CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Deep Learning Essentials

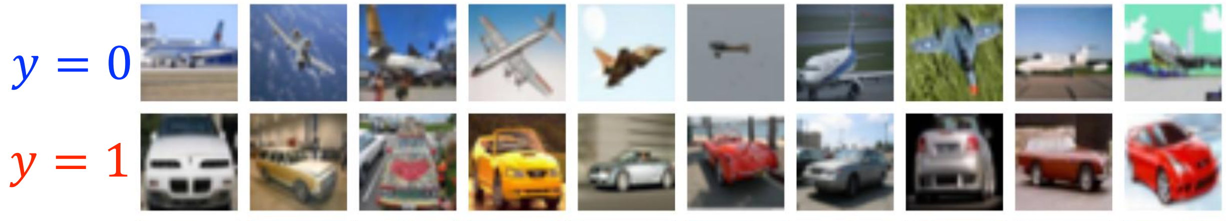
4. Neural Networks

Perceptron, MLP, Backpropagation, Vector-Jacobian product, Autograd

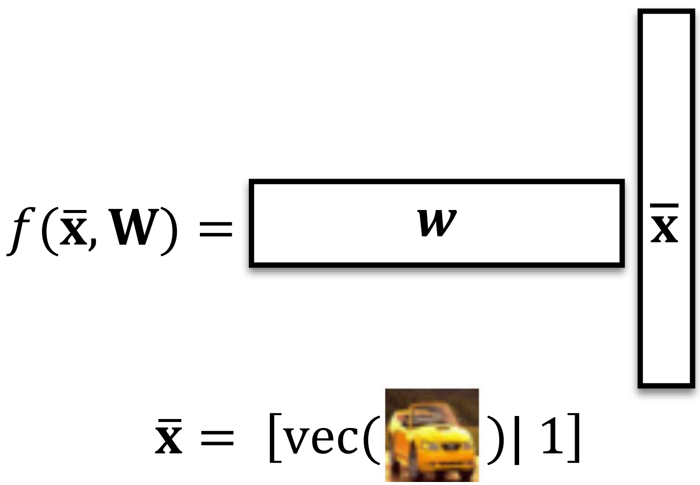
Lukáš Neumann

Adapted from B3B33UROB slides of Karel Zimmerman

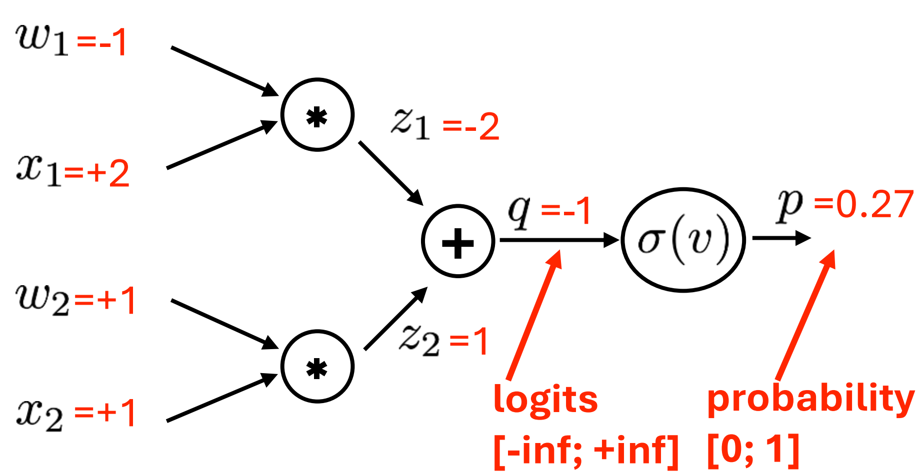
Computational graph



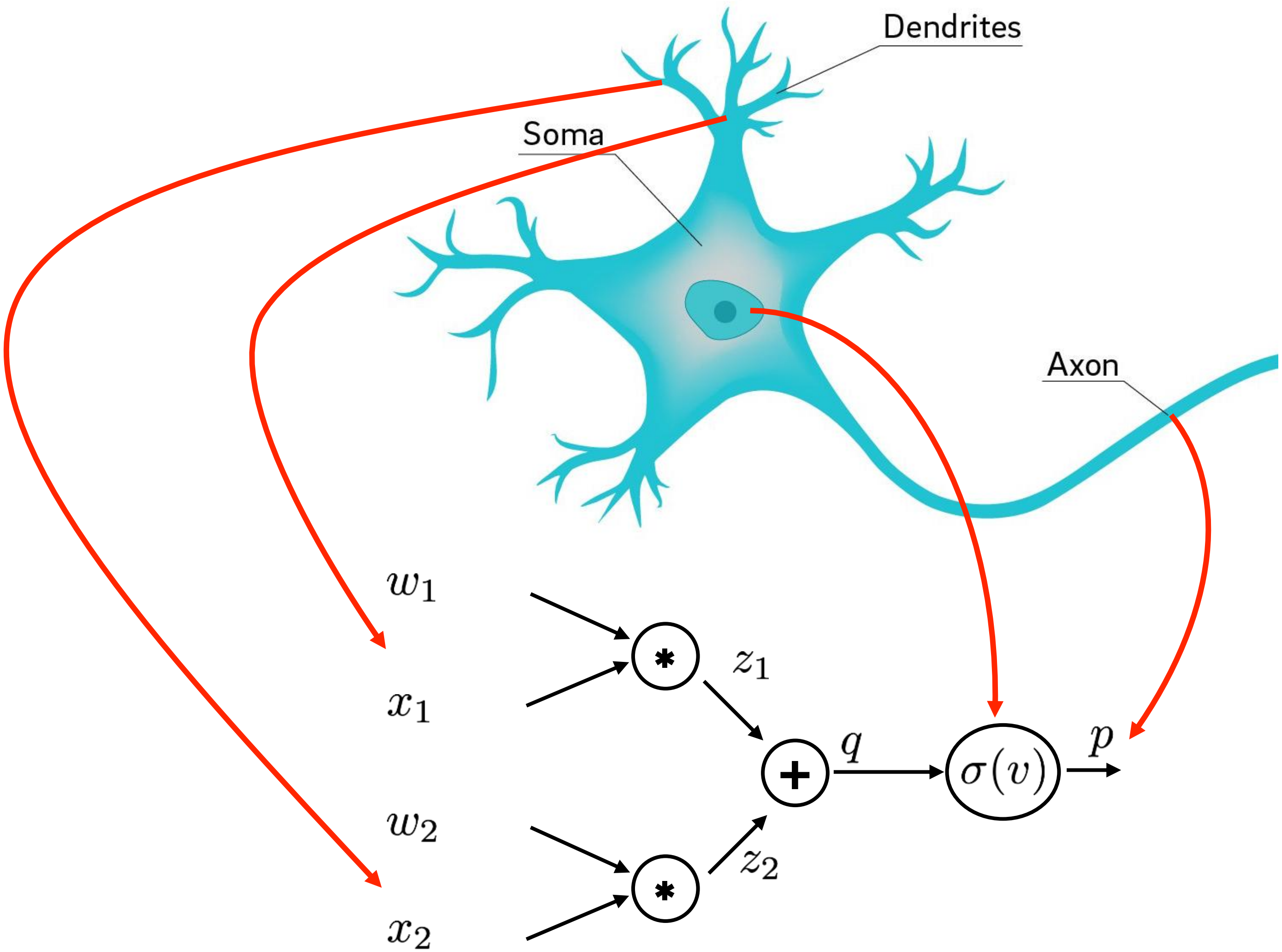
Linear Classifier



Computational graph

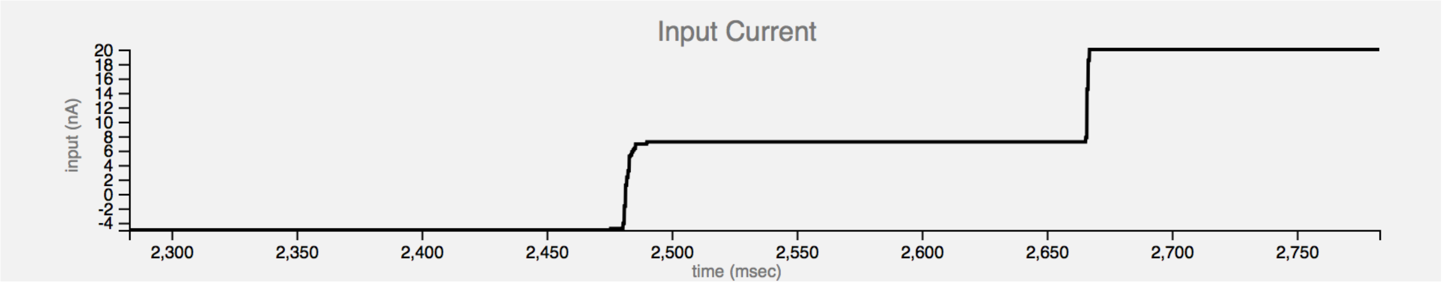


Biological neuron

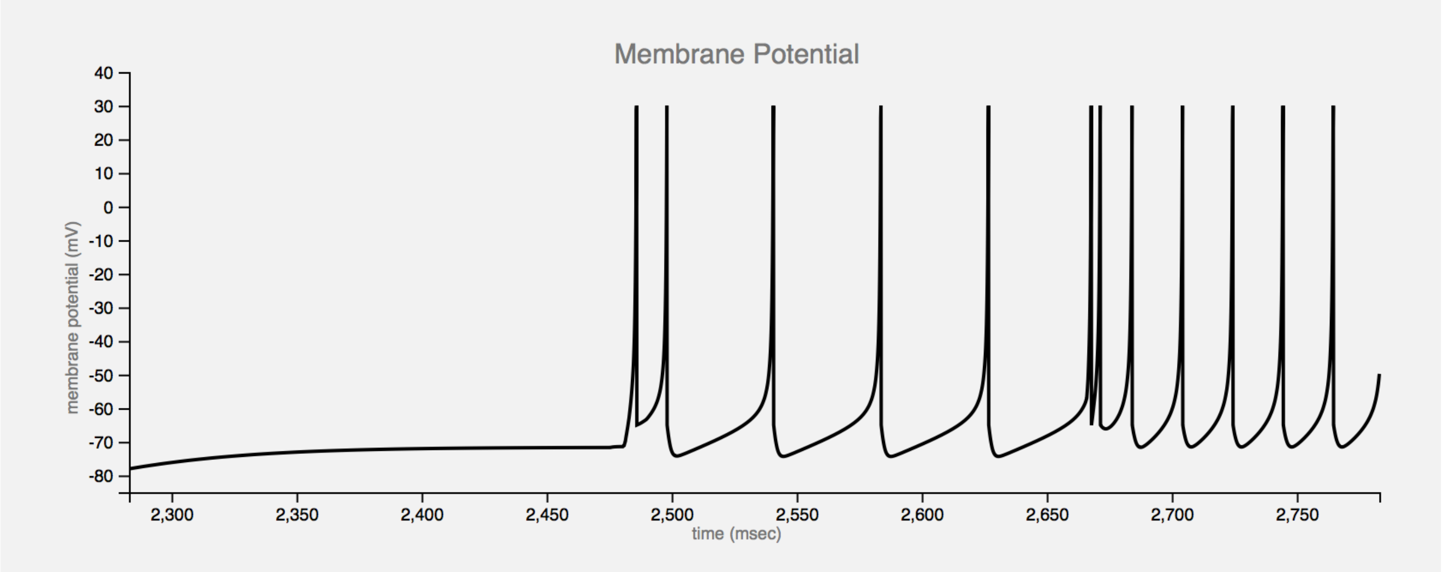


Biological neuron

Input:



Output:

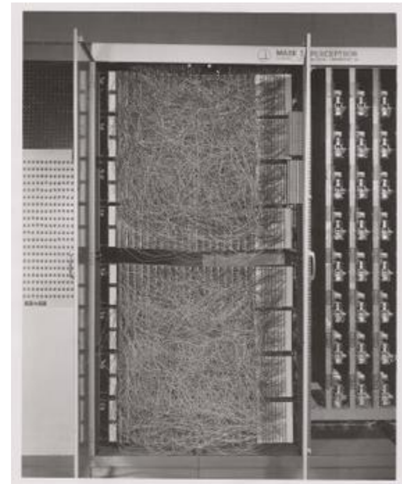


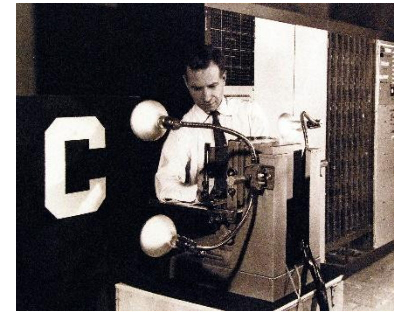
http://jackterwilliger.com/biological-neural-networks-part-i-spiking-neurons/

Perceptron

- **Mark I Perceptron** (1958)
- 20x20 grid of photocells
- 512 “neurons” (A-Units)
- 8 output units
- Weights **w** of A-Units adjustable by potentiometers

$$o = f\left(\sum_{k=1}^{n} i_k \cdot W_k\right)$$





THE NEW YORK TIMES, TUESDAY, JULY 8, 1958.

L 23

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.

Books of The Times

By CHARLES POORE

"IF this were an entirely accurate account of my life in Cork," the author of "Mrs. O' "* tells us, "I should probably be writing it behind bars. So I should say that it is impressionistically true when not always factually so."

Fair enough. However, when you have finished her entertaining book, you may want to go back to that preface and wonder whether the bit about *behind bars* is a pun or an Irish bull.

Why? Because she ran a pub in Cork. The idea of doing so came to her in London one afternoon when she found herself rather rich and completely free. "My decree absolute came through on the same day as my Great Aunt's legacy—not a fortune, but such a sum as I had never dreamed of owning or saving." The fact that she happened to choose for refreshment a place called Mooney's, in London, gave the notion a proper touch of predestination.

