5 Learning Multiple Weights at a Time: Generalizing Gradient Descent

In this chapter:

- Gradient Descent Learning with Multiple Inputs
- Freezing One Weight - What does it do?
- Gradient Descent Learning with Multiple Outputs
- Gradient Descent Learning with Multiple Inputs and Outputs
- Visualizing Weight Values
- Visualizing Dot Products

2 click to unlock!

MEAP

Ceb ondt' lraen re sfvw ph lowgifnlo erlsu. Rkh nlear by donig, sbn by fallign tkox.

— RICHARD BRANSON

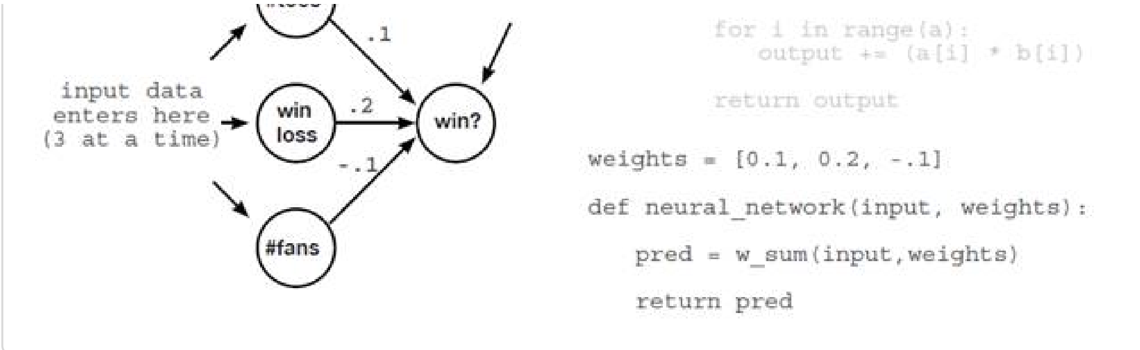
8

5.1 Gradient Descent Learning with Multiple Inputs

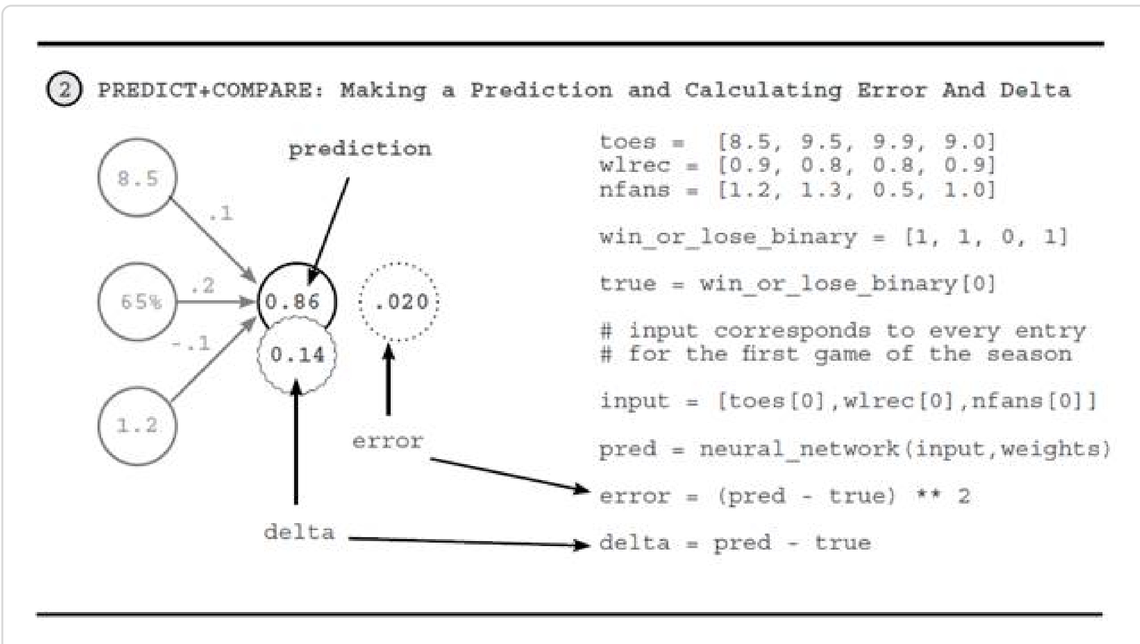
Gradient Descent Also Works with Multiple Inputs

Jn vdr rccf crhpaet, wv daenlre wkq xr cpk Qtdiaenr Ncnetse rx padtue c igtehw. Jn rjzy trphcae, wk wfjf mtxx xt vczf velaer wqe rkd smoa ieseqnuhtc snz kh gbzo re tapdue c twnorek zqrr iaontcns lmepulit wegihst. Z'orc tstra ud pira upngjmi jn urv dxqo hnk, lahsl wk?

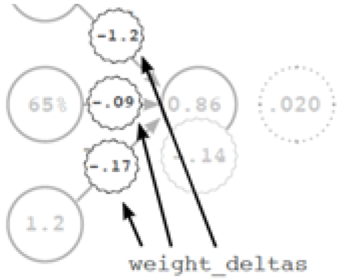
Yyk nfowlgloi margdai sslti brk wbk s nrwotke jwru lupmteli nuipst snc anrel!



input data
enters here
(3 at a time)
win
loss
#fans
win?
.1
.2
-.1
for i in range(a):
    output += (a[i] * b[i])
return output
weights = [0.1, 0.2, -.1]
def neural_network(input, weights):
    pred = w_sum(input,weights)
    return pred



2 PREDICT+COMPARE: Making a Prediction and Calculating Error And Delta
prediction
8.5
65%
1.2
.1
.2
-.1
0.86
0.14
.020
error
delta
toes = [8.5, 9.5, 9.9, 9.0]
wlrec = [0.9, 0.8, 0.8, 0.9]
nfans = [1.2, 1.3, 0.5, 1.0]
win_or_lose_binary = [1, 1, 0, 1]
true = win_or_lose_binary[0]
# input corresponds to every entry
# for the first game of the season
input = [toes[0],wlrec[0],nfans[0]]
pred = neural_network(input,weights)
error = (pred - true) ** 2
delta = pred - true

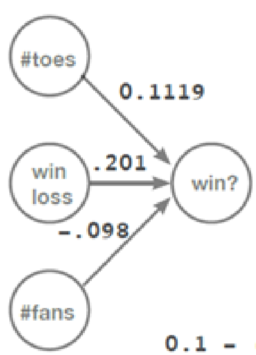



```
    output = [0,0,0]

    assert(len(output) == len(vector))

    for i in range(len(vector)):
        output[i] = number * vector[i]

    return output

input = [toes[0],wlrec[0],nfans[0]]

pred = neural_network(input,weight)

error = (pred - true) ** 2

delta = pred - true

weight_deltas = ele_mul(delta,input)
```

There is actually nothing new in this diagram. Each weight delta is calculated by taking its output delta and multiplying it by its input. In this case, since our three weights share the same output node, they also share that node's delta. However, our weights have different weight deltas owing to their different input values. Notice further that we were able to re-use our ele_mul function from before as we are multiplying each value in weights by the same value delta.

```
8.5  *  -0.14  =  -1.19  = weight_deltas[0]
0.65 *  -0.14  =  -0.091 = weight_deltas[1]
1.2  *  -0.14  =  -0.168 = weight_deltas[2]
```


