Machine Learning Crash Course Outline (Google Developers) - With Math Recaps

This outline details the key concepts and topics typically covered in a foundational Machine Learning Crash Course, similar to the one offered by Google Developers. It provides a structured pathway through the essentials of supervised machine learning, with a focus on practical applications and fundamental theory, now enhanced with brief mathematical equations and explanations.

1. Introduction to Machine Learning

- **1.1. What is Machine Learning?**
  - **Definition and core idea:** Algorithms learn patterns from data to make predictions or decisions without explicit programming.
  - **Comparison with traditional programming:** ML learns rules from data, while traditional programming explicitly defines rules.
  - **Real-world examples:** Image recognition, spam filtering, product recommendations, voice assistants.
- **1.2. Types of Machine Learning**
  - **Supervised Learning:** Learns from labeled data (input-output pairs).
    - **Regression:** Predicts continuous numerical values (e.g., house price).
    - **Classification:** Predicts discrete categories or classes (e.g., spam/not spam).
  - **Unsupervised Learning:** Finds patterns or structures in unlabeled data (e.g., grouping similar customers).
  - **Reinforcement Learning:** Learns optimal actions through trial-and-error in an environment (e.g., game playing AI).
- **1.3. The Machine Learning Workflow**
  - **Problem definition and data collection:** Clearly define what to solve and gather relevant data.
  - **Data preprocessing and feature engineering:** Clean, transform, and create new features from raw data.
  - **Model selection and training:** Choose an algorithm and teach it using the prepared data.
  - **Evaluation and hyperparameter tuning:** Assess model performance and optimize its settings.
  - **Deployment and monitoring:** Put the model into use and track its performance over time.

2. Core Concepts of Supervised Learning

- **2.1. Features and Labels**
  - **Features (x):** The input variables or attributes used for prediction (e.g., square footage of a house).

    - **Labels (y):** The output variable or target we want to predict (e.g., the price of a house).
    - **Examples:** For predicting house prices, features are size, location; label is price.
- **2.2. Models and Training**
    - **Model:** A mathematical representation or function (f) that learns the relationship between features (x) and labels (y), often denoted as $\hat{y}=f(x)$ where $\hat{y}$ is the prediction.
    - **Training:** The iterative process of adjusting the model's internal parameters (weights) to minimize errors on the training data.
    - **Prediction:** The output generated by the trained model ($\hat{y}$) when given new input features (x).
- **2.3. Loss and Cost Functions**
    - **Loss Function ($L(\hat{y},y)$):** Quantifies the error for a *single* prediction $\hat{y}$ compared to the actual label y.
    - **Cost Function (or Objective Function, $J(\theta)$):** The average loss over the *entire* training dataset, where θ represents the model's parameters. The goal of training is to minimize this overall error.
- **2.4. Optimization: Gradient Descent**
    - **Concept:** An iterative algorithm used to find the set of model parameters (θ) that minimizes the cost function ($J(\theta)$).
    - **Intuition:** Imagine descending a hill; each step moves you towards the lowest point (minimum loss).
    - **Gradient:** The partial derivative of the cost function with respect to each parameter, indicating the direction of steepest ascent. Gradient descent moves in the opposite direction.
    - Parameter Update Rule:
      $\theta_{new}=\theta_{old}-\alpha\cdot\nabla J(\theta_{old})$

      where α is the learning rate and $\nabla J(\theta_{old})$ is the gradient of the cost function.
- **2.5. Learning Rate (α)**
    - **Definition:** A hyperparameter that controls the step size at each iteration of gradient descent.
    - **Impact:** A too-high rate can overshoot the minimum; a too-low rate can make training very slow or get stuck in local minima.

3. Regression: Predicting Continuous Values

- **3.1. Linear Regression**
    - Hypothesis Function (for a single feature x_1): A simple linear equation:
      $\hat{y}=w_1x_1+b$

      where $\hat{y}$ is the predicted label, w_1 is the weight (slope), and b is the bias (y-intercept).
    - Hypothesis Function (for multiple features $x_1,\ldots,x_n$):
      $\hat{y}=w_1x_1+w_2x_2+\cdots+w_nx_n+b=w^Tx+b$

