Refer NB “6 CNN_CIFAR.ipynb”

Results were not too good. OK OK....

Data Augmentation

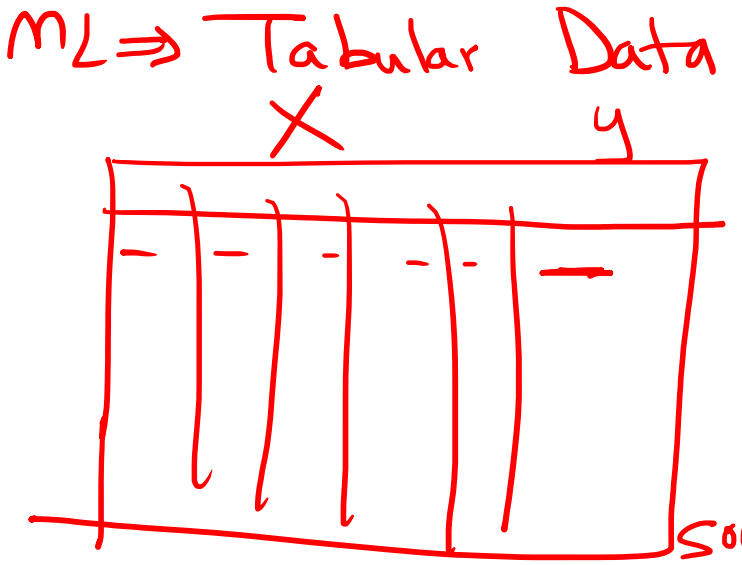
Improve our model ⟹ Data Augmentation

Translation Invariance : Humans can immediately identify this, but it is difficult for a computer / NN.

Cat

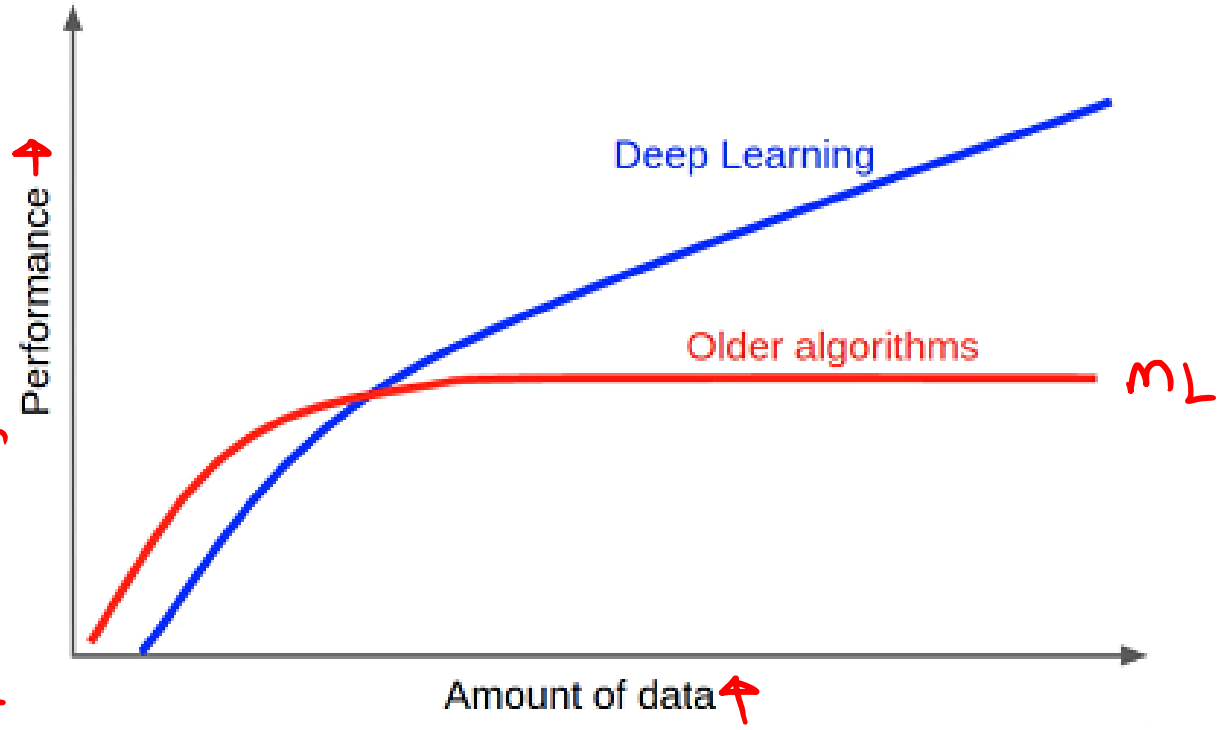
Same Cat

Why data is important?