Once in Ireland she made forays around the country. It did not take her very long to find the pub she wanted in Cork and buy it from a maiden lady who did not appreciate its seedy elegance. What names she signed to the deed we do not know, although this book is copyrighted by C. M. Forde. As author of it she calls herself, with royal simplicity, Claude, just Claude.

Named by Irish Friends

It was her Irish friends and customers who gave her the name of Mrs. O'. A reference to herself, near the end of the book, as one who holds in reserve "the resignation to the inevitable that lingers in the blood of those born in fatalistic East," marks the

Claude, author of "Mrs. O'."

witz with any attention, you would think of the long arid stretches with no hostelries."

Early on, as proprietor of her own pub, she had learned to tap a mighty keg of high stout, after just one lesson from a friendly rival called Foxy:

"Wrapping the barrel end of the tap in three thicknesses of newspaper as I had seen Foxy do, I placed the tap against the bung, raised the mallet, and thinking briefly that I should probably be the first foreigner ever to be killed by Guinness, I hit the tap two fairly light, quick blows."

Mallet's Force Augmented

It worked fine. The third whack was delivered at full strength. The tap went into place, the newspaper sealed the crack

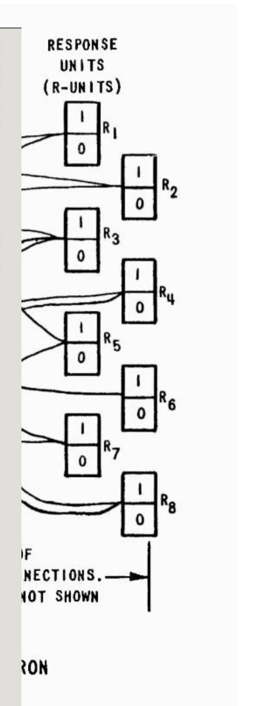
Books—Authors

A month before Joyce Cary's death last year the complete manuscript of his new book "Art and Reality" arrived at Harper's, his publisher. It is based in part on lectures prepared for delivery at Clare College, Cambridge. Mr. Cary described the book as "an attempt to examine the relation of the artist with the world as it seems to him and to see what he does with it." He examines the nature of intuition and truth in art, the difficulties of translating intuition into production and the character of originality in art. The book will be published Aug. 6.

Lucius Cornelius Sulla (138-78 B. C.), Roman general and politician, established a dictatorship in 82 B. C. through which he eliminated members of the opposing party. To later generations his name became a symbol of cold, calculating cruelty. He is the subject of a historical novel, "The Sword of Pleasure," by Peter Green, which World will issue Aug. 27. The story will be the first by the young English historical novelist to be published in this country.

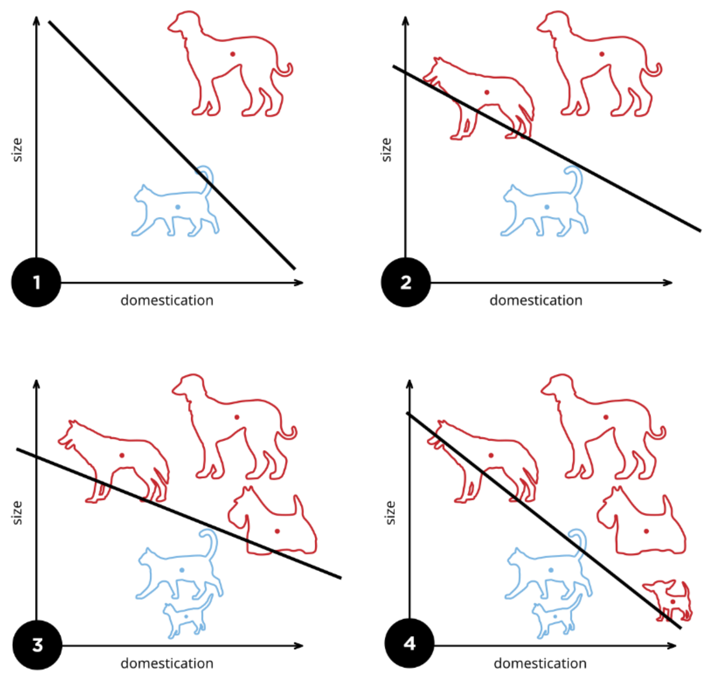
A young French writer will be introduced to the American reading public in September when Dutton issues "Jacob," by Jean Cabriès. Translated from the French by Gerard Hopkins, the novel is based on the story as told in Genesis and covers the period from Jacob's flight from Isaac's house to his reconciliation with his brother Esau. Described as a spiritual novel, it tells of a man's education.

"The Patchwork Hero," a novel by Michael Noonan, is planned for November publication by John Day. It is the story of a year in the life of a motherless boy. The youngster lives in a respectable village with his roistering tugboat-captain father and his father's friends, who share their festive and unpredictable hospitality.



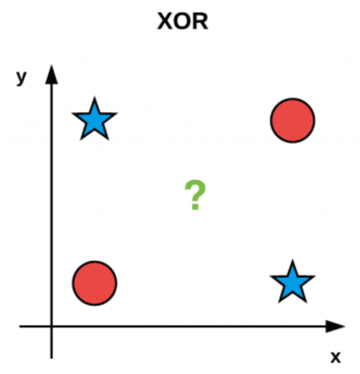
Perceptron

- The perceptron is learnt using the Perceptron algorithm
  - In each iteration, find a sample which is misclassified
  - If none exists, we are done
  - If there is a misclassified sample, update decision boundary so that the sample is now classified correctly



Perceptron

- Perceptron was very popular, until it was shown that it simply cannot work for certain type of problems such as the XOR problem [Minsky & Papert 1969]



- Adding another layer of neurons solves this, but the Perceptron learning algorithm no longer works → no one knew how to train Perceptrons with more than one layer

Backpropagation

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of the input units. If the input units are directly connected to the output units it is relatively easy to find learning rules that iteratively adjust the relative strengths of the connections so as to progressively reduce the difference between the actual and desired output vectors[2]. Learning becomes more interesting but more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input–output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning pro
networks which have a layer of input un
number of intermediate layers; and a la
the top. Connections within a layer or
layers are forbidden, but connections c
layers. An input vector is presented to t
the states of the input units. Then the stat
layer are determined by applying equatio
connections coming from lower layers. A
have their states set in parallel, but diffe
states set sequentially, starting at the
upwards until the states of the output un

The total input, x_j, to unit j is a linear fu
y_i, of the units that are connected to j ar
on these connections

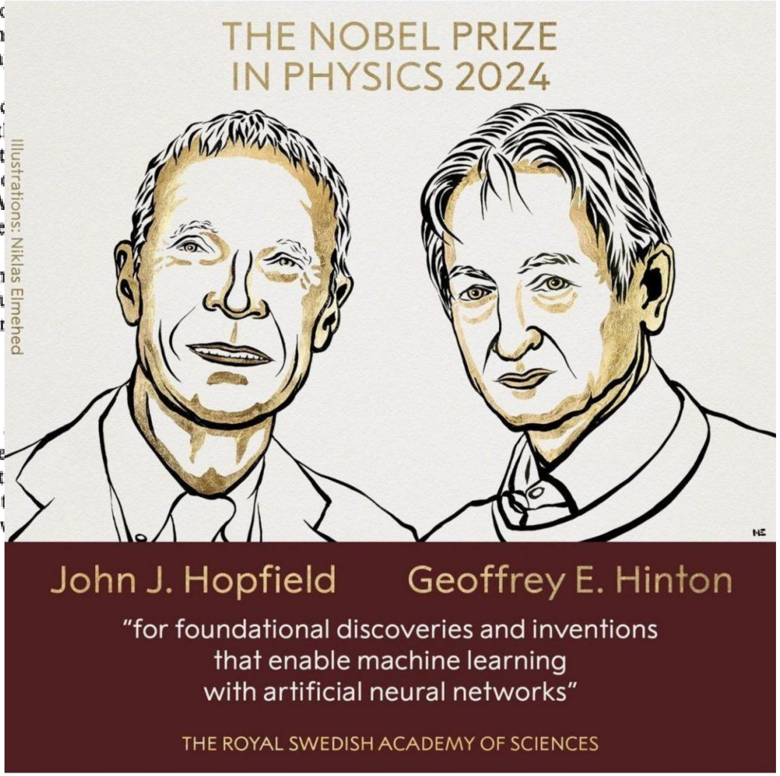
$$x_j = \sum_i y_i w_{ji}$$

Units can be given biases by introducing
unit which always has a value of 1. The
input is called the bias and is equivalent
opposite sign. It can be treated just like t

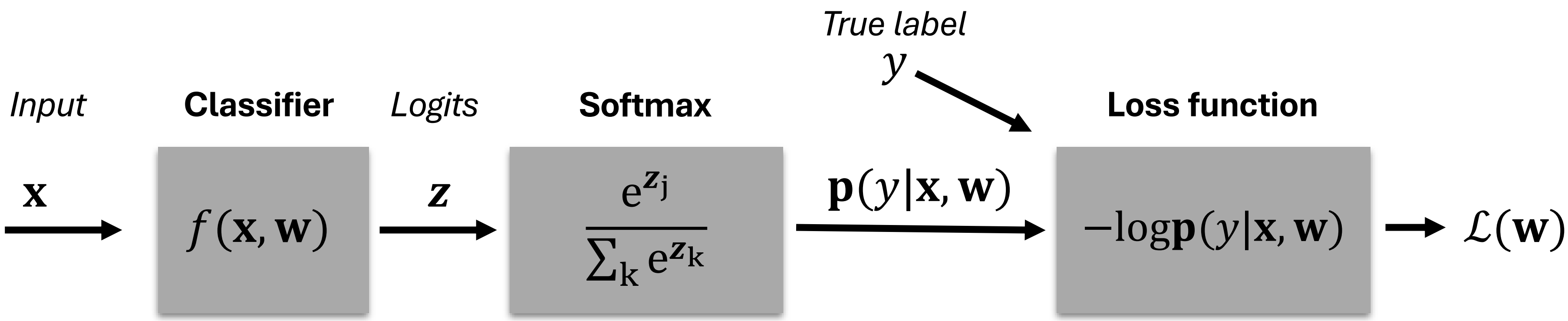
A unit has a real-valued output, y_j, w
function of its total input

$$y_j = \frac{1}{1+e^{-x_j}}$$

† To whom correspondence should be addressed.



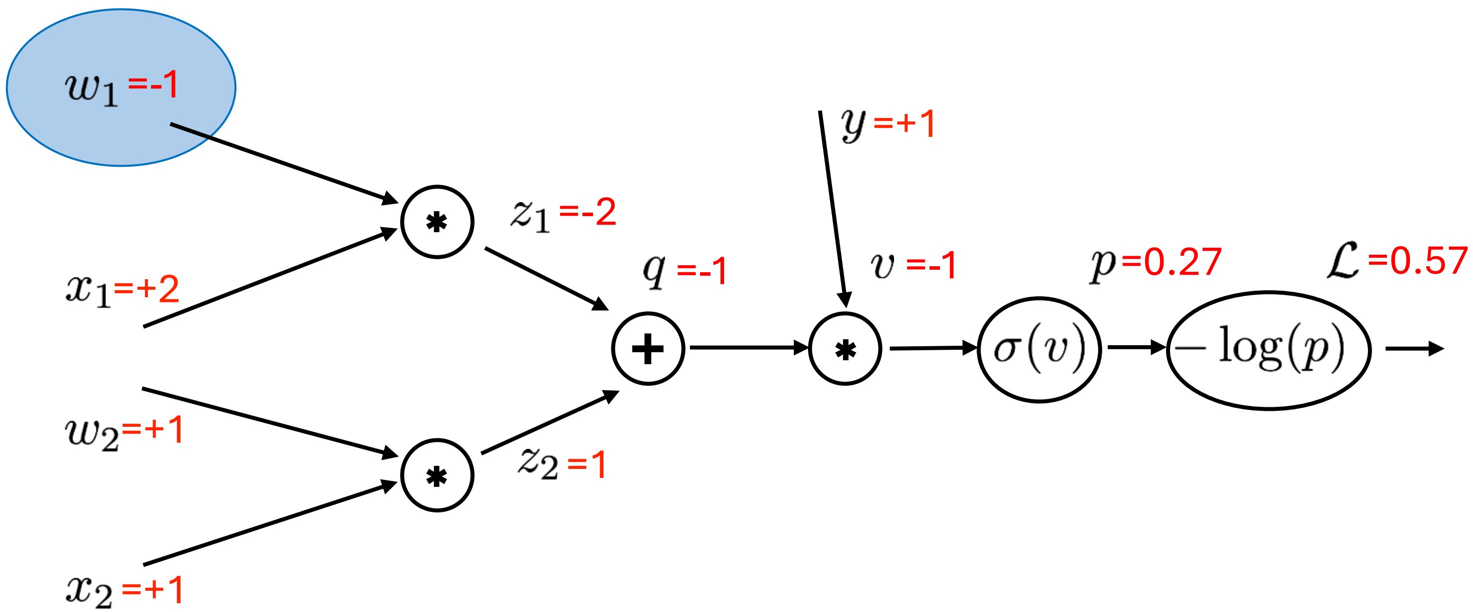
Training pipeline (Lecture 02)



1. Calculate loss for each sample $\mathbf{x}$ and then sum (average) over a training set batch
2. Calculate weight gradient with respect to the loss $\frac{\partial \mathcal{L}}{\partial \mathrm{w}}$
3. Update model weights $\mathbf{w}^{t+1} := \mathbf{w}^{t} - \lambda \frac{\partial \mathcal{L}}{\partial \mathrm{w}}$

