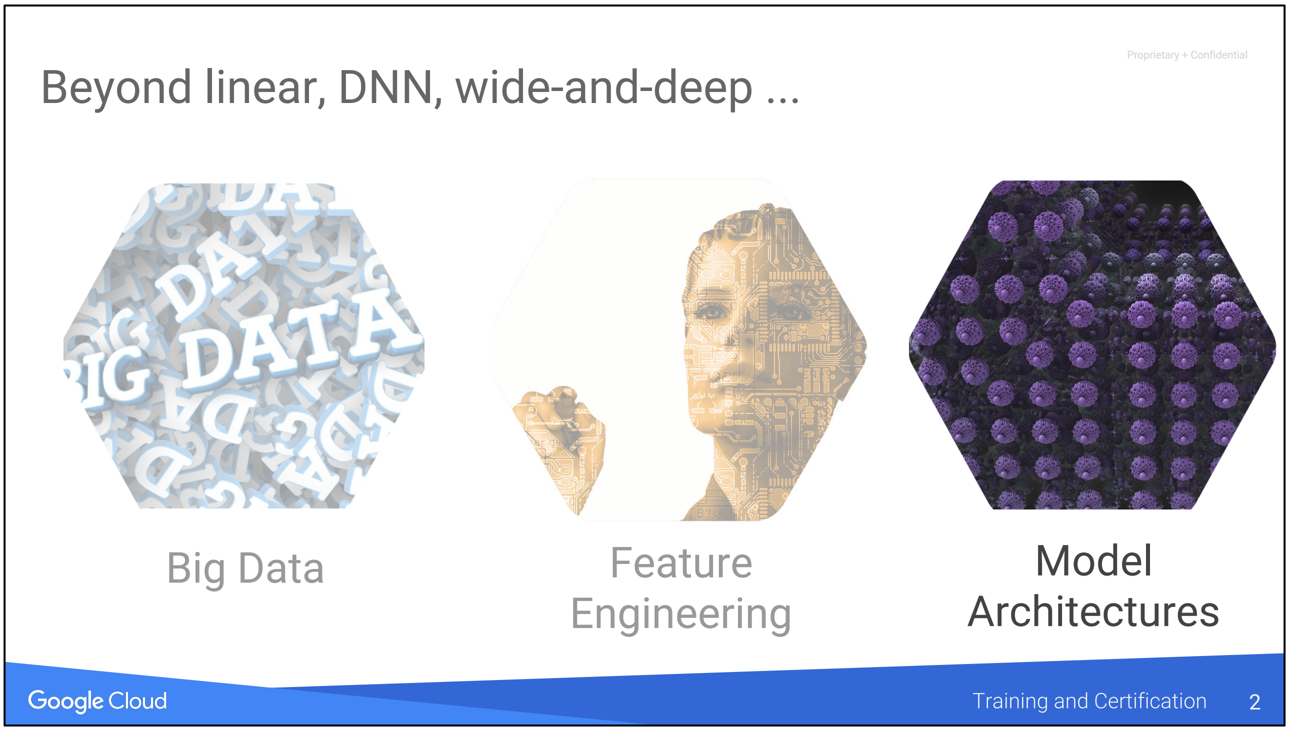
Self-study: Designing ML model architectures

Data Engineering on Google Cloud Platform

Notes:

SKIP



Beyond linear, DNN, wide-and-deep ...

Big Data

Feature Engineering

Model Architectures

Notes:

https://pixabay.com/en/large-data-dataset-word-895563/ (cc0)
https://pixabay.com/en/fractal-complexity-render-3d-1232494/ (cc0)
https://pixabay.com/en/robot-artificial-intelligence-woman-507811/ (cc0)

We briefly looked at wide-and-deep models in the previous chapter. Let's look at other model architectures.

Now that you know *how* to build ML, let's learn how to do it well in the rest of the course.
Ordered from easiest to most difficult.

Agenda

Packaged solutions + Lab

Image analysis + Demo

Embedding and sequences

Recommendation systems

Proprietary + Confidential

Packaged solutions in Datalab

What if we could:

- Take structured data and identify which columns are categorical ("wide") and which are continuous ("dense")
- Get a reasonable TensorFlow model built for us?

TRADE FLEXIBILITY FOR AGILITY

Notes:

https://pixabay.com/en/boy-idea-sad-eyes-school-thinking-1867332/ (cc0)

Lab: Structured data packaged solution

In this lab, you will learn:

- How to use the structured data packaged solution in Datalab

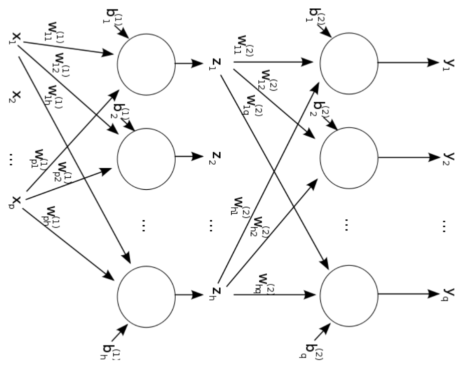
Agenda

Image analysis + Demo

Pixel value correlation has a structure

pixel_values (

)

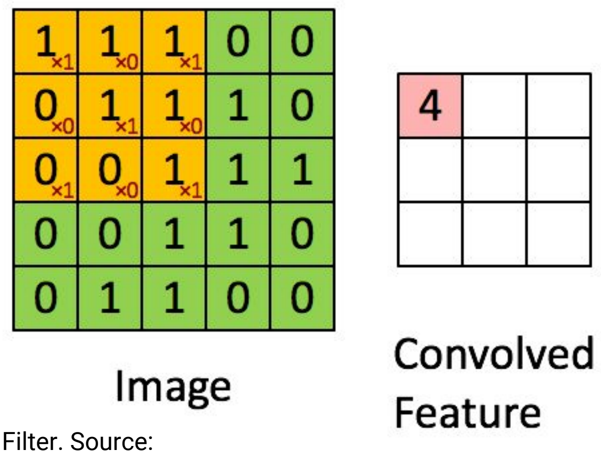


Notes:

https://pixabay.com/en/algodones-dunes-dunes-sand-dunes-1654439/ (cc0)
https://commons.wikimedia.org/wiki/File:Two_layer_ann.svg

It’s spatial contiguity; i.e., nearby pixels are likely to be similar.

Convolution is weighted averaging

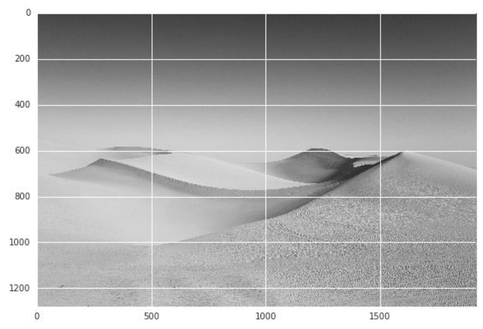


Convolution with 3×3 Filter. Source:
http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

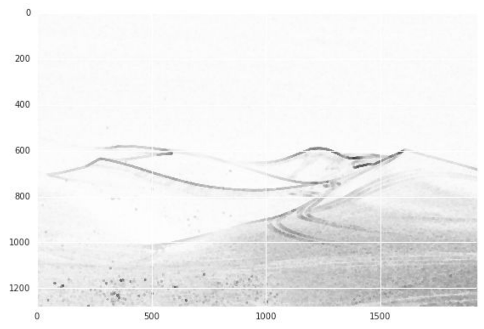
Notes:

The easiest way to understand a convolution is by thinking of it as a sliding window function applied to a matrix.

