### ****Q1. What is a loss function? How is it different from a cost function?****

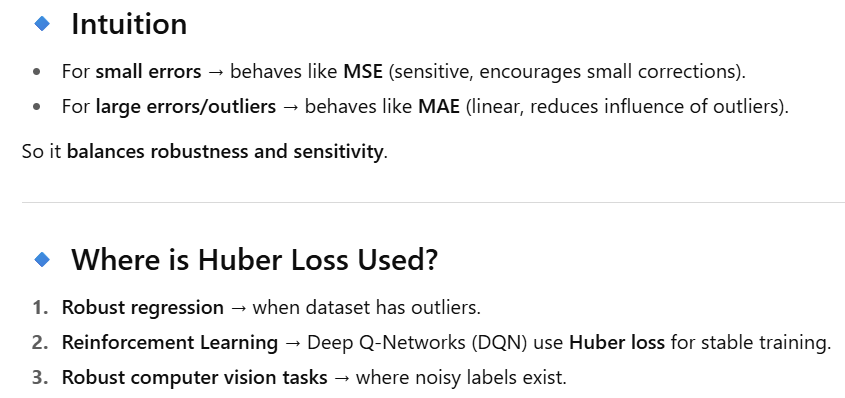
👉 **Loss function**: Error for a single training example.  
👉 **Cost function**: Average of loss over the dataset.  
(Though in practice, people use them interchangeably.)

**Q2. Explain the difference between MSE, MAE, and Huber loss.**

* **MSE (Mean Squared Error)**: Penalizes large errors heavily (quadratic). Smooth gradients, but sensitive to outliers.
* **MAE (Mean Absolute Error)**: Linear penalty → more robust to outliers. But gradient is constant, which slows learning.
* **Huber Loss**: Quadratic near 0, linear for large errors → combines benefits of both.

**Huber Loss** is a regression loss that is **quadratic for small errors** (like MSE) and **linear for large errors** (like MAE).

It’s less sensitive to outliers than MSE, but smoother than MAE (since it’s differentiable at 0).



### ****Q3. Why is cross-entropy loss preferred over MSE in classification?****

👉 MSE gives slow convergence because it doesn’t align well with softmax outputs (gradients get very small).  
👉 Cross-entropy directly measures the distance between probability distributions (predicted vs. true) → faster and more stable training.

**Q4. What is categorical cross-entropy vs. binary cross-entropy?**

* **Binary Cross-Entropy (BCE)**: For 2-class problems (output ∈ {0,1}).

L = −[ylog (p)+(1−y)log (1−p)]]

* **Categorical Cross-Entropy (CCE)**: For multi-class (softmax output).

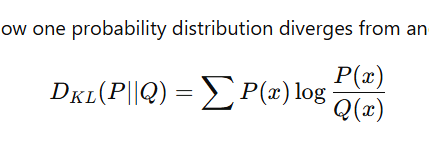
L = −∑iyilog (pi)

* **Sparse Categorical Cross-Entropy:**
  + **Explanation:** This is functionally identical to Categorical Cross-Entropy. The only difference is in how the true labels are provided.
  + **Key Difference:** Instead of a one-hot encoded vector [0, 0, 1, 0], you provide the label as an integer 2. This is more memory efficient when you have a large number of classes.
  + **Use Case:** Same as Categorical Cross-Entropy, but preferred for convenience and efficiency with large datasets and many classes.

If labels are one-hot → categorical; if labels are integers → sparse categorical.

**Q5. What is KL divergence loss? When is it used?**

👉 KL divergence measures how one probability distribution diverges from another:



Used in:

* Variational Autoencoders (VAEs).
* Knowledge Distillation (teacher vs. student network).

### ****Q6. What is focal loss and why is it useful?****

👉 Focal loss modifies cross-entropy by adding a factor that downweights well-classified examples → focuses on hard/misclassified examples.  
Used in object detection (e.g., RetinaNet) for **class imbalance**.

**Q7. How do you choose a loss function for regression vs. classification vs. ranking?**

* **Regression** → MSE, MAE, Huber.
* **Classification** → Cross-entropy, Focal loss.
* **Ranking** → Hinge loss, Contrastive loss, Triplet loss.

**Q9. What are custom loss functions? Can you give an example?**

👉 Custom losses encode business-specific goals. Example:

* In marketing: cost-sensitive loss → weight conversions higher than impressions.
* In imbalanced datasets: weighted BCE.  
  Example (weighted BCE):

L=−[w1 ylog(p) + w0 (1−y) log(1−p)]

**Q11. Why can’t accuracy be used as a loss function?**

👉 Accuracy is non-differentiable (step function). Optimizers like SGD need gradients, so accuracy can’t guide weight updates.