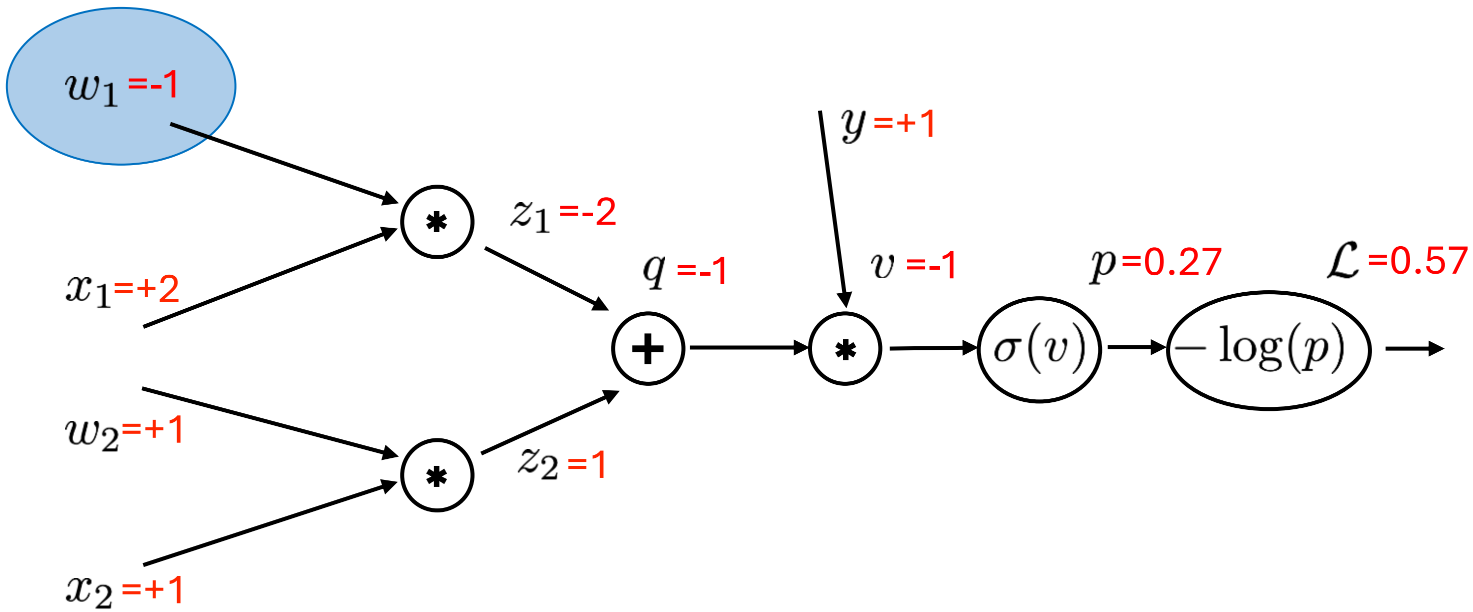
```
for _, (inputs, labels) in enumerate(training_data):
     optimizer.zero_grad()                    # Zero gradients
     logits = model(inputs)                   # Make predictions

     loss = loss_fn(logits, labels)           # 1. Compute the loss
     loss.backward()                          # 2. Calculate gradients
     optimizer.step()                         # 3. Update weights
```

Backpropagation



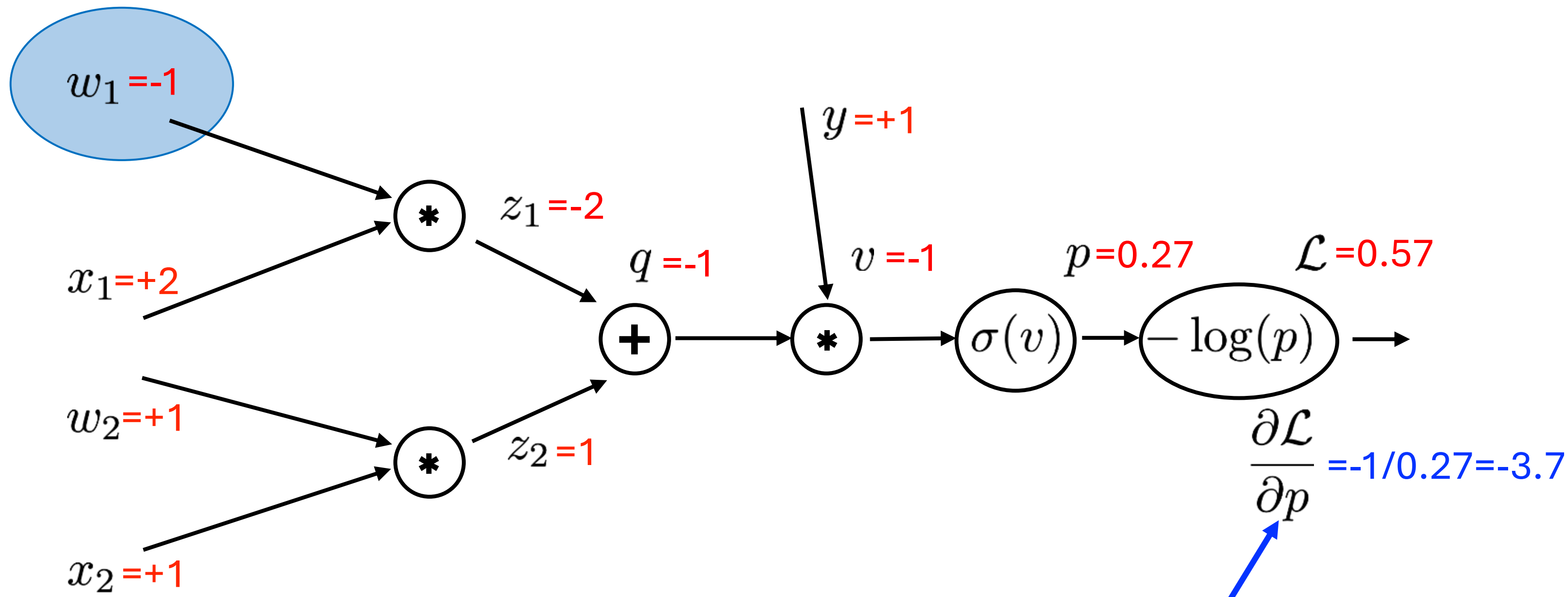
$$\frac{\partial \mathcal{L}}{\partial w_1} = ?$$

Backpropagation



$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p} \frac{\partial p}{\partial v} \frac{\partial v}{\partial q} \frac{\partial q}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

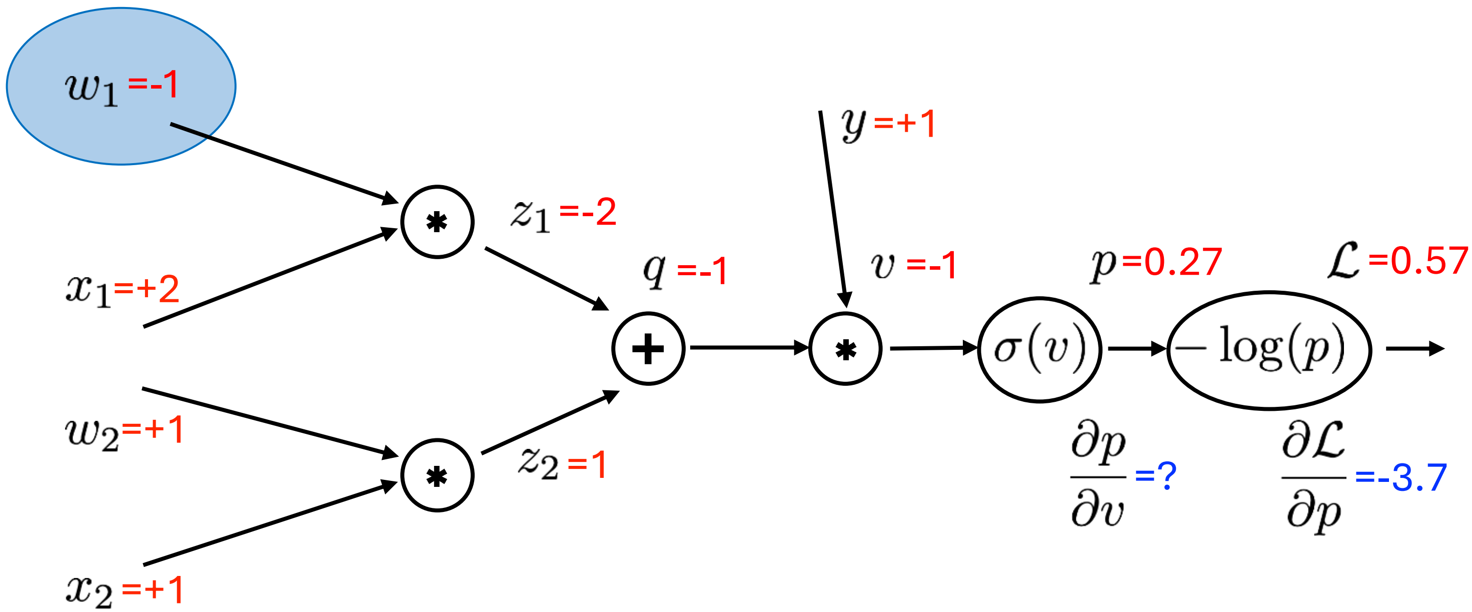
Backpropagation



$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial v}\frac{\partial v}{\partial q}\frac{\partial q}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial p} = \frac{\partial(-\log(p))}{\partial p} = -\frac{1}{p}}$$

Backpropagation



$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial v}\frac{\partial v}{\partial q}\frac{\partial q}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

Backpropagation

- Sigmoid function derivative

$$\sigma(\mathrm{v}) = \frac{1}{1 + \exp(-\mathrm{v})}$$

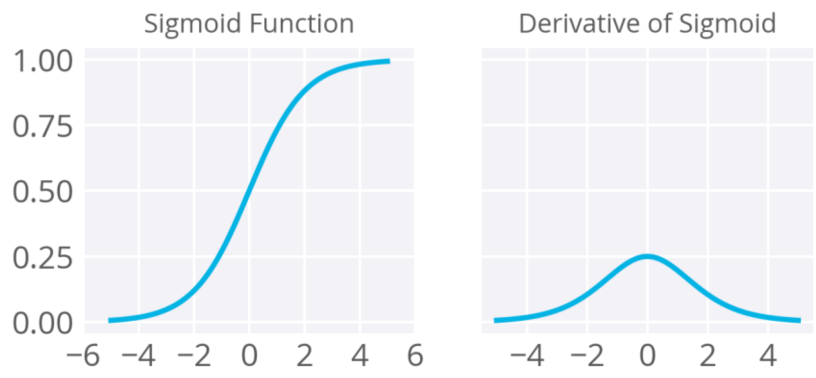
$$\frac{\partial \sigma(\mathrm{v})}{\partial \mathrm{v}} = -1 \cdot (1 + \exp(-\mathrm{v}))^{-2} \cdot \frac{\partial}{\partial \mathrm{v}} (1 + \exp(-\mathrm{v}))$$

$$= -1 \cdot (1 + \exp(-\mathrm{v}))^{-2} \cdot (-\exp(-\mathrm{v}))$$

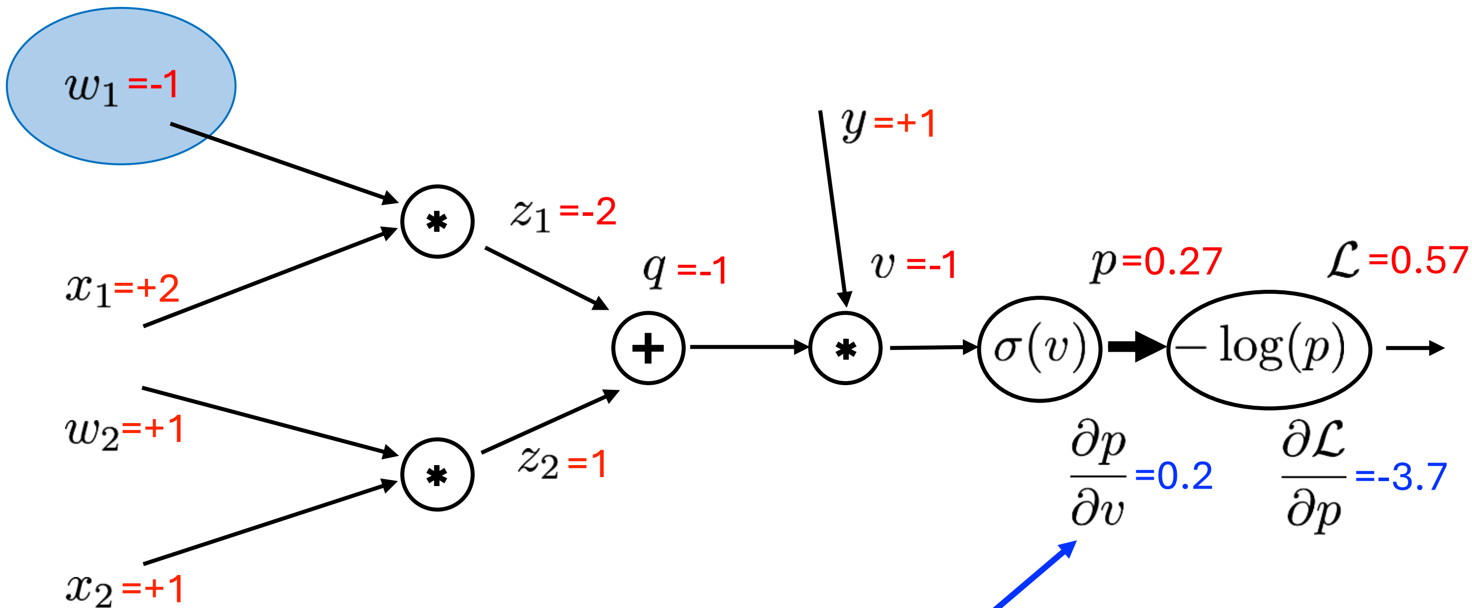
$$= (1 + \exp(-\mathrm{v}))^{-2} \cdot \exp(-\mathrm{v})$$