where w is the vector of weights and x is the vector of features.
    - **Parameters:** w (weights/slopes) determine feature importance; b (bias/intercept) shifts the line up or down.
    - **Training Objective:** Find the optimal w and b values that make the predicted y values as close as possible to the true y values, minimizing the cost.
- **3.2. Minimizing Loss for Linear Regression**
    - Mean Squared Error (MSE): The most common loss function for regression, calculating the average of the squared differences between predicted and actual values over m samples.
      MSE=m1i=1Σm(y(i)−y^(i))2

      where y(i) is the actual label for sample i, and y^(i) is the model's prediction for sample i.
    - **Applying Gradient Descent to MSE:** Gradient descent iteratively adjusts weights to reduce the MSE by moving in the direction opposite to the partial derivative of MSE with respect to each weight.
- **3.3. Practical Considerations**
    - **Visualizing linear regression:** Plotting the line of best fit through data points.
    - **Multivariate linear regression:** Extending the concept to models with multiple input features.

4. Classification: Predicting Discrete Categories

- **4.1. Logistic Regression (Binary Classification)**
    - **Problem:** Predicting one of two possible outcomes (e.g., 0 or 1, true/false, positive/negative).
    - Sigmoid Function (σ(z)): Squashes any real number input z into a value between 0 and 1, interpreting it as a probability.
      σ(z)=1+e−z1

      where z=wTx+b.
    - Loss Function (Binary Cross-Entropy / Log Loss): Measures the performance of a classification model whose output is a probability. It heavily penalizes confident wrong predictions. For a single sample:
      L(y^,y)=−[ylog(y^)+(1−y)log(1−y^)]

      where y is the true label (0 or 1) and y^ is the predicted probability.
- **4.2. Multi-class Classification**
    - **One-vs-Rest (OvR) or One-vs-All (OvA):** A strategy to handle multiple classes by training a separate binary classifier for each class against all other classes.
    - Softmax Function: An extension of the sigmoid function that takes a vector of arbitrary real values and transforms them into a probability distribution over multiple classes, where probabilities sum to 1. For a vector z=[z1,...,zK] (for K classes):
      P(yk∣z)=Σj=1Kezjezk

- **4.3. Evaluation Metrics for Classification**
  - Accuracy: The proportion of total predictions that were correct.
    Accuracy=Total Number of PredictionsNumber of Correct Predictions
  - Precision: The proportion of positive identifications that were actually correct.
    Precision=True Positives+False PositivesTrue Positives
  - Recall (Sensitivity): The proportion of actual positives that were correctly identified.
    Recall=True Positives+False NegativesTrue Positives
  - F1-score: The harmonic mean of Precision and Recall, providing a balance between them.
    F1=2·Precision+RecallPrecision·Recall
  - **Confusion Matrix:** A table summarizing true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), detailing classifier performance.
  - **ROC Curve and AUC (Area Under the Curve):** A graphical plot illustrating the diagnostic ability of a binary classifier system as its discrimination threshold is varied; AUC summarizes the overall performance.

5. Neural Networks: Going Deeper

- **5.1. Introduction to Neural Networks**
  - **Inspired by the human brain:** Loosely mimics biological neurons and their connections.
  - Neurons (Perceptrons): Basic processing units that receive inputs, apply weights, sum them, and pass through an activation function. For a single neuron:
    z=j=1ΣNwjxj+boutput=A(z)

    where A is the activation function.
  - **Layers:** Input (receives data), Hidden (performs computations), Output (produces final prediction).
- **5.2. Activation Functions**
  - **Purpose:** Introduce non-linearity to the network, allowing it to learn complex, non-linear relationships in data.
  - **Common types:**
    - ReLU (Rectified Linear Unit): Most common for hidden layers, outputs input if positive, 0 otherwise.
      A(z)=max(0,z)
    - Sigmoid: Used in output layers for binary classification (0 to 1).
      A(z)=1+e−z1
    - Tanh: Outputs values between -1 and 1.
      A(z)=ez+e−zez−e−z
- **5.3. Forward Propagation and Backpropagation (Conceptual)**
  - **Forward Propagation:** Input data moves through the network, layer by layer, calculating outputs until a final prediction is made.
  - **Backpropagation:** The error between the prediction and actual label is calculated. This

error is then propagated backward through the network, using the chain rule of calculus, to compute the gradients of the loss with respect to each weight and bias, which are then used to update parameters via gradient descent.

- **5.4. Deep Neural Networks**
  - **Networks with multiple hidden layers:** "Deep" refers to the presence of many hidden layers.
  - **Benefits:** Can learn more abstract, hierarchical, and complex features from data, leading to better performance on intricate tasks.

