

CSC 144V: Autonomous Vehicles

Lecture 2.2

Intro to Neural Networks Exercises Solution

Department of Computer Science,
Hofstra University

Exercise: XOR with a hidden layer

Step neurons output 1 if $w \cdot x + b > 0$, else 0. Hidden neuron h_1 = OR: $w = (1, 1)$, $b = -0.5$. Hidden neuron h_2 = NAND: $w = (-1, -1)$, $b = 1.5$. Output $y = h_1$ AND h_2 : $w = (1, 1)$, $b = -1.5$.

(a) Fill in the table. (b) Which function does the network compute? (c) Why can no single perceptron compute it?

x_1	x_2	h_1 (OR)	h_2 (NAND)	y
0	0			
0	1			
1	0			
1	1			

Exercise Solution: XOR with a hidden layer

- (b) The network computes **XOR**.
- (c) The classes $\{(0,0), (1,1)\}$ and $\{(0,1), (1,0)\}$ are **not linearly separable**: no single line $w \cdot x + b = 0$ separates them. In the hidden space (h_1, h_2) the points become $(0,1), (1,1), (1,1), (1,0)$, and the line $h_1 + h_2 = 1.5$ separates them.
- Without a non-linear activation the two layers would collapse into one linear map and XOR would stay impossible.

x_1	x_2	h_1 (OR)	h_2 (NAND)	y
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

Exercise: Activation functions

- (a) Fill in the table (ELU with $\alpha = 1$).
- (b) A network has 10 sigmoid layers. The derivative of the sigmoid is at most 0.25. Give an upper bound on the gradient factor contributed by the activations of all 10 layers.
- (c) Why does ReLU help, and what is its weakness?

z	$\sigma(z)$	$\sigma'(z) = \sigma(1 - \sigma)$	$\tanh(z)$	ReLU(z)	ELU(z)
-4					
-1					
0					
2					

Exercise Solution: Activation functions

- (b) $0.25^{10} \approx 9.5 \times 10^{-7}$: the gradient **vanishes** and the first layers barely learn.
- (c) ReLU has gradient 1 for $z > 0$ and is cheap, so gradients do not shrink. For $z < 0$ the gradient is 0: a neuron stuck there is **dead** ("dying ReLU"). Leaky ReLU and ELU keep a small non-zero slope for $z < 0$; ELU also converged fastest in Clevert et al. (2016).

z	$\sigma(z)$	$\sigma'(z) = \sigma(1 - \sigma)$	$\tanh(z)$	$\text{ReLU}(z)$	$\text{ELU}(z)$
-4	0.018	0.018	-0.999	0	-0.982
-1	0.269	0.197	-0.762	0	-0.632
0	0.5	0.25	0	0	0
2	0.881	0.105	0.964	2	2

Exercise: SoftMax and cross-entropy

A classifier outputs the logits $z = (1, 3, 0)$ for the classes car, pedestrian, cyclist. The true class is pedestrian (one-hot $y = (0, 1, 0)$).

(a) Compute the SoftMax probabilities p . (b) Compute the cross-entropy loss $L = -\ln p_{\text{true}}$. (c) Compute $\partial L / \partial z = p - y$.

(d) Add 10 to every logit: how do p and L change? (e) The network is confidently wrong: $z = (4, 0, 0)$ with the true class pedestrian. Compute L .

Class	z	$p = \exp(z) / \sum \exp(z)$	y	$p - y$
car	1		0	
pedestrian	3		1	
cyclist	0		0	
loss L				

Exercise Solution: SoftMax and cross-entropy

- (d) **Nothing changes**: SoftMax is invariant to adding a constant to all logits (the factor e^{10} cancels). Implementations subtract $\max(z)$ for numerical stability.
- (e) $p = (0.965, 0.018, 0.018)$, so $L = -\ln 0.018 \approx 4.04$, about 24 times the loss of (b). Cross-entropy punishes confident wrong answers heavily; the gradient $p - y$ pushes the logit of the true class up and the others down.

