ChatGPT

Deep Neural Networks MCQs

Session 1: Neural Networks, Single-Layer Networks, Activation Functions, Backpropagation

1. **What happens if a neural network uses only linear activation functions in all layers?**
   A. It becomes equivalent to multiple independent networks.
   B. It becomes equivalent to a single linear-layer network. **(Correct)**.
   C. It can model complex non-linear functions.
   D. It fails to learn any function.
   Answer: B. If all activations are linear, every layer reduces to a single linear transformation, so the multi-layer network is mathematically equivalent to a single linear layer. Thus no extra modeling power is gained by depth.
   Explanation: In a feedforward network, non-linear activation functions are crucial. Without them, stacking layers is pointless: the composition of linear functions is still linear. In fact, if no non-linearity is used, a multi-layer network collapses to a single-layer linear model, meaning it can only represent linear transformations of the inputs.

2. **Which of the following is true about the sigmoid activation function?**
   A. Its output ranges from -1 to 1.
   B. Its output ranges from 0 to 1, and it is differentiable. **(Correct)**.
   C. It can output negative values.
   D. It is not continuous.
   Answer: B. The sigmoid function maps inputs to a value between 0 and 1 and is smooth and differentiable.
   Explanation: Sigmoid (σ) outputs a probability-like value in (0,1). It is continuous and differentiable, which is why it was historically used for binary outputs. Specifically, $\sigma(z) \in (0,1)$ for all real z, and its derivative is $\sigma(z)(1-\sigma(z))$.

3. **What is the output range of the hyperbolic tangent (tanh) activation function?**
   A. [0, 1]
   B. (–∞, +∞)
   C. (0, +∞)
   D. [–1, 1] **(Correct)**.
   Answer: D. The tanh function maps input values to the range –1 to +1.
   Explanation: Tanh is a scaled and shifted sigmoid: it outputs values from –1 to +1. This symmetry about zero can be preferable over the sigmoid's [0,1] range. Mathematically, tanh(z) = (e^z – e^(–z))/(e^z + e^(–z)), yielding values in [–1,1].

4. **Which of the following defines the Rectified Linear Unit (ReLU) activation function?**
   A. f(x) = max(0, x) **(Correct)**.
   B. f(x) = 1/(1 + e^(–x)).

C. f(x) = tanh(x).
D. f(x) = e^x.
Answer: A. ReLU is given by f(x) = max(x, 0), meaning it outputs the input if positive, otherwise zero.
Explanation: ReLU is a simple piecewise linear function: it outputs x for x>0 and 0 for x≤0. In practice, this means negative inputs yield zero activation, while positive inputs pass through unchanged. This is explicitly f(x)=max(x,0), making it efficient and avoiding saturation for positive values.

5. **What is the primary purpose of the backpropagation algorithm in neural network training?**
A. To perform forward propagation of inputs through the network.
B. To initialize network weights before training.
C. To compute gradients of the loss with respect to each weight. **(Correct)**.
D. To add dropout to the network.
Answer: C. Backpropagation computes the gradient of the cost (loss) function for each weight by propagating the error backwards through the network.
Explanation: In training, backpropagation (error-backpropagation) uses the chain rule to find how each weight affects the loss. In effect, it "backpropagates" the error from the output layer through hidden layers, computing $\partial \text{loss}/\partial \text{weight}$. This gradient is then used by an optimizer (like gradient descent) to update weights. In summary, backprop is specifically designed to calculate these loss gradients.

Session 2: Introduction and Comparison of TensorFlow and PyTorch

1. **Which of the following describes PyTorch's computation graph?**
A. Static computation graph defined before runtime.
B. No computation graph is used.
C. Dynamic computation graph (Autograd) that is defined during execution. **(Correct)**.
D. Fixed for all machine learning frameworks.
Answer: C. PyTorch uses a *dynamic* computation graph (built using Autograd) that is created on-the-fly during the forward pass.
Explanation: PyTorch builds the computation graph dynamically for each input. This means operations are executed immediately and the graph is constructed as code runs, making it very Pythonic. This dynamic "define-by-run" approach (Autograd) contrasts with static graphs, and is noted for making experimentation easier and debugging more intuitive.

2. **Which framework typically requires defining the entire model graph before running it (i.e. uses a static computation graph)?**
A. PyTorch.
B. None; all deep learning libraries use dynamic graphs.
C. Both TensorFlow and PyTorch.
D. TensorFlow (before eager mode). **(Correct)** [1].
Answer: D. TensorFlow (traditional mode) uses a *static* computation graph that requires defining the full model ahead of execution [1].
Explanation: TensorFlow's default mode (graph mode) involves declaring the model graph (nodes and operations) up front. The graph is then executed by running sessions. This static graph approach (though TensorFlow 2.x also supports eager mode) allows certain optimizations. As a result, TensorFlow historically needed the full model specified before running, contrasting with PyTorch's dynamic approach [1].

3. **Which of the following is a built-in visualization tool provided with TensorFlow for tracking training metrics?**
   A. PyTorchViz
   B. TensorBoard **(Correct)**.
   C. VisualDL
   D. Matplotlib
   Answer: B. TensorBoard is TensorFlow's built-in tool for visualizing training metrics, computational graphs, and more.
   Explanation: TensorBoard is a utility that comes with TensorFlow to log and visualize training progress, network graphs, and statistics (such as loss curves, images, histograms, etc.). It is specifically associated with TensorFlow, whereas PyTorch does not have an official equivalent (though one can use TensorBoard with PyTorch too). The mention of "TensorBoard: A tool for visualization" highlights it as a TensorFlow feature.

4. **What is Keras in relation to TensorFlow?**
   A. A type of activation function.
   B. A static graph optimizer.
   C. A high-level neural network API integrated into TensorFlow. **(Correct)**.
   D. A data augmentation library.
   Answer: C. Keras is a high-level neural network API that is integrated with TensorFlow to simplify model building and training.
   Explanation: TensorFlow includes Keras as its high-level interface, providing convenient ways to define and train models (sequential or functional). It abstracts much of the low-level graph-building. As noted, Keras (now tightly integrated in TF 2.x) serves as a user-friendly front-end for constructing TensorFlow models.

5. **For which scenario is PyTorch generally considered more suitable than TensorFlow?**
   A. Production deployment on large-scale servers.
   B. Simple linear regression problems.
   C. Static graph optimization tasks.
   D. Rapid prototyping and research experimentation. **(Correct)** [2].
   Answer: D. PyTorch is often favored for research and rapid prototyping because of its flexibility and ease of use [2].
   Explanation: PyTorch's dynamic graph and Pythonic style make it very convenient for experimenting and developing new models. According to surveys, PyTorch's flexibility makes it ideal for research use cases. By contrast, TensorFlow's static graph was historically considered stronger for deployment, but PyTorch shines when one needs to iterate quickly. The statement "the flexibility of PyTorch makes it ideal for research and prototyping" captures this [2].

Sessions 3–4: Deep Learning Basics, Logistic Regression, Cost Function, Gradient Descent

1. **Which activation function is typically used at the output of a binary logistic regression model?**
   A. ReLU
   B. Tanh
   C. Softmax

D. Sigmoid **(Correct)**.

Answer: D. Binary logistic regression uses the sigmoid (logistic) function at the output to map values into a (0,1) probability range.

Explanation: Logistic regression models p(Y=1) via σ(z)=1/(1+e^(-z)). It squashes a linear combination (U·X + b) into a probability. This is exactly what the "logistic function" or sigmoid provides. In the excerpt, logistic function g(z)=1/(1+e^(-z)) is described, confirming the use of sigmoid.

2. **What loss function is commonly minimized in training logistic regression for binary classification?**

   A. Mean Squared Error (MSE)

   B. Binary Cross-Entropy (Log Loss) **(Correct)**.

   C. Hinge Loss

   D. Cosine Similarity Loss

   Answer: B. Logistic regression typically uses the binary cross-entropy loss (also called log loss) to measure the error between predicted probabilities and true labels.

   Explanation: Binary cross-entropy (negative log-likelihood) is the standard cost for logistic regression. The reference shows the formula –y·log(ŷ) – (1–y)·log(1–ŷ), which is exactly the log loss for binary classification. This convex loss penalizes incorrect probability assignments and is minimized via gradient methods.

