Deep Learning for Computer Vision

Lecture 5: Logistic Regression, the Joy of Backpropagation, and the Perceptron

Peter Belhumeur

Computer Science
Columbia University

Binary Classifier

Let's say we have set of training samples $(\mathbf{x}_\mathrm{i}, y_\mathrm{i})$ with $\mathbf{i} = 1, \ldots, \mathbf{N}$ and $\mathbf{x}_\mathrm{i} \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$.

The goal is to learn a classifier $\mathbf{f}(\mathbf{x}_\mathrm{i})$ such that

$$f(\mathbf{x}_i) = \begin{array}{ll} \geq 0 & y_i = +1 \\ < 0 & y_i = -1 \end{array}$$

$f(\mathbf{x}) \geq 0$

$f(\mathbf{x}) = 0$

$f(\mathbf{x}) < 0$

But let’s find a simple parametric separating surface by fitting to the training data using some pre-chosen criterion of optimality.

If we restrict the decision surface to a line, plane, or hyperplane, then we call this a **linear classifier**.

Linear Classifier

Let's say we have set of training samples $(\mathbf{x}_\mathrm{i}, y_\mathrm{i})$ with $\mathbf{i} = 1, \ldots, \mathbf{N}$ and $\mathbf{x}_\mathrm{i} \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$.

The goal is to learn a classifier $\mathbf{f}(\mathbf{x}_\mathrm{i})$ such that

$$f(\mathbf{x}_i) = \begin{array}{ll} \geq 0 & y_i = +1 \\ < 0 & y_i = -1 \end{array}$$

and where

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

Linear Classifier

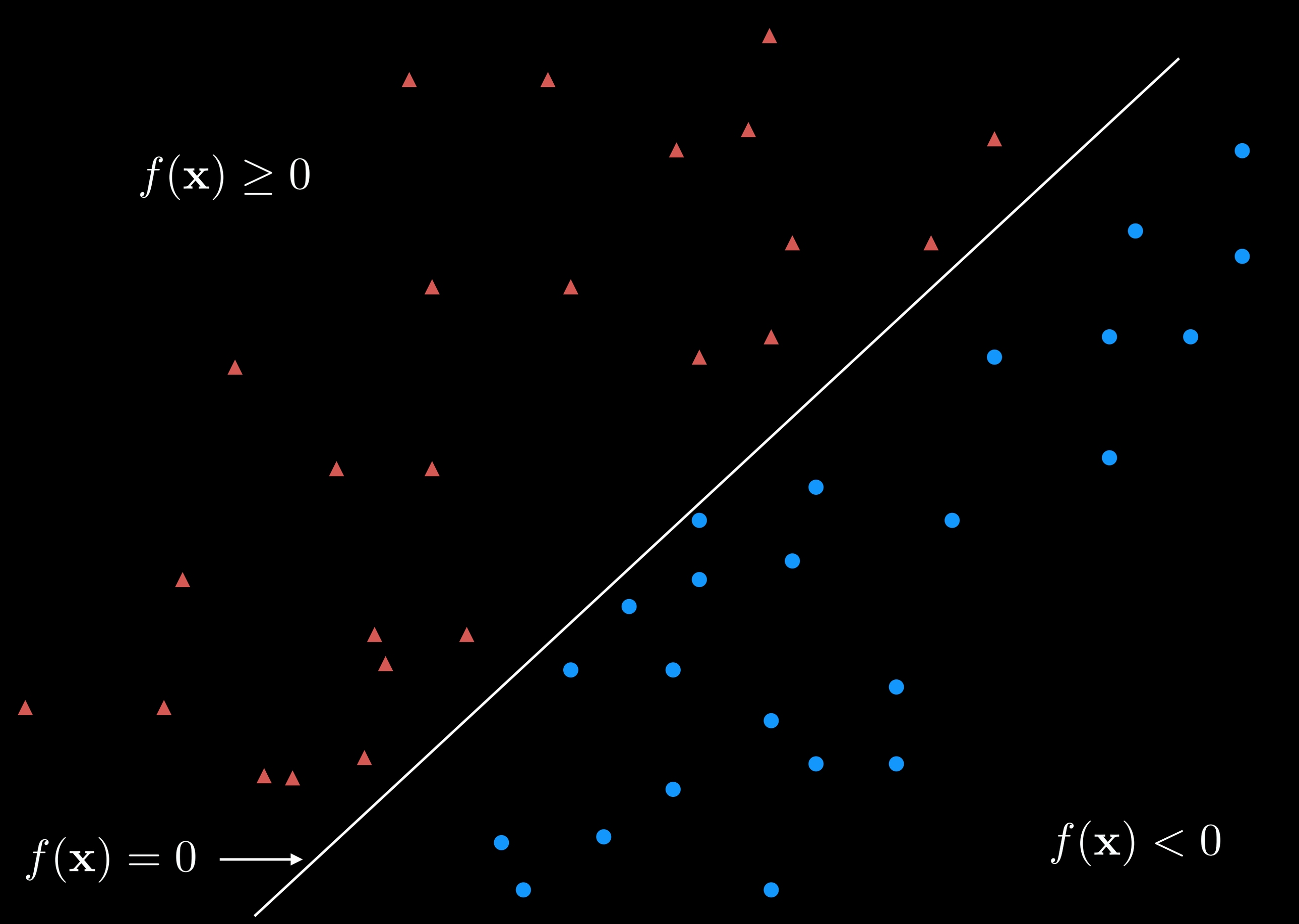
Note:

- The classifier is a line, plane, or hyperplane depending of the dimension $\mathbf{d}$

- $\mathbf{w}$ is normal to the line

- b is known as the bias.

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$


$f(\mathbf{x}) \geq 0$
$f(\mathbf{x}) = 0$
$f(\mathbf{x}) < 0$

$$\mathbf{w}^T\mathbf{x} + b \geq 0$$

$$\frac{\mathbf{w}}{||\mathbf{w}||}$$

$$\mathbf{w}^T\mathbf{x} + b = 0 \longrightarrow$$

$$\mathbf{w}^T\mathbf{x} + b < 0$$

Linear Classifier

How do we find the *weights* given by $\mathbf{w}$ and b?

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

Support Vector Machine

Choose *weights* $\mathbf{w}$ and b to maximize the margin between classes.

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

Linearly Separable Data

$$\mathbf{w}^T\mathbf{x} + b \geq 0$$

margin

$\mathbf{w}$

$$\mathbf{w}^T\mathbf{x} + b = 0$$

$$\mathbf{w}^T\mathbf{x} + b < 0$$

Logistic Regression

Introduces a non-linearity over a linear classifier that results in a different measure for our loss function

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \qquad \text{Linear}$$

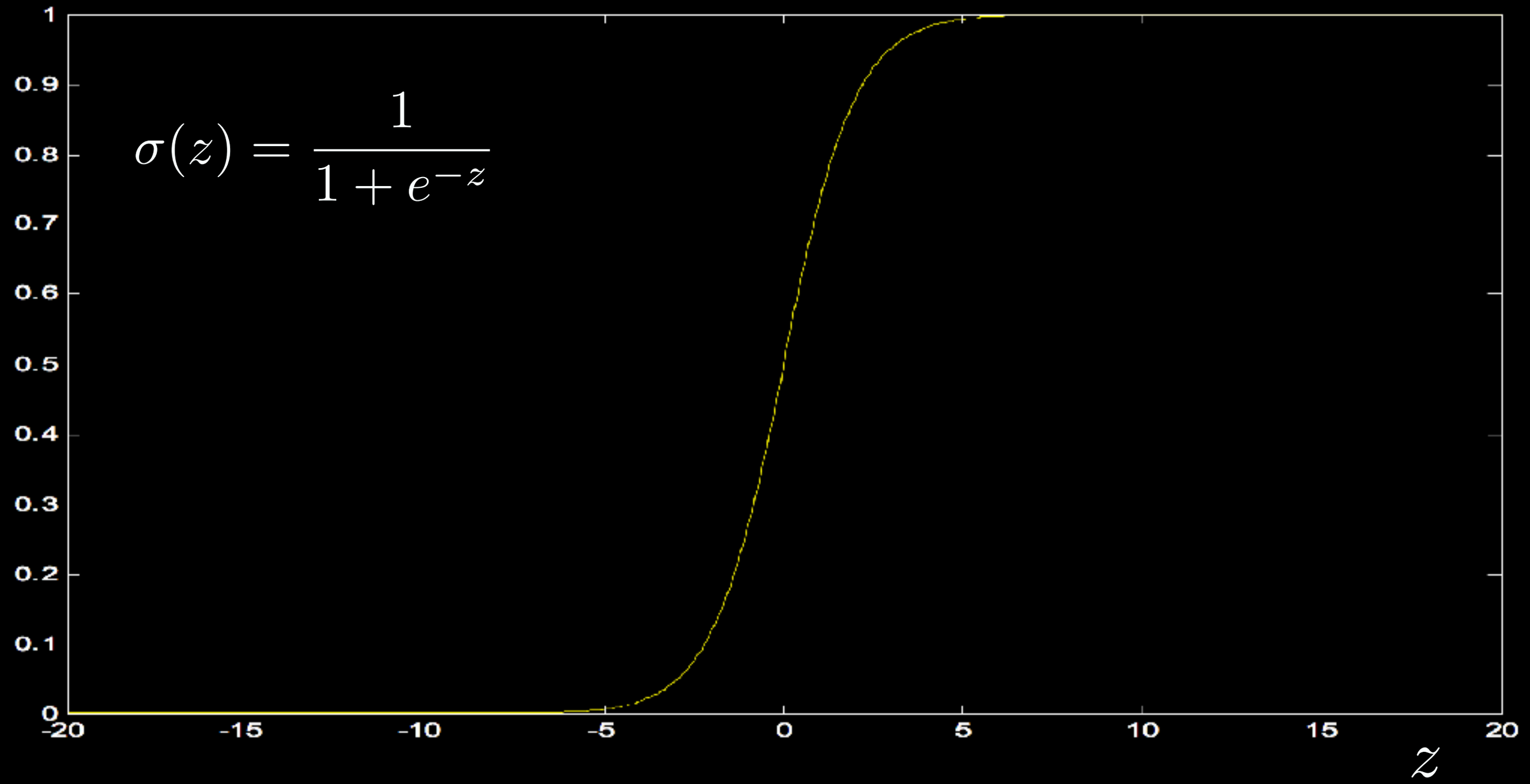
$$\sigma(f(\mathbf{x})) = \frac{1}{1 + e^{-f(\mathbf{x})}} \qquad \text{Sigmoid or Logistic Fcn}$$

The LR classifier is defined as

$$\begin{aligned} \text{if } \sigma(f(\mathbf{x})) \geq 0.5 \quad &\rightarrow \quad +1 \\ < 0.5 \quad &\rightarrow \quad -1 \end{aligned}$$

Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Sigmoid Function

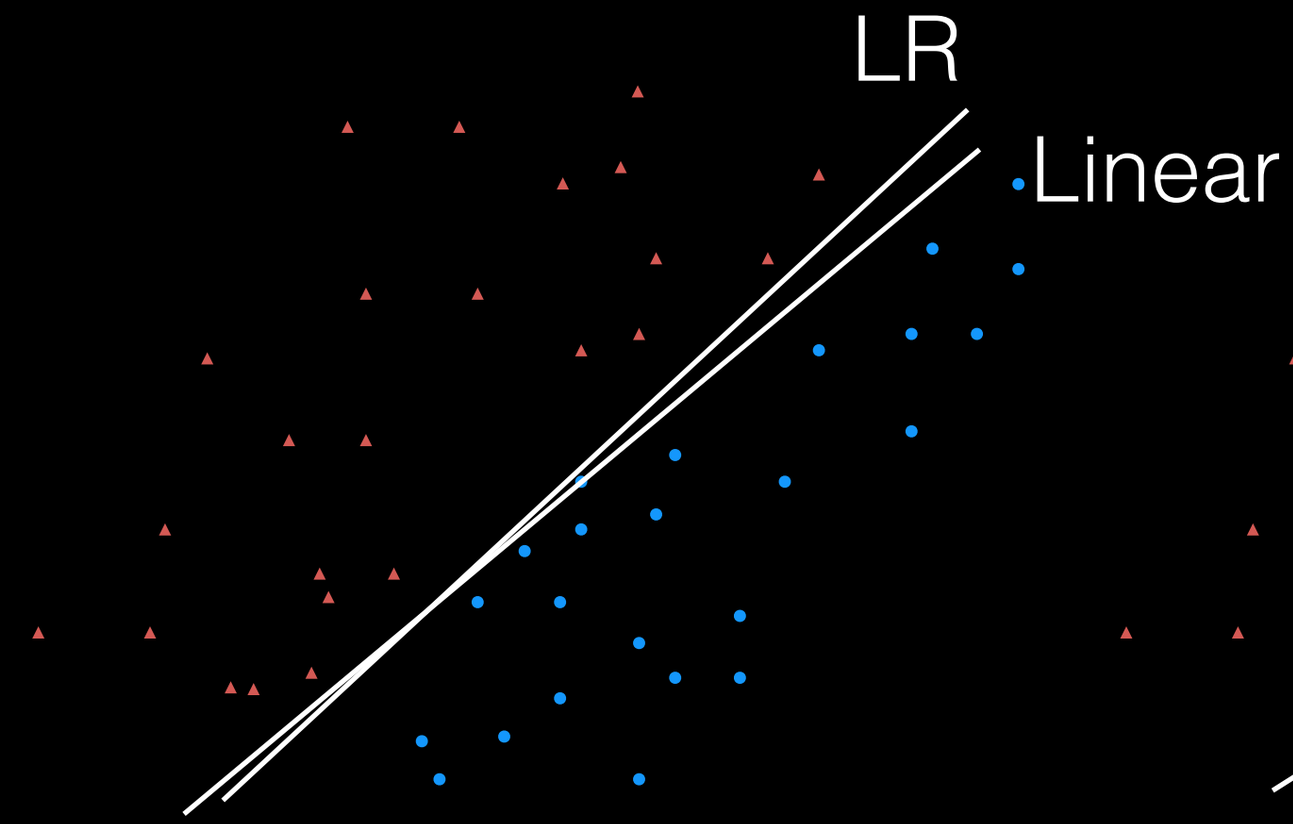
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

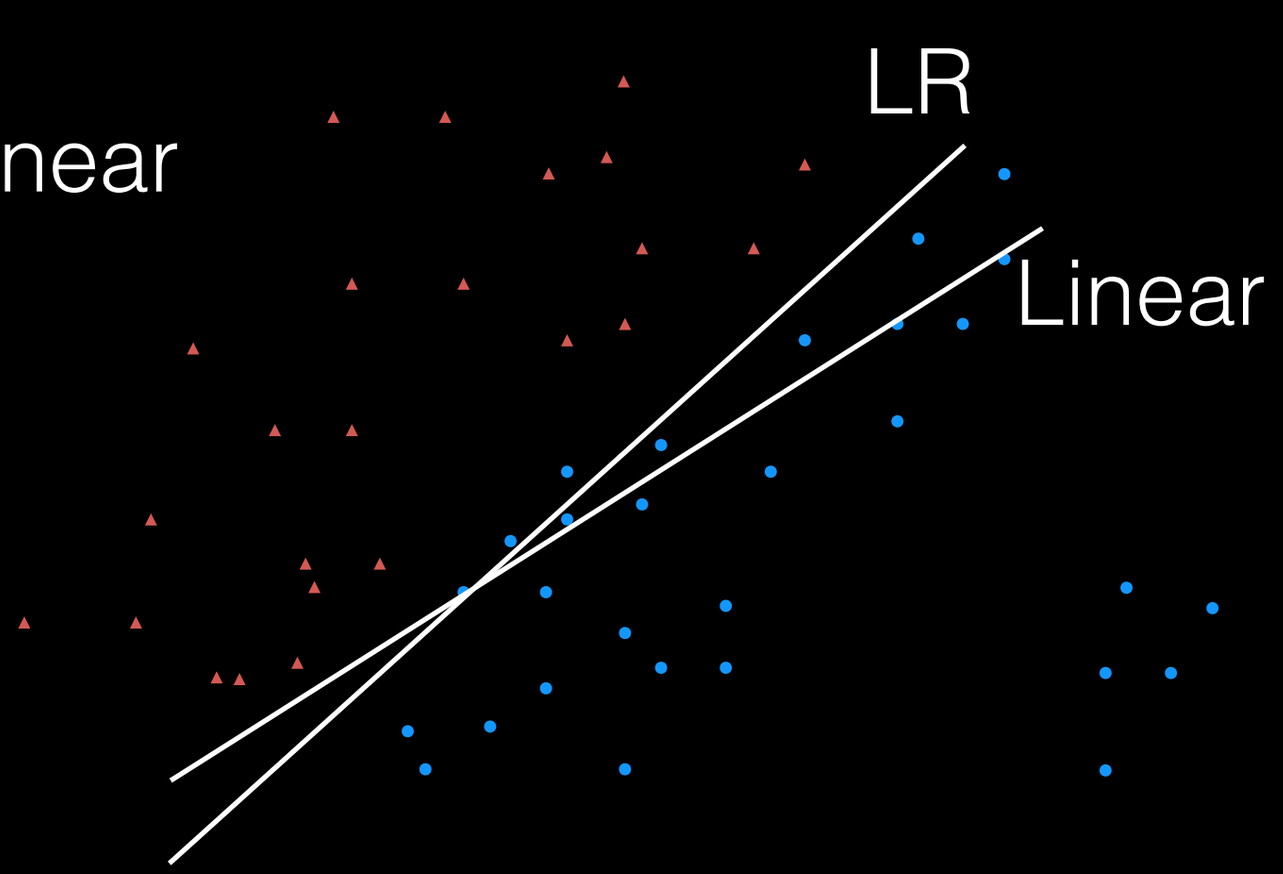
- As z goes from $-\infty$ to ∞, $\sigma(z)$ goes from 0 to 1
- It has a “sigmoid” or S-like shape

$$\sigma(0) = 0.5$$

Why a Sigmoid?

Like the hinge loss for the SVM, the sigmoids down-weights ***distant*** samples that are correctly classified:





Sigmoid maps to “probabilities”

- Recall our Bayesian Classifier from before…
- Let’s think of $\sigma(f(\mathbf{x}))$ as the *posterior* probability that $y = 1$, or more formally

$$P(y = 1|\mathbf{x}) = \sigma(f(\mathbf{x}))$$

- So if $\sigma(f(\mathbf{x})) \geq 0.5$, then class $y = 1$

Loss Function for LR

So for +1 and -1 samples we have

$$P(y=+1|\mathbf{x};\mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x})) = \frac{1}{1+e^{-f(\mathbf{x})}}$$

$$P(y=-1|\mathbf{x};\mathbf{w}) = 1-\sigma(\mathbf{w}^T\mathbf{x})) = \frac{1}{1+e^{+f(\mathbf{x})}}$$

or more concisely

$$P(y|\mathbf{x};\mathbf{w}) = \frac{1}{1+e^{-yf(\mathbf{x})}}$$

Maximum Likelihood Estimate

Now let's say **w** is given and $(\mathbf{x}_i, y_i)$ are independent, then the "likelihood" of the samples is

$$P(\mathbf{y}|\mathbf{x};\mathbf{w}) = \prod_i^N \frac{1}{1+e^{-y_i f(\mathbf{x}_i)}}$$

and we can find the ***Maximum Likelihood Estimate*** (MLE) for **w** by maximizing this joint probability.

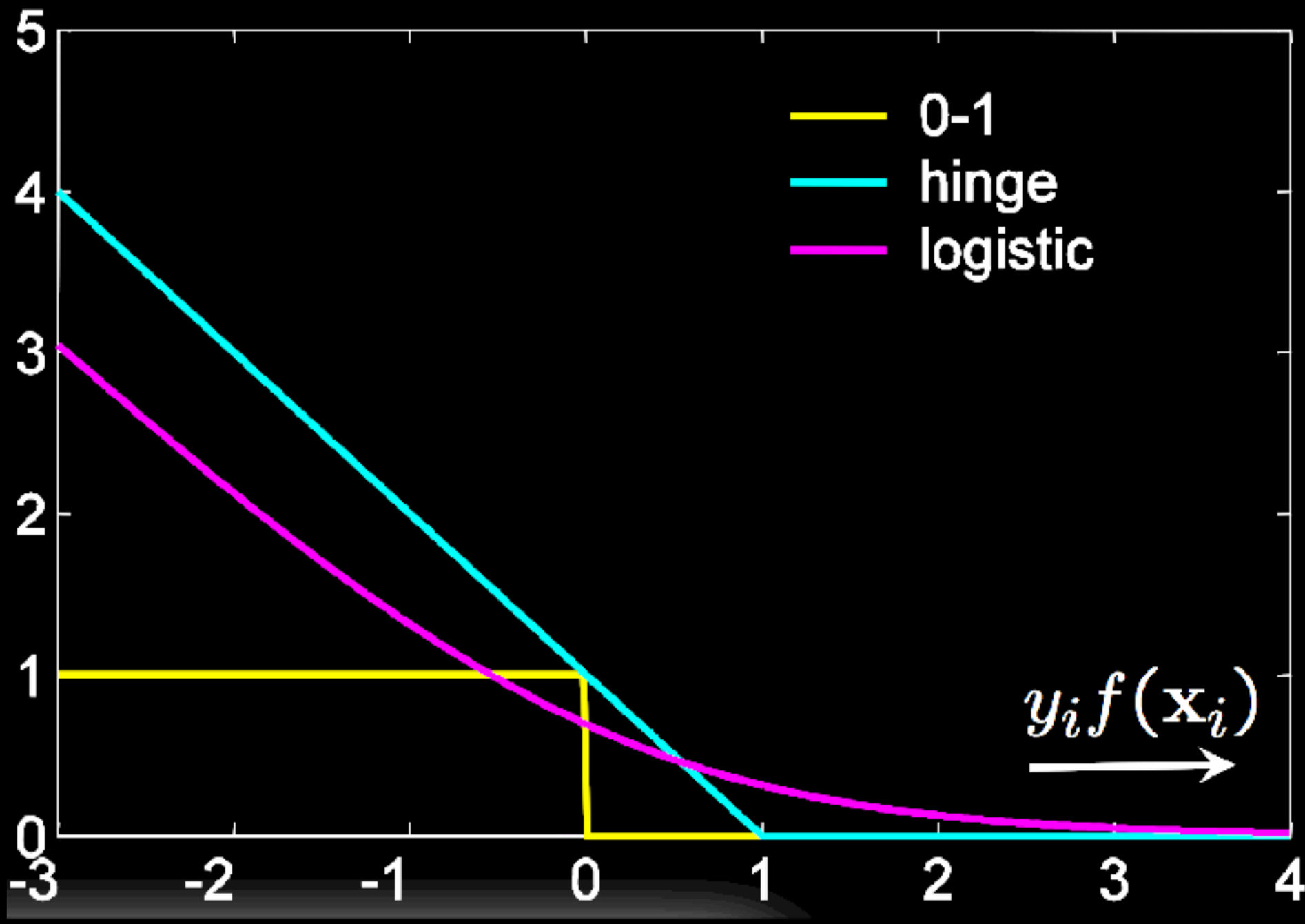
Maximum Likelihood Estimate

Or equivalently we can find **w** by minimizing the negative log likelihood

$$L(\mathbf{w}) = \sum_{i}^{N} \log(1 + e^{-y_i f(\mathbf{x}_i)})$$

As was the case with SVMs, we can think of this function as our loss function. It penalizes for where samples fall relative to the separating hyperplane $f(\mathbf{x}_i) = 0$.