Different weights do different things



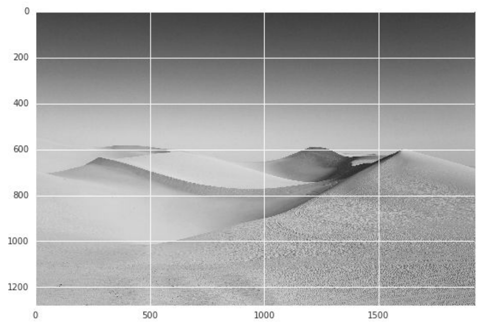
$$\boxed{\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}}$$



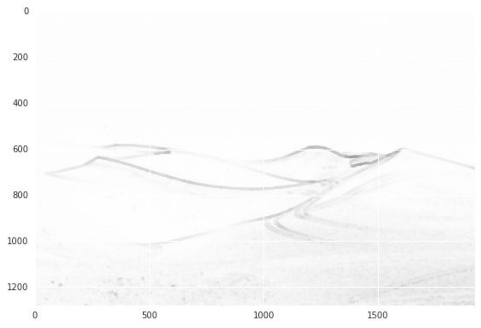
Notes:

At each pixel, take the 8 neighbors around it and apply the weights to the 9 values that you have. This is called a Sobel horizontal filter, and it finds horizontal edges in the image.

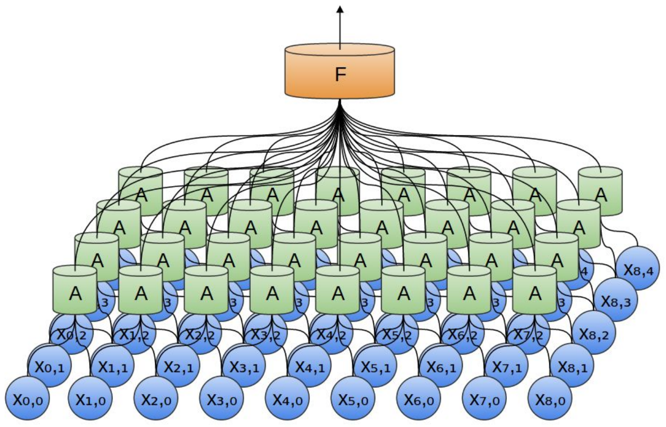
Larger windows → Larger scale features



$$\begin{bmatrix} 1 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -2 & 0 & -1 \end{bmatrix}$$

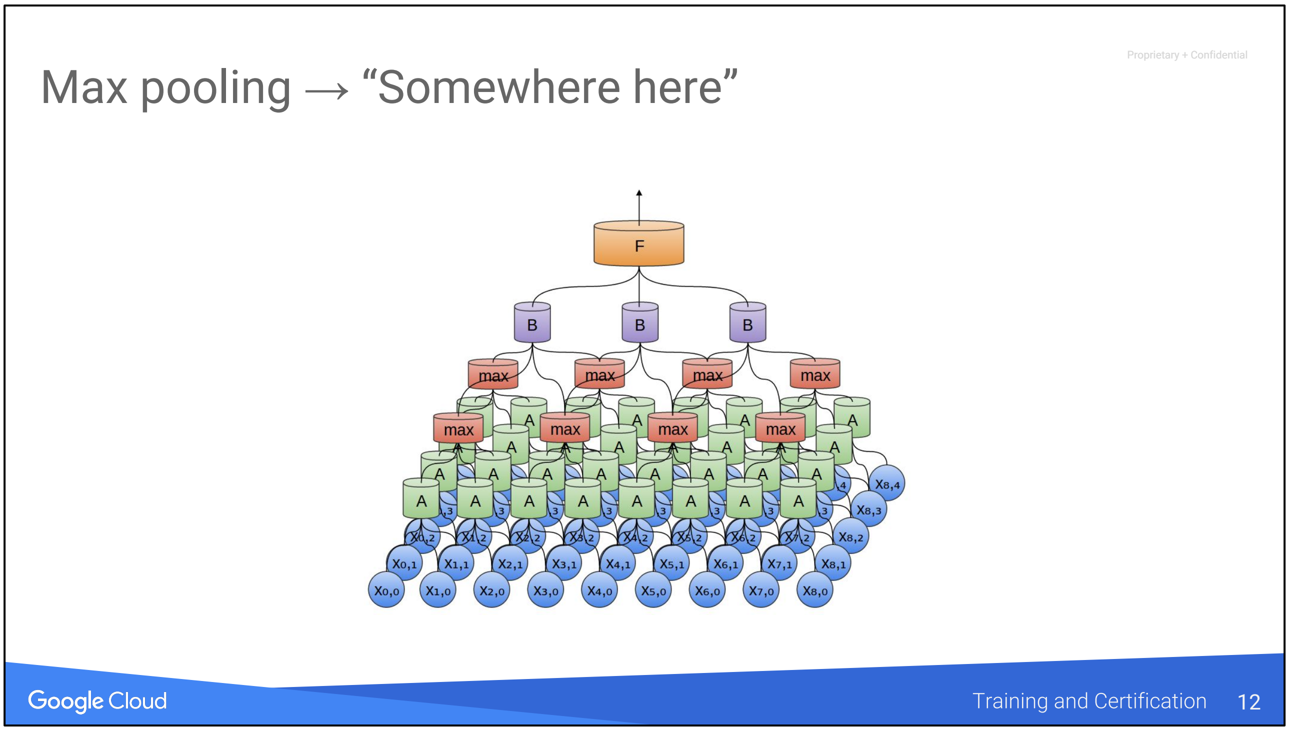


Convolution layers = Shared weights


F
A
X0,4 X1,4 X2,4 X3,4 X4,4 X5,4 X6,4 X7,4 X8,4
X0,3 X1,3 X2,3 X3,3 X4,3 X5,3 X6,3 X7,3 X8,3
X0,2 X1,2 X2,2 X3,2 X4,2 X5,2 X6,2 X7,2 X8,2
X0,1 X1,1 X2,1 X3,1 X4,1 X5,1 X6,1 X7,1 X8,1
X0,0 X1,0 X2,0 X3,0 X4,0 X5,0 X6,0 X7,0 X8,0

