



Introduction to Data Science

Training Feed-Forward Neural Networks

01 Oct 2026

Training Neural Networks

Roadmap

1. Define the error

Use unknown food prices to connect inputs, predictions, and targets.

2. Choose a useful weight change

Read a loss curve and calculate gradient-descent updates.

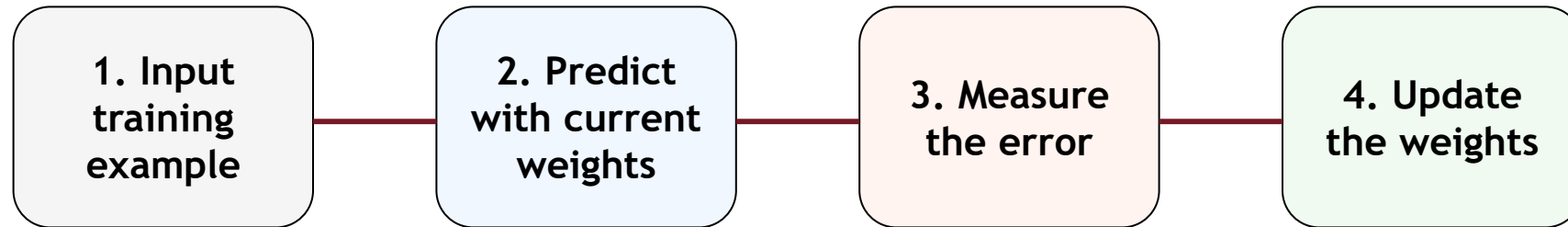
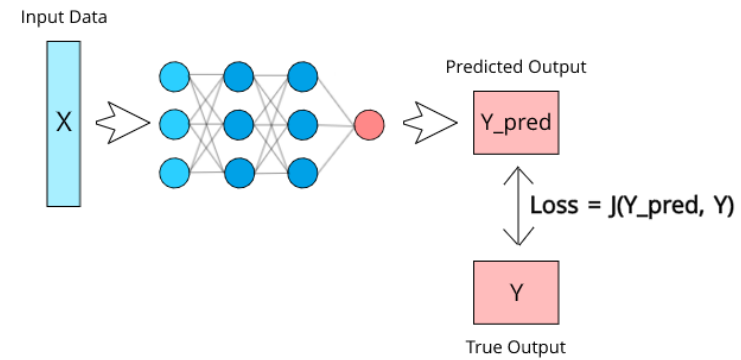
3. Train several layers

Use local derivatives to pass error information backward.

Training means changing parameters to reduce a measured error.

Training Neural Networks

One Training Loop at a Glance



The training data stay fixed. The weights are the quantities that change.

After one update, run the forward pass again and check whether the loss became smaller.

Training is a repeated loop: predict, compare, update, and repeat.

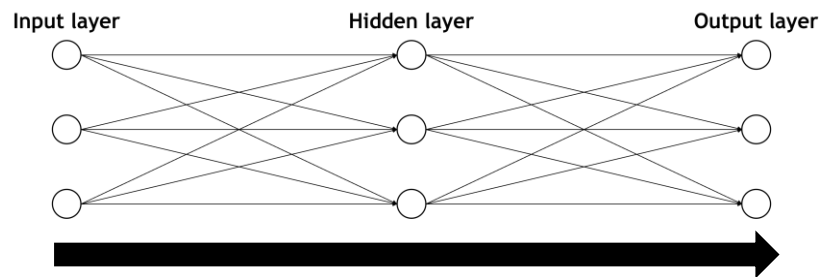
The Fast-Food Problem

Last Class: Use the Weights. Today: Find Them.

FORWARD COMPUTATION

Quantities and prices are known.

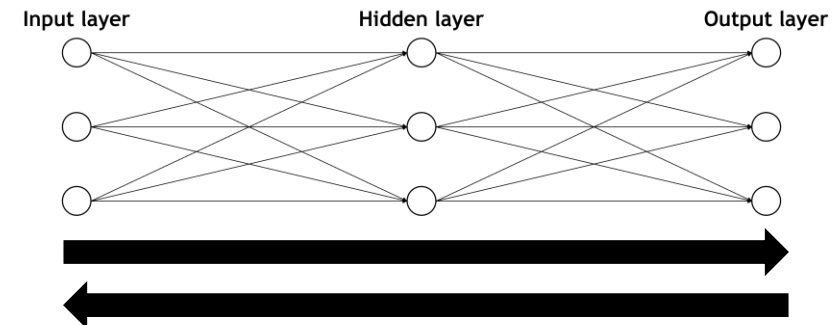
Multiply each quantity by its price, then add the costs.



TRAINING

Quantities and total prices are known.

Adjust the unknown item prices until predicted totals match the receipts.



$x \rightarrow \text{model with weights } w \rightarrow \text{prediction } y$

A forward pass calculates a prediction; training changes the weights.

The Fast-Food Problem

We See the Receipt, Not the Item Prices

OBSERVED RECEIPT

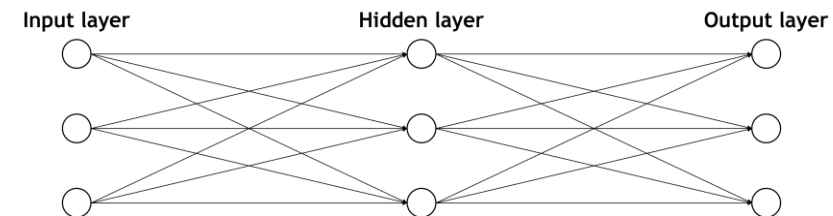
2 burgers	?
1 fries	?
3 sodas	?
Total	\$15

WHAT IS KNOWN?

We bought 2 burgers, 1 serving of fries, and 3 sodas.

The cashier reports a total of \$15.

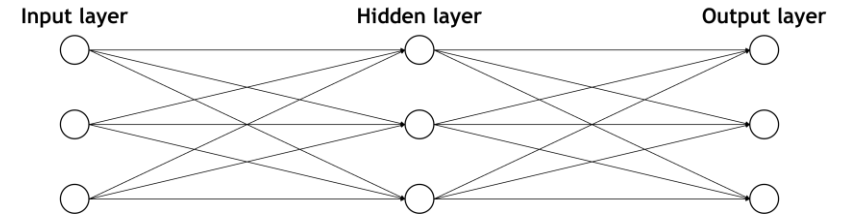
The three item prices are still unknown to the model.



The total price is the target that tells us whether our prediction is wrong.

The Fast-Food Problem

From Receipts to Unknown Unit Prices



Burger
= w_1

Fries
= w_2

Drink
= w_3

Receipt example:
2 burgers + 1 fries + 1 drink = \$13

Each receipt gives one input-output example: item counts are inputs, and the total bill is the target.

The weights w_1 , w_2 , w_3 are the model's current guesses for the unknown unit prices.

Training uses many receipts to find a set of weights that explains the totals well.

The Fast-Food Problem

Three Receipts Give Three Training Examples

Example	Burgers	Fries	Sodas	Target total
1	2	1	3	\$15
2	1	2	1	\$10
3	3	0	2	\$17

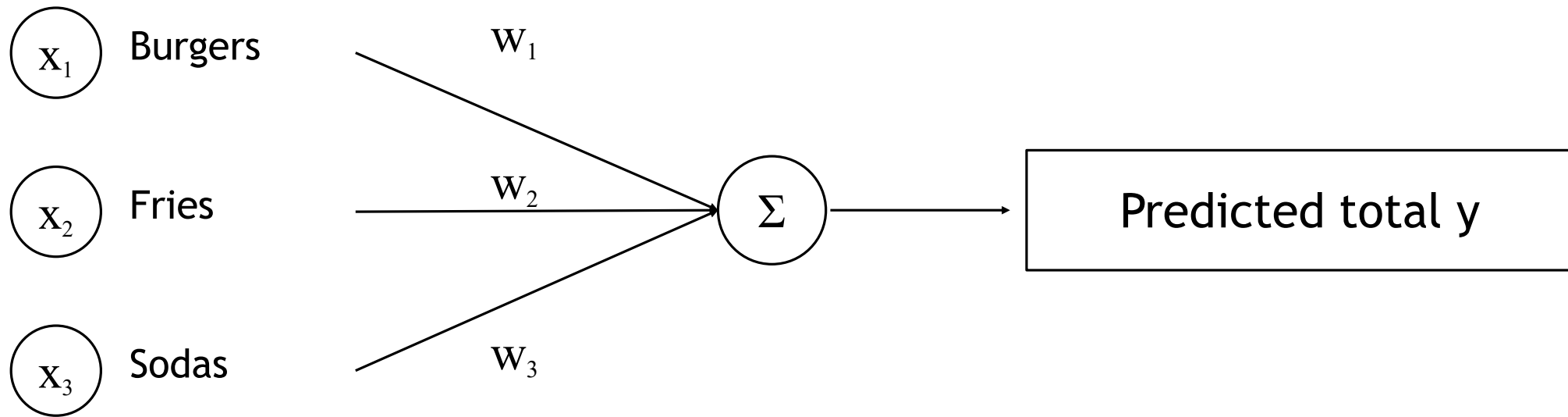
The quantities form x . The receipt total is t .