Logistic Loss



Logistic Regression

Our cost function for optimizing can be written:

$$\min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) = \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{N}\sum_{i}^{N} \log(1 + e^{-y_i\, f(\mathbf{x_i})})$$

regularization + loss function

with $\lambda = \frac{2}{NC}$ and $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$

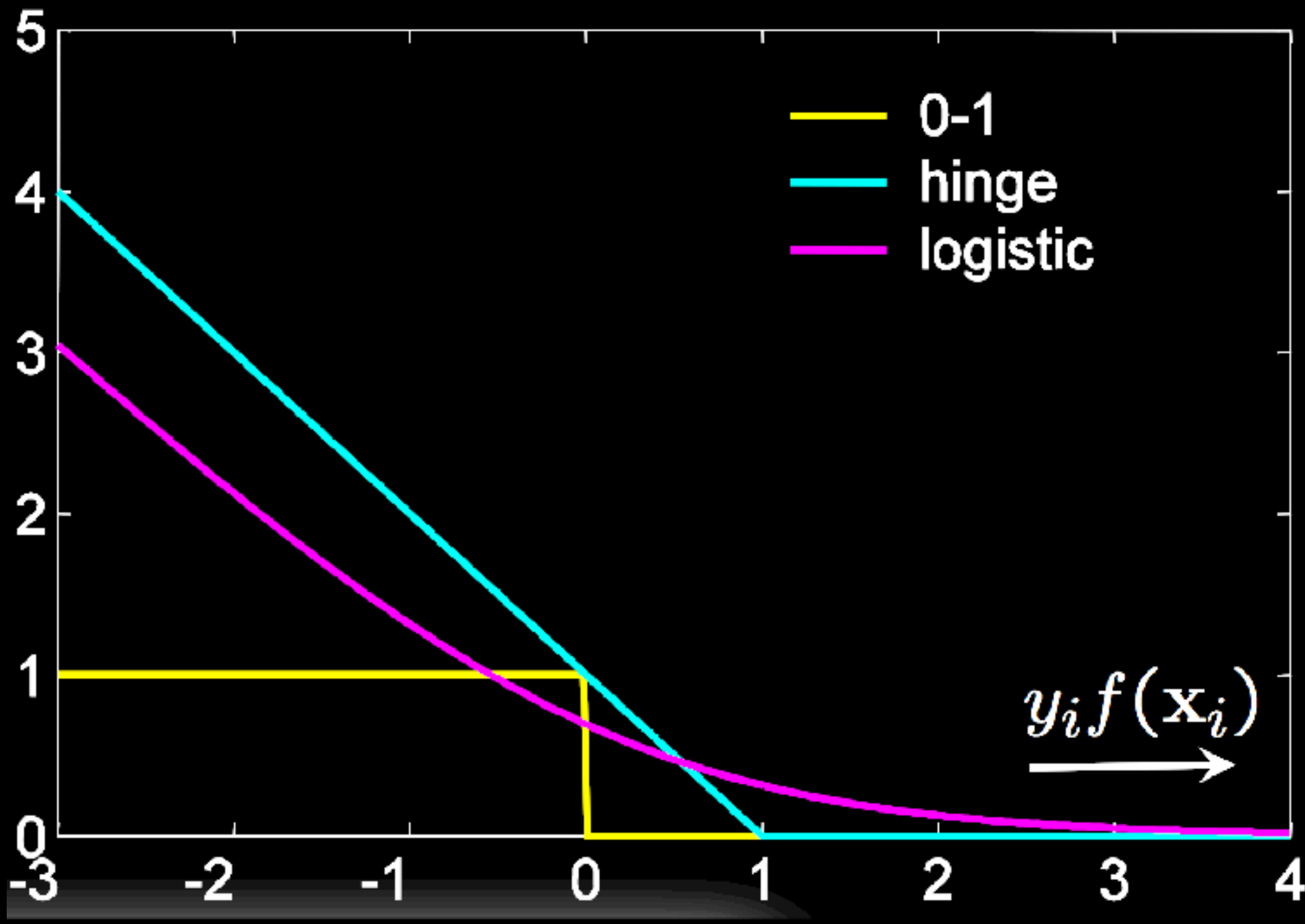
Logistic Regression

Our cost function for optimizing can be written:

$$\min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) = \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{N}\sum_{i}^{N} \log(1 + e^{-y_i\, f(\mathbf{x_i})})$$

- Correctly classified points $-y_i\, f(\mathbf{x_i})$ is negative, loss is near 0
- Incorrectly classified points $-y_i\, f(\mathbf{x_i})$ is positive, loss is large
- The regularization prevents overfitting and widens margin

Logistic Loss



Logistic Regression vs SVM

$$\min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) = \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{N}\sum_{i}^{N} \log(1 + e^{-y_i\, f(\mathbf{x_i})}) \qquad \text{LR}$$

$$\min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) = \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{N}\sum_{i}^{N} \max(0, 1 - y_i\, f(\mathbf{x_i})) \qquad \text{SVM}$$

regularization + loss function

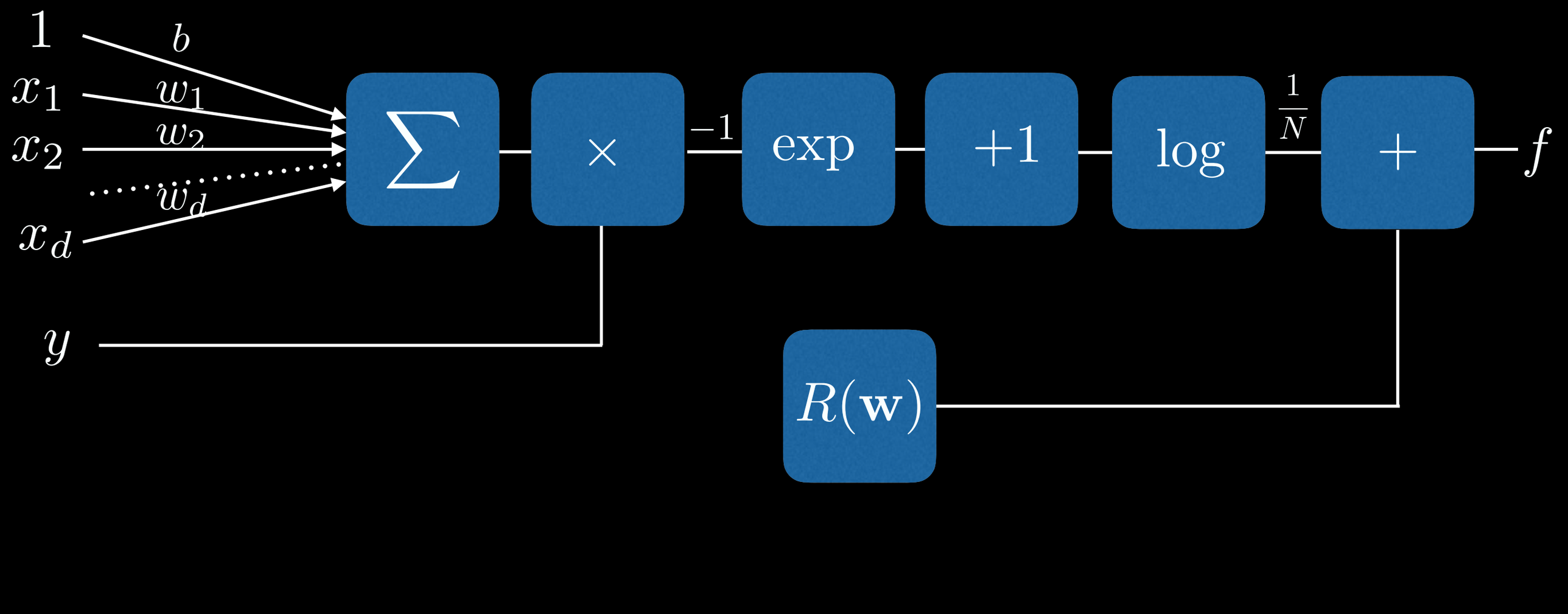
And the LR gradient can be written as:

$$\nabla_{\mathbf{w}} \mathcal{C}(\mathbf{w}) = \lambda \mathbf{w} - \frac{1}{N} \sum_{i}^{N} \frac{y_i \, \mathbf{x_i}}{1 + e^{y_i \mathbf{w}^T \mathbf{x}_i}}$$

An interlude with Jupyter Notebooks

Computational Graph

$$f(\mathbf{x}, \mathbf{w}) = \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{N}\sum_{i}^{N} \log(1 + e^{-y_i \, f(\mathbf{x_i})})$$



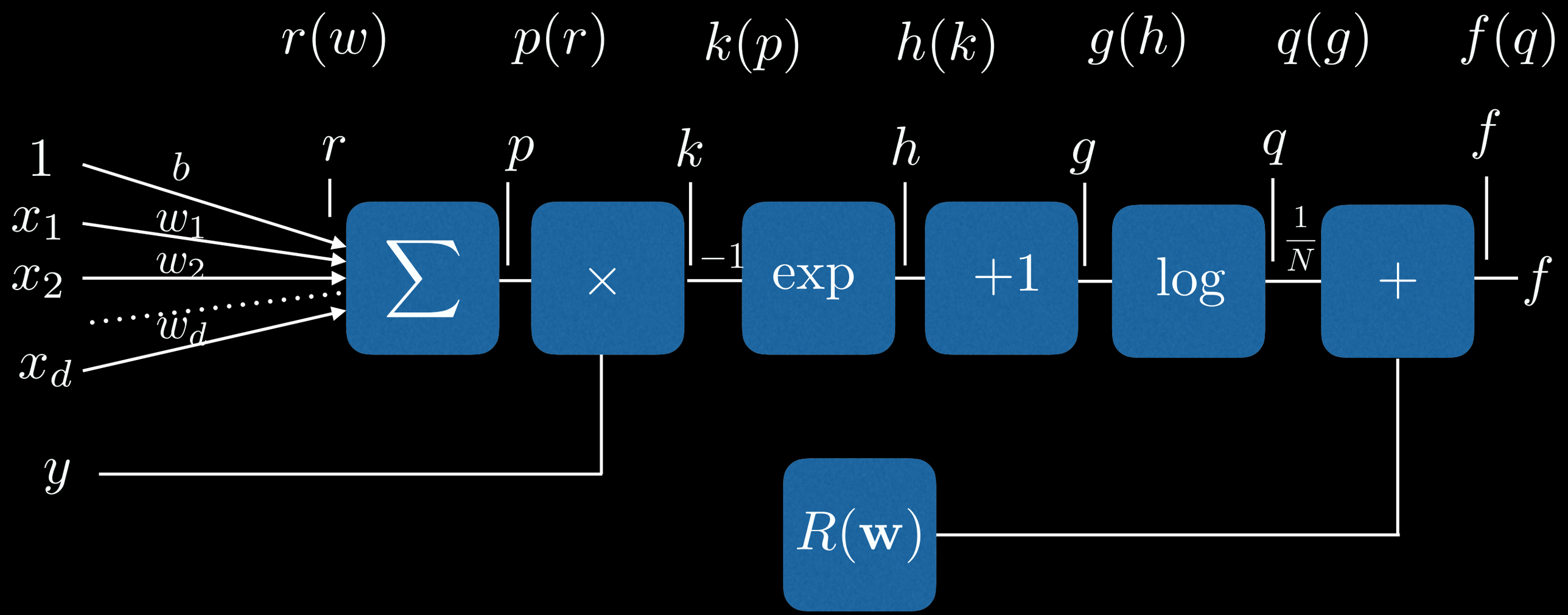
Given a training sample $\mathbf{x}$ let's compute the gradient of f with respect to the weights $\mathbf{w}$.

We have done this before analytically, but let's propagate the gradient back to the weights by using the chain rule!

Why?

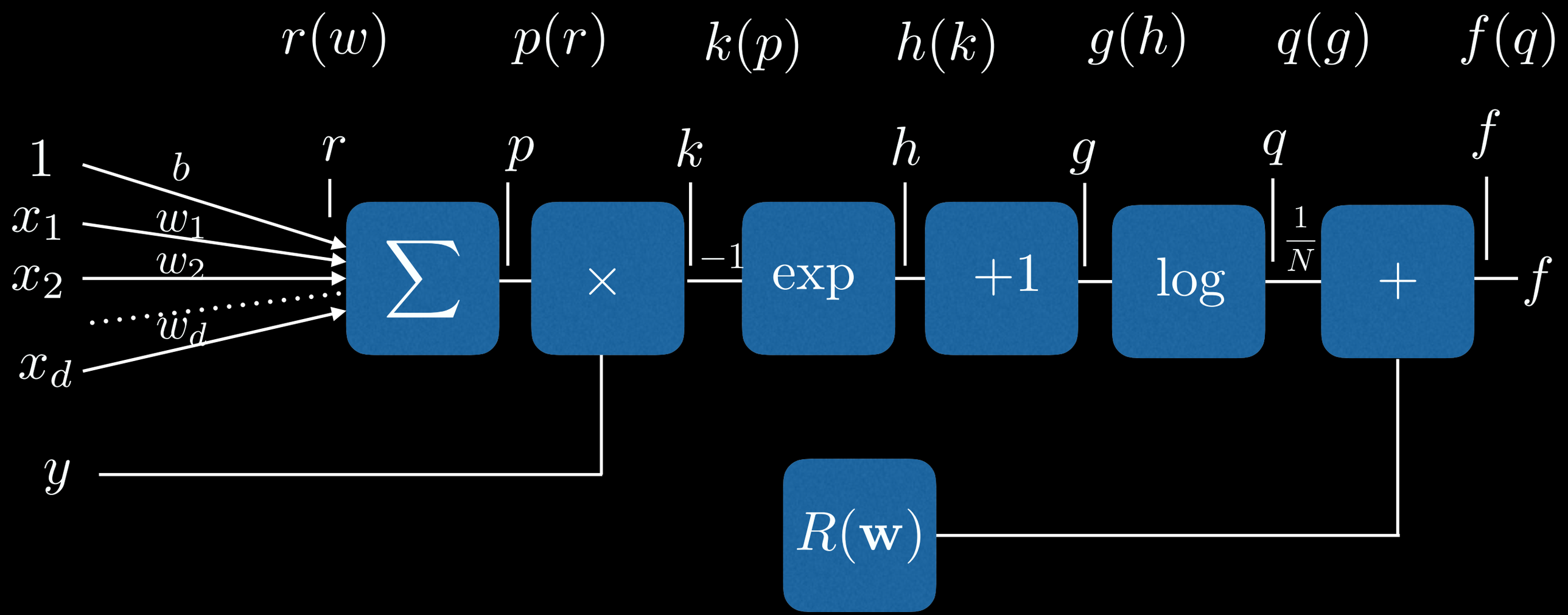
Because we cannot be in the business of deriving all of these gradient by hand as the computational graph gets more and more complicated! This method does not scale.

