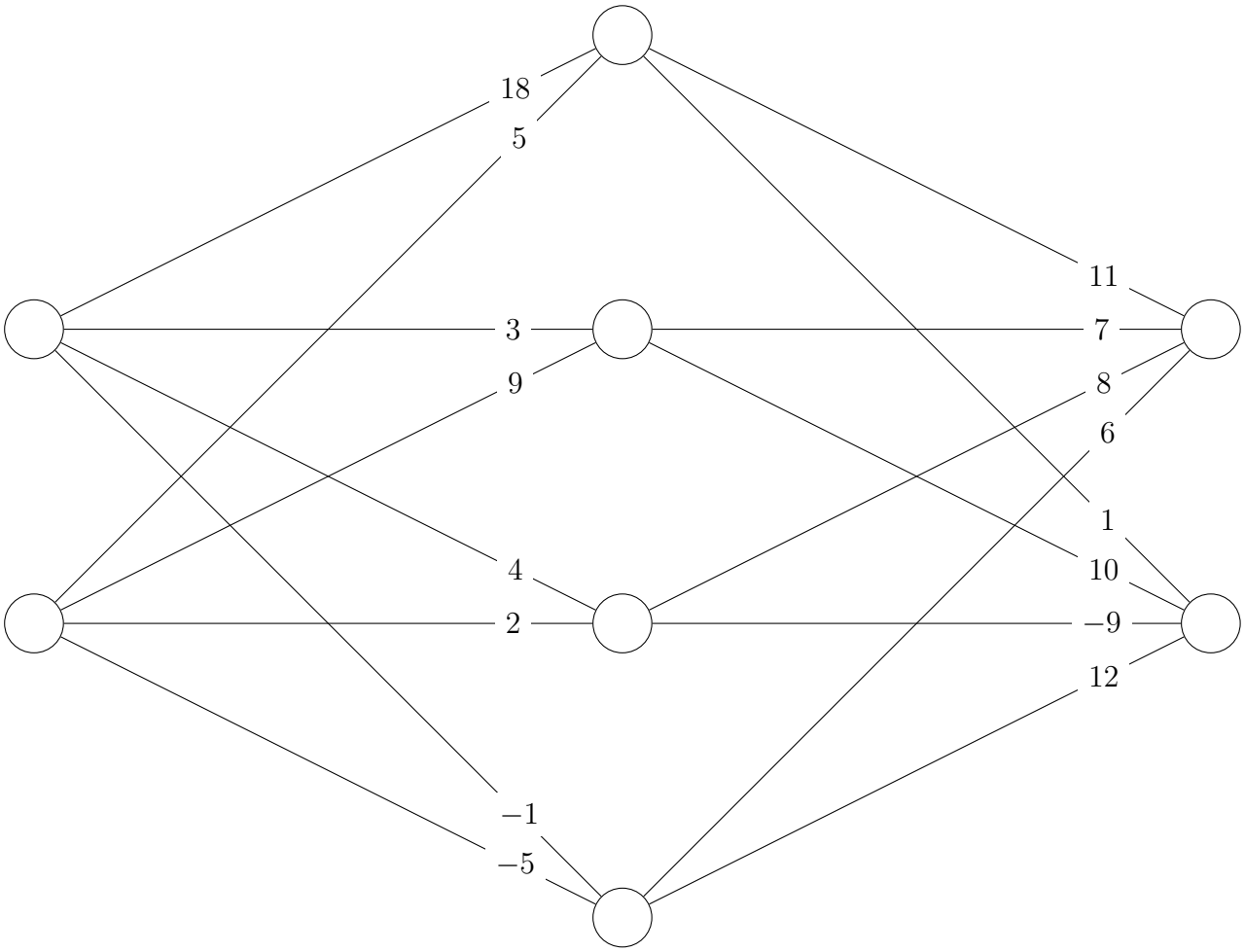
# Aufgabe 1: Pruning theory

Consider the following neural network with the weights as indicated on each connection:




We now want to prune the network via unstructured magnitude-based Pruning.

**1.1** Draw the resulting network after pruning 25 % of the network weights globally.

**1.2** Draw the resulting network after pruning 25 % of the network weights locally.

**1.3** Explain the difference between structured and unstructured pruning. What are the advantages and disadvantages of each of them?

**1.4** If you want to optimize your model for a smartphone, which pruning method will you use and why?

**1.5** In your own words, explain the basic concept behind optimal brain damage.

# Aufgabe 2: Implementing Unstructured Pruning

Now we want to implement local magnitude based unstructured pruning in python and pytorch step by step. For that, we first want to write the function to prune a single tensor. When we remove the weights with the lowest importance, the weight tensor W will become sparse.

The sparsity of W can be described by the following formula:

$$sparsity := \#Zeros/\#W = 1 - \#Nonzeros/\#W$$

Given a target sparsity s, W can then be multiplied with a binary mask M to disregard the pruned weights. M can be found by calculating:

$$v_thr = kthvalue(Importance, \#W * s)$$
$$M = Importance > v_thr$$
$$W_{sparse} = W * M,$$

with $kthvalue(X, k)$ being the function that finds the k-th smallest value of tensor X, $v_thr$ being the threshold under which to prune the weights and $Importance = |W|$.

**2.1** Complete the function `fine_grained_prune` in `exercise_3/pruning/pruning.py` to find and apply the pruning mask to a single tensor (*Tipp: Pytorch provides functions torch.kthvalue(), torch.gt() and torch.abs()*). You may use `test_fine_grained_prune()` which can be found in the same file to test your implementation.

**2.2** Now we want to use that function to prune a whole model. For this exercise we use an already a VGG model that's already been trained on FashionMNIST For model and training code consider `exercise_3/pruning/model.py`. The pretrained weights of the model are stored in `trained_dens_model.pth` and can be loaded with `load_dense_model()`. Alternatively, you can use the training script to train the model by yourself from scratch. Furthermore, a FinegrainePruner class that uses the previously implemented function is provided in `exercise_3/pruning/pruning.py` as well. It works as follows:

- It receives a sparsity dict which dictates the level of sparsity for each layer locally.
- For each convolutional and fully connected layer a pruning mask is applied and stored in a dictionary.
- When fine-tuning the mask is re-applied after computing each batch to assure that the zeroed weights stay zero.

Use the FinegrainedPruner to prune each layer of the model with a uniform sparsity of 0.6. What is the model size and test accuracy before and after pruning? Now fine-tune for 5 epochs on the training set. How does the model recover?

**2.3** We learned in the lecture that uniform pruning does not always yield optimal results due to some layers being more sensitive and others less sensitive to pruning. Implement the introduced heuristical method (see Figure **??**) and plot the sparsity rate curve for

each layer separately from 0.1 to 0.9. What do you observe? Are all layers similar in sensitivity?

**2.4** While we want to prune sensitive layers less, we also want to prune larger layers more aggressively to get a smaller model. For this the amount of parameters of each layer should also be considered. Because of this, plot the number of parameters each layer has. Can you observe relations to the sensitivity of the layers?

**2.5** Use the information from the previous two tasks to decide the sparsity of each layer non-uniformly. How sparse can you make the model while staying above 90% test accuracy?