**Q12. What happens if you use the wrong loss function?**

👉 Model may converge to a suboptimal solution. Examples:

* Using MSE for classification → slow convergence.
* Using cross-entropy for regression → nonsensical outputs.

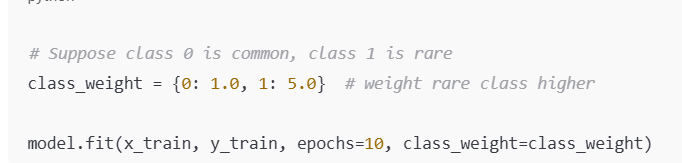
**Q13. How do ranking losses (contrastive / triplet) work?**

* **Contrastive loss**: Encourages similar pairs to have small distance, dissimilar pairs to be apart.
* **Triplet loss**: Minimizes distance(anchor, positive) – distance(anchor, negative) + margin.  
  👉 Used in face recognition, recommendation systems.

**Q14. How do you handle imbalanced datasets with loss functions?**

* Weighted cross-entropy.
* Focal loss.
* Oversampling + loss adjustments.  
  👉 Example: In medical imaging, false negatives are more costly → increase their weight in loss.

**Weighted Cross-Entropy** assigns **higher weight to rare classes** (or more costly classes), so the loss penalizes misclassifications more.



*“Weighted cross-entropy is a modified loss function that assigns higher weights to rare or costly classes in classification problems, making the model more sensitive to misclassifying them. It’s mainly used in imbalanced datasets like fraud detection or medical diagnosis.”*

pos\_weight = number\_of\_negative\_samples / number\_of\_positive\_samples.

**Q15. What’s the intuition behind label smoothing in loss functions?**

👉 Instead of hard labels (0 or 1), assign soft targets (e.g., 0.9 for true class, 0.1 spread across others).  
✅ Prevents overconfidence, improves generalization (used in Transformers).

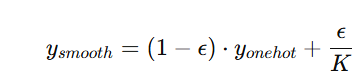
**🔹 What is Label Smoothing?**

👉 **Label smoothing** is a regularization technique where instead of using hard one-hot labels (e.g., [0,0,1,0]), we **soften the labels** by distributing a small amount of probability mass to all other classes.

Example (4-class problem, true class = 3):

* **Without smoothing (one-hot):** [0, 0, 1, 0]
* **With smoothing (ε=0.1):** [0.033, 0.033, 0.9, 0.033]

Formula:



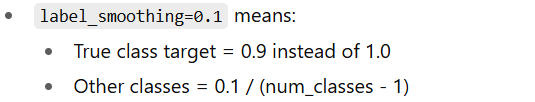
where **K = number of classes**, and **ε = smoothing factor** (typically 0.05–0.2).

**🔹 Where is Label Smoothing Used?**

1. **Classification tasks** with softmax output (image classification, NLP).
2. **Transformers (NLP/CV)** → used in **attention-based models** like BERT, GPT training.
3. **Image classification benchmarks** (ResNet, EfficientNet, Vision Transformer).
4. **Distillation** → when training a student from a teacher network.

**🔹 Why Use Label Smoothing?**

1. **Prevents overconfidence:**
   * With one-hot labels, the model can become too confident (p=0.999 for one class).
   * Overconfidence hurts generalization → poor performance on unseen data.
   * Label smoothing encourages the model to keep probabilities softer.
2. **Improves generalization:**
   * Reduces variance, helps avoid overfitting.
   * Especially useful in small datasets or imbalanced datasets.
3. **Stabilizes training:**
   * Avoids **infinite loss** when the model predicts 0 probability for the correct class.
   * Helps prevent gradient spikes.
4. **Better calibrated probabilities:**
   * Outputs are more interpretable (probabilities closer to true likelihood).



**🔹 Why this helps**

* Prevents overconfidence.
* Regularizes training.
* Improves calibration of predicted probabilities.

Optimisers

An optimizer is an algorithm used to update the parameters (weights and biases) of a neural network in order to minimize the loss function. It does this by calculating the gradients of the loss with respect to each parameter using backpropagation and then deciding how to change the parameters based on those gradients. The goal is to efficiently find the model's parameters that result in the lowest possible loss, guiding the model towards the best possible performance.

**1. What’s the difference between SGD and Adam optimizers?**

* **SGD**: Updates weights using gradients with a fixed learning rate (can add momentum).
* **Adam**: Combines **momentum** (first moment estimate) + **adaptive learning rate** (second moment estimate).
* **Key Point**: Adam converges faster, but SGD with momentum often generalizes better in large-scale vision tasks.

**2. Why do we need momentum in optimizers?**

* Without momentum, SGD may oscillate in directions of steep slopes (zig-zagging).
* Momentum adds a fraction of the previous update to the current update → accelerates in consistent directions, dampens oscillations.
* Analogy: Rolling a ball downhill with friction → smoother path.