Grant: Yes | No
email: Spam | Ham

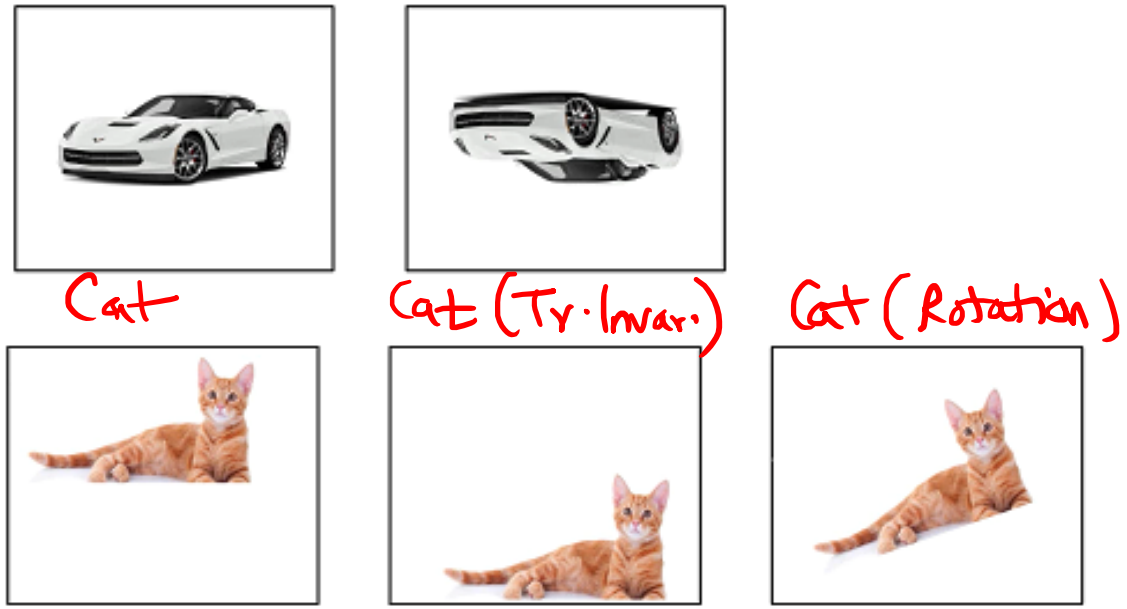
} u cannot create ur own data

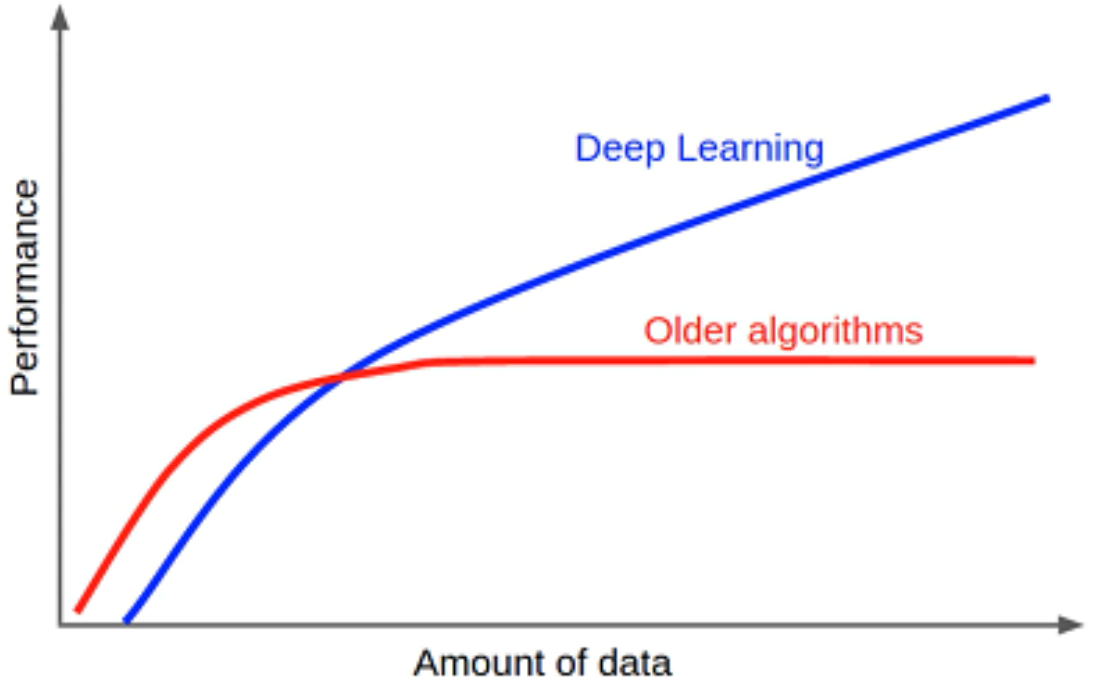


If I try to create the new data, it will be contaminated with my own biased opinions.

Why data is important?

Its perfectly fine to invent new data with images. Its meaning still remain the same.
And no biases will added.





Problem with this approach

Data takes up space

The more data I invent, the more space it takes up!

There are an endless number of ways I can invent new data