Class	z	$p = \exp(z) / \sum \exp(z)$	y	$p - y$
car	1	0.114	0	+0.114
pedestrian	3	0.844	1	-0.156
cyclist	0	0.042	0	+0.042
loss L		$-\ln 0.844 = 0.170$		

Exercise: Gradient descent: step size

Minimize $L(w) = (w - 3)^2$ by gradient descent, $w \leftarrow w - \eta \cdot dL/dw$, starting at $w_0 = 0$ ($dL/dw = 2(w - 3)$).

(a) Fill in the table for three learning rates. (b) Express $w(k+1) - 3$ as a function of $w(k) - 3$: for which η does it converge?

(c) With $N = 50,000$ training samples and a mini-batch size of 100, how many parameter updates are made in 10 epochs?

η	w_1	w_2	w_3	Behavior
0.1				
1.0				
1.2				

Exercise Solution: Gradient descent: step size

- (b) $w(k+1) - 3 = (1 - 2\eta)(w(k) - 3)$: converges iff $|1 - 2\eta| < 1$, i.e. $0 < \eta < 1$. $\eta = 1$ oscillates between 0 and 6 forever; $\eta = 1.2$ multiplies the error by -1.4 each step.
- (c) $50,000 / 100 = 500$ updates per epoch, so **5,000** updates in 10 epochs. Mini-batches make each step cheap and add gradient noise that can help escape poor local minima.

η	w_1	w_2	w_3	Behavior
0.1	0.6	1.08	1.464	converges slowly
1.0	6	0	6	oscillates
1.2	7.2	-2.88	11.23	diverges

Exercise: Backprop on a small graph

Computational graph $f(x, y, z) = (x + y) \cdot z$ with $x = -2$, $y = 5$, $z = -4$. Let $q = x + y$.

(a) Forward pass: compute q and f . (b) Backward pass: compute $\partial f / \partial f$, $\partial f / \partial q$, $\partial f / \partial z$, $\partial f / \partial x$, $\partial f / \partial y$ with the chain rule.

Node	Forward value	Gradient $\partial f / \partial \text{node}$
f		
q = x + y		
z	-4	
x	-2	
y	5	

Exercise Solution: Backprop on a small graph

- (b) Multiply gate: each input gets the **other input** times the upstream gradient: $\partial f / \partial q = z = -4$, $\partial f / \partial z = q = 3$.
- Add gate: **distributes** the upstream gradient unchanged: $\partial f / \partial x = \partial f / \partial q \cdot 1 = -4$ and $\partial f / \partial y = -4$.
- Chain rule: [local gradient] $\times$ [upstream gradient]. If a variable is used in several places, the gradients **add up**.

Node	Forward value	Gradient $\partial f / \partial \text{node}$
f	-12	1
q = x + y	3	-4
z	-4	3
x	-2	-4
y	5	-4

Exercise: Backprop: sigmoid neuron

One neuron $f = \sigma(w_0x_0 + w_1x_1 + w_2)$ with $w = (2, -3, -3)$ and $x = (-1, -2)$.

(a) Compute $s = w_0x_0 + w_1x_1 + w_2$ and $f = \sigma(s)$. (b) Using $\partial f / \partial s = f(1 - f)$, compute all gradients in the table.

(c) Apply one gradient-descent step on the weights with $\eta = 1$ (treat f as the loss). What are the new s and f ?

s	f	$\partial f / \partial s$	$\partial f / \partial w_0$	$\partial f / \partial w_1$	$\partial f / \partial w_2$	$\partial f / \partial x_0$	$\partial f / \partial x_1$

Exercise Solution: Backprop: sigmoid neuron

- (c) $w = w - \eta \partial f / \partial w = (2.197, -2.607, -3.197)$, so $s = -2.197 + 5.214 - 3.197 = \textbf{-0.18}$ and $f = \sigma(-0.18) \approx \textbf{0.455}$ (down from 0.731).
- The local gradient of the sigmoid gate is $f(1 - f) = 0.73 \cdot 0.27 \approx 0.2$. Every input gradient is [its multiplier] $\times$ 0.2: $\partial f / \partial w_0 = x_0 \cdot 0.2$ and $\partial f / \partial x_0 = w_0 \cdot 0.2$.

