Tutorial 1 exercices

course	🧠 Deep Learning
Status	new

1. **Questions on course structure and organization.**

(a) What are your expectations for this course? What do you hope to learn, and what level of effort and engagement do you expect will be required from you?

I expect to gain a deep understanding of how and why deep learning models learn, not just how to apply them. I hope to build strong intuition about architectures, optimization, and learning dynamics through hands on experimentation. The course will require sustained preparation, active participation, and serious time investment each week.

(b) How many ECTS credits is this course worth, and what does this imply in terms of weekly workload and personal investment?

This course is worth 6.5 ECTS credits. That corresponds to roughly 182 hours of total work, or about 20 hours per week on average. This implies a high and continuous personal workload beyond the contact hours.

(c) What is the purpose of the conceptual tutorials and the programming tutorials? How are you expected to participate in each, and how are you expected to share your work, questions, and difficulties with others?

Conceptual tutorials focus on reasoning, discussion, and understanding model behavior, while programming tutorials focus on implementing and experimenting with models in Python and PyTorch. In both, I am expected to come prepared, actively participate, and contribute to discussions. I should openly share my work, questions, and difficulties during the sessions.

(d) What is the course policy on the use of generative AI? How does this policy relate to the learning goals and grading philosophy of the course?

The use of generative AI is allowed but should not replace independent reasoning, coding, or analysis. Overreliance on AI tools undermines the learning objectives and will negatively affect participation and paper evaluation. The policy aligns grading with understanding, ownership, and insight rather than output alone.

(e) What is the role of the tutor in this course? What should you expect from the tutor, and what should the tutor expect from you?

The tutor's role is to guide discussion, challenge reasoning, and provide feedback rather than deliver solutions. I should expect support and critical questioning, not step by step answers. In return, the tutor expects me to be prepared, engaged, and reflective.

(f) Based on the syllabus, what concrete signs would indicate that you are engaging with the course in a way that is likely to lead to a good evaluation?

Clear signs of good engagement include consistent preparation, active participation in tutorials, and the ability to clearly explain models and experimental results. Constructive questioning, sharing failures, and responding to peer feedback are also important. A thoughtful, well reasoned final paper that demonstrates ownership of the work strongly indicates good engagement.

Deep feedforward networks, also often called feedforward neural networks,
or multilayer perceptrons (MLPs)

The goal of a feedforward network is to approximate some function f^*.

for example:

classifier $y = f^*(x)$ maps to input x to category y.

A **feedforward network** defines a mapping $y = f^*(x; \theta)$ and learns the value of the parameters θ that result in the best function approximation.

These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f , and finally to the output y.

There are no **feedback** connections in which outputs of the model are fed back into itself.

if they do feedback connections they are called **recurrent neural networks**.

they are called networks because they are typically represented by composing together many different functions.

For example, we might have three functions $f^{(1)}, f^{(2)}, \text{and } f^{(3)}$ connected in a chain, to form $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$

In this case, $f^{(1)}$ is called the first **layer** of the network, $f^{(2)}$ is called the second layer, and so on.

the overal length of the chain is called the **length.**

The final layer of a feedforward network is called the **output layer.**

During training, the model is given input–output pairs (x, y) that specify what the **output layer** should produce, with $y \approx f^*(x)$.

The **hidden layers** are not directly supervised by the data; instead, the learning algorithm determines how they should transform the input to best approximate the target function f^*.

Each hidden layer of the network is typically vector-valued. The dimensionality of these hidden layers determines the **width** of the model.

Feedforward networks can be understood as extensions of linear models, which are efficient and reliable to train but fundamentally limited because they can only represent linear functions and therefore cannot capture interactions between input variables.

To extend linear models to represent nonlinear functions of x, we can apply the linear model not to x itself but to a transformed input $\phi(x)$, where ϕ is a non-linear transformation.

We can think of ϕ as providing a set of features describing x, or as providing a new representation for x.

how to chose ϕ:

1. One option is to use a very generic ϕ, such as the infinite-dimensional ϕ that is implicitly used by kernel machines based on the RBF kernel. If $\phi(x)$ is of high enough dimension, we can always have enough capacity to fit the training set, but generalization to the test set often remains poor. Very generic feature mappings are usually based only on the principle of local smoothness and do not encode enough prior information to solve advanced problems.
2. Another option is to manually engineer ϕ. Until the advent of deep learning, this was the dominant approach. This approach requires decades of human effort for each separate task, with practitioners specializing in different domains such as speech recognition or computer vision, and with little transfer between domains.
3. The strategy of deep learning is to learn ϕ. In this approach, we have a model $y = f(x; \theta, w) = \phi(x; \theta)^\top w$. We now have parameters θ that we use to learn φ from a broad class of functions, and parameters w that map from $\phi(x)$ to the desired output. This is an example of a deep feedforward network, with ϕ defining a hidden layer. This approach is the only one of the three that gives up on the convexity of the training problem, but the benefits outweigh the harms. In this approach, we parametrize the representation as $\phi(x; \theta)$ and use the optimization algorithm to find the θ that corresponds to a good representation. If we wish, this approach can capture the benefit of the first approach by being highly generic—we do so by using a very broad family $\phi(x; \theta)$. This approach can also capture the benefit of the second approach. Human practitioners can encode their knowledge to help generalization by designing families $\phi(x; \theta)$ that they expect will perform well. The advantage is that the human designer only needs to find the right general function family rather than finding precisely the right function

XOR function

The XOR function (“exclusive or”) is an operation on two binary values, x_1 and x_2. When exactly one of these binary values is equal to 1, the XOR function returns 1.

The XOR function provides the target function $y = f^*(x)$ that we want to learn.

Our model provides a function $y = f(x; \theta)$ and our learning algorithm will adapt the parameters θ to make f as similar as possible to f^*.

We want our network to perform correctly on the four points

$$\mathbb{X} = \{[0, 0]^\top, [0, 1]^\top, [1, 0]^\top, \text{and } [1, 1]^\top\}$$

We can treat this problem as a regression problem and use a mean squared error (MSE) **loss function**. (simplifcation)

Evaluated on our whole training set, the MSE loss function is

$$j(\theta) = \frac{1}{4} \sum_{x \in \mathbb{X}} (f^*(x) - f(x; \theta))^2$$

Online vs batch learning

- **Online learning:** parameters are updated after each single training example or very small mini batch. Updates are noisy but fast and can adapt quickly.
- **Batch learning:** parameters are updated using the whole training set. Updates are stable but expensive and slow.

online is like reading sentence by sentence and trying to answer questions where as batch learning is trying to read the whole text and then answer the question

In a multi layer perceptron

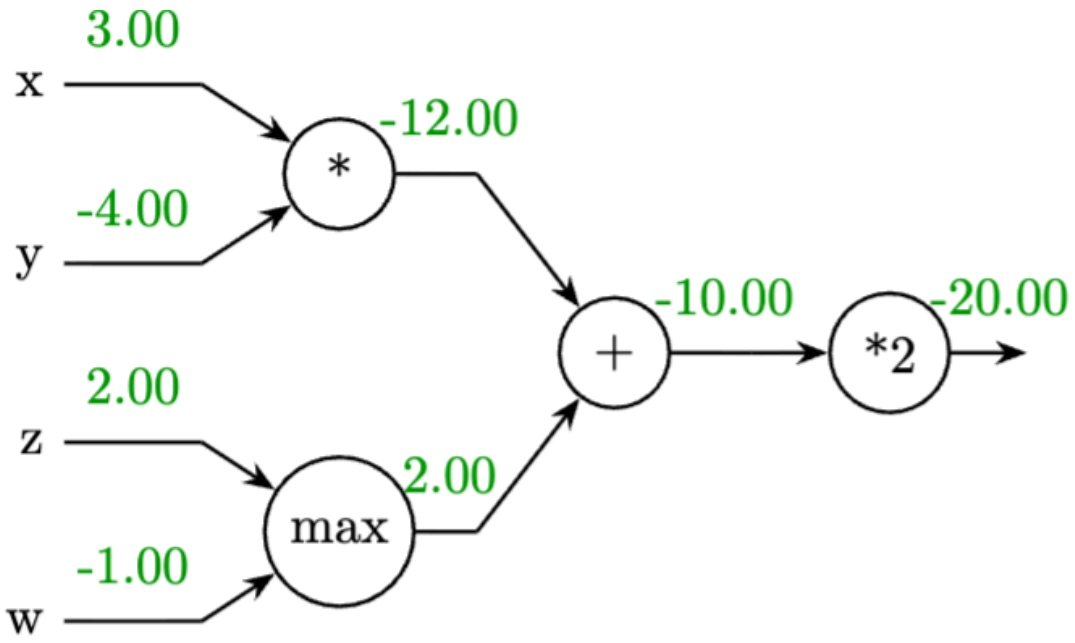
(a) **Iteration:** one parameter update step. In practice, one forward and backward pass on one mini batch or one example.

(b) **Epoch:** one full pass over the entire training dataset.