Shift to the left by 1, 2, 3, 4, 5, ... pixels

Right / up / down also

Rotation

Hyperparameter Optimization!!!

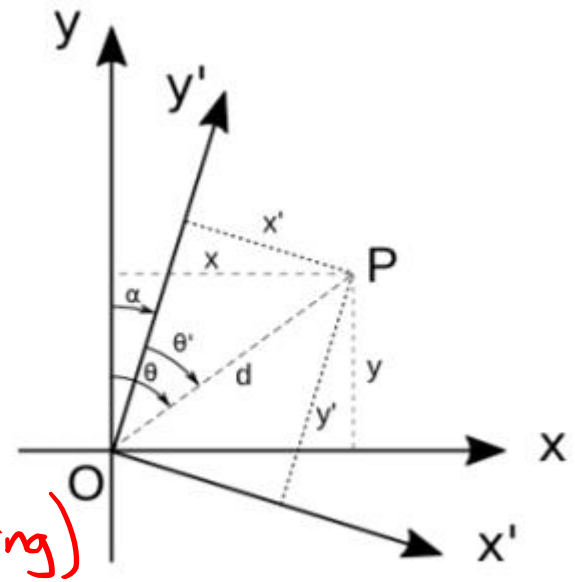
This can be done all automatically!

Have you ever considered how to rotate an image?

Use Tensorflow's Keras API instead

We need to know about generators / iterators

Data Augmentation
2hrs Youtube



Gen/Iterators: Play/Pause Button (A Song)

Walk (earphones - songs) ---play---> Uncle ---Pause---> Play ---> Friend ---Pause---> Play ---> ...repeat