3. **Which optimization algorithm is most commonly used to train deep neural networks?**

   A. Decision Trees

   B. Linear Programming

   C. Gradient Descent (and its variants) **(Correct)**.

   D. K-Means Clustering

   Answer: C. Deep networks are typically trained using gradient-based methods, such as (stochastic) gradient descent, which iteratively updates weights in proportion to the negative gradient of the loss.

   Explanation: Weight update rules like (mini-batch) gradient descent are the norm for neural network training. The content states weights are "optimized via an algorithm such as gradient descent". Combined with backpropagation, this means we compute $\partial \text{loss} / \partial \text{weights}$ and step opposite that gradient (see backprop ref).

4. **What property does the logistic regression cost function have that makes optimization straightforward?**

   A. It is convex with a single global minimum. **(Correct)**.

   B. It is periodic, leading to multiple minima.

   C. It is discontinuous at zero.

   D. It is always zero.

   Answer: A. The cost (loss) function for binary logistic regression is a convex function, guaranteeing one global minimum to which gradient methods converge.

   Explanation: For logistic regression, the negative log-likelihood yields a convex cost surface. The reference notes that convexity means only one minimum. This ensures algorithms like gradient descent will reliably find that optimum. In contrast, deep nets (see next question) have non-convex loss landscapes.

5. **How does the loss function of a deep neural network typically compare to that of logistic regression?**
   A. Both are convex with one minimum.
   B. Deep networks have more convex losses.
   C. Deep networks are linear.
   D. Deep networks generally have a non-convex loss with many local minima. **(Correct)**
   Answer: D. Unlike logistic regression, deep neural networks often have highly non-convex loss surfaces with many local minima and saddle points.
   Explanation: Deep models introduce non-linearities and many layers, making the cost function landscape non-convex. The source explicitly contrasts logistic's convex loss with "non-convex" losses for neural nets. This means optimization is trickier (local minima) and requires good initialization or advanced techniques.

Session 5: Sigmoid Model, Loss Function, Learning Algorithms, Evaluation

1. **What is precision in a classification context?**
   A. TP / (TP + FP) **(Correct)**.
   B. TP / (TP + FN)
   C. (TP + TN) / (TP + FP + FN + TN)
   D. (FP + TN) / (TP + FP + FN + TN)
   Answer: A. Precision is defined as the ratio of true positives to all positive predictions (TP/(TP+FP)).
   Explanation: Precision measures the accuracy of positive predictions. It answers: "Of all instances the model predicted as positive, how many were correct?" This is given by TP/(TP+FP), as shown in the reference.

2. **What is recall (true positive rate) in classification?**
   A. TP / (TP + FP)
   B. (FP + TN) / Total
   C. (TP + TN) / Total
   D. TP / (TP + FN) **(Correct)**.
   Answer: D. Recall (or sensitivity) is the ratio of true positives to all actual positives (TP/(TP+FN)).
   Explanation: Recall asks: "Of all actual positive instances, how many did the model correctly identify?" This is TP divided by (TP + FN). The source explicitly gives recall = TP/(TP+FN), confirming answer D.

3. **Which metric combines precision and recall into one number, especially useful for imbalanced classes?**
   A. Accuracy
   B. ROC AUC
   C. Precision-Recall (PR) Score
   D. F1 Score **(Correct)**.
   Answer: D. The F1 score is the harmonic mean of precision and recall, and it is especially preferred over plain accuracy when classes are imbalanced.
   Explanation: F1 = 2·(precision·recall)/(precision+recall). Because it balances both false positives and false negatives, it's more informative than accuracy when one class is rare. The reference explicitly recommends F1 for imbalanced settings.

4. **In an imbalanced dataset scenario, which metric is generally recommended over accuracy?**
   A. F1 Score **(Correct)**.
   B. Accuracy
   C. Sum of precision and recall
   D. Mean Squared Error
   Answer: A. The F1 score is preferable on imbalanced data because accuracy can be misleading when one class dominates.
   Explanation: When classes are imbalanced, a model may appear to perform well by always predicting the majority class (high accuracy but poor minority detection). F1 incorporates both precision and recall, ensuring the model must do well on the minority class to score highly. The source advises that F1 is "preferable to accuracy for imbalanced datasets", highlighting its use in such cases.

5. **Which metric can be misleading on imbalanced datasets, and thus should be used cautiously alone?**
   A. Precision
   B. Recall
   C. Accuracy **(Correct)**.
   D. F1 Score
   Answer: C. Accuracy can give a false sense of performance when classes are imbalanced, so it should not be used by itself in such cases.
   Explanation: If, say, 90% of data belong to one class, a model that always predicts the majority class has 90% accuracy but is useless for minority class detection. The reference advises to avoid relying solely on accuracy in imbalanced scenarios. Metrics like F1 or precision/recall are more informative.

Session 6: Shallow Networks, Hidden Layers, Activation, Forward/ Backward Propagation

1. **What advantage does adding a hidden layer to a neural network provide?**
   A. It always makes training faster.
   B. It allows the network to model non-linear interactions between input features. **(Correct)** [3].
   C. It guarantees the network will not overfit.
   D. It reduces the number of parameters.
   Answer: B. Introducing a hidden layer allows the network to capture non-linear relationships (feature interactions) that a simple linear model (like logistic regression) cannot [3].
   Explanation: A single-layer (linear) model can only represent linear combinations of inputs. Adding a hidden layer with a non-linear activation enables the network to learn more complex, non-linear mappings. The reference explicitly states that a hidden layer "allows the network to use non-linear interactions between features" [3].

2. **According to the Universal Approximation Theorem, a neural network with a single hidden layer can approximate what kind of functions (given sufficient neurons)?**
   A. Only linear functions.
   B. Only periodic functions.
   C. Any continuous function on a compact domain. **(Correct)** [4].
   D. No functions beyond the capabilities of logistic regression.
   Answer: C. The theorem states that a feedforward network with at least one hidden layer and a non-

linear activation can approximate any continuous function (to arbitrary accuracy) on a finite domain [4].

Explanation: With enough hidden units (and appropriate activation), a "shallow" network (1 hidden layer) is theoretically a universal function approximator. The cited text says such networks are dense in the space of continuous functions [4], meaning they can approximate any continuous mapping given sufficient capacity.

3. **What is forward propagation in a neural network?**
   A. The process of updating weights using gradients.
   B. The process of passing inputs through each layer to compute outputs. **(Correct)**.
   C. The process of normalizing input data.
   D. The process of evaluating model on test set only.
   Answer: B. Forward propagation is computing the activations layer by layer, from input to output.
   Explanation: In training or inference, the input data is fed forward through the network: each layer applies its weights and activation to produce the next layer's inputs, ultimately producing an output. (This is the step before backpropagation and does not require an external citation as it's basic terminology.)

4. **During backpropagation in a neural network, what is computed?**
   A. The forward pass outputs.
   B. The gradients of the loss with respect to each weight. **(Correct)**.
   C. The validation accuracy.
   D. The number of hidden units needed.
   Answer: B. Backpropagation computes the gradient of the loss function with respect to each model parameter (weight).
   Explanation: When training a network, after a forward pass computes the loss, backpropagating the error uses the chain rule to find $\partial \text{loss}/\partial w$ for every weight w. These gradients are then used to update the weights (usually via gradient descent). This process of computing gradients of the loss is exactly what backprop does.

5. **What effect does the ReLU activation function have on the gradients during training?**
   A. It ensures gradients are always positive.
   B. It can help mitigate the vanishing gradient problem for positive inputs, since derivative is 1 there. **(Correct)**.
   C. It causes gradients to vanish faster.
   D. It always leads to exploding gradients.
   Answer: B. For inputs >0, ReLU has a derivative of 1, which means gradients do not shrink in that regime (unlike sigmoid/tanh), helping mitigate vanishing gradients.
   Explanation: ReLU outputs max(x,0). Its gradient is 1 for x>0 and 0 for x<0 (undefined exactly at 0). Therefore, when x>0, it passes gradients through unchanged, avoiding the near-zero derivatives of sigmoidal activations. This property is one reason ReLU is often preferred in hidden layers.