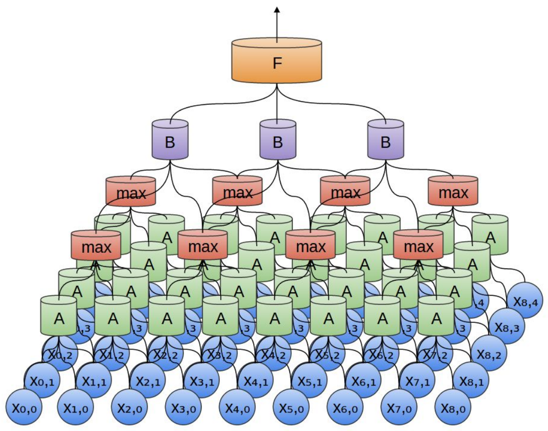
Notes:

Image from: http://colah.github.io/posts/2014-07-Conv-Nets-Modular/



Proprietary + Confidential

Max pooling → “Somewhere here”



Google Cloud

Notes:

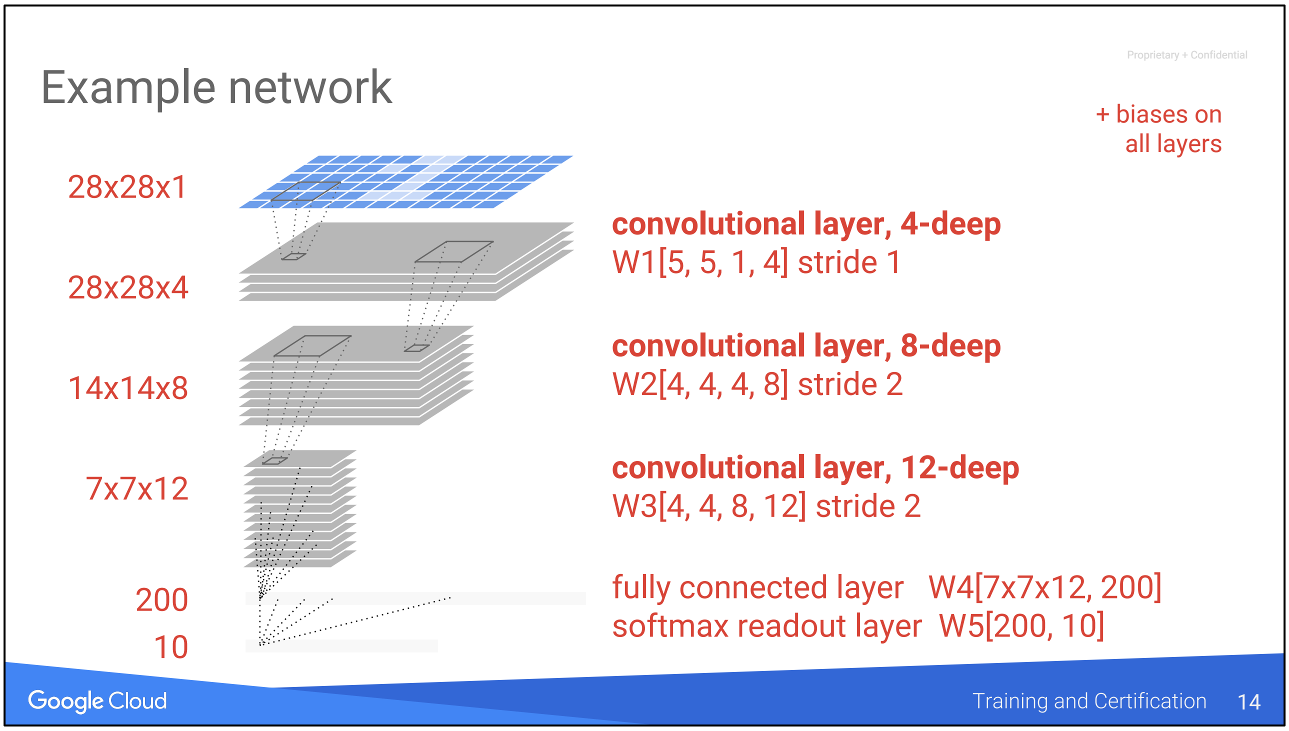
If you use convolution layers with overlapping windows and follow it by taking the maximum of these windows, you get translation invariance.

Image from: http://colah.github.io/posts/2014-07-Conv-Nets-Modular/

Convolution & max pooling in TF

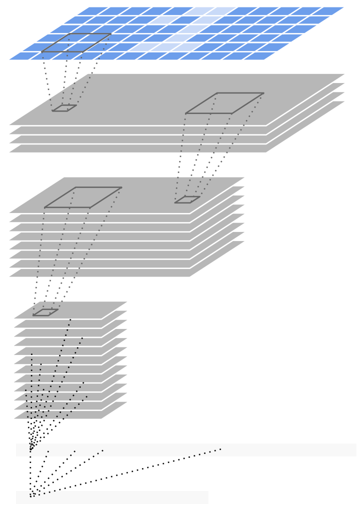
```
tf.contrib.layers.convolution2d(inputs, noutputs, kernel_size=[5,5], stride=[3,3])
```

```
tf.contrib.layers.max_pool2d(inputs, noutputs, kernel_size=[25,25], stride=[1,1])
```



Example network

+ biases on all layers



convolutional layer, 4-deep
W1[5, 5, 1, 4] stride 1

convolutional layer, 8-deep
W2[4, 4, 4, 8] stride 2

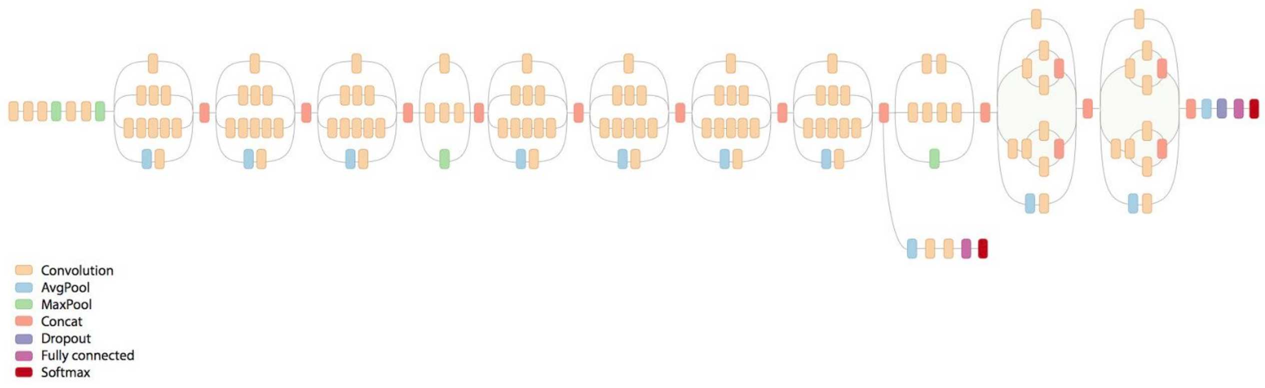
convolutional layer, 12-deep
W3[4, 4, 8, 12] stride 2

fully connected layer   W4[7x7x12, 200]
softmax readout layer W5[200, 10]

Notes:

28x28x1 is the size of the MNIST images. All images that are provided as input have to be the same size, so part of the preprocessing workflow will be to resize images.