Why it matters

Iterations control how often the model updates. Epochs control how much data the model has seen. Confusing the two leads to wrong intuition about learning speed, convergence, overfitting, and the effect of learning rates and batch size on training dynamics.

Back propagation on a computational graph.
The function $f(x, y, z, w) = 2((xy) + \max(z, w))$



(a) performe a forward pass

$f(x, y, z, w) = 2((xy) + \max(z, w))$

$$
\begin{aligned}
a &= xy = 3 \cdot (-4) = -12 \\
m &= \max(z, w) = \max(2, -1) = 2 \\
b &= a + m = -12 + 2 = -10 \\
f &= 2b = -20
\end{aligned}
$$

(b) apply backpropagation to compute the gradient of the output with respect to each input variable.

From $f = 2b$:
$\frac{\partial f}{\partial b} = 2$

From $b = a + m$:
$\frac{\partial f}{\partial a} = 2$
$\frac{\partial f}{\partial m} = 2$

From $a = xy$:
$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial x} = 2 \cdot y = 2 \cdot (-4) = -8$
$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial y} = 2 \cdot x = 2 \cdot 3 = 6$

From $m = \max(z, w)$: since $z > w$ here, locally $m = z$
$\frac{\partial m}{\partial z} = 1,\ \frac{\partial m}{\partial w} = 0$

So
$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial m}\frac{\partial m}{\partial z} = 2 \cdot 1 = 2$
$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial m}\frac{\partial m}{\partial w} = 2 \cdot 0 = 0$

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial w}\right) = (-8,\ 6,\ 2,\ 0).$$

(c)Clearly indicate the local gradients and how they are combined using the chain rule.

Each operation contributes a local sensitivity. The overall sensitivity is obtained by multiplying these local sensitivities along the computation graph, from output back to input.

Add gate

(a) The output gradient is copied unchanged to each input.

(b) Independent of the forward values.

Multiply gate

(a) Each input receives the output gradient multiplied by the other input.

(b) Depends directly on the forward values of the inputs.

Max gate

(a) The full gradient is passed only to the input that achieved the maximum; the other gets zero.

(b) Depends on which input was larger during the forward pass.

(a) Forward computation

Let the pre activation be

$s = x + y$

Neuron output (tanh activation)

$z = \tanh(s) = \tanh(x + y)$

(b) Upstream, local, downstream gradients

Upstream gradient (comes from later in the network)

- $g \equiv \dfrac{\partial L}{\partial z}$

Local gradients (gate derivatives inside this neuron)

- $\dfrac{\partial z}{\partial s} = 1 - \tanh^2(s) = 1 - z^2$
- $\dfrac{\partial s}{\partial x} = 1$
- $\dfrac{\partial s}{\partial y} = 1$

Downstream gradients (what we send to earlier inputs)

- $\dfrac{\partial L}{\partial x}, \dfrac{\partial L}{\partial y}$

(c) Chain rule to get gradients w.r.t. inputs

First compute the gradient into s:

- $\frac{\partial L}{\partial s} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s} = g(1 - z^2)$

Then distribute to inputs:

- $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial s}\frac{\partial s}{\partial x} = g(1 - z^2)$
- $\frac{\partial L}{\partial y} = \frac{\partial L}{\partial s}\frac{\partial s}{\partial y} = g(1 - z^2)$

6

False. Backpropagation requires a scalar objective because gradients are defined with respect to a scalar loss; vector valued outputs must be reduced to a scalar loss (or handled via Jacobians, which is not standard backprop).

7 Forward pass quantities needed for backpropagation

Intermediate activations and pre activations are required because local gradients depend on the values observed during the forward pass and cannot be recomputed correctly from gradients alone.

(a) Computational graph

$x, w \ \rightarrow \ s = w^\top x \ \rightarrow \ u = s + b \ \rightarrow \ \hat{y} = \sigma(u) \ \rightarrow \ L(\hat{y}, y)$

(b) Intermediate variables

- Linear combination $s = w^\top x$
- Affine output $u = s + b$
- Predicted probability $\hat{y} = \sigma(u)$
- Scalar loss L

(c) Direction of gradient flow

Gradients propagate backward from the loss L to $\hat{y}$, then to u, and then split toward w, b, and x.

(d) Gradients w.r.t. w **and** b

By the chain rule, the gradient of the loss is first passed through the sigmoid and affine nodes to obtain $\partial L/\partial u$; this quantity is then multiplied by the local gradients $\partial u/\partial w = x$ and $\partial u/\partial b = 1$, yielding the gradients for w and b.