**Machine Learning:**

The exam (questions are random so I do not know about other experiences) was heavy on Machine Learning topics. There were like 15 questions about this. Mostly related to:

* Overfit models
* How to deal with high RMSE, for example: make your model more complex and robust.
* Neurons, features, epoch, labels.
* D
* Dialogflow
* Cloud AutoML to label some logos within an image.
* Cloud Vision API, Speech to Text, etc.
* Tensorflow models in C++
* Cloud TPU and GPU

I recommend learning all the most important vocabulary for that, like labels, epoch, neurons, hidden layers, bias, weight, learn when to implement a linear regression model instead of classification or clustering.

* Differences between training and test data.
* Overfitting and underfitting, such as why they can happen, how to prevent them.
* Good understanding of ML types, including supervised learning, unsupervised learning, reinforcement learning, although I saw no questions on reinforcement.
* Not much on Tensorflow, but you should know the basic concepts.
* Good understanding of how neural networks (NN) work. There were questions on wide NN, deep NN, and both wide and deep NN.
* Regularization parameters, such as L1 and L2, including a couple of scenario-based questions of when to use each type.

GCP ML services

* Good understanding of each service, especially Natural Language API, such as sentiment and entity analysis
* A couple of questions on when it is beneficial for a customer to use ML services
* AI platform, including how it works and online versus batch predictions

Google provides a great ML service called **AutoML** to **quickly build models for you**. AutoML Vision is one of its products which you can start with a training set of as little as a dozen photo samples and AutoML takes care of the rest.

In machine learning, features are two types of features: **Categorical & Continuous.**  
**Categorical features** are features with **finite values**. For example: Country, education level and marital status.  
**Continuous features** are features with **numeric values** in a continuous range. For example, Income, latitude & longitude, and time.

For zip code, while it’s represented as numeric values, it’s considered categorical because it represents regions, which means, it marks each region with a number.

AutoML is used to train models and do damage detection.

Auto Vision is used is a pre trained model used to detect objects in images

**Cloud ML Engine** mainly does two things:  
- Enables you to train machine learning models at scale by running TensorFlow training applications in the cloud.  
- Hosts those trained models for you in the cloud so that you can use them to get predictions about new data.

Software supported by MLEngine is Tensorflow.

Cloud ML Engine is the service used to deploy your machine learning models. **C**loud ML Engine offers training and prediction services, which can be used together or individually.

**Cloud ML Engine is used to deploy models. It does not help to build the models.**

**gcloud ml-engine local train – run** a Cloud ML Engine training job locally  
This command runs the specified module in an environment similar to that of a live Cloud ML Engine Training Job.  
This is especially **useful in the case of testing distributed models**, as it allows you to validate that you are properly interacting with the Cloud ML Engine cluster configuration.  
Reference: <https://cloud.google.com/sdk/gcloud/reference/ml-engine/local/train>

**AI Products: -** <https://cloud.google.com/products/ai/>

<https://cloud.google.com/natural-language/><https://cloud.google.com/ml-engine/><https://cloud.google.com/vision><https://cloud.google.com/speech-to-text/>

speech-to-Text is for audio to text conversion.

Cloud **Speech-to-Text** is a service to **generate captions from videos by detecting the speaker’s language and speech**. Google Cloud Speech-to-Text enables developers to convert audio to text by applying powerful neural network models in an easy-to-use API. The API recognizes 120 languages and variants to support your global user base. You can enable voice command-and-control, transcribe audio from call centers, and more. It can process real-time streaming or prerecorded audio, using Google’s machine learning technology.

**Dialogflow** is used to do voice analytics on human computer interaction.

Refer: <https://cloud.google.com/dialogflow/docs/basics>

Cloud Video Intelligence: - <https://cloud.google.com/video-intelligence/> Cloud Video Intelligence can be used to perform content moderation.

Google **Cloud Video Intelligence** makes **videos searchable, and discoverable, by extracting metadata with an easy-to-use REST API.** You can now search every moment of every video file in your catalog. It quickly annotates videos stored in Google Cloud Storage, and helps you identify key entities (nouns) within your video; and when they occur within the video. Separate signals from noise, by retrieving relevant information within the entire video, shot-by-shot, -or per frame.  
Identify when inappropriate content is being shown in a given video. You can instantly conduct content moderation across petabytes of data and more quickly and efficiently filter your content or user-generated content.

Natural Language is for text analysis

**Cloud natural language service** is used **to derive insights from unstructured text, revealing the meaning of the documents and categorize articles**. It won’t help in extracting captions from videos.

Vision is for image analysis.