$$= \frac{1}{1 + \exp(-\mathrm{v})} \cdot \frac{\exp(-\mathrm{v})}{1 + \exp(-\mathrm{v})}$$

$$= \sigma(\mathrm{v}) \cdot (1 - \sigma(\mathrm{v}))$$



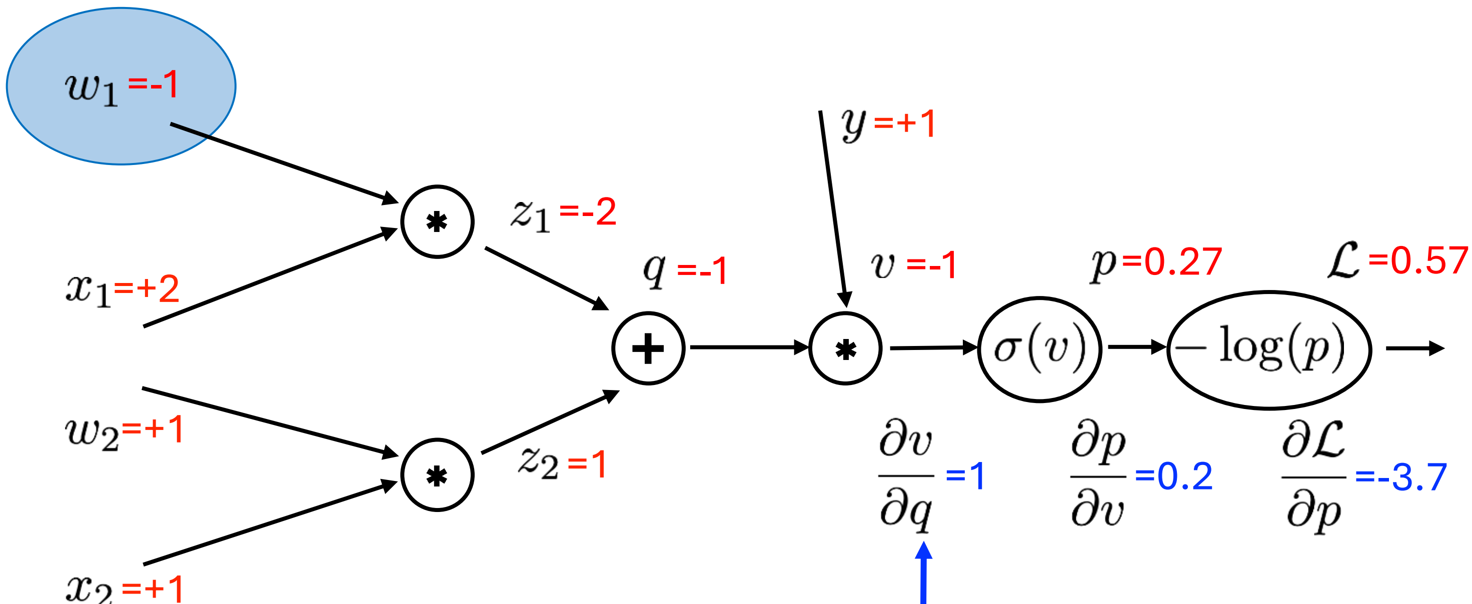
Backpropagation



$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p} \frac{\partial p}{\partial v} \frac{\partial v}{\partial q} \frac{\partial q}{\partial z_1} \frac{\partial z_1}{\partial w_1}$$

$$\boxed{\frac{\partial p}{\partial v} = \frac{\partial \sigma(v)}{\partial v} = \sigma(v)(1 - \sigma(v))}$$

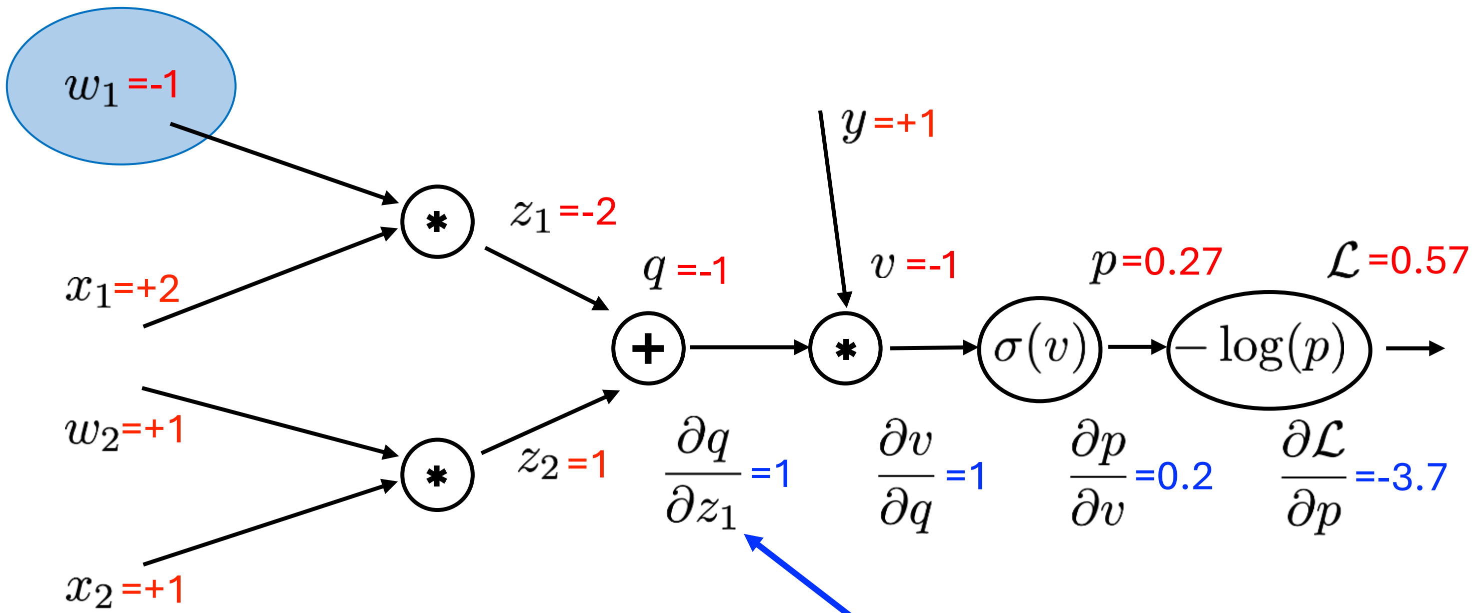
Backpropagation



$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial v}\frac{\partial v}{\partial q}\frac{\partial q}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

$$\boxed{\frac{\partial v}{\partial q} = \frac{\partial (yq)}{\partial q} = y}$$

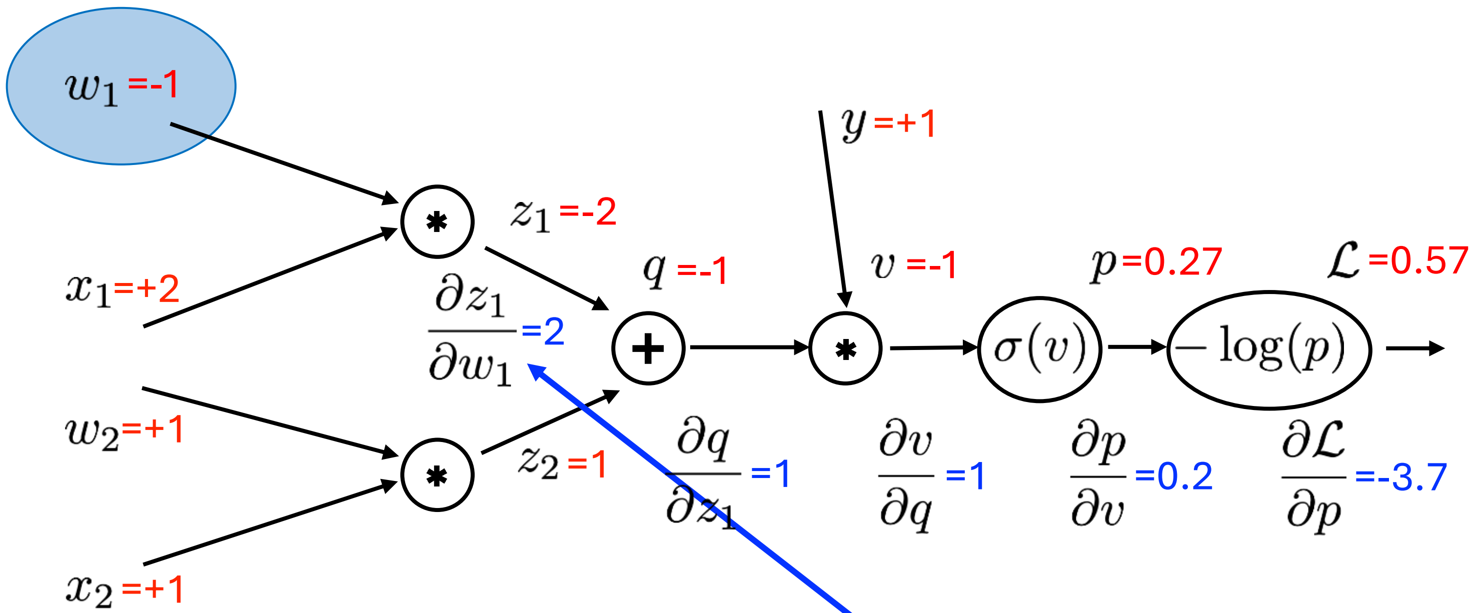
Backpropagation



$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial v}\frac{\partial v}{\partial q}\frac{\partial q}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

$$\boxed{\frac{\partial q}{\partial z_1} = \frac{\partial (z_1 + z_2)}{\partial z_1} = 1}$$

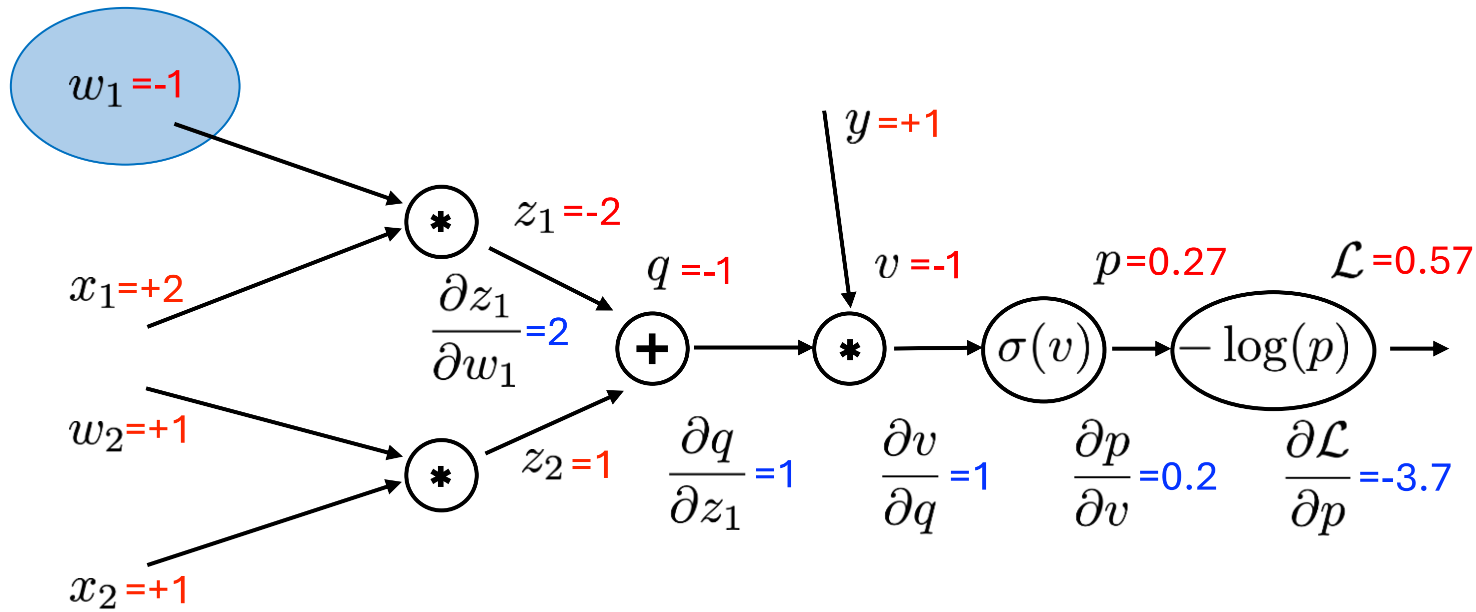
Backpropagation



$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial v}\frac{\partial v}{\partial q}\frac{\partial q}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

$$\boxed{\frac{\partial z_1}{\partial w_1} = \frac{\partial (w_1 x_1)}{\partial w_1} = x_1}$$

