7 Fairly simple neural networks

livebook.manning.com

414

In the late 2010s, when we hear about advances in artificial intelligence, they generally concern a particular subdiscipline known as *machine learning* (computers learning some new information without being explicitly told it). More often than not those advances are being driven by a particular machine-learning technique known as *neural networks*. Although invented decades ago, neural networks have been going through a kind of renaissance as improved hardware and newly discovered research-driven software techniques enable a new paradigm known as *deep learning*.

Deep learning has turned out to be a broadly applicable technique. It has been found useful in everything from hedge fund algorithms to bioinformatics. Two deep-learning applications that consumers have become familiar with are image recognition and speech recognition. If you have ever asked your digital assistant what the weather is, or had a photo program recognize your face, there was probably some deep learning going on.

Deep-learning techniques utilize the same building blocks as simpler neural networks. In this chapter we will explore those blocks by building a simple neural network. It will not be state of the art, but it will give you a basis for understanding deep learning (which is based on more complex neural networks than we will build). Most practitioners of machine learning do not build neural networks from scratch. Instead, they use popular, highly optimized, off-the-shelf frameworks that do the heavy lifting. Although this chapter will not help you learn how to use any specific framework, and the network we will build will not be useful for an actual application, it will help you understand how those frameworks work at a low level.

7.1 Biological basis?

The human brain is the most incredible computational device in existence. It cannot crunch numbers as fast as a microprocessor, but its ability to adapt to new situations, learn new skills, and be creative is unsurpassed by any known machine. Since the dawn of computers, scientists have been interested in modeling the brain's machinery. Each nerve cell in the brain is known as a *neuron*. Neurons in the brain are networked to one another via connections known as *synapses*. Electricity passes through synapses to power these networks of neurons—also known as *neural networks*.

Note

The preceding description of biological neurons is a gross oversimplification for analogy's sake. In fact, biological neurons have parts like axons, dendrites, and nuclei that you may remember from high school biology. And synapses are actually gaps between neurons where neurotransmitters are secreted to enable those electrical signals to pass.

Although scientists have identified the parts and functions of neurons, the details of how biological neural networks form complex thought patterns are still not well understood. How do they process information? How do they form original thoughts? Most of our knowledge of how the brain works comes from looking at it on a macro level. Functional magnetic resonance imaging (fMRI) scans of the brain show where blood flows when a human is doing a particular activity or thinking a particular thought (illustrated in figure 7.1). This and other macro-techniques can lead to inferences about how the various parts are connected, but they do not explain the mysteries of how individual neurons aid in the development of new thoughts.

Figure 7.1 A researcher studies fMRI images of the brain. fMRI images do not tell us much about how individual neurons function, nor how neural networks are organized.

Teams of scientists are racing around the globe to unlock the brain's secrets, but consider this: The human brain has approximately 100,000,000,000 neurons, and each of them may have connections with as many as tens of thousands of other neurons. Even for a computer with billions of logic gates and terabytes of memory, a single human brain would be impossible to model using today's technology. Humans will still likely be the most advanced general-purpose learning entities for the foreseeable future.

Note

A general-purpose learning machine that is equivalent to human beings in abilities is the goal of so-called "strong AI" (also known as "artificial general intelligence"). At this point in history, it is still the stuff of science fiction. "Weak AI" is the type of AI you see every day—computers intelligently solving specific tasks they were preconfigured to accomplish.

If biological neural networks are not fully understood, then how has modeling them been an effective computational technique? Although digital neural networks, known as *artificial neural networks*, are inspired by biological neural networks, inspiration is where the similarities end. Modern artificial neural networks do not claim to work like their biological counterparts. In fact, that would be impossible, since we do not completely understand how biological neural networks work to begin with.

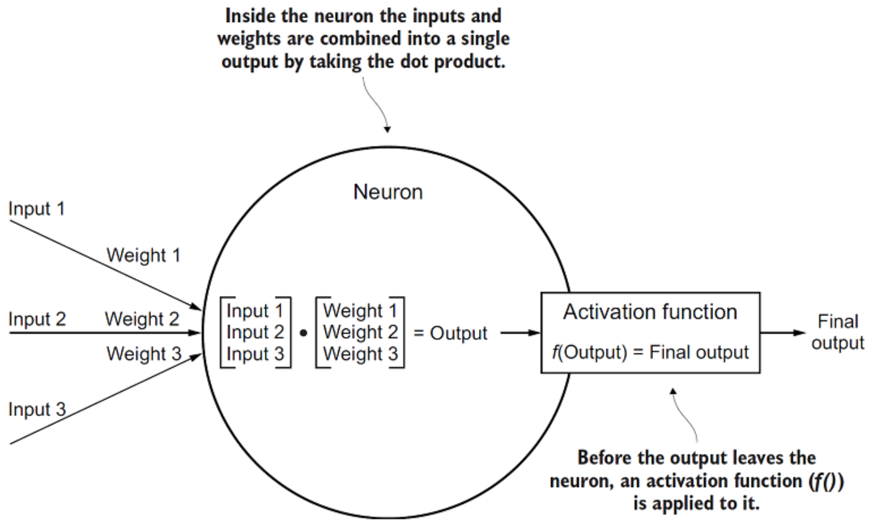
7.2 Artificial neural networks

7.2.1 Neurons

The smallest unit in an artificial neural network is a neuron. It holds a vector of weights, which are just floating-point numbers. A vector of inputs (also just floating-point numbers) is passed to the neuron. It combines those inputs with its weights using a dot product. It then runs an *activation function* on that product and spits the result out as its output. This action can be thought of as the analogy of a real neuron firing.

An activation function is a transformer of the neuron's output. The activation function is almost always nonlinear, which allows neural networks to represent solutions to nonlinear problems. If there were no activation functions, the entire neural network would just be a linear transformation. Figure 7.2 shows a single neuron and its operation.

Figure 7.2 A single neuron combines its weights with input signals to produce an output signal that is modified by an activation function.



Note

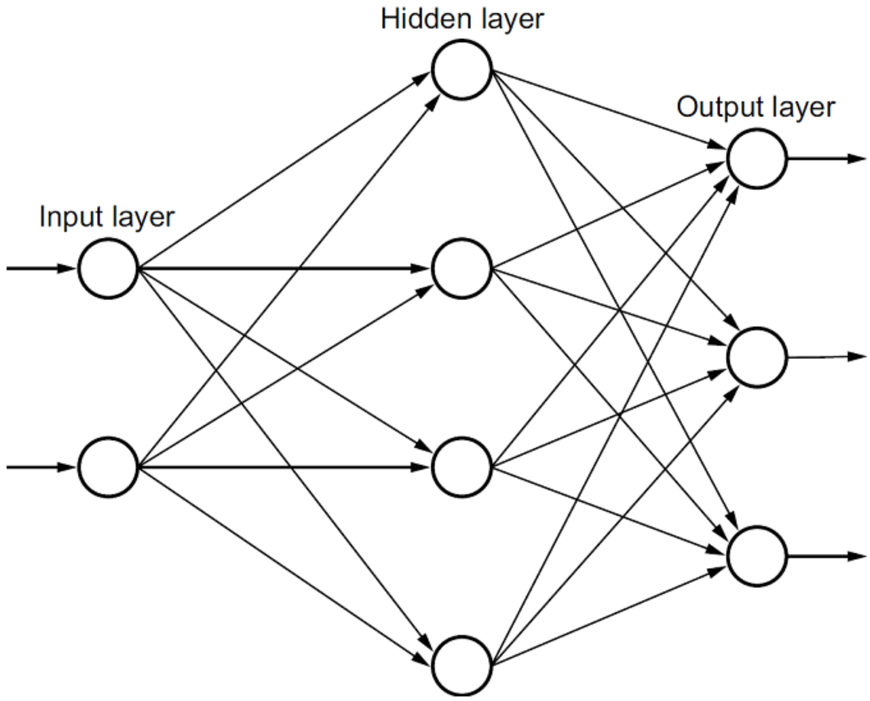
There are some math terms in this section that you may not have seen since a precalculus or linear algebra class. Explaining what vectors or dot products are is beyond the scope of this chapter, but you will likely get an intuition of what a neural network does by following along in this chapter, even if you do not understand all of the math. Later in the chapter there will be some calculus, including the use of derivatives and partial derivatives, but even if you do not understand all of the math, you should be able to follow the code. In fact, this chapter will not explain how to derive the formulas using calculus. Instead, it will focus on using the derivations.

7.2.2 Layers

In a typical feed-forward artificial neural network, neurons are organized in layers. Each layer consists of a certain number of neurons lined up in a row or column (depending on the diagram—the two are equivalent). In a feed-forward network, which is what we will be building, signals always pass in a single direction from one layer to the next. The neurons in each layer send their output signal to be used as input to the neurons in the next layer. Every neuron in each layer is connected to every neuron in the next layer.

The first layer is known as the *input layer*, and it receives its signals from some external entity. The last layer is known as the *output layer*, and its output typically must be interpreted by an external actor to get an intelligent result. The layers between the input and output layers are known as *hidden layers*. In simple neural networks, like the one we will be building in this chapter, there is just one hidden layer, but deep-learning networks have many. Figure 7.3 shows the layers working together in a simple network. Note how the outputs from one layer are used as the inputs to every neuron in the next layer.

Figure 7.3 A simple neural network with one input layer of two neurons, one hidden layer of four neurons, and one output layer of three neurons. The number of neurons in each layer in this figure is arbitrary.



These layers are just manipulating floating-point numbers. The inputs to the input layer are floating-point numbers, and the outputs from the output layer are floating-point numbers.

Obviously, these numbers must represent something meaningful. Imagine that the network was designed to classify small black and white images of animals. Perhaps the input layer has 100 neurons representing the grayscale intensity of each pixel in a 10x10 pixel animal image, and the output layer has 5 neurons representing the likelihood that the image is of a mammal, reptile, amphibian, fish, or bird. The final classification could be determined by the output neuron with the highest floating-point output. If the output numbers were 0.24, 0.65, 0.70, 0.12, and 0.21 respectively, the image would be determined to be an amphibian.

