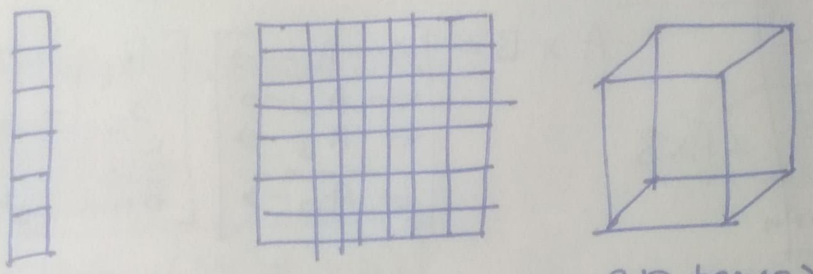
Tensorflow and Keras.

L-1. Tensorflow and Keras.
↳ TF lite for Android

Tensor:



1D tensor 2D-tensor 3D tensor

Basically n D array

Deeplearning → Tensor operations

Tensorflow
- → intricate
- → low level Control
- → longer

}
- lot of patience to learn
- read lot of others Code
- practice a lot.

Keras
- → Make deeplearning simple
- → High level Control (fast deployment)
- → easy to learn.

$\boxed{\text{Keras (frontend)}}$ → $\boxed{\text{Tensorflow (backend)}}$ + $\boxed{\text{Theano (backend)}}$

other :→ TF, Theano, caffee, PyTourch, MxNET

Highlevel → Keras, Keras 2 → Some level of low level control also.

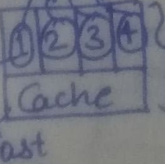
L-2 GPU vs CPU

GPU ↓ {Graphics processing Unit
- → Gaming
- → rendering
- → high Computation

CPU ↓ Central processing Unit

CPU

cores



1 chip

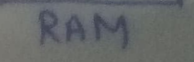
Cache

Chip fast memory

Speed

2-4 GHz

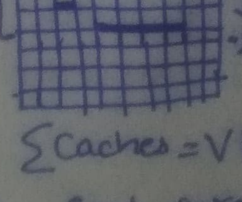
Giga Hertz



RAM

Highly parallelisable.

cores cache



GPU

Speed.

~200 MHz

$\sum$ Caches = VRAM

Each core has Cache in its Core only and Cannot share like CPU.

Deep-learning → Matrix-multiplication
→ Require GPU power

A_1B_1	A_2B_2	A_3B_3	--
--	--	--	=
--	--	--	A_nB_n

$= A \times B$

$$A \times B = \begin{bmatrix} \leftarrow A_1 \rightarrow \\ \leftarrow A_2 \rightarrow \\ \leftarrow A_3 \rightarrow \\ ---- \\ \leftarrow A_n \rightarrow \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ B_n \end{bmatrix} = \begin{bmatrix} A_1B_1 \\ A_2B_2 \\ --- \\ A_nB_n \end{bmatrix}$$

$\boxed{\text{GPU}}$
all core work simultaneously to find A×B matrix

L-3 Google Colab (intro)

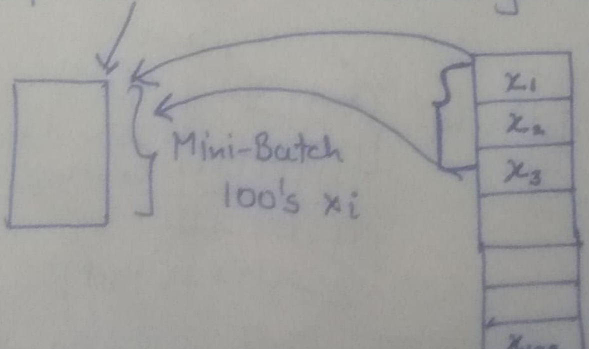
L-4 Install tensorflow.

L-5. Online documentation and tutorial

L-6 Softmax Classifier on MNIST dataset

MNIST dataset (one hot encoded)
{ 0,0,0,1,0,0,1 --- } 784 features.