Generators / Iterators

Loop from 0...10: `for i in range(10)`

In Python 2, `range(10) = [0,1,2,3,4,5,6,7,8,9] # a list!`

In Python 2, use `xrange(10) # does NOT create a list`

In Python 3, `print(range(10))` yields `range(0,10)`

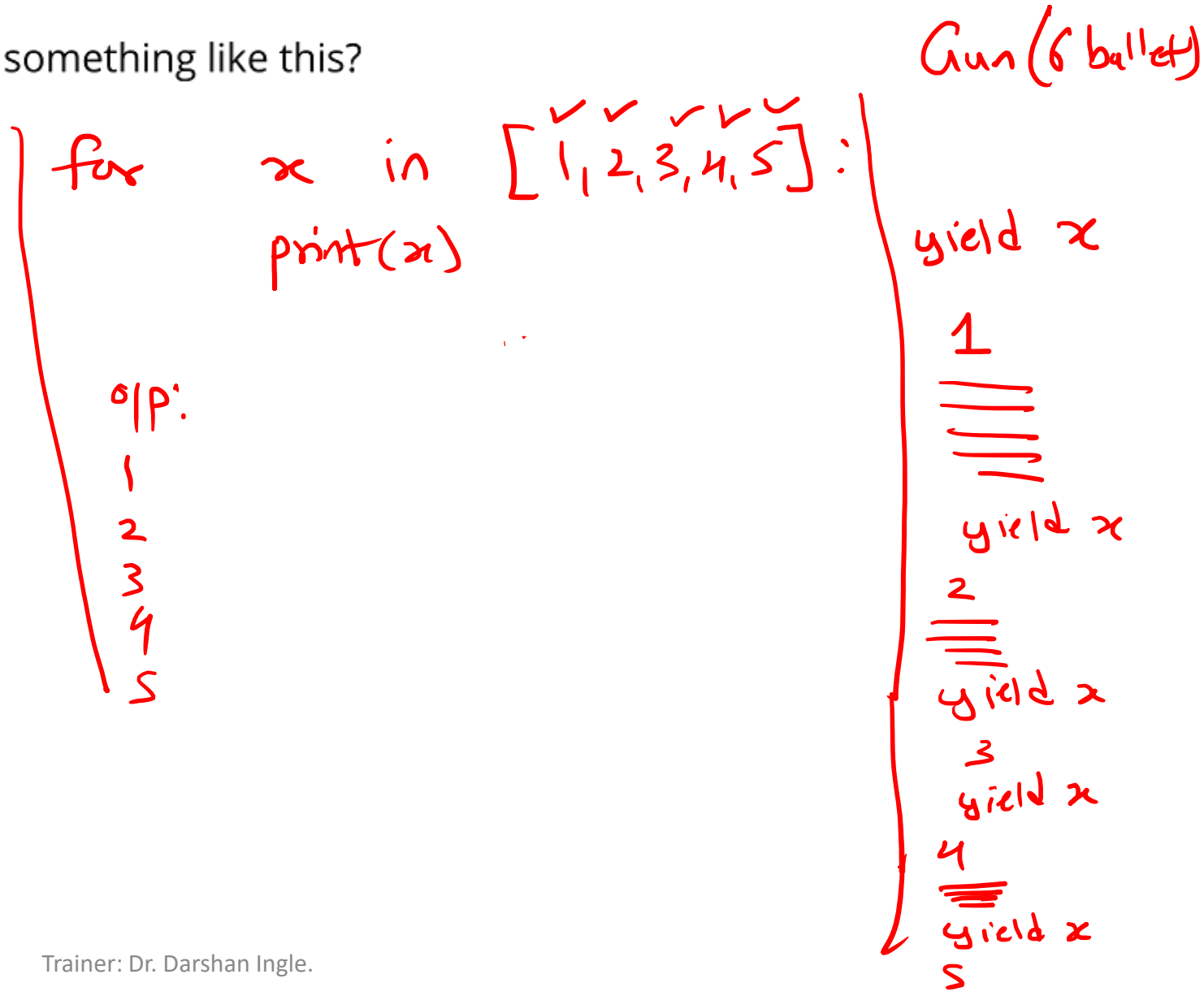
- Not a list!

```
for i in range(10):
    print(i)
```

Create your own generator

- Can you write your own function to do something like this?

```
for x in my_random_generator():
  print(x)



# 0.06654313
# -0.68315371
# 1.46795401
# -0.9017639
# 0.77572637
```



Yield

```
def my_random_generator():
  for i in range(10):
    x = np.random.randn()
    yield x
```

Notice: no list is ever created

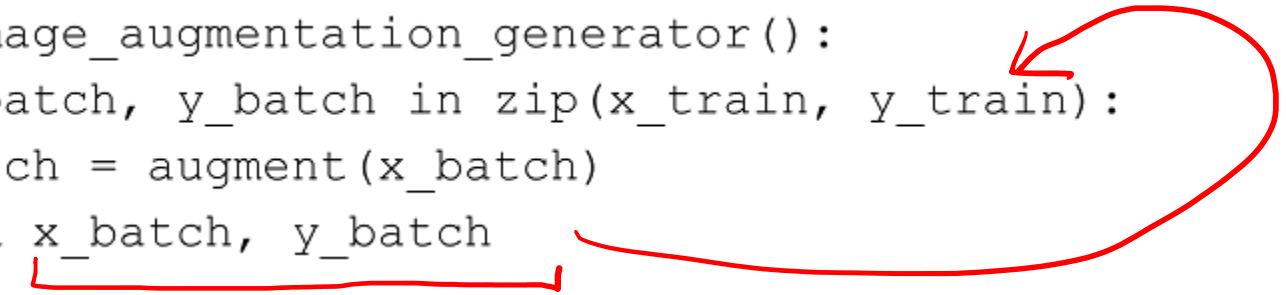
All values do not need to be stored in memory simultaneously

Apply this to Data Augmentation

You could similarly generate augmented data on the fly

Conceptually, think of it like this

```
def my_image_augmentation_generator():
  for x_batch, y_batch in zip(x_train, y_train):
    x_batch = augment(x_batch)
    yield x_batch, y_batch
```



How does it work in tf.keras?