s	f	$\partial f / \partial s$	$\partial f / \partial w_0$	$\partial f / \partial w_1$	$\partial f / \partial w_2$	$\partial f / \partial x_0$	$\partial f / \partial x_1$
1	0.731	0.197	-0.197	-0.393	0.197	0.393	-0.590

Exercise: Counting parameters

Count the learnable parameters (weights + biases) and compare fully connected (FC) layers with a convolution.

(a) Fill in the table. float32 uses 4 bytes per parameter. (b) Why is an FC layer a poor choice for camera images, and what does a CNN do differently?

Layer	Formula	Parameters
FC 784 → 128	$784 \cdot 128 + 128$	
FC 128 → 10	$128 \cdot 10 + 10$	
MLP 784-128-10 (total)	sum of the two	
ALVINN-like 960-4-30 (30×32 input)	$(960 \cdot 4 + 4) + (4 \cdot 30 + 30)$	
FC from 1920×1080×3 pixels to 1,000 neurons	$1920 \cdot 1080 \cdot 3 \cdot 1000 (+ 1000)$	
CONV 3×3, 3 → 64 channels	$(3 \cdot 3 \cdot 3) \cdot 64 + 64$	

Exercise Solution: Counting parameters

- (b) An FC layer needs one weight per input pixel and neuron: the count **explodes with image size**, the input size is fixed, and there is no translation invariance, so it overfits easily. A CONV layer **shares** a small kernel across the image: its parameter count does not depend on the image size (the lecture's MLP vs. CNN slide).

Layer	Formula	Parameters
FC 784 → 128	$784 \cdot 128 + 128$	100,480
FC 128 → 10	$128 \cdot 10 + 10$	1,290
MLP 784-128-10 (total)	sum of the two	101,770
ALVINN-like 960-4-30 (30×32 input)	$(960 \cdot 4 + 4) + (4 \cdot 30 + 30)$	3,994
FC from 1920×1080×3 pixels to 1,000 neurons	$1920 \cdot 1080 \cdot 3 \cdot 1000 (+ 1000)$	$\approx 6.2 \times 10^9$ (≈ 24.9 GB)
CONV 3×3, 3 → 64 channels	$(3 \cdot 3 \cdot 3) \cdot 64 + 64$	1,792

Exercise: Training curves and CV

Training and validation loss of a network at several epochs are given.

(a) Fill in the generalization gap and diagnose each epoch. (b) At which epoch would you stop (early stopping)?

(c) Name three ways to reduce the overfitting seen at epoch 40. (d) 5-fold cross-validation on 1,000 samples gives accuracies 82, 85, 80, 84 and 84 %. How many samples train and validate each fold, and what is the result?

Epoch	Train loss	Val loss	Gap (val – train)	Diagnosis
1	1.20	1.25		
5	0.60	0.70		
10	0.30	0.45		
20	0.10	0.50		
40	0.02	0.80		

Exercise Solution: Training curves and CV

- (b) Around **epoch 10**: the validation loss is lowest; afterwards training loss keeps falling while validation loss rises (overfitting). Keep the weights from that epoch.
- (c) **More (or augmented) training data**, **L2/L1 regularization**, **dropout**, a **smaller model**, or early stopping.
- (d) Each fold: **800 train / 200 validation**, every sample is used for validation exactly once. Result = average = **83 %** (std $\approx$ 2 points).

Epoch	Train loss	Val loss	Gap (val – train)	Diagnosis
1	1.20	1.25	0.05	underfitting (both high)
5	0.60	0.70	0.10	still learning
10	0.30	0.45	0.15	good fit, best val loss
20	0.10	0.50	0.40	overfitting starts
40	0.02	0.80	0.78	strong overfitting