Exercises
Introduction to Machine Learning
FS 2018

Series 4, Apr 10th, 2018 (ANNs)

Institute for Machine Learning
Dept. of Computer Science, ETH Zürich
Prof. Dr. Andreas Krause
Web: `https://las.inf.ethz.ch/teaching/introml-s18`
Email questions to:
Mojmir Mutny, `mmutny@inf.ethz.ch`

Problem 1 (Recurrent Neural Networks):

In the lecture so far, we saw feedforward artificial neural networks, which do not contain any cycles and for which the nodes do not maintain a persistent state over several runs. This exercise considers artificial neural networks with nodes that maintain a persistent state that can be updated. This kind of neural network is called a recurrent neural networks (RNN). As an example, consider the following RNN with

$$\begin{aligned} y_t &= Wx_t + Vs_t \\ s_{t+1} &= y_t \end{aligned}$$

from some initial state s_0, where t denotes the tth call of the RNN, i.e., x_t is the tth input.

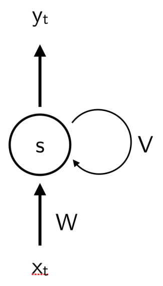


Figure 1

(a) What is the recurrent state in the RNN from Figure 1? Name one example that can be more naturally modeled with RNNs than with feedforward neural networks?

(b) As the state of an RNN changes over different runs of the RNN, the loss functions that we use for feedforward neural networks do not yield consistent results. For given dataset X, please propose a loss function (based on the mean square loss function) for RNNs and justify why you chose this loss function.

(c) For a dataset $X := (x_t, y_t)_1^k$ (for some $k \in \mathbb{N}$), show how information is propagated by drawing a feedforward neural network that corresponds to the RNN from Figure 1 for $k = 3$. Recall that a feedforward neural network does not contain nodes with a persistent state. (Hint: unfold the RNN.)

Solution 1:

(a) The reccurent state is denoted s. In this case it conincides with the output. Reccurent models are used to model data with temporal structure e.g. time series, speech, sound.

(b) We have a data $X = \{(x_t, y_t)\}$, where we assume that the data is ordered temporily. Thus, we define the loss function to be $L(U, W, s_0) = \sum_{t=1}^{T}(y(t) - f(x_t, s_{t-1}(U, W), U, W)$, where s_t is the previous reccurent state. The initial state s_0 needs to be specified and the problem depends on it as well.

(c) Check Figure 2.

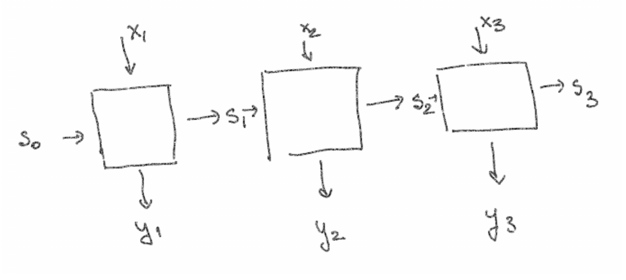


Figure 2

Problem 2 (Expressiveness of Neural Networks):

In this question we will consider neural networks with sigmoid activation functions of the form

$$\varphi(z) = \frac{1}{1+\exp(-z)}.$$

If we denote by v_j^l the value of neuron j at layer l its value is computed as

$$v_j^l = \varphi\left(w_0 + \sum_{i \in \text{Layer}_{l-1}} w_{j,i} v_i^{l-1}\right).$$

In the following questions you will have to design neural networks that compute functions of two Boolean inputs X_1 and X_2. Given that the outputs of the sigmoid units are real numbers $Y \in (0,1)$, we will treat the final output as Boolean by considering it as 1 if greater than 0.5 and 0 otherwise.

(a) Give 3 weights w_0, w_1, w_2 for a single unit with two inputs X_1 and X_2 that implements the logical OR function $Y = X_1 \vee X_2$.

(b) Can you implement the logical AND function $Y = X_1 \wedge X_2$ using a single unit? If so, give weights that achieve this. If not, explain the problem.

(c) It is impossible to implement the XOR function $Y = X_1 \oplus X_2$ using a single unit. However, you can do it using a multi-layer neural network. Use the smallest number of units you can to implement XOR function. Draw your network and show all the weights.

(d) Create a neural network with only one hidden layer (of any number of units) that implements

$$(A \vee \neg B) \oplus (\neg C \vee \neg D).$$

Draw your network and show all the weights.