Training compares each prediction y with its corresponding target t .

One row contains an input and its correct target.

The Fast-Food Problem

The Linear Neuron Is a Price Calculator



$$y = w_1x_1 + w_2x_2 + w_3x_3$$

Here, a weight is the model's current guess for one item price.

The Fast-Food Problem

Why Not Buy Each Item Separately?

A clever shortcut for this toy problem



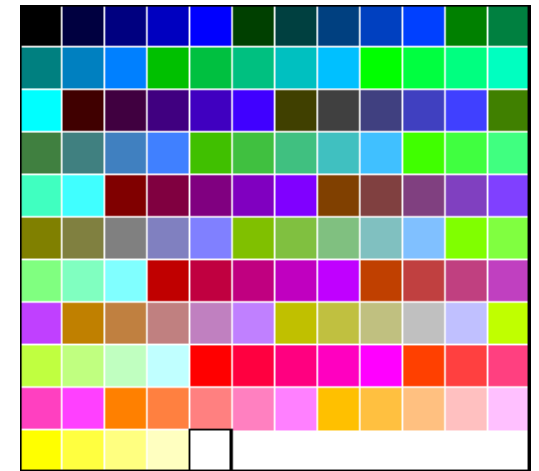
Buy one burger, then one fries, then one soda to reveal all three prices.

Why the chapter moves beyond this shortcut

For image recognition, there is no obvious “one item at a time” equivalent.

A more general strategy

Use many available examples and reduce their prediction errors.



We need a training method that extends beyond a specially chosen linear example.

Measuring Prediction Error

How Wrong Is One Prediction?

Target \$15	—	Prediction \$12	=	Residual \$3
----------------	---	--------------------	---	-----------------

The model predicted \$3 less than the receipt total.

A residual of zero means this prediction matches this target.

The residual tells us the size and sign of the mismatch.

The residual is the target minus the prediction: $t - y$.

Measuring Prediction Error

Residual and Squared Error: A Very Small Example

Target = 10
Prediction = 8
Residual $t - y = 2$
Squared error = 4

Target = 10
Prediction = 12
Residual $t - y = -2$
Squared error = 4

The sign of the residual tells us the direction of the mistake: underprediction or overprediction.

Squaring removes the sign and makes larger mistakes count more strongly.

Residual tells the direction of the error; squared error tells the size of the penalty.

Measuring Prediction Error

Why Square the Difference?

Prediction error	Residual	Squared residual
Under by 3	+3	9
Over by 3	-3	9
Exact match	0	0

Without squaring, +3 and -3 would add to zero despite two wrong predictions.

Squaring makes both mistakes contribute positively.

Squaring prevents positive and negative errors from cancelling in the loss.

Measuring Prediction Error

The Chapter's Squared-Error Objective

$$E = \frac{1}{2} \sum_i (t^{(i)} - y^{(i)})^2$$

i selects one training example

t is the correct target. y is the prediction produced by the current weights.

Square each residual, then add

The factor $\frac{1}{2}$ makes later derivatives simpler. It does not change the minimizer.

Choose weights that make the total squared error small.

Measuring Prediction Error

Evaluate One Current Guess

Current weights: $w = (4, 1, 1)$

Example	Target t	Prediction y	$\frac{1}{2}(t - y)^2$
1	15	12	4.5
2	10	7	4.5
3	17	14	4.5

$$E = 4.5 + 4.5 + 4.5 = 13.5$$

One set of weights is evaluated on every training example.

Measuring Prediction Error

What Changes During Training?

FIXED IN THIS EXAMPLE

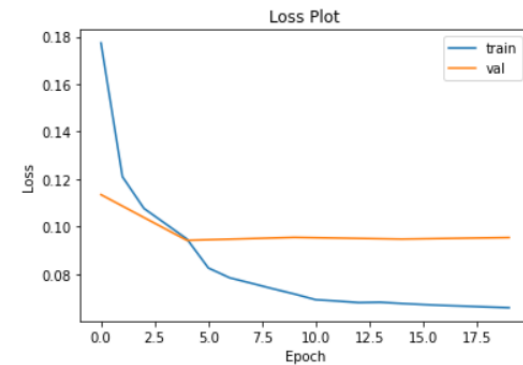
Item quantities x .

Observed receipt totals t .

ADJUSTED BY TRAINING

The price guesses w .

These new weights produce new predictions y and a new loss E .



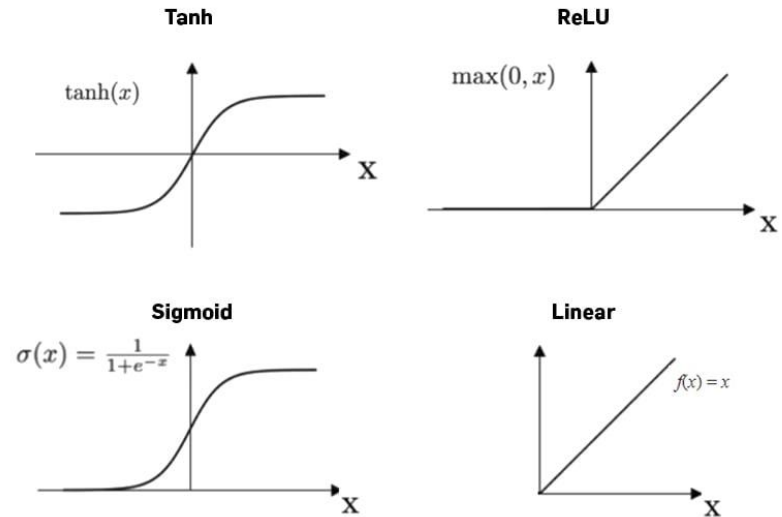
For fixed training data, changing the weights changes the predictions and the loss.

Measuring Prediction Error

Why Not Just Solve Linear Equations?

For a consistent linear model, the receipts can be written as linear equations.

But a network with sigmoid, tanh, or ReLU is no longer the Same linear-equation problem.

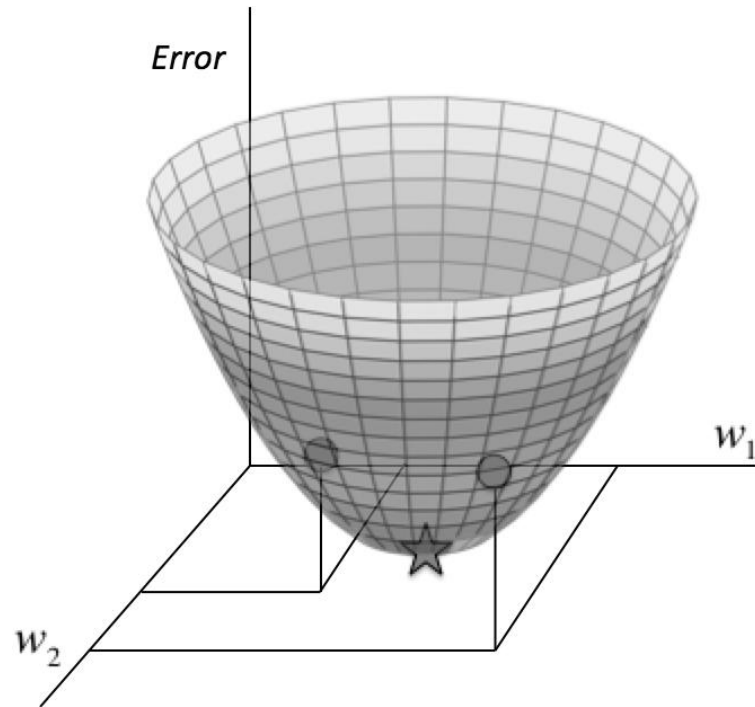


The chapter therefore develops a method that changes weights using the loss.