Backpropagation

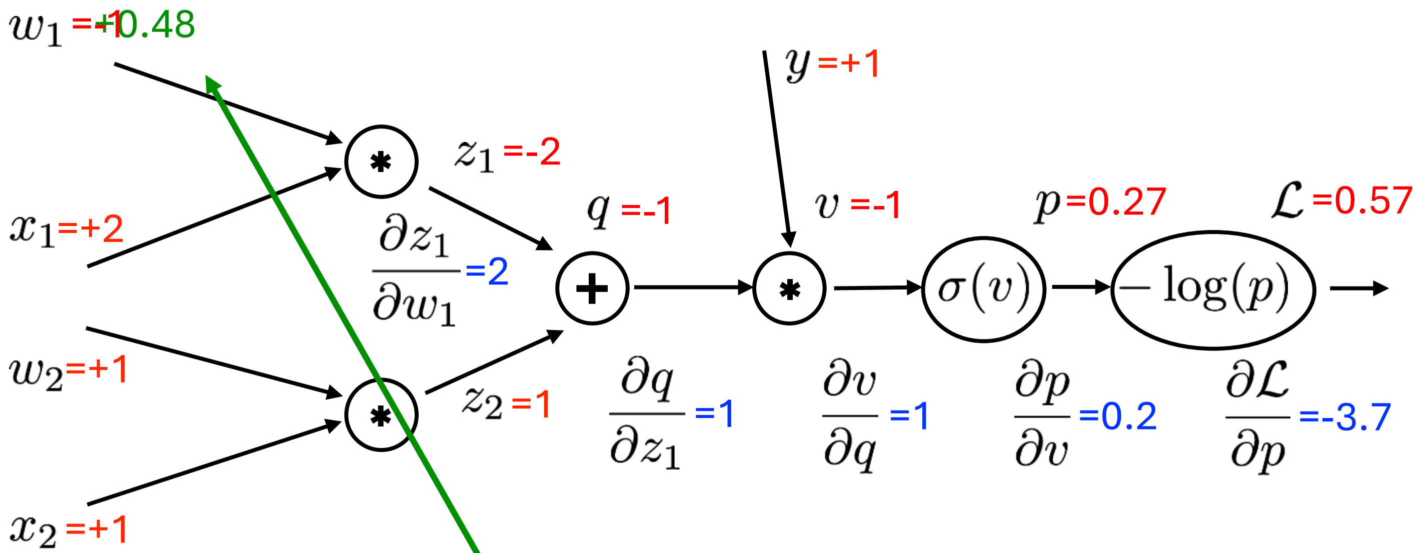


$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial v}\frac{\partial v}{\partial q}\frac{\partial q}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$ =-3.7*0.2*1*1*2 =-1.48

$$w_1 = w_1 - \alpha \frac{\partial \mathcal{L}}{\partial w1}$$

Learning Rate α = 1

Backpropagation

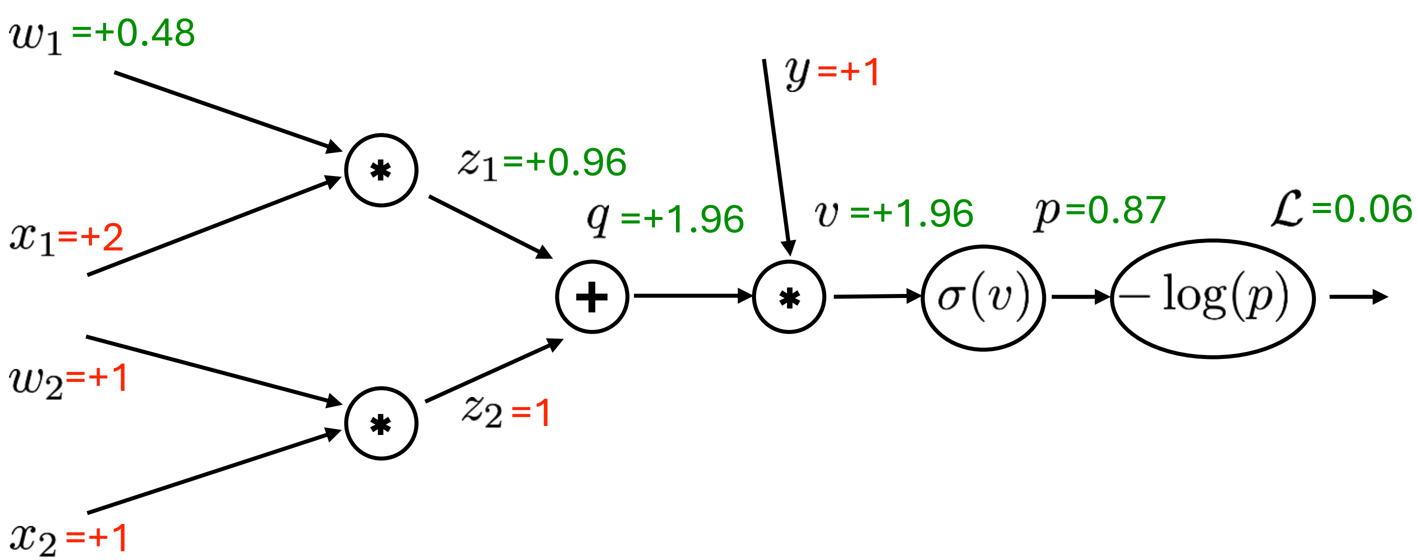


$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial v}\frac{\partial v}{\partial q}\frac{\partial q}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$ =-3.7*0.2*1*1*2 =-1.48

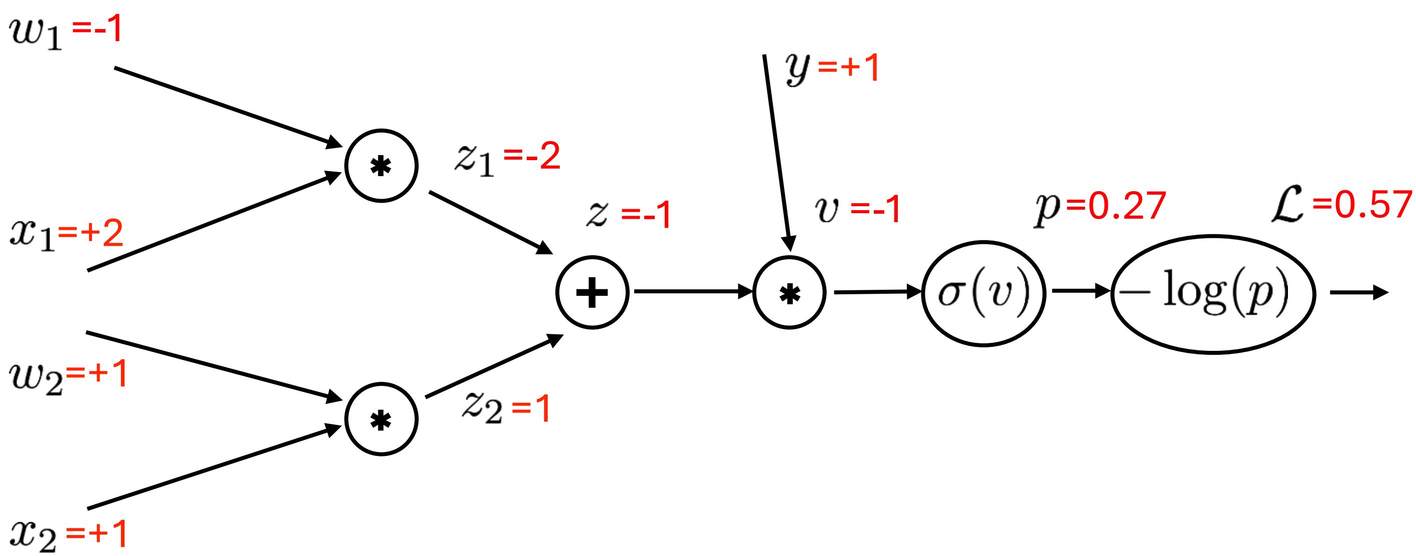
$$w_1 = w_1 - \alpha\frac{\partial \mathcal{L}}{\partial w1}$$ = +0.48

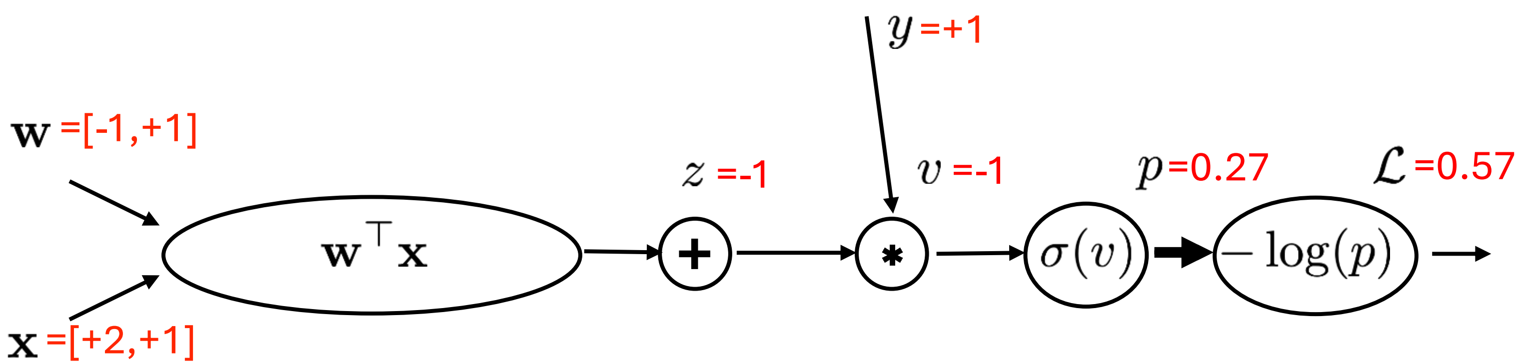
Learning Rate α = 1

Backpropagation

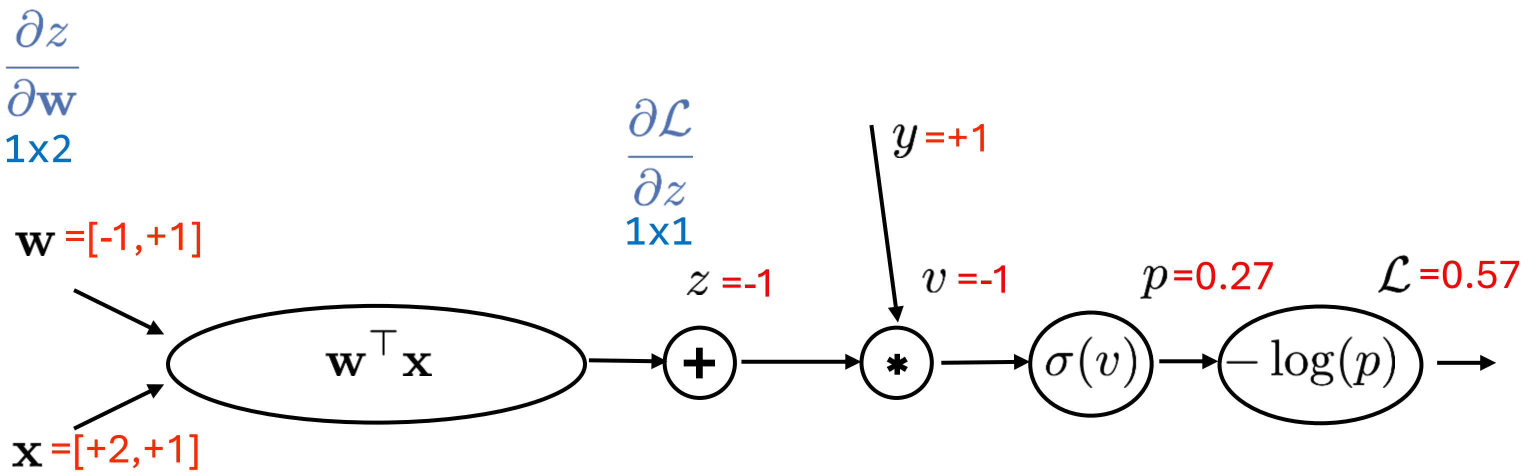


Backpropagation for vectors





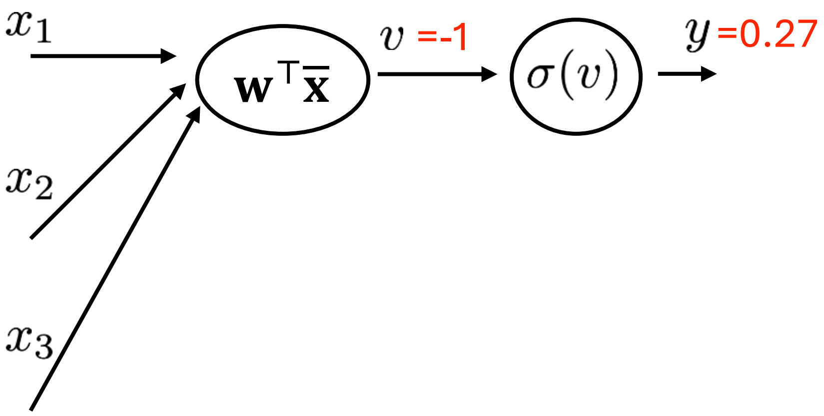
Backpropagation for vectors

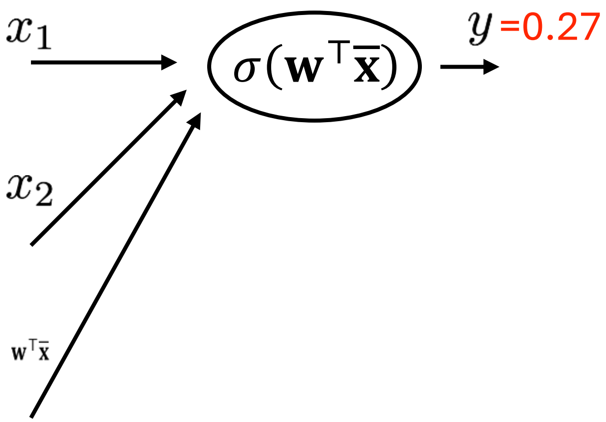


$$\underset{1\text{x}2}{\frac{\partial \mathcal{L}}{\partial \mathbf{w}}} = \underset{1\text{x}1}{\frac{\partial \mathcal{L}}{\partial z}} \underset{1\text{x}2}{\frac{\partial z}{\partial \mathbf{w}}}$$