**3. What is learning rate decay and why is it important?**

* A **high learning rate** → fast learning but risk of overshooting minima.
* A **low learning rate** → stable but slow convergence.
* Decay schedules (Step decay, Exponential decay, Cosine annealing, OneCycle) gradually reduce LR to fine-tune convergence.

**4. Why does Adam sometimes fail to generalize compared to SGD?**

* Adam adapts too aggressively to noisy gradients.
* It can find sharp minima instead of flat minima (poor generalization).
* **Fixes**: AdamW (decoupled weight decay), switching to SGD at later stages.

**5. What is Nesterov Accelerated Gradient (NAG) and how is it different from Momentum?**

* **Momentum**: Look at current gradient + past update.
* **NAG**: First move in momentum’s direction, then compute gradient at this *lookahead position*.
* Gives more accurate correction → better convergence.

**6. Explain the difference between L1/L2 regularization vs Weight Decay in optimizers like AdamW.**

* In **SGD**, L2 regularization = weight decay (same math).
* In **Adam**, L2 penalty doesn’t behave like true weight decay because adaptive LR scaling interferes.
* **AdamW**: Fixes this by decoupling weight decay from gradient updates → better regularization.

**7. What is the role of epsilon (ε) in Adam optimizer?**

* Small constant added to denominator when normalizing gradients.
* Prevents division by zero.
* Also stabilizes updates when second moment (variance) is very small.

**8. How do you choose the right optimizer for a problem?**

* **Small/simple dataset** → SGD with momentum.
* **Large NLP tasks** → Adam / AdamW.
* **Reinforcement Learning** → RMSProp (handles noisy gradients well).
* **Transfer learning** → Adam/AdamW (fast adaptation).

**9. What is gradient clipping and why is it used with optimizers?**

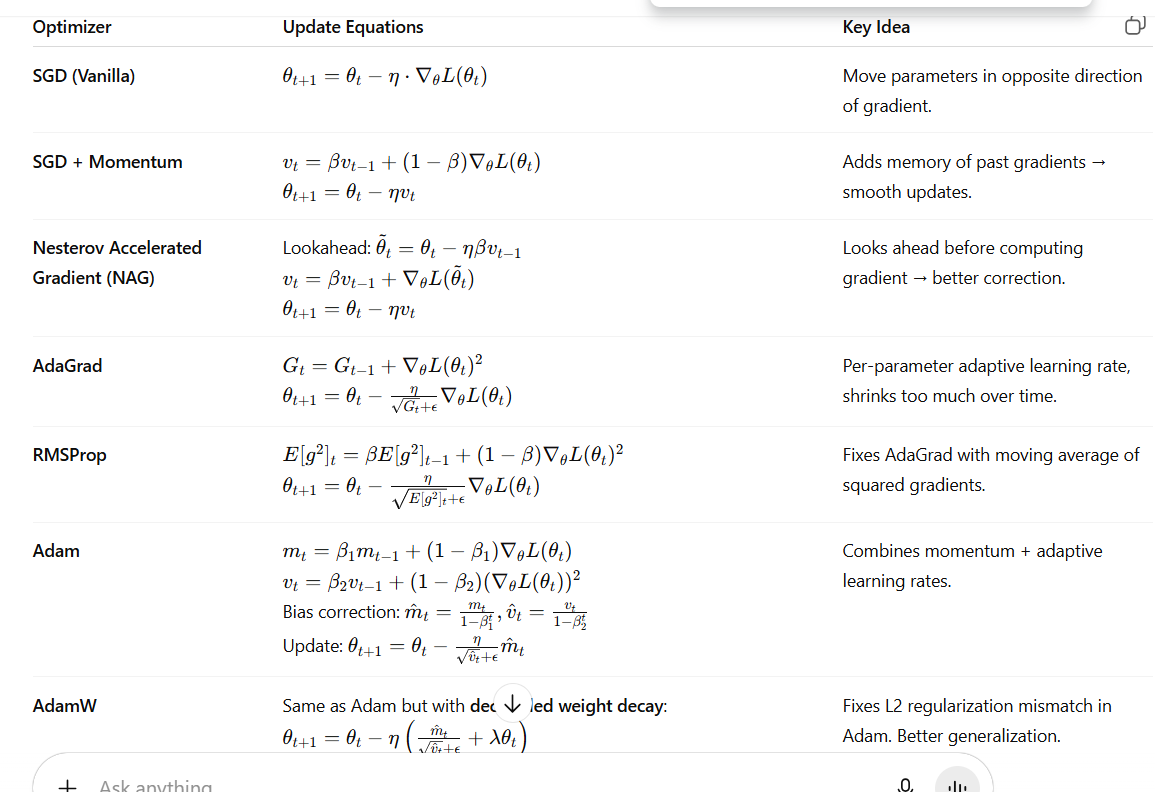
* Clip gradients (by norm or value) when they explode to prevent instability.
* Very important in **RNNs / Transformers** with long sequences.
* Works with any optimizer.