A general training method must also work when the network is nonlinear.

Gradient Descent

Read the Loss Surface



Each horizontal position is one choice of two weights.

The height is the loss for those weights.

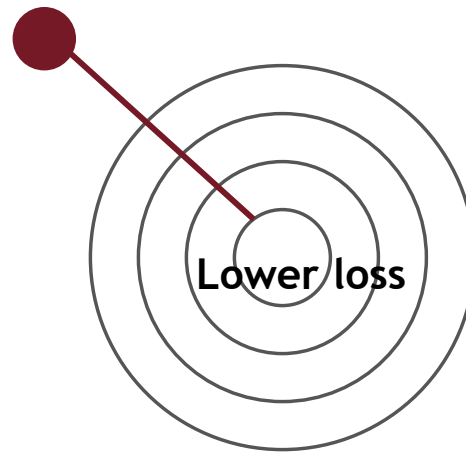
The low part of the bowl represents smaller error.

The horizontal axes are weights, not the input features.

Gradient Descent

Think of the Loss as a Landscape

Current weights



Each point means one choice of the weights.

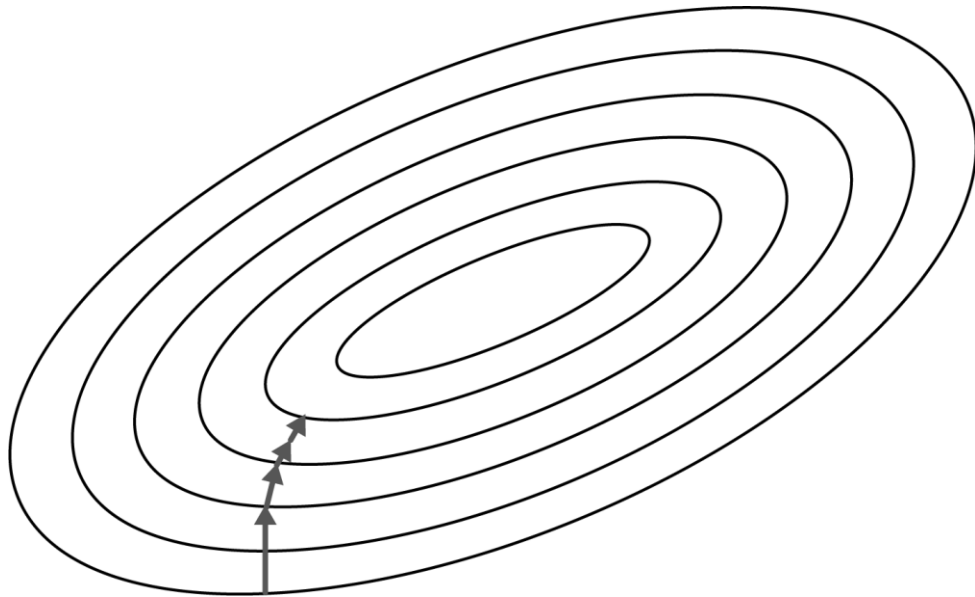
Height represents the loss: high means bad, low means better.

Gradient descent follows the local slope toward lower loss.

A gradient tells us which way to move in weight space, not in input space.

Gradient Descent

The Same Surface Seen from Above



Each ellipse connects weight settings with equal loss.

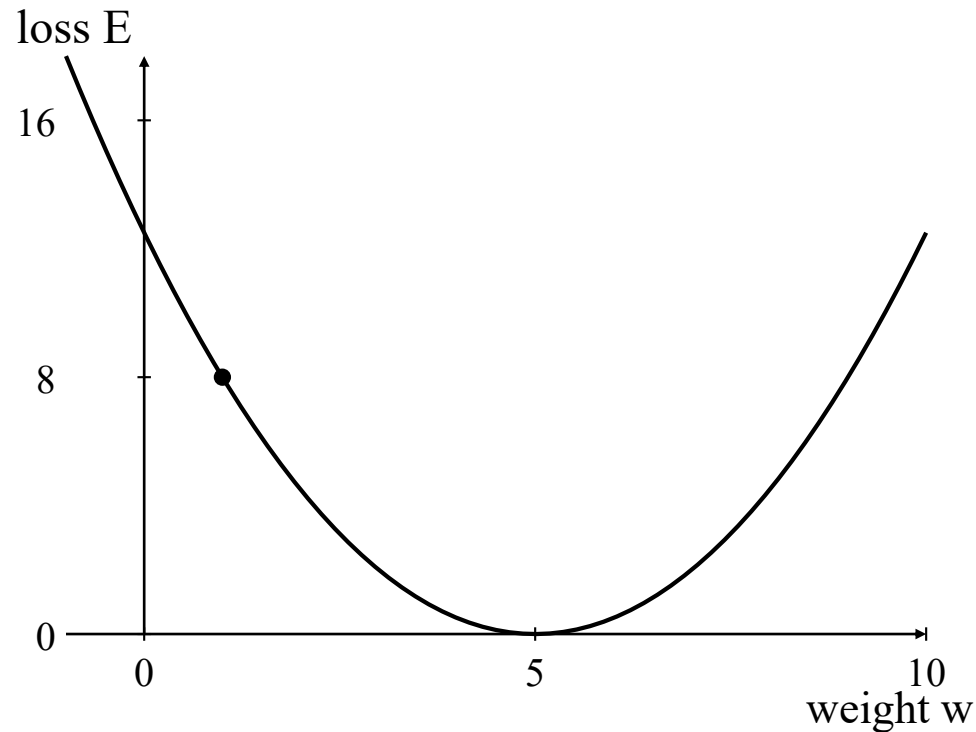
Moving toward inner contours lowers the loss in this example.

The arrows show a sequence of weight updates.

Points on one contour have the same loss.

Gradient Descent

Begin with Only One Adjustable Weight



Suppose $E(w) = \frac{1}{2}(w - 5)^2$.

At $w = 1$, the loss is 8.

Moving toward $w = 5$ reduces the loss.

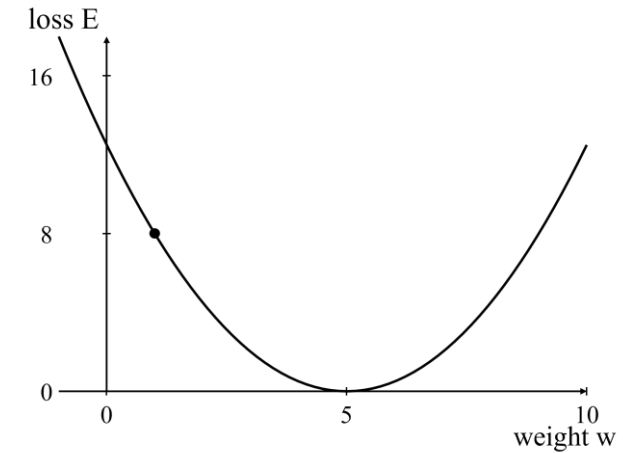
The target here is the bottom of the curve, not a particular input value.

A one-weight curve makes the direction of improvement easy to see.

Gradient Descent

A Derivative Measures Local Sensitivity

$$E(w) = \frac{1}{2}(w - 5)^2 \quad dE/dw = w - 5$$



Current weight	Derivative	Local interpretation
1	-4	Increasing w lowers E nearby.
5	0	The curve is flat at its minimum.
7	+2	Increasing w raises E nearby.

Ask how the loss changes when one weight changes a little.

Gradient Descent

From One Derivative to a Gradient

$$\nabla E = (\partial E / \partial w_1, \partial E / \partial w_2, \dots)$$

A partial derivative changes one weight while holding the others fixed.

For example, $\partial E / \partial w_1$ asks about the burger-price weight only.

The derivative sign shows which way loss changes; magnitude shows its sensitivity.

The gradient combines these sensitivities into one vector.

A gradient collects one partial derivative for each weight.

Gradient Descent

Why Do We Subtract the Gradient?

GRADIENT ∇E

The local direction of fastest increase in the loss.