An actual image classification network

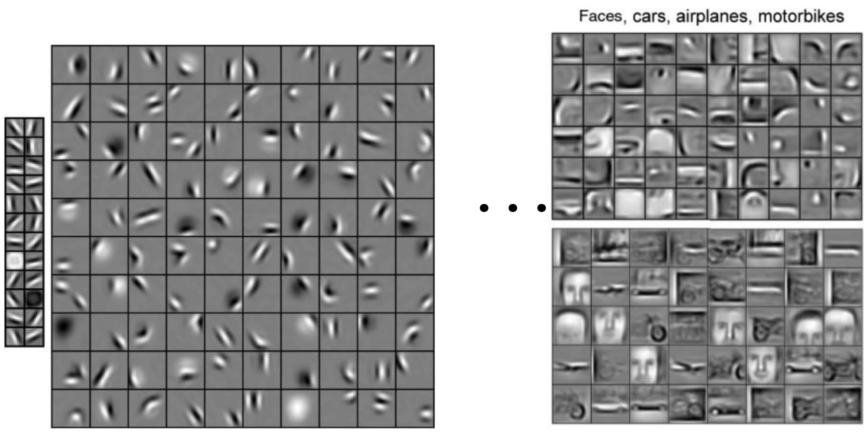


Inception v3 http://arxiv.org/abs/1512.00567

Notes:

How many examples do you need to train a monster like this?

Fortunately, the first layers are reusable

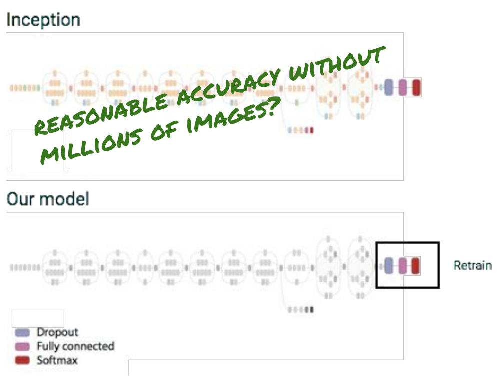


Images from http://www.cs.toronto.edu/~rgrosse/cacm2011-cdbn.pdf

Notes:

The last layers are specific to the categories of images you trained the model on, but the first few layers are very general purpose and can be reused.

Custom image classification through transfer learning—retrain last few layers

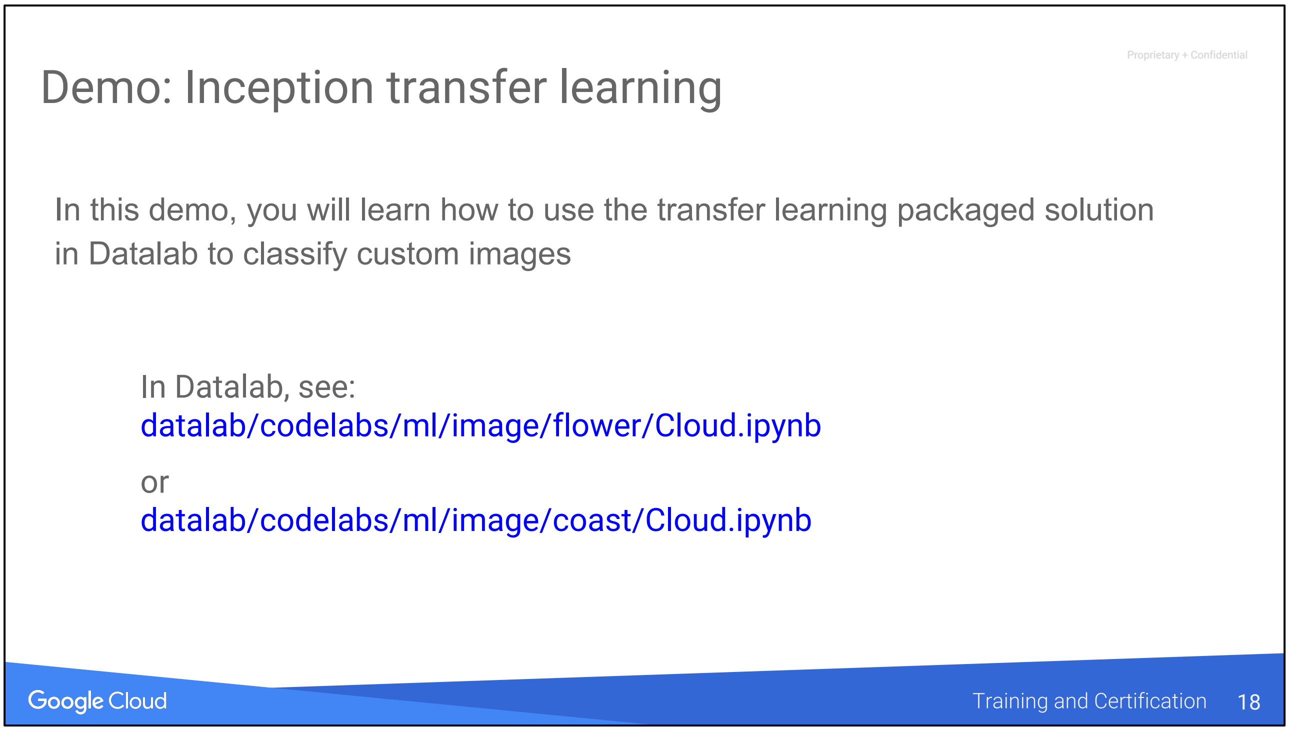


REASONABLE ACCURACY WITHOUT MILLIONS OF IMAGES?

https://cloud.google.com/blog/big-data/2016/12/how-to-train-and-classify-images-using-google-cloud-machine-learning-and-cloud-dataflow

Notes:

https://pixabay.com/en/boy-idea-sad-eyes-school-thinking-1867332/ (cc0)



Demo: Inception transfer learning

In this demo, you will learn how to use the transfer learning packaged solution in Datalab to classify custom images