Solution 2:

(a) We consider the following network $w = 0.5$, $w_1 = 1$ and $w_2 = 1$. We check whether the output we get is desired OR function. The network looks as follows,

$$A \vee B = \mathrm{round}(\varphi(w_0 + w_1 A + w_2 B))$$
$$A \vee B = \mathrm{round}(\varphi(-0.5 + A + B))$$

A	B	$A \vee B$	Network	Round
1	1	1	≈ 0.81	1
0	1	1	≈ 0.62	1
1	0	1	≈ 0.62	1
0	0	0	≈ 0.37	0

(b) $w_0 = -1.5$, $w_1 = 1$ and $w_2 = 1$. We check whether the output we get is desired AND function. The network looks as follows,

$$A \wedge B = \mathrm{round}(\varphi(w_0 + w_1 A + w_2 B))$$
$$A \wedge B = \mathrm{round}(\varphi(-1.5 + A + B))$$

A	B	$A \wedge B$	Network	Round
1	1	1	≈ 0.62	1
0	1	0	≈ 0.37	0
1	0	0	≈ 0.37	0
0	0	0	≈ 0.18	0

(c) We find the weights by choosing the weights of the first layer and optimizing over the weights of the last layer s.t. the inequalities are satisfied.

We use a network with one hidden layer and two states

$$A \oplus B = \mathrm{round}(\varphi(-\varphi(A+B) + 0.84\varphi(2A+2B))$$

A	B	$A \oplus B$	Network	Round
0	0	0	0.480010659844	0
0	1	1	0.502202727467	1
1	0	1	0.502202727467	1
1	1	0	0.486027265451	0

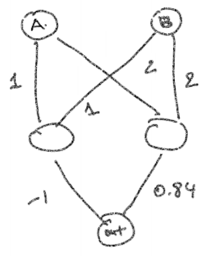


Figure 3

For sketch check Figure 3.

(d) There are multiple ways to proceed to solve this exercises. One universal way is to guess, however this might become too complicated for large expressions. Another approach is to use a numerical software to fit the network to match the logic. However, this might be too brute force approach for these general problems.

We provide here a third principled alternative to build such neural networks. I am not certain whether this method generalizes to any logical expressions (if you show it; good for you), however it works for this case.

Namely, we choose to transform the expression to *disjunctive normal form*, and then formulate a simplified classifier. Recall from your previous courses that any boolean algebra expression can be converted to a disjunctive normal form. In this form, the expression is cumulative OR of several expressions that contain just AND inside them. As AND, and OR operations can be modeled easily with our neural network, we can hope to perform the AND operation in one layer, and the OR operation in the second layer.

Specifically, after applying de Morgan laws, distributivity and symmetry to our expression many times, we arrive at the following minimal form,

$$(A \vee \neg B) \oplus (\neg C \vee \neg D) \iff (A \wedge C \wedge D) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg D) \vee (\neg B \wedge C \wedge D). \quad (1)$$

The detailed derivation works as follows.

$$\begin{aligned} &(A \vee \neg B) \oplus (\neg C \vee \neg D) \\ \iff &((A \vee \neg B) \wedge (C \wedge D)) \vee ((\neg A \wedge B) \wedge (\neg C \vee \neg D)) \end{aligned}$$

applying the distributivity law $((X \vee Y) \wedge Z \iff ((X \wedge Z) \vee (Y \wedge Z))$, we get

$$\iff ((A \wedge C \wedge D) \vee (\neg B \wedge C \wedge D)) \vee ((\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg D))$$

applying associativity and symmetry, we get

$$\iff (A \wedge C \wedge D) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg D) \vee (\neg B \wedge C \wedge D).$$

Our expression decomposes to 4 expressions that are combined via logical ORs. Thus, we propose to choose 4 cells each modeling the respective expression in OR stream (AND here), and in the last cell we take OR of all of them. Implementation of AND can be done simply using the intuition from previous exercises.

1. $(A \wedge C \wedge D)$: implements $h_1 = \varphi(\eta(A + C + D - 2.5))$
2. $(\neg A \wedge B \wedge \neg C)$: implements $h_2 = \varphi(\eta(-A + B - C - 0.4))$
3. $(\neg A \wedge B \wedge \neg D)$: implements $h_3 = \varphi(\eta(-A + B - D - 0.4))$
4. $(\neg B \wedge C \wedge D)$: implements $h_4 = \varphi(\eta(-B + C + D - 1.4))$

In order to get output 0 or 1, we take $\eta \rightarrow$large, e.g.$\eta = 200$.