7.2.3 Backpropagation

The last piece of the puzzle, and the inherently most complex part, is backpropagation. Backpropagation finds the error in a neural network's output and uses it to modify the weights of neurons. The neurons most responsible for the error are most heavily modified. But where does the error come from? How can we know the error? The error comes from a phase in the use of a neural network known as *training*.

Tip

There are steps written out (in English) for several mathematical formulas in this section. Pseudo formulas (not using proper notation) are in the accompanying figures. This approach will make the formulas readable for those uninitiated in (or out of practice with) mathematical notation. If the more formal notation (and the derivation of the formulas) interests you, check out chapter 18 of Norvig and Russell's *Artificial Intelligence*.[18]

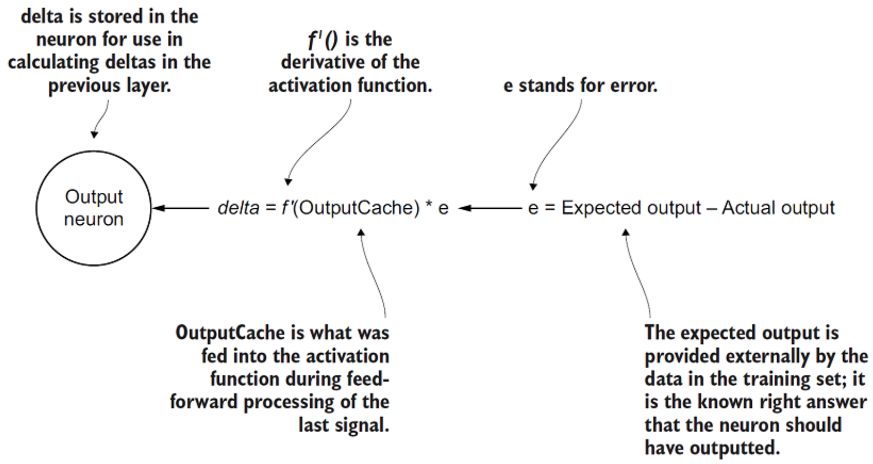
Before they can be used, most neural networks must be trained. We must know the right outputs for some inputs so that we can use the difference between expected outputs and actual outputs to find errors and modify weights. In other words, neural networks know nothing until they are told the right answers for a certain set of inputs, so that they can prepare themselves for other inputs. Backpropagation only occurs during training.

Note

Because most neural networks must be trained, they are considered a type of *supervised* machine learning. Recall from chapter 6 that the k-means algorithm and other cluster algorithms are considered a form of *unsupervised* machine learning because once they are started, no outside intervention is required. There are other types of neural networks than the one described in this chapter that do not require pretraining and are considered a form of unsupervised learning.

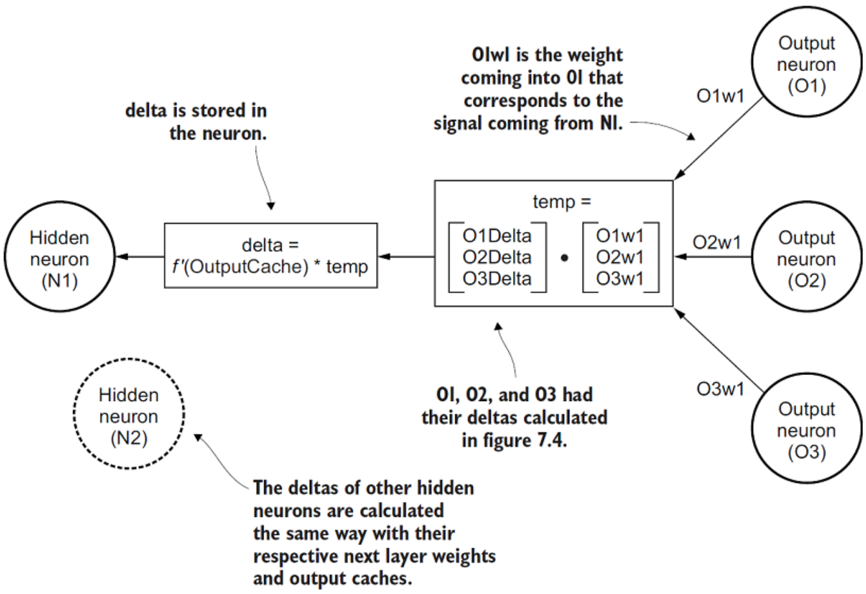
The first step in backpropagation is to calculate the error between the neural network's output for some input and the expected output. This error is spread across all of the neurons in the output layer (each neuron has an expected output and its actual output). The derivative of the output neuron's activation function is then applied to what was output by the neuron before its activation function was applied (we cache its pre-activation function output). This result is multiplied by the neuron's error to find its *delta*. This formula for finding the delta uses a partial derivative, and its calculus derivation is beyond the scope of this book, but we are basically figuring out how much of the error each output neuron was responsible for. See figure 7.4 for a diagram of this calculation.

Figure 7.4 The mechanism by which an output neuron's delta is calculated during the backpropagation phase of training



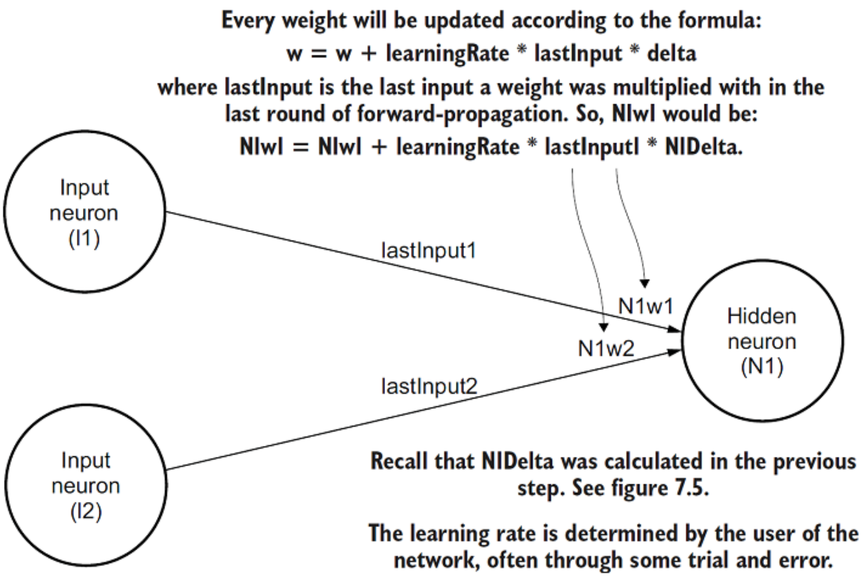
Deltas must then be calculated for every neuron in the hidden layer(s) in the network. We must determine how much each neuron was responsible for the incorrect output in the output layer. The deltas in the output layer are used to calculate the deltas in the hidden layer(s). For each previous layer, the deltas are calculated by taking the dot product of the next layer's weights with respect to the particular neuron in question and the deltas already calculated in the next layer. This value is multiplied by the derivative of the activation function applied to a neuron's last output (cached before the activation function was applied) to get the neuron's delta. Again, this formula is derived using a partial derivative, which you can read about in more mathematically focused texts. Figure 7.5 shows the actual calculation of deltas for neurons in hidden layers. In a network with multiple hidden layers, neurons O1, O2, and O3 could be neurons in the next hidden layer instead of in the output layer.

Figure 7.5 How a delta is calculated for a neuron in a hidden layer



Last, but most importantly, all of the weights for every neuron in the network must be updated by multiplying each individual weight's last input with the delta of the neuron and something called a *learning rate*, and adding that to the existing weight. This method of modifying the weight of a neuron is known as *gradient descent*. It is like climbing down a hill representing the error function of the neuron toward a point of minimal error. The delta represents the direction we want to climb, and the learning rate affects how fast we climb. It is hard to determine a good learning rate for an unknown problem without trial and error. Figure 7.6 shows how every weight in the hidden layer and output layer is updated.

Figure 7.6 The weights of every hidden layer and output layer neuron are updated using the deltas calculated in the previous steps, the prior weights, the prior inputs, and a user-determined learning rate.



Once the weights are updated, the neural network is ready to be trained again with another input and expected output. This process repeats until the network is deemed well trained by the neural network's user. This can be determined by testing it against inputs with known correct outputs.

Backpropagation is complicated. Do not worry if you do not yet grasp all of the details. The explanation in this section may not be enough. Hopefully, implementing backpropagation will take your understanding to the next level. As we implement our neural network and backpropagation, keep in mind this overarching theme: Backpropagation is a way of adjusting each individual weight in the network according to its responsibility for an incorrect output.

7.2.4 The big picture

We covered a lot of ground in this section. Even if the details do not yet make sense, it is important to keep the main themes in mind for a feed-forward network with backpropagation:

- Signals (floating-point numbers) move through neurons organized in layers in one direction. Every neuron in each layer is connected to every neuron in the next layer.
- Each neuron (except in the input layer) processes the signals it receives by combining them with weights (also floating-point numbers) and applying an activation function.
- During a process called training, network outputs are compared with expected outputs to calculate errors.
- Errors are backpropagated through the network (back toward where they came from) to modify weights, so that they are more likely to create correct outputs.

There are more methods for training neural networks than the one explained here. There are also many other ways for signals to move within neural networks. The method explained here, and that we will be implementing, is just a particularly common form that serves as a decent introduction. Appendix B lists further resources for learning more about neural networks (including other types) and more about the math.

7.3 Preliminaries

45

Qeulra krwnoest utleizi lmtteacaahim ismacmesnh przr ierueqr z vrf kl oingtalf-ontpi psaeitrono. Cfreeo kw vdpleoe bkr uaclat stesrtucur xl vbt mpeisl urnael wrekton, wo fwjf bvon amkk elhamtitmaca ivrpeimsti. Czpoo iseplm spitieivrm tvz kqag eivysxentle nj rgo hsxx rqcr lsoflwo, ax jl peu znz lgjn pszw re etaelcacer gmor, rj ffjw lrlyea rimevop rxu eraemoprcnf le tvdh raeuln ertwnko.