In Datalab, see:
datalab/codelabs/ml/image/flower/Cloud.ipynb

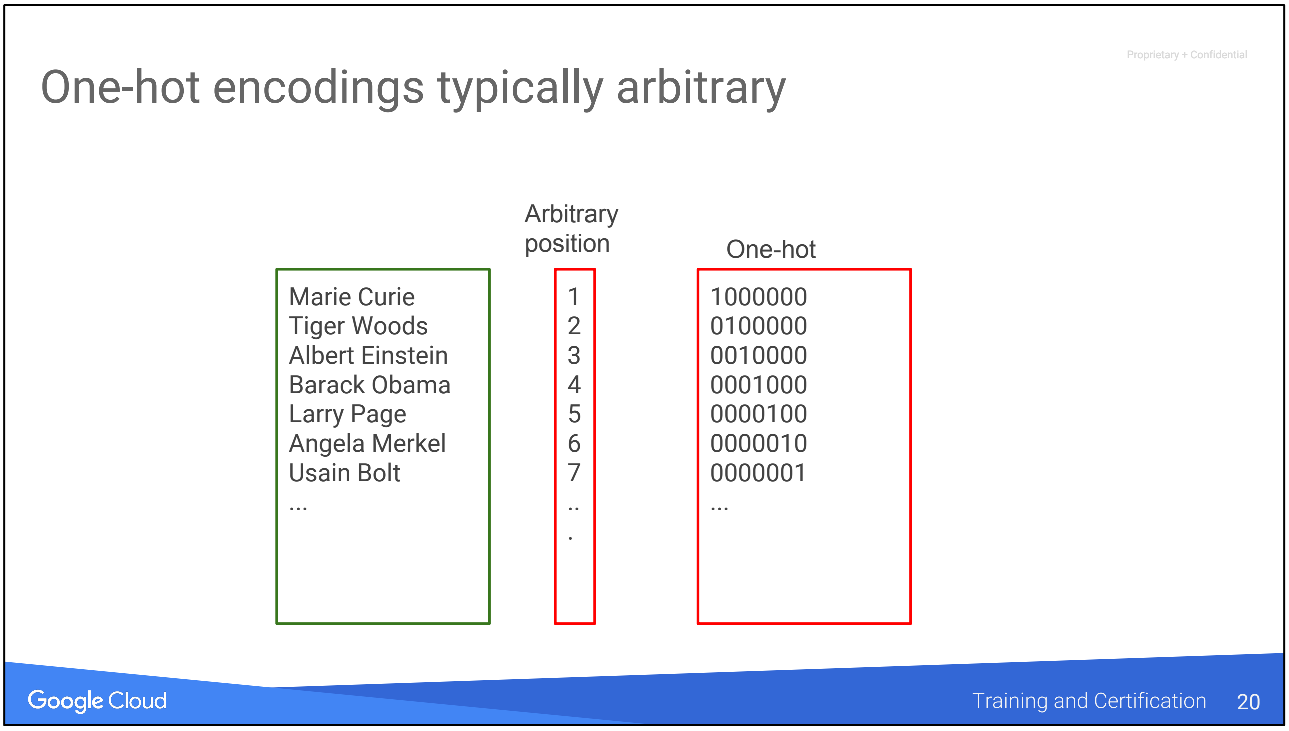
or
datalab/codelabs/ml/image/coast/Cloud.ipynb

Notes:

This comes with Datalab. The flowers is the original blog post. The “coast” is an example of applying it to a new set of images.

Agenda

Embedding and sequences



Proprietary + Confidential

One-hot encodings typically arbitrary

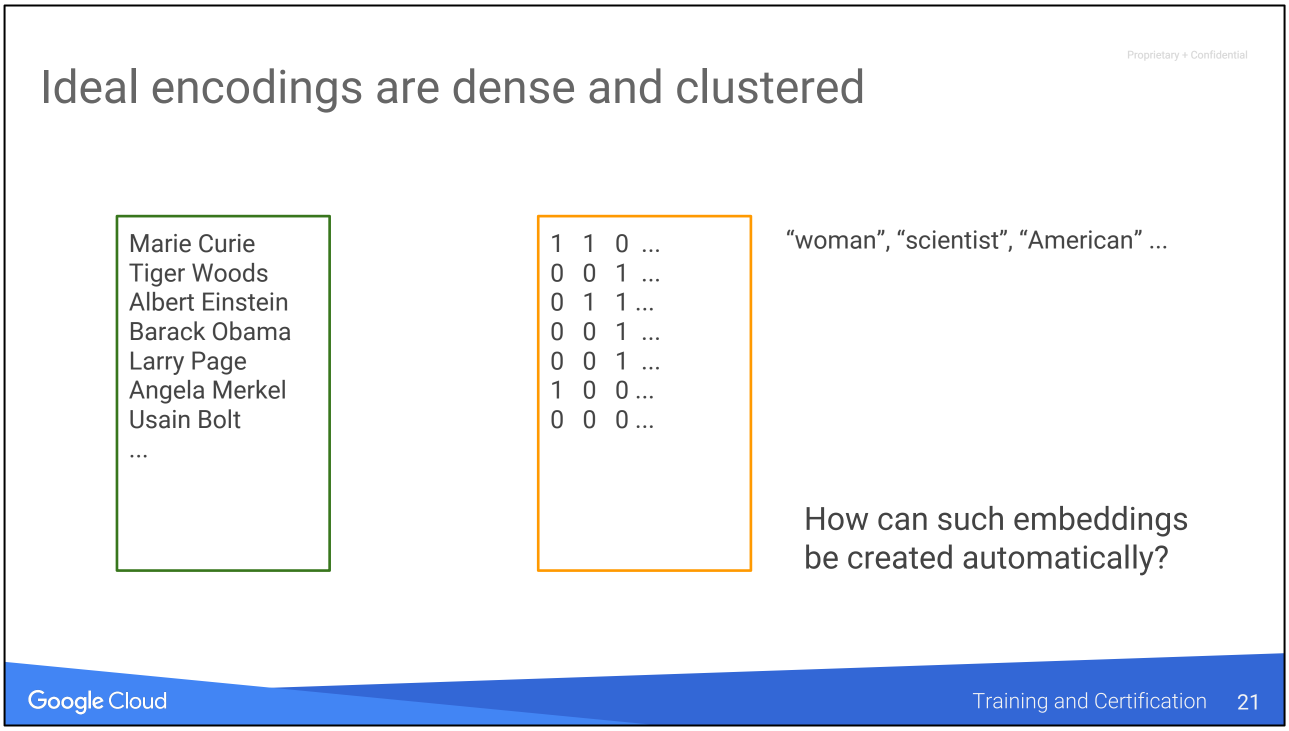
	Arbitrary position	One-hot
Marie Curie	1	1000000
Tiger Woods	2	0100000
Albert Einstein	3	0010000
Barack Obama	4	0001000
Larry Page	5	0000100
Angela Merkel	6	0000010
Usain Bolt	7	0000001
...	...	...

Notes:

The numbers on the right is the dense representation of the one-hot data on the left.

There is no information in terms of what id is assigned to what person, which is why we one-hot encode them, but this is sub-optimal, because things are not truly independent.

This is not just for persons. Any categorical column that is one-hot encoded has this problem. It could be products in your catalog, or parts in your assembly line. It is also not just for features. It also affects labels in your image classification.