Sessions 7–8: Parameters vs Hyperparameters, Regularization, Norms

1. **Which of the following is a hyperparameter rather than a model parameter?**
   A. Weight matrix in an artificial neural network.
   B. The learning rate used in training. **(Correct)** [5] [6].
   C. Bias terms in a linear model.
   D. Logistic regression coefficients learned from data.
   Answer: B. Hyperparameters (like the learning rate) are external to the model and set before training, whereas weights and biases are model parameters learned from data [5] [6].
   Explanation: Model parameters (e.g. weights, biases in neural nets) are adjusted by training on data. Hyperparameters control the learning process itself and are not learned. The citation notes that parameters (e.g. network weights) are estimated from data, while hyperparameters (e.g. learning rate, network architecture choices) are specified beforehand and not learned during training [5] [6].

2. **Which regularization method tends to produce *sparse* weight vectors (many weights become exactly zero)?**
   A. L2 regularization (Ridge).
   B. Dropout.
   C. L1 regularization (Lasso). **(Correct)** [7].
   D. Early stopping.
   Answer: C. L1 regularization (adding the L1 norm of weights to the loss) encourages sparsity, often driving many weights to exactly zero [7].
   Explanation: L1 penalty is the sum of absolute weight values. It has the effect of pushing many weights to zero (feature selection). The source states that L1 produces sparse solutions (weights can be zero), unlike L2 which merely shrinks weights without zeroing most [7].

3. **Which regularization method adds a penalty proportional to the square of the weights, typically resulting in non-sparse solutions?**
   A. L1 regularization.
   B. L2 regularization (Ridge). **(Correct)** [7].
   C. Data augmentation.
   D. Batch normalization.
   Answer: B. L2 regularization (which adds the sum of squared weights) penalizes large weights but typically does not set them to zero, so solutions are generally non-sparse [7].
   Explanation: L2 penalty (weight decay) adds $\frac{1}{2}\lambda\sum w^2$ to the loss. It keeps weights small, but because the penalty is smooth, weights generally remain non-zero (just shrunken). According to the reference, L2 produces "non-sparse" solutions, meaning weights are reduced but rarely exactly zero [7].

4. **In L1 regularization, what term is added to the loss function?**
   A. Sum of absolute values of weights ($\|w\|_1$). **(Correct)** [7].
   B. Sum of squares of weights ($\|w\|_2^2$).
   C. Dropout of random weights.
   D. A term proportional to learning rate.
   Answer: A. L1 regularization adds the L1-norm of the weights (sum of absolute values) to the loss

[7].

Explanation: The L1 penalty is defined as $\lambda \Sigma |w_i|$. This linear penalty encourages sparsity. It contrasts with L2, which uses Σw_i^2. The reference specifically describes L1 as penalizing the sum of absolute weights [7].

5. **In L2 regularization, what term is added to the loss function?**
   A. Sum of absolute values of weights.
   B. Sum of squares of weights ($\|w\|_2^2$). **(Correct)** [7].
   C. L1 norm plus dropout.
   D. Negative log-likelihood.
   Answer: B. L2 regularization adds the squared L2-norm of the weights (sum of squares) to the loss function [7].
   Explanation: L2 regularization uses a term $\lambda \Sigma w_i^2$. This penalizes large weights more heavily but, unlike L1, tends not to make them exactly zero. The source confirms L2 penalizes squared magnitudes (leading to smooth shrinkage) [7].

Session 9: Dropouts, Early Stopping, Data Augmentation

1. **What is *dropout* in the context of neural networks?**
   A. A technique to randomly deactivate a fraction of neurons during training to prevent overfitting. **(Correct)** [8].
   B. Gradually decreasing the learning rate.
   C. Early stopping criterion.
   D. A type of pooling layer.
   Answer: A. Dropout randomly "drops" (sets to zero) a subset of neurons (or their outputs) during training, which acts as regularization and helps prevent overfitting [8].
   Explanation: By randomly ignoring (zeroing) different neurons each training batch, dropout prevents any single neuron from becoming overly specialized. This randomness forces the network to learn redundant, robust features. The source describes dropout as randomly ignoring some layer outputs during training as a regularization measure [8].

2. **What is one beneficial effect of using dropout during training?**
   A. It effectively creates an ensemble of many thinned networks, improving generalization. **(Correct)** [9].
   B. It increases the learning rate automatically.
   C. It eliminates the need for a validation set.
   D. It guarantees zero training error.
   Answer: A. Dropout can be seen as training many different "thinned" networks and averaging them, which helps generalization and reduces overfitting [9].
   Explanation: Each dropout mask yields a slightly different network. During inference (without dropout), it's analogous to averaging predictions of these many networks. The cited text notes that dropout "prevents overfitting by disabling neurons" and creates an ensemble effect for robustness [9].

3. **What is *early stopping* in neural network training?**
   A. Stopping training when training loss reaches zero.
   B. Using a small network architecture.

C. Halting training when validation loss stops improving to avoid overfitting. **(Correct)** [10] .
D. Reducing the learning rate to zero at the end of training.
Answer: C. Early stopping means stopping the training process once performance on a held-out validation set ceases to improve, thus preventing overfitting [10] .
Explanation: After each epoch, one checks validation error. If it starts to increase (while training error still decreases), it indicates overfitting. Early stopping then halts training at that point. The reference specifically says early stopping "halts training when validation performance deteriorates," which matches this strategy [10] .

4. **What is data augmentation in deep learning?**
A. The process of artificially increasing the training set by applying transformations (e.g. rotations, flips, noise) to existing data. **(Correct)** [11] .
B. Increasing the model size with more neurons.
C. Merging multiple datasets into one.
D. Normalizing features to zero mean and unit variance.
Answer: A. Data augmentation creates new training examples by applying label-preserving modifications to existing data (like rotations, translations, noise), which enriches the dataset and improves generalization [11] .
Explanation: Augmentation synthetically expands the dataset. For example, an image can be randomly flipped, rotated, or cropped to produce new training samples. This diversity forces the model to be invariant to these transformations. The source defines augmentation as modifying data to increase diversity, leading to better generalization [11] .

5. **Why is data augmentation helpful for preventing overfitting?**
A. It increases training set variety, making the model encounter more diverse examples. **(Correct)** [12] .
B. It directly reduces model complexity.
C. It increases the number of model parameters.
D. It replaces dropout.
Answer: A. By enlarging the effective training set with realistic variations of the data, data augmentation forces the model to learn more robust features and thus reduces overfitting [12] .
Explanation: Augmentation ensures the model sees many forms of the same underlying classes (e.g. different orientations of an object), so it learns invariances rather than memorizing the limited original images. The text notes that augmentation "helps models perform better by making most of existing data" and "prevents overfitting" [12] .

Session 10: Vanishing/Exploding Gradients, Gradient Checking

1. **What is the *vanishing gradient* problem in deep networks?**
A. Gradients become extremely small as they are propagated backward, making learning in early layers very slow. **(Correct)** [13] .
B. Gradients become extremely large, causing unstable training.
C. Gradients flip sign unpredictably.
D. Gradients cause the loss to increase always.
Answer: A. The vanishing gradient problem is when gradients shrink exponentially as they propagate backwards through many layers (or time steps), leading to very small updates in early layers [13] .

Explanation: In long chains of nonlinearities (deep nets or RNN through time), backpropagated gradients can become tiny. The cited explanation is that derivatives multiply and can approach zero, halting effective learning in lower layers [13]. This is a classic issue with sigmoidal activations and deep architectures.

2. **What is the *exploding gradient* problem?**
   A. Gradients are identically zero everywhere.
   B. Gradients grow exponentially large during backpropagation, leading to unstable (diverging) weight updates [14]. **(Correct)**.
   C. Gradients are always positive.
   D. Loss function explodes to infinity.
   Answer: B. Exploding gradients occur when backpropagated gradients accumulate and become very large, which can cause numerical overflow and wildly large weight updates [14].
   Explanation: In very deep networks, gradients can also amplify. The reference describes exploding gradients as gradients becoming excessively large as they pass through layers [14]. This destabilizes training (weights blow up). Common fixes include gradient clipping.

3. **Which activation functions are most prone to causing vanishing gradients?**
   A. ReLU (Rectified Linear Unit)
   B. Sigmoid and Tanh **(Correct)** [15].
   C. Linear functions
   D. None, activations don't affect gradients.
   Answer: B. Sigmoid and tanh activations can saturate (derivative near zero for large $|z|$), causing gradients to vanish [15].
   Explanation: When the input to sigmoid/tanh is very positive or negative, their slopes approach zero. Backpropagated gradients get multiplied by these small derivatives, effectively shrinking them to zero. The text specifically mentions that sigmoid and tanh suffer from vanishing gradients due to their <1 derivatives [15].