**Cloud Vision** offers **both pretrained models via an API and the ability to build custom models using AutoML Vision to provide flexibility depending on your use case**.  
Cloud Vision API enables developers to understand the content of an image by encapsulating powerful machine learning models in an easy-to-use REST API. It quickly classifies images into thousands of categories (such as, “sailboat”), detects individual objects and faces within images, and reads printed words contained within images. You can build metadata on your image catalog, moderate offensive content, or enable new marketing scenarios through image sentiment analysis.

Cloud Vision API enables you to derive insight from your images with our powerful pretrained API models or easily train custom vision models with AutoML Vision Beta. The API quickly classifies images into thousands of categories (such as “sailboat” or “Eiffel Tower”), detects individual objects and faces within images, and finds and reads printed words contained within images. AutoML Vision lets you build and train custom ML models with minimal ML expertise to meet domain-specific business needs.

Google Cloud provides a machine learning service called AutoML to quickly build models for you. AutoML Vision is one of its products which you can start with a training set as little as a dozen photo samples and AutoML takes care of the rest.  
While iterating on your model, if the model’s quality levels are not up to expectations, you can go back to earlier steps to improve the quality:  
AutoML Vision allows you to sort the images by how “confused” the model is, by the true label and its predicted label. Look through these images and make sure they’re labeled correctly.  
Consider adding more images to any labels with low quality.  
You may need to add different types of images (e.g. wider angle, higher or lower resolution, different points of view).  
Consider removing labels altogether if you don’t have enough training images.  
Remember that machines can’t read your label name; it’s just a random string of letters to them. If you have one label that says “door” and another that says “door\_with\_knob” the machine has no way of figuring out the nuance other than the images you provide it.  
Augment your data with more examples of true positives and negatives. Especially important examples are the ones that are close to the decision boundary (i.e. likely to produce confusion but still correctly labeled).  
Specify your own TRAIN, TEST, VALIDATION split. The tool randomly assigns images, but near-duplicates may end up in TRAIN and VALIDATION which could lead to overfitting and then poor performance on the TEST set.  
Once you’ve made changes, train and evaluate a new model until you reach a high enough quality level.

**AutoML Vision API** is a service to recognize and **derive insights from images by either using pre-trained models or training a custom model based on a set of photographic.**

**Machine Learning Engine** is a **managed service that allows developers and scientists to build their own models and run them in production**. This means you have to build your own model to generate text from videos that needs much effort and experience to build such a model. So, it’s not a practical solution for this scenario.

**Google Kubernetes Engine** is a **managed, production-ready environment for deploying containerized applications**. It brings out the latest innovations in developer productivity, resource efficiency, automated operations, and open-source flexibility to accelerate your time to market.

The **CUSTOM tier** is not a set tier, but rather **enables you to use your own cluster specification**. When you use this tier, set values to configure your processing cluster according to these guidelines:  
You **must set** TrainingInput.masterType to specify the **type of machine to use for your master node**.  
You may set TrainingInput.workerCount to specify the number of workers to use.  
You may set TrainingInput.parameterServerCount to specify the number of parameter servers to use.  
You can specify the type of machine for the master node, but **you can’t specify more than one master node.**

<https://cloud.google.com/ml-engine/docs/tensorflow/machine-types>

<https://cloud.google.com/ml-engine/docs/training-overview#job_configuration_parameters>

**Dropout Regularization**: It is a regularization method **to remove a random selection of the fixed number of units in a neural network layer.** The more units dropped out, the stronger the regularization.

**Precision** is the formula to check **how accurate the model is when most of the outputs are positive. In other words, if most of the output is yes.**

**Recall**: It is the formula to check **how accurate the model is when most of the outputs are negatives. In other words, if most of the output is no.**

**Gradient Descent:** It is an optimization **algorithm to find the minimal value of a function.** Gradient descent is **used to find the minimal RMSE or cost function.**

**Feature Engineering** is the process of deciding which data is important for the model.

**Hyperparameters** are the **variables that govern the training process itself**.

For example, p**art of setting up a deep neural network is deciding how many hidden layers of nodes to use between the input layer and the output layer, and how many nodes each layer should use.** These variables are not directly related to the training data, but these are configuration variables. Note that **parameters change during a training job, while hyperparameters are usually constant during a job.**

If model parameters are variables that get adjusted by training with existing data, your hyperparameters are the variables about the training process itself. **Weights and biases are variables that get adjusted during the training process, so they are not hyperparameters.**  
Reference: <https://cloud.google.com/ml-engine/docs/hyperparameter-tuning-overview>

This is a case of underfitting – not overfitting (for over fitting the model will have extremely low training error but a high testing error) – so we need to make the model more complex

**Overfitting**: 100% accuracy is an indicator that the validation data may have somehow gotten mixed in with the training data. You will need new validation data to generate recognizable errors.

