TF 2 and Keras (Sequential API)

[ ] ↳ 13 cells hidden

TF 2 and Keras: Functional API

[ ] ↳ 5 cells hidden

Dropout: Regularization through randomization

[ ] ↳ 5 cells hidden

GPUs vs CPUs

↳ 9 cells hidden

Table of contents

4 & 2

+ Code + Text

Reconnect

TF 2 and Keras (Sequential API)

[ ] ↳ 13 cells hidden

TF 2 and Keras: Functional API

[ ] ↳ 5 cells hidden

Dropout: Regularization through randomization

[ ] ↳ 5 cells hidden

GPUs vs CPUs

↳ 9 cells hidden

2 / 2

Table of contents

TF 2 and Keras (Sequential API)

[ ] ↳ 13 cells hidden

TF 2 and Keras: Functional API

[ ] ↳ 5 cells hidden

Dropout: Regularization through randomization

[ ] ↳ 5 cells hidden

GPUs vs CPUs

↳ 9 cells hidden

3 / 3

TF 2 and Keras (Sequential API)

```
[ ] import numpy as np
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Activation, Dropout
    from tensorflow.keras.utils import to_categorical, plot_model
    from tensorflow.keras.datasets import mnist
    import matplotlib.pyplot as plt
```

```
[ ] print(tf.__version__)
```

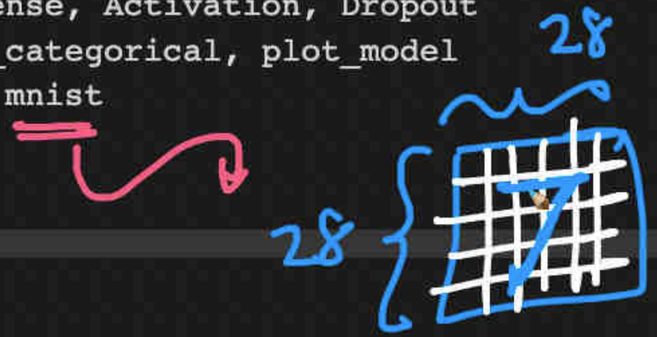
```
2.8.2
```

```
[ ] # load mnist dataset
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
```

TF 2 and Keras (Sequential API)

```
[ ] import numpy as np
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras import layers
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Activation, Dropout
    from tensorflow.keras.utils import to_categorical, plot_model
    from tensorflow.keras.datasets import mnist
    import matplotlib.pyplot as plt
```




```
[ ] print(tf.__version__)
```

```
2.8.2
```

```
[ ] # load mnist dataset
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
```

Q

Classical ML — 100 class classfn: }

↓

1 vs rest → Logistic regrsn

KNN → ✓

Dim red (UMAP) + { Tree → DT, GBDT, RF ... (each class → good amount of data)

✓ [softmax classifier ←

Chrome File Edit View History Bookmarks Profiles Tab Window Help

TF2_Keras_Intro.ipynb - Colab | srivastava14a.pdf | Dropout layer | Layer weight regularizers | 1502.03167.pdf | BatchNormalization layer

colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLlbnXk-5MCI3g#scrollTo=yS4KAX28pCKh

+ Code + Text

RAM Disk

```
[4] # load mnist dataset
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

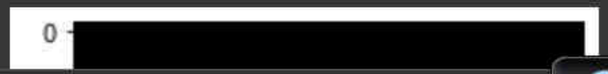
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```


```
idx=1234
plt.imshow(x_train[idx], cmap='gray', interpolation='none')
print(y_train[idx])
```

```
3
```

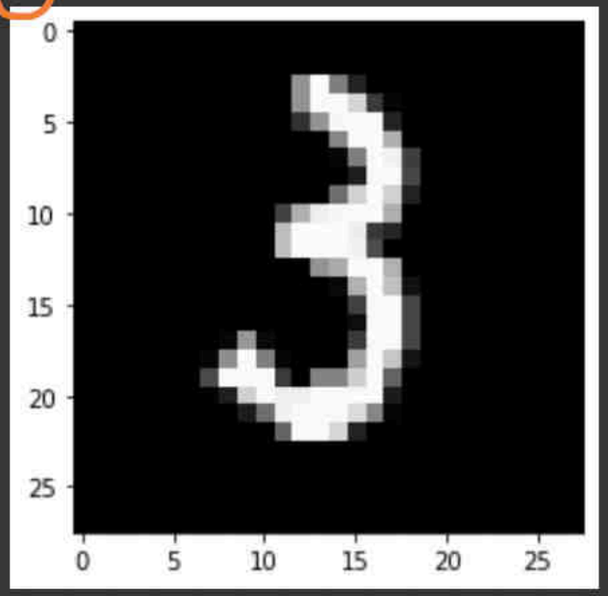


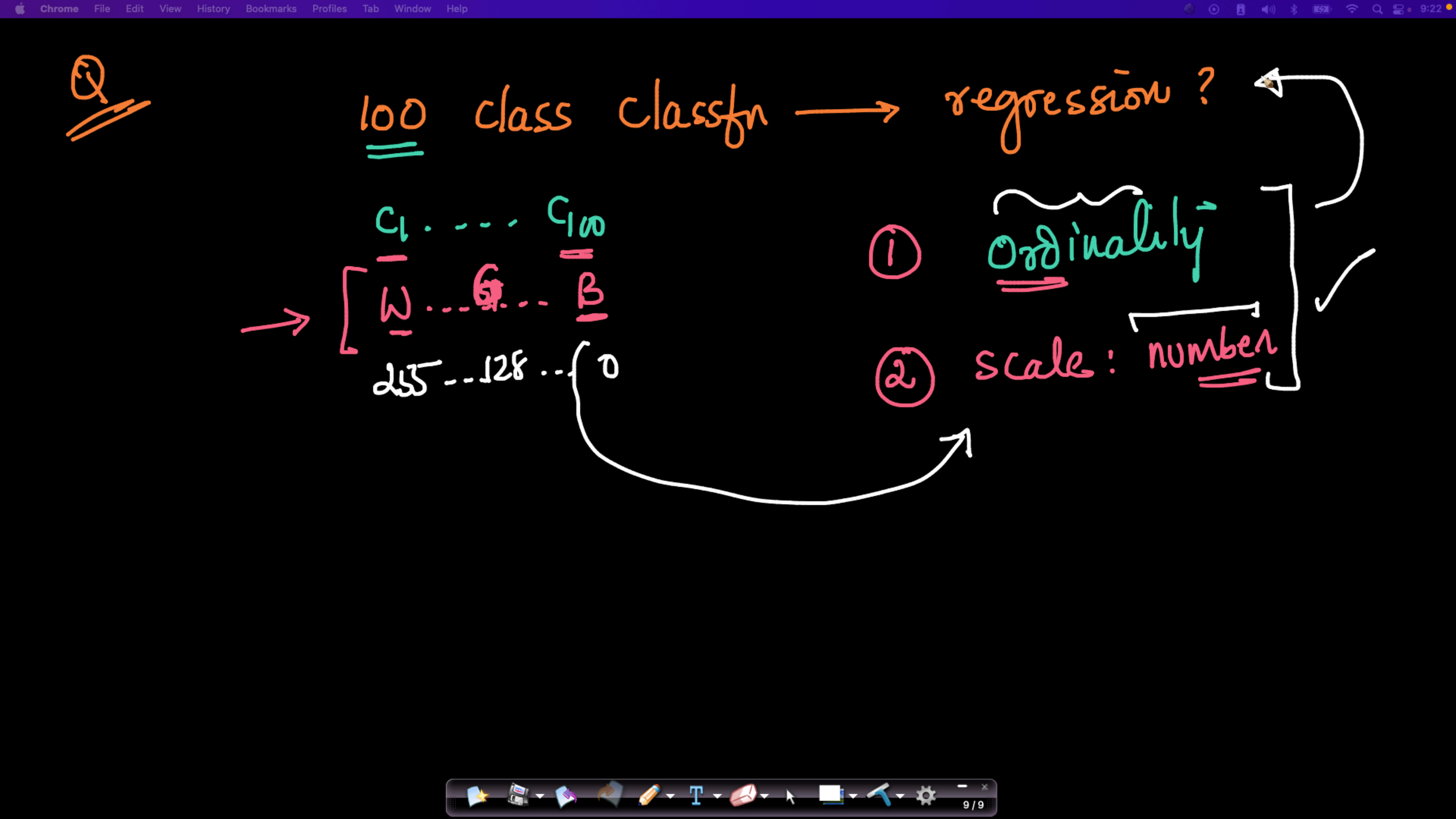
7 / 7

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

```
[ ]  idx=1234
     plt.imshow(x_train[idx], cmap='gray', interpolation='none')
     print(y_train[idx])
```

```
3
```




Q
100 class classfn → regression ?
C1 .... C100
W ... Gr ... B
255 .. 128 .. 0
① Ordinality
② scale : number

Q

$H_p = -\Sigma p_i \log p_i$

$P_1 \; P_2 \; \cdots \; P_{10}$

$P_1 \; P_2$

$0.5 \quad 0.5$

$(-2) \cdot 0.5 \log 0.5$

$= 0.3010$

$P_1 \; P_2 \; \cdots \; P_{10}$

$0.1 \quad 0.1 \quad \cdots \quad 0.1$

$(-10) \cdot 0.1 \cdot \log 0.1 = 1.0$

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:27

TF2_Keras_Intro.ipynb - Colab | srivastava14a.pdf | Dropout layer | Layer weight regularizers | 1502.03167.pdf | BatchNormalization layer

colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLibnXk-5MCI3g#scrollTo=IrBxXCgxpNaf

Update

+ Code + Text

RAM
Disk

```
# number of class labels
num_labels = len(np.unique(y_train))
print(num_labels)
```

```
10
```

```
# convert y_i's to one-hot vectors
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

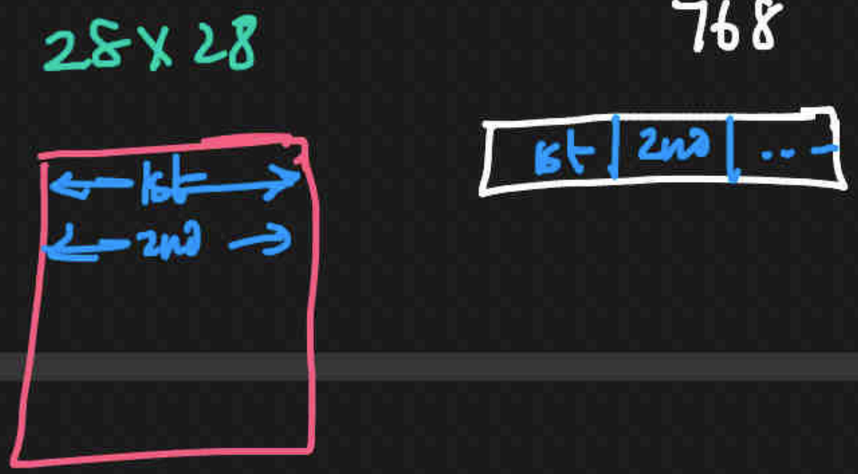
```
# Image: 2D matrix to 1D vector
image_size = x_train.shape[1]
input_size = image_size * image_size

# 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255
```

```
# Hyper-params of NN
batch_size = 128 # typically 2^n
hidden_units = 256
```

11 / 11

```
[ ] # convert y_i's to one-hot vectors
    y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)
```

```
[ ] # Image: 2D matrix to 1D vector
    image_size = x_train.shape[1]
    input_size = image_size * image_size

    # 2D to 1D + Normalize (0-255)
    x_train = np.reshape(x_train, [-1, input_size])
    x_train = x_train.astype('float32') / 255
    x_test = np.reshape(x_test, [-1, input_size])
    x_test = x_test.astype('float32') / 255
```




```
[ ] # Hyper-params of NN
    batch_size = 128 # typically 2^n
    hidden_units = 256
```

```
[ ] # Sequential API

    # 3-layer MLP with ReLU
    model = Sequential()
    model.add(Dense(hidden_units, input_dim=input_size))
    model.add(Activation('relu'
```

```
# convert y_i's to one-hot vectors
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
# Image: 2D matrix to 1D vector
image_size = x_train.shape[1]
input_size = image_size * image_size

# 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255
```

28x28 → 768개

```
# Hyper-params of NN
batch_size = 128 # typically 2^n
hidden_units = 256
```

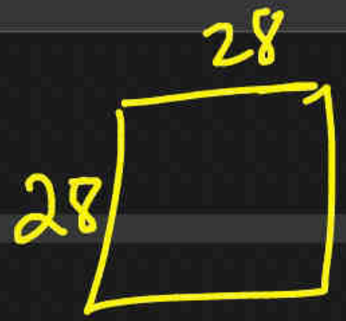
```
# Sequential API

# 3-layer MLP with ReLU
model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
```

```
# convert y_i's to one-hot vectors
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
# Image: 2D matrix to 1D vector
image_size = x_train.shape[1]
input_size = image_size * image_size

# 2D to 1D + Normalize (0-255)
x_train = np.reshape(x_train, [-1, input_size])
x_train = x_train.astype('float32') / 255
x_test = np.reshape(x_test, [-1, input_size])
x_test = x_test.astype('float32') / 255
```

```
# Hyper-params of NN
batch_size = 128 # typically 2^n
hidden_units = 256
```

```
# Sequential API

# 3-layer MLP with ReLU
model = Sequential()
model.add(Dense(hidden_units, input_dim=input_size))
```



black
0 : min
255 : max

while
std → mean-centring & var-scaling

Images → Normlzn 0-1
↓
Min-Max scaling
↓ ↓
0 1

```
[ ]  image_size = x_train.shape[1]
     input_size = image_size * image_size

     # 2D to 1D + Normalize (0-255)
     x_train = np.reshape(x_train, [-1, input_size])
     x_train = x_train.astype('float32') / 255
     x_test = np.reshape(x_test, [-1, input_size])
     x_test = x_test.astype('float32') / 255
```

60k images

```
     # Hyper-params of NN
     batch_size = 128 # typically 2^n
     hidden_units = 256
```


```
[ ]  # Sequential API

     # 3-layer MLP with ReLU
     model = Sequential()
     model.add(Dense(hidden_units, input_dim=input_size))
     model.add(Activation('relu'))
     model.add(Dense(hidden_units))
     model.add(Activation('relu'))
     model.add(Dense(num_labels))
     # this is the output for one-hot vector
     model.add(Activation('softmax'))
```

```
[ ]  # Image: 2D matrix to 1D vector
     image_size = x_train.shape[1]
     input_size = image_size * image_size

     # 2D to 1D + Normalize (0-255)
     x_train = np.reshape(x_train, [-1, input_size])
     x_train = x_train.astype('float32') / 255
     x_test = np.reshape(x_test, [-1, input_size])
     x_test = x_test.astype('float32') / 255
```