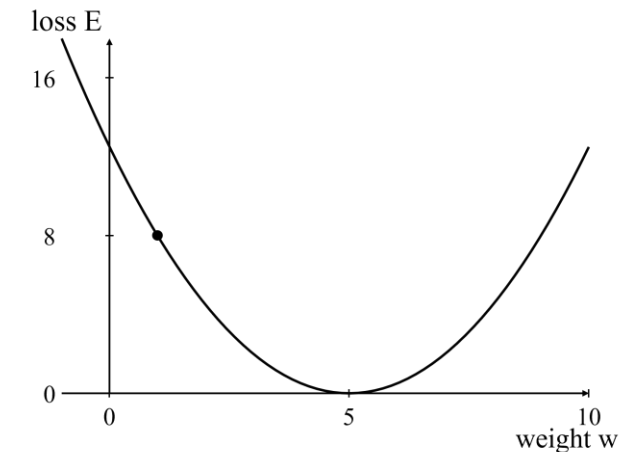
Following it would go uphill.

NEGATIVE GRADIENT $-\nabla E$

The opposite direction.

A sufficiently small step moves downhill.

$$W_{\text{new}} = W_{\text{old}} - \epsilon \nabla E$$



Use the negative gradient to move toward smaller loss.

Gradient Descent

One Complete Gradient-Descent Step

1. Evaluate the derivative at the current weight

$w = 1$, so $dE/dw = 1 - 5 = -4$.

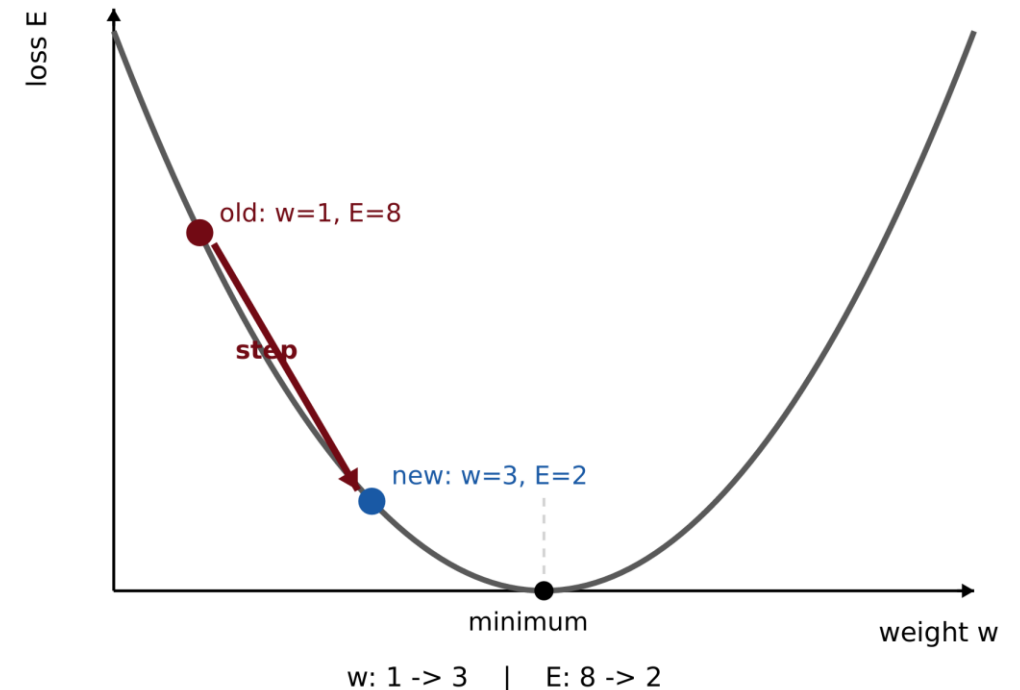
2. Choose a learning rate

Use $\epsilon = 0.5$ for this simple quadratic example.

3. Update and check

w becomes $1 - 0.5 \times (-4) = 3$.

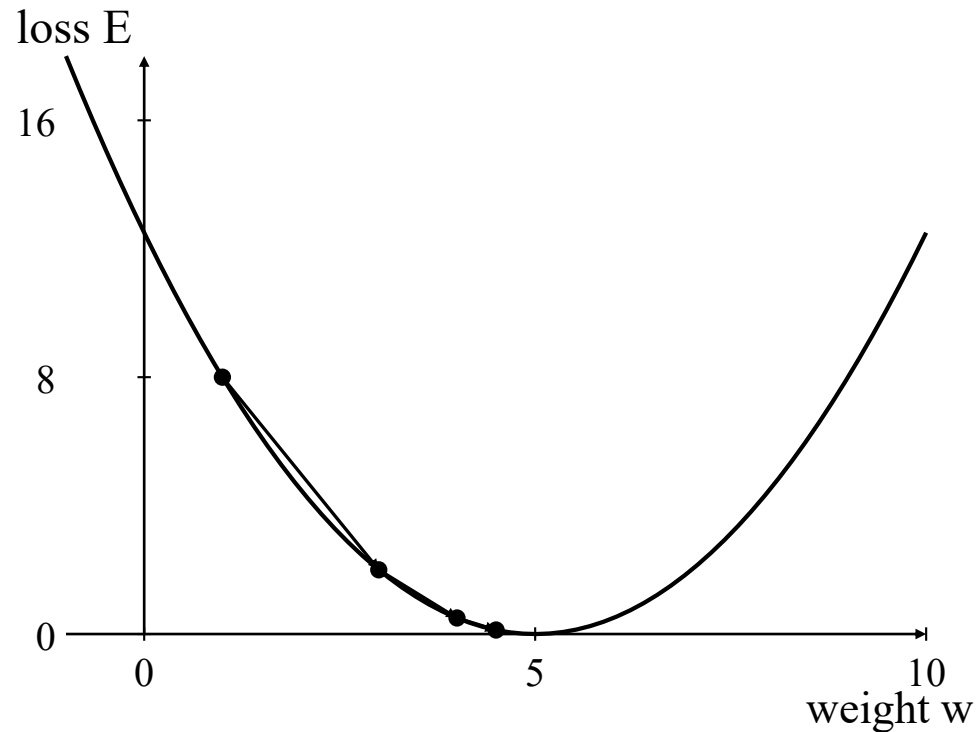
The loss falls from 8 to 2.



A negative derivative makes this weight increase when we subtract it.

Gradient Descent

Repeat: Recompute the Slope After Each Step



Step	w	E
0	1	8
1	3	2
2	4	0.5
3	4.5	0.125

Do not reuse the slope from $w = 1$ forever.

The gradient is evaluated again at the new weights.

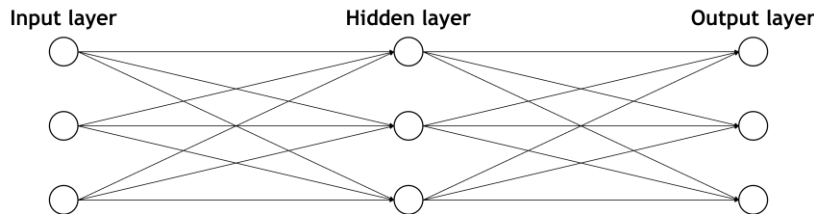
Learning Rates

A Parameter and a Hyperparameter Are Different

PARAMETER: w

A value inside the prediction formula.

Example: the current burger-price guess.



HYPERPARAMETER: ϵ

A setting of the training procedure.

It multiplies the gradient to control the step.

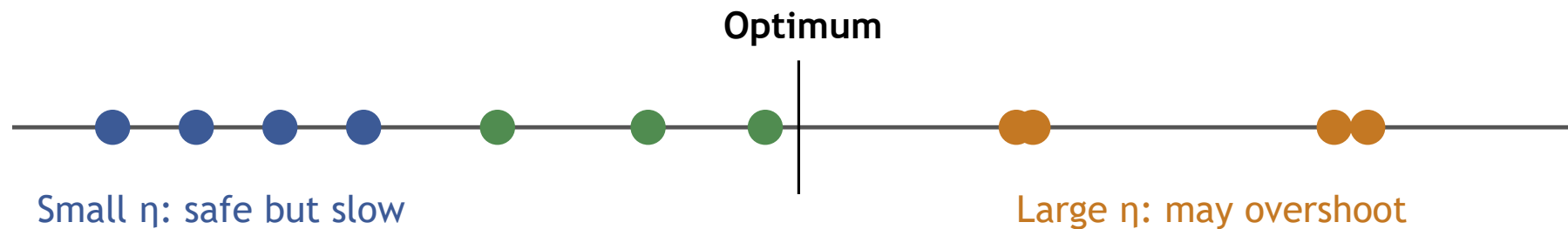
$$\text{weight change} = - \text{learning rate} \times \text{gradient}$$

Training updates the weights; the learning rate controls the update size.