Overfitting results when a **model performs well on the training set, generating only a small error, but struggles with new or unknown data**. In other words, the **model overfits itself to the data. Instead of training a model to pick out general features in a given type of data, an overtrained model learns only how to pick out specific features found in the training set.**

Overfitting happens when a model p**erforms well on a training set**, **generating only a small error while giving the wrong output for the test set.** This happens because the model is **only picking up specific features input found in the training set instead of picking out the general features** of the given training set.  
To **solve overfitting**, the following would help in improving the model’s quality:  
- **Increase the number of examples**, the more data a model is trained with, the more use cases the model can be training on and better improves its predictions.  
- **Tune hyperparameters** are related to the number and size of hidden layers (for neural networks), and regularization, which **means using techniques to make your model simpler such as using a dropout method to remove neuron networks or adding “penalty” parameters to the cost function.**  
- **Remove features by removing irrelevant features**. Feature engineering is a wide subject and **feature selection is a critical part of building and training a model**. Some algorithms have built-in feature selection, but in some cases, data scientists need to cherry-pick or manually select or **remove features for debugging and finding the best model output.**

From the brief explanation, to solve the overfitting problem in the scenario, you need to:  
Increase the training set.  
Decrease features parameters.  
Increase regularization.

Overfitting happens when a model performs well on a training set, generating only a small error while giving the wrong output for the test set. This happens because the model is only picking up specific features input found in the training set instead of picking out the general features of the given training set.  
The opposite of overfitting is underfitting. Underfitting occurs when there is still room for improvement in the test data. This can happen for a number of reasons: If the model is not powerful enough, it is over-regularized or has simply not been trained long enough. This means the network has not learned the relevant patterns in the training data.  
To solve overfitting, the following would help improve the model’s quality:  
Increase the number of examples, the more data a model is trained with, the more use cases the model can be trained on and better improves its predictions.  
Tune hyperparameters are related to the number and size of hidden layers (for neural networks), and regularization, which means using techniques to make your model simpler such as using a dropout method to remove neuron networks or adding “penalty” parameters to the cost function.  
Remove features by removing irrelevant features. Feature engineering is a wide subject and feature selection is a critical part of building and training a model. Some algorithms have built-in feature selection, but in some cases, data scientists need to cherry-pick or manually select or remove features for debugging and finding the best model output.

From the brief explanation, to solve the overfitting problem in the scenario, you need to:  
 – Increase the training set.  
 – Decrease features parameters.

Selecting and crafting the right set of **feature columns** is key to learning an effective model.  
**Bucketization** is a process of **dividing the entire range of a continuous feature into a set of consecutive bins/buckets, and then converting the original numerical feature into a bucket ID (as a categorical feature) depending on which bucket that value falls into.**  
Using each base feature column separately may not be enough to explain the data. To learn the differences between different feature combinations, we can add crossed feature columns to the model.  
Reference:  
<https://www.tensorflow.org/tutorials/wide#selecting_and_engineering_features_for_the_model>

The columns can be grouped into two types categorical and continuous columns:  
A column is called **categorical if its value can only be one of the categories in a finite set**. For example, the **native country** of a person (U.S., India, Japan, etc.) or the **education level** (high school, college, etc.) are categorical columns.  
A column is called **continuous if its value can be any numerical value in a continuous range.** For example, the **capital gain of a person** (e.g. $14,084) is a continuous column.  
**Year of birth and income are continuous columns. Country is a categorical column.**  
You could use bucketization to turn year of birth and/or income into categorical features, but the raw columns are continuous.  
Reference: <https://www.tensorflow.org/tutorials/wide#reading_the_census_data>

If you know the set of all possible feature values of a column and there are only a few of them, you can use categorical\_column\_with\_vocabulary\_list. Each key in the list will get assigned an auto-incremental ID starting from 0.  
What if we don’t know the set of possible values in advance? Not a problem. We can use categorical\_column\_with\_hash\_bucket instead. What will happen is that each possible value in the feature column occupation will be hashed to an integer ID as we encounter them in training.  
Reference: <https://www.tensorflow.org/tutorials/wide>

Google has built the **Tensor Processing Unit (TPU)** to make it possible for data scientists to achieve business and research breakthroughs ranging from network security to medical diagnoses. **Cloud TPU is the custom-designed machine learning ASIC that powers Google products like Translate, Photos, Search, Assistant, and Gmail.**

Refer GCP documentation – Understanding Neural Networks: - <https://cloud.google.com/blog/products/gcp/understanding-neural-networks-with-tensorflow-playground>

With **more neurons in a single hidden layer**, you **can capture more features**. And **having more hidden layers means more complex constructs that you can extract from the dataset.**

The **data is unlabeled**, and the **unsupervised** learning technique of Clustering can be applied to categorize the data.

