# CS 577 — Fall 2025 — Homework 2

**Problem 1 — [5] point(s)**. Draw the computational graph for the following function. Then compute `wr.grad`, `wi.grad`, and `wo.grad` using backpropagation.

```
x1 = ag.Scalar(2.0, label="z1\nleaf(x1)")
h0 = ag.Scalar(3.0, label="z2\nleaf(h0)")
wr = ag.Scalar(4.0, label="z3\nleaf(wr)")
wi = ag.Scalar(5.0, label="z4\nleaf(wi)")
wo = ag.Scalar(6.0, label="z5\nleaf(wo)")

z1 = x1
z2 = h0
z3 = wr
z4 = wi
z5 = z3*z2 # wr*h0
z6 = z4*z1 # wi*x1
z7 = z5+z6
z8 = ag.relu(z7) # relu(wr*h0 + wi*x1)
z9 = wo
z10 = z8*z9
z10.backward()

print(wr.grad, wi.grad, wo.grad)
```

You are allowed run the above using '`ex2.ipynb`' from Lecture 3 on canvas. However, you must explicitly explain step-by-step what happens during each iteration of the back-propagation, e.g., during the 0-th iteration, what is the node being visited in the computation graph. For which nodes are the `grad` field updated. Repeat this for the 1-st, 2nd and so on iterations. A print-out of the computer-based calculation is not an acceptable answer.

---

**Problem 2 — [5] point(s)**. Implement `def max(a,b)` for `ag.Scalar`

```
def max(a, b):
    # input: a and b are instances of ag.Scalar
    # output: should be the maximum of a and b
    # [...]
    output._backward = _backward
    return output
```

by filling in the function in `prob2.ipynb`. Do this for `def min(a,b)` as well. For the backward function, if there are ties between `a.value` and `b.value` you can break ties arbitrarily. There is a "Grad check" at the end of the jupyter notebook. If your implementation is correct, the Grad check code block should run silently.

Hint: if $f(a,b) = \max(a,b)$ what is $\frac{\partial f}{\partial a}(a,b)$ when $a \neq b$?

---

**Problem 3 — [10] point(s)**. Go to `prob3.ipynb` provided by filling in missing code block at "`YOUR ANSWER HERE`".