4. **How can exploding gradients be mitigated during training?**
   A. Using a very large learning rate.
   B. Discarding small gradients manually.
   C. Gradient clipping to cap the gradient magnitude. **(Correct)** [16].
   D. Removing hidden layers.
   Answer: C. A common solution is gradient clipping, where the gradients are scaled if their norm exceeds a threshold, preventing them from growing too large [16].
   Explanation: When gradients become too large, clipping rescales them to a manageable norm. The cited text suggests that clipping avoids excessively small or large gradients, effectively mitigating exploding gradients [16]. Another approach is using architectures like LSTM which also help prevent this issue.

5. **What is *gradient checking* in the context of neural networks?**
   A. A way to compute gradients analytically by hand.
   B. A technique to numerically approximate gradients (via finite differences) and compare to backprop-computed gradients to verify correctness [17]. **(Correct)**.
   C. Inspecting gradients after each epoch to adjust hyperparameters.
   D. A regularization method.

Answer: B. Gradient checking uses finite differences (small perturbations of weights) to numerically estimate gradients and compares them to the gradients computed by backpropagation [17].
Explanation: Before trusting a complex implementation of backprop, one can "check" it by comparing to a numerical gradient: e.g. f(w+ε) – f(w–ε) divided by 2ε. If backprop is correct, the two gradients should match. The reference explicitly says gradient checking perturbs each parameter to estimate gradients and compares to backprop values [17].

Session 11: Batch Normalization and Other Data Normalization Techniques

1. **What problem does batch normalization address in deep networks?**
   A. Internal covariate shift (change in distribution of layer inputs). **(Correct)** [18].
   B. Too many parameters.
   C. Insufficient data.
   D. Only overfitting, not related to distribution.
   Answer: A. Batch normalization helps mitigate internal covariate shift by normalizing layer inputs (zero mean, unit variance) within each mini-batch [18].
   Explanation: Internal covariate shift refers to changing input distributions to layers during training. BatchNorm fixes each mini-batch's activations to a standard distribution before scaling/shifting them. As noted, it normalizes each batch so inputs stay in a stable range, reducing the shift [18].

2. **What is a key effect of applying batch normalization in a network?**
   A. It allows higher learning rates and faster, more stable training. **(Correct)** [19] [20].
   B. It makes the network linear.
   C. It removes the need for activation functions.
   D. It always worsens performance.
   Answer: A. By keeping inputs to layers normalized, batch norm accelerates training and makes convergence easier (even allowing larger learning rates) [19] [20].
   Explanation: With normalized inputs, gradients behave more predictably. The cited text remarks that this keeps inputs in a "stable range" and makes training "faster and more stable" [19]. It also states batch norm lets one use higher learning rates and combats gradient issues [20].

3. **Aside from normalizing training data or layer inputs, what other normalization technique is common?**
   A. Scaling features to zero mean and unit variance (standardization). **(Correct)** [21].
   B. One-hot encoding of all numerical features.
   C. Using raw integer values only.
   D. Ignoring feature scales.
   Answer: A. It is common to standardize input features to zero mean and unit variance before training, which improves gradient descent behavior and mitigates gradient issues [21].
   Explanation: Properly scaled inputs ensure that all features contribute similarly to gradient updates. The reference explains that normalization (e.g. standardization) leads to faster convergence and helps mitigate vanishing/exploding gradients [21]. This is a data preprocessing step complementary to techniques like batch norm.

4. **What is the effect of min-max scaling on input features?**
   A. It centers features around zero mean with unit variance.
   B. It projects features onto a Gaussian distribution.
   C. It scales feature values into the [0,1] range. **(Correct)** [22] .
   D. It transforms features into one-hot vectors.
   Answer: C. Min-max scaling (normalization) linearly rescales features so their values lie between 0 and 1 (using (x–min)/(max–min)) [22] .
   Explanation: Min-max normalization maps the minimum value of a feature to 0 and the maximum to 1. The formula shown normalizes any x to x' = (x – min)/(max – min) [22] , ensuring all features are within [0,1]. This can be useful when the activation ranges are bounded (e.g. sigmoid outputs).

5. **Which of the following best explains why normalizing (scaling) input data is helpful?**
   A. It avoids dividing by zero during computation.
   B. It ensures features are on comparable scales, improving convergence speed and stability [21] . **(Correct)**.
   C. It is required for models to work at all.
   D. It always increases the number of epochs needed.
   Answer: B. Scaling input features to a similar range (e.g. zero mean, unit variance) speeds up gradient descent convergence and helps prevent extreme gradients [21] .
   Explanation: If features vary widely in scale, the gradient descent steps will oscillate or converge slowly. Normalization makes the optimization landscape more regular. The cited text specifically notes that normalization “mitigates the vanishing or exploding gradient problem” and leads to faster convergence [21] .

Session 12: Optimization Algorithms including ADAM, Mini-batch GD

1. **What is stochastic gradient descent (SGD)?**
   A. Updating weights using the entire training set for each step.
   B. Updating weights using a single random training example per step. **(Correct)** [23] .
   C. An optimizer that requires no gradients.
   D. Only used for regression tasks.
   Answer: B. SGD updates the model parameters using the gradient computed from one randomly selected training example (or a very small batch) at each iteration [23] .
   Explanation: Pure SGD picks one example, computes its gradient, and steps the weights. This results in noisy but frequent updates. The reference indicates that SGD computes the update on a single example, making it stochastic [23] .

2. **What is mini-batch gradient descent?**
   A. A method that uses a small batch of training examples to compute each update [24] . **(Correct)**
   B. Same as SGD.
   C. Only uses two examples at a time.
   D. An optimizer that requires closed-form solutions.
   Answer: A. Mini-batch gradient descent splits the training set into small batches (e.g. 32–256 samples) and performs a weight update using each mini-batch [24] .
   Explanation: Mini-batch GD is a compromise between full-batch and stochastic GD. It computes the

gradient on a subset (batch) of data. The excerpt notes that the dataset is split into “small batches,” making each update less noisy than SGD but more frequent than full-batch updates [24].

3. **Why is mini-batch gradient descent often preferred over pure SGD or full-batch GD?**
   A. It ignores gradients altogether.
   B. It provides faster convergence and computational efficiency by balancing update variance and cost [25]. **(Correct)**
   C. It requires no hyperparameters.
   D. It always finds the global minimum.
   Answer: B. Mini-batch GD often converges faster because it reduces variance compared to SGD (leading to steadier updates) while still requiring less computation per update than full-batch GD [25].
   Explanation: The reference explains that mini-batches need fewer updates than SGD (so faster per epoch) and have smaller gradient variance than very small batches, yielding efficient learning. This blend of speed and stability makes mini-batch a common default [25].

4. **What does the Adam optimizer combine?**
   A. Momentum and Nesterov acceleration.
   B. Adagrad and SGD.
   C. AdaGrad and RMSProp. **(Correct)** [26].
   D. Batch and mini-batch updates.
   Answer: C. Adam combines ideas from AdaGrad (adaptive learning rates) and RMSProp (decaying averaged squared gradients) to adaptively scale updates for each parameter [26].
   Explanation: Adam (Adaptive Moment Estimation) computes individual adaptive learning rates using both the first moment (mean of gradients) and second moment (uncentered variance). The source states that Adam merges AdaGrad and RMSProp techniques and is effective for sparse or noisy data [26].

5. **Why is Adam often recommended as a default optimizer in deep learning?**
   A. It always yields the best final model accuracy.
   B. It converges faster and generally performs well across many problems. **(Correct)** [27].
   C. It uses no hyperparameters.
   D. It does not use gradients.
   Answer: B. Adam is popular because it tends to achieve good results quickly and is robust; many practitioners use it as a default choice [27].
   Explanation: Adam automatically adjusts the learning rate for each parameter using running averages of gradients. The reference highlights that Adam “achieves good results fast” and is often a reliable default optimizer for training deep models [27]. Its efficiency and minimal need for tuning make it widely used.

Session 13: RMSProp, Momentum, Other Gradient Descent Variants

1. **What is momentum in the context of gradient descent?**
   A. A random restart strategy.
   B. Adding the previous update (scaled) to the current update to accelerate learning. **(Correct)** [28].
   C. Using a different learning rate for each parameter.
   D. Ignoring past gradients.