In **unsupervised learning**, the goal is to identify meaningful patterns in the data. To accomplish this, the machine must learn from an unlabeled data set. In other words, the model has no hints how to categorize each piece of data and must infer its own rules for doing so. (Clustering).

In **supervised machine learning**, you feed the features and their corresponding labels into an algorithm in a process called training. During training, the algorithm gradually determines the relationship between features and their corresponding labels. This relationship is called the model. Often in machine learning, the model is complex. (Regression and classification).

**Association rules** are to identify relationships.

**Wide learning** model is good for **memorization**

**Deep learning** model is **generalization**.

Both Wide and Deep learning models can **help build a good recommendation engine**. Wide Deep learning together:- <https://ai.googleblog.com/2016/06/wide-deep-learning-better-together-with.html>

Can we teach computers to learn like humans do, by combining the power of memorization and generalization? It’s not an easy question to answer, but by jointly training a wide linear model (for memorization) alongside a deep neural network (for generalization), one can combine the strengths of both to bring us one step closer. At Google, we call it Wide & Deep Learning. It’s useful for generic large-scale regression and classification problems with sparse inputs (categorical features with a large number of possible feature values), such as **recommender systems, search, and ranking problems.**

**Wide and Deep Learning:** The human brain is a sophisticated learning machine, forming rules by memorizing everyday events (“sparrows can fly” and “pigeons can fly”) and generalizing those learnings to apply to things we haven’t seen before (“animals with wings can fly”). Perhaps more powerfully, memorization also allows us to further refine our generalized rules with exceptions (“penguins can’t fly”). As we were exploring how to advance machine intelligence, we asked ourselves the question—can we teach computers to learn like humans do, by combining the power of memorization and generalization?  
It’s not an easy question to answer, but by jointly training a wide linear model (for memorization) alongside a deep neural network (for generalization), one can combine the strengths of to bring us one step closer. At Google, we call it Wide & Deep Learning. It’s useful for **generic large-scale regression and classification problems with sparse inputs** (categorical features with a large number of possible feature values), such as **recommender systems, search, and ranking problems**.

**Regression** is the **supervised learning** task for modeling and **predicting continuous, numeric variables.** Examples include predicting real-estate prices, stock price movements, or student test scores.

**Classification** is the **supervised learning** task for modeling and **predicting categorical variables**. Examples include predicting employee churn, email spam, financial fraud, or student letter grades.

**Clustering** is an **unsupervised learning** task for **finding natural groupings of observations** (i.e., clusters) **based on the inherent structure** within your dataset.  
Examples include customer segmentation, grouping related items in e-commerce, and social network analysis.  
Reference: <https://elitedatascience.com/machine-learning-algorithms>

**Traditional** machine learning relies on shallow nets, **composed of one input and one output layer, and at most one hidden layer in between**. **More than three layers (including input and output) qualifies as “deep” learning**. So deep is a **strictly defined**, technical term that **means more than one hidden layer**.

In deep-learning networks, **each layer of nodes trains on a distinct set of features based on the previous layers output**. The **further you advance** into the neural net, the **more complex the features your nodes can recognize**, since they aggregate and recombine features from the previous layer.

