**1. Is it okay to initialize all the weights to the same value, even if using He initialization?**

**No, it is not okay.** Even if you use He initialization (which is usually random), setting all weights to the **same value** leads to **symmetry problems**. When all neurons in a layer have the same weights, they learn the same features and behave identically, preventing the network from effectively learning diverse patterns.

**2. Is it okay to initialize the bias terms to 0?**

**Yes, it is generally okay.**

* Bias terms are not affected by the symmetry problem, so initializing them to 0 does not harm the learning process.
* However, in some cases (e.g., ReLU neurons), using **small positive values** (like 0.01) can help prevent "dead neurons."

**3. Three advantages of ELU over ReLU**

1. **Handles negative values better:** Unlike ReLU, which outputs zero for negative inputs (causing dead neurons), ELU smoothly continues for negative values.
2. **Faster convergence:** The exponential component of ELU leads to **faster weight updates** in deep networks.
3. **Zero-centered activations:** Unlike ReLU, ELU outputs values centered around zero, reducing vanishing/exploding gradient problems.

**4. When to use different activation functions?**

* **ELU (Exponential Linear Unit):**
  + Best for **deep networks** where vanishing gradients are a concern.
  + Works well when speed and performance are priorities.
* **Leaky ReLU (and variants like Parametric ReLU):**
  + Helps prevent "dead neurons" by allowing small negative values.
  + Useful when ReLU networks suffer from neurons being stuck at zero.
* **ReLU (Rectified Linear Unit):**
  + **Default choice** for hidden layers in deep learning models.
  + Simple and computationally efficient.
* **Tanh (Hyperbolic Tangent):**
  + Good for **shallow networks** as it outputs values between -1 and 1.
  + Zero-centered, unlike sigmoid.
* **Logistic (Sigmoid) function:**
  + Used **only for binary classification** tasks.
  + Can cause vanishing gradients in deep networks.
* **Softmax:**
  + Used in the **output layer** for **multiclass classification**.
  + Converts logits into probabilities that sum to 1.

**5. What happens if the momentum hyperparameter is too close to 1 (e.g., 0.99999) in a MomentumOptimizer?**

* The optimizer may **overshoot the optimal point**, leading to unstable training.
* It will **retain too much past velocity**, making it **slow to adapt to new gradients**.
* Training can **fail to converge** properly or oscillate.

**6. Three ways to produce a sparse model**

1. **L1 Regularization (Lasso):**
   * Adds an L1 penalty to the loss function, forcing some weights to become exactly zero.
2. **Pruning:**
   * Removes low-weight connections after training, reducing the number of parameters.
3. **Weight Quantization:**
   * Reduces the precision of weights, leading to a smaller, sparse model.

**7. Does dropout slow down training? Does it slow down inference?**

* **Training:** Yes, dropout **slows down training** because it randomly disables neurons, requiring more updates for convergence.
* **Inference:** No, dropout does not slow down inference. In fact, inference is **faster** because dropout is turned off, and the full network is used with scaled weights.