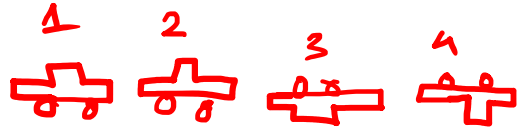
```
from tensorflow.keras.preprocessing.image import
ImageDataGenerator

data_generator = ImageDataGenerator(
  width_shift_range=0.1,
  height_shift_range=0.1,
  horizontal_flip=True)
```

Other args: `rotation_range, width_shift_range, height_shift_range, brightness_range, shear_range, zoom_range, horizontal_flip, and vertical_flip`

How does it work in tf.keras?

```
data_generator = ImageDataGenerator(...)
train_generator = data_generator.flow(
  x_train, y_train, batch_size)
```

```
steps_per_epoch = x_train.shape[0] // batch_size
r = model.fit_generator(
  train_generator,
  steps_per_epoch=steps_per_epoch,
  epochs=50)
```

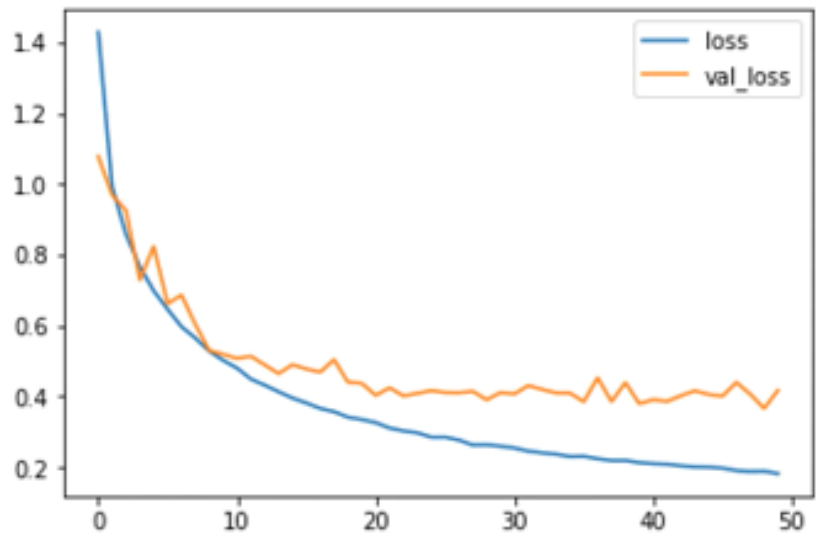


Business as usual

`fit_generator` returns history, so plot loss per iteration, etc.

```
# Plot loss per iteration
import matplotlib.pyplot as plt
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.legend()
```

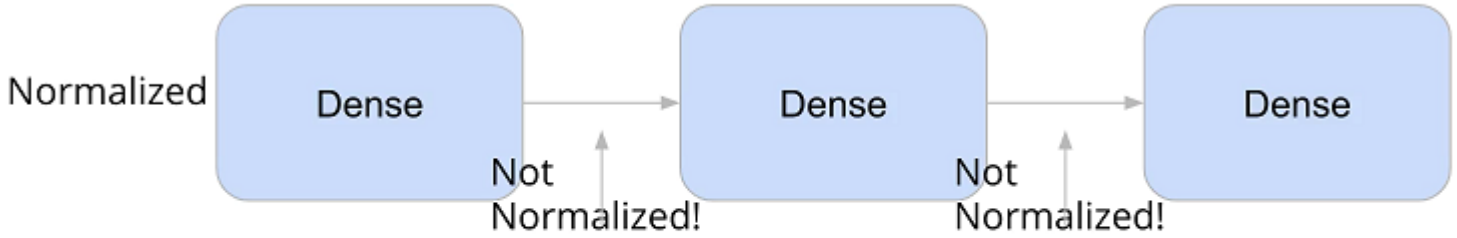
```
<matplotlib.legend.Legend at 0x7f92f7ca6908>
```



Batch Normalization

x_train /255.0
y_train/ 255.0

- Early on, we noted that it's important to normalize/standardize data before passing it into algorithms like linear/logistic regression → Standard/ MinMax/ Robust Scaling
- Problem: because this operation is done only on *input data*, only the first layer sees normalized data (after being transformed by Dense layer it's no longer normalized)



$$z = (x - \mu)/\sigma$$

∴ Go for Batch Normalization.