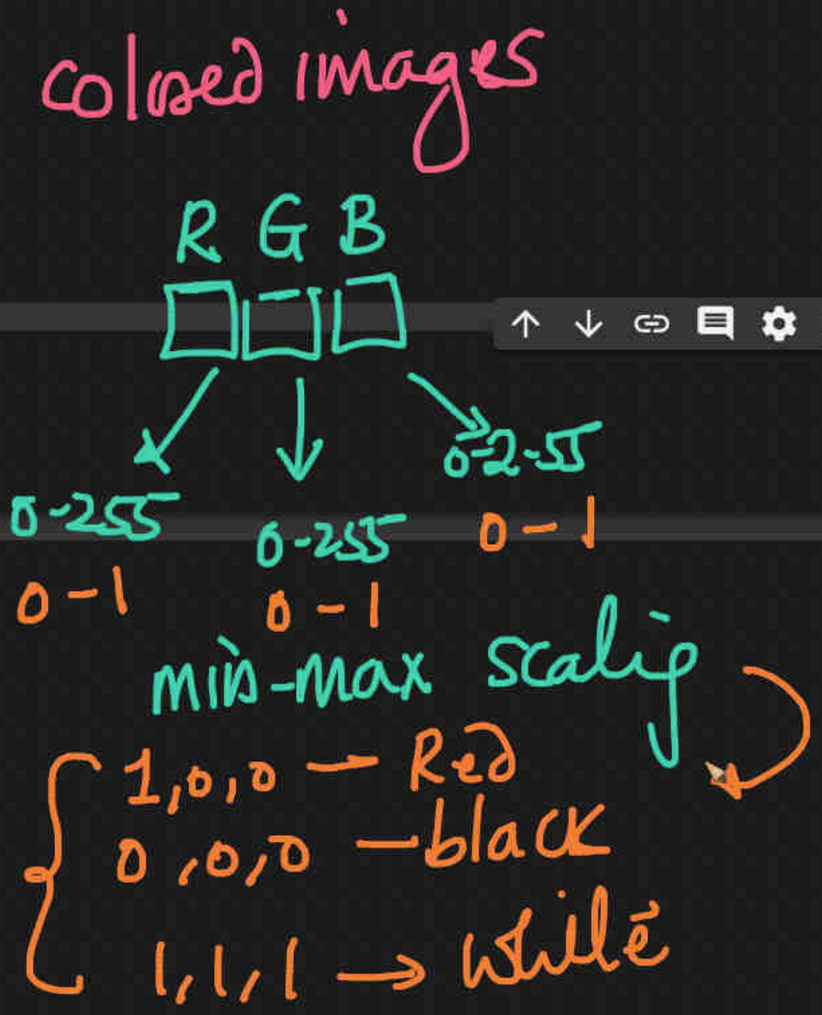
```
     # Hyper-params of NN
     batch_size = 128 # typically 2^n
     hidden_units = 256
```

```
[ ]  # Sequential API

     # 3-layer MLP with ReLU
     model = Sequential()
     model.add(Dense(hidden_units, input_dim=input_size))
     model.add(Activation('relu'))
     model.add(Dense(hidden_units))
     model.add(Activation('relu'))
     model.add(Dense(num_labels))
     # this is the output for one-hot vector
     model.add(Activation('softmax'))
```

```
[ ]  image_size = x_train.shape[1]
     input_size = image_size * image_size

     # 2D to 1D + Normalize (0-255)
     x_train = np.reshape(x_train, [-1, input_size])
     x_train = x_train.astype('float32') / 255
     x_test = np.reshape(x_test, [-1, input_size])
     x_test = x_test.astype('float32') / 255
```

```
     # Hyper-params of NN
     batch_size = 128 # typically 2^n
     hidden_units = 256
```

```
[ ]  # Sequential API

     # 3-layer MLP with ReLU
     model = Sequential()
     model.add(Dense(hidden_units, input_dim=input_size))
     model.add(Activation('relu'))
     model.add(Dense(hidden_units))
     model.add(Activation('relu'))
     model.add(Dense(num_labels))
     # this is the output for one-hot vector
     model.add(Activation('softmax'))
```



TF2_Keras_Intro.ipynb - Colab | srivastava14a.pdf | Dropout layer | Layer weight regularizers | 1502.03167.pdf | BatchNormalization layer

colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLlbnXk-5MCI3g#scrollTo=SN71uszYqbS9

+ Code + Text

RAM
Disk

```
[ ]  image_size = x_train.shape[1]
     input_size = image_size * image_size

     # 2D to 1D + Normalize (0-255)
     x_train = np.reshape(x_train, [-1, input_size])
     x_train = x_train.astype('float32') / 255
     x_test = np.reshape(x_test, [-1, input_size])
     x_test = x_test.astype('float32') / 255
```

mini-batch SGD

epoch → iterns

batch-size

```
     # Hyper-params of NN
     batch_size = 128 # typically 2^n
     hidden_units = 256
```

```
[ ]  # Sequential API

     # 3-layer MLP with ReLU
     model = Sequential()
     model.add(Dense(hidden_units, input_dim=input_size))
     model.add(Activation('relu'))
     model.add(Dense(hidden_units))
     model.add(Activation('relu'))
     model.add(Dense(num_labels))
     # this is the output for one-hot vector
     model.add(Activation('softmax'))
```

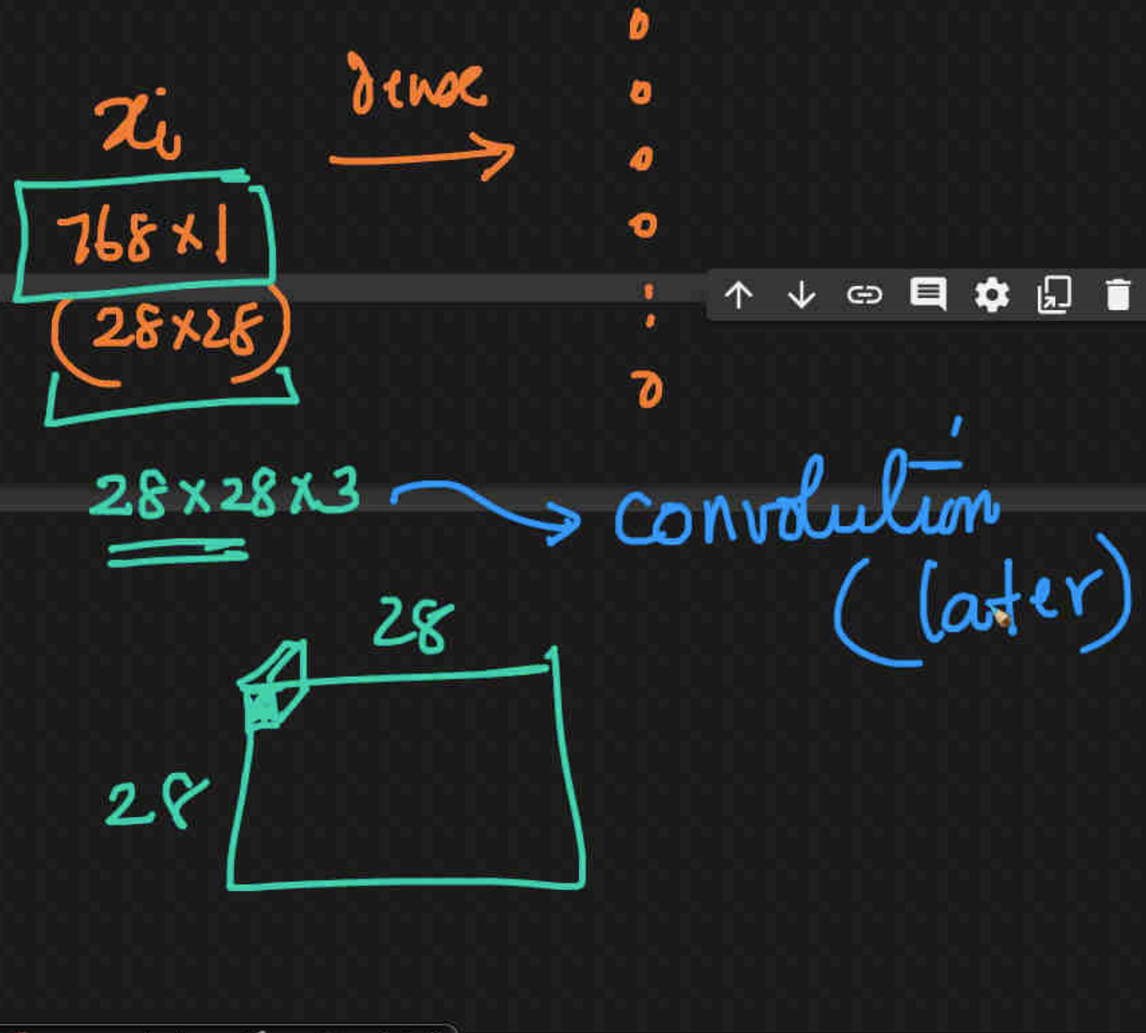
18 / 18

+ Code + Text

RAM Disk

```
[ ]  # Image: 2D matrix to 1D vector
     image_size = x_train.shape[1]
     input_size = image_size * image_size

     # 2D to 1D + Normalize (0-255)
     x_train = np.reshape(x_train, [-1, input_size])
     x_train = x_train.astype('float32') / 255
     x_test = np.reshape(x_test, [-1, input_size])
     x_test = x_test.astype('float32') / 255
```

```
     # Hyper-params of NN
     batch_size = 128 # typically 2^n
     hidden_units = 256
```




```
[ ]  # Sequential API

     # 3-layer MLP with ReLU
     model = Sequential()
     model.add(Dense(hidden_units, input_dim=input_size))
     model.add(Activation('relu'))
     model.add(Dense(hidden_units))
     model.add(Activation('relu'))
     model.add(Dense(num_labels))
     # this is the output for one-hot vector
     model.add(Activation('softmax'))
```

19 / 19

```
[ ]  # Image: 2D matrix to 1D vector
     image_size = x_train.shape[1]
     input_size = image_size * image_size

     # 2D to 1D + Normalize (0-255)
     x_train = np.reshape(x_train, [-1, input_size])
     x_train = x_train.astype('float32') / 255
     x_test = np.reshape(x_test, [-1, input_size])
     x_test = x_test.astype('float32') / 255
```

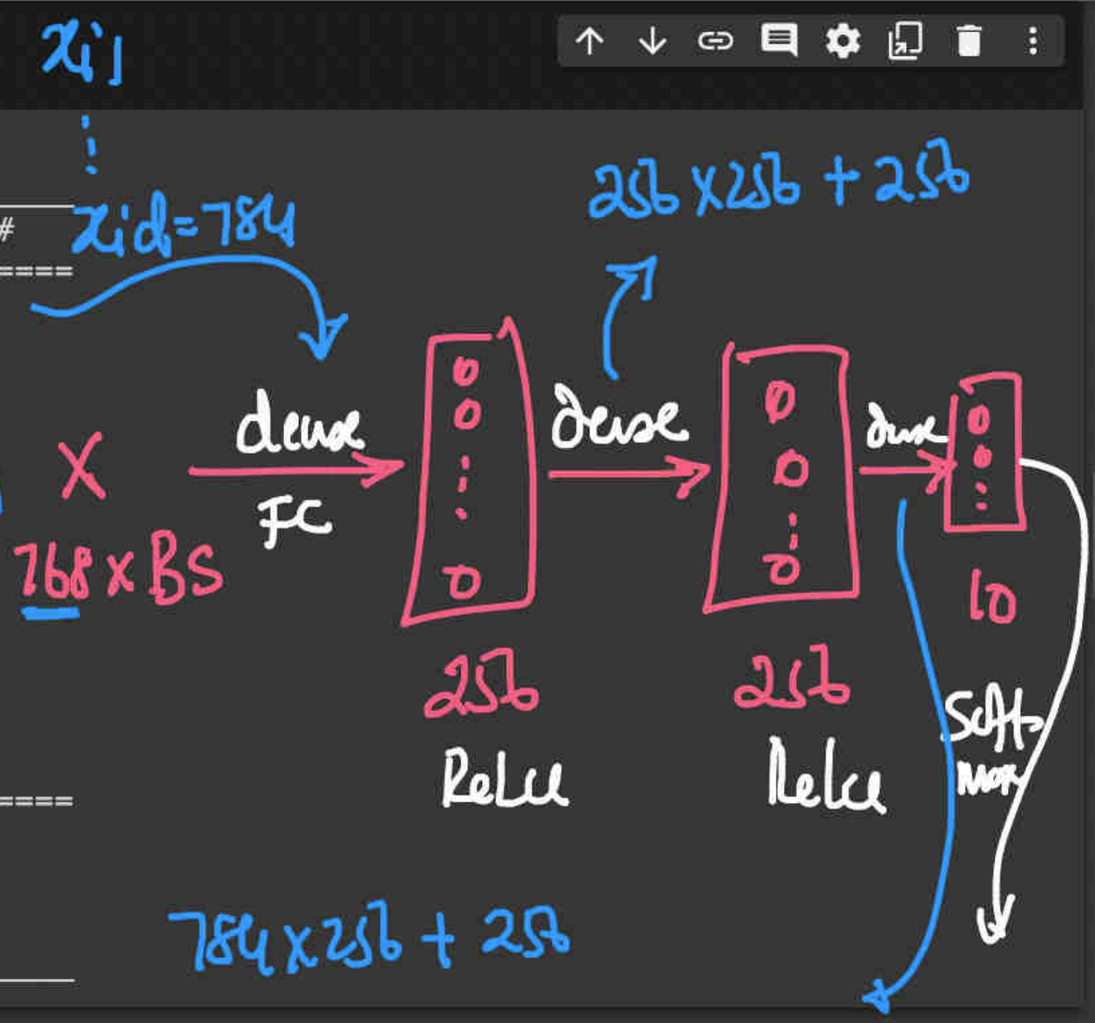
Tensor Flow

```
     # Hyper-params of NN
     batch_size = 128 # typically 2^n
     hidden_units = 256
```

```
[ ]  # Sequential API

     # 3-layer MLP with ReLU
     model = Sequential()
     model.add(Dense(hidden_units, input_dim=input_size))
     model.add(Activation('relu'))
     model.add(Dense(hidden_units))
     model.add(Activation('relu'))
     model.add(Dense(num_labels))
     # this is the output for one-hot vector
     model.add(Activation('softmax'))
```

```
model.add(Activation('softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense (Dense)                (None, 256)               200960

 activation (Activation)      (None, 256)               0

 dense_1 (Dense)              (None, 256)               65792

 activation_1 (Activation)    (None, 256)               0

 dense_2 (Dense)              (None, 10)                2570

 activation_2 (Activation)    (None, 10)                0

=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
_________________________________________________________________
```




```
[ ] model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
```

```
activation (Activation)      (None, 256)               0

dense_1 (Dense)              (None, 256)               65792

activation_1 (Activation)    (None, 256)               0

dense_2 (Dense)              (None, 10)                2570

activation_2 (Activation)    (None, 10)                0

=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
_________________________________________________________________
```

```
[ ] model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```