Answer: B. Gradient descent with momentum maintains a velocity vector that is a decaying average of past gradients, effectively adding a fraction of the previous update into the current one [28].
Explanation: The momentum term "remembers" the direction of past updates, helping the optimization build speed in consistent directions and damp oscillations. The equations show that the update v incorporates α·v (previous velocity) minus the current gradient, illustrating this effect [28].

2. **What is RMSProp?**
   A. A constant learning rate optimizer.
   B. An adaptive learning-rate method that divides the learning rate by a moving average of recent squared gradients [29]. **(Correct)**
   C. A regularization technique.
   D. An unsupervised learning algorithm.
   Answer: B. RMSProp (Root Mean Square Propagation) adapts the learning rate for each weight by dividing by an exponentially decaying average of past squared gradients [29].
   Explanation: RMSProp addresses AdaGrad's diminishing learning rates by using a "leaky" or exponential average of squared gradients. The reference indicates that as the average grows, the effective learning rate for that weight shrinks, stabilizing training. It is explicitly described as adaptive in this way [29].

3. **How does RMSProp improve upon AdaGrad?**
   A. It uses momentum instead of gradients.
   B. It decays the accumulated gradients exponentially (a "leaky" sum), preventing the learning rate from shrinking too much [30]. **(Correct)**
   C. It turns off adaptation after a fixed number of epochs.
   D. It uses the minimum gradient instead of the average.
   Answer: B. Unlike AdaGrad which sums up all past squared gradients, RMSProp uses an exponential moving average ("leaky" integration), which prevents the accumulated value from growing indefinitely and thereby keeps learning rates from shrinking too drastically [30].
   Explanation: AdaGrad's sum of squares can make the denominator very large over time (causing tiny steps). RMSProp's fix is to "forget" old gradients gradually by exponential weighting. The cited text explicitly mentions making AdaGrad's aggregation "leaky" via moving average [30].

4. **Which of the following describes a key difference between SGD with momentum and vanilla SGD?**
   A. Momentum uses Adam's formulas.
   B. Momentum maintains a velocity term, effectively integrating past gradients [28]. **(Correct)**
   C. Momentum does not use gradients at all.
   D. There is no difference; they are the same algorithm.
   Answer: B. SGD with momentum computes a "velocity" that accumulates past gradient directions, unlike vanilla SGD which uses only the current gradient. The formula shows v_t = α·v_{t-1} – ε·grad, indicating this accumulation [28].
   Explanation: The momentum update adds a portion (α) of the previous velocity, meaning updates persist in consistent directions. This is unlike plain SGD, which would only use –ε·grad (and would correspond to α=0). The provided snippet confirms this difference in update rules [28].

5. **Which optimization method scales the learning rate per parameter based on recent gradient magnitudes?**

A. Vanilla SGD
B. RMSProp **(Correct)** [29]
C. SGD with constant momentum
D. Genetic algorithms
Answer: B. RMSProp (and similarly Adam) adapt each parameter's learning rate according to a running average of the squared gradients [29].
Explanation: The defining feature of RMSProp is adaptive per-parameter step sizes. It divides by $\sqrt{}$(running average of squared gradients), so if a parameter's gradients are large, its learning rate effectively shrinks [29]. This contrasts with SGD/momentum which uses the same LR for all parameters.

Sessions 14–15: TensorFlow Data Structures and MNIST Applications

1. **Which TensorFlow class is typically used to represent an input data pipeline?**
   A. `tf.Tensor`
   B. `tf.Variable`
   C. `tf.data.Dataset` **(Correct)** [31].
   D. `tf.constant`
   Answer: C. `tf.data.Dataset` is TensorFlow's class for representing a potentially large set of data elements (e.g. training examples) that can be efficiently fed into a model [31].
   Explanation: The `tf.data.Dataset` API is designed for input pipelines, handling streaming, shuffling, batching, etc. The reference explicitly describes it as representing a "potentially large set of elements" in a dataset, highlighting its use for managing training data [31].

2. **Approximately how many training examples are in the MNIST dataset?**
   A. 60,000 **(Correct)** [32].
   B. 6,000
   C. 600
   D. 600,000
   Answer: A. The MNIST dataset contains 60,000 training images and 10,000 test images [32].
   Explanation: MNIST is a standard benchmark of handwritten digits. According to the TensorFlow Datasets description, it has 60,000 examples for training and 10,000 for evaluation [32].

3. **What is the dimensionality of each MNIST image?**
   A. 32×32 grayscale.
   B. 28×28 pixels in grayscale. **(Correct)** [33].
   C. 224×224 RGB.
   D. 1024×1024 RGB.
   Answer: B. Each MNIST image is 28×28 pixels in grayscale (single channel) [33].
   Explanation: The dataset description notes `image: uint8 (28, 28, 1)`, meaning height=28, width=28, and 1 color channel (grayscale) [33].

4. **How many classes are in the MNIST classification task?**
   A. 2
   B. 5
   C. 10 **(Correct)** [33].

D. 100
Answer: C. MNIST consists of digits 0 through 9, so there are 10 classes [33].
Explanation: The description explicitly shows `label: int64 0-9`, indicating labels range from 0 to 9 (ten classes) [33].

5. **What data type are MNIST image pixels, and what is their value range?**
A. `float32` in [–1, 1]
B. `int32` in [0, 1000]
C. `uint8` in [0, 255] **(Correct)** [34].
D. `string` values.
Answer: C. MNIST images are stored as `uint8` values in the range 0 to 255 (standard 8-bit grayscale) [34].
Explanation: The TFDS info shows dtype=uint8 and values in [0, 255] for pixel intensities [34]. Models often normalize these to [0,1] or [-1,1], but raw MNIST uses 0–255.

Session 16: Sequential Modeling, RNNs and Applications

1. **What is a key feature of Recurrent Neural Networks (RNNs) compared to feedforward networks?**
A. RNNs do not use activation functions.
B. RNNs maintain a hidden state that carries information across time steps [35]. **(Correct)**
C. RNNs can only process fixed-size input.
D. RNNs always have fewer parameters than feedforward nets.
Answer: B. RNNs have recurrent (hidden) states that "remember" information from prior time steps, enabling them to capture sequential dependencies [35].
Explanation: The cited text emphasizes that a recurrent neuron retains a hidden state representing information about previous inputs in the sequence [35]. This memory enables RNNs to process sequences like text or time series, as they carry forward context.

2. **Which of the following tasks is most naturally suited to RNNs?**
A. Image classification of isolated images.
B. Classifying tabular data with fixed features.
C. Sequential data tasks like language modeling and time series prediction [36]. **(Correct)**
D. Simple linear regression.
Answer: C. RNNs excel at tasks involving sequential or temporal data (e.g. predicting next word in a sentence), since they can use past inputs to inform current outputs [36].
Explanation: The example given describes reading a sentence and predicting the next word by remembering previous words [36]. Such sequential dependency problems (language, audio, time series) are where RNNs are typically applied.

3. **What problem do standard RNNs often encounter when trying to learn long-term dependencies?**
A. Vanishing gradient problem, which makes it hard to learn from distant past inputs [37]. **(Correct)**
B. Their predictions become too accurate.
C. They cannot use more than one hidden layer.
D. They only work with images.
Answer: A. RNNs face the vanishing gradient problem for long sequences, meaning gradients shrink

and earlier layers learn very slowly, limiting long-term dependency learning [37].
Explanation: In deep sequence problems, gradients propagated back in time diminish (vanish), so the model forgets early inputs. The reference explicitly notes vanishing gradients hamper learning of long-range dependencies [37].

4. **How do LSTM (Long Short-Term Memory) networks address the vanishing gradient problem?**
A. By using linear activations only.
B. By incorporating gating mechanisms (input, forget, output gates) to regulate the flow of information [38]. **(Correct)**
C. By having no hidden states.
D. By reducing all weights to zero.
Answer: B. LSTMs add gated units (input, forget, output gates) in each cell, allowing gradients to flow unchanged through time steps and thus solving the vanishing gradient issue [38].
Explanation: Each LSTM cell maintains a cell state that can carry information across many steps. The gates control what information to keep or discard, enabling the network to preserve long-term dependencies. The excerpt notes that LSTM cells have input, forget, and output gates to overcome vanishing gradients [38].