6. Overfitting and Regularization

- **6.1. Underfitting vs. Overfitting**
  - **Underfitting:** Model is too simple, fails to capture the underlying patterns in the data (high bias), leading to poor performance on both training and test data.
  - **Overfitting:** Model learns the training data *too* well, memorizing noise and specific examples, resulting in excellent training performance but poor generalization to new, unseen data (high variance).
- **6.2. Strategies to Combat Overfitting**
  - **More Data:** Increasing the size and diversity of the training dataset is the most effective solution.
  - **Feature Selection:** Choosing only the most relevant features to prevent the model from learning from noise.
  - **Early Stopping:** Monitoring validation loss during training and stopping when it starts to increase (indicating overfitting).
  - **Regularization:** Techniques that add a penalty term to the loss function, discouraging large model weights and reducing complexity.
    - L1 Regularization (Lasso): Adds penalty proportional to the absolute value of weights ($\Sigma|\theta i|$).
      $JL1(\theta)=J(\theta)+\lambda i\Sigma|\theta i|$

      Promotes sparsity (some weights become exactly zero), useful for feature selection.
    - L2 Regularization (Ridge / Weight Decay): Adds penalty proportional to the square of weights ($\Sigma\theta i2$).
      $JL2(\theta)=J(\theta)+\lambda i\Sigma\theta i2$

      Shrinks weights towards zero, reducing their impact but rarely making them exactly zero.
    - λ (lambda) is the regularization strength hyperparameter.
  - **Dropout:** A technique used in neural networks where randomly selected neurons and their connections are temporarily ignored during training, forcing the network to learn more robust features.

7. Data Preprocessing and Feature Engineering

- **7.1. Importance of Data Quality:** The performance of a machine learning model is heavily dependent on the quality and relevance of the input data ("Garbage in, garbage out").
- **7.2. Handling Missing Values:** Strategies to deal with incomplete data.
  - **Imputation (mean, median, mode):** Filling in missing values with statistical measures.
  - **Deletion:** Removing rows or columns with missing values (if few).
- **7.3. Handling Outliers:** Dealing with extreme data points that can skew model training.
  - **Removal, capping, transformation:** Methods to mitigate their impact.
- **7.4. Feature Scaling:** Transforming numerical features to a similar scale.
  - Standardization (Z-score normalization): Rescales data to have a mean of 0 and a standard deviation of 1. For a feature x:
    $x_{scaled} = \sigma x - \mu$

    where μ is the mean and σ is the standard deviation.
  - Normalization (Min-Max scaling): Rescales data to a fixed range, typically 0 to 1.
    $x_{scaled} = x_{max} - x_{min} x - x_{min}$
  - **Importance:** Prevents features with larger numerical ranges from disproportionately influencing the model and helps optimization algorithms converge faster.
- **7.5. Encoding Categorical Features:** Converting non-numerical, categorical data into a numerical format that ML models can understand.
  - **One-Hot Encoding:** Creates new binary (0 or 1) columns for each category, where only one is "hot" (1) for a given observation. E.g., "Red", "Green", "Blue" becomes [1,0,0], [0,1,0], [0,0,1].
- **7.6. Feature Crosses:** Creating new features by multiplying or combining existing features to represent interactions.
  - **Concept:** Allows the model to learn non-linear relationships that individual features might not capture (e.g., combining latitude and longitude for location insight).
  - **Enables the model to learn complex relationships:** For example, (FeatureA AND FeatureB) or FeatureA * FeatureB.

8. Training Best Practices and Data Splitting

- **8.1. Dataset Splitting**
  - **Training Set:** The largest portion of the data, used for the model to learn patterns and adjust its weights.
  - **Validation Set:** A separate portion used *during* training to evaluate the model's performance on unseen data and tune hyperparameters, preventing overfitting.
  - **Test Set:** A completely held-out portion of data used *only once* at the very end to provide an unbiased evaluation of the final model's generalization ability.
  - **Common splits:** Typical divisions like 70% training, 15% validation, 15% testing, or 80% training, 10% validation, 10% testing.
- **8.2. Hyperparameter Tuning**
  - **Concept:** Finding the optimal values for parameters that are *not* learned by the model during training (e.g., learning rate, number of layers, batch size, regularization strength,

