Technical Interview Preparation

ML Fundamentals

Bias v.s. Variance

Bias

- error from overly simplistic assumptions → underfitting

Variance

- error from excessive sensitivity to the training data → overfitting

Tradeoff

- bias ↓ (reducing) → variance ↑ (increasing), vice versa
- balance (sweetspot) → generalization error ↓ (minimize)

Overfitting & Regularization

Overfitting

- learning a noise (pattern) specific to the training data > (rather than) generalizable features

Techniques for mitigating the overfitting

- **Regularization**
  - discourage large weights → limit the model's capacity
  - learned function to be smoother → toward simpler solution (variance ↓)
  - added to the loss function → less sensitive to the training data
  - small amount of bias → variance ↓
  - **L2** (Ridge regression)
    - $\mathcal{L}(w) = \|y - Xw\|^2 + \lambda\|w\|^2$
    - add penalty $\lambda\|w\|^2$ → proportional to the squared magnitude of the weights
    - soft penalty Squared → smaller weights
  - **L1** (Lasso)
    - add penalty $\lambda\|w\|_1$ → penalizing the absolute values of the weights
    - hard penalty → encourages sparsity in weights feature selection
- **Dropout**
  - randomly disabling perceptrons → preventing the nework relying too heavily on specific activations
  - more robust representations
- **Early stopping**
  - monitoring validation performance → stopping training before overfitting
  - preventing the model from fitting noise → complexity ↓
- **Cross Valiaton**

Normalization

- making input features or activations having similar scales → stabilizing & accelerating optimization
  - preventing features with large magnitudes from dominating gradient updates
- normalizing + scaling & shifting (w/ learnable parameters)

- reasons
  - gradient magnitude $\propto$ input feature scale
    - → update biased toward certain dimensions
  - ill-conditioned Hessian
    - eigenvalues spread widely → unstable optimization
- effects
  - Convergence ↑ & sensitivity ↓ to the learning rate

Feature Normalization

- normalizing input features (e.g., zero mean, unit variance)
  - statistics (mean, variance) computed per feature across the training dataset
  - each feature (dimension) → more equally contributing to optimization
- no learnable parameters
- effective usage
  - commonly used as a preprocessing step

Batch Normalization

- normalizing activations (=outputs of the nodes) using mini-batch statistics (mean, variance) per channel across batch
- applied during training
  - during inference: use moving average statistics collected during training
- internal covariant shift (↓)
  - distribution of activations changes as parameters of previous layers update
  - making optimization stable & faster convergence
- learnable parameters for scaling & shifting
  - γ (scale), β (shift) for each feature dimension(channel)
- how to be trained

    - mean and variance computed per mini-batch
    - parameters updated via backpropagation
    - running parameters updated via exponential moving average (EMA)
- effective usage
    - CNNs, large-batch training

Layer Normalization

- normalizing activations across feature dimensions per sample
- applied during training and infernce (no train-test discrepancy)
- learnable parameters for scaling & shifting
    - γ (scale), β (shift) for each feature dimension
- effective usage
    - widely used in transformers and sequence models
        - stable for variable-length inputs
        - suitable for small-batch settings

Instance Normalization

- normalizing activations per sample and per channel
    - statistics (mean, variance) computed
        - across spatial dimensions (H, W)
        - for each channel independently
        - for each sample independently
- applied during training and inference
    - no train-test discrepancy
    - no dependence on batch statistics
- learnable parameters for scaling & shifting
    - γ (scale), β (shift) for each channel

- interpretation
  - removes instance-specific contrast and illumination statistics
  - preserves content structure while normalizing style-related variations
- effective usage
  - image generation and style transfer
  - tasks where instance-level appearance variation is undesirable
  - small-batch or batch-size-1 settings

Adaptive Instance Normalization (AdaIN)

- an extension of Instance Normalization
  - used for conditional normalization
- core idea
  - align the channel-wise statistics of a content feature
    - to those of a conditioning style feature
- formulation
  - given content feature x and style feature y:
    - $\mathrm{AdaIN}(x, y) = \sigma(y) \cdot \frac{x - \mu(x)}{\sigma(x)} + \mu(y)$
- interpretation
  - normalize content features using instance normalization
  - then inject style information via scale and shift
    - replacing (γ, β) with style-dependent statistics
- comparison to Instance Normalization
  - Instance Norm
    - $x \mapsto \gamma \cdot \frac{x - \mu(x)}{\sigma(x)} + \beta$
    - γ, β are learnable parameters
  - AdaIN

- entropy ↓ → more confident or peaked distribution → ≈deterministic distribution

Cross-Entropy

$H(p, q) = -\sum_i p_i \log q_i$

- measures the expected surprisal when
  - true distribution is p
  - but outcomes are encoded using model distribution q
  - how "surprised" we are when using q to explain data from p
- equivalent to negative log-likelihood
  - used as the standard loss for classification
  - penalizes assigning low probability to true labels

KL Divergence

$\mathrm{KL}(p||q) = \sum_i p_i \log \frac{p_i}{q_i} = H(p, q) - H(p)$

- measure how much information is lost when q is used to approximate p
  - extra surprise due to mismatch between p and q
- minimizing cross-entropy ⇔ minimizing KL divergence
  - since $H(p)$ is fixed
    - entropy $H(p)$: intrinsic uncertainty of the data

Dropout

- randomly deactivate neurons during training
  - prevent the network from relying too heavily on specific activations → overfitting ↓