Learning Rates

Learning Rate Means Step Size

Good η : faster progress



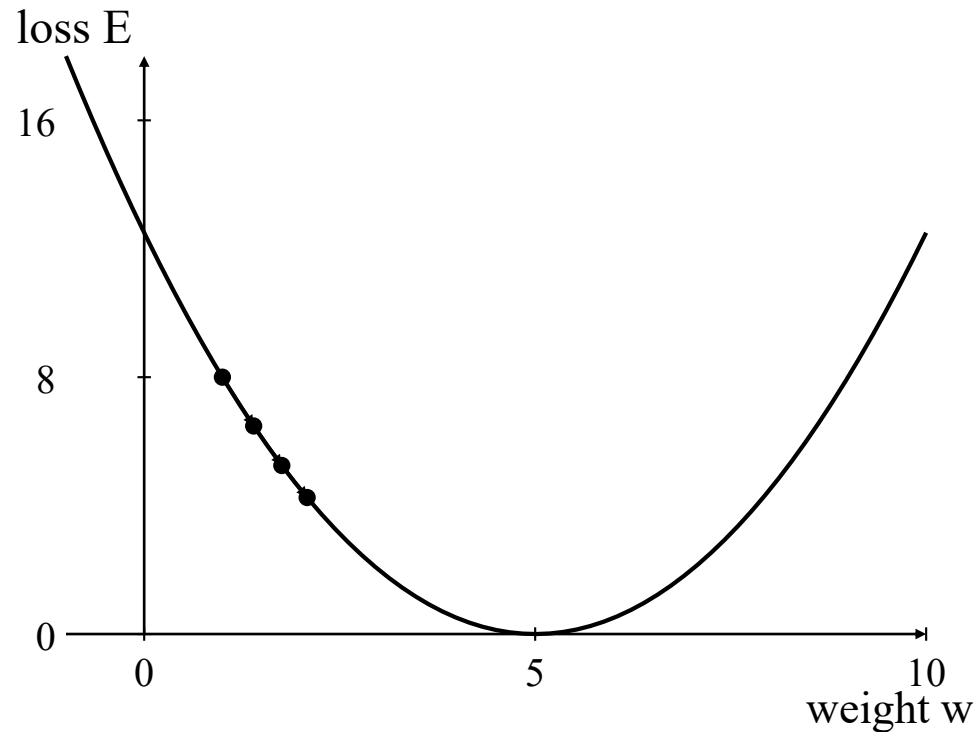
The learning rate does not tell us where to go.

The gradient gives the direction, and the learning rate sets the step length.

A useful learning rate is large enough to move, but small enough to stay stable.

Learning Rates

A Small Step Can Be Slow



Use the same loss and start at $w = 1$.

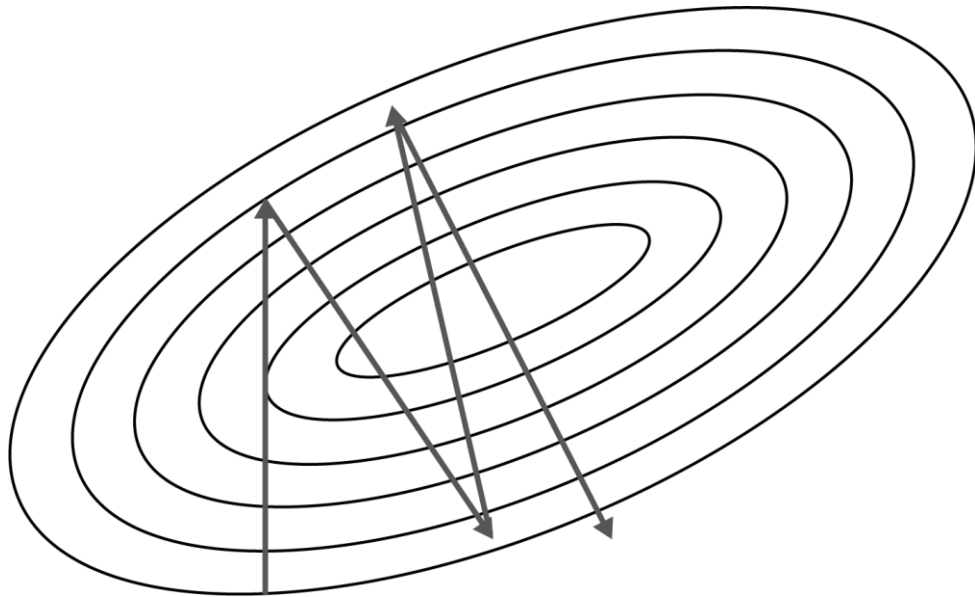
With $\epsilon = 0.1$, the weights become 1.4, 1.76, and 2.084.

The loss decreases, but the weights approach 5 slowly.

A small learning rate may require many updates.

Learning Rates

A Larger Step Is Not Always Better



A large update can cross past the low-loss region.

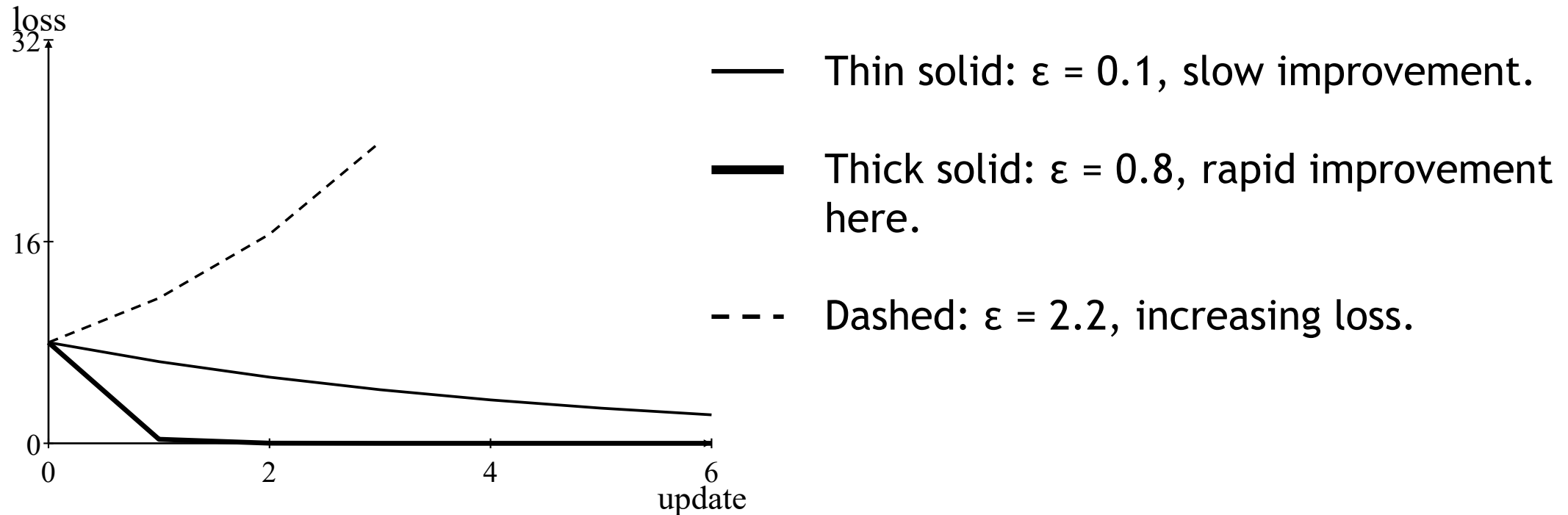
The next update may cross back even farther.

A learning rate that is too large can increase the loss instead of reducing it.

Overshooting can make a sequence of updates unstable.

Learning Rates

Compare Three Step Sizes on the Same Curve



The useful learning-rate scale depends on the loss being optimized.