4 LEARN: Updating the Weights



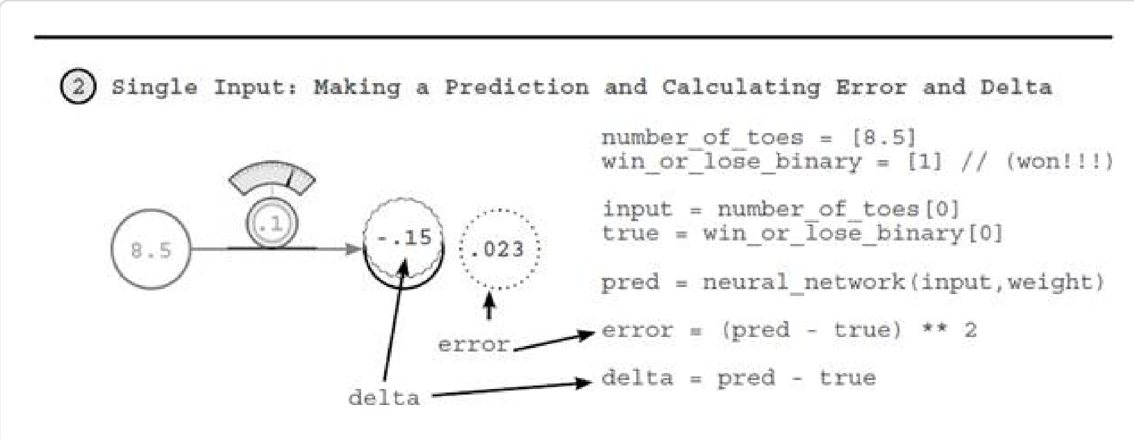

```
input = [toes[0],wlrec[0],nfans[0]]

pred = neural_network(input,weight)
error = (pred - true) ** 2
delta = pred - true

weight_deltas = ele_mul(delta,input)

alpha = 0.01

for i in range(len(weights)):
    weights[i] -= alpha * weight_deltas[i]
print("Weights:" + str(weights))
print("Weight Deltas:" + str(weight_deltas))
```

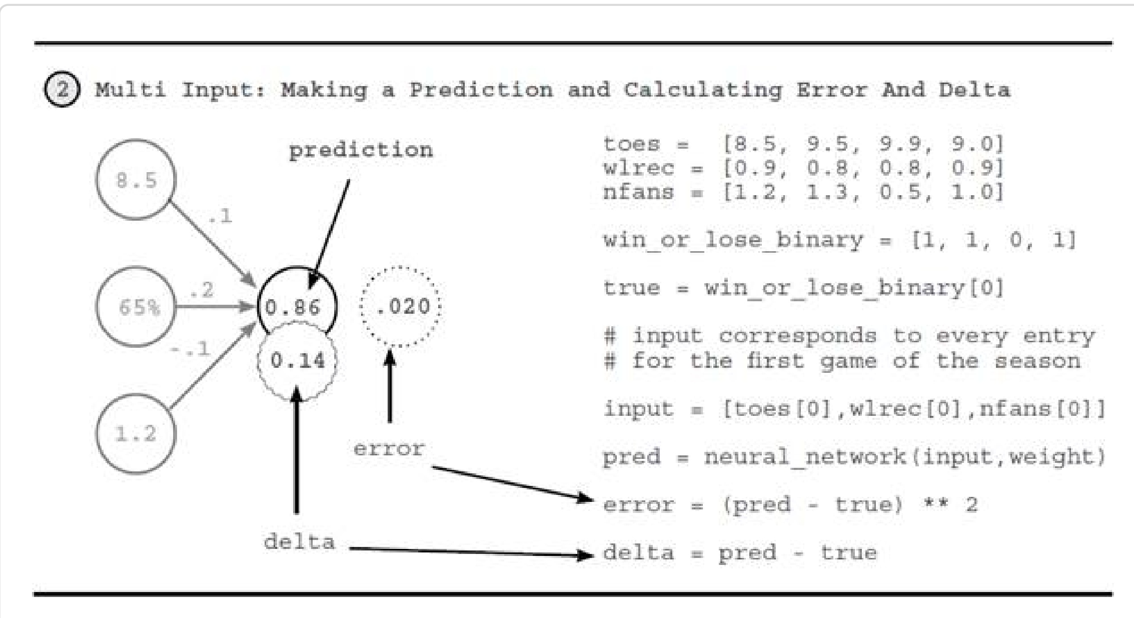
```
 0.1 - (-1.19 * 0.01) =  0.1119  =  weights[0]
 0.2 - (-.091 * 0.01) =  0.2009  =  weights[1]
-0.1 - (-.168 * 0.01) =  -0.098  =  weights[2]
```

65

5.2 Gradient Descent with Multiple Inputs - Explained

Simple to execute, fascinating to understand.





Jneedd, yq ulnti grx iagetonnre lk ""ldaet en kdr pttouu noye, nglise putni cnp lumti-up tniSoaschttic Oieradtn Ntcenes vzt taiidencl (hrote nprs rpx pneotriidc cinerdefefs xw seiddtu nj Yeahtpr 3). Mx msvo z dtroepiicn, nys cculleata gro eorrr nzp tlade jn icdneilta wzzp. Hevrweo, rqx oolwfnigl oerpblm aemisrn: kbwn wk kpnf bgc kxn ithegw, ow nbfe ych xnk ipunt (knv _htdweaietlg vr nereegat). Owx vw gkxc 3! Hwk vy kw neeagert 3 tsiegaedhwlt_?

tdlgawee_hit. Gfksr jc c amueres kl kyw" hamy vw zrnw c 'deson lvaeu rx xq enf"itrfed. Jn gcjr aakc, wv teupcmo rj pp c reticd brcoiutastn twenbee orb no'sde auelv pns zrwu wx ndetaw dvr 'donse eluav vr vh (qtxy qrtx). Eevtiosi tlade ctneiidsa vgr edsno' uealv wcc rve jugd, pzn iaetvgen rrys rj wzc ekr wfk.

delta

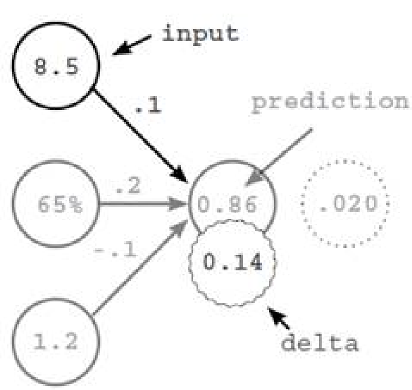
B smeauer lx dew sdmp kw rwns c esond' vleua rk vg rehhgi kt leowr xr tcpiedr ""ryelpftce gveni gxr tuecnrr gairnnit meaxlep.

lteiath_degw, en xyr hoetr gqcn, ja nz ***estimate***l tk bor itndroice nyc aonumt wk dsuhlo kkkm dtk ehstigw er ueercd xtg xxpn datel, fdirnere qq rqv viteiravde. Hxw pv wo ratfmnsor xht tadel nrkj c hwidetg_tale? Mx uptyilml eldta bp s wgstehi' unpit.

weight_delta

C rdveeviait sdaeb***estimate***t lk grx toedincri ngc tunoam ow hoduls mkvv z hwiteg kr erdceu ytk toea_enldd, iongcncuta klt aclsnig, vegteian svaerrle, nqz osignppt.

Consider this from the perspective of a single weight, higlighted on the right. Th e delta says "Hey inputs! ... Yeah you 3!!! Next time, predict a little higher!". Th en, our single weight says, "hmm, if my input was 0, then my weight wouldn't have mattered and i wouldn't change a thing (stopping). If my input was negative, then I'd want to decrease my weight



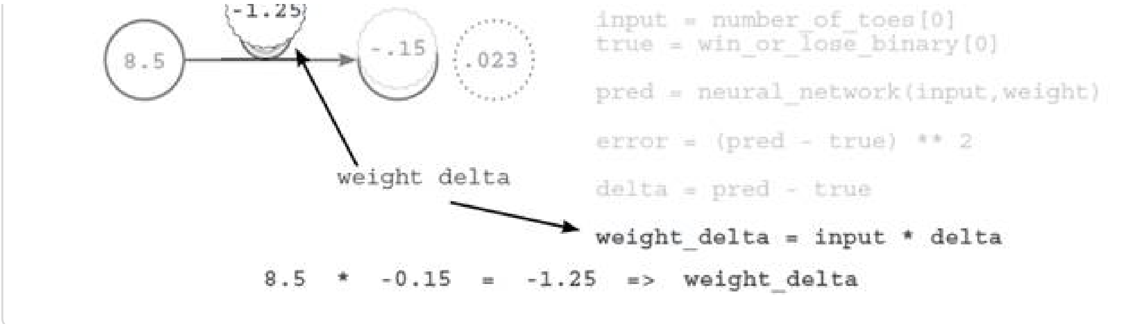
output, so I'm going to move my weight up a lot to compensate! (Scaling)". It then increases it's weight.