5. **What does backpropagation through time (BPTT) mean in the context of RNNs?**
A. Training RNNs layer by layer instead of all at once.
B. Applying standard backpropagation to the unrolled RNN across time steps. **(Correct)**.
C. Ignoring all past time steps during backprop.
D. A type of gradient clipping.
Answer: B. BPTT refers to unfolding the RNN across time (unrolling the recurrent structure) and then applying standard backpropagation through this expanded network [39].
Explanation: The network is "unfolded" over the sequence length, creating a deep feedforward graph in time. Backpropagation is then applied through this unrolled graph. The text describes "RNN unfolding" and how it enables BPTT, which updates weights across time-dependent layers [40].

Sessions 17–18: CNNs, Inception Networks, Transfer Learning, Pooling, Use Cases

1. **What is the primary purpose of a pooling layer in a CNN?**
A. To increase the spatial dimensions of feature maps.
B. To reduce the spatial dimensions of feature maps while retaining important information [41]. **(Correct)**
C. To combine multiple network models.
D. To perform convolution with larger kernels.
Answer: B. Pooling layers downsample feature maps (reduce width and height) by summarizing local regions, which reduces parameters and provides translation invariance [41].
Explanation: The source states that pooling "reduces the spatial dimensions" of the input. This compression helps decrease computation and induces invariance to small shifts. Common pooling (max or average) slides a window and summarizes (e.g. max or average) the activations in that window [41].

2. **What does max pooling do in a CNN?**
   A. It multiplies all values in the pooling window.
   B. It selects the largest value within each pooling window, preserving strong activations [42]. **(Correct)**
   C. It selects the smallest value in each window.
   D. It averages all values in the window.
   Answer: B. Max pooling takes the maximum element from each region of the feature map covered by the filter [42].
   Explanation: For each pooling region, max pooling outputs the largest activation (the most prominent feature). The text says it selects the maximum element in the window, effectively keeping the most salient feature in that patch [42].

3. **What does average pooling do in a CNN?**
   A. It selects the largest value in the pooling region.
   B. It computes the average of all values in each pooling window, giving a smoother, more generalized summary [43]. **(Correct)**
   C. It selects the smallest value.
   D. It applies a Gaussian blur to the feature map.
   Answer: B. Average pooling outputs the mean of the values within each pooling window [43].
   Explanation: Unlike max pooling, average pooling takes all values in the region and averages them. This yields a downsampled map that reflects the average presence of features. The source notes that average pooling "gives the average of features present in a patch" [43].

4. **What is transfer learning in the context of CNNs?**
   A. Training a model on one task and then reusing (transferring) parts of it for a new related task. **(Correct)** [44] [45].
   B. Moving data from GPU to CPU.
   C. Converting models between TensorFlow and PyTorch.
   D. Training two models simultaneously on different data.
   Answer: A. Transfer learning involves taking a pretrained network (e.g. trained on a large image dataset) and using its learned features for a new task, either by feature extraction or fine-tuning [44] [45].
   Explanation: The idea is that a model trained on a general task (like ImageNet classification) has learned useful visual features. As the text says, a pretrained model on a large dataset "serves as a generic model of the visual world" and can be repurposed for new tasks [44] [45].

5. **What is *feature extraction* in transfer learning?**
   A. Training an entirely new model from scratch.
   B. Using a pretrained network as a fixed feature extractor and only training a new classifier on top [46]. **(Correct)**
   C. Extracting features manually from data.
   D. Removing features from the dataset.
   Answer: B. Feature extraction means freezing the pretrained base model and only training new top layers (classifier layers) on the features it produces [46].
   Explanation: In this approach, the convolutional base learned from a large dataset is used as-is. Only the final layer(s) are trained on the new task's data. The reference describes this by saying we "use

the representations learned by a previous network" and add a new classifier, without retraining the base [46].

6. **What is *fine-tuning* in transfer learning?**
   A. Lowering the learning rate dynamically.
   B. Unfreezing some top layers of a pretrained model and jointly training them with the new classifier layers [47]. **(Correct)**
   C. Using a smaller batch size.
   D. Reducing the dropout rate to 0.
   Answer: B. Fine-tuning involves taking a pretrained model, unfreezing some higher layers, and continuing training (along with the new top layers) so that the network's features adapt to the new task [47].
   Explanation: Instead of keeping the base fixed, fine-tuning allows the feature extractor itself to be adjusted. The source explains that one "unfreezes a few of the top layers" and trains them with the new classifier, making the learned features more specific to the target task [47].

7. **What is a hallmark of the Inception (GoogLeNet) architecture?**
   A. It uses only 1×1 convolutions.
   B. It processes images sequentially without parallel paths.
   C. It parallelizes convolutions of different filter sizes in each block [48]. **(Correct)**
   D. It has no activation functions.
   Answer: C. Inception modules apply multiple convolution filters (e.g. 1×1, 3×3, 5×5) in parallel and concatenate their outputs, allowing the network to capture multi-scale features efficiently [48].
   Explanation: The key idea of Inception is to run parallel convolutions of different sizes on the same input (as the quote describes), then stack their feature maps. This lets the network decide which filter size best captures features at each layer, without excessively increasing depth [48].

8. **How does adding a 1×1 convolution in an Inception module help reduce computational cost?**
   A. It increases the number of channels.
   B. It acts as a bottleneck that reduces the number of feature channels before expensive larger convolutions. **(Correct)** [49].
   C. It replaces max pooling.
   D. It has no effect on parameters.
   Answer: B. A 1×1 convolution can reduce the depth (number of channels) of the input, thus decreasing the parameters for subsequent larger convolutions [49].
   Explanation: By using 1×1 filters to compress the channel dimension, Inception modules lower the computation needed for deeper convolutions. The excerpt explicitly notes that inserting a 1×1 convolution "reduces the number of channels" fed into a 5×5 layer, thereby reducing parameters and cost [49].

9. **What is transfer learning particularly useful for in CNNs?**
   A. When you have a very large labeled dataset for your target task.
   B. When your target dataset is small, and you leverage features learned from a large dataset [45]. **(Correct)**
   C. For unsupervised clustering.
   D. When training must be done without GPUs.
   Answer: B. Transfer learning is most beneficial when one has a limited dataset for the new task;

using a model pretrained on a large dataset provides a rich feature set to start from [45].
Explanation: If data is scarce, starting from a model trained on, say, ImageNet avoids training from scratch. The tutorial notes that a model trained on a "large and general" dataset will have generic useful features, letting you "take advantage of these learned feature maps" instead of learning entirely anew [45].

10. **Which pooling operation would be most appropriate just before a final classification layer to reduce each feature map to a single value?**
A. 2×2 max pooling.
B. Global average pooling. **(Correct)** [50].
C. 2×2 average pooling with stride 2.
D. 1×1 convolution.
Answer: B. Global average pooling computes one number per channel (averaging over the entire feature map), reducing each channel to 1×1 size [50].
Explanation: Global pooling summarizes each feature map completely. The text describes global average pooling as computing the average over the entire map, outputting a single value per channel [50]. This is often used to prepare for a final dense layer in classification networks.

Session 19: GANs and Implementation Basics

1. **What does *GAN* stand for?**
A. Generative Automatic Network
B. Generative Adversarial Network **(Correct)** [51].
C. Graphical Augmentation Network
D. Generalized Adaptive Network
Answer: B. GAN stands for *Generative Adversarial Network*, a framework where two neural networks compete in a zero-sum game [51].
Explanation: The original GAN formulation by Goodfellow et al. defines it as a generative model using adversarial training. The reference explicitly calls it a generative adversarial network and describes the two-network competitive setup [51].

2. **In a GAN, what roles do the generator and discriminator play?**
A. Generator classifies images; discriminator generates images.
B. Generator and discriminator share the same task.
C. Generator creates fake samples to fool the discriminator; discriminator learns to distinguish real from fake samples [52]. **(Correct)**
D. Both generate data and do no classification.
Answer: C. The generator produces synthetic data (e.g. images) aiming to fool the discriminator, while the discriminator tries to tell apart real training samples from the generator's fake samples [52].
Explanation: This adversarial process is central to GANs: the discriminator outputs a probability of "realness," and the generator updates its parameters to make its outputs more convincing. The text notes the generator learns to "fool the discriminator," and the discriminator learns to detect that deception [52].

3. **What is the training objective in a standard GAN?**
A. A regression loss on generated images.