```
[ ] # train the network
    %%time
    model.fit(x_train, y_train, epochs=20, batch_size=batch_size)

    #60000/128 ~ 469

    Epoch 1/20
```

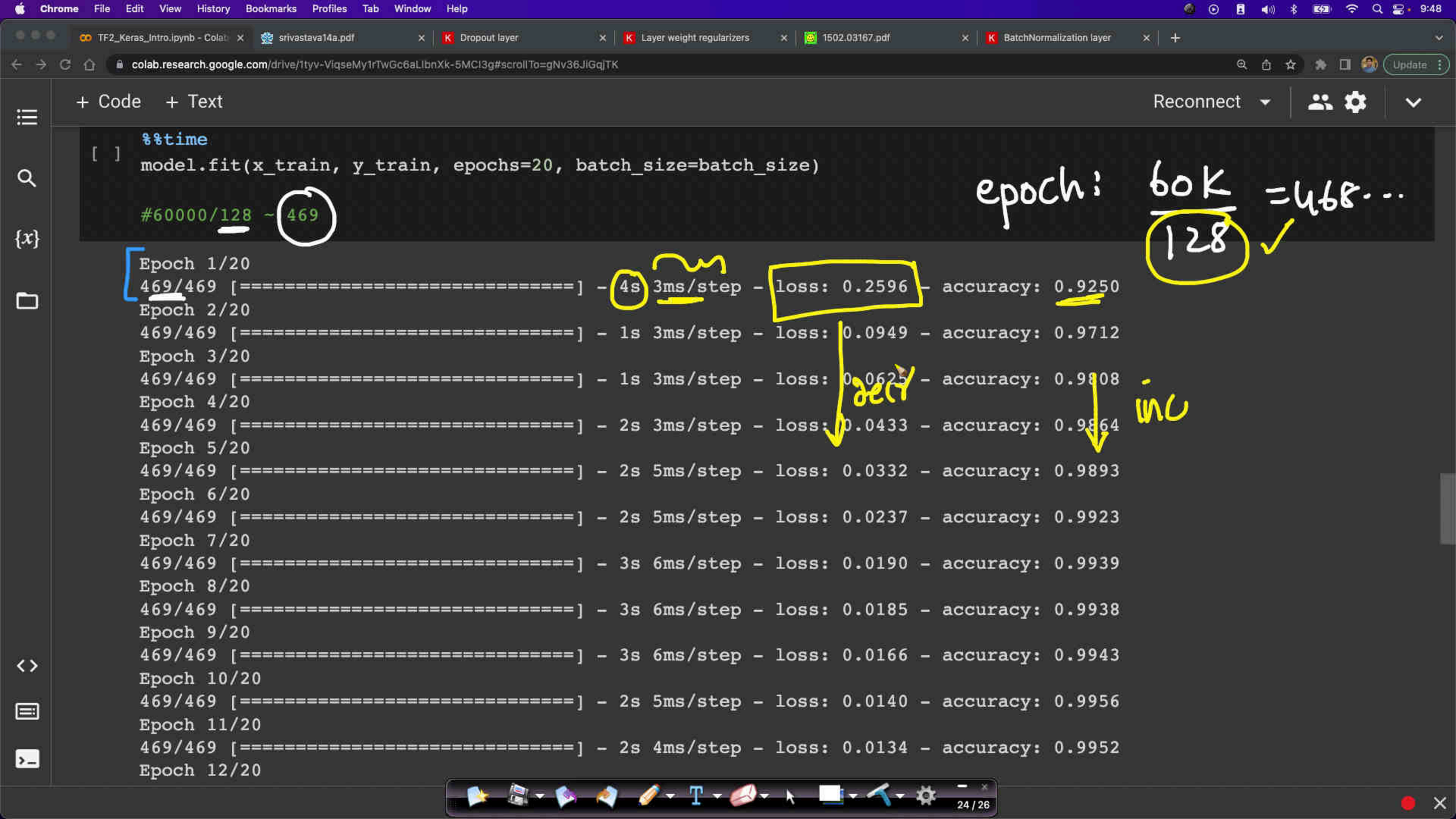
```
 activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
_________________________________________________________________
```

```
[ ] model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```

```
[ ] # train the network
    %%time
    model.fit(x_train, y_train, epochs=20, batch_size=batch_size)

    #60000/128 ~ 469
```

128

```
Epoch 1/20
469/469 [==============================] - 4s 3ms/step - loss: 0.2596 - accuracy: 0.9250
Epoch 2/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0949 - accuracy: 0.9712
Epoch 3/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0625 - accuracy: 0.9808
Epoch 4/20
469/469 [==============================] - 2s 3ms/step - loss: 0.0433 - accuracy: 0.9864
```


Chrome File Edit View History Bookmarks Profiles Tab Window Help
TF2_Keras_Intro.ipynb - Colab
srivastava14a.pdf
Dropout layer
Layer weight regularizers
1502.03167.pdf
BatchNormalization layer
colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLibnXk-5MCI3g#scrollTo=gNv36JiGqjTK
+ Code
+ Text
Reconnect
%%time
model.fit(x_train, y_train, epochs=20, batch_size=batch_size)
#60000/128 ~ 469
epoch: 60k/128 =468...
Epoch 1/20
469/469 [==============================] - 4s 3ms/step - loss: 0.2596 - accuracy: 0.9250
Epoch 2/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0949 - accuracy: 0.9712
Epoch 3/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0625 - accuracy: 0.9808
Epoch 4/20
469/469 [==============================] - 2s 3ms/step - loss: 0.0433 - accuracy: 0.9864
Epoch 5/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0332 - accuracy: 0.9893
Epoch 6/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0237 - accuracy: 0.9923
Epoch 7/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0190 - accuracy: 0.9939
Epoch 8/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0185 - accuracy: 0.9938
Epoch 9/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0166 - accuracy: 0.9943
Epoch 10/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0140 - accuracy: 0.9956
Epoch 11/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0134 - accuracy: 0.9952
Epoch 12/20
dec
inc
24 / 26

```
[ ] model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```

```
[ ] # train the network
    %%time
    model.fit(x_train, y_train, epochs=20, batch_size=batch_size)

    #60000/128 ~ 469
```

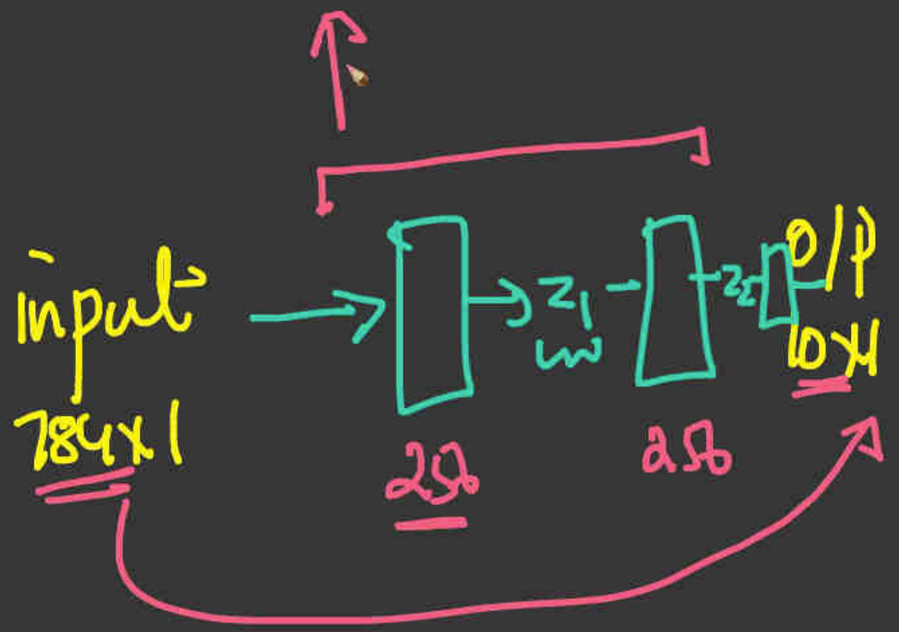
```
Epoch 1/20
469/469 [==============================] - 4s 3ms/step - loss: 0.2596 - accuracy: 0.9250
Epoch 2/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0949 - accuracy: 0.9712
Epoch 3/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0625 - accuracy: 0.9808
Epoch 4/20
469/469 [==============================] - 2s 3ms/step - loss: 0.0433 - accuracy: 0.9864
Epoch 5/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0332 - accuracy: 0.9893
Epoch 6/20
469/469 [==============================] - 2s 5ms/step - loss: 0.0237 - accuracy: 0.9923
Epoch 7/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0190 - accuracy: 0.9939
Epoch 8/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0185 - accuracy: 0.9938
Epoch 9/20
```

```
Epoch 11/20
469/469 [==============================] - 2s 4ms/step - loss: 0.0134 - accuracy: 0.9952
Epoch 12/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0103 - accuracy: 0.9969
Epoch 13/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0113 - accuracy: 0.9961
Epoch 14/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0100 - accuracy: 0.9965
Epoch 15/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0073 - accuracy: 0.9976
Epoch 16/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0082 - accuracy: 0.9973
Epoch 17/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0064 - accuracy: 0.9980
Epoch 18/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0071 - accuracy: 0.9976
Epoch 19/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0091 - accuracy: 0.9968
Epoch 20/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0076 - accuracy: 0.9973
CPU times: user 36.1 s, sys: 4.74 s, total: 40.8 s
Wall time: 38.6 s
<keras.callbacks.History at 0x7fec604b7190>
```

```
# test dataset  performance
loss, acc = model.evaluate(x_test,
                           y_test,
```

+ Code + Text Reconnect

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               200960

 activation (Activation)     (None, 256)               0

 dense_1 (Dense)             (None, 256)               65792

 activation_1 (Activation)   (None, 256)               0

 dense_2 (Dense)             (None, 10)                2570

 activation_2 (Activation)   (None, 10)                0

=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
_________________________________________________________________
```




```
[ ] model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```

```
[ ] # train the network
```

```
Epoch 16/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0082 - accuracy: 0.9973
Epoch 17/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0064 - accuracy: 0.9980
Epoch 18/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0071 - accuracy: 0.9976
Epoch 19/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0091 - accuracy: 0.9968
Epoch 20/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0076 - accuracy: 0.9973
CPU times: user 36.1 s, sys: 4.74 s, total: 40.8 s
Wall time: 38.6 s
<keras.callbacks.History at 0x7fec604b7190>
```

```python
# test dataset  performance
loss, acc = model.evaluate(x_test,
                           y_test,
                           batch_size=batch_size,
                           verbose=0)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
print(loss)

# Overfit or Underfit?
```

```
Test accuracy: 98.3%
0.09489700198173523
```

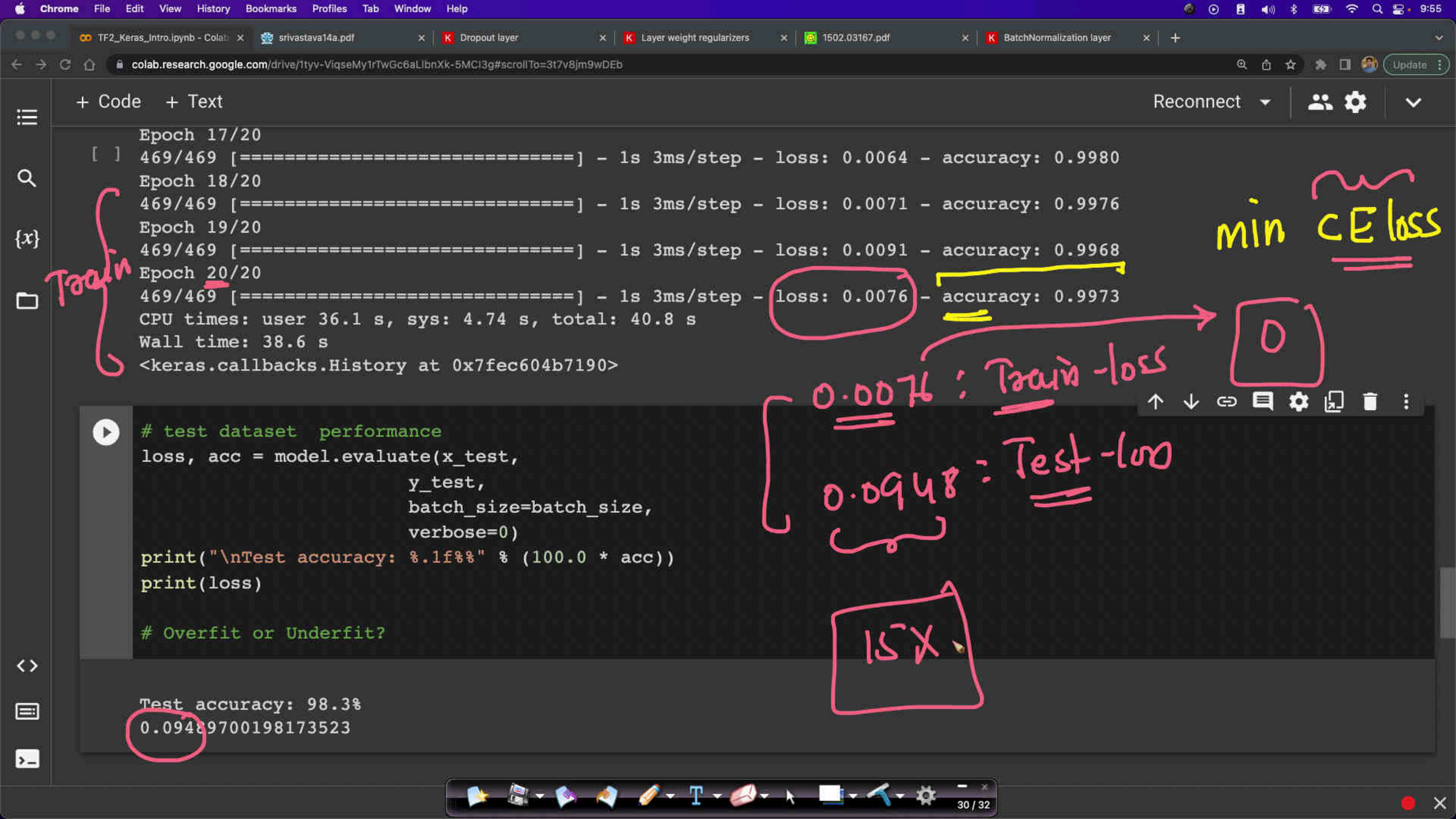
```
Epoch 18/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0071 - accuracy: 0.9976
Epoch 19/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0091 - accuracy: 0.9968
Epoch 20/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0076 - accuracy: 0.9973
CPU times: user 36.1 s, sys: 4.74 s, total: 40.8 s
Wall time: 38.6 s
<keras.callbacks.History at 0x7fec604b7190>
```