Ideal encodings are dense and clustered

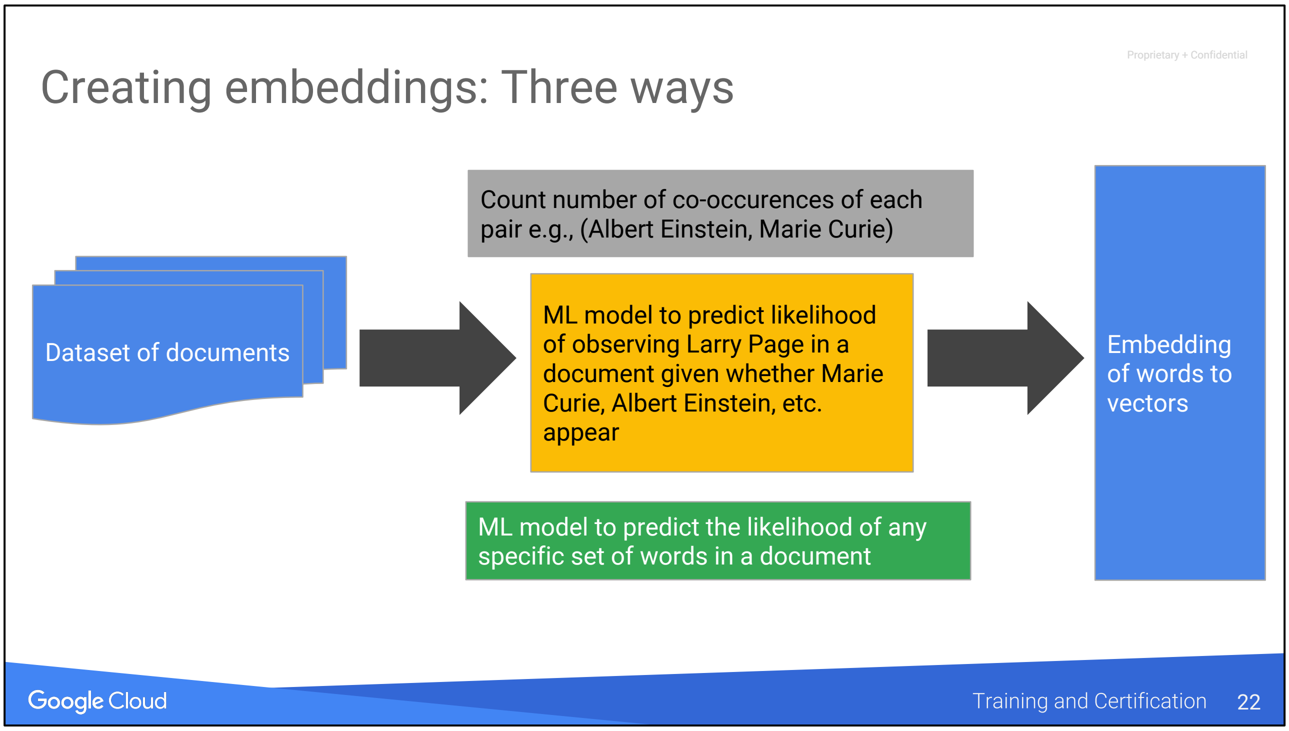
Marie Curie	1 1 0 ...
Tiger Woods	0 0 1 ...
Albert Einstein	0 1 1 ...
Barack Obama	0 0 1 ...
Larry Page	0 0 1 ...
Angela Merkel	1 0 0 ...
Usain Bolt	0 0 0 ...
...	

"woman", "scientist", "American" ...

How can such embeddings be created automatically?

Notes:

Dense: went from having one dimension for every object (7) to just 3.



Proprietary + Confidential

Creating embeddings: Three ways

Dataset of documents
Count number of co-occurences of each pair e.g., (Albert Einstein, Marie Curie)
ML model to predict likelihood of observing Larry Page in a document given whether Marie Curie, Albert Einstein, etc. appear
ML model to predict the likelihood of any specific set of words in a document
Embedding of words to vectors

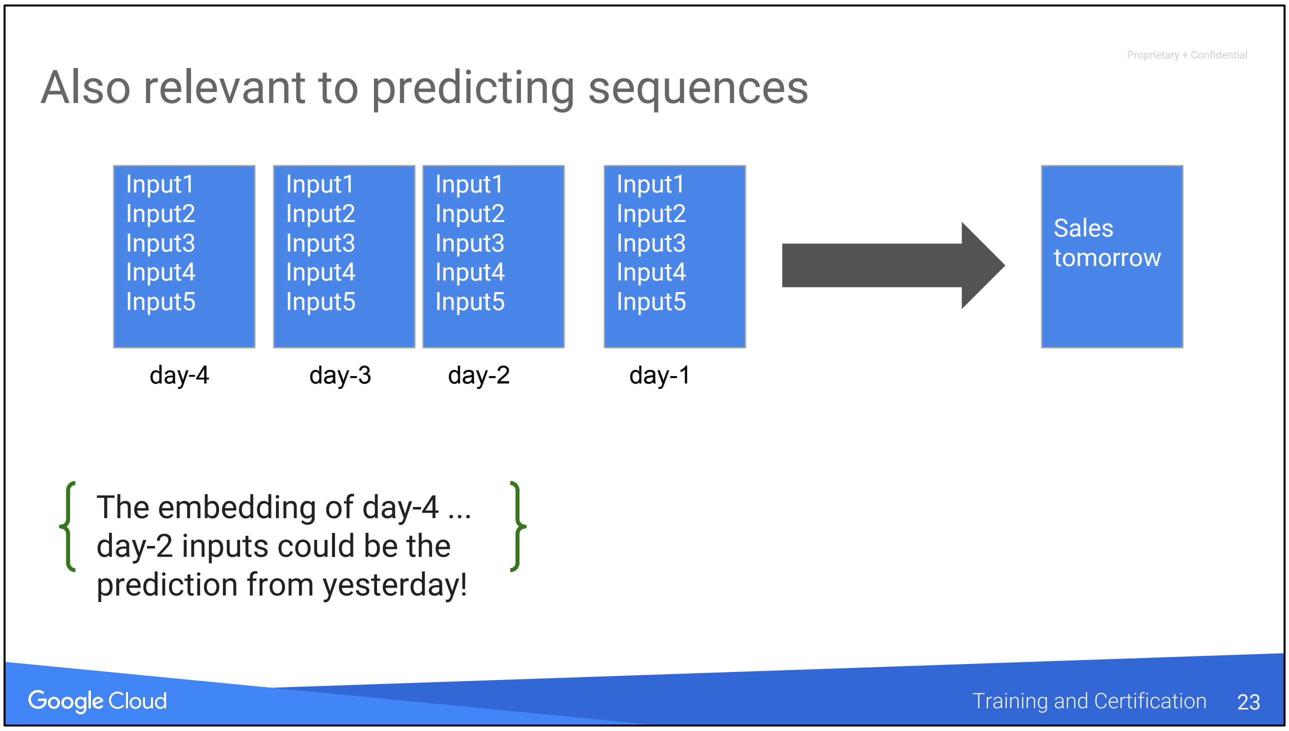
Notes:

For example, for a product catalog, you would look at all your customers' purchases and count how many times diapers & baby bottles cooccur (might be higher than the number of times that diapers and bicycles cooccur).

The first method is called Distributional Statistical Model (DSM). Once you have the cooccurences, various encoding methods exist to put highly cooccuring pairs together but these are pretty heuristic. The latter two approaches are much more hands-off -- all that you have to do is to throw away the output nodes and use the remaining NN to be the vector corresponding to the word.

The second is called Continuous Bag-of-Words (CBOW). It is usually done for small datasets and relatively few words. Essentially, you have one input node for each word and that input node is 0/1 depending on whether the word appears in the document (or a smaller section of the document, termed the context).