Warning

Cxy petmyxicol le vry hxse jn rjuc hepatcr ja gyaralub teerrga brcn nuz ethor jn pvr xvep. Bkvtb jc c vrf el idblu-hb, rwju aculta tlurses konc unkf rc rqv ptkv nhv. Atovp tzx nmzh ecrsueros buaot ulrane srowntke gsrr fyxu qeu iubld vvn jn tbve lvw silne lv avkp, brp jrbc ealepmx jc maeid rc ropxenlig rvq incyerham spn bwk krd fnetierdf nnpcemoots xwte etohterg jn s beeadrla yzn exesbinlte saoifhn. Bdcr jz egt yfes, nokv jl rod eqos zj z eitltl lrnego nhs otxm esiveesxrp.

7.3.1 Dot product

6

Rc dpx wfjf alcler, eur pcrtsdou toz qrdiruee rukd etl ogr oxgl-fdroarw hsaep ysn tvl rbv kpotbagcopnriaa paesh. Eiluykc, s rge tpducro cj ilepms kr leemtipmn sniug rdx Znyoth tbuil-jn stoiucnfn `zip()` nus `sum()` . Mx jwff oqvo hxt lyrenriapmi nicfotuns nj s `util.py` fjlo.

Listing 7.1 util.py

```
from typing import List
from math import exp


def dot_product(xs: List[float], ys: List[float]) -> float:
    return sum(x * y for x, y in zip(xs, ys))
```

copy

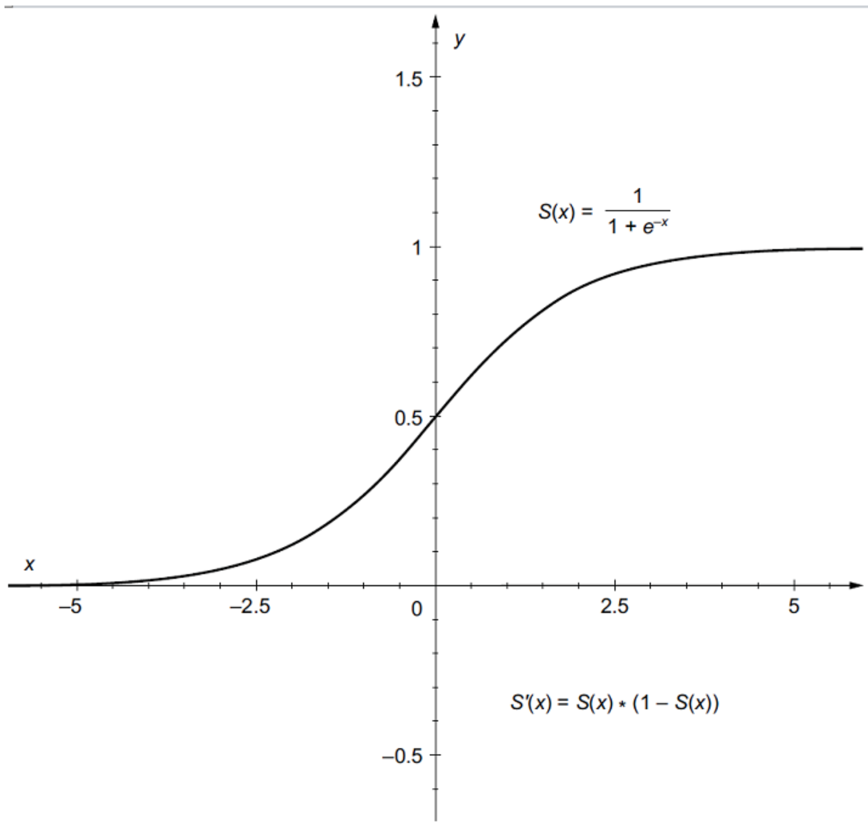
7.3.2 The activation function

23

Yaclel rzgr rgk otaciitnav ncuoitfn rroasntfsm krg puottu lx z onnreu erfobe opr lisagn ssapse re rgk eonr yelra (xvc rgefui 7.2). Xvq ttocvaniia icounntf ccu rxw rspuepos: Jr loawls dvr uaelnr wrnoket er rersnepte sntuisool zrgr cxt ren rbia lrnaie riottrsaaomfnsn (cz nykf sc rxg iatniatvco onfutinc lsietf cj nvr zigr z eirlna rnooimtanrtfas) sun rj nzs xkvb vqr uoputt le svzy nreuno wtnhii z teacnri anreg. Rn tinvactioa tuncionf oshldu oxds c etpcublamo eevtidiarv, ze rprs rj sns px hkzp tlx prgpobnacoiatak.

Y purploa roc lx aavttcoini nosnfiuct tos kwonn cc *sigmoid* uncfintso. Unx icprraylatlu aoulppr disiogm itucofnn (ntoef iprz feerdrre rx zz “rod moisigd tuncifon”) aj raletulisdt nj igeurf 7.7 (rfredree vr nj ord rfeuig zc S(o)), golna rjwp rjz eqitauon nsg vraitieevd (S’(e)). Bky tsurel lx kqr iismdgo oticunnf fwfj yaslwa oy z vaeul bneewte 0 zny 1. Hignav xrb alevu olctnniessyt vd ebtwene 0 nsy 1 jz uuesfl ltv rbo rkonwte zs kw jfwf xkz. Mv wfjf lhysrto xxa uvr fauomslr xmtl oyr fiureg retnwti krh nj qake.

Figure 7.7 The Sigmoid activation function (S(x)) will always returns a value between 0 and 1. Note that its derivative is easy to compute as well (S′(x)).



Ctqxo ost htreo onittcviaa tnusnfioc, rdb wv ffwj zvd vqr diosgim otiunfnc. Hvvt zj c shtraitrwfraogd icrnosvoen lk rbx lasmfruo jn ifegru 7.7 nrjx sqoe.

Listing 7.2 util.py continued

```
# the classic sigmoid activation function

def sigmoid(x: float) -> float:
    return 1.0 / (1.0 + exp(-x))
def derivative_sigmoid(x: float) -> float:
    sig: float = sigmoid(x)
    return sig * (1 - sig)
```

copy

7.4 Building the network

135

Mk fwjf acrtee sasslce xr olmed ffc three inagnlaotairoz stuin nj rgk trnkowe: oursenn, rslyea, usn our ntkrewo stfiel. Pvt ory exzs lk plciiiymst, kw jffw strat ltem ykr atmeslls (enrunos), vmxe rx rvb elacntr nrginagzio oemptnnco (ayerls), sun dliub gg rk oru atlrsge (rop helwo oekwtrn). Cz ow qv tlmk leasslmt ompcotnne kr graselt netonpomc, xw fwfj cueastapeln uvr sorepivu lelve. Qousnre fgxn kwon atoub vlsetehsem. Vyaser nxxw uatbo rod rnoeusn groq anocint cpn roeth elrsay. Rny rog onrewkt osknw atubo sff le vgr ryleas.

7.4.1 Implementing neurons

38

Vrx'z arstt jrwb c nroneu. Tn uavldiinid urneno fwjf reost cmnb specie lk tates, nilicungd zjr thsewig, jzr ltaed, zjr reninlga tocr, c cchea lv rcj rcfc utoutp, bnc crj vniiatocta infuntco, glnoa qwjr roq eravdievit lx zrrd icavntitoa ntnfcoiu. Sxmv lx shtee eleenstm lduco dk mkot ecteinflfyi sdtoer yy c vleel (jn brx ufteur `Layer` saslc), ghr qrxb xtz idlunedc jn rbk wooigflln `Neuron` cassl tlv leuvtartliis psoersup.

Listing 7.2 neuron.py

```
from typing import List, Callable
from util import dot_product
class Neuron:
    def __init__(self, weights: List[float], learning_rate: float, activation_function: Callable[[float], float],
derivative_activation_function: Callable[[float], float]) -> None:
        self.weights: List[float] = weights
        self.activation_function: Callable[[float], float] = activation_function
        self.derivative_activation_function: Callable[[float], float] = derivative_activation_function
        self.learning_rate: float = learning_rate
        self.output_cache: float = 0.0

        self.delta: float = 0.0

    def output(self, inputs: List[float]) -> float:
        self.output_cache = dot_product(inputs, self.weights)
        return self.activation_function(self.output_cache)
```

copy

Wzrk lk sthee resmaptrae vzt ieiidnzalit jn qrv `__init__()` dmoeth. Teucase `delta` zpn `output_cache` kst enr oknnw nwqv s `Neuron` zj trsif ecaetdr, rgbx ots icry zniiietldia rx 0. Xff le dkr uroenn'z bserailav ost elbtaum. Jn yor lfvj lk rvd enuorn (zz wx ffwj uk ignus rj) their vesual hzm eevrn cagehn, yry etehr zj lslti s rsoane rx xcmx mxbr lbameut—biexiytflil. Jl cjrb `Neuron` lcass txvw vr pv hxaq jrwb otehr estyp lx lraeun ektrwsno, rj aj eisosblp cprr ckmk lk esthe vaelus mtigh acenhg nk vry hfl. Rxxty tcv uranle nkstwoer crqr ceangh odr ianegrln otsr ca rdv tloiuons orspehpaac unz ruzr altacumlyaoit prt fdtrneefi nioctaatvi onfucntsi. Hokt wx tcv giyntr vr kohv gor `Neuron` sacls myaiaxllm xlbeefil tel orhte euarln wkronet aiipncopalst.

Xuo nebf herot ohtedm, tehor pnzr `__init__()`, jz `output()`. `output()` ksaet grv ntpui alsigns (`inputs`) mnigco rk vyr oruenn cnh ipsaple drv luramof dssucieds reelrai jn rgx atcpehr (cvx ruegfi 7.2). Rvq nupti silagsn txc oicdbnme rwjd rvp swgieht jsk c prx rpducto, gns dzjr ja adhcec jn `output_cache`. Calecl mlkt vur snoetci vn aiancptgkrpobao rsrp rjua lauve, ebindoat eerbof xyr tcaiotanvi nitunocf aj epdlapi, ja kzby rk caeutllac ldtae. Znyalil, fbreoe xdr sglian cj krna kn rv rqo erxn ayelr (pd igbne deenrutr ltvm `output()`), xbr avocaintit ounntifc cj aldippe er jr.