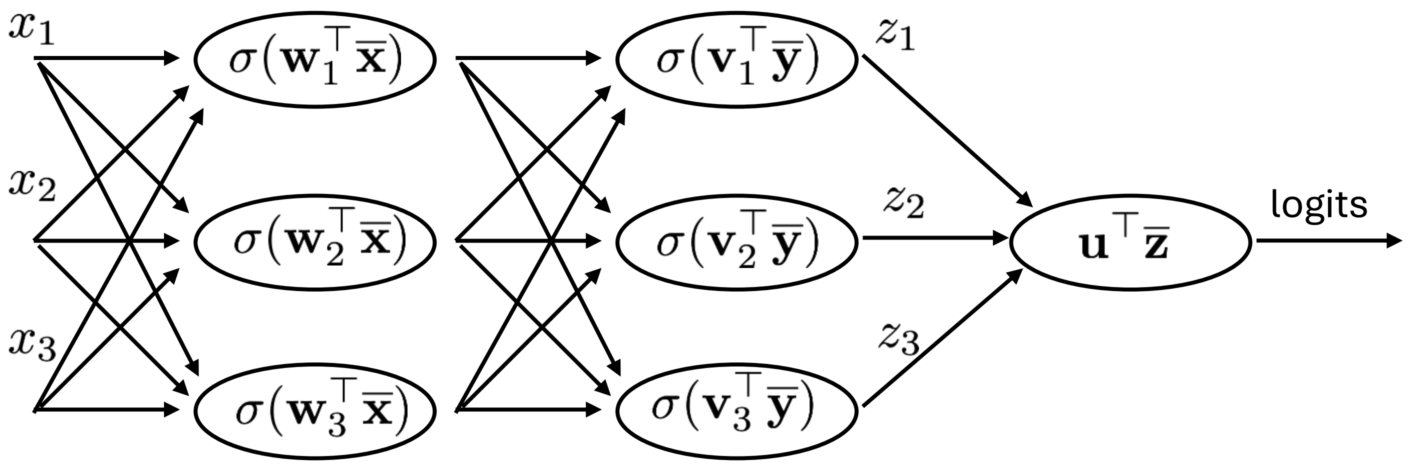
Multi-layer perceptron (MLP)

- Aka fully-connected neural network, fully-connected layers
- Consists of at least two layers of “artificial neurons”

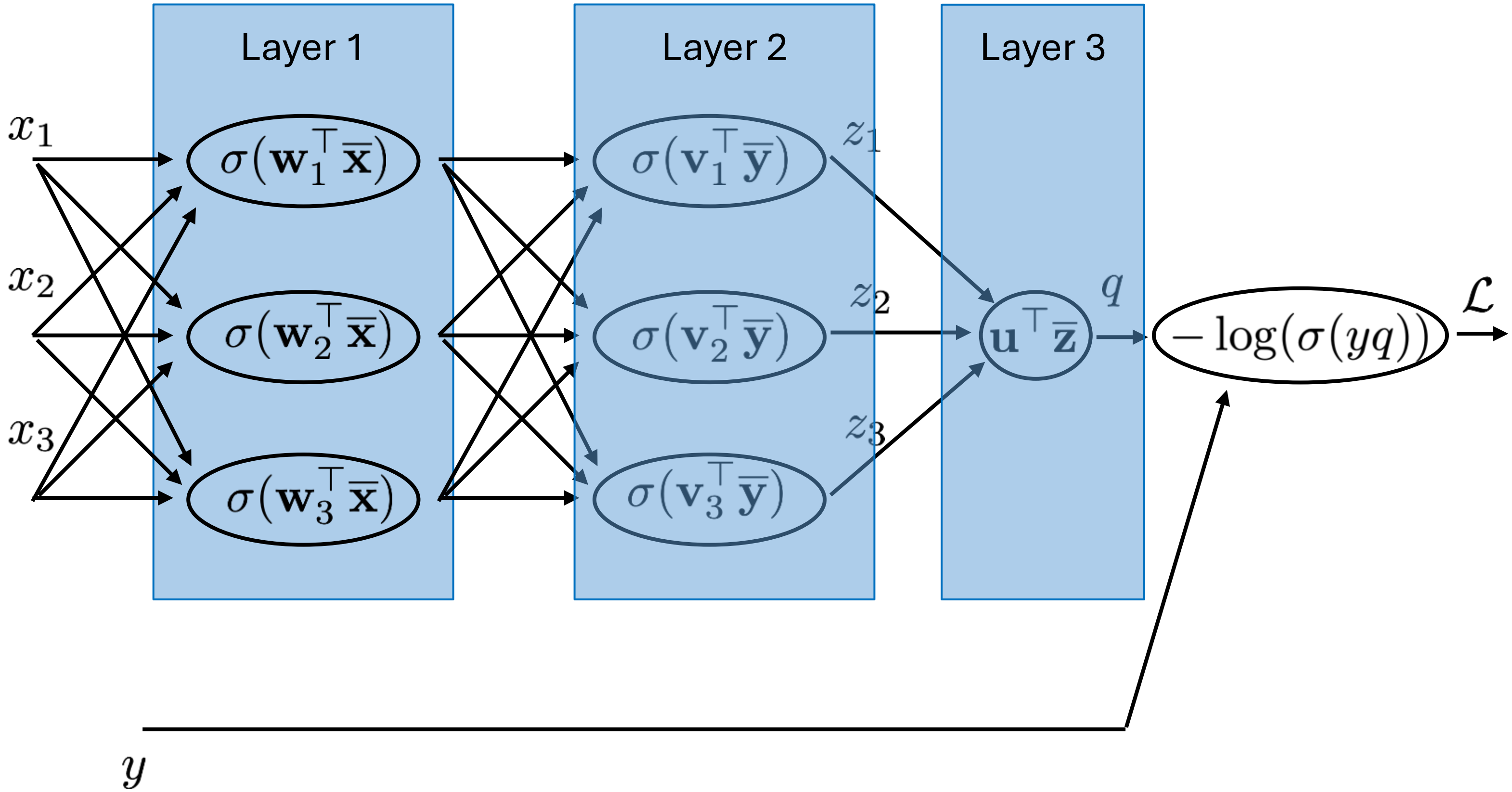




Multi-layer perceptron (MLP)

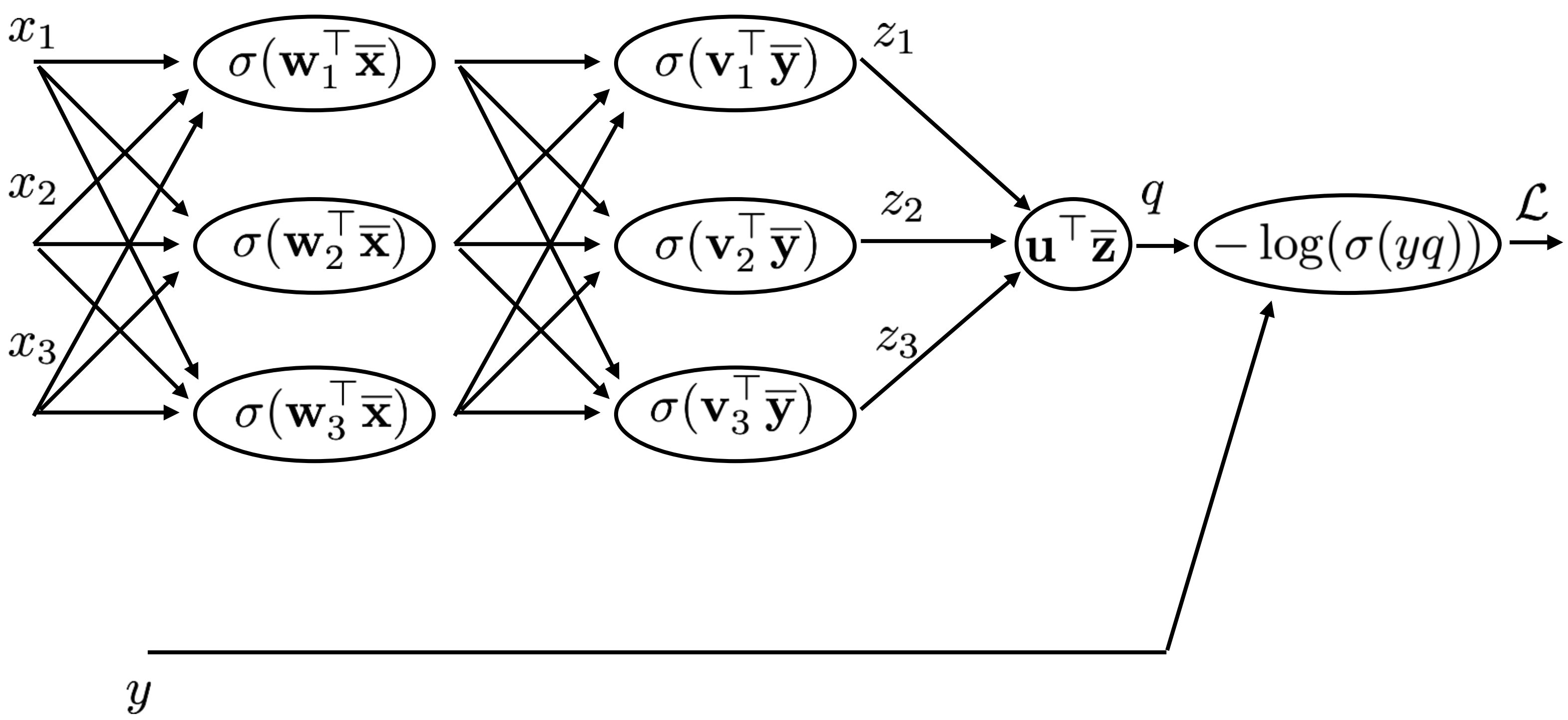


Multi-layer perceptron (MLP)



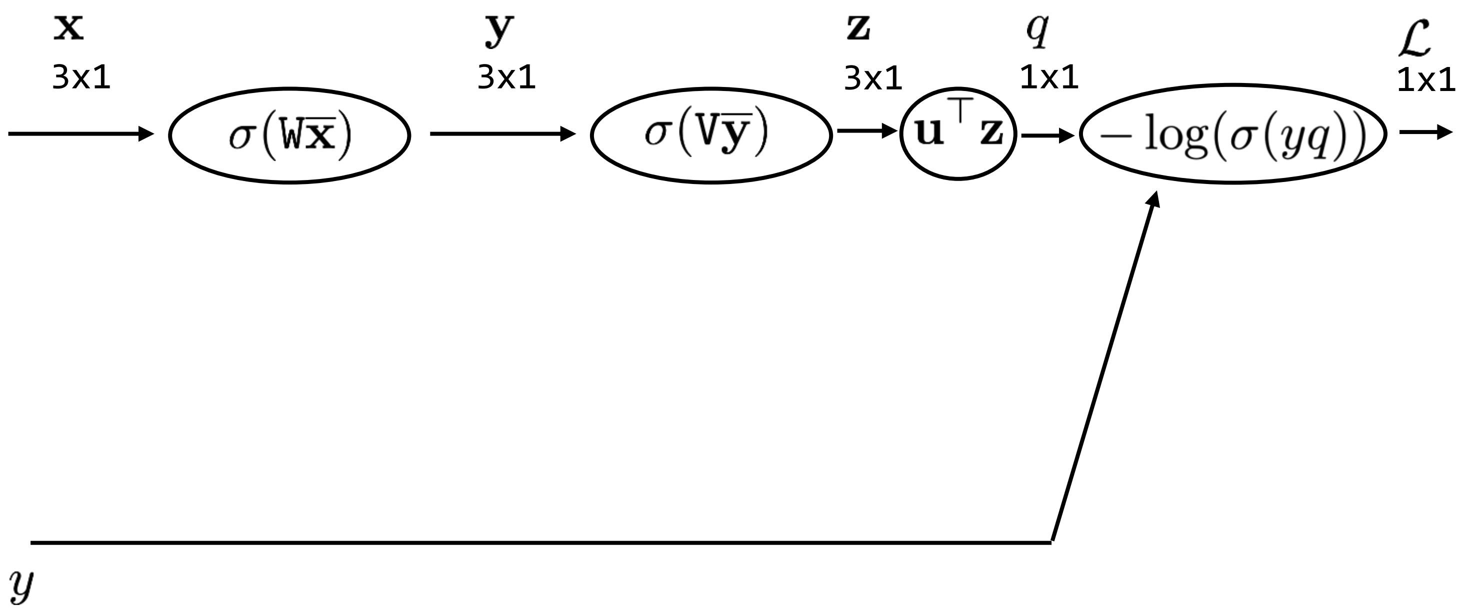
Multi-layer perceptron (MLP)

- Can approximate arbitrary continuous function when number of hidden layers and training samples $\rightarrow \infty$ (*Universal approximation theorem*)
- Computationally expensive for high-dimensional data (images, videos)
- Extremely prone to overfitting



Multi-layer perceptron (MLP)