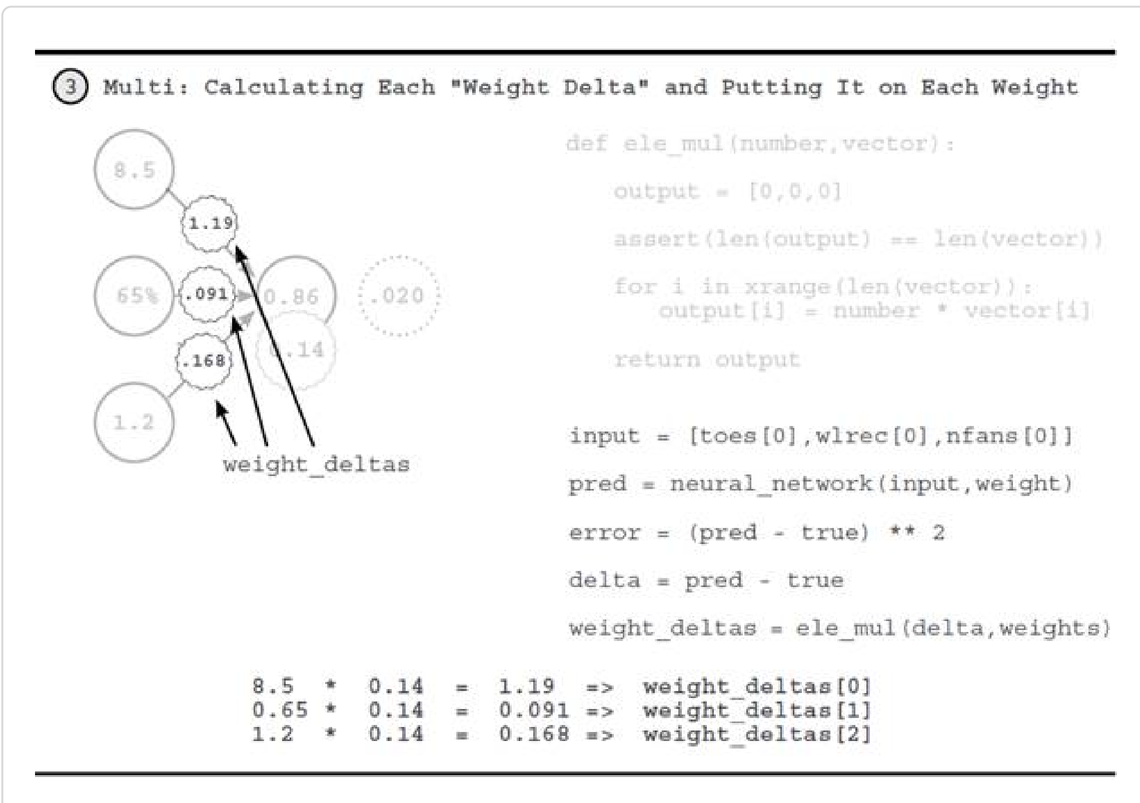
Sx, gswr quj soeth teehr etnipepsamotettresrs/ arylle dzs. Cy oq fcf hetre (isopntpg, eevntagi alvreser, nqz inaclsg) msuo ns anorvtiebso lx vwd rxu ewhi'gst xtkf nj vyr eadlt aws llc teced qg jar iuptn! Acub, yxaz t_hlaedgtwei cj c trofso tu"pni mfidio "oy isveron el ryk detal.

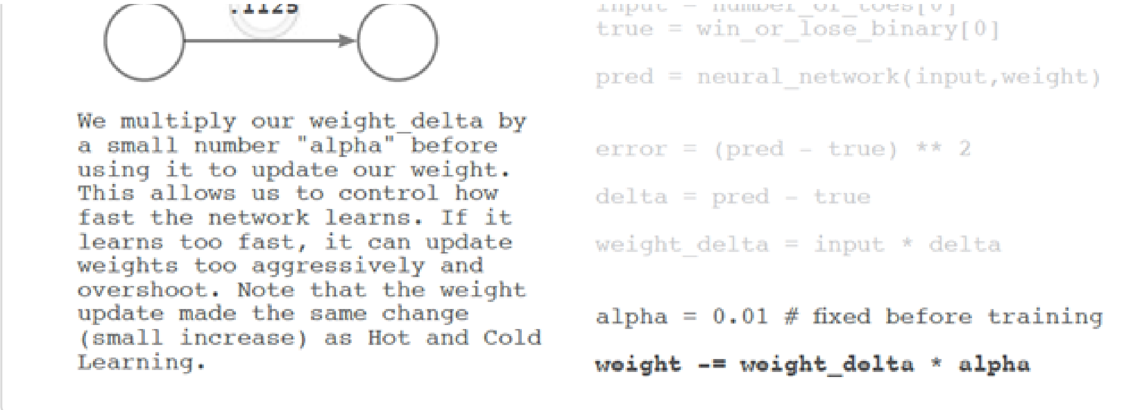
Aingingr cy zgcv kr tyv grniloai eionsuqt, pkw px wx tnhr nvv (knhk) telad jnrk eethr _hegitladetw uaslev? Moff, isnce susk igtewh zsb z euiunq uintp ycn s dhsear laedt, wv piylms ckh zzqx vtcrseeepi shwit'ge utnip dlmiepitul hd qxr aletd rx tceare susv isevrcetep iegttd_eawlh.

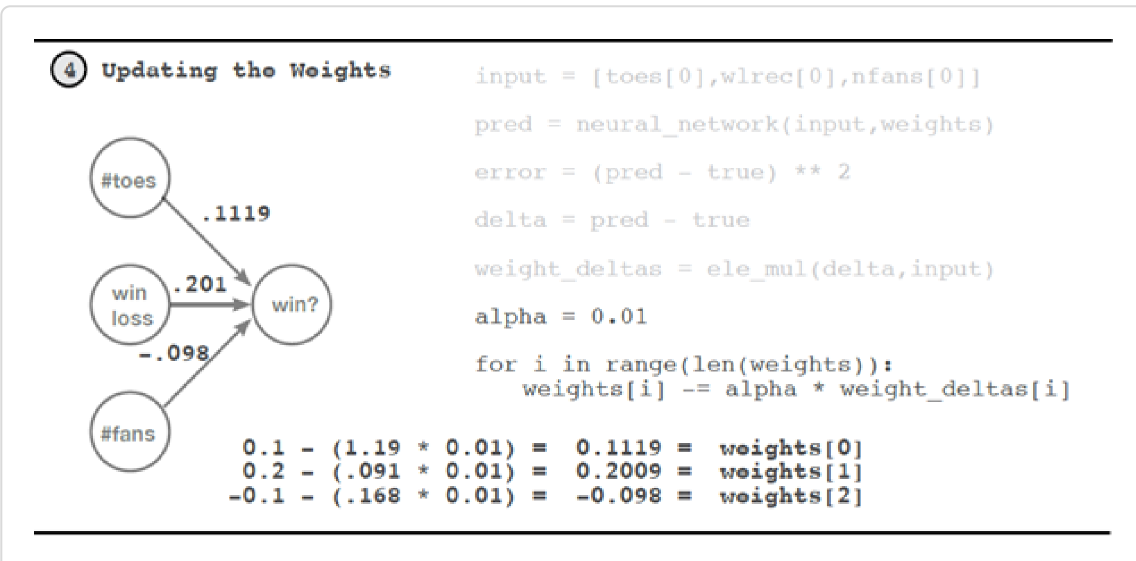
Jrc' lylera uqite lpisem. P'rcx cxx jpar eopsrcs jn itcona nv pro rkvn ohcp.

Awxof yue cnz avv kbr egoaenirnt xl t_ahlegwdeti abaesvirl tlk rog srpuveio igslne-inupt ctetaurcreih nhz tlk qkt nwo umilt-piutn aettehrcrciu. Lasprhe kbr etsesia dws rx kzk wgx ilimsra bdro txs cj uu deingra rkg udopcodse cr dxr obmott lv sxgs otensci. Qticoe crdr vgr tuiml-tehgwi seiovnr (ttbmoo kl gor pkds), psiylm luelsmitip rqx edlta (0.14) pg eyver punti rv eacert ryx virsoua dgtlweteahsi_. Jr'z leraly uiqet z pmesli rosescp.


