Prof. Dr. Andreas Hotho, Albin Zehe, Konstantin Kobs
Lehrstuhl für Data Science

4./5.5.2021

1. Assignment in “Machine Learning for Natural Language Processing”

Summer Term 2021

1 General Questions

1. Identify three concrete cases where Deep Learning is used for language processing in real-world applications!

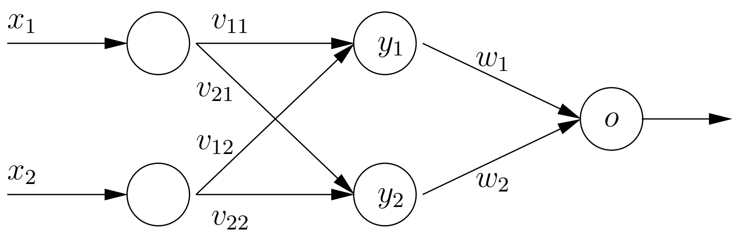
> a) Google Translate
> b) Predictive Keyboards (Smartphones)
> c) Voice Recognition

2. Name three common activation functions and their derivatives!

> - Sigmoid: $f(z) = \frac{1}{1+e^{-z}}$, $f'(z) = f(z) \cdot (1 - f(z))$
> - Hyperbolic Tangent: $f(z) = tanh(z)$, $f'(z) = 1 - tanh^2(z)$
> - Rectified Linear Unit (ReLU): $f(z) = max(0, z)$,
>
> $$f'(z) = \begin{cases} 0, z \leq 0 \\ 1, z > 0 \end{cases}$$

2 Neural Networks and Circuit Diagrams

Given the following neural network with `sigmoid` as activation function:

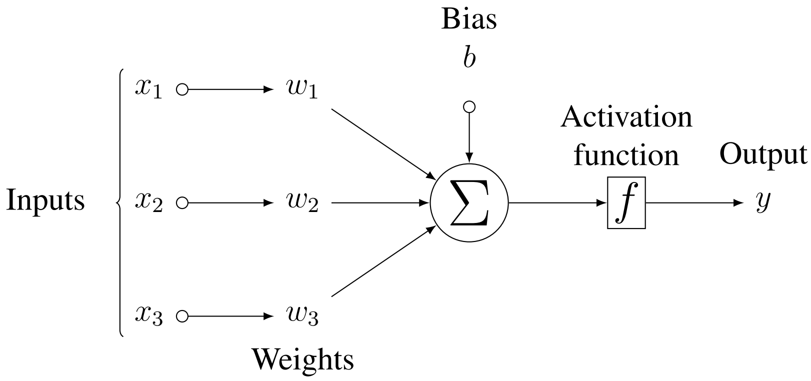


1. Represent the network as three equations (one each for y_1, y_2 and o)! We do not use any bias in this network.

$$
\begin{aligned}
y_1 &= \sigma(x_1 v_{11} + x_2 v_{12}) \\
y_2 &= \sigma(x_1 v_{21} + x_2 v_{22}) \\
o &= \sigma(y_1 w_1 + y_2 w_2)
\end{aligned}
$$

2. Convert the network to a circuit diagram as the one shown in the lecture!
Hint: Draw the diagram as large as possible. You will need space for the back-propagation.

Remember that a neuron's internal structure looks like this:



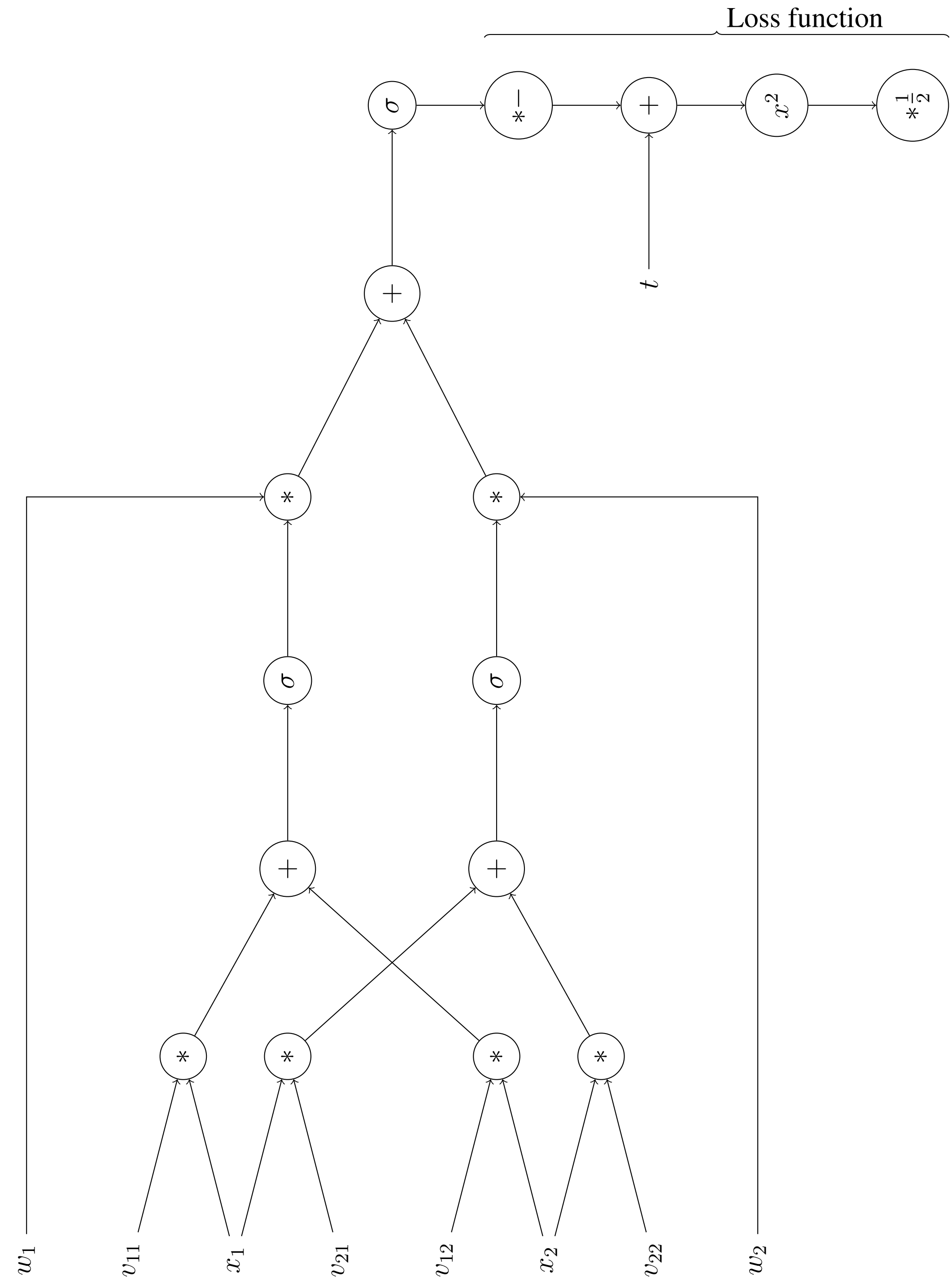
For the circuit diagram, see the next solution.

3. To measure the quality of our weights, we will use the following loss function:

$$L = \frac{1}{2} \cdot (t - o)^2,$$

where o is the output of the network and t is the target (correct) label.

Add the loss function to the circuit diagram from the last subtask!



4. Now that we have a circuit diagram, we can rather easily do backpropagation.

Determine the partial derivatives for all *weights* ($v_{11}, v_{12}, v_{21}, v_{22}, w_1$ and w_2) in the network!

Assume the following initial values: $v = \begin{pmatrix} 0.5 & 0.75 \\ 0.25 & 0.25 \end{pmatrix}$, $w = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$.