```
# test dataset  performance
loss, acc = model.evaluate(x_test,
                           y_test,
                           batch_size=batch_size,
                           verbose=0)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
print(loss)

# Overfit or Underfit?
```

```
Test accuracy: 98.3%
0.09489700198173523
```

Test

TF 2 and Keras: Functional API


Chrome File Edit View History Bookmarks Profiles Tab Window Help
TF2_Keras_Intro.ipynb - Colab
srivastava14a.pdf
Dropout layer
Layer weight regularizers
1502.03167.pdf
BatchNormalization layer
colab.research.google.com/drive/1tyv-ViqseMy1rTwGc6aLibnXk-5MCI3g#scrollTo=3t7v8jm9wDEb
+ Code + Text
Reconnect
Epoch 17/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0064 - accuracy: 0.9980
Epoch 18/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0071 - accuracy: 0.9976
Epoch 19/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0091 - accuracy: 0.9968
Epoch 20/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0076 - accuracy: 0.9973
CPU times: user 36.1 s, sys: 4.74 s, total: 40.8 s
Wall time: 38.6 s
<keras.callbacks.History at 0x7fec604b7190>
Train
min CE loss
0
0.0076 : Train-loss
0.0948 : Test-loss
ISX
test dataset performance
loss, acc = model.evaluate(x_test,
y_test,
batch_size=batch_size,
verbose=0)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
print(loss)
# Overfit or Underfit?
Test accuracy: 98.3%
0.09489700198173523
30 / 32

```
469/469 [==============================] - 1s 3ms/step - loss: 0.0071 - accuracy: 0.9976
Epoch 19/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0091 - accuracy: 0.9968
Epoch 20/20
469/469 [==============================] - 1s 3ms/step - loss: 0.0076 - accuracy: 0.9973
CPU times: user 36.1 s, sys: 4.74 s, total: 40.8 s
Wall time: 38.6 s
<keras.callbacks.History at 0x7fec604b7190>
```

```
# test dataset  performance
loss, acc = model.evaluate(x_test,
                           y_test,
                           batch_size=batch_size,
                           verbose=0)
print("\nTest accuracy: %.1f%%" % (100.0 * acc))
print(loss)

# Overfit or Underfit?
```

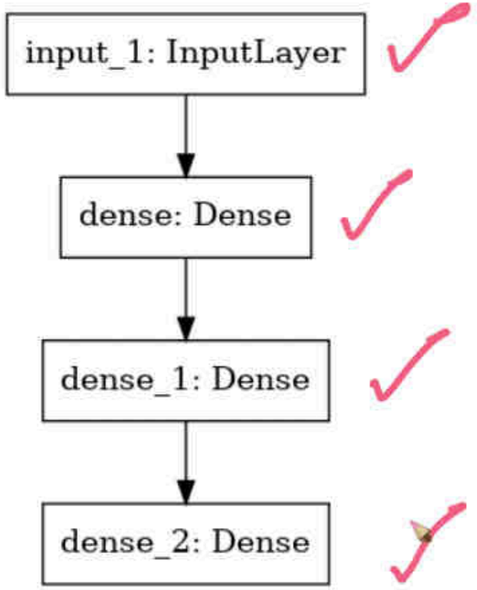
```
Test accuracy: 98.3%
0.09489700198173523
```

TF 2 and Keras: Functional API

```
Total params: 55,050
Trainable params: 55,050
```

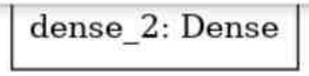
You can also plot the model as a graph:

```
keras.utils.plot_model(model, "my_first_model.png")
```



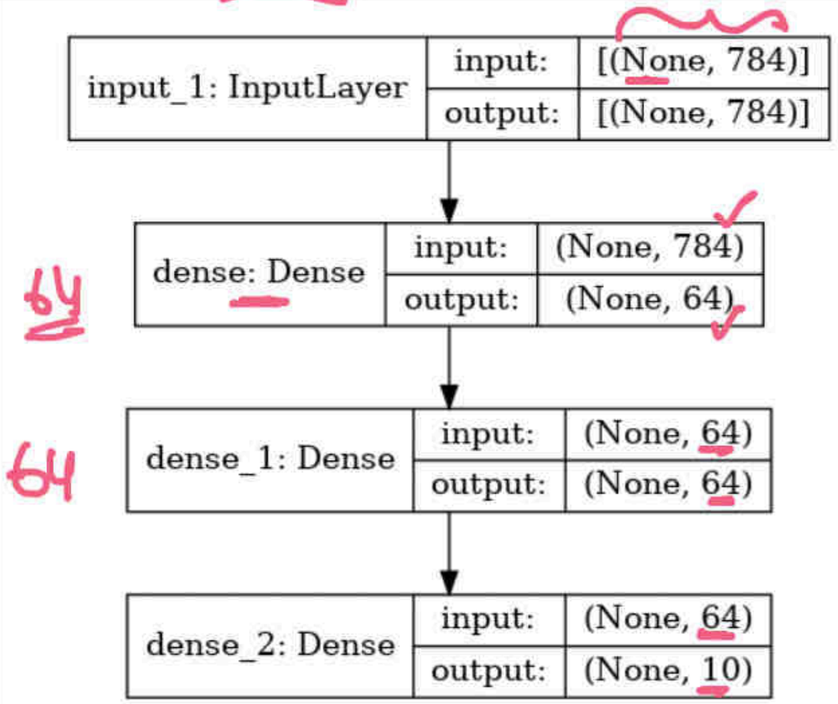
And, optionally, display the input and output shapes of each layer in the plotted graph:

```
keras.utils.plot_model(model, "my_first_model_with_shape_info.png", show_shapes=True)
```



And, optionally, display the input and output shapes of each layer in the plotted graph:

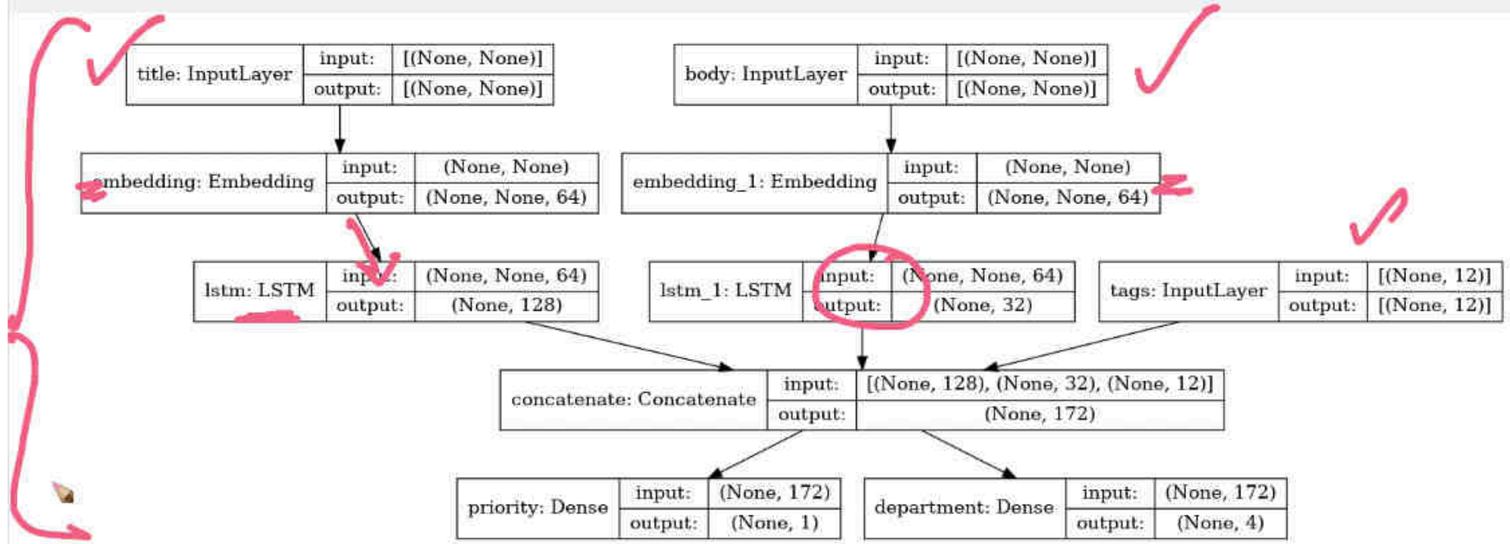
```
keras.utils.plot_model(model, "my_first_model_with_shape_info.png", show_shapes=True)
```



This figure and the code are almost identical. In the code version, the connection arrows are replaced by the call operation.

A "graph of layers" i way to create models that closely

```
keras.utils.plot_model(model, "multi_input_and_output_model.png", show_shapes=True)
```



When compiling this model, you can assign different losses to each output. You can even assign different weights to each loss -- to modulate their contribution to the total training loss.

```
model.compile(
    optimizer=keras.optimizers.RMSprop(1e-3),
    loss=[
        keras.losses.BinaryCrossentropy(from_logits=True),
        keras.losses.CategoricalCrossentropy(from_logits=True),
    ],
    loss_weights=[1.0, 0.2],
)
```

↳ 13 cells hidden

TF 2 and Keras: Functional API

```
[ ] print(x_train.shape)
```

```
(60000, 784)
```

```
[ ] inputs = keras.Input(shape=(784,))
    inputs.shape
```

```
TensorShape([None, 784])
```

```
[ ] # Functional API: much more intuitive
    # Refer: https://www.tensorflow.org/guide/keras/functional
    x1 = layers.Dense(hidden_units, activation="relu")(inputs)
    x2 = layers.Dense(hidden_units, activation="relu")(x1)
    outputs = layers.Dense(num_labels, activation="softmax")(x2)

    model = keras.Model(inputs=inputs, outputs=outputs, name="simple_model")
    model.summary()
```

```
[ ] inputs = keras.Input(shape=(784,))
    inputs.shape
```

```
TensorShape([None, 784])
```

```
[ ] # Functional API: much more intuitive
    # Refer: https://www.tensorflow.org/guide/keras/functional
    x1 = layers.Dense(hidden_units, activation="relu")(inputs)
    x2 = layers.Dense(hidden_units, activation="relu")(x1)
    outputs = layers.Dense(num_labels, activation="softmax")(x2)

    model = keras.Model(inputs=inputs, outputs=outputs, name="simple_model")
    model.summary()
```

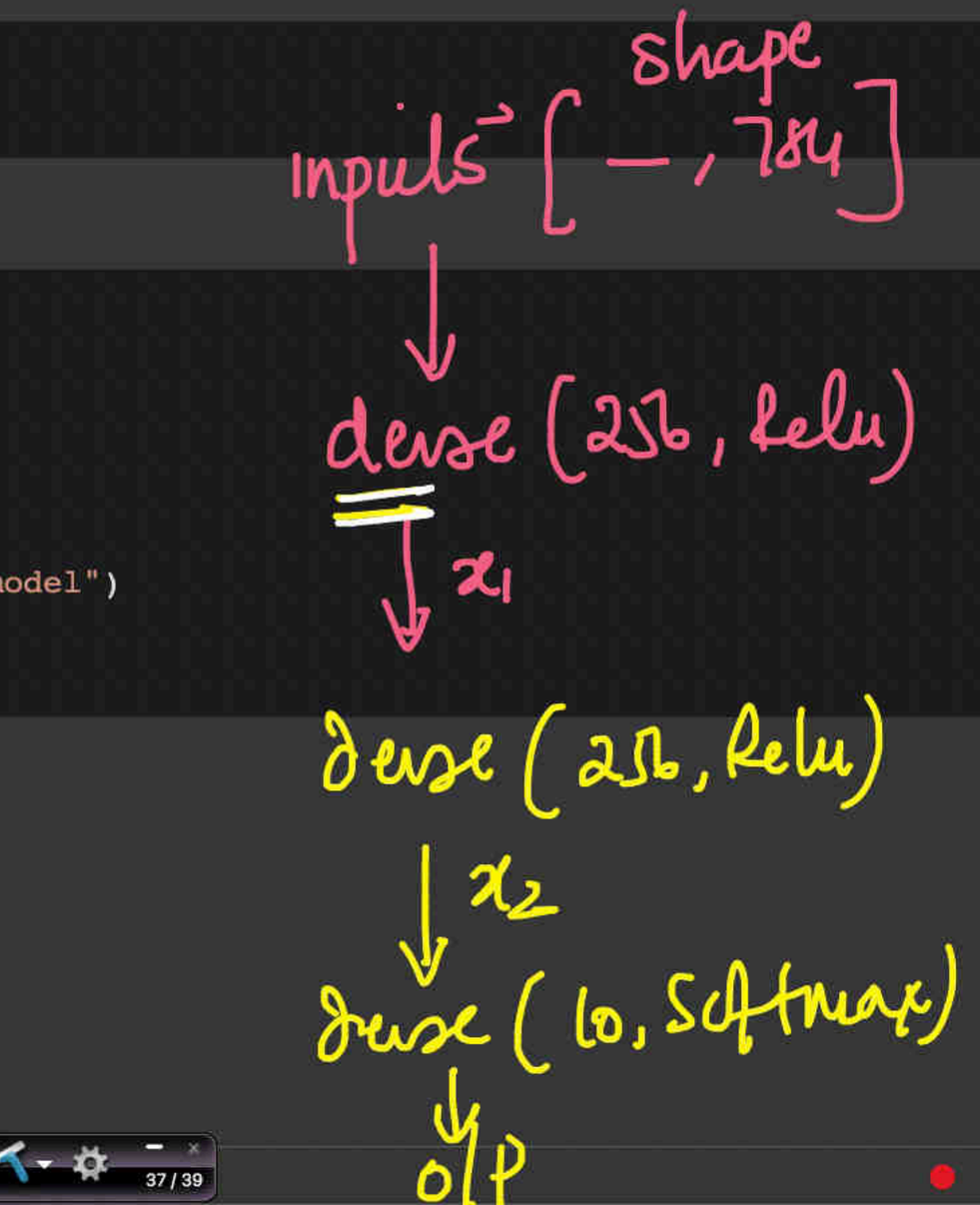
```
Model: "simple_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense_3 (Dense)             (None, 256)               200960

 dense_4 (Dense)             (None, 256)               65792
```

```
[ ] inputs = keras.Input(shape=(784,))
    inputs.shape
```

```
TensorShape([None, 784])
```

```
[ ] # Functional API: much more intuitive
    # Refer: https://www.tensorflow.org/guide/keras/functional
    x1 = layers.Dense(hidden_units, activation="relu")(inputs)
    x2 = layers.Dense(hidden_units, activation="relu")(x1)
    outputs = layers.Dense(num_labels, activation="softmax")(x2)

    model = keras.Model(inputs=inputs, outputs=outputs, name="simple_model")
    model.summary()
```