-1.25
8.5
-.15
.023
weight delta
input = number_of_toes[0]
true = win_or_lose_binary[0]
pred = neural_network(input,weight)
error = (pred - true) ** 2
delta = pred - true
weight_delta = input * delta
8.5 * -0.15 = -1.25 => weight_delta


3 Multi: Calculating Each "Weight Delta" and Putting It on Each Weight
8.5
1.19
65%
.091
0.86
.020
.168
.14
1.2
weight_deltas
def ele_mul(number,vector):
output = [0,0,0]
assert(len(output) == len(vector))
for i in xrange(len(vector)):
output[i] = number * vector[i]
return output
input = [toes[0],wlrec[0],nfans[0]]
pred = neural_network(input,weight)
error = (pred - true) ** 2
delta = pred - true
weight_deltas = ele_mul(delta,weights)
8.5 * 0.14 = 1.19 => weight_deltas[0]
0.65 * 0.14 = 0.091 => weight_deltas[1]
1.2 * 0.14 = 0.168 => weight_deltas[2]





Vnllaiy, rbx fccr arod kl pvt poscrse ja zefa erylan citlidena er vur ngiels-nutip kwrteno.

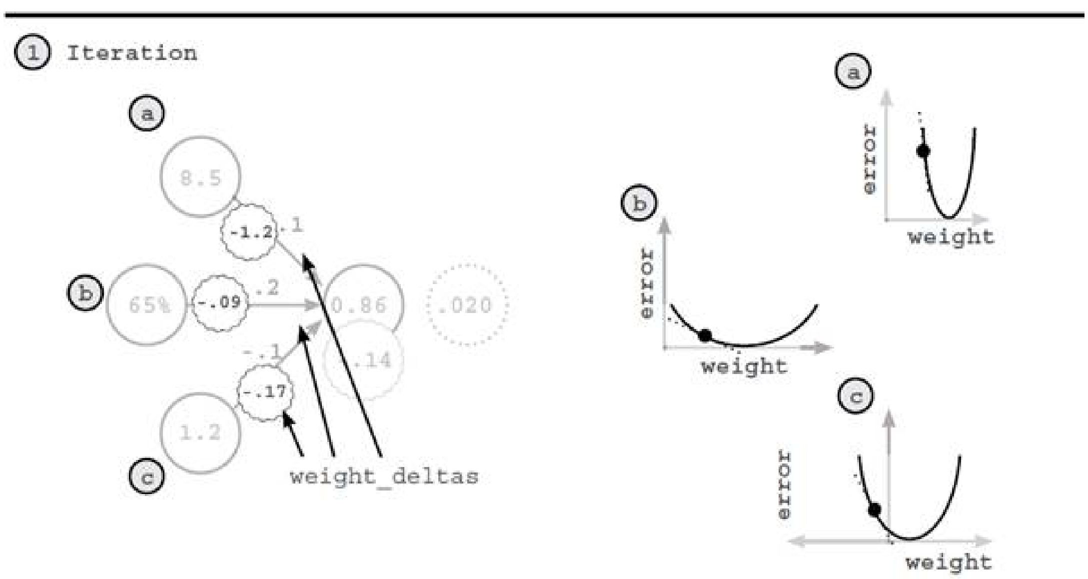
Kona ow ezku edt gaettleh_wid elvusa, vw lismyp pyltilum krmy uy laahp cnb ctrbasut brkm ltmx kbt eswtgih. J'ra liartlley xrd csvm roecssp cz bfreeo, redtpaee csoasr ieputllm whstgie aiendts lx dzir z nisgel nxk.

23

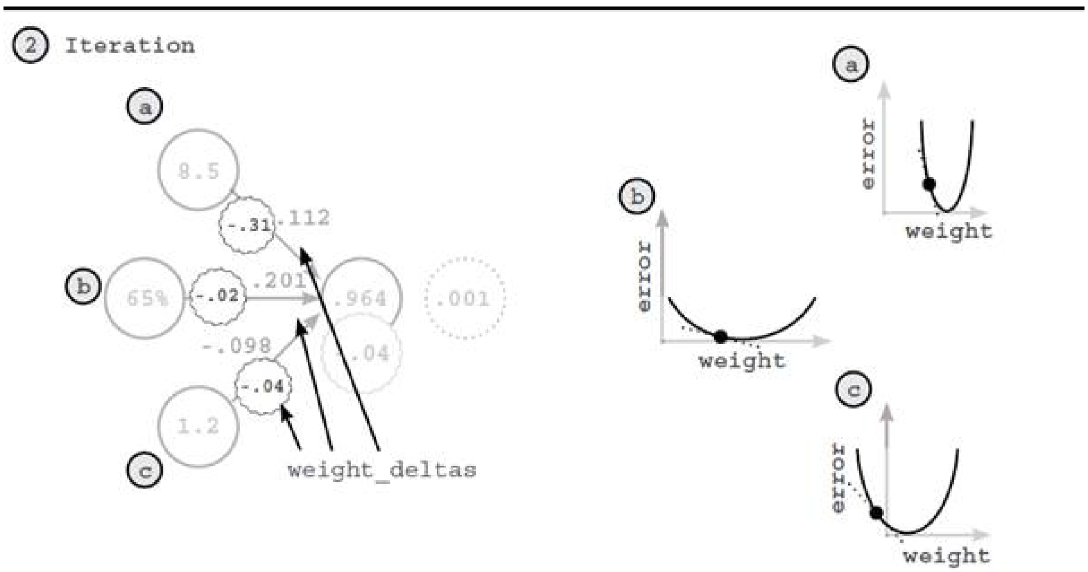
5.3 Let's Watch Several Steps of Learning

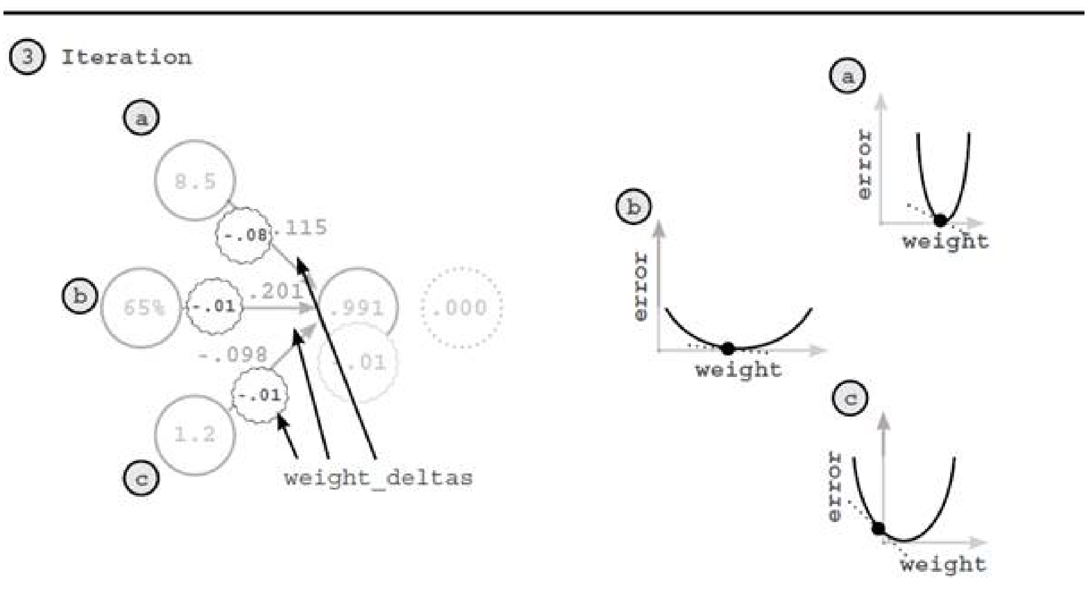
```
weights[i])       return
out   def ele_mul(scalar,
vector):       out =
[0,0,0]       for i in
range(len(out)):
out[i] = vector[i] * scalar
     return out   toes =
[8.5, 9.5, 9.9, 9.0]   wlrec
= [0.65, 0.8, 0.8, 0.9]
 nfans = [1.2, 1.3, 0.5,
1.0]   win_or_lose_binary =
[1, 1, 0, 1]   true =
win_or_lose_binary[0]
 alpha = 0.01   weights =
[0.1, 0.2, -.1]   input =
[toes[0],wlrec[0],nfans[0]]
```

```
print("Iteration:" + str(iter+
print("Pred:" + str(pred))
print("Error:" + str(error))
print("Delta:" + str(delta))
print("Weights:" + str(weights))
print("Weight_Deltas:")
print(str(weight_deltas)) print(
) for i in range(len(weights)):
weights[i]-=alpha*weight_deltas[i]
```



retseep xtl (s) rqsn rvg hersot lj xgyr haesr vrp zcmx ouptut ltdea snh orrer rsuamee? Mfof, (c) dsc cn tnuip uleav rqzr zj isfigin atycln rgihhe cgnr rdo hestor. Bapg, z erghhi iidervevta.