B. A zero-sum game where the generator minimizes log(1–D(G(z))) and the discriminator maximizes it [53]. **(Correct)**
C. Maximizing the mutual information between generator and discriminator.
D. There is no objective function.
Answer: B. GANs are trained with a minimax objective: the discriminator tries to maximize log(D(x)) + log(1–D(G(z))), while the generator tries to minimize log(1–D(G(z))) (i.e. fool the discriminator) [53].
Explanation: The Wikipedia definition shows the classic GAN game: the two networks' payoff is a zero-sum objective L = E[log D(real)] + E[log(1–D(fake))]. The generator aims to minimize this (fooling D), and D aims to maximize it [53].

4. **What is a typical use case for GANs mentioned in the text?**
A. Exact numerical predictions
B. Generating new samples (e.g. photos) that mimic the training data distribution [54]. **(Correct)**
C. Clustering unlabeled data.
D. Reinforcement learning for games.
Answer: B. GANs are particularly known for learning to generate new data (like realistic images) that have the same statistical characteristics as the training set [54].
Explanation: The article states that given a dataset, a GAN can generate new samples that "look at least superficially authentic" and share the data distribution's characteristics [54]. While GANs have applications in semi-supervised or reinforcement learning, their hallmark is generative modeling.

5. **GAN training is often compared to what biological process?**
A. Natural selection.
B. Mimicry with an arms race between species. **(Correct)** [55].
C. Photosynthesis.
D. Neuron firing.
Answer: B. The GAN setup is likened to mimicry in evolutionary biology, where one species evolves to mimic another and there is a co-evolutionary "arms race" between them [55].
Explanation: The text uses the analogy of mimicry: the generator mimics real data, and the discriminator is like a species evolving to detect the mimicry. This dynamic is described as an "evolutionary arms race" [55].

Session 20: Model Tuning, Autoencoders, LSTM, Trends, Case Studies

1. **What is an autoencoder?**
A. A supervised network that classifies input data.
B. An unsupervised neural network that encodes inputs into a compressed representation and then decodes them back, learning to reconstruct the input [56]. **(Correct)**
C. A type of GAN.
D. A CNN for image classification.
Answer: B. An autoencoder is an unsupervised model with an encoder that compresses inputs to a lower-dimensional "code" and a decoder that reconstructs the original inputs from that code [56].
Explanation: As described, autoencoders learn to represent the data in a reduced dimension (the bottleneck) and then reconstruct it. The goal is that the bottleneck captures essential features. The reference defines it exactly as compressing and then reconstructing the data [56].

2. **What can the bottleneck (code) layer of an autoencoder be used for after training?**
   A. Labeling data for supervised learning.
   B. Feature extraction or dimensionality reduction for other tasks. **(Correct)** [57] .
   C. Inverting the images.
   D. Generating adversarial examples.
   Answer: B. The hidden code (bottleneck) layer serves as a reduced representation of the input and can be used for dimensionality reduction or as input features for other learning tasks (e.g. classification) [57] .
   Explanation: After training an autoencoder to reconstruct inputs, the code layer provides a compact set of features. The article lists dimensionality reduction and anomaly detection among applications [57] , showing that the code is useful as a reduced representation.

3. **Which of the following is *not* an application of autoencoders?**
   A. Dimensionality reduction.
   B. Anomaly detection.
   C. Image denoising.
   D. Real-time reinforcement learning policy. **(Correct)**
   Answer: D. Autoencoders are commonly used for dimensionality reduction, anomaly detection, denoising, etc., but they are not directly used as RL policies.
   Explanation: The text lists several applications (dimension reduction, anomaly detection, denoising, etc.) [57] . Real-time RL policies are outside the typical use cases of autoencoders. Thus D is not an application of autoencoders.

4. **How does an LSTM cell differ from a simple RNN cell?**
   A. LSTM uses convolution instead of full connections.
   B. LSTM has specialized gates (input/forget/output) allowing it to preserve gradients over longer sequences [38] . **(Correct)**
   C. LSTM has fewer parameters than RNN.
   D. LSTM only works on images, not sequences.
   Answer: B. LSTM cells incorporate gating mechanisms (input, forget, output gates) that regulate information flow, enabling the network to learn long-term dependencies without vanishing gradients [38] .
   Explanation: The defining feature of LSTM is its internal gating structure. These gates allow it to decide which information to keep or discard at each time step. The reference highlights these three gates as key to overcoming vanishing gradients [38] .

5. **Which technique is helpful for hyperparameter tuning of deep models?**
   A. Grid search or random search with validation sets. **(Correct)**.
   B. Completely ignoring validation performance.
   C. Always using the same hyperparameters for all tasks.
   D. Training without any test data.
   Answer: A. Systematic searches (grid or random) over hyperparameter combinations, evaluated on a validation set, are standard for tuning.
   Explanation: Though not explicitly cited, practical guidance is to use search methods or Bayesian optimization to pick learning rates, regularization strength, etc., based on validation performance. This is common knowledge in model tuning.

6. **What recent trend in model architectures is hinted at in modern deep learning (2024–2025)?**
   A. Exclusive use of fully connected networks.
   B. The rise of Transformer-based architectures and self-attention in vision and language tasks. **(Correct)**.
   C. The complete abandonment of convolutional networks.
   D. Manual feature engineering over end-to-end learning.
   Answer: B. Recent years have seen transformer models (with self-attention) becoming dominant in many areas like NLP and increasingly in vision.
   Explanation: While not directly cited, current trends (since 2020) include the use of Vision Transformers, diffusion models, and graph neural networks. Transformer architectures like BERT, GPT, ViT, etc. have become very influential.

7. **Which case study demonstrates using a pretrained CNN for a new classification task?**
   A. Training a small CNN from scratch on MNIST.
   B. Using InceptionV3 pretrained on ImageNet and fine-tuning on a custom dataset (as shown in [122]). **(Correct)**.
   C. Using k-means for image clustering.
   D. Solving MNIST with a GAN.
   Answer: B. The InceptionV3 case in [122] is an example of transfer learning: a pretrained ImageNet model is used for a 6-class classification problem, illustrating feature extraction/fine-tuning.
   Explanation: The referenced Medium article explicitly describes using a pretrained InceptionV3 model and training only its final layer for a new classification task [58] [59]. This is a prototypical case of model reuse.

8. **What is batch normalization's effect on internal covariate shift?**
   A. It ignores it.
   B. It exacerbates it by adding noise.
   C. It reduces it by normalizing batch inputs. **(Correct)** [18].
   D. It replaces backprop.
   Answer: C. BatchNorm explicitly normalizes each batch's activations to reduce shifts in layer input distribution during training [18].
   Explanation: By ensuring each mini-batch's features have mean 0 and variance 1 before scaling, batch norm stabilizes the distribution seen by each layer, directly addressing internal covariate shift [18].

9. **Which regularization technique can be interpreted as training an ensemble of subnetworks?**
   A. L2 regularization.
   B. Dropout. **(Correct)** [9].
   C. Weight decay.
   D. Data normalization.
   Answer: B. Dropout can be seen as implicitly training many thinned networks and averaging them, akin to an ensemble [9].
   Explanation: As each dropout mask yields a different network, at test time it's like averaging these models. The text says dropout provides an "ensemble effect," improving robustness [9].

10. **Why might one use an LSTM autoencoder?**
    A. For unsupervised encoding of sequential data, capturing temporal patterns. **(Correct)**.

B. To generate random noise.
C. For solving linear equations.
D. It's not a real model.
Answer: A. An LSTM autoencoder uses LSTM cells in the encoder/decoder and is suited for sequences (e.g. time series anomaly detection) by learning compressed sequential representations.
Explanation: Although not cited above, LSTM autoencoders are a known pattern: they encode sequences into a fixed-size vector (via LSTM encoder) and then decode. They capture temporal dependencies and are used for tasks like anomaly detection in time series.

11. **Which technique can help adjust learning rate during training?**
A. Data shuffling.
B. Learning rate scheduling (e.g. reduce-on-plateau). **(Correct)**.
C. Backpropagation.
D. Batch size tuning.
Answer: B. Learning rate schedules (exponential decay, step decay, or adaptive schemes) are used to lower (or occasionally increase) the learning rate during training.
Explanation: In practice, schedulers reduce LR after certain epochs or if validation plateaus. This is a standard tuning technique to ensure convergence. (Though not in citations, this is a common model tuning method.)