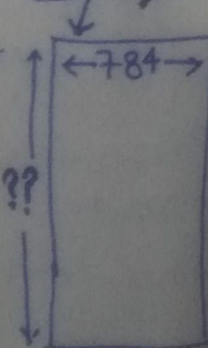
TF
→ Constant → Cannot change a constant
→ Variable $\xrightarrow[\text{update}]{\text{Can}}$ W (weights), b (biases) ⇒ $\sigma(W^T x + b) = y$
→ placeholders → memory location



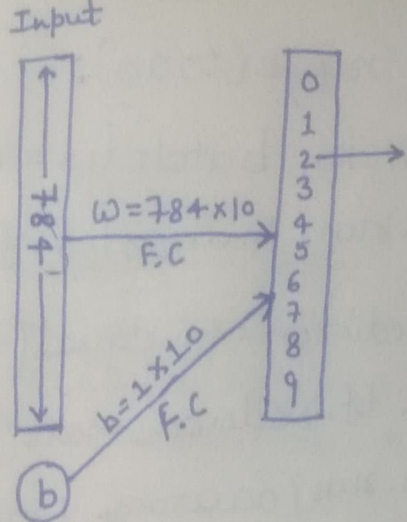
→ x = tf.placeholder(tf.float32, [None, 784])

tf.float32: means placeholder stores floating values

⇒ None means dimensions not defined.



```
W= tf.Variable(tf.zeros([784,10]))
b=tf.Variable(tf.zeros([10])
```



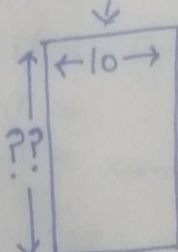

```
y=tf.nn.softmax(tf.matmul(x,w) +b)
```

predicted labels.

$$\boxed{\hat{y} = \text{softmax}(W^T x + b)}$$

```
y_ = tf.placeholder(tf.float,[None,10])
```

actual label



Reduced Sum

```
a=tf.constant([[1, 3], [2, 0], [0, 1]])
b= tf.reduce-sum(a)
c= tf.reduce_sum(a,0)
d= tf.reduce_sum(a,1)
tensors= [b,c, d]
for tensor in tensors:
    result = tf.Session().run(tensor)
    print(result)
```

$$a = \begin{bmatrix} 1 & 3 \\ 2 & 0 \\ 0 & 1 \end{bmatrix}$$

⇒ b = 7

⇒ c = [3 4]

⇒ d = [4 2 1]

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
                                reduction_indices=[1])
```

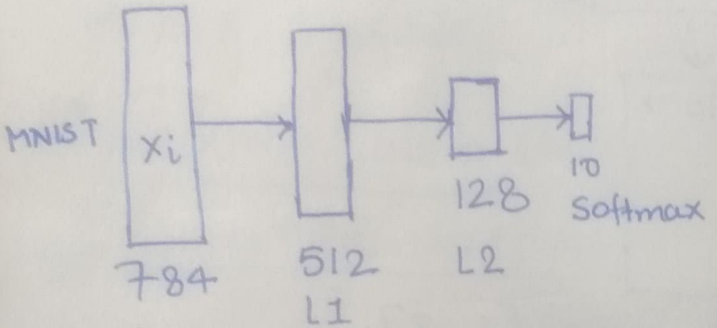
```
train_step= tf.train.GradientDescentOptimizer(0.05).minimize
                                                (crossentropy)
```

learning rate

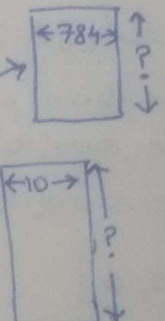
```
sess = tf.InteractiveSession()
tf.global_variable_initializer().run()
```

```
→ for _ in range(1000):
      batch_xs, batch_ys = mnist.train.next_batch(100)
      sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
→ correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
→ accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
→ print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