```
Model: "simple_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense_3 (Dense)             (None, 256)               200960

 dense_4 (Dense)             (None, 256)               65792
```



```
[ ] inputs = keras.Input(shape=(784,))
    inputs.shape
```

```
TensorShape([None, 784])
```

```
[ ] # Functional API: much more intuitive
    # Refer: https://www.tensorflow.org/guide/keras/functional
    x1 = layers.Dense(hidden_units, activation="relu")(inputs)
    x2 = layers.Dense(hidden_units, activation="relu")(x1)
    outputs = layers.Dense(num_labels, activation="softmax")(x2)

    model = keras.Model(inputs=inputs, outputs=outputs, name="simple_model")
    model.summary()
```

```
Model: "simple_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense_3 (Dense)             (None, 256)               200960

 dense_4 (Dense)             (None, 256)               65792
```

Dropout: Regularization through randomization

5 cells hidden

GPUs vs CPUs

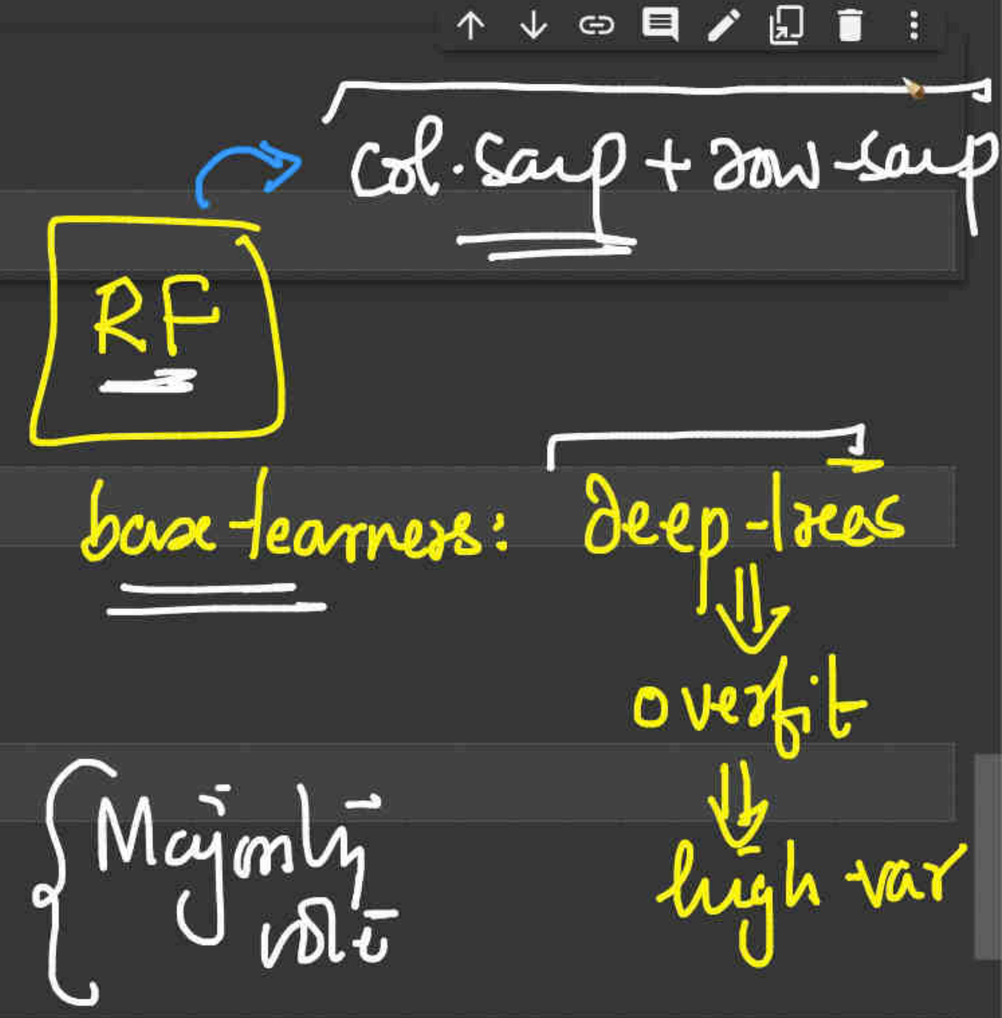
9 cells hidden

Tensorboard and Computational Graph

2 cells hidden

Batch Normalization

4 cells hidden



Dropout: Regularization through randomization

↳ 5 cells hidden

GPUs vs CPUs

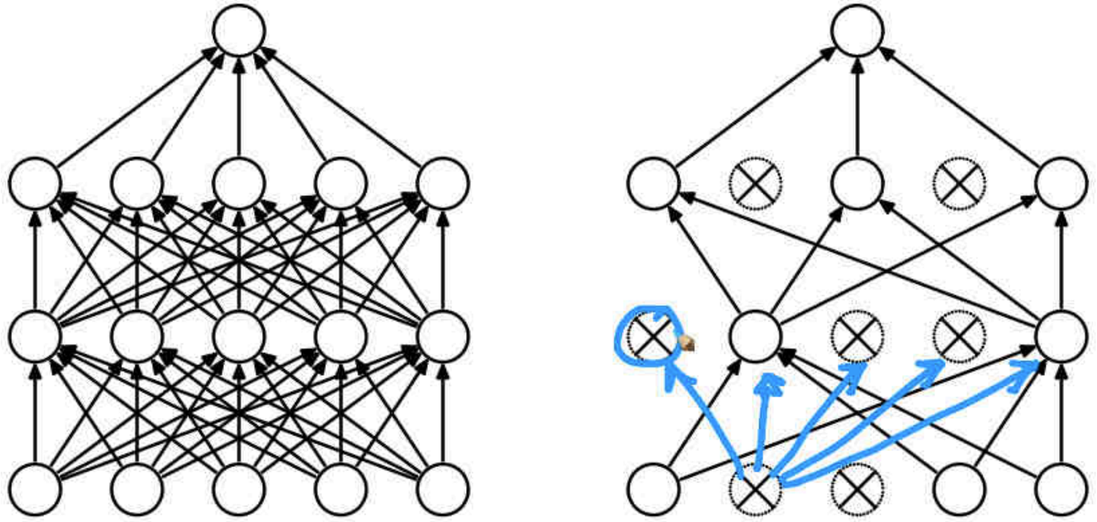
[ ] ↳ 9 cells hidden

Tensorboard and Computational Graph

[ ] ↳ 2 cells hidden

Batch Normalization

[ ] ↳ 4 cells hidden



(a) Standard Neural Net (b) After applying dropout.

randomly ignore/ drop some neurons

x ↑

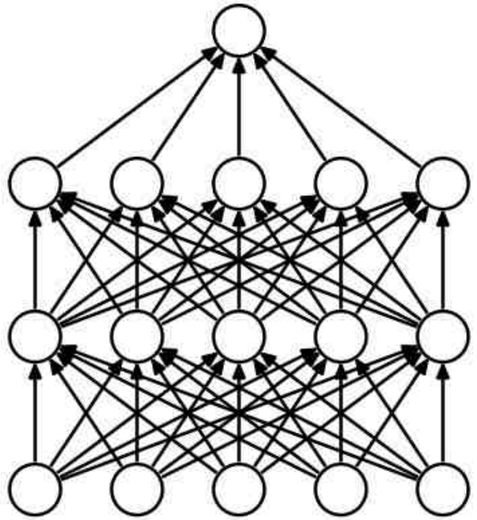
Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the pred hare parameters.

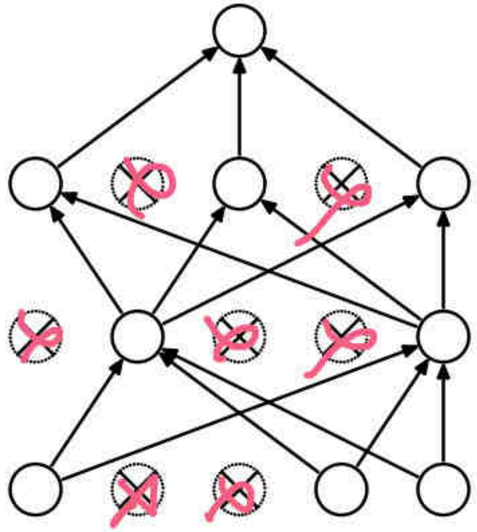
Train

FP & BP

learn



(a) Standard Neural Net



(b) After applying dropout.

drop-out rate = 0.2

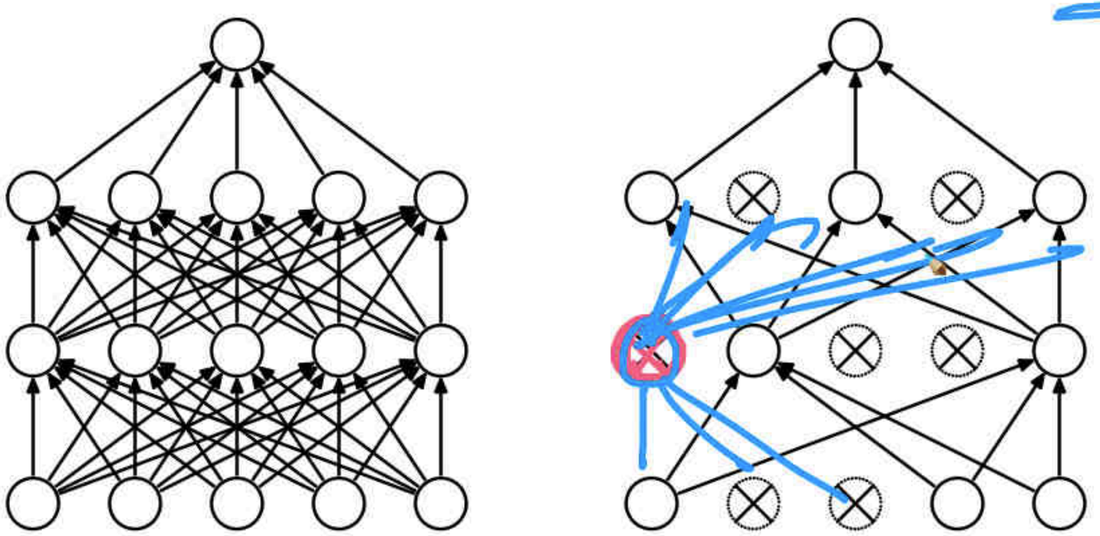
randomly

$r = 0 - 1.0$

$r \leq 0.2$

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the pred hare parameters.



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

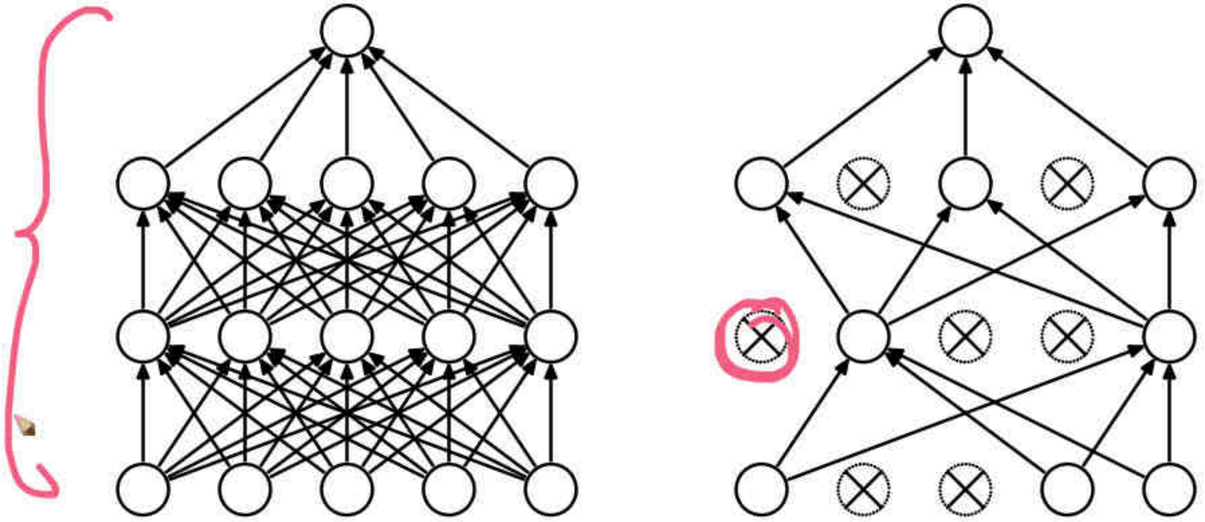
its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the pred hare parameters.

Train

r=20%

item: 2
FP+BP

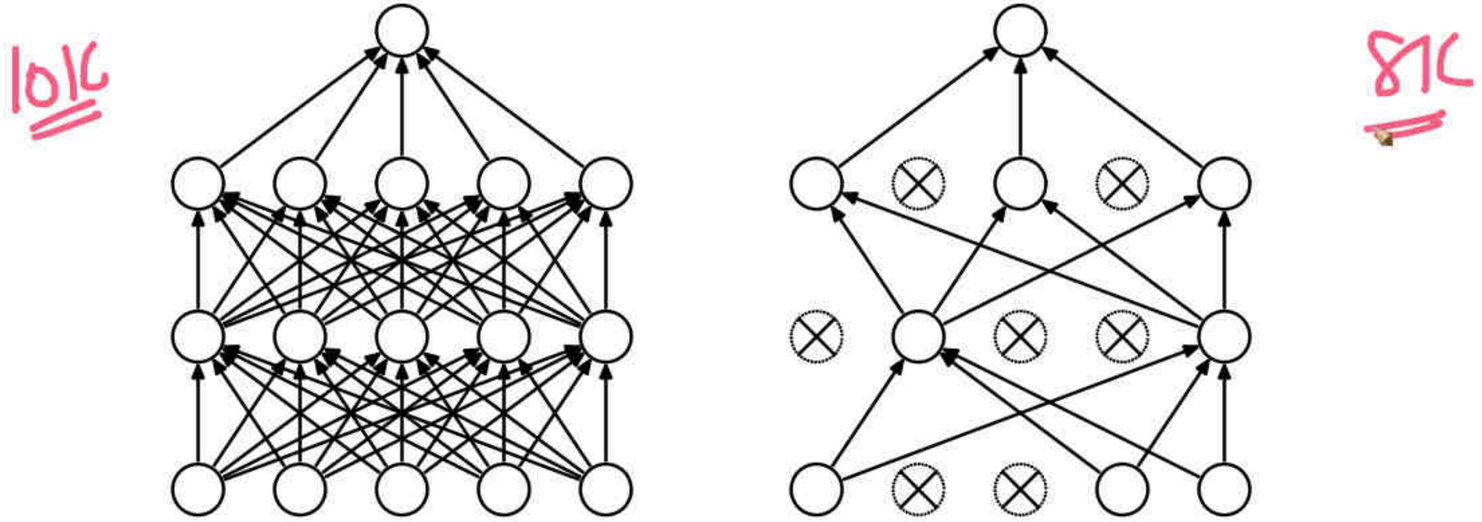
→ each item is faster
→ less chance of overfitting



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

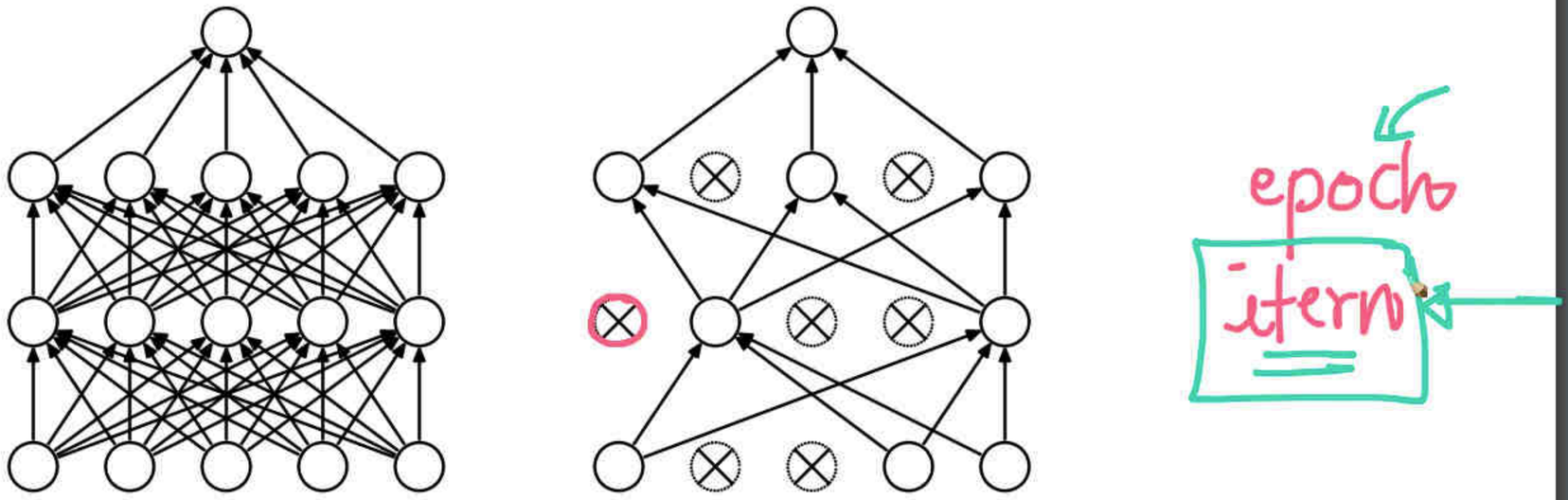
its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the pred hare parameters.



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

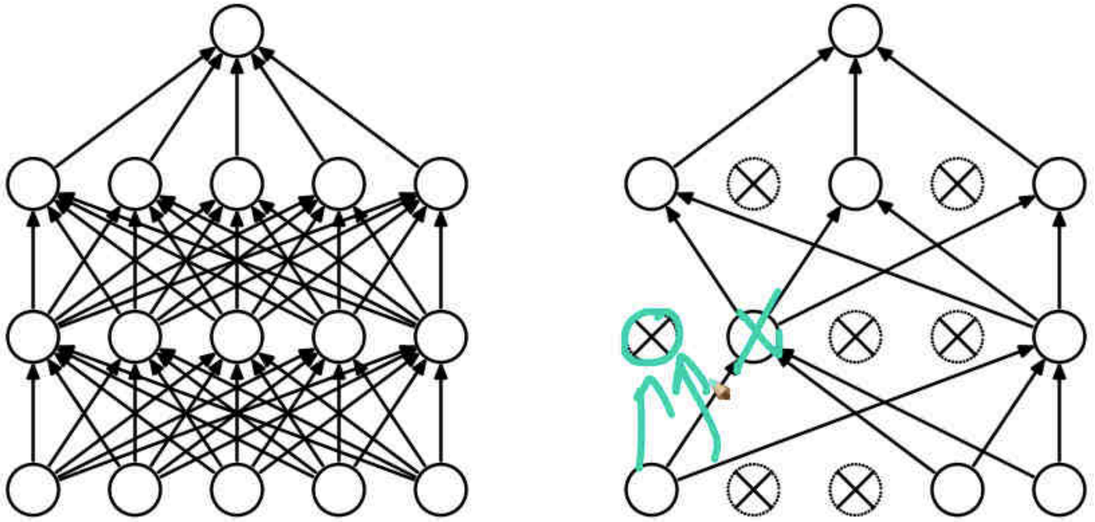
its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the pred hare parameters.



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the predi[illegible]hare parameters.

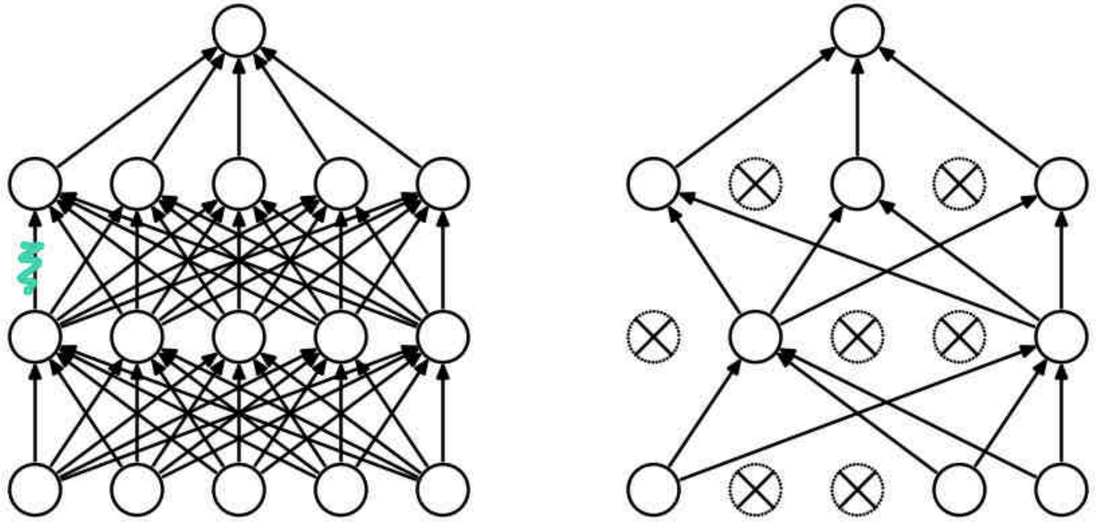


(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the predi[illegible]hare parameters.