Subsequently to define OR, we require that at least one of these is activated, thus final layer becomes easy,

$$H(h1, h2, h3, h4) = \varphi(h1 + h2 + h3 + h4 - 0.5).$$

In full,

$$\text{output}(A, B, C, D) \quad = \quad H(h1(A, C, D), h2(A, B, C), h3(A, B, D), h4(B, C, D)). \tag{2}$$

We check the truth table,

A	B	C	D	$(A \vee \neg B) \oplus (\neg C \vee \neg D)$	Round
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	1	1
0	1	0	0	1	1
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	0	0
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	1	1

For a figure see Figure 4.

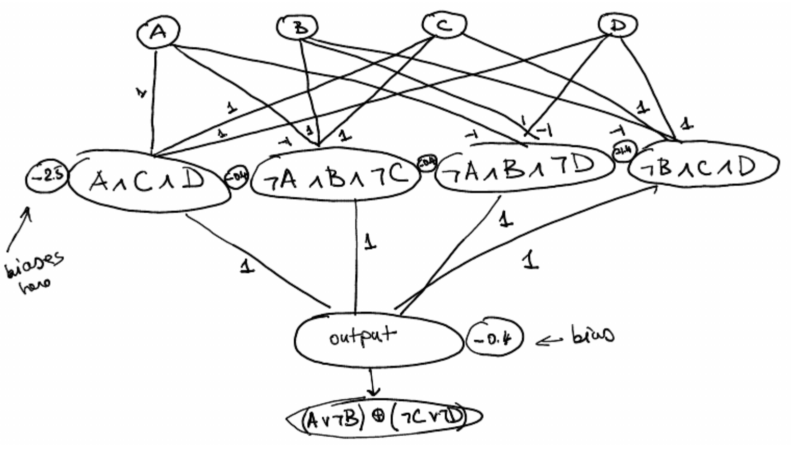
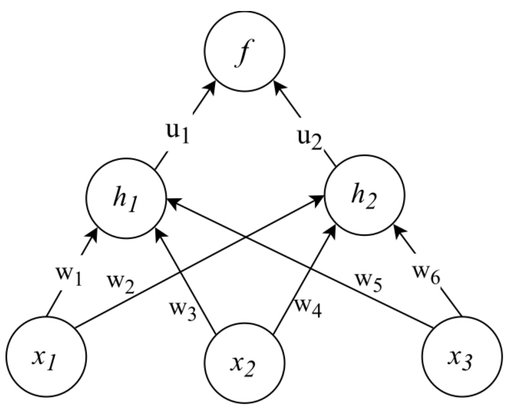


Figure 4

Problem 3 (Exam question: Artificial Neural Networks):

Consider the following neural network with two logistic hidden units h_1, h_2, and three inputs x_1, x_2, x_3. The output neuron f is a linear unit, and we are using the squared error cost function $E = (y - f)^2$. The logistic function is defined as $\rho(x) = 1/\left(1 + e^{-x}\right)$.

[*Note: You can solve part (c) without using the solution for part (b).*]



(a) Consider a single training example $\boldsymbol{x} = [x_1, x_2, x_3]$ with target output (label) y. Write down the sequence of calculations required to compute the squared error cost (called forward propagation).

(b) A way to reduce the number of parameters to avoid overfitting is to tie certain weights together, so that they share a parameter. Suppose we decide to tie the weights w_1 and w_4, so that $w_1 = w_4 = w_{\text{tied}}$. What is the derivative of the error E with respect to w_{tied}, i.e. $\nabla_{w_{\text{tied}}} E$?

(c) For a data set $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \cdots, (\boldsymbol{x}^{(n)}, y^{(n)})\}$ consisting of n labeled examples, write the pseudocode of the stochastic gradient descent algorithm with learning rate η_t for optimizing the weight w_{tied} (assume all the other parameters are fixed).

Solution 3:

Past Exam question,Detailed solution not provided

(a) basic algebra defined via the DAG

(b) apply chain rule

(c)
```
1. Pick a random guess $w^0_{tied}$, learning rate $\eta_k$
2. Repeat with $k = 1 \dots$ as iteration index
    (a) Sample a data point $j \sim \mathrm{Unif}[n]$
    (b) $w^{k+1}_{tied} = w^k_{tied} - \eta_k \nabla_{w_{tied}} E(w^k_{tied})$
3. Until happy with solution
```