The Delta Rule

Turn the Gradient into a Weight-Update Formula

$$\Delta w_k = \varepsilon \sum_i x_k^{(i)}(t^{(i)} - y^{(i)})$$

For one example, multiply its residual by the corresponding input value.

Multiply by the learning rate. Add contributions across examples.

Then set $w_{k,\text{new}} = w_{k,\text{old}} + \Delta w_k$.

The delta rule is gradient descent applied to a linear neuron with squared error.

The Delta Rule

Where Does the Linear Update Come From?

$$\partial E / \partial y = y - t$$

The prediction is compared with the target.

$$\partial y / \partial w_k = x_k$$

The input scales the effect of its weight.

$$\partial E / \partial w_k = (y - t)x_k$$

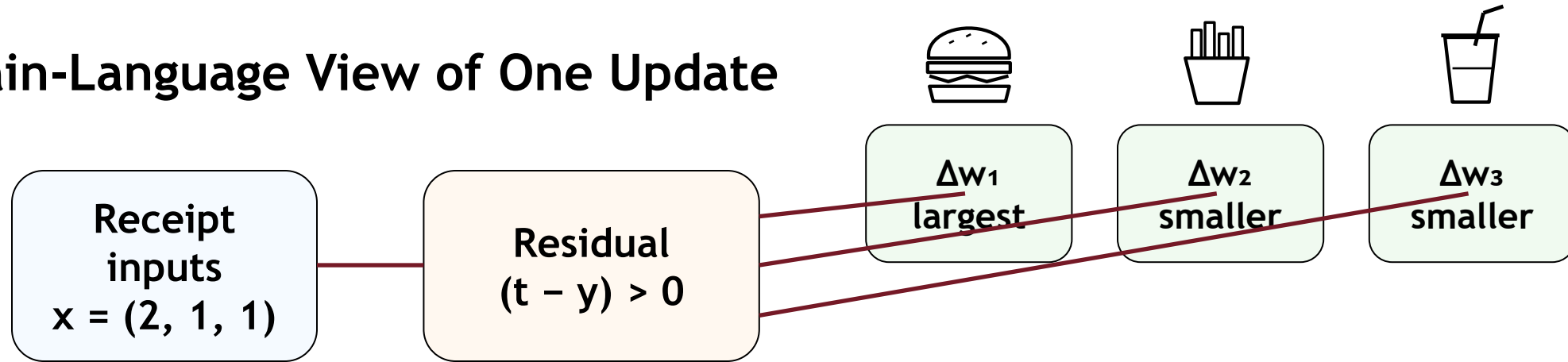
Multiply the two effects, then subtract ε times this gradient.

For one training example. Add these gradients when using a batch.

Error sensitivity and input sensitivity combine through the chain rule.

The Delta Rule

Plain-Language View of One Update



If the model underpredicts and an input is positive, the corresponding weight should increase.

The same residual reaches every weight, but each gradient is scaled by that input's size.

The delta rule says: use the error signal, then allocate a bigger change to a bigger input.

The Delta Rule

One Receipt: Compute the Three Gradients

$$\mathbf{x} = (2, 1, 3), \quad \mathbf{w} = (4, 1, 1), \quad t = 15$$

$$y = 12, \text{ so } y - t = -3$$

Weight	Input	Gradient $(y - t)x$
Burger	2	$-3 \times 2 = -6$
Fries	1	$-3 \times 1 = -3$
Soda	3	$-3 \times 3 = -9$

The same residual affects each weight, scaled by that weight's input.

The Delta Rule

Update All Three Price Guesses

$$W_{\text{new}} = W_{\text{old}} - 0.05 \times \text{gradient}$$

$$\text{Burger: } 4 \qquad - 0.05 \times (-6) \qquad = 4.30$$

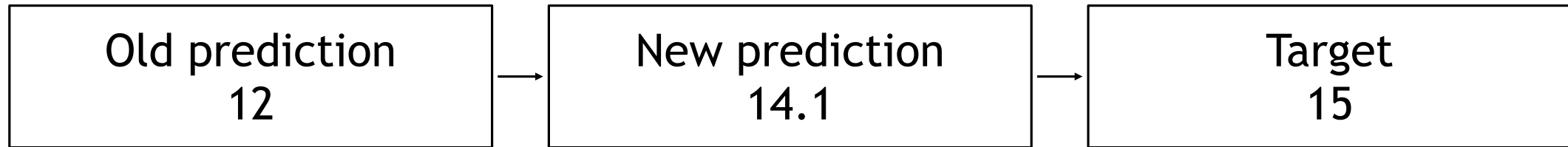
$$\text{Fries: } 1 \qquad - 0.05 \times (-3) \qquad = 1.15$$

$$\text{Soda: } 1 \qquad - 0.05 \times (-9) \qquad = 1.45$$

All weights use gradients computed at the same old parameter values.

The Delta Rule

Does the New Prediction Improve?



$$y_{\text{new}} = 2(4.30) + 1(1.15) + 3(1.45) = 14.1$$

The residual decreases from 3 to 0.9.

For this receipt, the loss decreases from 4.5 to 0.405.

After updating, run the forward calculation again to measure the effect.

The Delta Rule

Read the Sign Before Doing the Arithmetic

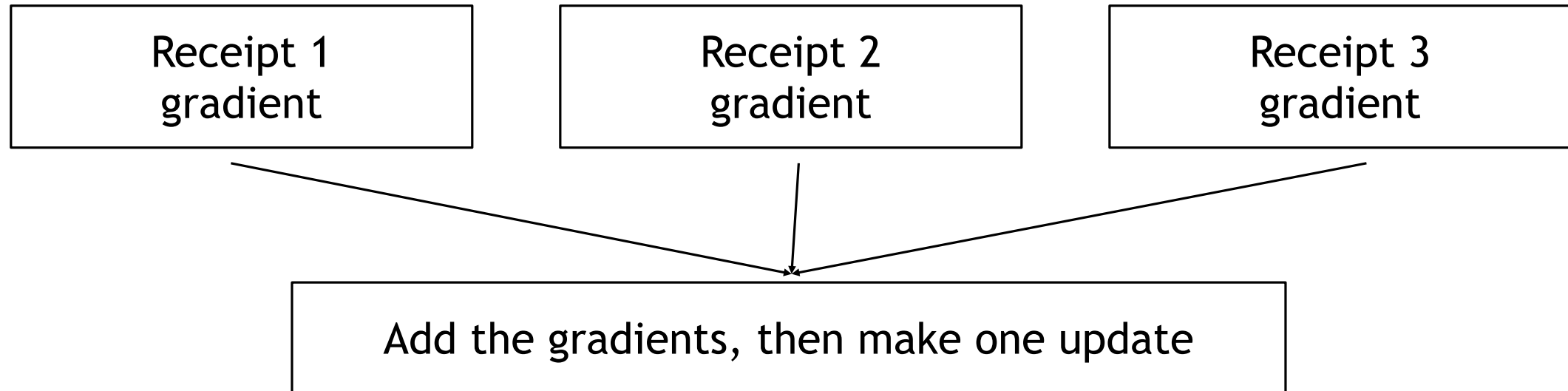
Case for a positive input	Residual $t - y$	Delta-rule change
Prediction too small	Positive	Increase the weight.
Prediction too large	Negative	Decrease the weight.
Prediction matches target	Zero	No change from this example.

If this input is zero, this example contributes zero to that weight's gradient.

For positive inputs, underprediction increases the corresponding weights.

The Delta Rule

Several Receipts: Add the Contributions



Keep the weights fixed while calculating the contributions for this batch.

A full-batch update aggregates gradients before changing the weights.

The Delta Rule

Quick Check: Which Way Should the Weight Move?

$$x = 2, \quad w = 4, \quad t = 10, \quad \varepsilon = 0.1$$

1. Predict

What is $y = wx$? Is it too high or too low?

2. Update

Use $\Delta w = \varepsilon x(t - y)$. What is the new weight?

Use the residual sign to predict the update direction.

The Delta Rule

Quick Check: Work Through the Answer

Prediction

$y = 4 \times 2 = 8$. The target is 10, so the residual is 2.

Weight change

$\Delta w = 0.1 \times 2 \times 2 = 0.4$. The new weight is 4.4.