γ=0.2



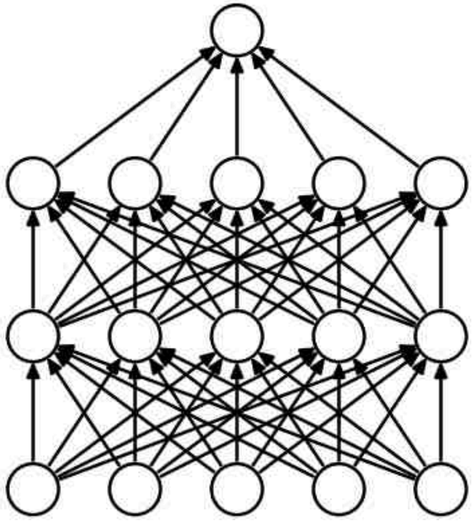
(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

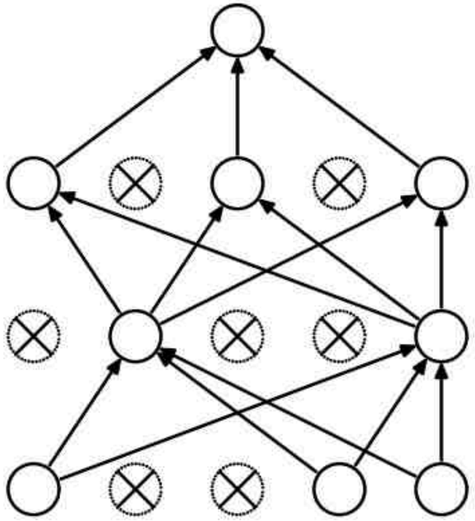
its posterior probability given the training data. This can sometimes be approximated quite well for simple or ih, 2008), but we

Train → FP → BP

Test/Eval



(a) Standard Neural Net



(b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

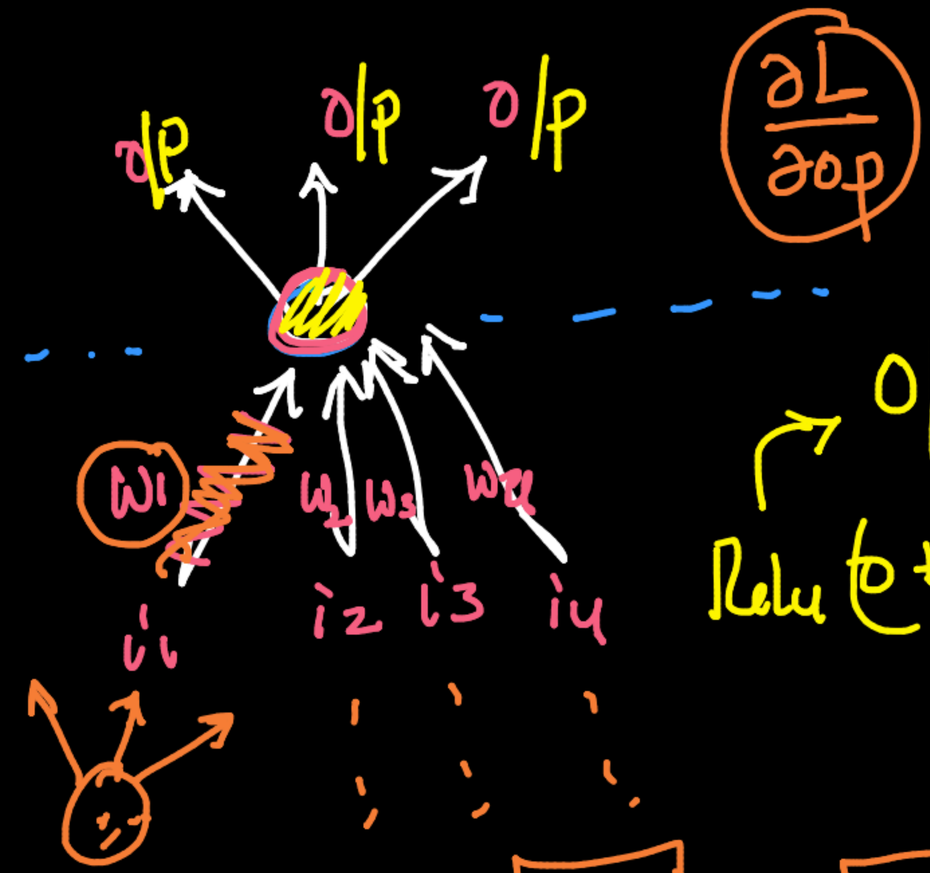
its posterior probability given the training data. This can sometimes be approximated quite well for simple or small models (Xiong et al., 2011; Salakhutdinov and Mnih, 2008), but we would like to approach the performance of the Bayesian gold standard using considerably less computation. We propose to do this by approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters.

Model combi[illegible]e learning meth-

Chrome File Edit View History Bookmarks Profiles Tab Window Help 10:37

Train

- FP:
- BP:



Dropout (edge-level)

o/p

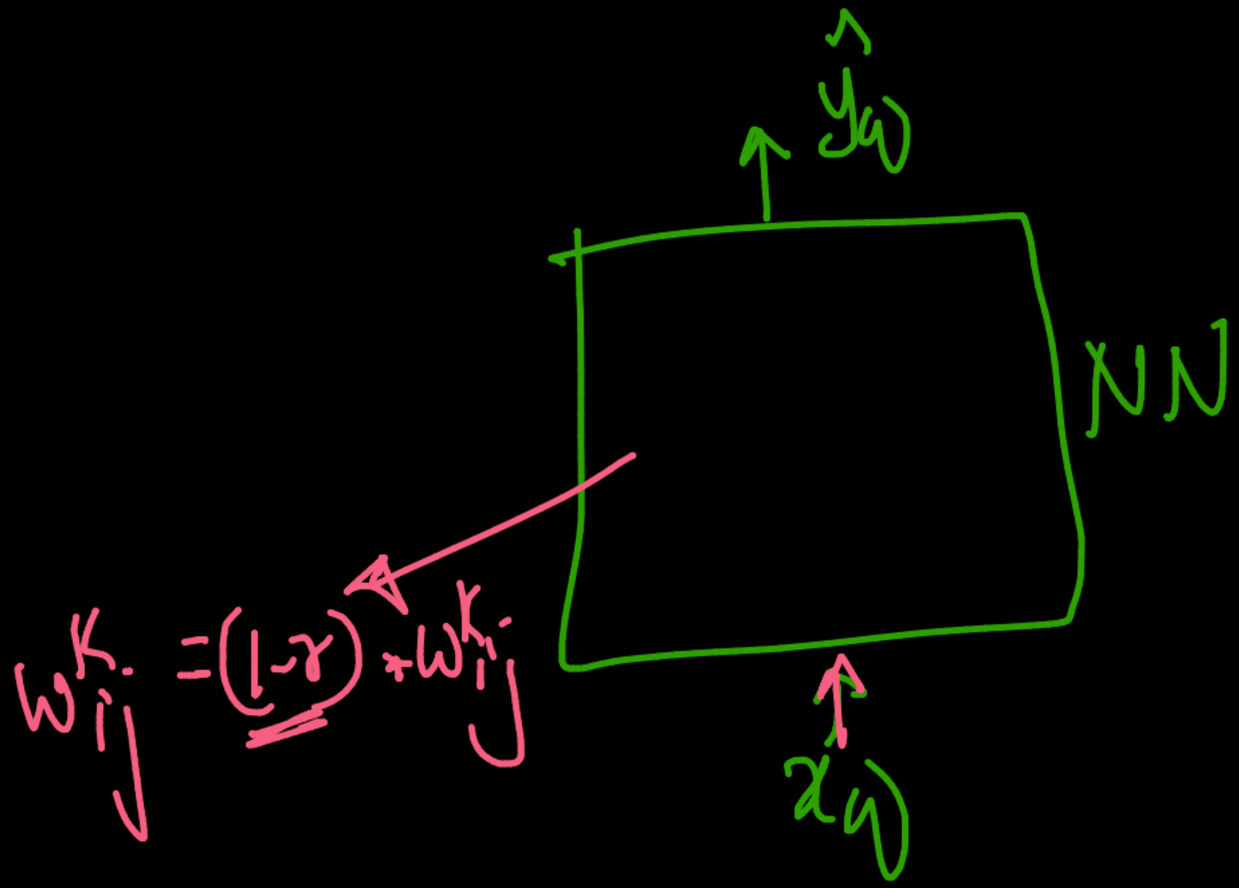
Relu (0 + i2w2 + i3w3 + i4w4)

w_1 is not updated: $\frac{\partial L}{\partial op}$ ... are not backpropagated on w_1 edge

50 / 52

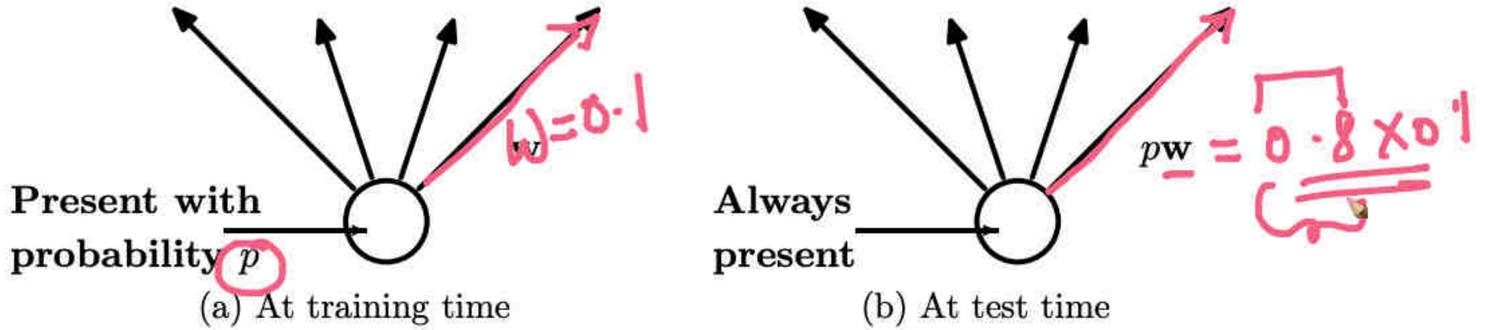
Test:

$r = 0.2$



$w_{ij}^{k} = (1-r) * w_{ij}^{k}$

do NOT skip/loop any edge



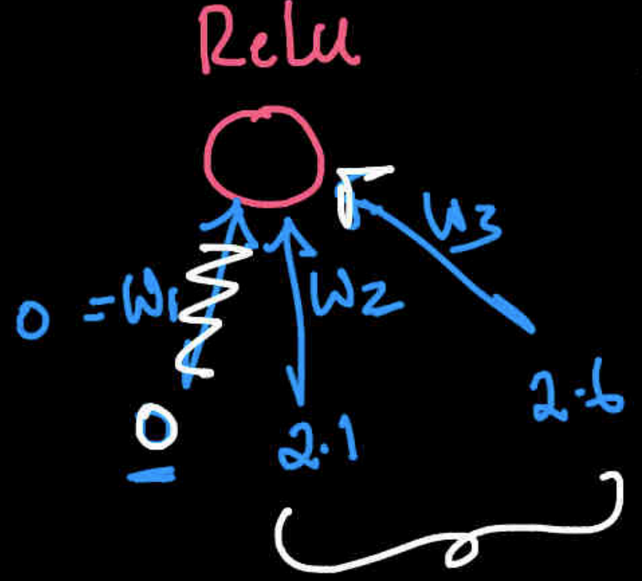
(a) At training time (b) At test time

Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Applying dropout to a neural network amounts to sampling a "thinned" network from it. The thinned network consists of all the units that survived dropout (Figure 1b). A neural net with n neural networks. These networks

Q

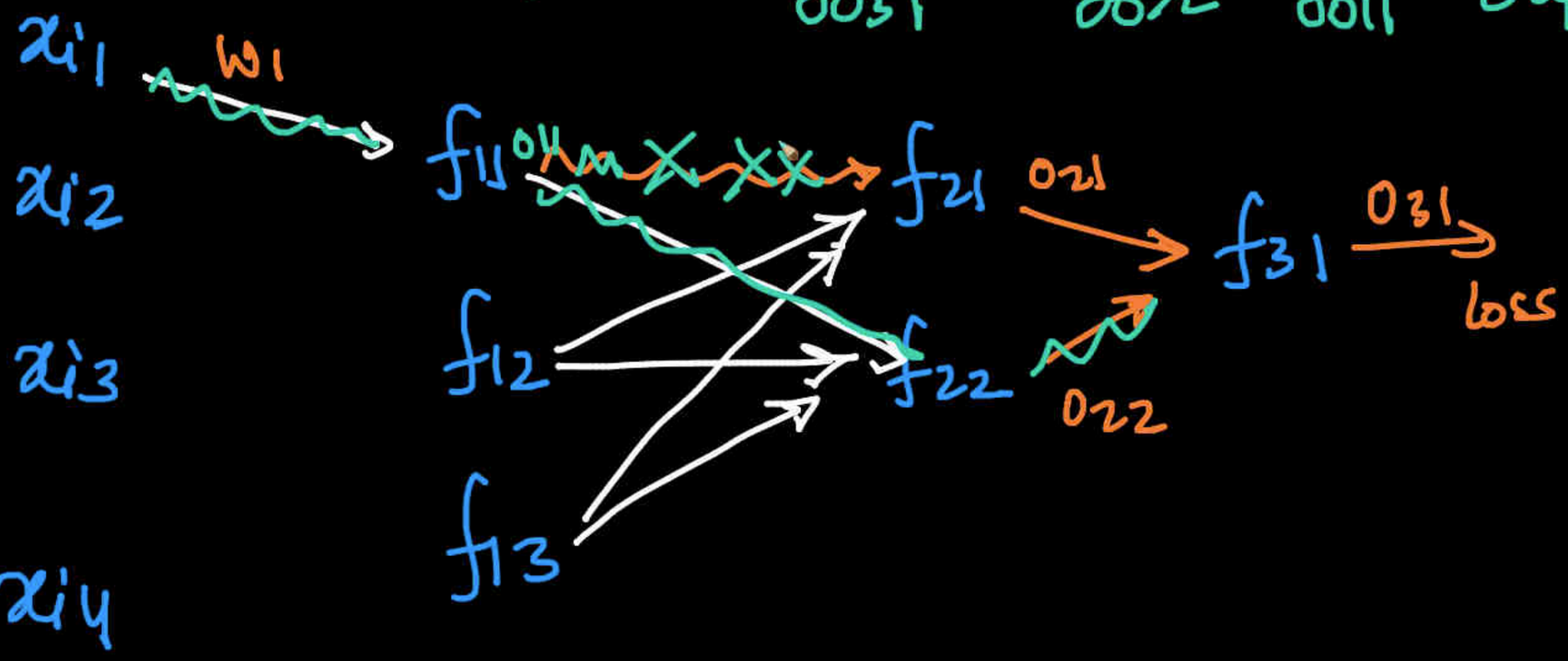
$\gamma = 0.1 - 0.6$

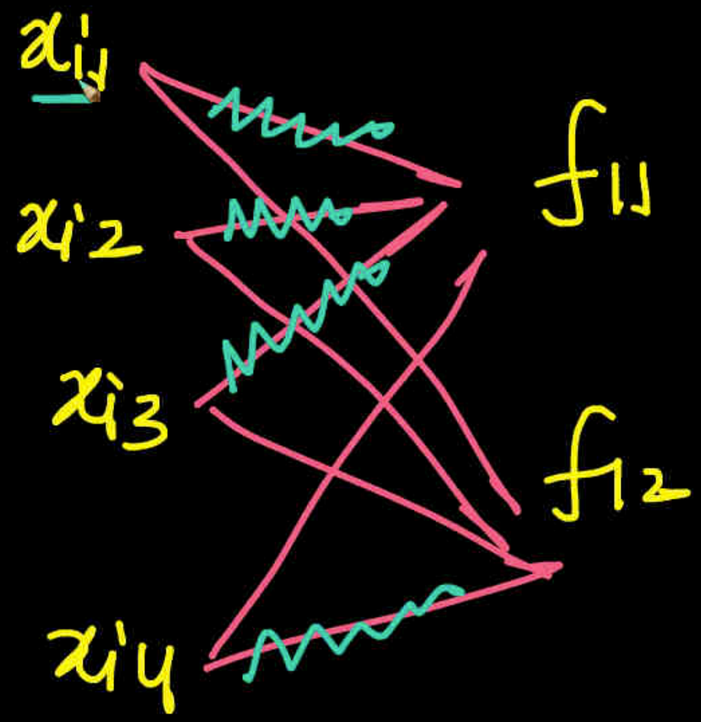


$\text{Relu}(0 + 2.1 \times W_2 + 2.6 \times W_3)$

Q

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial o_{31}} \cdot \frac{\partial o_{31}}{\partial o_{22}} \frac{\partial o_{22}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w}$$





$\gamma = 0.5$

ets.

In order to test the robustness of ropout, classification experiments were one with networks of many different ar- hitectures keeping all hyperparameters, in- uding p, fixed. Figure 4 shows the test rror rates obtained for these different ar- hitectures as training progresses. The ame architectures trained with and with- ut dropout have drastically different test rrors as seen as by the two separate clus- ers of trajectories. Dropout gives a huge nprovement across all architectures, with- ut using hyperparameters that were tuned pecifically for each architecture.

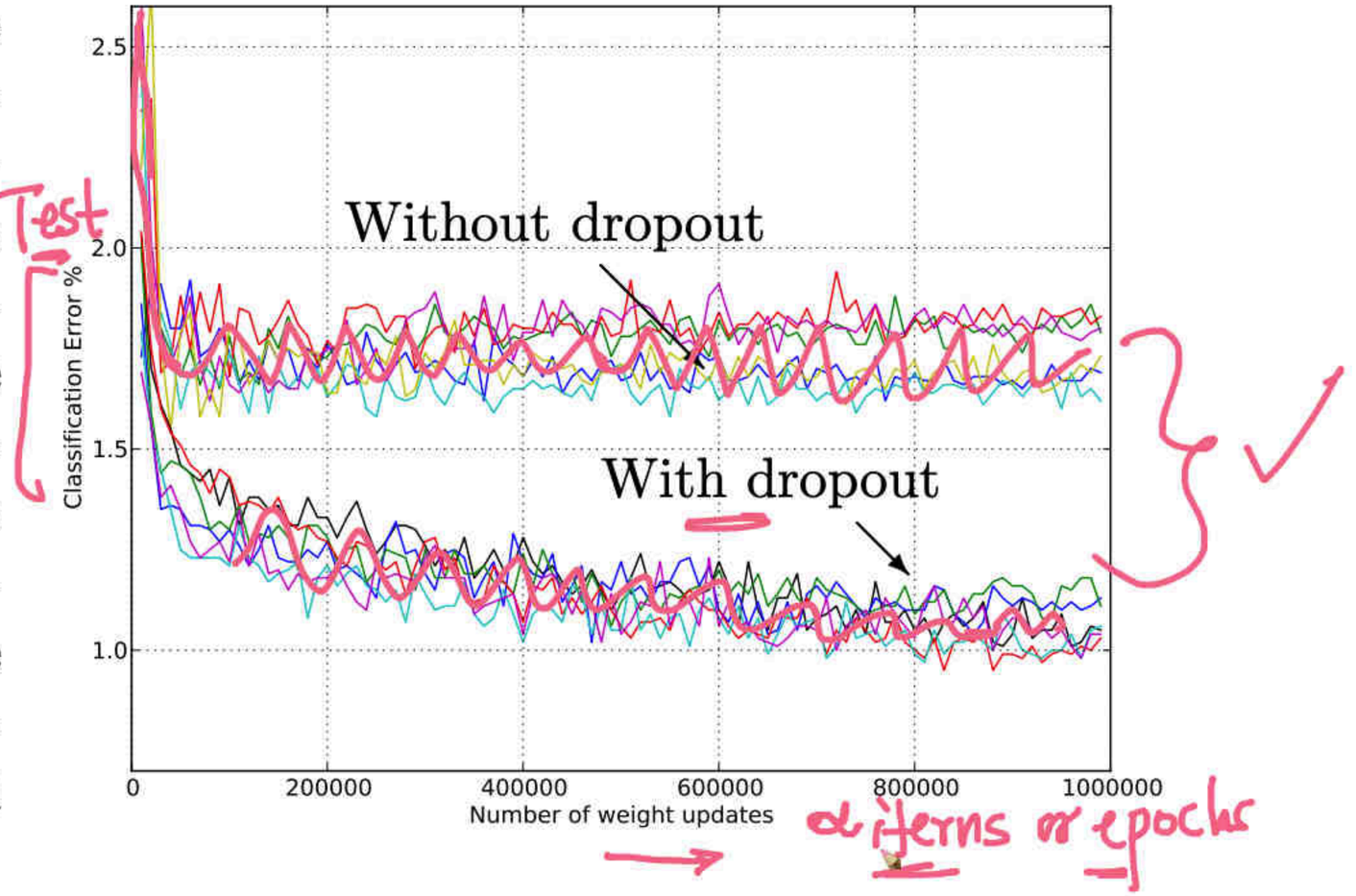


Figure 4: Test error for different architectures

.1.2 Street View House Nu t dropout. The net-

Dropout: Regularization through randomization