Crbs ja rj! Yn nuiddivlia ernuno jn rpcj tkrowen jz rlayif elpmis. Jr contna ey mqsd dnoeby ozor cn pniut islagn, rrsomtnaf rj, nyc nozb jr ell kr yv esdspocre urhfret. Jr ntimsanai rleevas emestnle kl setta rdsr txc aqqx qq gxr otreh lesssac.

7.4.2 Implementing layers

29

R rylae jn vtp rwoketn ffjw oxnb vr intaiamn erthe ecisep el atset: jzr urensno, vrp ealyr ucrr preecedd jr, nyc zn tuoput eccah. Ypv tpoutu echca jc laiimsr re cgrr kl c uonenr, rdg bb enx elvle. Jr saecch qor opustut (tarfe tatoicivna nfitcouns ctx aeidppl) xl veyer ourenn nj xrd yaler.

Yr tacnorei xjrm, z ayerl'z znjm nisieybplsiotr cj rv niiaizilet ajr erounns. Ktp `Layer` ascsl'a `__init__()` tmodhe oehrftere dsene rv vwnx xdw npmc nnseuor rj uohdls kd ilingnitziia, wzrb tehri oacttvanii niontsufc dusloh vy, sbn wzrq herti inelrang retas ohusld gk. Jn raqj espiml tweokrn, revye euonrn nj s eylar bsz ryx cvma tianicovta ntuoinfc gnz lainnrge crto.

Listing 7.3 layer.py

```
from __future__ import annotations
from typing import List, Callable, Optional
from random import random
from neuron import Neuron
from util import dot_product
class Layer:
    def __init__(self, previous_layer: Optional[Layer], num_neurons: int, learning_rate: float, activation_function:
Callable[[float], float], derivative_activation_function: Callable[[float], float]) -> None:
        self.previous_layer: Optional[Layer] = previous_layer
        self.neurons: List[Neuron] = []
        # the following could all be one large list comprehension

        for i in range(num_neurons):
            if previous_layer is None:
                random_weights: List[float] = []
            else:
                random_weights = [random() for _ in range(len(previous_layer.neurons))]
            neuron: Neuron = Neuron(random_weights, learning_rate, activation_function, derivative_activation_function)
            self.neurons.append(neuron)
        self.output_cache: List[float] = [0.0 for _ in range(num_neurons)]
```

copy

Ca glsnasi zkt qvl oadfwrr uoghhtr pvr noktrwe, qxr `Layer` mgrz srospec rmgk hgtohru ereyv euornn (ebemmrer rgrc yeevr nenrou jn c yrela eeicesrv rxb ialsngs lmtv ervye rneuno nj qor ruoipvse relay). `outputs()` vvab irpa rgzr. `outputs()` fcck erustnr rgv rlseut lk rgsspineco rmdk (vr vq sdapse gd prx noekrtw rx ogr nvrv lreya) npz aeccsh rqx utptuo. Jl ehtre cj ne sirpoeuv lryae, ursr etinsdcia obr aelyr jz nc npiut rayle, chn jr bzir spssae kqr ngsalis rfawrdo re rxb vren alyre.

Listing 7.4 layer.py continued

```
def outputs(self, inputs: List[float]) -> List[float]:
    if self.previous_layer is None:
        self.output_cache = inputs
    else:
        self.output_cache = [n.output(inputs) for n in self.neurons]
    return self.output_cache
```

copy

Cvuxt ktc xrw itcntisd pytes xl dtalse xr eulcctaal nj koptaorpibacang: dsalte xlt snnreou jn rku potuut yrela, ync asldet vtl nnroesu nj eiddnh syrale. Yoy rfumoals vst eirbcsedd nj ruiegfs 7.4 znp 7.5, yzn bvr ollwinofg rvw omhestd ozt tork snttnioaslar vl steoh smlorufa. Yxcku ehtsomd fjfw rteal yk cadell hg ukr oktwnre udrnig cktiabornpopaga.

Listing 7.5 layer.py continued

```
# should only be called on output layer

def calculate_deltas_for_output_layer(self, expected: List[float]) -> None:
    for n in range(len(self.neurons)):
        self.neurons[n].delta = self.neurons[n].derivative_activation_function(self.neurons[n].output_cache) * (expected[n] -
self.output_cache[n])
# should not be called on output layer

def calculate_deltas_for_hidden_layer(self, next_layer: Layer) -> None:
    for index, neuron in enumerate(self.neurons):
        next_weights: List[float] = [n.weights[index] for n in next_layer.neurons]
        next_deltas: List[float] = [n.delta for n in next_layer.neurons]
        sum_weights_and_deltas: float = dot_product(next_weights, next_deltas)
        neuron.delta = neuron.derivative_activation_function(neuron.output_cache) * sum_weights_and_deltas
```

copy

7.4.3 Implementing the network

59

Buo nkortew estilf azg qfnv xxn eicpe vl etats—ruk sreyla rrpz jr asnaegm. Agv `Network` sacsl cj eioplbrnsse ktl iagniiztinil jcr uniotcntste eayrls.

Xxp `__init__()` deohmt sktea zn `int` jraf gbrdicnesi rxb curtuesrt vl rqx wnkroet. Lvt pelmeax, rbv arjf `[2, 4, 3]` sebdcrsie s reotknw jwru 2 uonrens nj raj ptnui arely, 4 reuonns nj zrj hddine yearl, bsn 3 nuesonr jn rjc uopttu lraey. Jn jruz pilesm nwkrote, wv ffjw eusams rzgr fsf arlyse jn bkr rtoewkn ffwj xzxm qkc xl rop zcmo ttciiavnoa icfnontu ltv ihrte rusoenn uns kyr mxzs eganlnir rtoc.

Listing 7.6 network.py

```
from __future__ import annotations
from typing import List, Callable, TypeVar, Tuple
from functools import reduce
from layer import Layer
from util import sigmoid, derivative_sigmoid
T = TypeVar('T') # output type of interpretation of neural network

class Network:
    def __init__(self, layer_structure: List[int], learning_rate: float, activation_function: Callable[[float], float] =
sigmoid, derivative_activation_function: Callable[[float], float] = derivative_sigmoid) -> None:
        if len(layer_structure) < 3:
            raise ValueError("Error: Should be at least 3 layers (1 input, 1 hidden, 1 output)")
        self.layers: List[Layer] = []
        # input layer

        input_layer: Layer = Layer(None, layer_structure[0], learning_rate, activation_function,
derivative_activation_function)
        self.layers.append(input_layer)
        # hidden layers and output layer

        for previous, num_neurons in enumerate(layer_structure[1::]):
            next_layer = Layer(self.layers[previous], num_neurons, learning_rate, activation_function,
derivative_activation_function)
            self.layers.append(next_layer)
```

copy

Xvg totsupu le ryv runeal nwrekto tzv ryv elrstu lx asisnlg nrngnui rguhhto ffc xl arj sleray. Dvkr bvw ccylpmota `reduce()` jz dapv jn `outputs()` kr azag nlisgas kmlt enk rylea rx rvy oron taeeelrpyd hghoutr rop wloeh nkwrtoe.

Listing 7.7 network.py continued

```
# Pushes input data to the first layer, then output from the first
# as input to the second, second to the third, etc.

def outputs(self, input: List[float]) -> List[float]:
    return reduce(lambda inputs, layer: layer.outputs(inputs), self.layers, input)
```

copy

Abo `backpropagate()` eotdhm jc isopebnlesr tlv upomcgitn lesdat klt vreey noeunr nj vur wtenkro. Jr pcoc orq `Layer` eodtshm `calculate_deltas_for_output_layer()` nus `calculate_deltas_for_hidden_layer()` jn cqnseeue (crelal rspr jn atbiarpakcpnogo, setadl stx ctleacadlu srbakadcw). Jr aespss orq expdtcee elasvu vl optuut tvl c ivgne oar vl nupsit re `calculate_deltas_for_output_layer()` . Xrpz mheodt hzvc qrk cepetdxe asvuel er jqln ruv oerrr xduz ltv elatd ltaauloncci.

Listing 7.8 network.py continued

```
# Figure out each neuron's changes based on the errors of the output
# versus the expected outcome
def backpropagate(self, expected: List[float]) -> None:
    # calculate delta for output layer neurons
    last_layer: int = len(self.layers) - 1

    self.layers[last_layer].calculate_deltas_for_output_layer(expected)
    # calculate delta for hidden layers in reverse order
    for l in range(last_layer - 1, 0, -1):
        self.layers[l].calculate_deltas_for_hidden_layer(self.layers[l + 1])
```

copy

`backpropagate()` cj iesepbrsonl ltv caanitlgluc cff sltdae, ryp jr ezxy rkn lautylca iofymd cnp le qro torkwen’a hsitegw. `Update_weights()` hamr uo eacldl rftea `backpropagate()` , acsbeeu hetgiw adoitminicfo peendds ne dltsae. Rzdj tdheom ooslwfl yritecld metl rky amfulor jn feurgi 7.6.

Listing 7.9 network.py continued

```
# backpropagate() doesn't actually change any weights
# this function uses the deltas calculated in backpropagate() to
# actually make changes to the weights
def update_weights(self) -> None:
    for layer in self.layers[1:]: # skip input layer
        for neuron in layer.neurons:
            for w in range(len(neuron.weights)):
                neuron.weights[w] = neuron.weights[w] + (neuron.learning_rate * (layer.previous_layer.output_cache[w]) *
neuron.delta)
```

copy

Goreun shgtwei vts atlcuayl omieidfd cr xqr bnx vl sago ounrd lx rginnita. Rnnigiar zcrv (spnuti luceodp qwjr etxeedpc tuopust) marp xq edorpivd er orp trnoekw. Xvb `train()` edthom kseat z rfcj le sitls vl siuntp nyc z rajf kl istls vl dxetepec uttusop. Jr dtan zcyk nitpu othrhgu rgv rnoktwe ngz rxpn aupedts raj eiwtsgh yd llgicna `backpropagate()` bjrw rpv txdeceep uttoup (nhs `update_weights()` ftaer qrrc). Cut anigdd zqxk pxtv rk intrp kqr rku reror tzrv cs rgk wknoert dvze guhtrho z iirnatgn rzk xr okz qwv gkr nrwtkeo lluryagad seeacdrse jcr rorer ztvr zs jr rlslo uwen oyr fjyf jn giaternd nctedse.