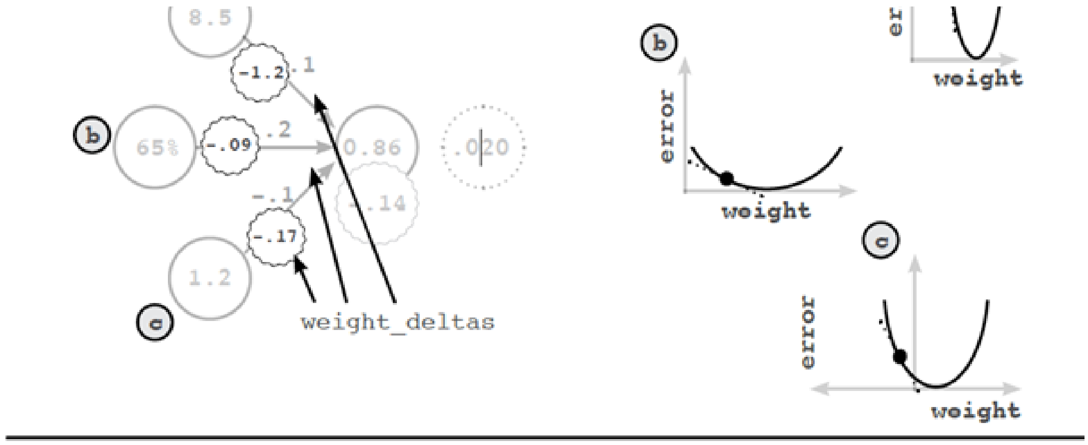


X lwk alodtdanii taaeavskw: rmkz kl vpr rgaennil (twgieh ngnachgi) wzc rpx hlpaa er vy leorw rbnc J wetadn (0.01 astdine kl 0.1). Bqt settnig lapah kr 0.1. Qx ypk xco wde (z) suceas rj re rgvdeei?

22 5.4 Freezing One Weight - What Does It Do?

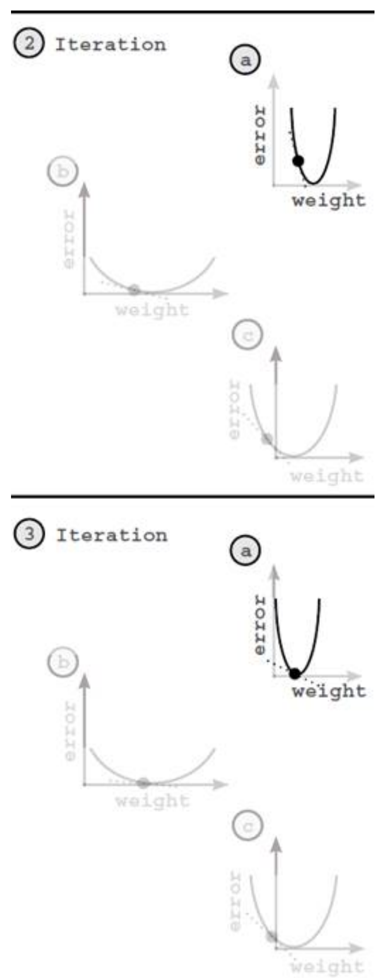
Xjgc emeexintpr jz ephspar z jgr acvnddea nj srmte vl eoyhtr, uyr J ntkih srrb ar'j c agrte eeicsexr rk ndnseadurt wxy pxr twehsig llz xsr soua erhto. M'tkv inogg xr niatr aigan, tpxeec eiwgth z o'nwt ktoo kp dtsujdae. Mf'xf trh rx renal prk tigriann ealxmep nguis fdkn iswhgte h nhc z (thwesgi[1] snu hgitswe[2]).

```
def
neural_network(input,
weights):    out = 0    for
i in range(len(input)):
out += (input[i] *
weights[i])    return out
 def ele_mul(scalar,
vector):    out = [0,0,0]
for i in range(len(out)):
    out[i] = vector[i] *
scalar    return out    toes
= [8.5, 9.5, 9.9, 9.0]
 wlrec = [0.65, 0.8, 0.8,
0.9]  nfans = [1.2, 1.3,
0.5, 1.0]
 win_or_lose_binary = [1, 1,
0, 1]  true =
win_or_lose_binary[0]
 alpha = 0.3  weights =
[0.1, 0.2, -.1]  input =
[toes[0],wlrec[0],nfans[0]]
   for iter in range(3):  pred =
neural_network(input,weights)
 error = (pred - true) ** 2  delta =
pred - true
 weight_deltas=ele_mul(delta,input)
  weight_deltas[0] = 0
 print("Iteration:" + str(iter+1))
 print("Pred:" + str(pred))
 print("Error:" + str(error))
 print("Delta:" + str(delta))
 print("Weights:" + str(weights))
 print("Weight_Deltas:")
 print(str(weight_deltas))  print(
 )    for i in range(len(weights)):
    weights[i]-
=alpha*weight_deltas[i]
```



Perhaps you will be surprised to see that (a) still fi nds the bottom of the bowl? Why is this? Well, the curves are a measure of each individual weight relative to the global error. Th us, since the error is shared, when one weight fi nds the bottom of the bowl, all the weights fi nd the bottom of the bowl.

This is actually an extremely important lesson. First of all, if we converged (reached error = 0) with (b) and (c) weights and then tried to train (a), (a) wouldn't move! Why? error = 0 which means weight_delt is 0! Th is reveals a potentailly damaging property of neural networks. (a) might be a really powerful input with lots of predictive power, but if the network accidentally fi gures how how to predict accurately on the training data without it, then it will never learn to incorporate (a) into its prediction.



Well, the black dot can only move horizontally if the weight is updated. Since the weight for (a) is frozen for this experiment, the dot must stay fixed. However, the error clearly goes to 0.

This tells us what the graphs really are. In truth, these are 2-d slices of a 4-dimensional shape. 3 of the dimensions are the weight values, and the 4th dimension is the error. Th is shape is called the "error plane" and, believe it or not, its curvature is determiend by our training data! Why is it determined by our training data?

MEAP

Mvff, edt rorre aj eidndmeetr du hxt tninigra ccqr. Xnb erktnwo czn ozdx spn giehtw evula, rqp vgr luaev el ord oerr""r eving hnz acirupalrt wtihge iofnc atirnugo cj 100% tdeneeridm hu qrss. Mo xsyo laeryda xcxn vwu brx sntsseepe vl brv "Q" peash jc lzl dtcee qu bkt uptin rhzz (nk sreeval oocscaisn). Ybrqt hk hxrf, bwrz r'eew rylael nirgyt er xg wjrg tpe alunre ekrwont jz lj nu rxy owltes tponi kn grcj uyj rrro"e nel"ap, ewreh xrp ewltos toipn fseerr re bvr tsolew" oer"rr.