L-7 MLP-Initialization



\# n_hidden_1 = 512
\# n_hidden_2 = 128
\# n_input = 784
\# n_classes = 10



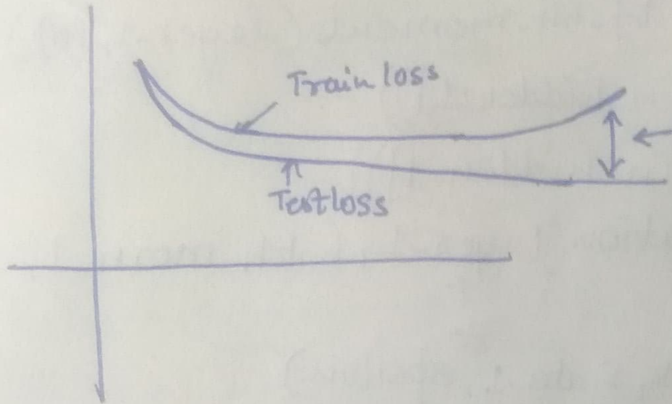

```
xi ← x = tf.placeholder(tf.float32, [None, 784])
yi ← y_ = tf.placeholder(tf.float32, [None, 10])

dropout { keep_prob = tf.placeholder(tf.float32)
        { keep_prob_input = tf.placeholder(tf.float32)
                                   → Single value.

# weights initialization.
   ↙ dictionary
weights_sgd = {
    'h1' : tf.Variable(tf.random_normal([n_input, n_hidden_1], stddev=0.039, mean=0)),
    'h2' : ———"———
    'out' : ———"———
}

biases = {
    'b1' : tf.Variable(tf.random_normal([n_hidden_1])),
    'b2' : ———"———
    'out' : ———"———
}
```

L-8 Model1-Sigmoid activation



← If this start to diverge then you can say model has started to overfit try to use regulization & dropout.

weight in adam + sigmod −0.5 to 0.5.
" " SGD + Sigmoid −0.1 to 0.1 ← much small

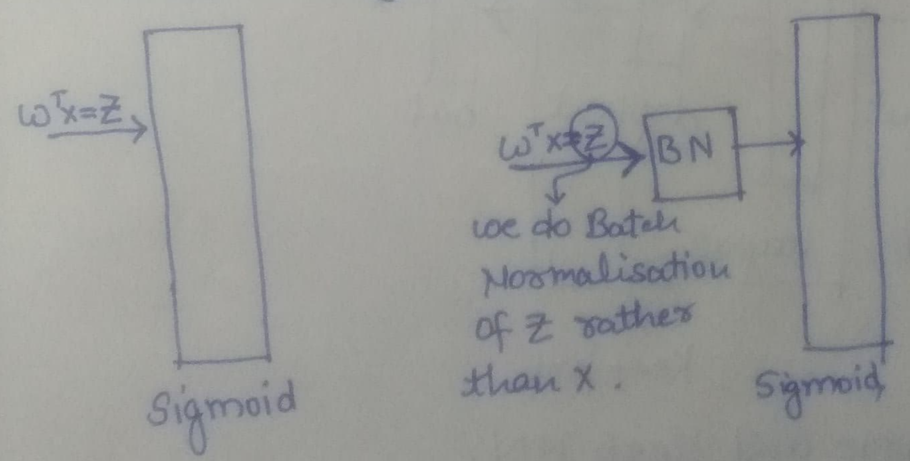
and we don't our weights to too large & too small.

L-9 - Model2- Relu Activation

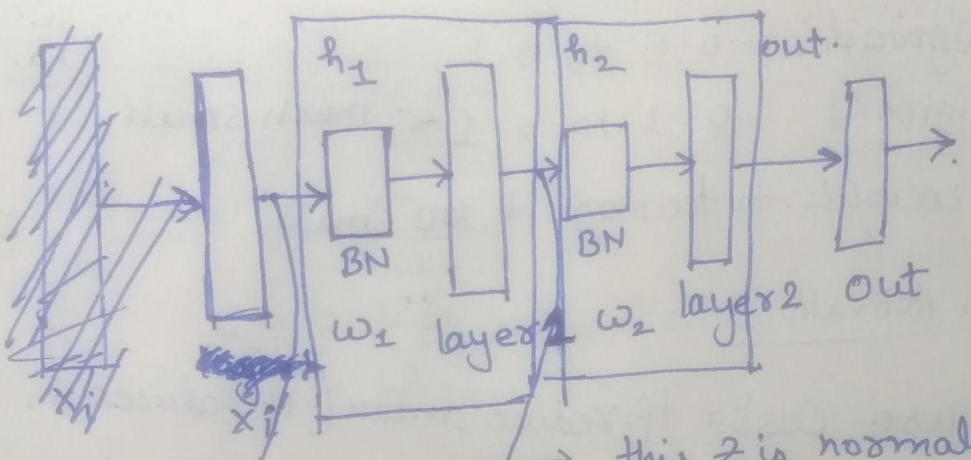
final layer use Cost = tf.reduce_mean(-tf.reducesum (y_ * tf.log(y), reduction_indices=[1])
softmax

final sigmoid layer use cost = tf.reduce_mean(tf.nn.softmax_cross entropy_with_logits(logits=y-sgd, labels=y_)
→ reason No idea.