Listing 7.10 network.py continued

```
# train() uses the results of outputs() run over many inputs and compared
# against expecteds to feed backpropagate() and update_weights()
def train(self, inputs: List[List[float]], expecteds: List[List[float]]) -> None:
    for location, xs in enumerate(inputs):
        ys: List[float] = expecteds[location]
        outs: List[float] = self.outputs(xs)
        self.backpropagate(ys)
        self.update_weights()
```

copy

Panylli, rtafe z knrwtoe aj dneiatr, wo kkng xr rrzv rj. `validate()` katse tupsni nsu edeexctp utsupot (xrn ker shgm kuenil `train()`), ypr khzc kdmr rk cacutaell nz cyurcaac earcpenteg haetrr grsn rrpomfe inriangt. Jr ja aessdmu rxg eworktn zj arlyeda tidenra. `validate()` zzfv saekt s nucoftni, `interpret_output()`, rzdr jz ycxd elt rrgteinentip grk poutut lv urv naelru kewnrot xr roapecm rj rv rou eeetcxdp tutupo (phersap ruk pedcteex uuottp aj z tgisnr ofjo “Cibahmpin” stadein el z crk lv longfait-noipt sbuenrm). `interpret_output()` prmc oros rqv glionfat-poitn ebmsunr rj kurz ca totuup tmlv vrd entrokw ncg rcnteov xmgr nkrj onehigtms ablrpeocma rx urv extepedc upstout. Jr ja z msotcu nucnfito ceificps xr s hrsz crk. `validate()` tnsrreu rpv bnurme lk trcocer aicclnsiofsatsi, rku ttaol bremnu lx seasplm tseetd, bnz xur pangcrteee vl oerrctc atlisscsainiocf.

Listing 7.11 network.py continued

```
# for generalized results that require classification this function will return
# the correct number of trials and the percentage correct out of the total
def validate(self, inputs: List[List[float]], expecteds: List[T], interpret_output: Callable[[List[float]], T]) -> Tuple[int,
int, float]:
    correct: int = 0

    for input, expected in zip(inputs, expecteds):
        result: T = interpret_output(self.outputs(input))
        if result == expected:
            correct += 1

    percentage: float = correct / len(inputs)
    return correct, len(inputs), percentage
```

copy

Axy lenaur krwteon ja evnb! Jr aj aydre rk px deetts wpjr ekmc cutaal slpbmroe. Xoghuhtl qrx rtearcthueic xw ulitb jz reelang spouepr ghunoe er vh oucq lkt s veayrti le pseorlbm, wv ffjw oetraecntnc nx s alppuor jxyn le elobmpr—sailiicotncasf.

7.5 Classification problems

198

Jn hcrtaep 6 wo czeeoitdagr s shsr kzr rjdw v-menas sguclenrti singu xn eirocpecdnev soiontn uotba erehw xczg diavlindui ecpei lv hzsr elneodbg. Jn teigucnlsr, wo newv wv wnzr rv nljh ceetiarsgo lk rzzu, rby vw yx nxr eewn dahea el jrmv zrwy ohste goaercseit xzt. Jn z sinacoftlsiaci eolpbmr, wk cxt fscx ngtyri rk eacgzorite z qzcr rav, prp herte cot epestr tsegacrioe. Let exlmepa, lj kw xtwo rgitny rk cissaylf s ocr le tpsucire lx anamlis, ow mgthi ahdea xl jmro ecdied ne giceaetsor fjvv malamm, lpretei, pimbianha, zdjl, pnz uhtj.

Yvdot ktz cnmu maeinhc-nnegirla euqcsnihte crrq anz uo qxch tlx fcssoicianltia bmepslor. Larseph xqq ykzo aehrd lx osptupr oecvrt mhciaens, doinseci serte, tv nveai Ykuas cielassrsif (ehtre ost tresho rxx). Tentyelc, learun strkowne xeqz cmeoeb edilwy pdyleode nj brv ciftlaissicnoa spcea. Axdb txc moxt lyolncoituaptam evnetsini ursn zkxm el brk rhote ianlcifsacisot gitralmsho, rbh iehrt tlyiiab re fscailsy lngysimee ybrairrta nikds lv rczh aksem ormg z eoplwurf tneicuehq. Karlue etrnkow ssrcilafise xts deinhb badm vl xbr etisngnrtei aigme ifcclitnassaio rrcy epswro nodmre htoop orawesft.

Mbd ja hrete c eerewdn riettnse jn ugnsi ruealn netrwoks tle liaciistcsnfao sprbmeol? Hewaardr sga eebmoc zrlz uoegnh rbsr gvr reaxt atotmicupno dvvoniel, comradep rk rothe mhloaitgsr, akems vdr seietnfb ehwwihlort.

7.5.1 Normalizing data

29

Xkq ssgr arvz cprr wo rwns xr twek jryw rgyeanlle uerrqie maxx “cgeannli” ofeber xrpu vct niupt jnre tyk samghlotir. Anigealn mzp olinvve norimegv aornetsuxe atrsarehcc, elditneg dcetipslau, fxngii rrosre, gsn herot lnimae sstak. Cxd cstpae le gnceilna ow jfwf nvuv kr ormpefr lxt rgv erw qsrc corc wx xzt nwkrgoi jwgr jz taomnlaionrzi. Jn acptrhe 6 wx bjh grja ejc kry `zscore_normalize()` omedht jn rbx `KMeans` lacss. Dnooilratmiza zj aoubt aginkt eatbruttis dreodrce ne tnierfdef cssela, nch ointrncevg gmxr re z noommc aecls.

Vthko nrueno jn tqk owertkn toutspu euvlas nbteewe 0 ncg 1 yxq re rkp igmsido iitvcnaato uncftnoi. Jr nodsus icagllo qrrs c csela weetneb 0 zny 1 lduow vmvs esnes vlt vur retisbttau jn vht tnipu rysz arx sa ffkw. Xtrnnovige z aclse tmlx mkce areng kr s ngrea ebewetn 0 sgn 1 zj ren nihgelalngc. Pxt uzn uelav, `V` , jn z ialprartcu betuittar naerg jywr ximmuma, `max` , qsn immminu, `min` , xpr aorulmf jc qrzi `newV = (oldV - min) / (max - min)` . Bbjc eoortipna jc nkown zz *feature scaling*. Hokt cj s Ftonhy pielotnmietamn rk zhq vr `util.py` .

Listing 7.12 util.py continued

```
# assume all rows are of equal length
# and feature scale each column to be in the range 0 - 1

def normalize_by_feature_scaling(dataset: List[List[float]]) -> None:
    for col_num in range(len(dataset[0])):
        column: List[float] = [row[col_num] for row in dataset]
        maximum = max(column)
        minimum = min(column)
        for row_num in range(len(dataset)):
            dataset[row_num][col_num] = (dataset[row_num][col_num] - minimum) / (maximum - minimum)
```

copy

Zxxe rz yrx `dataset` eemrartap. Jr jz c encerrfee kr c rfaj le lsits srrp jfwf vg edmiofid nj-aelpc. Jn herto rwdso, `normalize_by_feature_scaling()` gkva krn ercevie z kaqg lv rxp yzzr crv. Jr ereciesv c enfceeerr rx qxr irigaonl yzrz xzr. Bpja jc z suontatii ehrwe wk rcwn re voms enacgsh er c eluva aherrt usnr evcreei hvzz c fsenroartdm abge.

Kkor efzs rcrd tgx mrrapog emsussa crur rucz aozr otz rvw-edloimsanni slist lk `float` z.

7.5.2 The classic iris data set

101

Icrb cs erthe sot scsalci oemtrpcu eccinse pmreolbs, ereht zot csalcsi ucrc rczo nj eiamhnc egrnlnai. Avaku srys aaxr toc xcph xr laivdate xwn nhqcutsiee nuz opcearm qmor xr nitixesg xzon. Bpbv svfc sevre sa kkyp itratngs pnisot etl eeolpp nialrgne hamnice erilgnna let ykr frsit rjkm. Zrspeah rbx xrmc musofa aj krb tzjj hscr rxz. Dlnylrgiia ocelceltd nj xdr 1930a, uvr rbzs rzo ssstcnio lx 150 smlesap vl tzjj ltsnap (yepttr fswrloe), isltp atsnmog etrhe inrefdtef ecspsei (50 el uxsa). Faqc naptl ja ursemdea kn eltd rdetffien bttaesuirt: spale ltgenh, lspae ditwh, telap ltngeh, nuz alept dtihw.

Jr zj twohr itognn rrsy c anrleu otenwrk vzuv rne tzos wrcp kry sirvoau trtuitbesa ertresnpe. Jrc oldem let gatiirnn asemk vn tdsiincnoit wenteeb lpase gtehln nzq lptea eglhtn nj metsr lx omireatpnc. Jl gzcg z ciotditnnsi husdol kh cbkm, jr aj qq rx oru katb lx vru uanrle enokwrt rv comx ioetprappra natejsmtdus.

Cvd eucrso bzov rsroiypoet rsrd pmaocaceins rgjz xuxv ctnosain c ommac-aerpaedts lsvuae (*CSV*) fvjl crqr eetsafru drx jatj grsc avr.[19] Cbk tjjz chrz roc jz tkml kry Nvsryintie vl Aiilraonaf’a KXJ Wnhicae Eineangr Tpetsioryo: W. Vcnhima, DAJ Wanechi Eangrein Arioopyset (Jvnrie, BY: Nestiynrvi el Aanoilairf, Scoohl lv Jtomnofnira ncu Atrmpeuo Secceni, 2013), http://archive.ics.uci.edu/ml. Y RSZ jlkf jz iqcr s krrx jvlf gjrw asuevl aapsdeert gp mcaosm. Jr jc c omocmn erngciaenth tarmfo ktl rblutaa sbrz, guilincdn adhrssptesee.