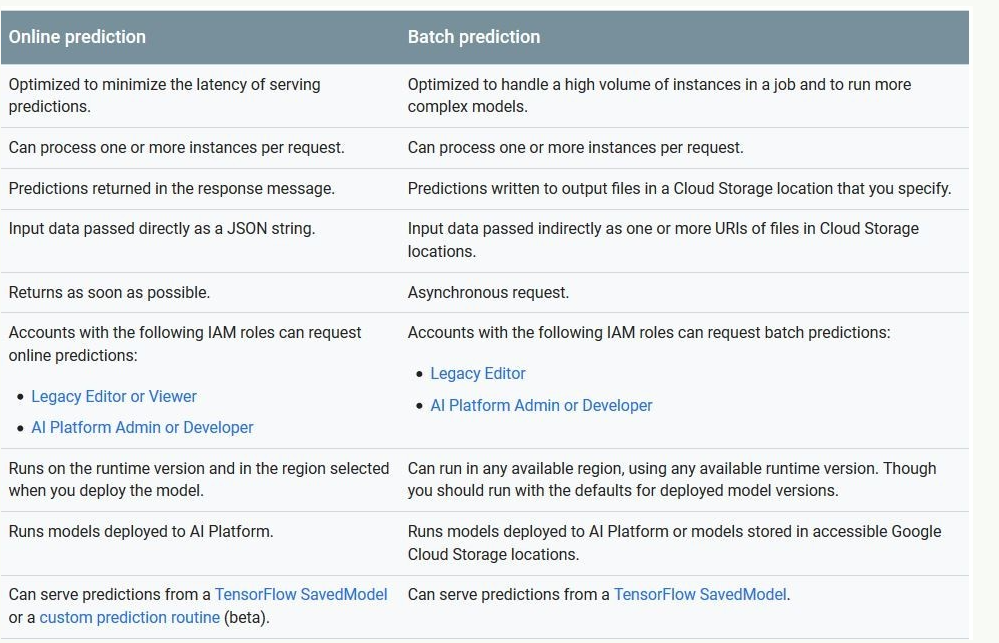
A neural network is a simple mechanism thats implemented with basic math. The only difference between the traditional programming model and a neural network is that you let the computer determine the parameters (weights and bias) by learning from training datasets.  
Reference: <https://cloud.google.com/blog/big-data/2016/07/understanding-neural-networks-with-tensorflow-playground>

A neural network with only **one hidden layer would be unable to automatically recognize high-level features of faces, such as eyes, because it wouldn’t be able to “build” these features using previous hidden layers that detect low-level features, such as lines**.

Feature engineering is difficult to perform on raw image data.  
**K-means Clustering** is an **unsupervised learning** method used to **categorize unlabeled data.**  
Reference: <https://deeplearning4j.org/neuralnet-overview>

AI Platform provides two ways to get predictions from trained models: **online prediction (sometimes called HTTP prediction), and batch prediction.** In both cases, you **pass input data to a cloud-hosted machine-learning model and get inferences for each data instance.**  
<https://cloud.google.com/ai-platform/prediction/docs/online-vs-batch-prediction>

**Online prediction** – **passes input as a JSON string** and returns the output as soon as possible.  
. Optimized to **minimize the latency** of serving predictions.  
. Predictions returned in the response message.  
**Batch prediction** –  
. Optimized to handle a **high volume of instances** in a job and to run more **complex models**.  
. Predictions are written to output files in a **Cloud Storage** location that you specify.



There are two problems with one-hot encoding. First, it has high dimensionality, meaning that instead of having just one value, like a continuous feature, it has many values, or dimensions. This makes computation more time-consuming, especially if a feature has a very large number of categories. The second problem is that it doesnt encode any relationships between the categories. They are completely independent from each other, so the network has no way of knowing which ones are similar to each other.  
Both of these problems can be solved by representing a categorical feature with an embedding column. The idea is that each category has a smaller vector with, let’s say, 5 values in it. But unlike a one-hot vector, the values are not usually 0. The values are weights, similar to the weights that are used for basic features in a neural network. The difference is that each category has a set of weights (5 of them in this case).  
You can think of each value in the embedding vector as a feature of the category. So, if two categories are very similar to each other, then their embedding vectors should be very similar too.  
Reference: <https://cloudacademy.com/google/introduction-to-google-cloud-machine-learning-engine-course/a-wide-and-deep-model.html>

Training the model with the whole training set is not the right approach in Machine Learning because you ought to test the model before considering it accurate enough for production. Usually, a training set is split into 70-30% sets, the first for training while the second is for testing and tuning the model’s parameters.

**BigQuery ML:**

BigQuery ML enables users to create and execute machine learning models in BigQuery using standard SQL queries. BigQuery ML democratizes machine learning by enabling SQL practitioners to build models using existing SQL tools and skills. BigQuery ML increases development speed by eliminating the need to move data.  
BigQuery ML empowers data analysts to use machine learning through existing SQL tools and skills. Analysts can use BigQuery ML to build and evaluate ML models in BigQuery. Analysts no longer need to export small amounts of data to spreadsheets or other applications, and analysts no longer need to wait for limited resources from a data science team.

<https://cloud.google.com/bigquery-ml/docs/bigqueryml-intro><https://www.youtube.com/watch?v=BanOYQVl30I>

*amount of data is relatively low and also varied. A model built using only this data wouldn't be accurate. AutoML is appropriate because it uses transfer learning based on other similar data.*