultaneously

Generator learning to fool the discriminator and the discriminator learning to accurately separate real from fake data

GANs have worked particularly well for image generation tasks. For example, GANs can create photorealistic, high-resolution images of people who have never existed

Unsupervised Learning and Transfer Learning



A demonstration of how a generative model has learned to use different directions in **z** space to represent different aspects of faces. We can actually perform arithmetic in **z** space. The images here are all generated from the learned model and show what happens when we decode different points in **z** space. We start with the coordinates for the concept of "man with glasses," subtract off the coordinates for "man," add the coordinates for "woman," and obtain the coordinates for "woman with glasses." Images reproduced with permission from (Radford *et al.*, 2015).

Unsupervised Learning and Transfer Learning

Transfer learning and multitask learning

For transfer learning experience with one learning task helps an agent learn better on another task

freeze the first few layers of the pretrained model that serve as feature detectors

modify the parameters of the higher levels only
- problem-specific features and do classification

Common to start with pretrained model such as the RoBERTa model

Followed by fine-tuning the model in two ways:
- giving it examples of the specialized vocabulary used in the desired domain
- training the model on the task it is to perform

Multitask learning is a form of transfer learning in which we simultaneously train a model on multiple objectives

Applications

Vision
- success of the AlexNet deep learning system in the 2012 ImageNet competition that propelled deep learning into the limelight
- supervised learning task with 1,200,000 images in 1,000 different categories
- top-5 error rate has been reduced to less than 2%— below error rate of trained human (5%)

Natural language processing
- machine translation and speech recognition
- end-to-end learning, the automatic generation of internal representations for the meanings of words, and the interchangeability of learned encoders and decoders
- end-to-end learning outperforms classical pipelines
- re-representing individual words as vectors in a high-dimensional space—so-called word embeddings

Reinforcement learning
- decision-making agent learns from a sequence of reward signals that provide some indication of the quality of its behavior
- optimize the sum of future rewards
- learn a value function, a Q-function, a policy, and so on

Summary

- Neural networks represent complex nonlinear functions with a network of parameterized linear-threshold units.
- The back-propagation algorithm implements a gradient descent in parameter space to minimize the loss function.
- Convolutional networks are particularly well suited for image processing and other tasks where the data have a grid topology.
- Recurrent networks are effective for sequence-processing tasks including language modeling and machine translation.