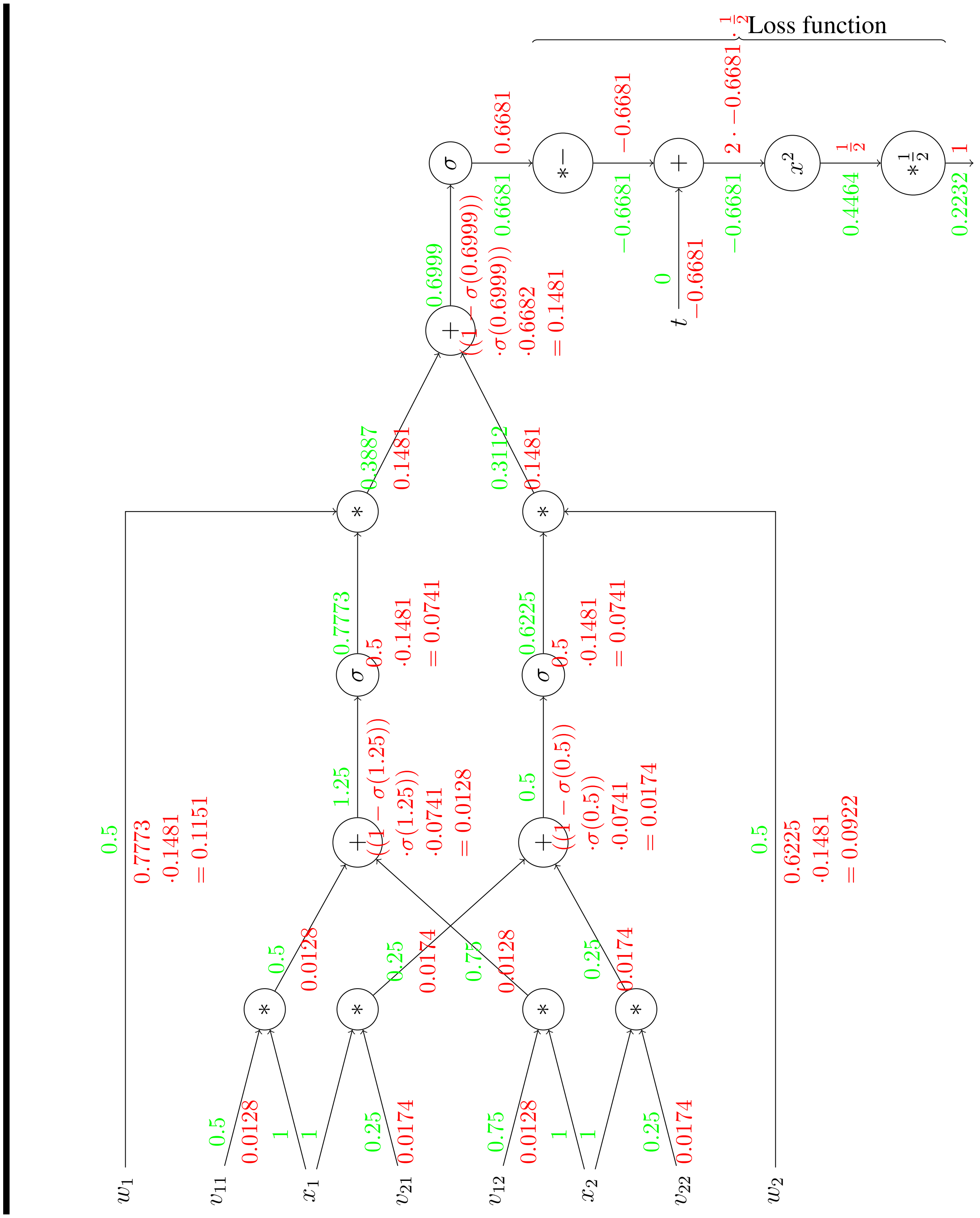
The inputs are $(x_1, x_2) = (1, 1)$ and the target label is $t = 0$.

Remember the intuitive properties of $+$ and $*$ from the slides:

- $+$ just passes the gradient through, both inputs have the same partial derivative as the output
- $*$ "switches" the input, that is: if the gradient after the $*$ is 2 and the inputs are 1 and 3, then first input gets the partial derivative $3 \cdot 2 = 6$ and the second input gets $1 \cdot 2 = 2$.

Here are some gradients that you will need:

- $f_a(x) = ax \to \frac{df}{dx} = a$
- $f(x) = x^2 \to \frac{df}{dx} = 2x$



Loss function
w_1
v_{11}
x_1
v_{21}
v_{12}
x_2
v_{22}
w_2
t

? Something to think about

5. Backpropagation would enable us to compute a gradient for the *input values* (x_1, x_2), too. What could we do with the information about the input's gradient?

> The gradient of inputs is used, for example, in the creation of *adversarial examples*. These inputs are specifically crafted to confuse a network, for example to achieve a misclassification.
>
> For more details, see `https://arxiv.org/abs/1412.6572`.

3 Python

In the lecture, some commonly used optimisers for neural networks were introduced. Implement functions for `sgd_update`, `nesterov_momentum_update` and `adam_update` in Python! Each function should

- take as input
  - the current value of *one* input parameter,
  - a function returning the gradient regarding this parameter, and
  - the necessary hyper-parameters such as the learning rate, and
- return the new value for this parameter after the update.

You do *not* have to implement backpropagation in this assignment!

```python
import numpy as np

def sgd_update(x, dx, learning_rate):
    x -= learning_rate * dx(x)
    return x

v = 0
def nesterov_momentum_update(x, dx, learning_rate, mu):
    global v

    x_ahead = x + mu * v
```

```python
    v = mu * v - learning_rate * dx(x_ahead)
    x += v
    return x

m = 0
v = 0
def adam_update(x, dx, learning_rate, beta1, beta2):
    global m, v

    m = beta1 * m + (1 - beta1) * dx(x)
    v = beta2 * v + (1 - beta2) * (dx(x) ** 2)

    x += -learning_rate * m / (np.sqrt(v) + 1e-8)

    return x
```