The third is called skip-gram and because it is a smoother model, it really comes on its own when you have lots of data. Here, the input is a set of words and the ML model predicts whether that combination is likely to happen or is just noise. Here, each context is treated as an observation, so you get complete coverage over the entire input space.

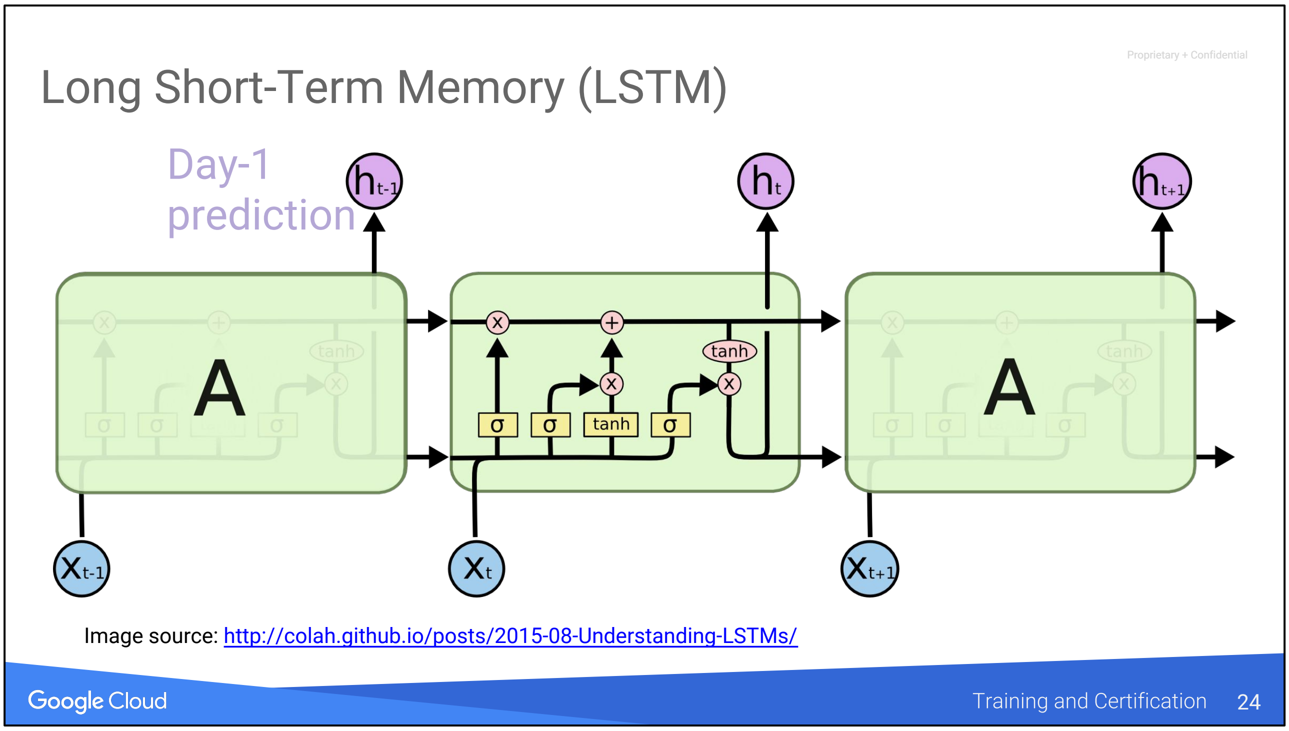


Also relevant to predicting sequences

{ The embedding of day-4 ... day-2 inputs could be the prediction from yesterday! }

Notes:

Yesterday’s prediction captures all that state.



Long Short-Term Memory (LSTM)

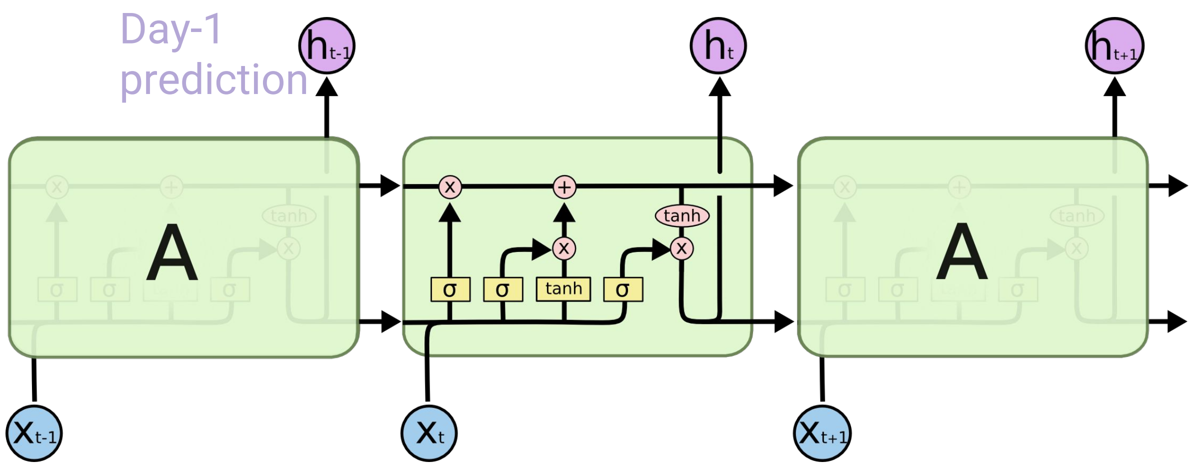
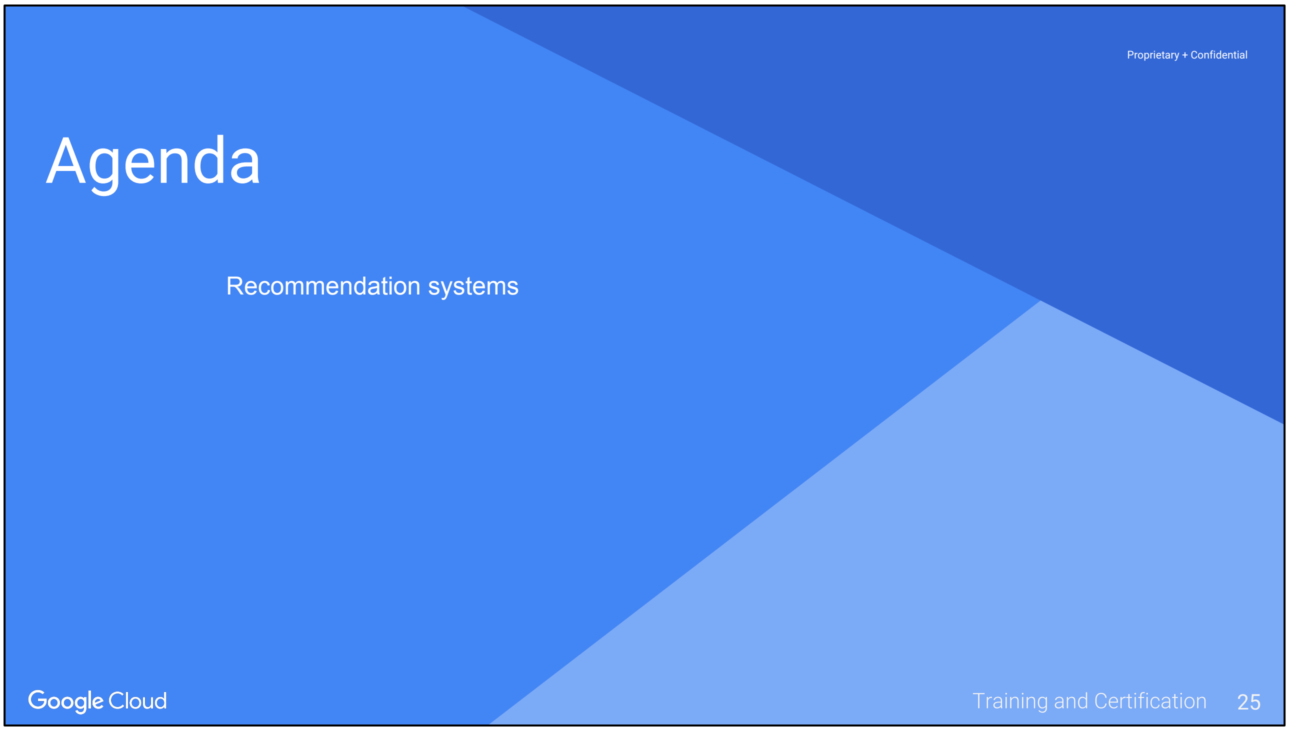