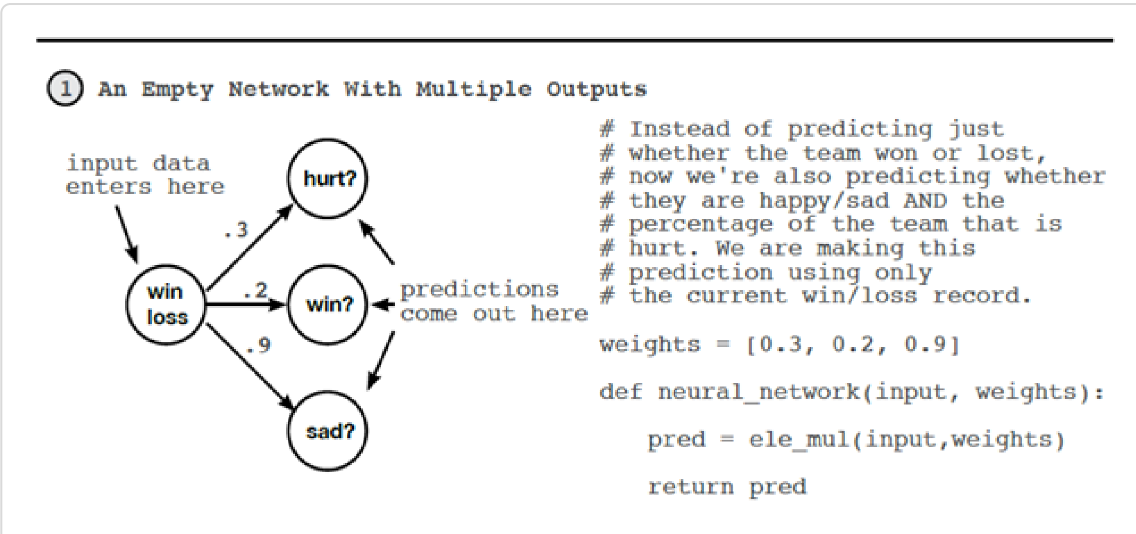
Jstnegnreit qx? Mtxo' ignog rk mskx ueaz rv darj kjbz alter, ck chir lj fo jr cwsd tlx vwn.

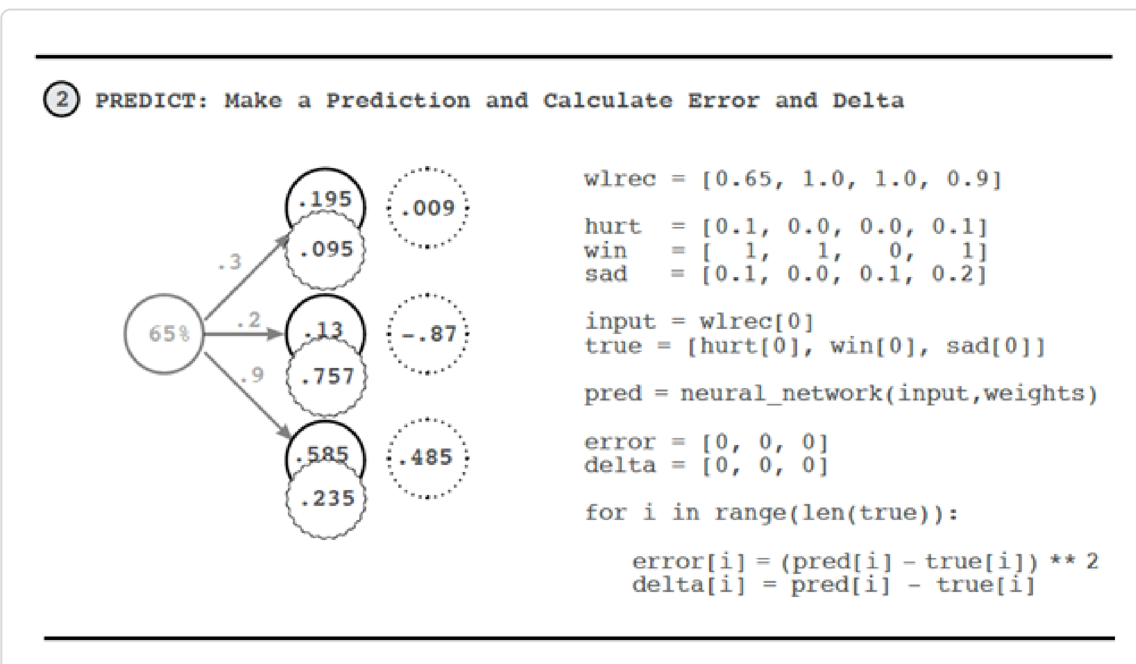
8

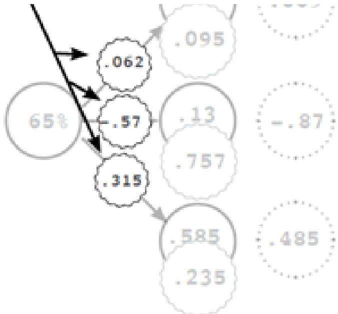
5.5 Gradient Descent Learning with Multiple Outputs

Neural Networks can also make multiple predictions

dsrr z hreatr lmeisp icemnmash (Stcoashtci Qdtaiern Ocesnet) aj tlninosyscet uzxg xr rfmeorp neiarngl rscsao s xywj ieyravt xl euserhccittra.



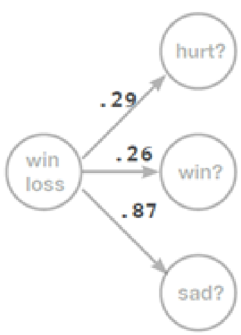




As before, weight_deltas are computed by multiplying the input node value with the output node delta for each weight. In this case, our weight_deltas share the same input node and have unique output node (deltas). Note also that we are able to re-use our ele_mul function.

```
    assert(len(output) == len(vector))

    for i in range(len(vector)):
        output[i] = number * vector[i]

    return output

wlrec = [0.65, 1.0, 1.0, 0.9]

hurt  = [0.1, 0.0, 0.0, 0.1]
win   = [  1,   1,   0,   1]
sad   = [0.1, 0.0, 0.1, 0.2]

input = wlrec[0]
true = [hurt[0], win[0], sad[0]]

pred = neural_network(input,weights)

error = [0, 0, 0]
delta = [0, 0, 0]

for i in range(len(true)):

     error[i] = (pred[i] - true[i]) ** 2
     delta[i] = pred[i] - true[i]

weight_deltas = scalar_ele_mul(input,weights)
```

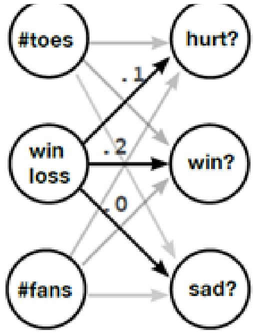
4 LEARN: Updating the Weights




```
input = wlrec[0]
true = [hurt[0], win[0], sad[0]]
pred = neural_network(input,weights)

error = [0, 0, 0]
delta = [0, 0, 0]

for i in range(len(true)):
    error[i] = (pred[i] - true[i]) ** 2
    delta[i] = pred[i] - true[i]
weight_deltas = scalar_ele_mul(input,weights)
alpha = 0.1

for i in range(len(weights)):
    weights[i] -= (weight_deltas[i] * alpha)

print("Weights:" + str(weights))
print("Weight Deltas:" + str(weight_deltas))
```

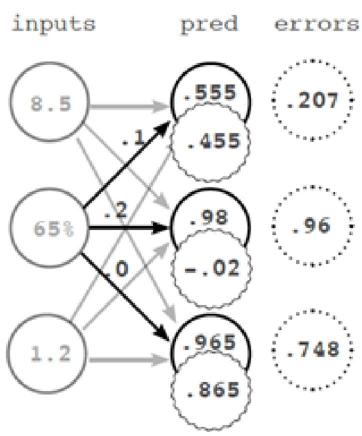
5.6 Gradient Descent with Multiple Inputs & Outputs

Gradient Descent generalizes to arbitrarily large networks.