etc.).
    - **Techniques:**
        - **Grid Search:** Exhaustively trying all combinations of a predefined set of hyperparameter values.
        - **Random Search:** Randomly sampling hyperparameter combinations, often more efficient than grid search for high-dimensional hyperparameter spaces.
- **8.3. Batching**
    - **Stochastic Gradient Descent (SGD):** Model weights are updated after processing *each individual training example*. This leads to noisy but frequent updates.
    - **Batch Gradient Descent:** Weights are updated *after* processing the *entire training dataset*. Provides stable updates but can be slow for large datasets.
    - Mini-batch Gradient Descent: Weights are updated after processing small, fixed-size subsets (mini-batches) of the training data. This is the most common and efficient approach, balancing stability and speed. For a mini-batch of size B: θnew=θold−α·B1i∈batchΣ∇L(y^(i),y(i))

9. Introduction to TensorFlow (Google's Primary ML Library)

- **9.1. TensorFlow Core Concepts**
    - **Tensors:** The fundamental data structure in TensorFlow, representing multi-dimensional arrays (similar to NumPy arrays).
    - **Operations:** Mathematical computations performed on tensors (e.g., tf.add(), tf.matmul()).
    - **Graphs (older TF versions):** Computation was represented as a static dataflow graph; required tf.Session to run.
    - **Eager Execution (modern TF):** Operations run immediately, like standard Python, making debugging easier and code more intuitive.
- **9.2. Building Simple Models with Keras (High-Level API in TensorFlow)**
    - **Sequential API for simple, layer-by-layer models:** A straightforward way to build neural networks by stacking layers in a linear fashion.
    - Defining layers (e.g., tf.keras.layers.Dense): Specifying the type and number of neurons in each layer. A Dense layer computes: output=A(xW+b)

      where x is the input, W is the weight matrix, b is the bias vector, and A is the activation function.
    - **Compiling models (model.compile()):** Configuring the training process by specifying the optimization algorithm (e.g., 'adam'), loss function (e.g., 'mse', 'binary_crossentropy'), and evaluation metrics (e.g., 'accuracy').
    - **Training models (model.fit()):** The method to execute the training loop over the dataset.
    - **Making predictions (model.predict()):** Using the trained model to generate outputs for new inputs.

10. Beyond the Basics (Brief Overview / Next Steps)

- **10.1. Convolutional Neural Networks (CNNs)**
  - **Purpose:** Specialized neural networks highly effective for processing grid-like data, particularly image data.
  - **Core Components:**
    - **Convolutional layers:** Learn spatial hierarchies of features by applying filters (kernels) that slide over the input. The output of a convolution is a feature map.
    - **Pooling layers (e.g., Max Pooling):** Reduce the spatial dimensions of the feature maps, making them more robust to small shifts and distortions, and reducing computation.
- **10.2. Recurrent Neural Networks (RNNs)**
  - **Purpose:** Designed for sequential data where the order of information matters (e.g., text, time series, speech). They have internal memory.
  - **Core Idea:** An output from a previous step is fed as input to the current step.
  - **Common types:**
    - **LSTMs (Long Short-Term Memory):** Address the vanishing gradient problem in standard RNNs, allowing them to learn long-term dependencies through specialized "gates" (input, forget, output gates).
    - **GRUs (Gated Recurrent Units):** A simpler variant of LSTMs with fewer gates.
- **10.3. Embeddings**
  - **Concept:** Representing discrete data (like words, categories, or user IDs) as dense, low-dimensional vectors in a continuous space. Words with similar meanings are mapped to nearby points in the embedding space.
  - **Benefits:** Captures semantic relationships and allows models to handle categorical data more effectively than one-hot encoding for large vocabularies.
- **10.4. Ethics and Fairness in ML**
  - **Bias in data and models:** Discussing how biases present in training data (e.g., underrepresentation of certain groups) can lead to unfair or discriminatory model predictions.
  - **Interpretability and explainability:** Understanding why a model makes certain predictions (e.g., LIME, SHAP), crucial for trust, accountability, and debugging.
- **10.5. Introduction to Production ML (e.g., TensorFlow Extended - TFX)**
  - **Concept of MLOps (Machine Learning Operations):** The set of practices for deploying and maintaining ML systems in production reliably and efficiently. Encompasses continuous integration, delivery, and monitoring for ML.
  - **Pipelines for continuous training and deployment:** Automating the ML workflow from data ingestion and validation to model training, evaluation, serving, and monitoring, ensuring models stay up-to-date and performant in real-world scenarios.