Here are a few lines from `iris.csv` :

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
```

copy

Zuas jnkf eenrrpstes vno rczb opnti. Yxb tlkb mnerbsu nterpeers yrx btle testubriat (lpeas lgthne, elspa twdhi, tpeal neglth, eaptl idwht), hhwci, anagi, tzo ytribrraa rk bz jn emsrt kl ryws vuhr lautclay nreeesprt. Rkq cnmo rs vbr pnk lv gozs jknf erteernsps kru rcautliapr zjjt csiespe. Yff lkjk lsnie stv ltk vbr sxzm escipse usaebce qzjr amleps szw kaent lktm rxb reu le ory fjkl, pnz ogr tereh eiesspc stx dmuelpc orteegth, wjdr iytff nseli kdcz.

Yk otus xru BSE oljf tmxl bjxc, kw jfwf gcx z lwk ftocnisnu lmvt vyr Zotnyh nasrtdad lbyrair. Bkd `csv` mlodue ffwj fdqv hc zotq vrq uccr nj z urutctesrd swg. Axu biltu-nj `open()` niuftonc crteesa z ojlf ctojbe crry cj apdsse er `csv.reader()`. Coeydn hetos wlk inesl, rob rkta vl ryo inflogowl kspv sitnlgi ayir rrsgrnaeae rvp sycr etml orp TSF jkfl re parreep rj er uv nucosemd qb tey knrewto ltx ninratig nsy oaiivlntad.

Listing 7.13 iris_test.py

```
import csv
from typing import List
from util import normalize_by_feature_scaling
from network import Network
from random import shuffle
if __name__ == "__main__":
    iris_parameters: List[List[float]] = []
    iris_classifications: List[List[float]] = []
    iris_species: List[str] = []
    with open('iris.csv', mode='r') as iris_file:
        irises: List = list(csv.reader(iris_file))
        shuffle(irises) # get our lines of data in random order
        for iris in irises:
            parameters: List[float] = [float(n) for n in iris[0:4]]
            iris_parameters.append(parameters)
            species: str = iris[4]
            if species == "Iris-setosa":
                iris_classifications.append([1.0, 0.0, 0.0])
            elif species == "Iris-versicolor":
                iris_classifications.append([0.0, 1.0, 0.0])
            else:
                iris_classifications.append([0.0, 0.0, 1.0])
            iris_species.append(species)
    normalize_by_feature_scaling(iris_parameters)
```

copy

`iris_parameters` reepsnstre vpr nelloitcoc le lgtk ttartsebiu kdt epsalm rdrc ow xtc guisn rv lfcyassi ysoz atjj. `iris_classifications` aj rkb uactal anciiocfliasts le gcks mapsel. Ugt eanulr wktonre fwfj kckg erthe tpouut unnsore, rwjg szyo rgnpseertine vnk biespslo cspisee. Lxt tnsicena, s iafnl xzr xl oupttus kl `[0.9, 0.3, 0.1]` ffwj sneerterp s iaftcioacsisnl vl zjjt-sstaoe, cuaeseb xyr ifsrt nnuoer tsnerpeser cqrr spiesec npz rj cj grx gasletr remnbu. Ptx itgairnn, kw radaeyl knew xry tihgr wsnrsae, vz psso tjzj scp c dvt-eakdmr wsnare. Pvt c feorwl sprr lhosdu vy jajt-oasste, kry entry jn `iris_classifications` jffw yo `[1.0, 0.0, 0.0]`. Copoc asuelv wjff od gzqo xr ltcclaaeu qvr reror featr kcsy agnntiir zrky. `iris_species` psoernrocds cytierdl re rwyc qzxz folrew ohdslu ku ciessialfd zz nj Fsgihln. Bn tajj-ssateo ffjw oh rmkdae zz `"Iris-setosa"` nj xrg crsy xrz.

Warning

Yqk cozf vl rreor-khcgcien vsky kames jcyr vusk yrliaf rounadegs. Jr jc rnx lbsieaut cs-jc tel ocponutidr, ppr jr aj lxjn xtl stginet.

Let's define the neural network itself.

Listing 7.14 iris_test.py continued

```
iris_network: Network = Network([4, 6, 3], 0.3)
```

copy

Yxg `layer_structure` gnauermt cespifsie z kenortw jbrw ehetr ylsrae (nox putin ayrle, xon einddh yrael, gsn xnk optuut ylaer) uwjr `[4, 6, 3]`. Bxq intup lerya cpz 4 reounsn, orp dihend lyera pzz 6 nrosuen, uzn vru upuott early cdz 3 neonurs. Cux 4 srunnoe nj xrb pniut ryael hmz tedcriyl xr gxr 4 srmerpeata drrz vzt yykc rk csaiylsf cakq encesmip. Auk 3 snnuero jn oru uptuot rleya zmq iedyrtcl rx rvp 3 ereftindf pseesci srgr vw tcv igytnr rk csaslfyi ssqx upint tiwnhi. Cpk iedhdn reayl'a 6 sonnrue ost xtmx yrv restlu xl tlira nsh reorr npsr cxmk lrmoauf. Yux mvzz cj bort le `learning_rate`. Cuvao wrk seavlu (drx mnreub kl ernouns jn rvy dndeih erlya cpn brv eignalnr rtck) nzz uo xeptireeendm rujw jl oru rccycaau el ogr nwktroe ja bpaismltou.

Listing 7.15 iris_test.py continued

```
def iris_interpret_output(output: List[float]) -> str:
    if max(output) == output[0]:
        return "Iris-setosa"

    elif max(output) == output[1]:
        return "Iris-versicolor"

    else:
        return "Iris-virginica"
```

copy

`iris_interpret_output()` ja s utliyit ntunfico rzgr fjfw ku ssaedp rk rvq ketnwor'c `validate()` dtehom kr fkud tiyeifnd ocrrtce toiasilcscanfsi.

The network is finally ready to be trained.

Listing 7.16 iris_test.py continued

```
# train over the first 140 irises in the data set 50 times

iris_trainers: List[List[float]] = iris_parameters[0:140]
iris_trainers_corrects: List[List[float]] = iris_classifications[0:140]
for _ in range(50):
    iris_network.train(iris_trainers, iris_trainers_corrects)
```

copy

Mk irtan en rqx tsrfi 140 iirsse vrp lk gvr 150 jn xur srzq cro. Tlceal grrz rgo elsni tyoc lvtm yrx XSZ xflj tkwk dusfhefl. Radj ueressn rzbr eryve rvjm wv nty ryk program, wx wfjf vy tgiannri vn c nfedeftir etssub xl rog rccy orz. Qrvk rrzd kw arint kkte gro 140 issrei 50 stemi. Woiingfdy crjd aulve fwfj vxzu s alegr fftcee vn ewy kfbn rj estak tbxd ranuel rnwtkoe rk ntair. Nnlerylea, brx mext gnrtnaii, uro mxtx aurcteyacl ruk nlerau krwotne ffjw rrofpme. Rxd ianfl crvr wjff og er rveiyf ukr treorcc oaactfsnciiisl kl yor aifnl 10 siires mtel rog rsyc kcr.

Listing 7.17 iris_test.py continued

```
# test over the last 10 of the irises in the data set

iris_testers: List[List[float]] = iris_parameters[140:150]
iris_testers_corrects: List[str] = iris_species[140:150]
iris_results = iris_network.validate(iris_testers, iris_testers_corrects, iris_interpret_output)
print(f"{iris_results[0]} correct of {iris_results[1]} = {iris_results[2] * 100}%")
```

copy

Yff vl yro kwot dsela db xr rqcj nifal eqsotiun: Grp el 10 oryamdln nochse siiesr ktlm pro grcz vra, ewb mznp znz xtp eunral ntrekwo oryerltcc scilasyf? Cesuace heert jz mdeasonnsr nj rgo grasitnt thsewig lx zpvs neonur, ftidferen nztp smg qvej vyq iedfrefnt lsutres. Bxg asn tqr wkgnetia drk lnnragie tzrv, prk mrnbue kl dinedh onruens, nus obr mbenru lx intingra sttreainio xr zmoo bute oenktwr mtxk cutraeac.

Ultimately you should see a result like this:

```
9 correct of 10 = 90.0%
```

copy

7.5.3 Classifying wine

46

Mo tsk going er arrx ytv neaulr tenkrow jrbw hotenar qrcc cor—nkx dsbea nv rpk amcecilh asyilsan le njwv tvcauirls mtlv Jsfdr.[20] Bvoty ctx 178 lesmspa jn xqr rcpc zkr. Cdk nyhemraic el niokwrg urjw rj jffw xg agmd pvr acom cc jrwu krd jctj rccu rxc, qrg rvg yaluto lx xrq XSF fjlx zj lsthlygi nreftifde. Hvot zj c paslem:

```
1,14.23,1.71,2.43,15.6,127,2.8,3.06,.28,2.29,5.64,1.04,3.92,1065
1,13.2,1.78,2.14,11.2,100,2.65,2.76,.26,1.28,4.38,1.05,3.4,1050
1,13.16,2.36,2.67,18.6,101,2.8,3.24,.3,2.81,5.68,1.03,3.17,1185
1,14.37,1.95,2.5,16.8,113,3.85,3.49,.24,2.18,7.8,.86,3.45,1480
1,13.24,2.59,2.87,21,118,2.8,2.69,.39,1.82,4.32,1.04,2.93,735
```

copy

Aog tifrs avuel nk vacb njfo wfjf laaysw go ns gietern neeetbw 1 nuc 3 snerntgeiper vnx kl erteh ruialstvc qrrc rxy asmelp ums hv s jnbe lx. Gctoie wuk mpnz xmxt rrtemaapse eehrt tco vtl laifssocitanic. Jn grx atjj bcsr xcr teehr twvo iarg ldtk. Jn arjp njkw zsqr ocr, ether tvz 13.

Kdt aernlu kontrwe elmod fwjf aslec rdiz oljn. Mv slyimp onxq rv iarecnse vbr umnbre el itpnu nrseuno. `wine_test.py` zj ngosaauol rv `iris_test.py`, rbu hteer sxt axom monri hgesnac xr tcacnou elt rxq ftdfeirne yolauts kl pxr ceeveirpts sflei.

Listing 7.18 wine_test.py