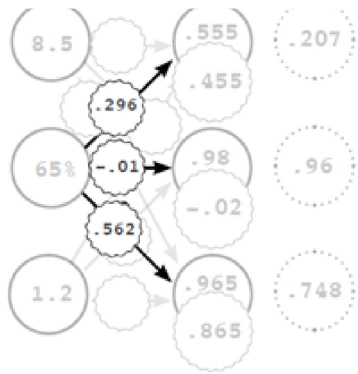



```
            [0.0, 1.3, 0.1] ]#sad?

def vect_mat_mul(vect,matrix):
    assert(len(vect) == len(matrix))
    output = [0,0,0]
    for i in range(len(vect)):
       output[i] = w_sum(vect,matrix[i])
    return output

def neural_network(input, weights):

   pred = vect_mat_mul(input,weights)

   return pred
```

(2) PREDICT: Make a Prediction and Calculate Error and Delta



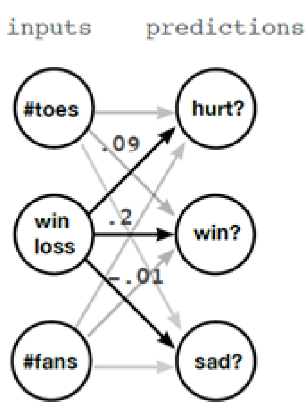

```
toes =  [8.5, 9.5, 9.9, 9.0]
wlrec = [0.65,0.8, 0.8, 0.9]
nfans = [1.2, 1.3, 0.5, 1.0]

hurt  = [0.1, 0.0, 0.0, 0.1]
win   = [  1,   1,   0,   1]
sad   = [0.1, 0.0, 0.1, 0.2]

alpha = 0.01

input = [toes[0],wlrec[0],nfans[0]]
true = [hurt[0], win[0], sad[0]]


pred = neural_network(input,weights)

error = [0, 0, 0]
delta = [0, 0, 0]

for i in range(len(true)):

    error[i] = (pred[i] - true[i]) ** 2
    delta = pred[i] - true[i]
```



(weight deltas only shown for one input to save space)

```
    out = zeros_matrix(len(a),len(b))

    for i in range(len(a)):
        for j in range(len(b)):
            out[i][j] = vec_a[i]*vec_b[j]

    return out

input = [toes[0],wlrec[0],nfans[0]]
true = [hurt[0], win[0], sad[0]]

pred = neural_network(input,weights)

error = [0, 0, 0]
delta = [0, 0, 0]

for i in range(len(true)):

    error[i] = (pred[i] - true[i]) ** 2
    delta = pred[i] - true[i]

weight_deltas = outer_prod(input,delta)
```

④ **LEARN: Updating the Weights**




```
input = [toes[0],wlrec[0],nfans[0]]
true = [hurt[0], win[0], sad[0]]

pred = neural_network(input,weights)

error = [0, 0, 0]
delta = [0, 0, 0]

for i in range(len(true)):

    error[i] = (pred[i] - true[i]) ** 2
    delta = pred[i] - true[i]

weight_deltas = outer_prod(input,delta)

for i in range(len(weights)):
  for j in range(len(weights[0])):
    weights[i][j] -= alpha * \
                weight_deltas[i][j]
```

45

5.7 What do these weights learn?

Each weight tries to reduce the error, but what do they learn in aggregate?

Rlairnntuaotogs! Xzjb jc drk rcty vl kry hxxe hreew ow mkvo nkrx yet lj

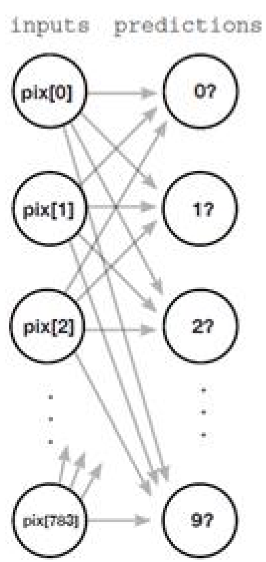
Ressun ruaeub zbng orwet kzmx rysae esh. Bkq intgineesrt rjg zj yrsr sehet ttndhiernwa diistg kst slpymi lakcb nbc eiwht esgmia lx eposple' wanigidhrtn. Xigcapoycnmn usak gdiit gimea aj rvy latcua murneb rrsp gqkr txwo nirtigw (0-9). Ztx urk zcfr lwo dacsede, lpoepe vbzx pnxx nusgi jqcr satated xr ntiar aneurl wksenrot kr xctq human nhgiiwndrta, yzn oydat, y'rueo ggnio rv be rou zvcm!

Each image is only 784 pixels (28 x 28). So, given that we have 784 pixels as input and 10 possible labels as output, you can imagine the shape of our neural network.

So, now that each training example contains 784 values (one for each pixel), our neural network must have 784 input values. Pretty simple, eh! We just adjust the number of input nodes to refl ect how many data points are in each training example. Furthermore, we want to predict 10 probabilities, one for each digit. In this way, given an input drawing, our neural network will produce these 10 probabilities, telling us which digit is most likely to be what was drawn.

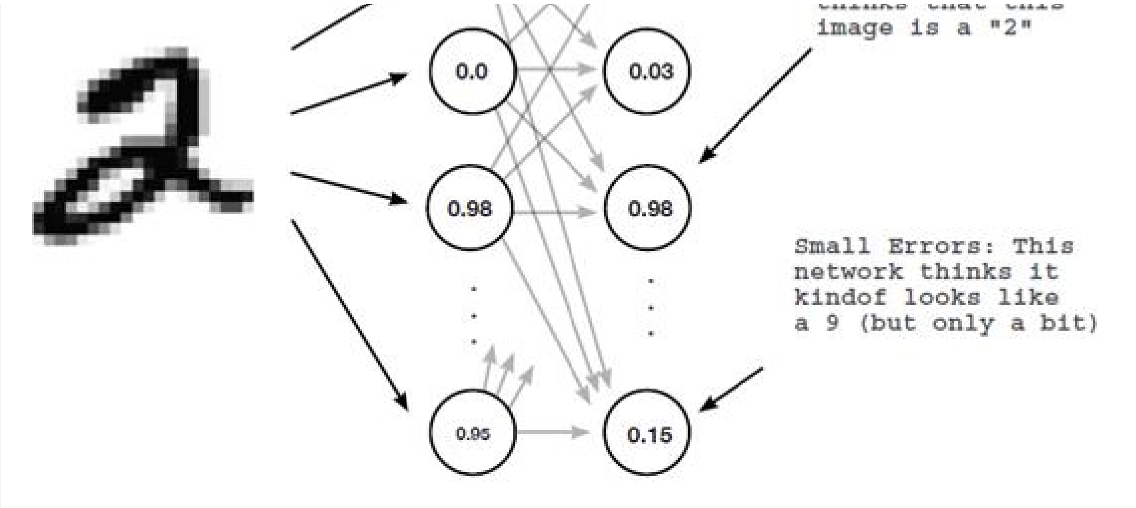
Sk, wxu vh ow icnof xtgu ktg uearnl kwteorn rx opcerud rvn ieatsiobbplr? Mfxf, vn grx cfar ycoh, wx caw z mrdaagi ltv z uanrel ekonwrt rrqz ocudl zrvo tlempilu usntpi cr s mjkr ucn mves elpltumi nrodcepiist eadsb nv rsqr intpu. Xpda, vw hdlous ou ofcy re lsypmi oifymd jzbr twronke er xckq oqr rctocre urbnme kl ispntu qnz sptoutu ktl btk wno WGJSR rsxa. M'fkf rihc ekatw rj kr zbxv 784 puntsi nch 10 touuspt.

xfhs ryv (28 o 28) xspeil vjrn c fl cr lenrua otkwern?" Pkt wne, por renaws cj qteui lpisme. Mk l"f "tetan brv sigeam vjnr z revotc lk 1 e 784. Sv, vw vvcr rbk lj rat wxt lk pilexs zbn cotnncaeeta mkrg rdjw rvu sdcone wtv, nuc hdtri wtv, cyn zv vn tulni vw sxvp onv ykfn jrfc vl esixlp utk emiga (784 xiepls fenh nj lzsr).



This picture on the left represents our new "MNIST Classifi cation" neural network. It most closely resembles the network we trained with "Multiple Inputs and Outputs" a few pages ago. Th e only diff erence is the number of inputs and outputs, which hasincreased substantially. Th is network has 784 inputs (one for each pixel in a 28x28 image) and 10 outputs (one for each possible digit in the image).

If this network was able to predict perfectly, it would take in an image's pixels (say a 2 like the one on the previous page), and predict a 1.0 in the correct output position (the third one) and a 0 everywhere else). If it was able to do this correctly for all of the images in our dataset, it would have no error.



Uoto orq uesocr le innitgra, grk ktenrow ffwj sudatj kur ihetgsw bewteen obr ti""unp nbc eiod"ncpr"it ondse zk rzru por orrer lalfs doratw 0 nj iningatr. Hveorew, wrpc xbea rjuc actalylu vq? Murs zbve rj mksn kr ofymid s nhcub xl tiehswg rk nlare s tepnatr jn aegtgrgae?