Chain Rule



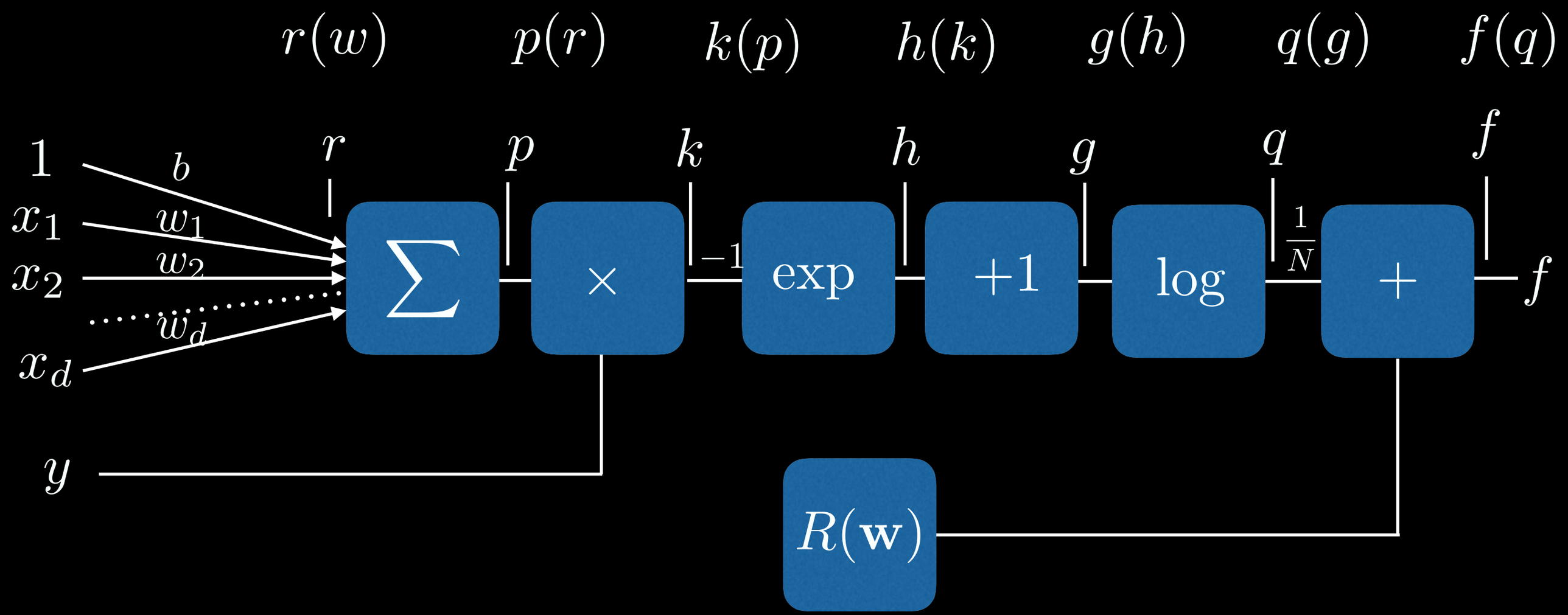
$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial g}\frac{\partial g}{\partial h}\frac{\partial h}{\partial k}\frac{\partial k}{\partial p}\frac{\partial p}{\partial r}\frac{\partial r}{\partial w} + \frac{\partial R}{\partial w}$$

Chain Rule



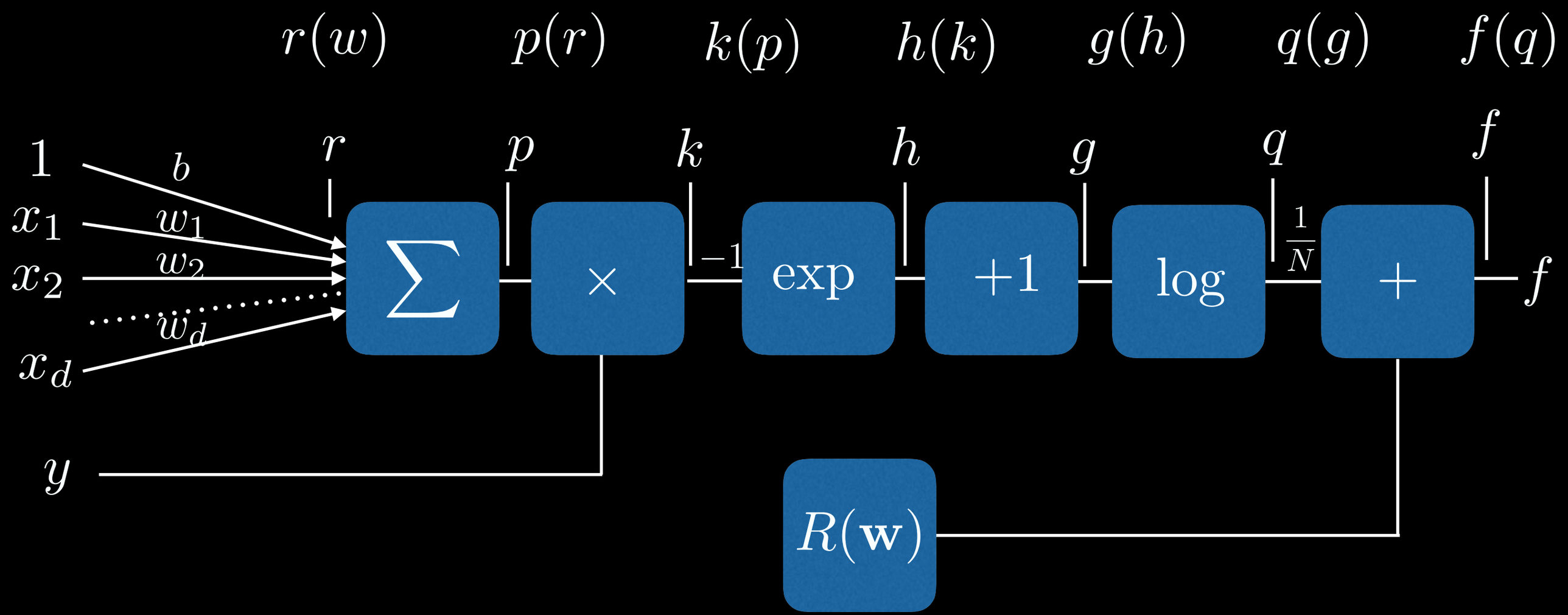
$$\frac{\partial f}{\partial q} = \frac{1}{N}$$

Chain Rule



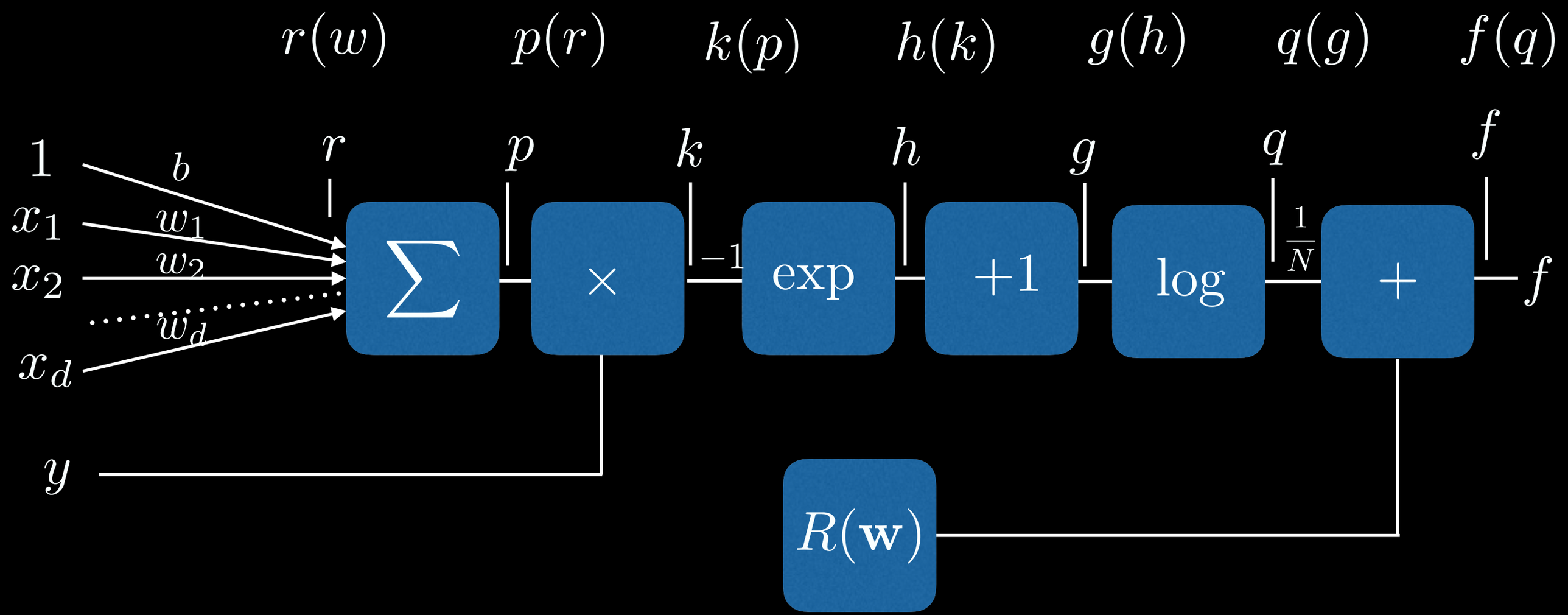
$$\frac{\partial q}{\partial g} = \frac{1}{g}$$