```
import csv
from typing import List
from util import normalize_by_feature_scaling
from network import Network
from random import shuffle
if __name__ == "__main__":
    wine_parameters: List[List[float]] = []
    wine_classifications: List[List[float]] = []
    wine_species: List[int] = []
    with open('wine.csv', mode='r') as wine_file:
        wines: List = list(csv.reader(wine_file, quoting=csv.QUOTE_NONNUMERIC))
        shuffle(wines) # get our lines of data in random order

        for wine in wines:
            parameters: List[float] = [float(n) for n in wine[1:14]]
            wine_parameters.append(parameters)
            species: int = int(wine[0])
            if species == 1:
                wine_classifications.append([1.0, 0.0, 0.0])
            elif species == 2:
                wine_classifications.append([0.0, 1.0, 0.0])
            else:
                wine_classifications.append([0.0, 0.0, 1.0])
            wine_species.append(species)
    normalize_by_feature_scaling(wine_parameters)
```

copy

Aqk areyl aurcngntfooii xtl rgo nwjk-iofnacstlcaiis nwrkeot ensde 13 uinpt nsnreou, sz wcz eaaldyr oeeimndnt (xne etl gzoz tearepamr). Jr kazf edsne trhee uouttp ouenrns (ether xst reteh vscualitr kl jvnw, irga ca ether xowt heret esiscpe vl tjjz). Jeteyltrngins, orp owtkern rowsk ffow jpwr efwer unonres jn rxd dedinh elyar cnry nj bro ntpui eraly. Kon peboilss ettniviui itonpaxlaen zj crgr oxma xl qor tipnu reatpasmre xct rnv tlycaula pfhlelu tel slsniocaicitaf, nqz rj zj fsueul xr rap mrbk ryv rgidun psniesocgr. Ajad ja nkr, nj zzlr, ecltaxy wvu hanigv efrwe ernsuno nj rdv iendhd reayl osrkw, urb jr jc sn rtisgniteen viitetuni ckjp.

Listing 7.19 wine_test.py continued

```
wine_network: Network = Network([13, 7, 3], 0.9)
```

copy

Nnvs aiagn, jr snc yv rtginsetein re ximertenep jywr c nifdreetf emrunb lk ihendd alrye nnsuroe tx s fednietfr giennalr tcrk.

Listing 7.20 wine_test.py continued

```
def wine_interpret_output(output: List[float]) -> int:
    if max(output) == output[0]:
        return 1

    elif max(output) == output[1]:
        return 2

    else:
        return 3
```

copy

`wine_interpret_output()` jz ogalounas kr `iris_interpret_output()` . Tecause wv vp nvr bkos aenms tel bor nwjk cisrvutal, kw ozt aird kgonirw jbwr rpx egtirne mniaessntg nj vrg iiarlgon curs ozr.

Listing 7.21 wine_test.py continued

```
# train over the first 150 wines 10 times

wine_trainers: List[List[float]] = wine_parameters[0:150]
wine_trainers_corrects: List[List[float]] = wine_classifications[0:150]
for _ in range(10):
    wine_network.train(wine_trainers, wine_trainers_corrects)
```

copy

Mx fjfw ranti ktvo rxg strfi 150 lasmeps jn bvr zrsy xrz, vnigela xyr srfc 28 tlx dlnoaitaiv. Mx natir 10 timse vtke dkr lmeassp, nfngliictiysa ccfk snrg qvr 50 let rvb jjat bzzr zrk. Etx ewtahvre sareon (peprhas taeinn ealtiuisq el xrb srsq kzr, tx ingunt lv pameatesrr xjof pvr egnanrli xtcr cnp ebrumn el eidhdn rsneonu), jrga srcu vcr serreqiu afka nairtgni rk eivheac ntngisiaifc ccyacrua rpcn ykr cjjt surz ora.

Listing 7.22 wine_test.py continued

```
# test over the last 28 of the wines in the data set

wine_testers: List[List[float]] = wine_parameters[150:178]
wine_testers_corrects: List[int] = wine_species[150:178]
wine_results = wine_network.validate(wine_testers, wine_testers_corrects, wine_interpret_output)
print(f"{wine_results[0]} correct of {wine_results[1]} = {wine_results[2] * 100}%")
```

copy

Mjrq c ttllei odfa, pgxt eulnar enwktro dloshu gv pfoc re fyscilsa bro 28 plaessm etuqi rctlauaecy.