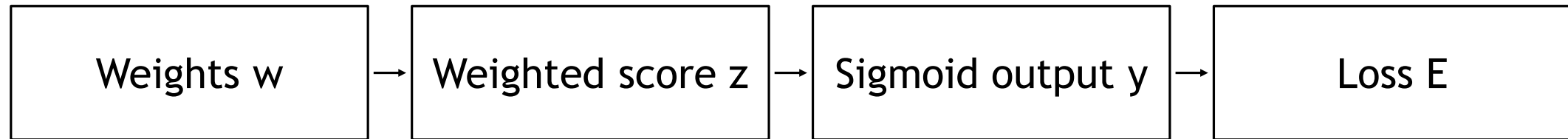
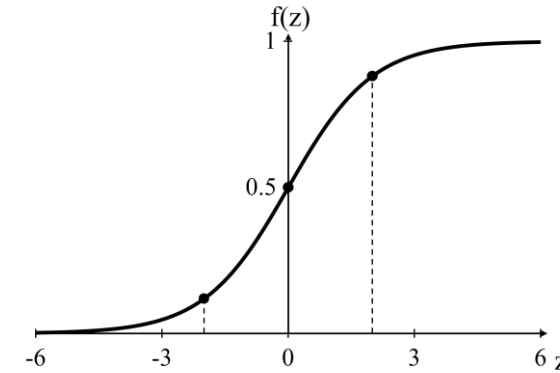
New prediction

y becomes $4.4 \times 2 = 8.8$, closer to the target 10.

The weight increases because this positive input was underpredicted.

Sigmoidal Neurons

Gradient Descent with Sigmoidal Neurons



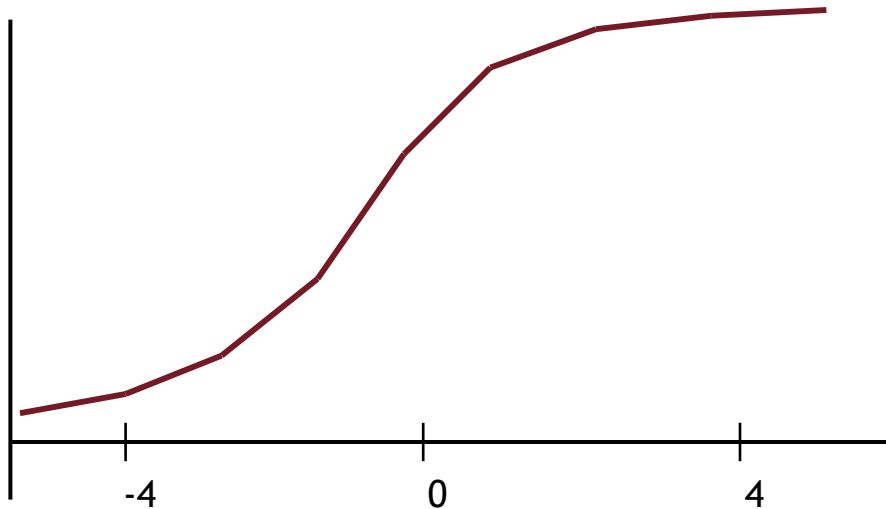
$$z = \sum_k w_k x_k \quad y = 1 / (1 + \exp(-z))$$

A weight affects z , which affects y , which affects E .

The output now passes through a nonlinear function before the loss is measured.

Sigmoidal Neurons

A Sigmoid Neuron Behaves Like a Soft Switch



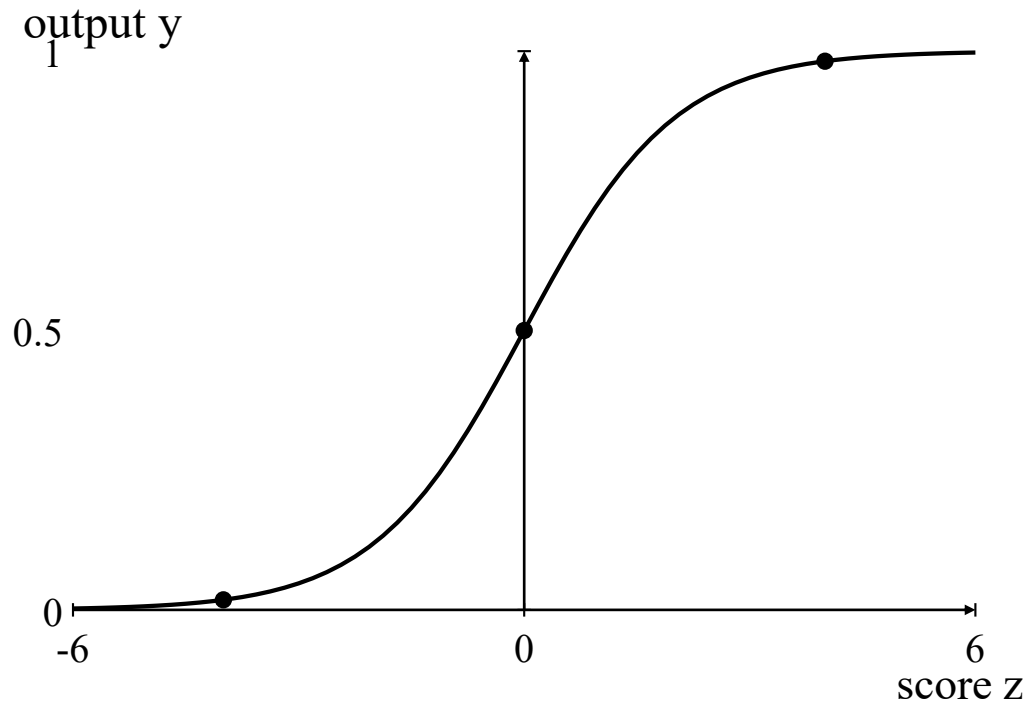
Large negative score → output near 0
Score near 0 → output changes quickly
Large positive score → output near 1

Near the middle, a small score change matters a lot. Near the ends, the same score change matters less.

Sigmoid is not a hard threshold; it is a smooth transition from “off” to “on”.

Sigmoidal Neurons

The Sigmoid Is Most Sensitive Near the Middle



Near $z = 0$, the output changes noticeably when z changes.

Far from zero, the curve is flatter.

The derivative quantifies this sensitivity.

A small score change has less effect when the sigmoid curve is almost flat.

Sigmoidal Neurons

A Convenient Derivative

$$dy/dz = y(1 - y)$$

Sigmoid output y	Derivative $y(1 - y)$	Interpretation
0.50	0.2500	More sensitive.
0.99	0.0099	Nearly flat.
0.01	0.0099	Nearly flat.

The sigmoid derivative can be computed from its output: $y(1 - y)$.

Sensitivity measures how much the output y changes when the score z changes slightly.

Sigmoidal Neurons

Separate the Three Local Effects

$$\partial z / \partial w_k = x_k$$

Weight $\rightarrow$ weighted score.

$$dy/dz = y(1 - y)$$

Weighted score $\rightarrow$ sigmoid output.

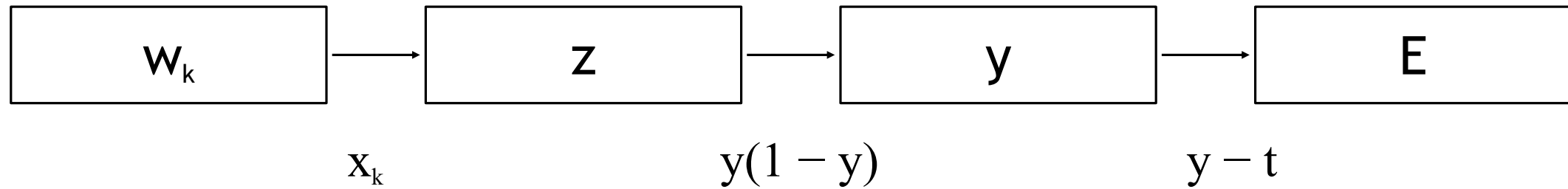
$$\partial E / \partial y = y - t$$

Output $\rightarrow$ squared error.

Each derivative answers one small “what changes next?” question.