Chain Rule



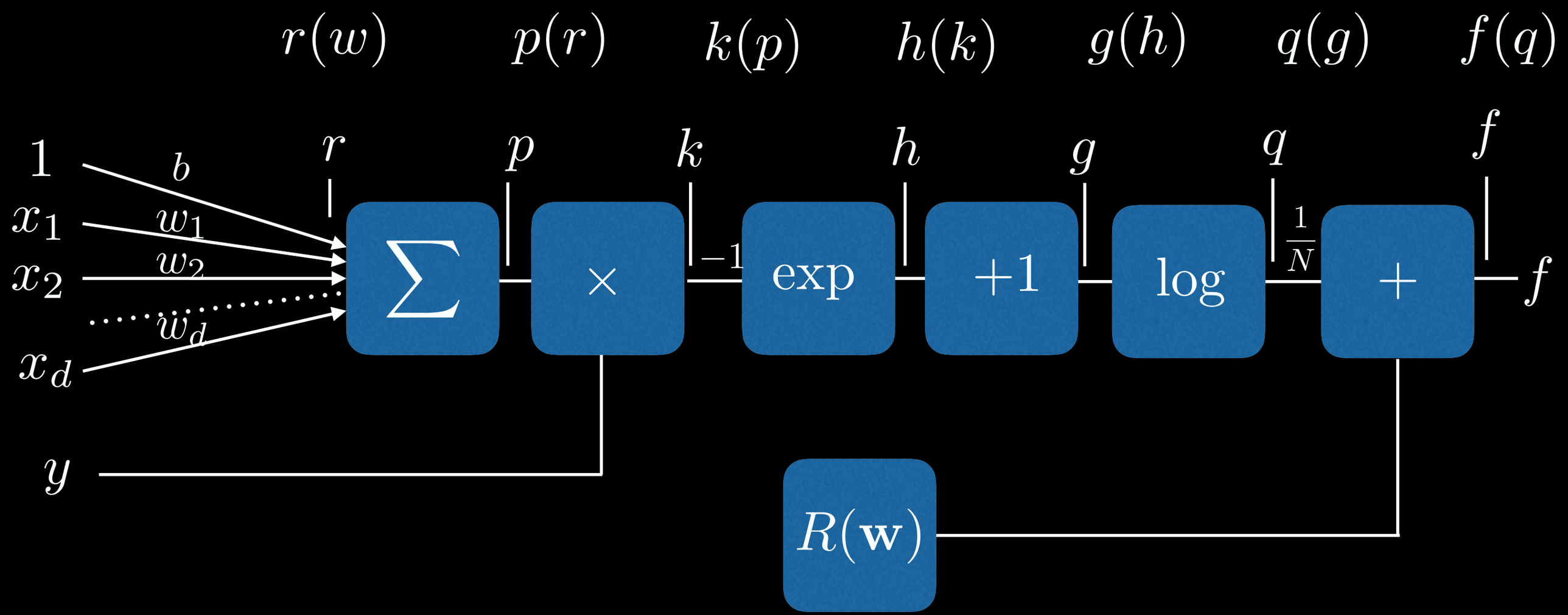
$$\frac{\partial g}{\partial h} = 1$$

Chain Rule



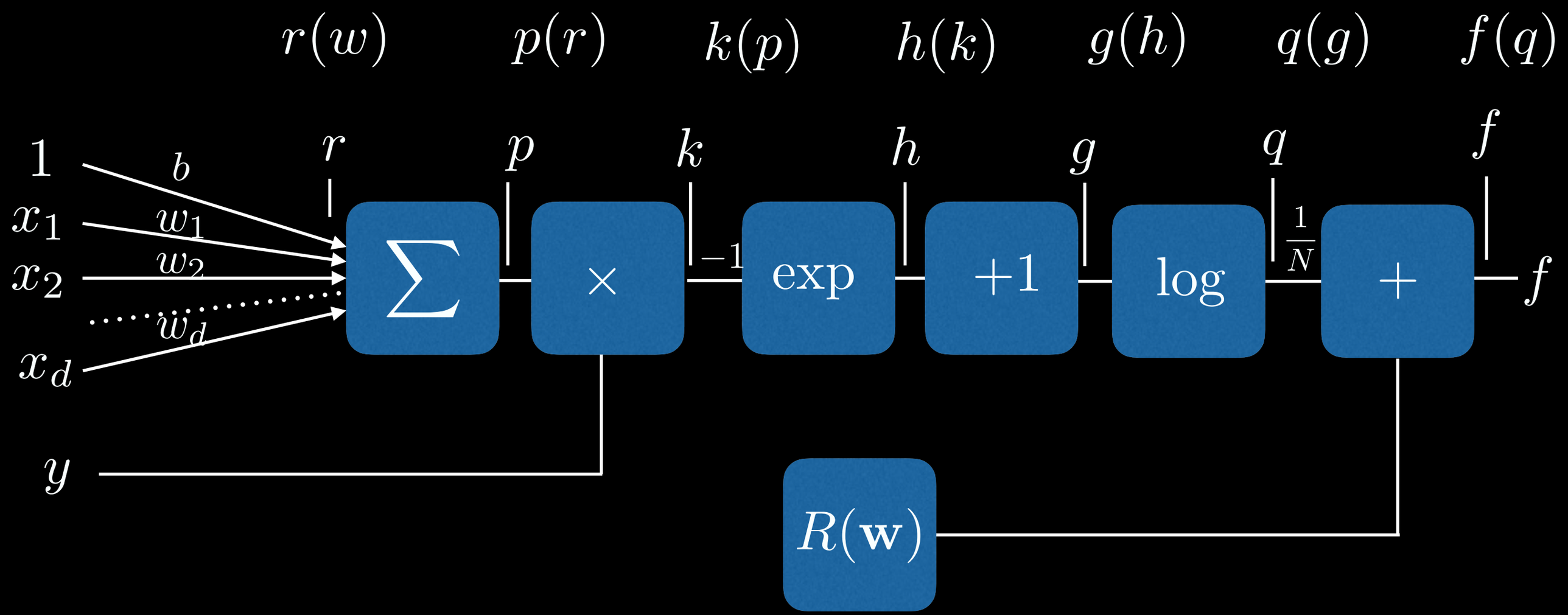
$$\frac{\partial h}{\partial k} = -e^{-k}$$

Chain Rule



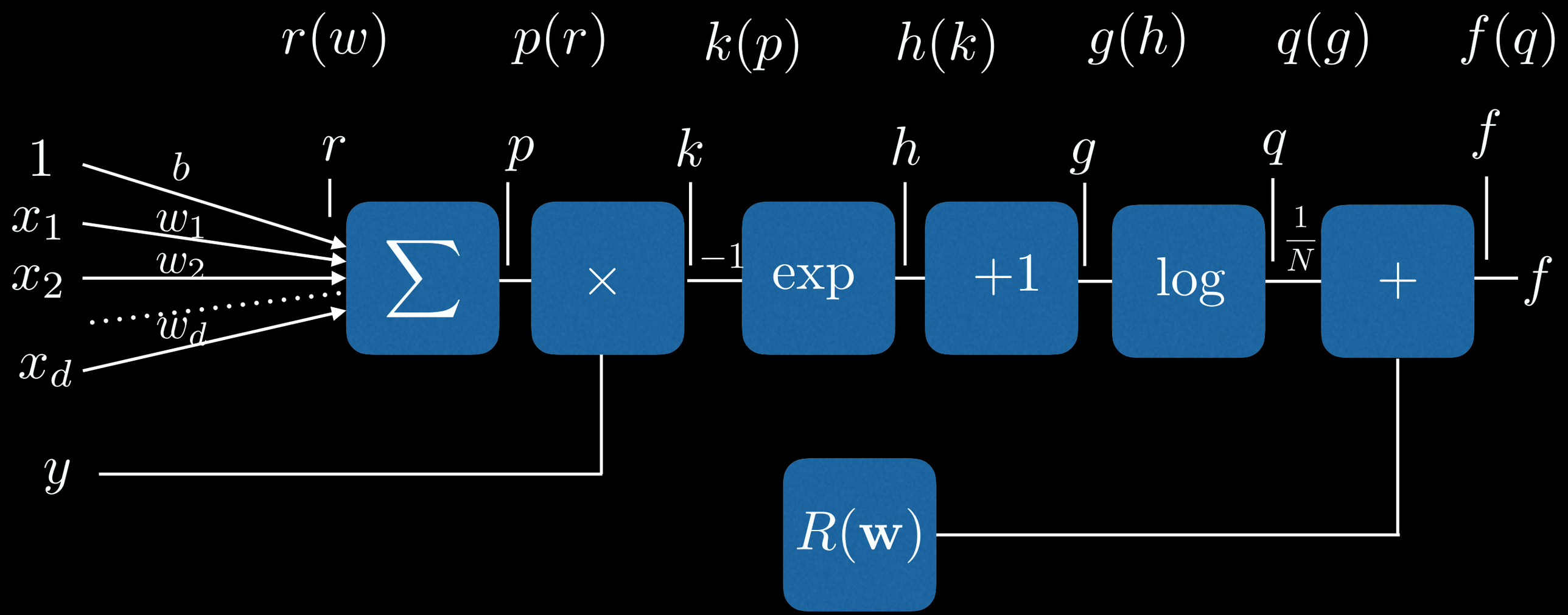
$$\frac{\partial k}{\partial p} = y$$

Chain Rule



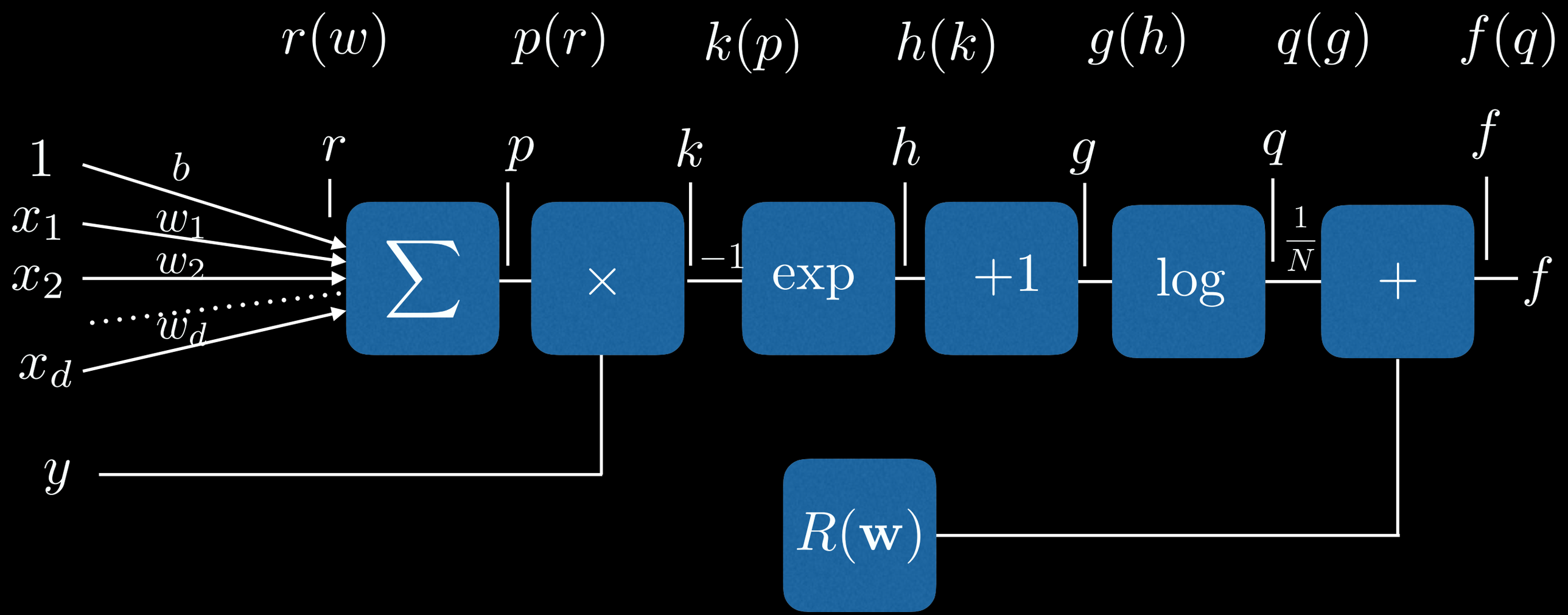
$$\frac{\partial p}{\partial r} = 1$$

Chain Rule



$$\frac{\partial r}{\partial w} = x$$

Chain Rule



$$\frac{\partial f}{\partial w} = \frac{1}{N} \times \frac{1}{g} \times 1 \times -e^{-k} \times y \times 1 \times x + \lambda w$$

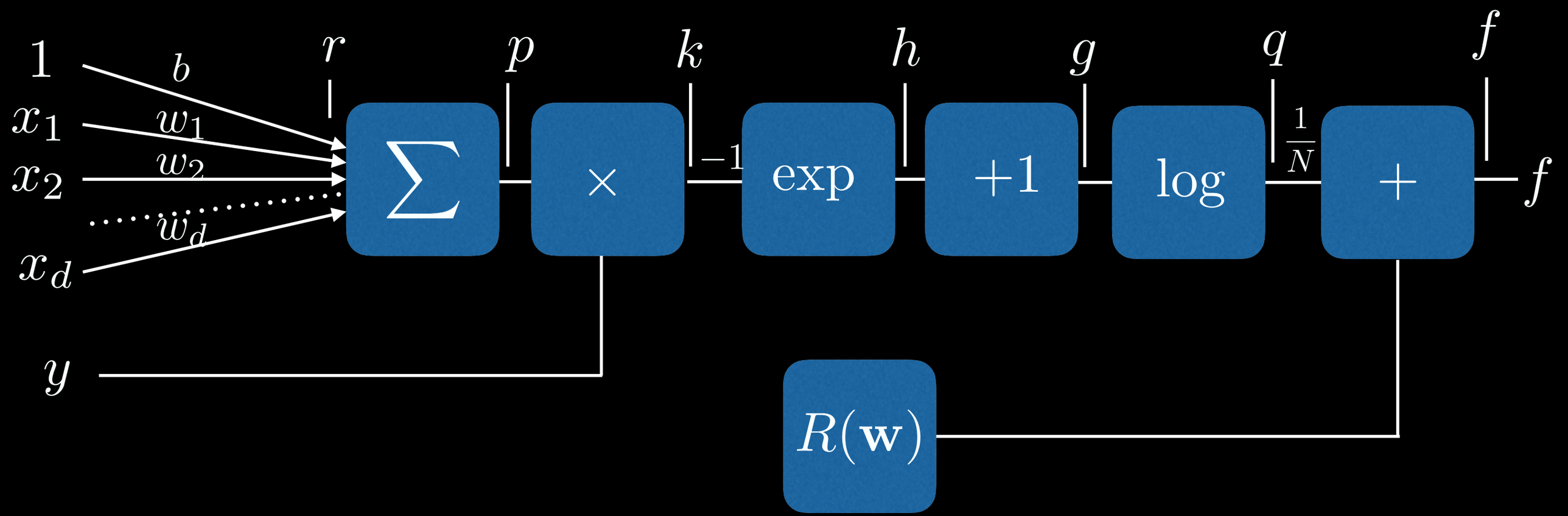
Let’s push a random sample forward through the computational graph and record the intermediate values.