```
# Model with Dropout
x = layers.Dense(hidden_units, activation="relu")(inputs)
x = Dropout(0.3)(x)
x = layers.Dense(hidden_units, activation="relu")(x)
x = Dropout(0.3)(x)
outputs = layers.Dense(num_labels, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs, name="simple_model")
model.summary()
```

```
Model: "simple_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense_6 (Dense)             (None, 256)               200960

 dropout (Dropout)           (None, 256)               0

 dense_7 (Dense)             (None, 256)               65792
```

Dropout: Regularization through randomization

```
# Model with Dropout
x = layers.Dense(hidden_units, activation="relu")(inputs)
x = Dropout(0.3)(x)
x = layers.Dense(hidden_units, activation="relu")(x)
x = Dropout(0.3)(x)
outputs = layers.Dense(num_labels, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs, name="simple_model")
model.summary()
```

```
Model: "simple_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense_6 (Dense)             (None, 256)               200960

 dropout (Dropout)           (None, 256)               0

 dense_7 (Dense)             (None, 256)               65792
```

```
Epoch 23/30
469/469 [==============================] - 1s 3ms/step - loss: 0.0253 - accuracy: 0.9915
Epoch 24/30
469/469 [==============================] - 2s 3ms/step - loss: 0.0272 - accuracy: 0.9913
Epoch 25/30
469/469 [==============================] - 2s 3ms/step - loss: 0.0259 - accuracy: 0.9911
Epoch 26/30
469/469 [==============================] - 2s 3ms/step - loss: 0.0240 - accuracy: 0.9917
Epoch 27/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0224 - accuracy: 0.9926
Epoch 28/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0224 - accuracy: 0.9927
Epoch 29/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0227 - accuracy: 0.9924
Epoch 30/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0219 - accuracy: 0.9924
CPU times: user 54 s, sys: 6.47 s, total: 1min
Wall time: 1min 22s
<keras.callbacks.History at 0x7fec60036c50>
```