12. **In model tuning, what is *cross-validation* used for?**
A. Ensuring random data splitting.
B. Estimating model performance reliably by training on multiple train/validation splits. **(Correct)**.
C. Cross-entropy calculation.
D. Data normalization.
Answer: B. Cross-validation (e.g. k-fold) is used to validate model performance across multiple splits, reducing dependency on a single train/val partition and aiding hyperparameter tuning.
Explanation: While deep nets often use a single held-out validation set, cross-validation is a general technique to more robustly measure generalization performance by averaging across folds.

13. **Why might one incorporate *early stopping* in model training?**
A. To reduce the model's parameter count.
B. To stop training when further improvement on validation set is unlikely (prevent overfitting). **(Correct)** [10].
C. To speed up data loading.
D. To ensure zero training error.
Answer: B. Early stopping terminates training when validation performance stops improving, thereby avoiding overfitting [10].
Explanation: This was already covered: it's exactly to prevent training beyond the point where the model starts to overfit the training data. The reference confirms that strategy [10].

14. **Which model architecture is a successor to LSTMs for many sequence tasks?**
A. Simple RNN.
B. Transformer (with self-attention). **(Correct)**.
C. Perceptron.
D. CNN.
Answer: B. Transformer models (e.g. those using self-attention) have largely surpassed LSTMs in

NLP and increasingly in vision, due to better long-range modeling.
Explanation: Modern trends involve Transformers like BERT, GPT for sequence modeling, which rely on attention rather than recurrence. (This is a known trend beyond the given sources.)

15. **What is *fine-tuning* in model reuse, and why is it useful?**
A. Training without validation.
B. Only adjusting hyperparameters.
C. Starting from pretrained weights and continuing training on new data, allowing model to adapt to a specific task. **(Correct)**.
D. Removing layers from a pretrained model.
Answer: C. Fine-tuning means you start from a pretrained model and continue (some) training on your task data, which can quickly adapt generic features to your specific problem.
Explanation: This builds on transfer learning: instead of freezing the base, one "unfreezes" layers and retrains. It's useful when your new dataset is somewhat large or similar, to specialize the model. (Covered conceptually in [121] and [122].)

16. **Which of the following is an example of hyperparameter tuning?**
A. Adjusting the learning rate based on validation loss. **(Correct)**.
B. Training on the test set.
C. Removing dropout layers.
D. Hard-coding weights.
Answer: A. Selecting or adapting the learning rate, batch size, regularization factors, etc., based on validation performance is an example of hyperparameter tuning.
Explanation: Hyperparameter tuning involves techniques like grid search over learning rates, monitoring loss to reduce learning rate, etc. It is distinct from training the model's parameters.

17. **What is *overfitting*?**
A. When a model learns the training data too well and fails to generalize to new data. **(Correct)**.
B. When a model underperforms on training data.
C. When the model's parameters become infinite.
D. When training loss reaches exactly zero.
Answer: A. Overfitting occurs when a model captures noise or idiosyncrasies of the training set and thus performs poorly on unseen data.
Explanation: This is a basic ML concept: an overfit model has low training error but high validation/test error. Regularization, dropout, early stopping, etc., are ways to combat it.

18. **What is *dropout rate* in dropout regularization?**
A. The number of layers to drop.
B. The probability (e.g. 0.5) that a given neuron is set to zero during training. **(Correct)**.
C. The fraction of training data ignored each epoch.
D. The learning rate multiplied by epoch.
Answer: B. The dropout rate is the fraction of units that are randomly disabled (output set to zero) in each forward pass during training.
Explanation: For example, dropout=0.5 means each neuron has a 50% chance of being dropped each step. (Standard definition; related to [61] content on dropout.)

19. **What role does a validation set play during training?**
    A. It trains the model.
    B. It is used to tune hyperparameters and monitor overfitting, separate from the test set. **(Correct)**.
    C. It is only used for final evaluation.
    D. It is discarded.
    Answer: B. The validation set provides an unbiased evaluation of the model fit during training and is used for hyperparameter tuning (such as early stopping) without using test data.
    Explanation: This is standard ML practice: training for fitting, validation for tuning, test for final check. (Implicit in tuning early stopping discussion.)

20. **Which of the following is *not* typically tuned via hyperparameters?**
    A. Learning rate.
    B. Network architecture (number of layers/neurons).
    C. Weights of the network. **(Correct)**
    D. Regularization strength.
    Answer: C. Network weights (parameters) are learned from data, not set as hyperparameters; hyperparameters are user-chosen settings like learning rate or architecture.
    Explanation: Weights are adjusted by training; hyperparameters are fixed before training.

Sources: Educational content and official documentation on deep learning concepts and frameworks [2] [3] [4] [60] [7] [8] [9] [61] [11] [12] [13] [15] [14] [16] [17] [18] [19] [20] [62] [21] [23] [24] [25] [27] [26] [63] [30] [29] [28] [64] [31] [65] [35] [39] [38] [37] [41] [42] [43] [44] [66] [47] [48] [49].

---

[1] [2] PyTorch vs TensorFlow: Comparative Guide of AI Frameworks 2025
https://opencv.org/blog/pytorch-vs-tensorflow/

[3] web.stanford.edu
https://web.stanford.edu/~jurafsky/slp3/slides/7_NN_Apr_28_2021.pdf

[4] Universal approximation theorem - Wikipedia
https://en.wikipedia.org/wiki/Universal_approximation_theorem

[5] [6] [60] What is the Difference Between a Parameter and a Hyperparameter? - MachineLearningMastery.com
https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/

[7] Fighting Overfitting With L1 or L2 Regularization: Which One Is Better?
https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization

[8] [9] [10] [61] Dropout Regularization in Deep Learning - GeeksforGeeks
https://www.geeksforgeeks.org/deep-learning/dropout-regularization-in-deep-learning/

[11] [12] A Complete Guide to Data Augmentation | DataCamp
https://www.datacamp.com/tutorial/complete-guide-data-augmentation

[13] [14] [15] [16] Vanishing and Exploding Gradients Problems in Deep Learning - GeeksforGeeks
https://www.geeksforgeeks.org/deep-learning/vanishing-and-exploding-gradients-problems-in-deep-learning/

[17] Grad check - Improving Deep Neural Networks: Hyperparameter tun - DeepLearning.AI
https://community.deeplearning.ai/t/grad-check/413129

18 19 20 62 What is Batch Normalization In Deep Learning? - GeeksforGeeks
https://www.geeksforgeeks.org/deep-learning/what-is-batch-normalization-in-deep-learning/

21 22 What is Normalization in Machine Learning? A Comprehensive Guide to Data Rescaling | DataCamp
https://www.datacamp.com/tutorial/normalization-in-machine-learning

23 24 25 Differences Between Gradient, Stochastic and Mini Batch Gradient Descent | Baeldung on Computer Science
https://www.baeldung.com/cs/gradient-stochastic-and-mini-batch

26 27 63 Gentle Introduction to the Adam Optimization Algorithm for Deep Learning - MachineLearningMastery.com
https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

28 29 30 64 RMSProp vs Momentum in Deep Learning - Data Science Stack Exchange
https://datascience.stackexchange.com/questions/76408/rmsprop-vs-momentum-in-deep-learning

31 tf.data.Dataset | TensorFlow v2.16.1
https://www.tensorflow.org/api_docs/python/tf/data/Dataset

32 33 34 65 mnist | TensorFlow Datasets
https://www.tensorflow.org/datasets/catalog/mnist

35 36 37 38 39 40 Introduction to Recurrent Neural Networks - GeeksforGeeks
https://www.geeksforgeeks.org/machine-learning/introduction-to-recurrent-neural-network/

41 42 43 50 CNN | Introduction to Pooling Layer - GeeksforGeeks
https://www.geeksforgeeks.org/deep-learning/cnn-introduction-to-pooling-layer/

44 45 46 47 66 Transfer learning and fine-tuning | TensorFlow Core
https://www.tensorflow.org/tutorials/images/transfer_learning

48 49 58 59 Understanding Architecture Of Inception Network & Applying It To A Real-World Dataset | by Alifia Ghantiwala | Medium
https://gghantiwala.medium.com/understanding-the-architecture-of-the-inception-network-and-applying-it-to-a-real-world-dataset-169874795540

51 52 53 54 55 Generative adversarial network - Wikipedia
https://en.wikipedia.org/wiki/Generative_adversarial_network

56 57 Dimensionality Reduction using AutoEncoders in Python - Analytics Vidhya
https://www.analyticsvidhya.com/blog/2021/06/dimensionality-reduction-using-autoencoders-in-python/