Chain Rule

- Let's say weights $\mathbf{w} = 0$ to start.
- And we have a sample $\mathbf{x} = [1, 2]$ and $y = +1$.
- Let's calculate the gradient wrt the weights **w**.
- And we have N = 1 here.

Chain Rule

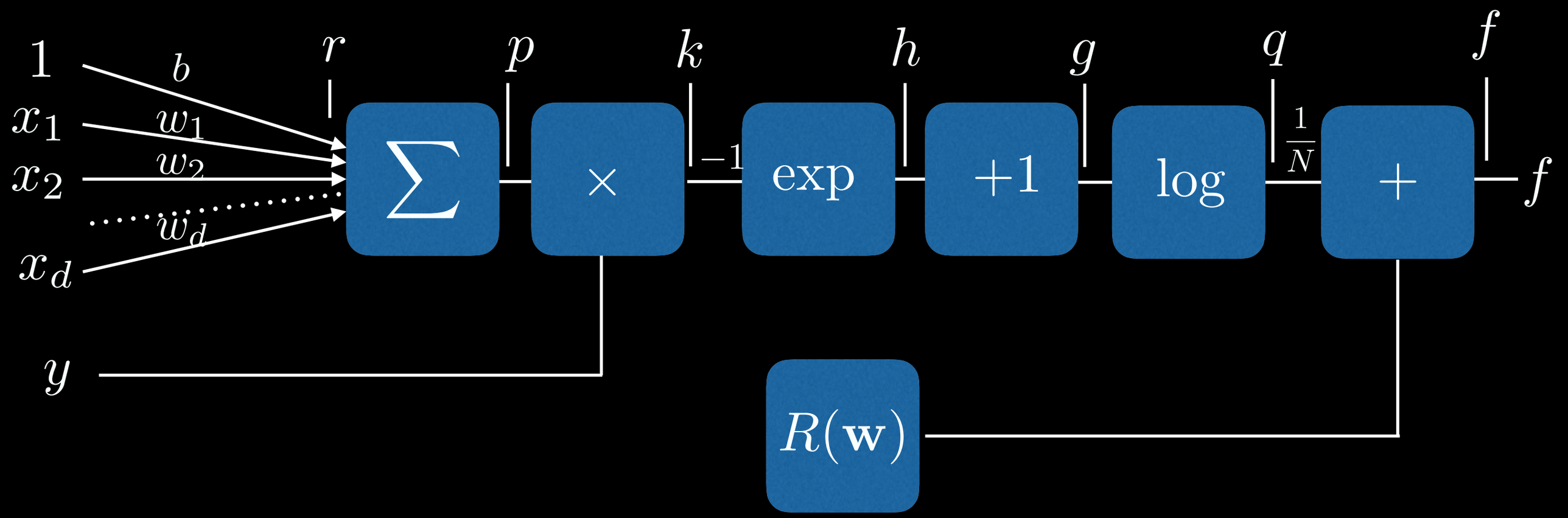
$$r = 0 \quad p = 0 \quad k = 0 \quad h = 1 \quad g = 2 \quad q = 0.69 \quad f = 0.69$$



$$\frac{\partial f}{\partial w} = \frac{1}{N} \times \frac{1}{g} \times 1 \times -e^{-k} \times y \times 1 \times x + \lambda w$$

Chain Rule

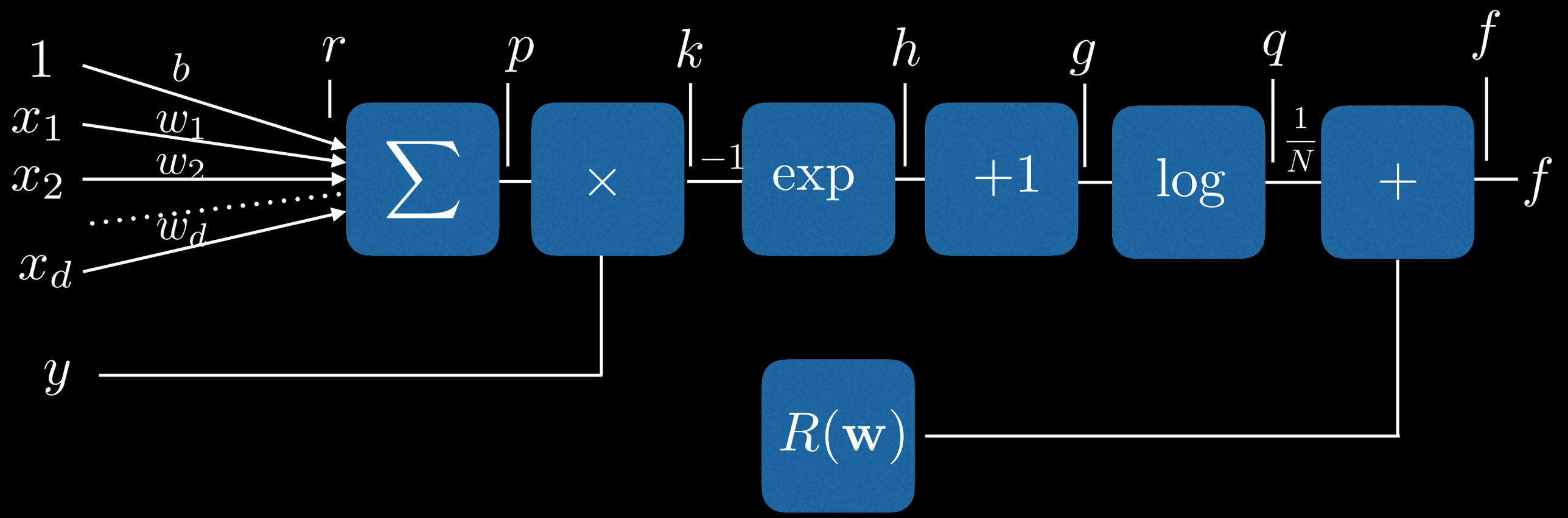
$$r = 0 \quad p = 0 \quad k = 0 \quad h = 1 \quad g = 2 \; q = 0.69 \quad f = 0.69$$



$$\frac{\partial f}{\partial w_1} = \frac{1}{1} \times \frac{1}{2} \times 1 \times -e^{-0} \times 1 \times 1 \times 1 + 0$$

Chain Rule

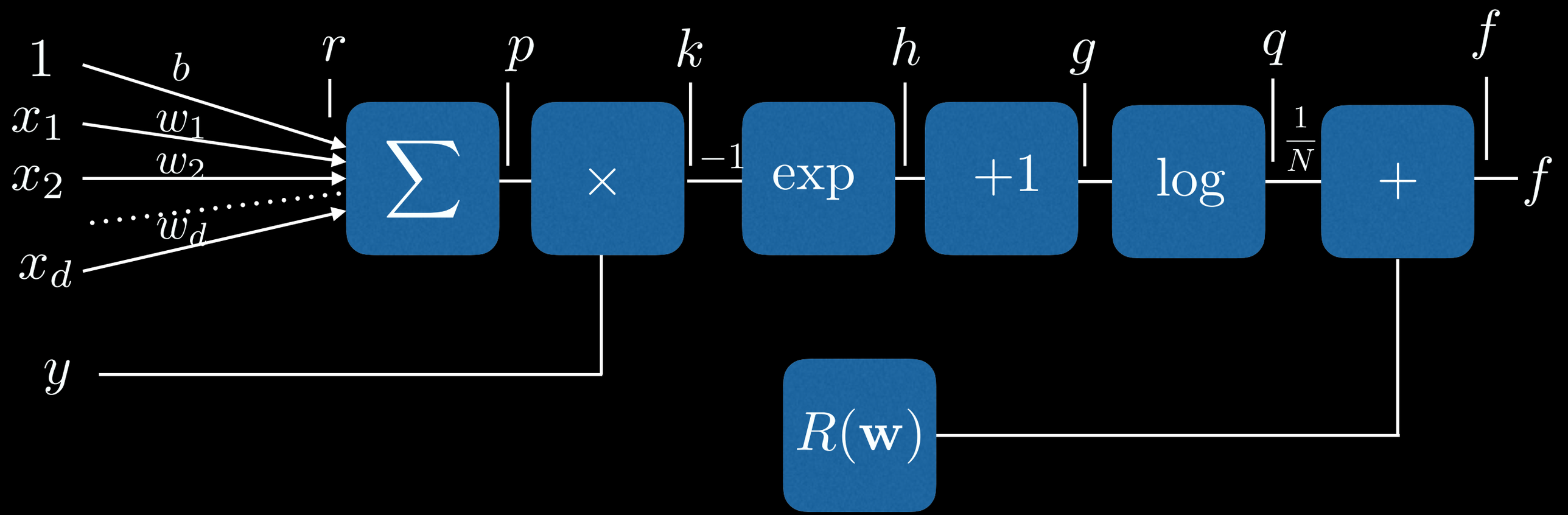
$$r = 0 \quad p = 0 \quad k = 0 \quad h = 1 \quad g = 2 \; q = 0.69 \quad f = 0.69$$



$$\frac{\partial f}{\partial w_1} = -0.5$$

Chain Rule

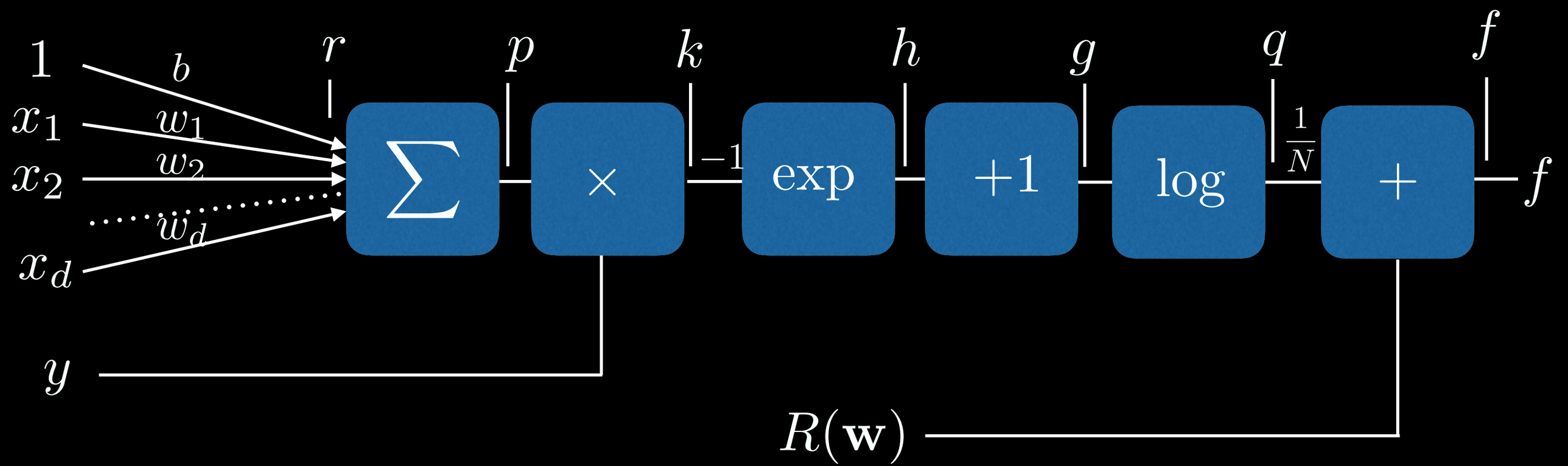
$$r = 0 \quad p = 0 \quad k = 0 \quad h = 1 \quad g = 2 \; q = 0.69 \quad f = 0.69$$



$$\frac{\partial f}{\partial w} = \frac{1}{N} \times \frac{1}{g} \times 1 \times -e^{-k} \times y \times 1 \times x + \lambda w$$