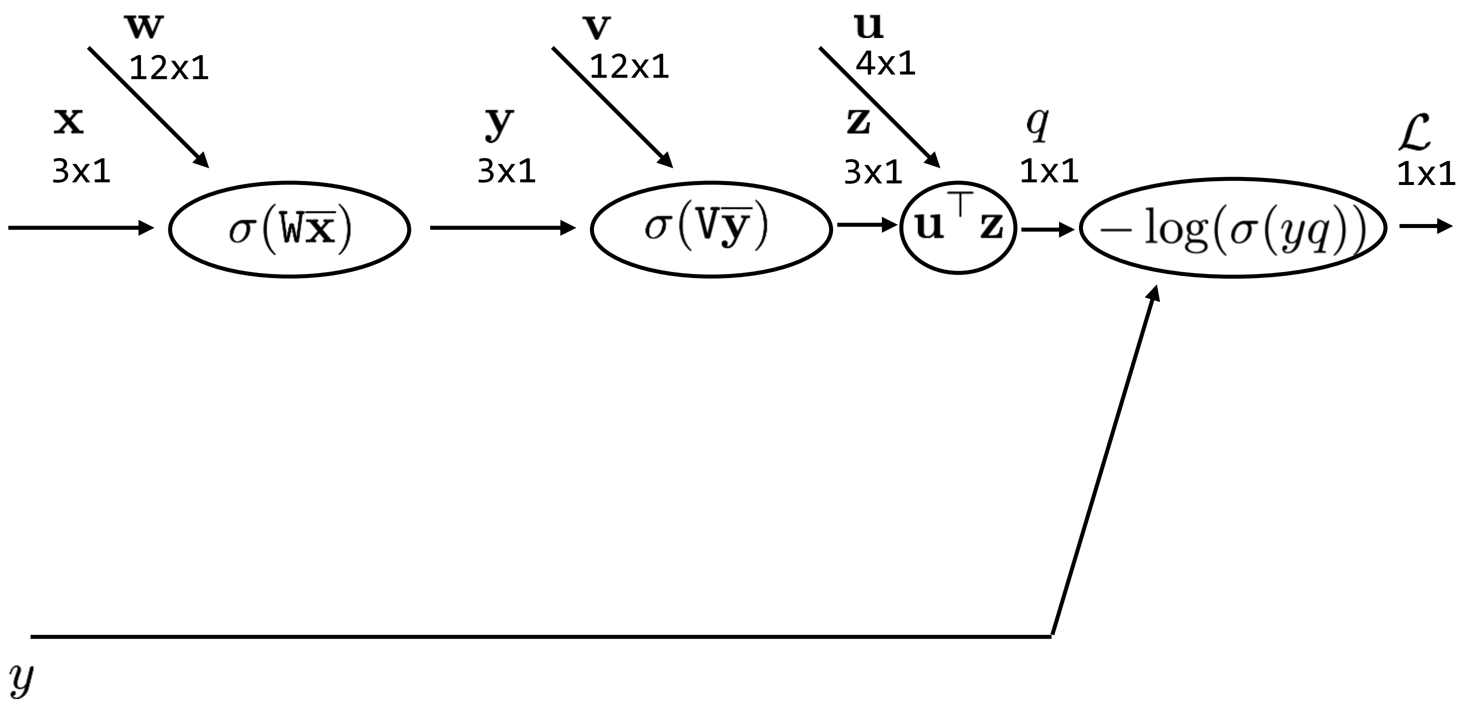
- **MLP in vector notation**



Multi-layer perceptron (MLP)

- **MLP in vector notation**

$$\mathbf{w} = \mathrm{vec}(\mathtt{W}) \qquad \mathbf{v} = \mathrm{vec}(\mathtt{V})$$



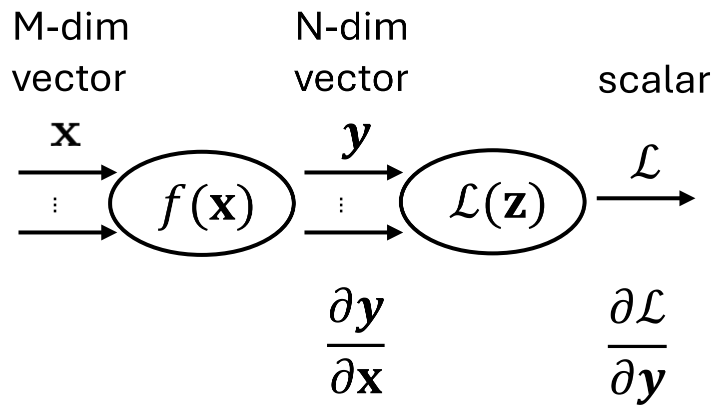
Jacobian matrix

- For a function $f\,(\mathbf{x})$: $\mathbb{R}^{\mathrm{n}} \to \mathbb{R}^{\mathrm{m}}$ the Jacobian matrix J_{f} is defined as

$$\mathbf{J}_{\mathbf{f}} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathsf{T}} f_1 \\ \vdots \\ \nabla^{\mathsf{T}} f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}_{\mathrm{NxM}}$$

- Each entry of the matrix is the partial derivative of the i-th element of the function $f\,(\mathbf{x})$ wrt to j-th element of the input $\mathbf{x}$

Chain rule for Jacobians



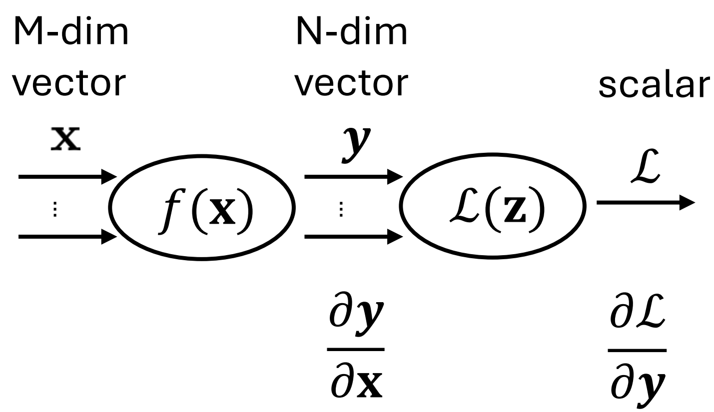
$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial \mathbf{x}}$$

Jacobian
1xM

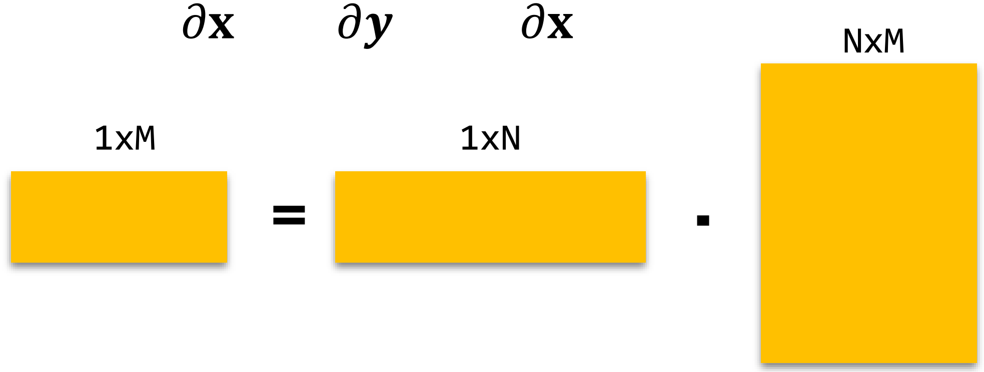
Jacobian
1xN

Jacobian
NxM

Chain rule for Jacobians

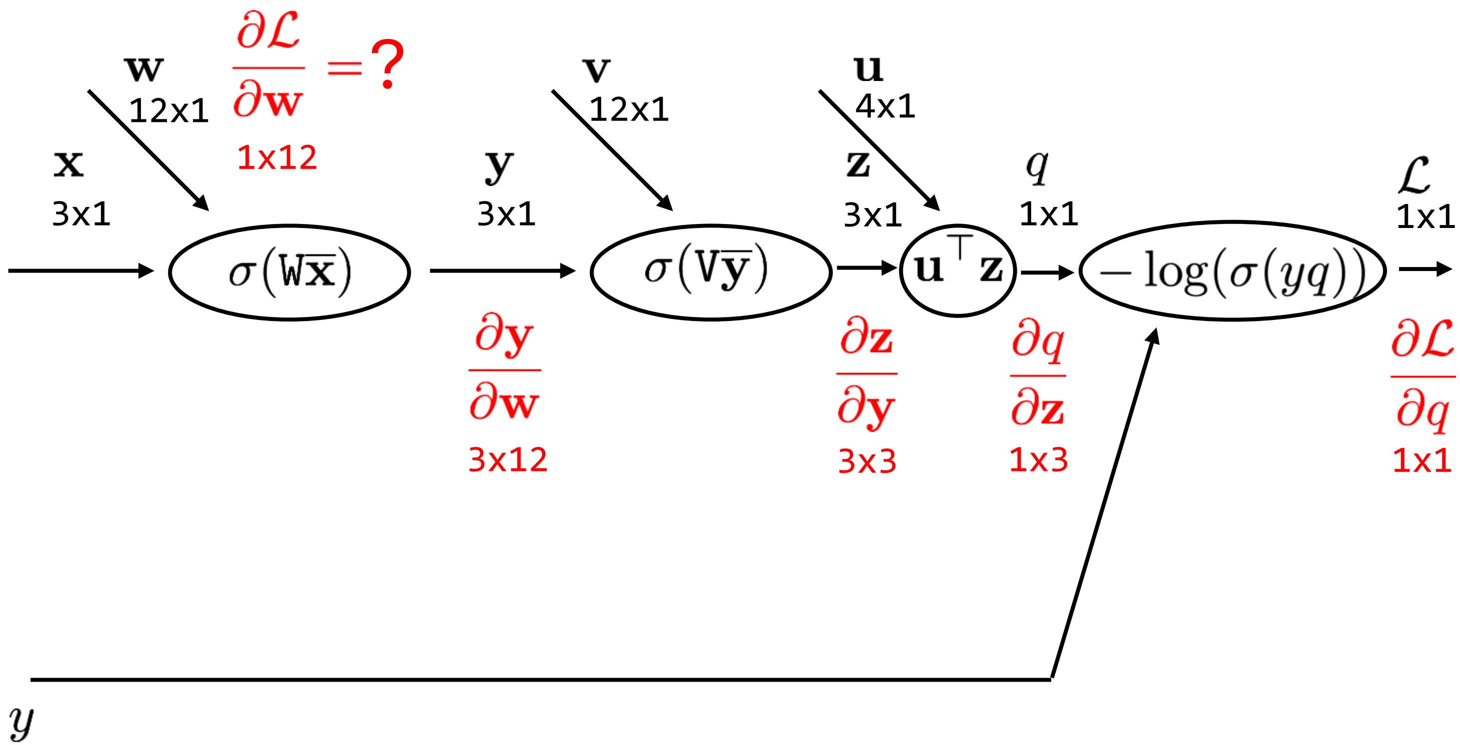


$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial \mathbf{x}}$$



Multi-layer perceptron (MLP)

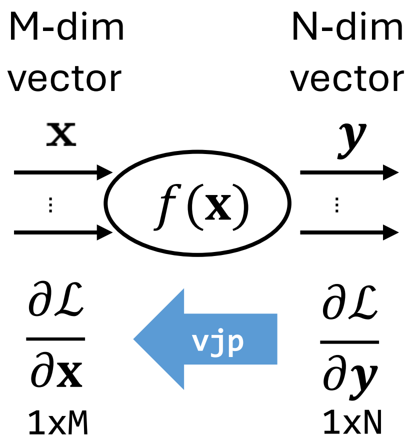
$\mathbf{w} = \mathrm{vec}(\mathtt{W}) \qquad \mathbf{v} = \mathrm{vec}(\mathtt{V})$



$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial q}\frac{\partial q}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{w}}$$

(1x12) (1x1)(1x3)(3x3)(3x12)

Vector-Jacobian Product




```
def vjp(v, (x,y)):
        return v^T · ∂f(x)/∂x
```

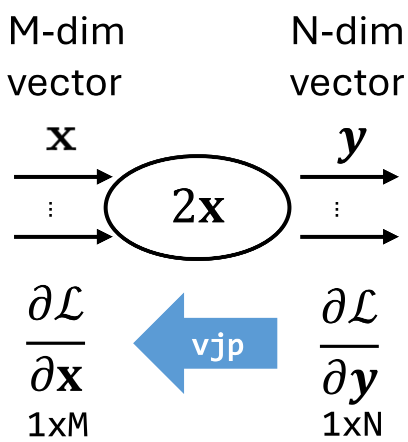
$$\mathbf{v}^{\top} \cdot \boxed{\frac{\partial \mathrm{f}(\boldsymbol{x})}{\partial \boldsymbol{x}}}$$

$$\mathrm{J} = \frac{\partial \mathrm{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{y}}{\partial \mathbf{x}}$$

NxM

$$\mathrm{vjp}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}}, (x, y)\right) = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}}$$

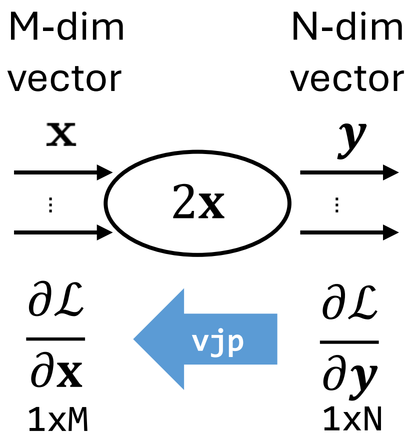
Vector-Jacobian Product




```
def vjp(v, (x,y)):
```