**10. Explain the concept of adaptive optimizers (AdaGrad, RMSProp, Adam).**

* **AdaGrad**: Per-parameter LR shrinks over time → good for sparse data, bad for deep nets (too aggressive decay).
* **RMSProp**: Fixes AdaGrad by using exponential moving average of squared gradients.
* **Adam**: Adds momentum on top of RMSProp → balances speed & stability.

✅ **Summary for interview prep:**

* Know **SGD vs Adam** trade-offs.
* Understand **momentum, NAG, weight decay, epsilon, learning rate schedules**.
* Be able to explain why **AdamW is preferred** in modern DL.
* Be ready to derive/update equations if interviewer asks.



**Q: "What is the main idea behind Adam (Adaptive Moment Estimation)?"**

**Answer:**  
"Adam is one of the most popular optimizers because it combines the concepts of **Momentum** and **Adaptive Learning Rates** (like RMSprop).

It calculates two things:

1. **First Moment (mt):** An exponentially decaying average of past gradients (like momentum).
2. **Second Moment (vt):** An exponentially decaying average of past *squared* gradients (like RMSprop, which estimates the variance/unsteadiness).

It then uses bias corrections for these moments and takes a step that is influenced by both the 'velocity' (momentum) and the 'stability' (adaptive learning rate) of each parameter.

* **Benefits:**
  + Works well on problems with noisy or sparse gradients.
  + Requires little tuning of the learning rate (the default lr=0.001 often works great).
  + It's generally a good default choice for a wide range of problems.
* **Drawbacks:** It can sometimes converge to a worse final solution compared to well-tuned SGD with Momentum, especially on generative tasks like GANs or language modeling."

**Q: "What is a learning rate scheduler? Name common types."**

**Answer:**  
"A learning rate scheduler is a method to adjust the learning rate during training according to a pre-defined rule. It's used because a large learning rate is good for rapid progress early on, but a smaller learning rate is needed later to fine-tune the parameters and converge properly.

Common types include:

* **Step Decay:** Reduce the learning rate by a factor (e.g., 0.1) every N epochs.
* **Exponential Decay:** Reduce the learning rate exponentially.
* **Cosine Annealing:** Decreases the learning rate following a cosine curve, and can optionally restart periodically. This is very powerful and can help escape local minima.
* **ReduceLROnPlateau:** Dynamically reduces the learning rate when a monitored metric (e.g., validation loss) has stopped improving."

**Q: "What is the 'epsilon' parameter in Adam?"**

**Answer:**  
"Epsilon is a very small number (e.g., 1e-8) added to the denominator in the Adam update rule. It's a **numerical stability** term to prevent division by zero. You almost never need to change this default value."

**Q: "What is gradient clipping and why is it used?"**

**Answer:**  
"Gradient clipping is a technique that caps the magnitude of gradients during backpropagation before the update step. It's primarily used to combat the **exploding gradients** problem, which is common in Recurrent Neural Networks (RNNs) and Transformers.

There are two common types:

1. **Clip by value:** If a gradient is above a threshold, set it to the threshold.
2. **Clip by norm:** Scale the entire gradient vector if its L2 norm exceeds a threshold.

This prevents overly large parameter updates that can cause the training process to become unstable and fail."

**Weight initialization:**

| **Activation Function** | **Recommended Gain** | **In PyTorch: calculate\_gain("...")** | **Notes** |
| --- | --- | --- | --- |
| **Linear / Identity** | **1.0** | **"linear"** | **Used for final layers before logits/softmax** |
| **Sigmoid** | **1.0** | **"sigmoid"** | **No scaling needed** |
| **Tanh** | **5/3 ≈ 1.67** | **"tanh"** | **Helps prevent saturation** |
| **ReLU, GELU** | **√2 ≈ 1.41** | **"relu"** | **Standard He scaling** |
| **Leaky ReLU (slope a)** | **√(2 / (1 + a²))** | **"leaky\_relu", param=a** | **e.g. a=0.01 → ≈1.41, a=0.2 → ≈1.34** |
| **ELU** | **√1.55 ≈ 1.25** | **"selu"** | **Equivalent to SELU in PyTorch** |
| **SELU** | **1.0** | **"selu"** | **Self-normalizing nets** |
| **Softmax (logits)** | **1.0** | **"linear"** | **Use for final classifier layers** |