Sigmoidal Neurons

The Chain Rule Multiplies Along the Path



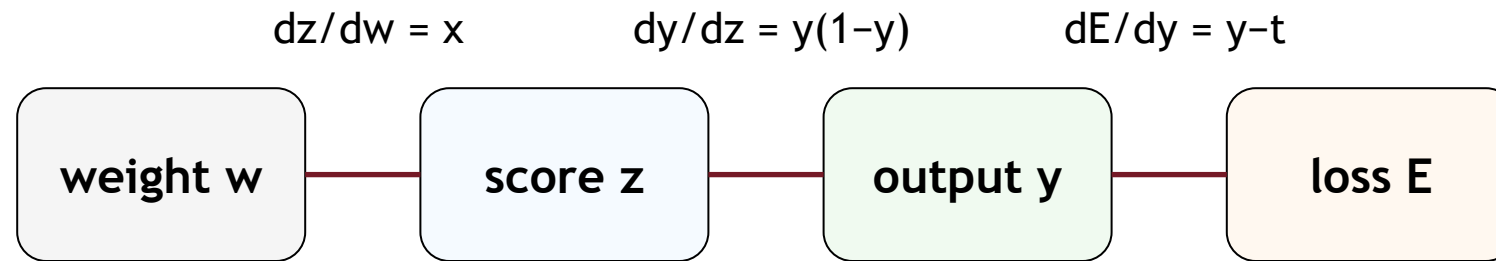
$$\partial E / \partial w_k = (y - t) \times y(1 - y) \times x_k$$

Multiply derivatives along a chain.

Combine local sensitivities to obtain the weight's effect on the loss.

Sigmoidal Neurons

Chain Rule = Multiply Small Local Effects



Multiply the local effects: $dE/dw = (dE/dy)(dy/dz)(dz/dw)$

Backpropagation works because a long dependency can be broken into small, reusable pieces.

Sigmoidal Neurons

One Sigmoid Update with Small Numbers

$$x = 2, \quad w = 0, \quad t = 1, \quad \varepsilon = 1 \qquad z = \sum wx \qquad y = 1 / (1 + \exp(-z))$$

Forward

$$z = 0, \quad y = 0.5, \quad \text{and } E = \frac{1}{2}(1 - 0.5)^2 = 0.125.$$

Backward

$$\text{The gradient is } (0.5 - 1) \times 0.5 \times 0.5 \times 2 = -0.25. \qquad \partial E / \partial w = (y - t) \times y(1 - y) \times x$$

Update and check

w becomes 0.25. The new output is about 0.622, and E is about 0.071.

The update is still “old weight minus learning rate times gradient.”

Sigmoidal Neurons

Compare the Linear and Sigmoid Updates

LINEAR NEURON

$$\Delta w_k = \varepsilon x_k (t - y)$$

SIGMOID NEURON

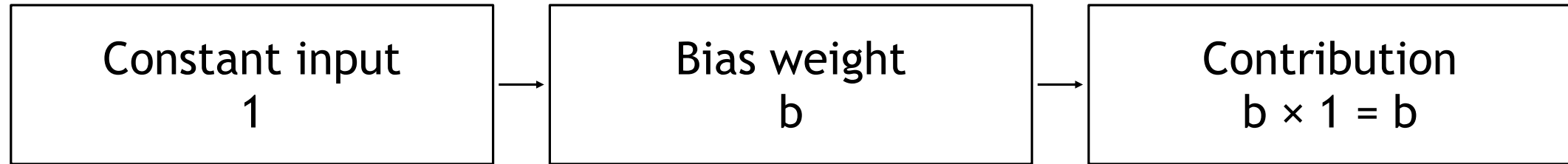
$$\Delta w_k = \varepsilon x_k y (1 - y) (t - y)$$

These equations show one example. For a batch, add its examples' contributions.

The extra $y(1 - y)$ factor accounts for the sigmoid's local sensitivity.

Sigmoidal Neurons

Where Did the Bias Go?



The chapter omits bias terms to keep the derivation short.

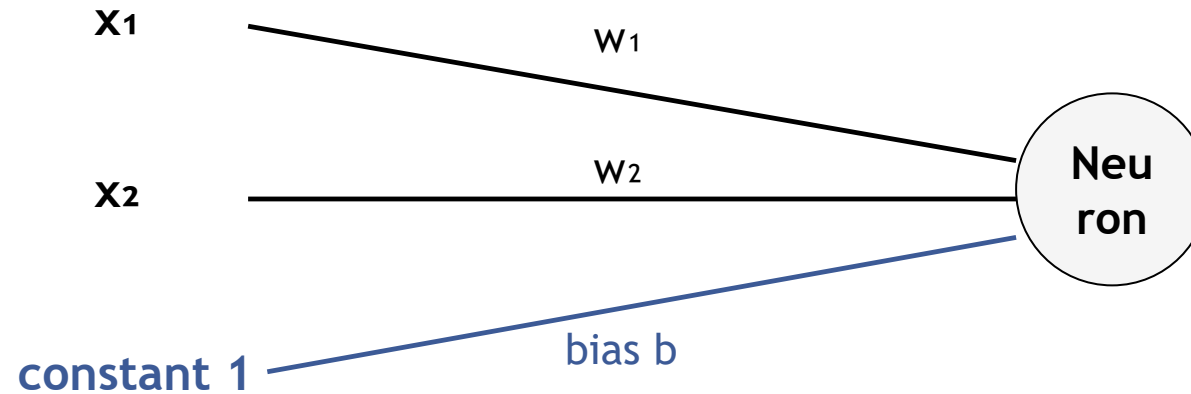
Adding a fixed input of 1 brings the bias into the same calculation.

For a sigmoid, $\partial E / \partial b = (y - t)y(1 - y)$.

A bias can be treated as a weight on an input that is always 1.

Sigmoidal Neurons

Bias = A Weight on a Constant Input



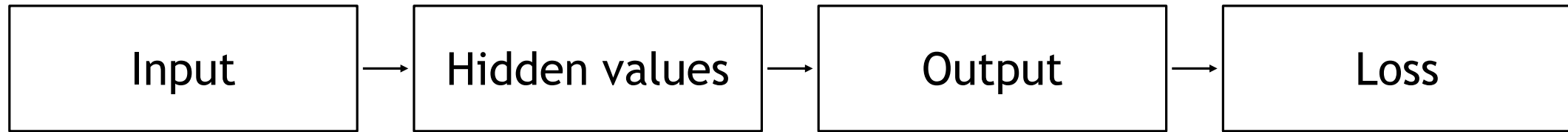
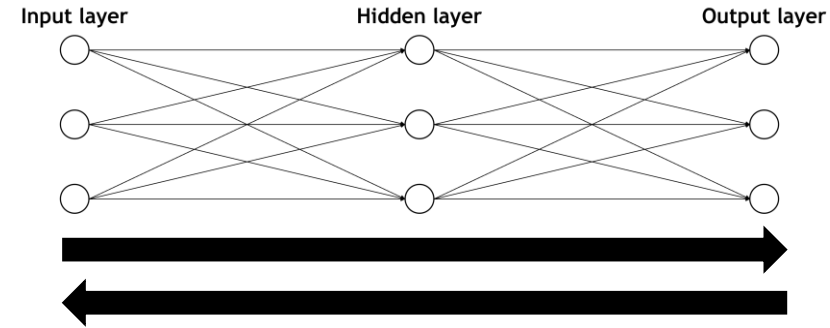
Treat the bias as a weight on an input that is always equal to 1.

Then the same update logic applies to the bias and the ordinary weights.

Bias shifts the neuron's score even when the ordinary inputs are zero.

Backpropagation

The Backpropagation Algorithm



We know the final target, but we do not supply a target for every hidden neuron.

Instead, ask how a small change in a hidden value would change the final loss.

Backpropagation computes these sensitivities from the output layer backward.

A hidden weight matters because it changes later computations and the final loss.

Summary

Quick Check

1. Prediction or training?

Which values change when the same receipt is used for a new training step?

2. Direction of a step

If $\partial E / \partial w$ is negative, does gradient descent increase or decrease w ?

Explain what changes and why the update can reduce the loss.

Summary

Explanations

1. Change the model parameters

The receipt's quantities and target stay fixed. New weights give new predictions.

2. Subtract a negative number

The weight increases. For a small enough step, this moves in a local descent direction.

Training connects prediction errors to specific changes in the weights.

Summary

What Did We Learn?

Define a loss

Measure the mismatch between predictions and known targets.

Follow a useful update direction

Use the negative gradient, scaled by the learning rate.