Neural Networks - Tips

1. Data
   - Discrete values → separate inputs
     - Prefer bipolar (-1 | 1) to binary (0 | 1) input
     - Forces learning (zero values mean edges do not change)
   - Continuous values → 1 input
   - Normalize (scale) data

Continuous:	Discrete:
$\frac{x_i - \mu}{\sigma^2}$	$\frac{x_i - x_{min}}{x_{max} - x_{min}}$

2. Activation Functions
   - Match with data type and range
   - Sum of Products (SOP)

$$\sigma = \sum_i w_i x_i + \text{bias}$$

   - Threshold

$$y = \begin{cases} 1, & \sigma > T \\ 0, & \text{otherwise} \end{cases}$$

   - Binary Sigmoid

$$y = \frac{1}{1 + e^{-\sigma}}$$

   - Bipolar Sigmoid

$$y = \frac{2}{1 + e^{-\sigma}} - 1$$

   - Hyperbolic Tangent

$$y = \frac{e^{\sigma} - e^{-\sigma}}{e^{\sigma} + e^{-\sigma}}$$

3. Output Activations
   - Classification: Threshold / Sigmoid
   - Regression: Linear (SOP)
   - Multiple classes: Softmax

$$y_i = \frac{e^{\sigma_i}}{\sum_k e^{\sigma_k}}$$

4. Weight Initialization
   - Avoid saturation points (i.e. activations or derivatives → 0)

     $\frac{-1}{\sqrt{n}} < w < \frac{1}{\sqrt{n}}$ where n = #nodes in input layer

5. Error (Cost) Functions
   - Sum of Squared Error (SSE)

     $$E = \sum_k (y_k - t_k)^2$$

   - Probabilistic (use Cross Entropy)

     $$E = \sum_k y_k \ln(y_k)$$

6. Training
   - Training set size

     #samples > 10 * #weights

   - Training set structure
     - Split into Training (60%), Validation (20%), Testing (20%)
     - Use cross-validation if data is limited
   - Learning Rate (η)
     - Fixed: use small value (< 0.1)
     - Adaptive: start large, decrease over time or if Error increases
   - Momentum: add small proportion of update from ($n-1^{st}$) iteration

     $$w_i = w_i + \eta E_i x_i + \alpha \Delta w_i (n-1)$$

   - Mini-batches: perform gradient descent on subsets of training data
   - Termination
     - Fixed: # Epochs, set amount of Time
     - Adaptive: stop training (#Epochs) when Validation error increases

7. Network Architecture
   - #hidden layers: one is often sufficient, two can approximate *any* function
   - #nodes in hidden layer: 2/3 (#inputs + #outputs)
     - Adaptive: constructive/destructive approaches, based on performance
   - #edges: use iterative weight decay to remove non-essential edges