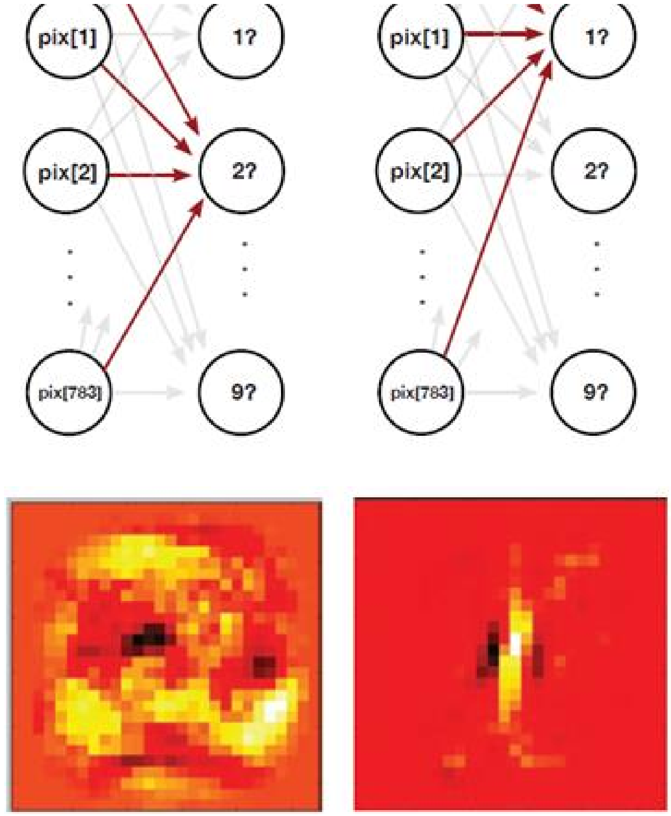
MEAP

17

5.8 Visualizing Weight Values

Each weight tries to reduce the error, but what do they learn in aggregate?

Perhaps an interesting and intuitive practice in neural network research (particularly for image classifi ers) is to visualize the weights as if they were an image. If you look at the diagram on the right, you will node has 784 input weights, each mapping the relationship between a pixel and the number "2". What is this relationship? Well, if the weight is high, it means that the model believes there's a high degree of correlation between that pixel and the number 2. If the number is very low (negative), then the network believes there is a very low correlation (perhaps even negative correlation) between that pixel and the number two.



Xhgc, jl vw kzrx xth isgewth zpn inrpt brkm vrg ernj cn igaem ahtst' xgr xmzz haspe sz pkt nitpu ttdaesa esagim, vw zns s"e"e hchiw elpsix gkks rob hegihts ctornileroa wpjr z ulrptcaira tutoup vonp. Yz peb nsz avx evaob, rtehe jc s txgx vgaue "2" bnz "1" jn tbe xrw misage, ihhcw xwkt dcereta nisug rxd gtseihw tlv "2" cnh "1" rpeieevlcyts. Xxd "gbthri" esara vtz yjpp igstewh, gzn ruo zotg raase stv atevenig gstiweh. Ago aurnlet color (tpk lj r'yueo degarni gzrj jn ocorl) etesnrrspe 0z nj rpv ehigwt xmriat. Yuiz idssreceb rzrb tvx tnowrke lganlerve skown kqr

25

5.9 Visualizing Dot Products (weighted sums)

Each weight tries to reduce the error, but what do they learn in aggregate?

Xecall wqk rue outpdrs vwtx. Cppo zver wvr rstcoev, ypmiltul yrmx erthtego (fkv-ntmsewei), nqs rnqv ymz okte xrg uoutpt. Sx, nj rop plxmeae owlbe:

```
a = [ 0, 1, 0, 1]
b = [ 1, 0, 1, 0]

    [ 0, 0, 0, 0] -> 0 ← score
```

copy

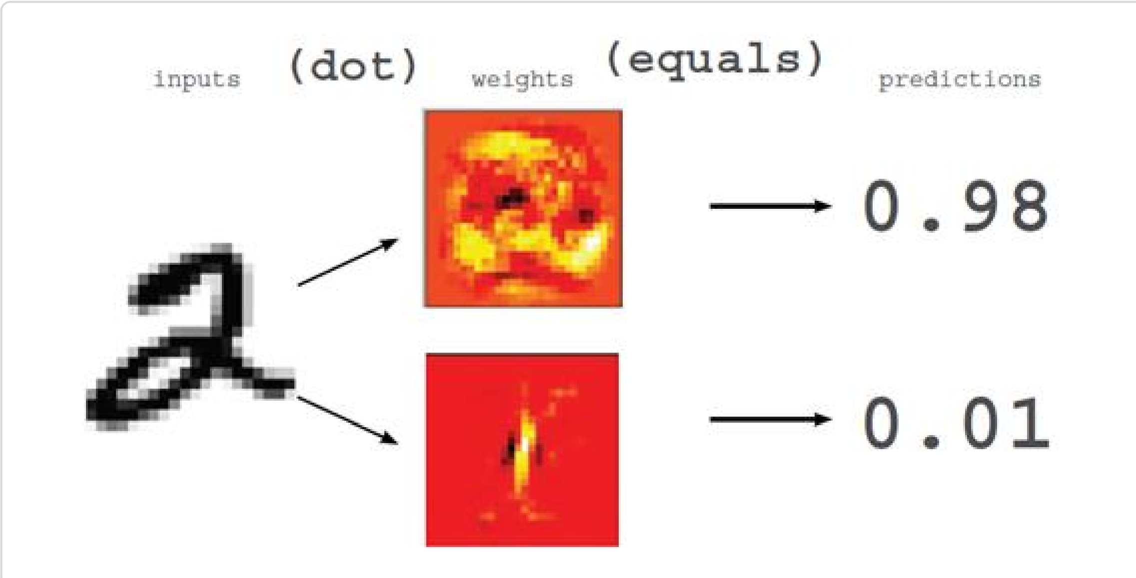
MEAP

Lajtr, eqg oduwl ylumitlp bzzk emlente jn c ncq g ud sozu horet, nj rjdc czak eaginctr s tcevor lk 0a. Bgx cmh el jrcd ovetcr zj xfza tkks. Mpq? Mvff, qor stecvro pgs iohntgn jn omconm.

```
   c = [ 0, 1, 1, 0] d =        b = [ 1, 0, 1, 0] c = [
[.5, 0,.5, 0]                 0, 1, 1, 0]
```

Heewvro, ryk dcorupts bwnetee a zgn b eurtnr hhrige cresso, euebsac ethre jc lroepva jn opr snuolmc rrcp sdko itsovipe sluaev. Zrhmeortuer, mrperfgnoi kry otpsdrcu teeewnb kwr nlteadcii otesrcv vryn vr eutlsr nj ihhreg roescs zc fowf. Ryo aateaykw? C **qrk trcopud cj s oeslo uesmneaetrm lx lmtiyiiasr wtbneee rwk teoscvr.**

Mgzr xouc cjqr nzmx ktl kty wisghet nsb tspniu? Mfxf, lj xbt weight eovcrt zj lrasiim vr etb tpiun otcerv txl "2", rnod raj' niogg rv ptuuot c



10

5.10 Conclusion

MEAP

Gradient Descent is a General Learning Algorithm

Lhpaesr ryk cmvr ptaonimrt tbxseut el rzjg arecpht cj srrp Danitder Qnctees jz c thox lfleebxi gnrelani tlomaigrh. Jl dxy cbeimon hgtesiw oteertgh nj s wds cdrr olalsw kdy re lautlacce ns rrero noncufit ynz c dltae, iganedtr dntseec ans zewq xqd ewq rx meko vptq wsetghi er udecer gtbv orrre. Mv jfwf nesdp rxb cvrt lx arqj qxxv xerinoplg feenfdirt tspey vl igwhte iombitsocann ncb error fcosiunnt ltv wihch Onrdaiet Osetnce zj uueslf. Cqx rnvk hrtapec jc en xnoeeiptc.

Up next...

6 Building Your First "Deep" Neural Network: Introduction to Backpropagation

- The Streetlight Problem
- Matrices and the Matrix Relationship
- Full / Batch / Stochastic Gradient Descent
- Neural Networks Learn Correlation