```
[ ] # test dataset  performance
    loss, acc = model.evaluate(x_test,
                               y_test,
                               batch_size=batch_size,
                               verbose=0)
    print("\nTest accuracy: %.1f%%" % (100.0 * acc))
    print(loss)
```

```
469/469 [==============================] - 2s 4ms/step - loss: 0.0224 - accuracy: 0.9926
Epoch 28/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0224 - accuracy: 0.9927
Epoch 29/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0227 - accuracy: 0.9924
Epoch 30/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0219 - accuracy: 0.9924
CPU times: user 54 s, sys: 6.47 s, total: 1min
Wall time: 1min 22s
<keras.callbacks.History at 0x7fec60036c50>
```

```
[ ] # test dataset  performance
    loss, acc = model.evaluate(x_test,
                               y_test,
                               batch_size=batch_size,
                               verbose=0)
    print("\nTest accuracy: %.1f%%" % (100.0 * acc))
    print(loss)

    # Overfit or Underfit?
    # Can we add more layers?
```

```
Test accuracy: 98.4%
0.07712692767381668
```

Q

$\gamma = 0.2$

w'_{11}

x_{i1} [illegible] f_{11}

x_{i2} f_{12}

x_{i3} f_{13}

x_{i4}

$w'_{4\times3}$

w'_{11} : init to randomly value

itro:

w'_{11} : 0.1 (let)

iter 1:

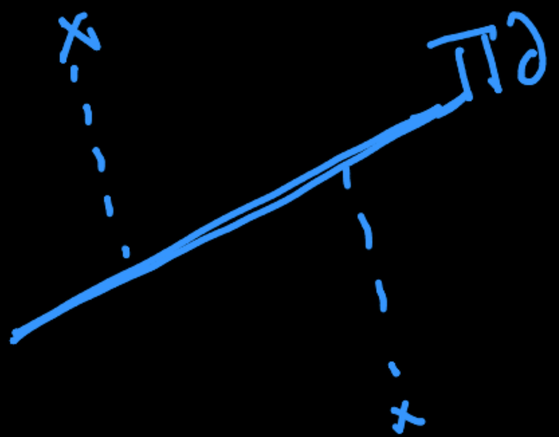
$w'_{11} = 0.1$ (no update)

iter 2:

$w'_{11} = 0.1 - \eta \frac{\partial L}{\partial w'_{11}}$

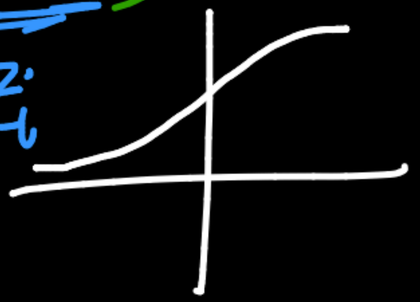
Q

log-reg :- decision boundary :- linear

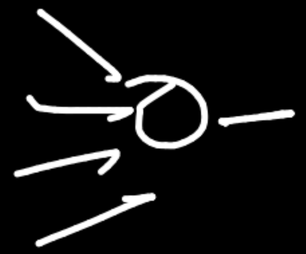


$\sigma(w^T x_i + b) \rightarrow$ 0 to 1

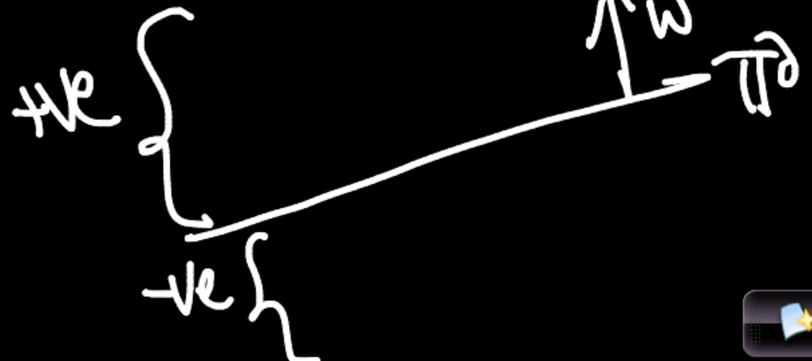
z_i



$P(y_i = 1 \mid x_i, w, b)$

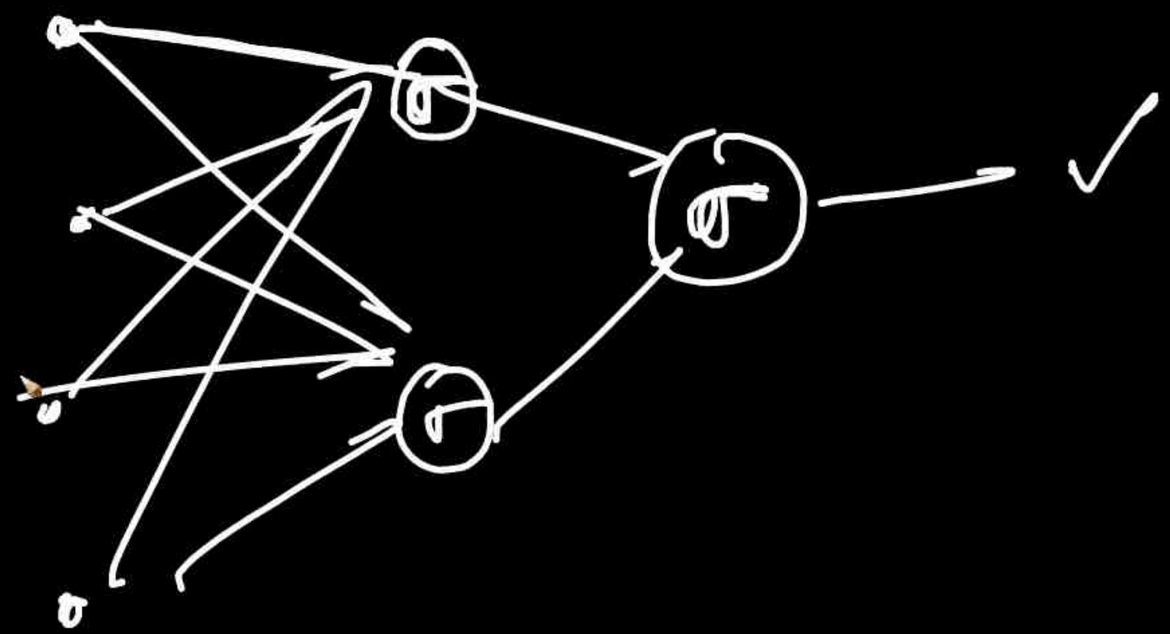


Perceptron :-



$P(w^T x_i + b)$

$P(z_i) \begin{cases} 1 & z_i > 0 \\ 0 & 0/w \end{cases}$



Q

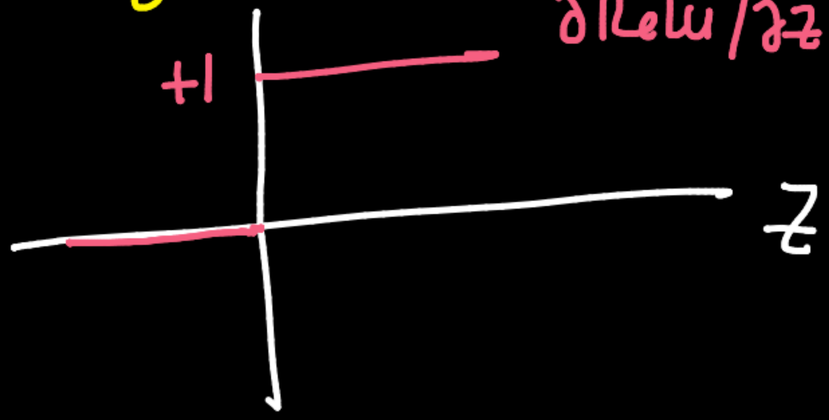
$$Relu(z) = \max(0, z)$$

$z = 3x+5$

$$\frac{\partial\, Relu(3x+5)}{\partial x} = \frac{\partial\, Relu(z)}{\partial z} \cdot \frac{\partial (3x+5)}{\partial x}$$

$z \to 0$

$\partial Relu / \partial z$



$= +1 \cdot 3$

$= +3$ ✓

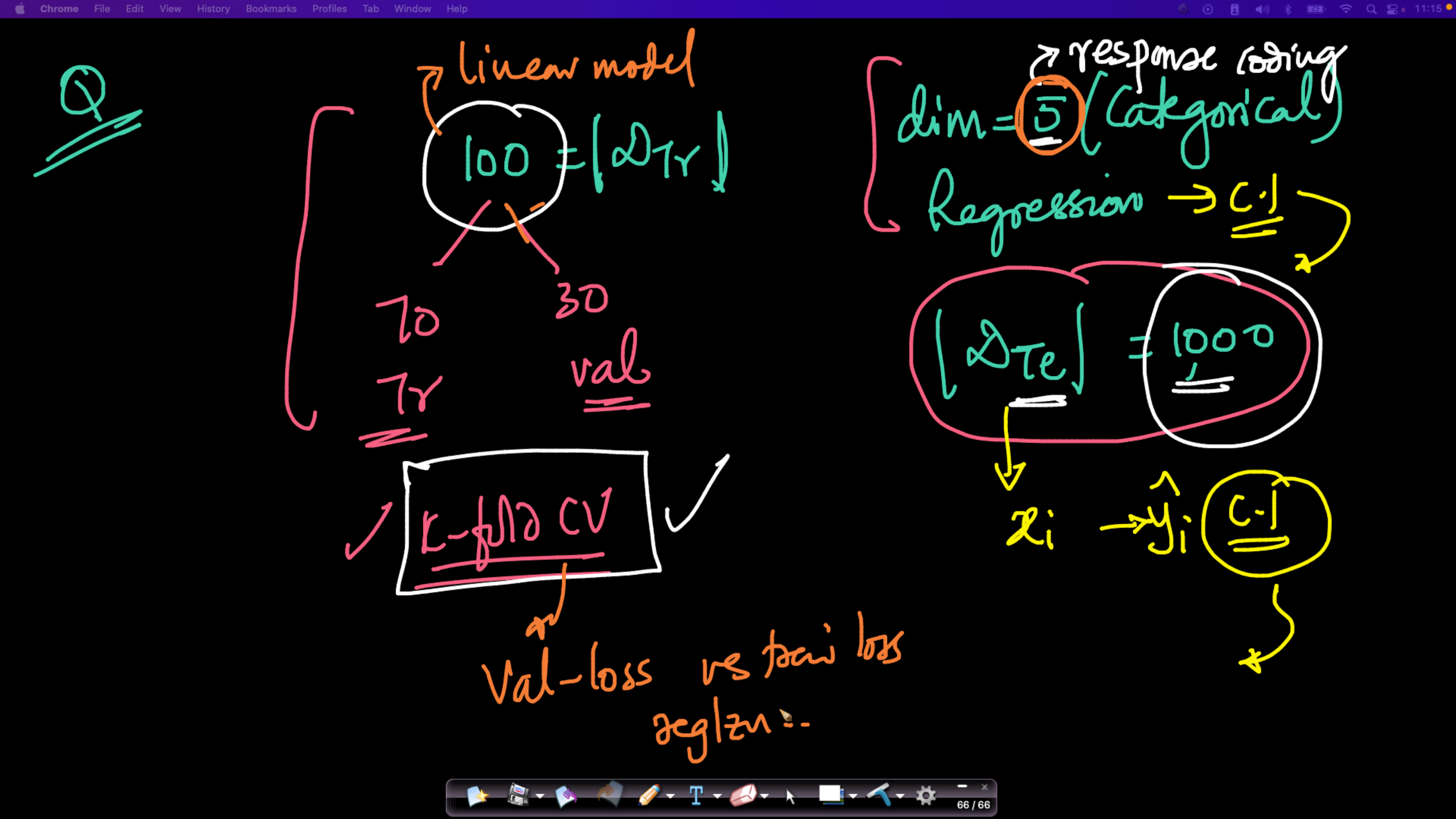
Q

OHE ← C_1 - - - - C_{100}

100 -class classfn → regression

[Tree-based →

[Linear


Q
linear model
100 = |Dtr|
70
30
Tr
val
K-fold CV
Val-loss vs train loss
response coding
dim = 5 (Categorical)
Regression → C.I
|Dte| = 1000
xi → yi
C.I

Q

$X^1_{n \times 2}$

↓

2-classes

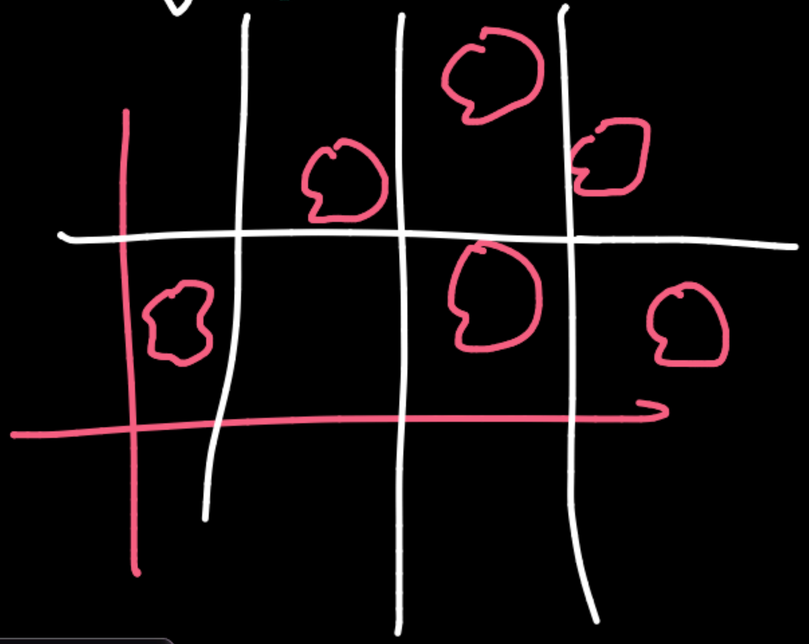
(50-50)

$X^2_{n \times 2}$

↓

20-classes

(5-5-----5)



```
Test accuracy: 98.4%
0.07712692767381668
```

[ ]

GPUs vs CPUs

CPU

[ ] ↳ 4 cells hidden

GPU

↳ 3 cells hidden

Tensorboard and Computational Graph