Computational Graph for Multi-layer Perceptron (MLP)

**Definitions:**
- **Input:** \( x \in \mathbb{R}^n; \ y \in \mathbb{R}^n \)
- **Our MLP:** \( f(x; W) \) weight matrix parameterizing our MLP
- **Loss function:** \( L(f(x, W), y) \)
- **\( \sigma() \): activation function, e.g., tanh, sigmoid

*For simplicity, we consider \( A(x, W) = \sigma(x^T W) \) as a primitive.

How to calculate \( \frac{\partial L}{\partial W^{(i)}} \)?

\[
V_i = A(\cdot, W^{(i)}) = \sigma(x_i^T W^{(i)}) = \sigma(x_i^T W_{i,:} + x_i)
\]

\[
\frac{\partial L}{\partial W^{(i)}} = \frac{\partial L}{\partial V_i} \frac{\partial V_i}{\partial W^{(i)}} = \frac{\partial L}{\partial V_i} \sigma'(x_i^T W^{(i)}) x_i
\]

*Here we use \( x = \langle V_{\text{input}} \rangle \), the set of nodes coming into \( V_i \).

For MLP, we optimize over \( W \), all weights in our MLP. We can then use \( \frac{\partial L}{\partial W^{(i)}} \) in e.g., gradient descent updates:

\[
W^{(i)} \leftarrow W^{(i)} - \alpha \frac{\partial L}{\partial W^{(i)}}
\]

\( \alpha \) learning rate