Image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

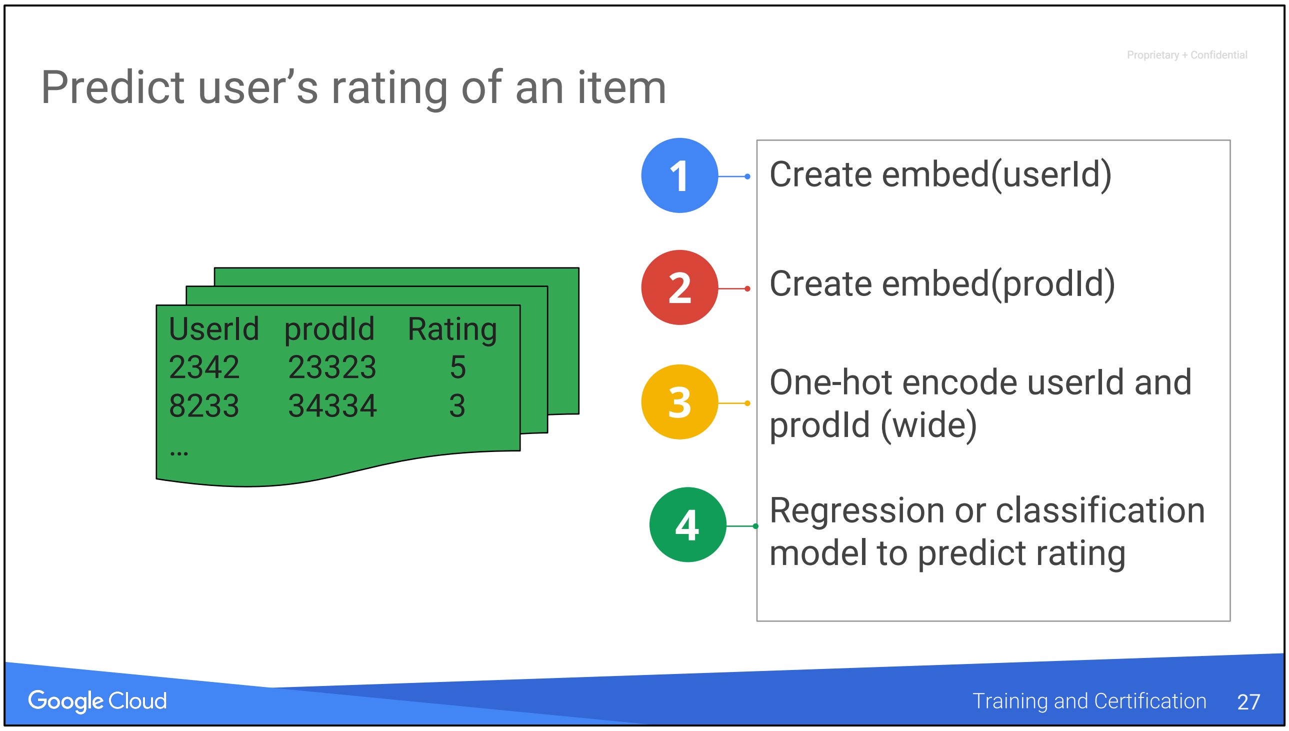
Notes:

The sigmoid node outputs a value between [0,1], so essentially a weighted average of h(t-1) and a function of the current inputs is input to the neural network -- this is the short-term memory. Meanwhile, the internal nodes accumulate over the entire dataset and function like the count-based model in the embedding diagram; i.e., they create long-term memory.



Agenda

Recommendation systems



Predict user's rating of an item

1. Create embed(userId)
2. Create embed(prodId)
3. One-hot encode userId and prodId (wide)
4. Regression or classification model to predict rating

Notes:

The point is to predict the rating of an item the user hasn't rated. This is very sparse, since most users will rate only 5-10 products.

To create embeddings, you have 3 ways:

One is to simply use distributive statistics as discussed in earlier slide. In CPB100, we used an algorithm called Alternating Least Squares to do this using Spark on Dataproc.

Second (and better) is to create a predictive model using the continuous bag of words approach:
To create embedding of userId, predict the likelihood of a user liking a product given the users who liked that product.

To create embedding of prodId, predict the likelihood of a product being liked by a user given the other products that (s)he liked.

Third (and best) when you have a lot of data, you can do the skip-gram approach:

To create embedding of userId, predict the likelihood of any specific set of users liking a given product.

To create embedding of prodId, predict the likelihood of a given user liking any specific set of products.

It could be classification if you treat the ratings as separate classes and do a soft-max.

It could be regression if you treat the ratings as a continuous number in the range [1,5].

Resources

Wide-and-deep	https://www.tensorflow.org/versions/master/tutorials/wide_and_deep/index.html
Image classification	https://www.tensorflow.org/versions/master/tutorials/deep_cnn/index.html
Transfer learning	https://www.tensorflow.org/versions/master/how_tos/image_retraining/index.html
Image generation	https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/deepdream/deepdream.ipynb
Word embeddings	https://www.tensorflow.org/versions/master/tutorials/word2vec/index.html
Recurrent models	https://www.tensorflow.org/versions/master/tutorials/recurrent/index.html
Sequence-to-sequence	https://www.tensorflow.org/versions/master/tutorials/seq2seq/index.html
Recommendation systems	http://arxiv.org/abs/1606.07792 https://github.com/songgc/TF-recomm

cloud.google.com