L-10 Model 3- Sigmoid with Batch Normalisation.



```
layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
batch_mean_1, batch_var_1 = tf.nn.moments(layer_1, [0])
Scale_1 = tf.Variable(tf.ones([n_hidden_1]))
beta_1 = tf.Variable(tf.zeros([n_hidden_1]))
layer_1 = tf.nn.batch_normalization(layer_1, batch_mean_1,
                                           ↑
                                           z
        batch_var_1, beta_1, Scale_1, epsilon)
```

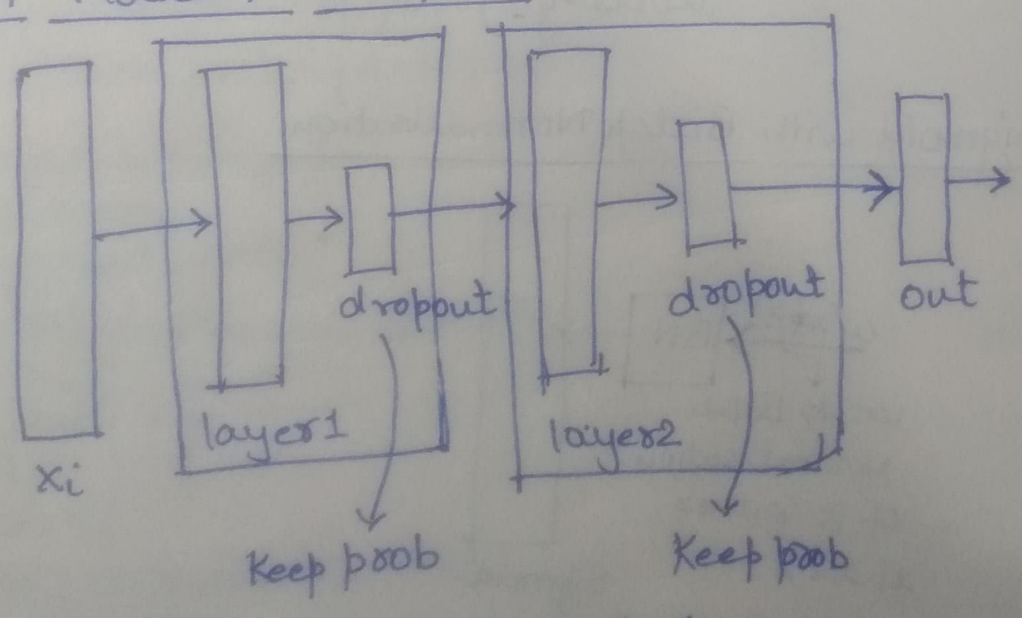


this z is normalised before passing to layer2.

$z = w^T x + b$ $z = w^T x + b$

this z is not normalised before passing to Layer1

L-11 Model 4 - Dropout.



dropout good for large and deep NN.

784-512-128-10

		Multiclass log loss	Accuracy
1	Sigmoid + Adam →	1.48	97.92%
	Sigmoid + SGD →	$\boxed{2.29}$	$\boxed{9.74\%}$
2	Relu + Adam →	1.48	97.85%
	Relu + SGD →	$\boxed{1.823}$	$\boxed{10.66\%}$
3	Sigmoid + BN + Adam →	1.48	98.11%
	Sigmoid + BN + SGD →	2.0606	$\boxed{12.5\%}$
4	Relu + dropout + Adam →	1.48	97.59%
	Relu + dropout + SGD →	2.14	65.53%

Adam is very good optimiser So sigmoid & relu does not show much difference.

SGD is not very good optimizer So we can see the difference between sigmoid & relu.

Compare it with 1 loss is less and accuracy has huge improvement.

++

++

L-12 MNIST classification in Keras

Keras → deployment
→ fast
→ high-level

L-13 Hyperparameter tuning in Keras

lot of hyperparameter

- ① # layers
- ② # Activation unit in Each
- ③ Relu, Sigmoid ?? which model to use.
- ④ Dropout rate

How to use both of them together

Sklearn → Grid Search → Random Search } → hyperparameter

Keras - Hyperparameter

→ keras.wrapper.scikit_learn
refer notebook.

① Define model
② Add layer
③ Compile
④ run

Other alternative to scikitlearn
- Hyperopt
- hyperas.
} → Covered in Case study.

Sklearn ⟷ Keras → TF