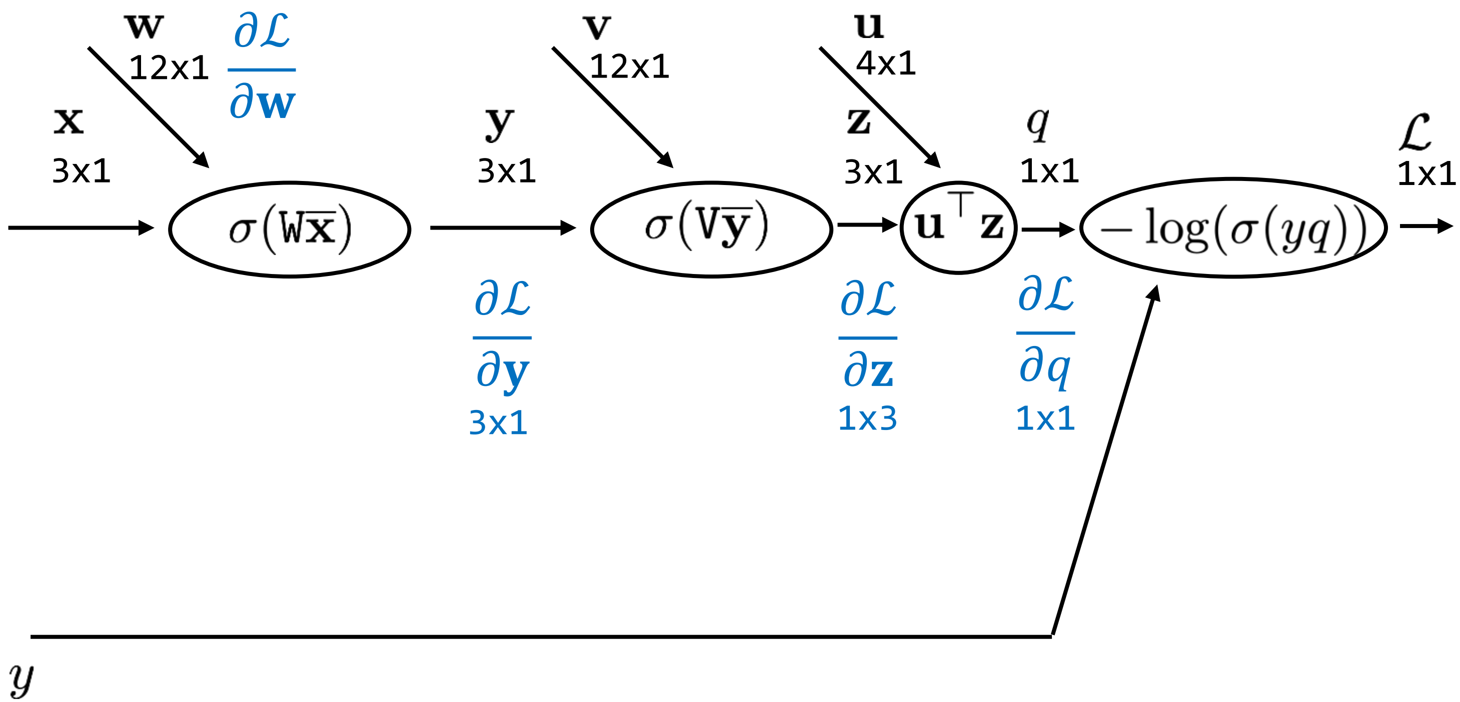
$$\text{return} \quad \mathbf{v}^{\top} \cdot \begin{bmatrix} 2 & & & & & \\ & 2 & & & & \\ & & 2 & & & \\ & & & \ddots & & \\ & & & & 2 & \\ & & & & & 2 \end{bmatrix}$$

Vector-Jacobian Product




```
def vjp(v, (x,y)):


    return 2 * v
```

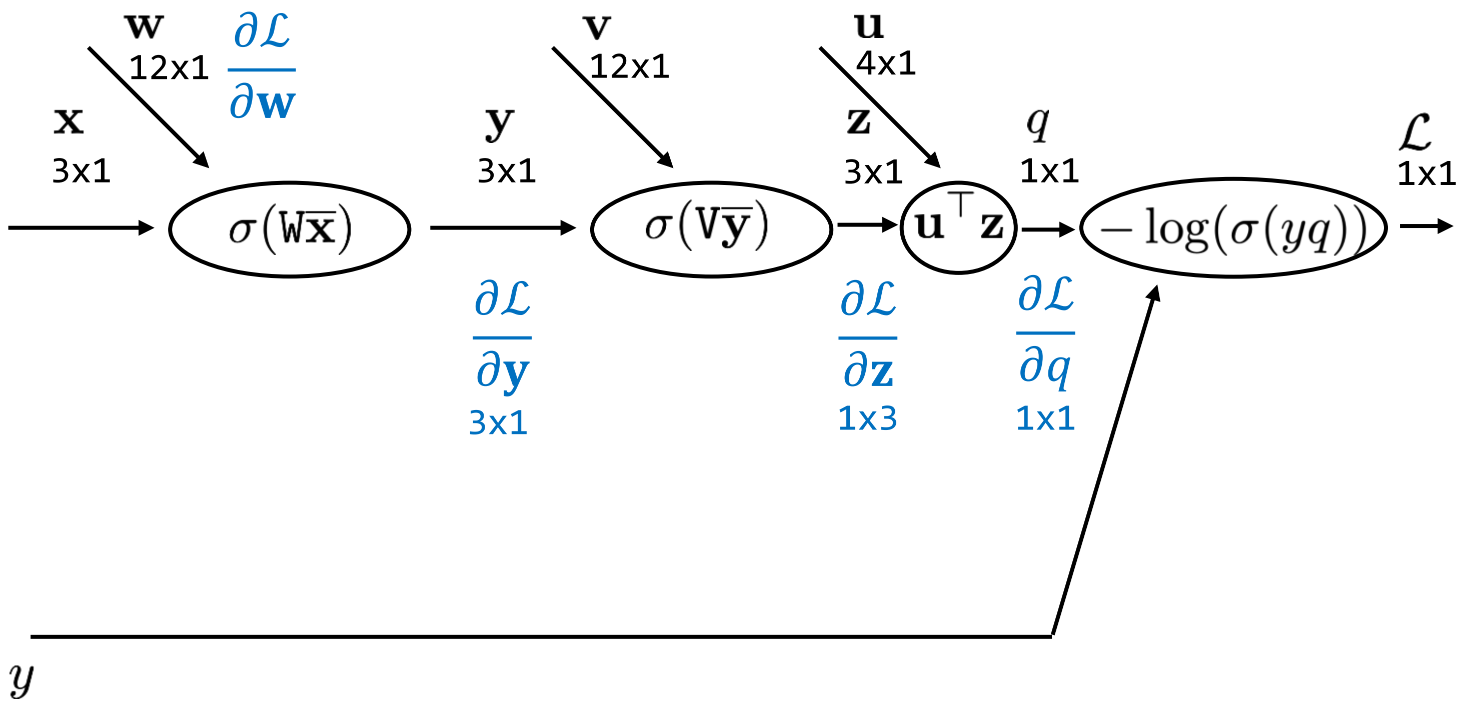
Vector-Jacobian Product



$$\underset{(1\times12)}{\frac{\partial \mathcal{L}}{\partial \mathbf{w}}} = \cancel{\underset{(1\times1)}{\frac{\partial \mathcal{L}}{\partial q}} \underset{(1\times3)}{\frac{\partial q}{\partial \mathbf{z}}} \underset{(3\times3)}{\frac{\partial \mathbf{z}}{\partial \mathbf{y}}} \underset{(3\times12)}{\frac{\partial \mathbf{y}}{\partial \mathbf{w}}}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial w} = \texttt{vjp(}\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}}\texttt{,(w, x))}$$

Vector-Jacobian Product

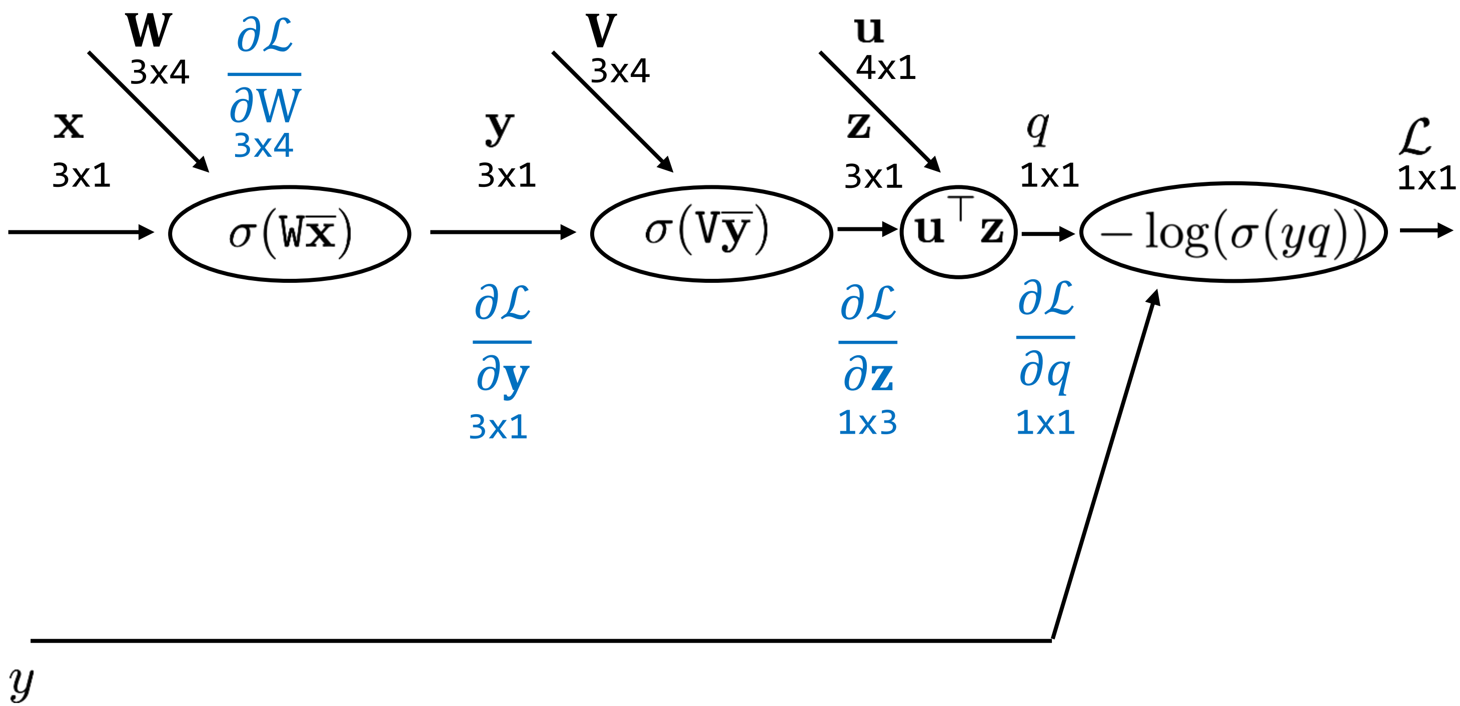
1. Usually more efficient (building MxN Jacobians not required)
2. Preserves dimensionality of the inputs



$$\underset{(1\times12)}{\frac{\partial \mathcal{L}}{\partial \mathbf{w}}} = \cancel{\underset{(1\times1)(1\times3)(3\times3)(3\times12)}{\frac{\partial \mathcal{L}}{\partial q}\frac{\partial q}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{w}}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial w} = \texttt{vjp(}\frac{\partial \mathcal{L}}{\partial \mathbf{y}}\texttt{,(w, x))}$$

Vector-Jacobian Product

- We can implement Vector-Jacobian Product to preserve matrix notation
- The resulting matrix is reshaped Jacobian / **gradient** of loss $\mathcal{L}$ wrt to the elements of $\mathbf{W}$



Autograd

- Feature of modern deep-learning frameworks that allows automated gradient calculation by defining VJPs for all supported operations

```
x = torch.tensor([[1,2], [3, 4]], requires_grad=True)
z = x.sum()
print(z)
tensor(10, grad_fn=<SumBackward0>)
```

```
loss = torch.sigmoid(x).sum()
grad = torch.autograd.grad(loss, x)[0]
print(grad)
tensor([[0.1966, 0.1050],
        [0.0452, 0.0177]])
```

Autograd

- Feature of modern deep-learning frameworks that allows automated gradient calculation by defining VJPs for all supported operations

```
prediction = model(some_input)

loss = (ideal_output - prediction).pow(2).sum()
print(loss)
tensor(126.2008, grad_fn=<SumBackward0>)

loss.backward()
print(model.layer2.weight.grad[0][0:5])
tensor([0.1966, 0.1050, 0.0452, 0.0177, 0.1050])
```

```
for _, (inputs, labels) in enumerate(training_data):
      optimizer.zero_grad()                 # Zero gradients
      logits = model(inputs)                # Make predictions

      loss = loss_fn(logits, labels)        # 1. Compute the loss
      loss.backward()                       # 2. Calculate gradients
      optimizer.step()                      # 3. Update weights
```

Summary

- Neural net is a function created as concatenation of simpler functions (neurons)

- **Multi-Layer Perceptron** (MLP) = fully-connected neural network where all outputs are connected to all inputs of the previous layer

- Training neural networks = gradient optimization of neuron weights wrt to some loss

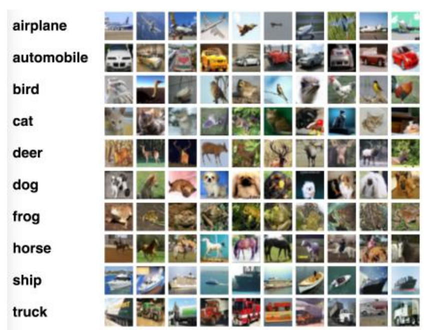
- Gradient calculation is implemented as backward concatenation of Vector-Jacobian Products (no numeric or symbolic differentiation)

- Deep learning frameworks already have very efficient implementations of VJP

Summary

Error

	Linear	MLP	CNN Next lecture!
MNIST	8%	2%	0.2%
CIFAR-10	63%	55%	1%



Competencies gained for the test

- Ability to draw a computational graph
- Compute backpropagation in computational graph
- Single and multi-layer perceptron, their pros and cons