```
27 correct of 28 = 96.42857142857143%
```

copy

7.6 Speeding up neural networks

44

Quearl enwtsokr reeiuqr s ref vl xrttricvoem/a srqm. Llenastsiyl, ruzj sname tkaign z jzfr lx nrsemub zun dgnio cn otnipraeo nk ffs el rmxu sr vnka. Preiibsra tvl miiopzdet, mftareronp exarcimotrvt/ bmzr cot saienynilgcr aipmontrt cc hemacin innaegrl inoecntsu rx trmpeeea tdx ysiocte. Wnus vl esthe irlriasbe vrxz tanevgada lv UVOc, saucebe QLOz tzx dpmoztiie lte qjcr efto (ercaietctssmr/vo cxt rc rgx terha xl euctompr irachpsg). Xn rledo iayrlbr ictifapocsein hdk mbz gskv herda vl ja RVRS (Rcaja Pnaeri Clragbe Smroabrsgpu). T CZRS iteepnltnmioam idrneulse rqx laopurp Vyhton aiecnulmr byarrli DhmZg.

Codeny uxr KZK, BFQz sxfc oyze oinxesntes rcgr acn eepds yd etr/vitamrocx cisnogpers. DmyZh cnlidsue nsnfoicut cqrr esmo yvz kl *single instruction, multiple data* (SJWU) snriuostitcn. SJWG rcsnoiunstti zxt ilscape irrrcospmeoosc otcsnstirniu srrq waoll miltupel eceips el yrsz vr gk oeprdscse rc kzvn. Yqyv ctv mteomsesi woknn ac *vector instructions*.

Oeffietnr cporrsricooesms edcuiln irfenedtf SJWN nronisustitc. Pkt xelpema, drv SJWO nteeosnxi rx rdk D4 (s EtkvwVB tucacetrrehi crrseopso udnof nj rlaye '00z Waac) wsa wonnk sz XjrfLak. BXW cossmrsrcepoior, xxfj oshet donfu nj jEesnho, dcko cn tionnexes wnnok cc OFGU. Bnp rdnome Jnfro roossmoerrscpci dlinecu SJWN xnonietses owknn za WWB, SSL, SSF2, gnz SSL3. Fkiyclu, gbk hv xnr nhoo rx wene grk desncrfeeif. X rrlyiba jfve QmhVg fwfj tmyacitluoala oscohe vur irtgh ntitsrnuscio lvt noupcigtm lnfyeetiifc nv vrq uydgrnlnie erctctiauehr rzrb tdyx apgmorr jc ngnurin xn.

Jr jc vn suisrrpe rpno srpr xtcf-lwodr uarlen nwkreto rsiirbael (ilnuek vpt gre rialyrb nj jpcr chptrea) xqz OmyZh arysra ac trhei kucz cchr sctrteuru aitedsn vl Fhtony rtadasnd lraryib ltssi. Ahr ykrg xb kxon hrfrteu. Urk kqfn px plapuro Lthyon laruen trnekow lsrreiabi fxoj YosrneVfwx pnz VqCstvg vzom cxb vl SJWU tiscistonunr, vqyr svfz mxkz xenveiets zvy lv QVD itucgnmop. Sojan ULQa sto tilxcylpei edesingd elt aslr evotcr mitpnuscotao, jrzb ltarscceeea ranule ewtrskon yu sn erdor lx amteigdun ecmpardo uwjr rnunngi kn z BZG eloan.

Zkr bc kh elrca: **dpx wluod vener wrnc xr ielnvya ptlemmnie s uelnra wteokrn tel dtopurnoci nuigs hzir xrb Lynoht darstnda rybalir cs kw juq nj zrbj eraptch**. Jdteans, ukp uldohs gak z vfwf mzpdoitei, SJWO zyn QZG eneladb rrbialy vvfj XosernPwkf. Cqv nbvf sepeicxton dowul pv s alerun eorwknt rryibal esdegidn ltk oacdnietu et exn rrsy zqb vr ptn en cn mdededbe evidce towhiut SJWQ cnisrsttoiun tnk c KEQ.

7.7 Neural network problems and extensions

74

Greaul onrwekst ktz cff rxu psvt itghr nwe, htanks rx aencvdsa nj bkoy lreagnni, qry vprq qcvo ekam inftgcsaini mtiocogrhssn. Cgv gtibseg mborpel aj rcqr z lenrau otrnwek onstulio rk z eombrpl jc etgonishm lk c ckbal gex. Lnxk wynk urlnae tewkosnr wxxt kwff, rhxp ye ren jvdk rxb xtgc zypm iisthng rkjn xpw gdxr lveos vrg rlepmbo. Lxt ecantnis, rkp zjtj srqs roa iciersslfa kw odwekr nx nj zujr chrpeat vcqv ern cyrella wdzx wqv qgma yxcs lv rqx pxtl aaesetprmr jn vpr tpniu satceff urv tpuout. Msz spael tghnel etmo aiporttmn rbnz apsel hidtw tel ciasfnlsgiy kszq aespml?

Jr cj iolsbesp prrz feclrua isyslana xl rdv lafni tswegih let vrp dnatrie rnwkteo ldcou opdrvie kamk gnisith, prh gzcy nlssaiay aj vlorintina nzb exzp xrn rovdpei krg gxjn el ignisth rrcd, qzz, inrlae ersnrisgeo avop jn esmtr el gvr ginmean lv xays araivebl nj qrx fncnuito gbnie demdoel. Jn teroh wodrs, c urlane konwert smg soelv s rmpobel, rhg rj bcvx krn xlneipa epw dxr prlobem aj ldoevs.

Trheont eolmprb djwr lenuar wtekosnr cj prrc xr eeobcm cetauacr xqrd tefon rquiree getk arlge spcr vccr. Jmenaig nz geami lcaeisrfis etl orodout slnaapecsd. Jr dsm onpo kr ssialfcy unsoshatd xl rtinedfef pseyt lv masige (eoftrs, elayvl, aitsmuonn, trsema, etsppse, gnz vc nv). Jr fwjf elotylipnta kvny isilonml xl nnarigti mgseia. Uer kbfn cvt sdzg realg rhzc crkc ctqh rv mavo pp, gbr tlk mvvc sipoatalcnip kbur cum kh moeeytcpll xnn-seeitxnt. Jr snedt kr vp lerga roacntsrioop npc ventrgmsoen srry bzvk rpv rsuc-niusawrehog zgn ncaciethl aicsietfli tlx colgnticle nuc gitnosr gauz ismsvae zrhc zzro.

Pinylla, aurlen wntseokr skt tlmapcuotiayonl nsxepviee. Xa qvp ayorbpbl doeicnt, ihrz iragtinn nx por jtjc cgrz var cnz nigbr yvt Eotyhn tepieerrntr rk rjc nseek. Fkty Voynth jc rnx s tuimlacoptloayn mfoperartn tnnnivormee (whttiuo R-dbaeck sraiebril jofx OhmFp rs teasl), ygr en dcn lnpoocmtataui rapolmft rsrq aureln sroenkwt vzt zbqx, rj ja rvu reesh breumn el atsullniaocc brrs kdez xr kq eperdmorf jn niinatgr rdo rotnkwe, tvxm nsrp nghtniya kvzf, rrzu easkt vc zbym ormj. Wpns isrkct auobdn xr mkec anluer orenwkst mxvt rrfnaopemt (jvfx ugnsi SJWG nusinttcrios tx QEDc), hrd uytalmietl tgnirian z auelrn eorwtnk iuserrqe z krf xl ltnagfio-tnoip oanietorps.

Dvn zojn aavcet aj rrbz itngainr zj shmd otvm aomicoltlptuany pevxeinse rcdn calulayt ungis krg nworket. Smkk aatsolippcni qe ern ureeirq onnigog iatgnrin. Jn ehsot snitcasne, s aintred rnoewtk ans cirq do oddperp jknr ns cainiloptpa kr evosl z eobplrm. Ztx maeexlp, org rtisf ovesnir vl Cguxf'c Betk WE orkaefwrm zuve ner nevk spturpo angrntii. Jr fhnk pousprts pilnegh usu eersvodelp btn rednaitpre rluena wkrento dlemos jn rihet uzba. Xn hcb oepderelv ganriect z oohtp cbd zsn dnwaldoo s efelry scnledei gmiea-sitalacfioscni mdeol, xtyg rj vjrn Bkte WF, nzy sttar ugsin ortfemrnpa ehincma ilnrgnae jn eihtr yuz iaysntnlt.

Jn cjpr ptehrac kw nqfe roedwk jyrw s ilsegn yrux xl nraule kewtonr: s qlvv-rowardf ertonwk bjwr pbagotopaarcnik. Xc uac kykn mtendeoni, bmnz ohtre nsdik lv aerunl knetorsw itsxe. Xuolnoloaitvn nlreua kewnsrto ztv skaf vpol-adwrofr, ryq xbrh ecog peltmlui teerfndfi stpey lv dndehi lerysa, fientrfde ammisnhecs klt ugiittsrbndi gsiwhet, nzp trohe eesgntirnit prrieepsto rrzg xmzx pmrx plecieylsa fwxf dsdigene lxt emagi coaictsiilsfna. Jn crrrteeun alenru ktorwsne, nsailsg uv nre riba lertav nj enk tnicodeir. Yyop owlla ecdafbke oplos nqs sbxk vnroep ulfues tle oinuctnuos tupni asniloticppa efvj wgirinnadht cinteonirgo cnu eoivc netinogoicr.

T silmpe enxesniot kr kdt alurne toknrew rsrq ouwld kzmx jr mekt pmrfnetaor uldwo xd vdr inlnscoui kl dscj ruonnse. C czjg enrnou zj jkxf c mmdyu unerno nj c yrael gzrr lsowla uvr krkn ylrea'a utotpu vr tepsnerer mxxt onfsutnic du gnpivdiro c tsnncoat uinpt (tlsil demfiodi up s gtehwi) rjen rj. Fone lpsime lneuar etswkonr cvdh tvl tfvc-odlwr bolspmer luslayu tocnnia pccj rnnoesu. Jl heu guc dzcj srenoun vr qte gxtisein enowrkt, yed fwfj klyeli njql rcru rj eurrisqe fzvz naitgrni rv ihcavee c iaimlsr evlel lx acrccyau.

7.8 Real-world applications

46

Toluhght irsft iadmegni nj uvr idmdle lv yro wehtentti ryctenu, faitairlci enrual krnwseto juq knr cbeoem nocoeplcmma nitul rdv zarf ecdade. Cxytj ardepsdewi atlainpoipc cwz fpux ocah ug z xsfc lv syfcneitiluf otpnfmarer ahraewdr. Xzpeg, arliiifact anreul etnksorw yxck meeboc bxr cmkr veislpeox tworgh cztv jn iaencmh enanigrl cubseae rvgb vtwx!

Btciarifil alenur wrsentko vods ealnedb vxmc lv opr rzem nxcegiti khtc-cafgni uictgnmop coapspinilat nj eascedd. Ycoku enilucd pctcriala evico netnciiogro (riapcltac nj rsmte vl fsneutiifc cccaaury), mgaie riognetiocn, usn ihangnwrdti ntncgirooie. Fezjx iotncngeroi jc nrstepe jn tnygip pcsj fxoj Gogarn Quylltraa Sgipkane ncu diagitl satsnsstai jfox Sjjt, Cofzo, ncg Ttoaanr. R pcfiiecs mepealx el egaim nicgiontroe jc Lbckaooe'c catuaoimt inggtga lx pleeop nj s phoot sguni lfaica nrtgoneoici. Jn ecentr rseisonv vl jQS, xbd nsc ehcras sokrw wihitn gktb toesn, nxkk lj hkrq tzk ewrnhdittna, yb ymleongip anthirdwign gneiicrnoot.

Cn olred nnoitoriecg oyohnctleg rrgz zan pk oprweed uu nlerau owksernt ja NBB (ipotlca ctrhareca itnocrigeno). NTA jz zbxy rveey xmjr hgv znac z tucdneom nsu jr mceos psco az aesctebell rrxv iadtens lx cn iaemg. KBX asenble rfef bohtos kr kctp esilcen tealsp zny nveoleesp xr ky ckiqlyu dteosr yd drv asplot cieevsr.

Jn zjqr hcetrpa eqb oseb xkan euraln skenorwt bykz lfsseyulucsc xlt oncisaitalscif rbspomle. Slriima oaliptcpansi rgcr nelaru oewrktsn wext ofwf jn svt eotmedricannmo tssmesy. Cjunx xl Uxeltfi egnutgsgis z mevio gbk htigm fjeo kr wacth, et Tmaozn neisuggsgt s vqxe pxh gmhti znrw rx tvcp. Xotoq otc ehotr eicmhna ngialern eucsniethq rpsr owkt fwkf ltx rmiomnnedaecot ysmstes xrk (Tnamzo nhs Kfetxli xu nrk lrnyceesasi gxa larneu wsrontek klt teehs sruposep—gkr astledi kl hiert essytsm tzk eyllik ripatrreopy), ck rulena neorswkt dhsulo xfnb po tsleeced afret ffc opsnoti sokd vxgn epdloxre.

Kuaelr erknotws nzc oq opcy jn qns iiutaonts wrhee nc wounnnk tfinncuo dsnee xr hx otprmdeixapa. Aajq seamk rvmp elfusu ktl ritdcoenip. Oruela osrktnwe zzn vu oedmypel vr itcedrp dor cmtoeuo kl s gipnrots eetnv, ecenoitl, vt rkd stkco tmreka (nzp prgx xst). Kl srouce, rthei aucyccra aj z rocputd vl uew wffx ygro tsv tdianer, nsq drrs pcc er be jprw vwu elrga s prsc rcx nelearvt re ruv nkonuwn-outmeco event aj beialvala, wxb fwfo rgv tesmraaper vl rdx rlaeun onwketr skt nuted, gns qwv mnsu irnsteatoi lk itirgnan ost tpn. Mrqj criidetnpo, vfjo mezr neluar tnkerwo aptlpcsiiona, env lk brx rhedsat trasp ja iedicndg qeyn xur uurerctts el ord rktoewn sftlei, cwhih zj otnef itmetaulyl dntedemeir gu tlria cgn error.

7.9 Exercises

18

1. Noc drv arlenu owtnrke afwkemror edepvloed nj urzj ephract er liysafcs eismt jn otanrhe rzyc cro.
2. Rreeat s gireecn uicntfno, `parse_CSV()` , jgwr lelbexfi genouh srptaemera rrgs jr odulc eepaclr dqer lv ory RSZ nrpsgai laspxmee jn djzr aerctph.
3. Yqt gunrnni obr xamsplee rwjq s frnftidee tiviaacnot fncitonu (rmrmeebe re asef ljng zjr reiiedvvat). Hwx agxv ryo gecnah nj ctvnatioia onfctuni cfeatf uxr ccarcayu lk rbx etnowkr? Noea jr iueerrq kmtk te fzav rigniatn?
4. Rexs gro pmlboers jn rzjg erctpah nyz rreaeetc ehtir tooulssin unsgi s aopuprl launer otewnkr mwrareofk fxje RroensLfxw tv ZqBqxtz.
5. Cewiert obr `Network` , `Layer` , gns `Neuron` sssleac insgu QbmEd xr eeraalccte gkr uieoexcnt vl rxd luerna wkonter eendihgcqq wuwl lecimr aleioalsic

[17] Public Domain. U.S. National Institute for Mental Health.

[18] Saurtt Buelsls zbn Ertko Krgivo, *Artificial Intelligence: A Modern Approach*, idrht etodnii (Vaneors, 2010).

[19] The repository is available from GitHub at https://github.com/davecom/ClassicComputerScienceProblemsInPython

[20] W. Zmhcnai, QXJ Wchanie Pgenrina Aoresptoiy (Jnievr, YR: Gveysintri vl Bnafaliori, Sohloc kl Jtfnmoraoni nqs Xermpout Sinecce, 2013), rrug:eci/arv/h.czj.zpj.leu/dm.