Chain Rule

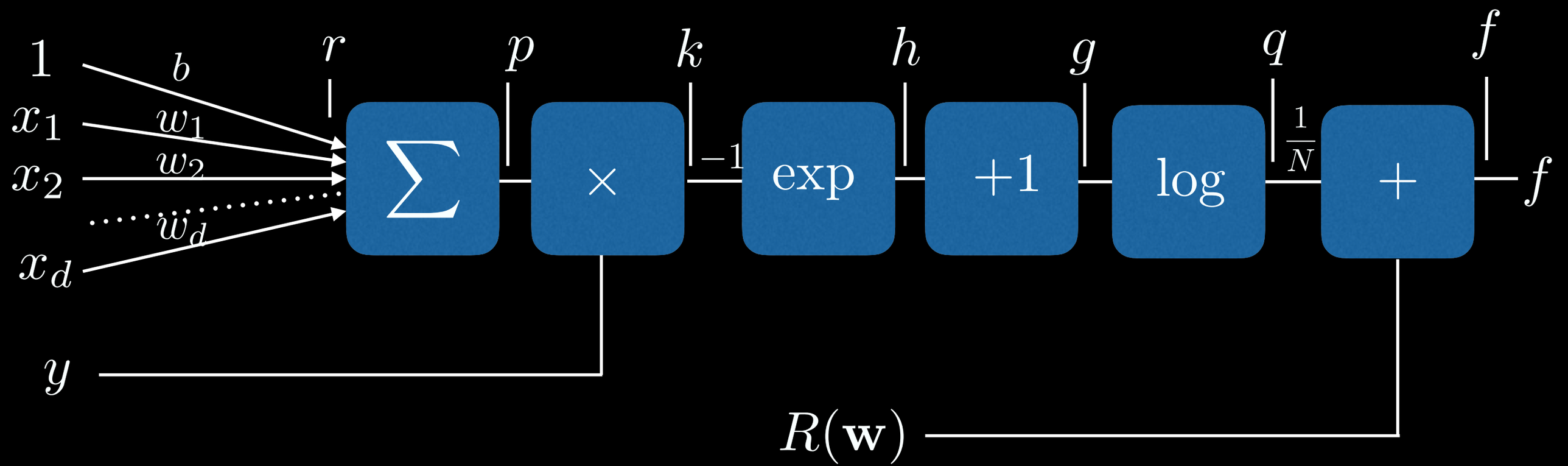
$$r = 0 \quad p = 0 \quad k = 0 \quad h = 1 \quad g = 2 \; q = 0.69 \quad f = 0.69$$



$$\frac{\partial f}{\partial w_2} = \frac{1}{1} \times \frac{1}{2} \times 1 \times -e^{-0} \times 1 \times 1 \times 2 + 0$$

Chain Rule

$$r = 0 \quad p = 0 \quad k = 0 \quad h = 1 \quad g = 2 \; q = 0.69 \quad f = 0.69$$



$$\frac{\partial f}{\partial w_2} = -1.0$$

Wow! That took a while. But you can see how this would generalize.

We have our gradient and now we need to move in the direction opposite the gradient!

Gradient Descent

- Recall $\mathbf{w} = [0, 0]^T$ from before
- Let's choose a learning rate $\eta = 0.1$
- Then we update as $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla_{\mathbf{w}_t} f(\mathbf{w}_t)$

Gradient Descent

- So $\mathbf{w}_2 = \mathbf{w}_1 - 0.1\nabla_{\mathbf{w}_1} f(\mathbf{w})$

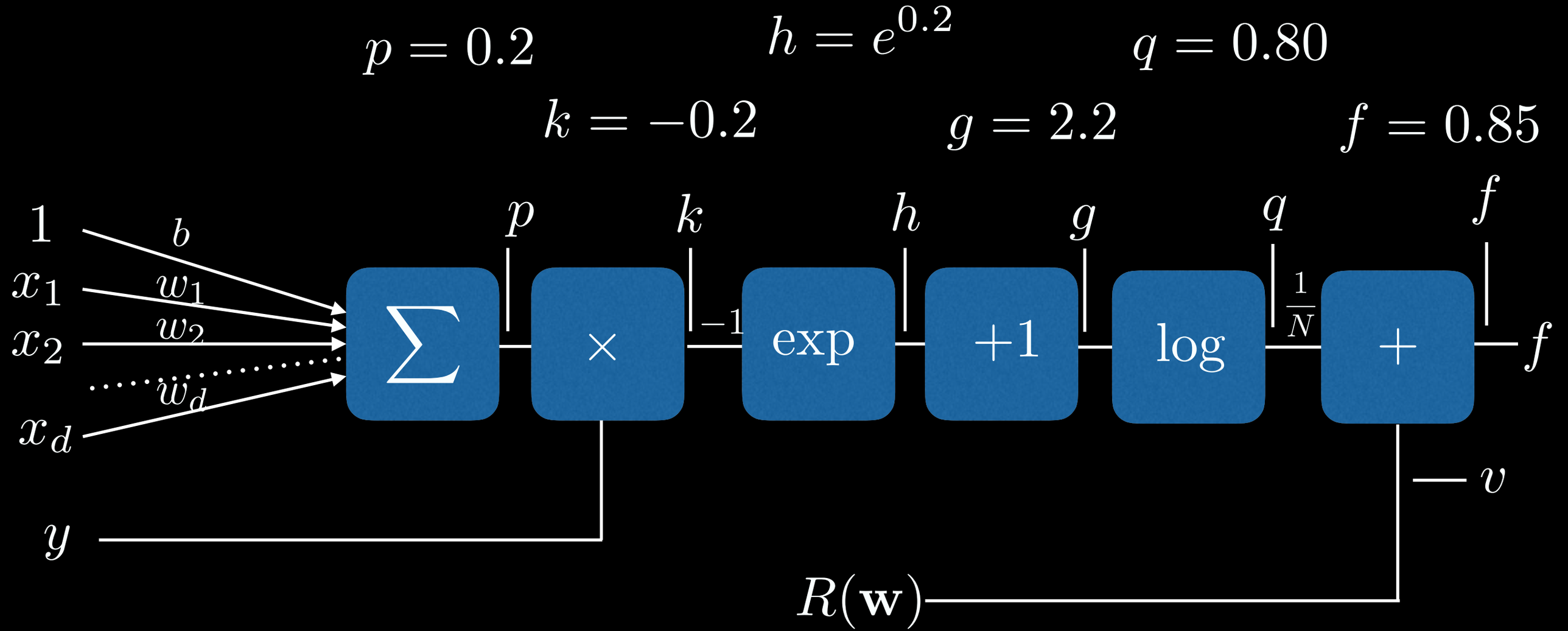
- And we get $\mathbf{w}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - 0.1 \begin{bmatrix} -0.5 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 0.05 \\ 0.1 \end{bmatrix}$

Let’s push another random point through and do it all again.

Chain Rule

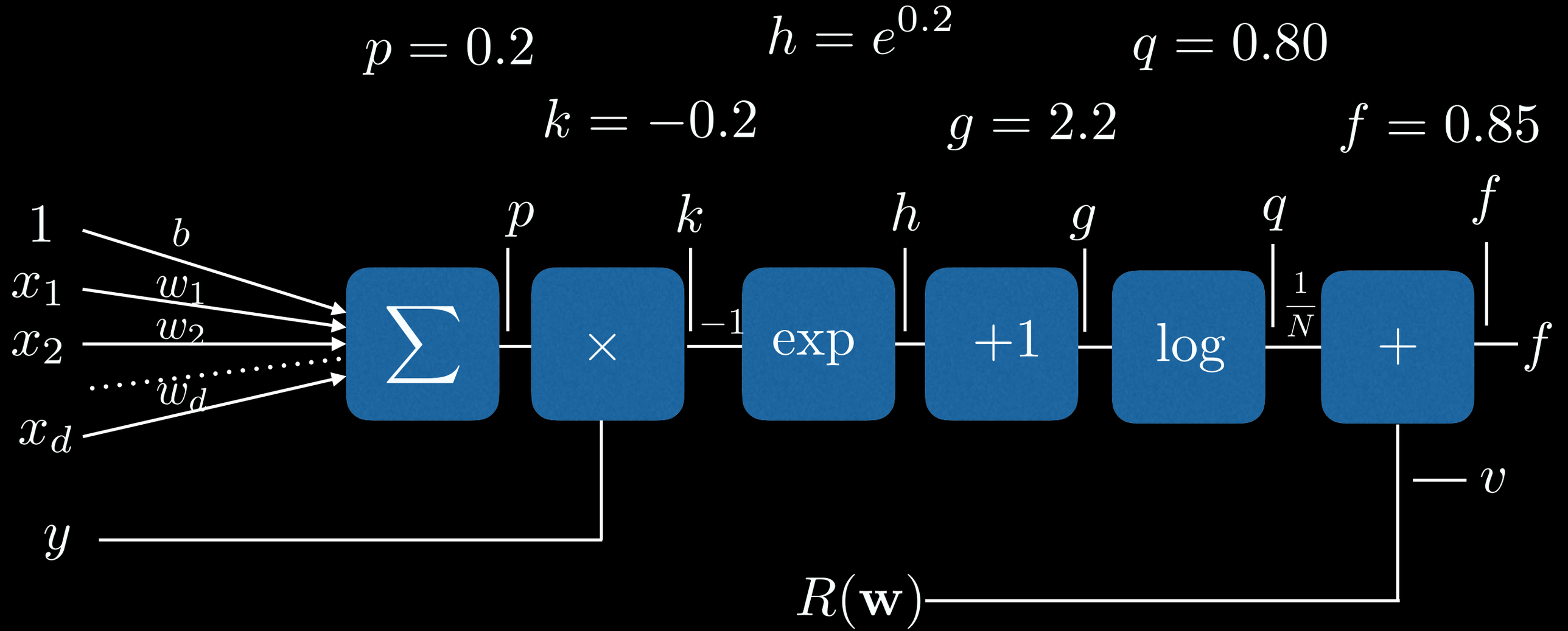
- Now $\mathbf{w} = \begin{bmatrix} 0.05 \\ 0.1 \end{bmatrix}$
- And our new sample $\mathbf{x}$ = [2, 1] and y = -1.
- Let's calculate the gradient wrt the weights **w**.
- Again we have N = 1 here.

Chain Rule



$$\nabla_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{N} \times \frac{1}{g} \times 1 \times -e^{-k} \times y \times 1 \times \mathbf{x} + \lambda \mathbf{w}$$

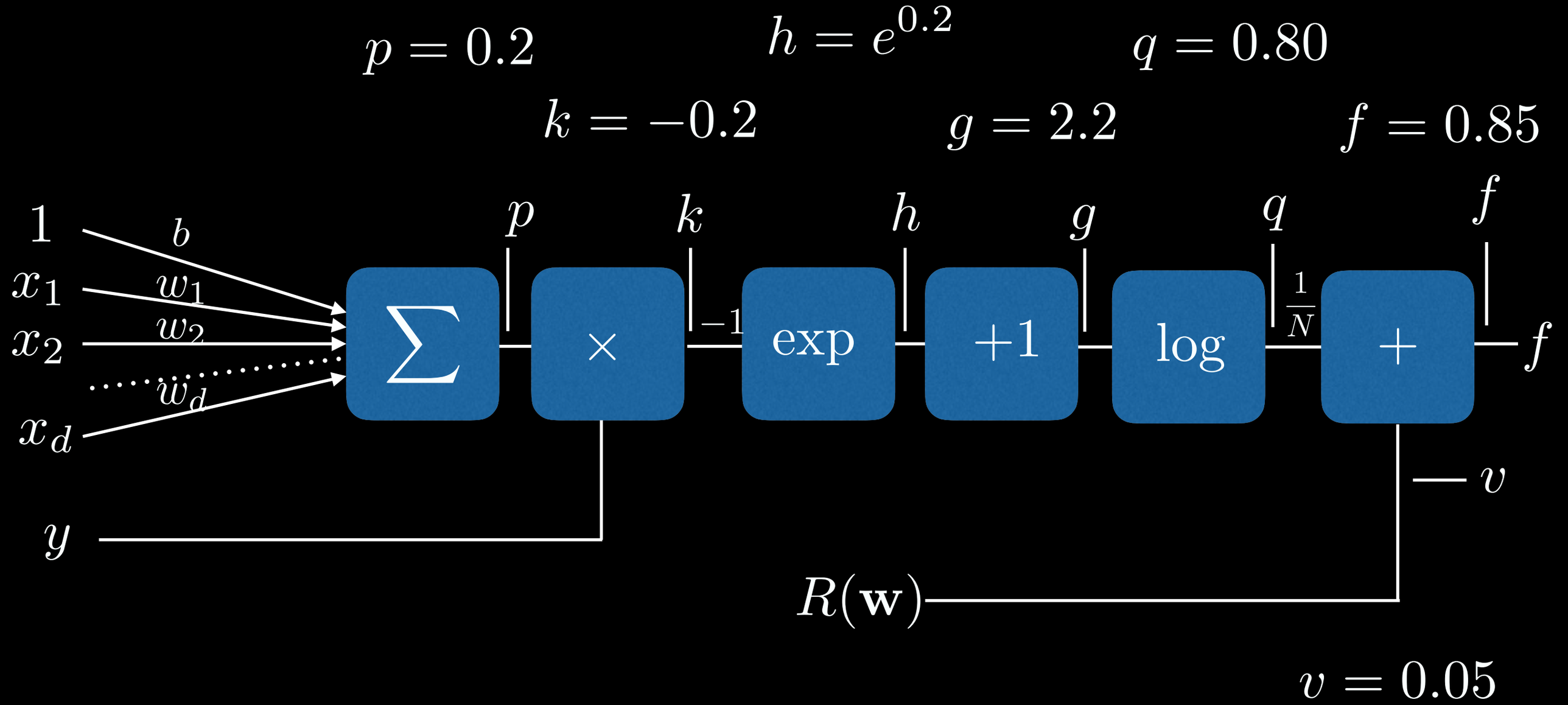
Chain Rule



$$v = 0.05$$

$$\nabla_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{1} \times \frac{1}{2.2} \times 1 \times -e^{0.2} \times -1 \times 1 \times \mathbf{x} + \mathbf{w}$$

Chain Rule

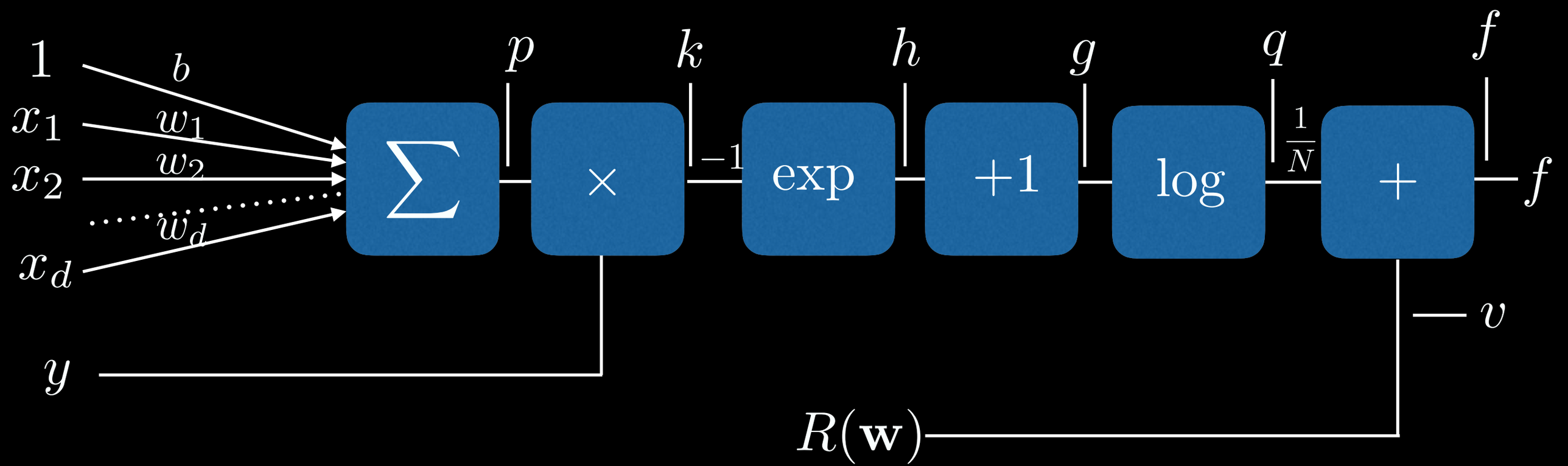


$$\nabla_{\mathbf{w}} f(\mathbf{w}) = 0.56\mathbf{x} + \mathbf{w}$$

Chain Rule

$p = 0.2$ $h = e^{0.2}$ $q = 0.80$

$k = -0.2$ $g = 2.2$ $f = 0.85$



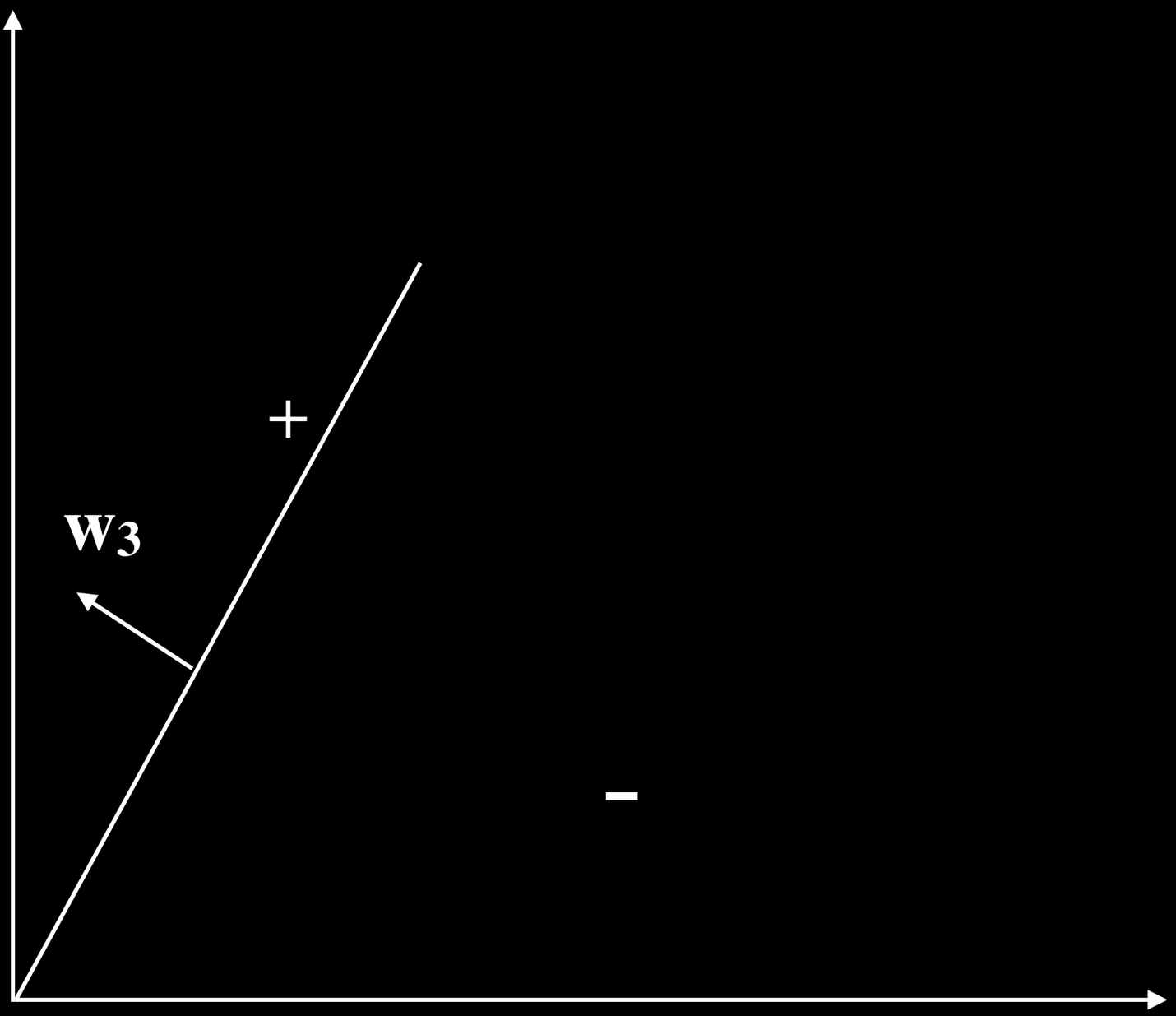
$v = 0.05$

$$\nabla_{\mathbf{w}} f(\mathbf{w}) = 0.56\mathbf{x} + \mathbf{w} = \begin{bmatrix} 1.16 \\ 0.65 \end{bmatrix}$$

Gradient Descent

- So $\mathbf{w}_3 = \mathbf{w}_2 - 0.1 \nabla_{\mathbf{w}_2} f(\mathbf{w_2})$

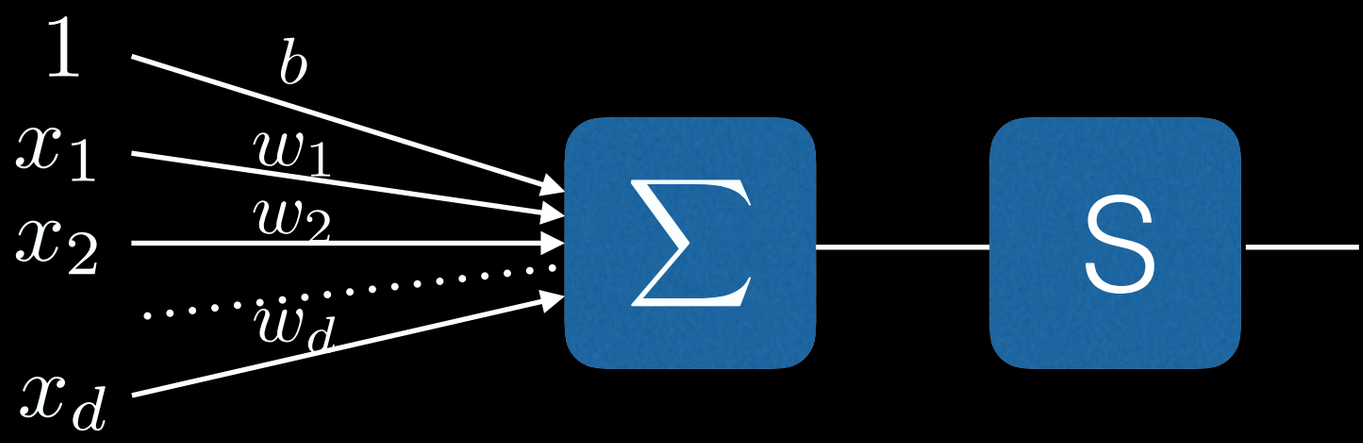
- And we get $\mathbf{w}_3 = \begin{bmatrix} 0.05 \\ 0.1 \end{bmatrix} - 0.1 \begin{bmatrix} 1.16 \\ 0.65 \end{bmatrix} = \begin{bmatrix} -0.066 \\ 0.035 \end{bmatrix}$


+
W3
-

A Simple Neural Network

The Perceptron

[Rosenblatt 57]



$$P(y = 1|\mathbf{x}, \mathbf{w}, b)$$

$$f(\mathbf{x}; \mathbf{w}) = S(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + e^{-w_1x_1 - \ldots - w_dx_d + b}}$$

$$S(z) = \frac{1}{1 + e^{-z}}$$

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

1958 New York Times...