Batch Normalization

- To start, recall: In Tensorflow, when we call `model.fit()`, we are doing **batch** gradient descent

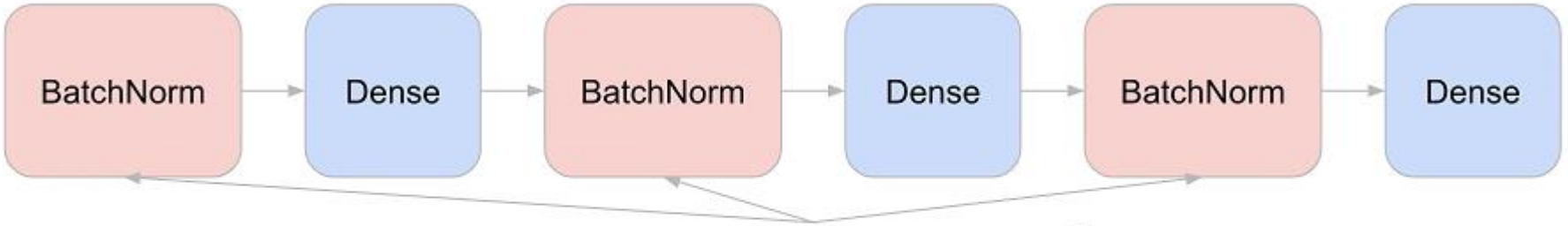
```
for epoch in range(epochs):
    for x_batch, y_batch in next_batch(x_train, y_train):
        w ← w - learning_rate * grad(x_batch, y_batch)
```

1 // looping through each epoch

2 // we look at a chunk of data on each step & do G.D. w.r.t. this chunk/batch of data.

Batch Normalization

- What if we had a *layer* that would look at each batch, calculate the mean and standard deviation on the fly, and standardize based on that?



$$z = (x - \mu)/\sigma$$

mean S.D

Batch Norm as Regularization

Problem of Overfitting ↓

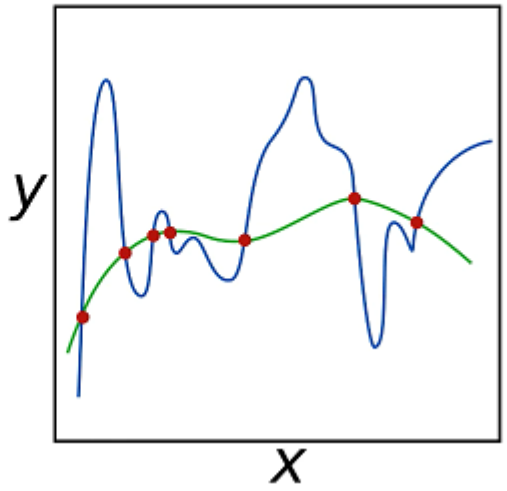
Can help with overfitting

Since every batch is slightly different, you'll get a slightly different μ_B, σ_B

They are not the true mean / std of the whole dataset

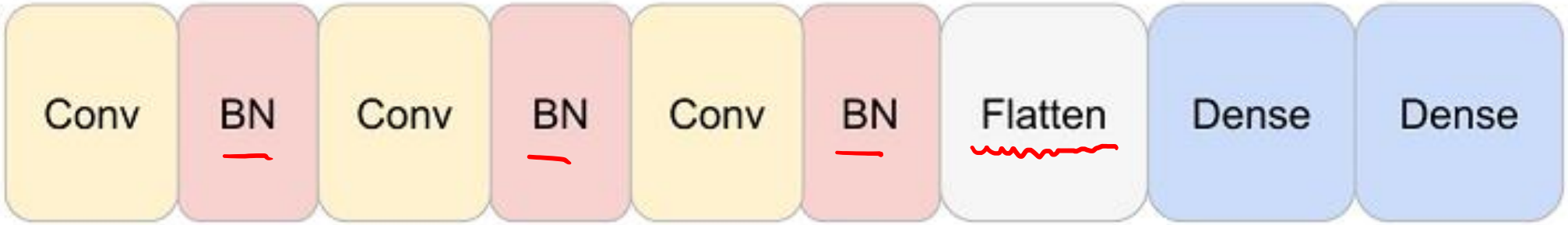
This is essentially noise, and using noise during training makes the neural network impervious to noise

- Rather than fitting to the noise! (a.k.a. "overfitting")



Where is Batch Norm used?

In b/w Convolutional Layers



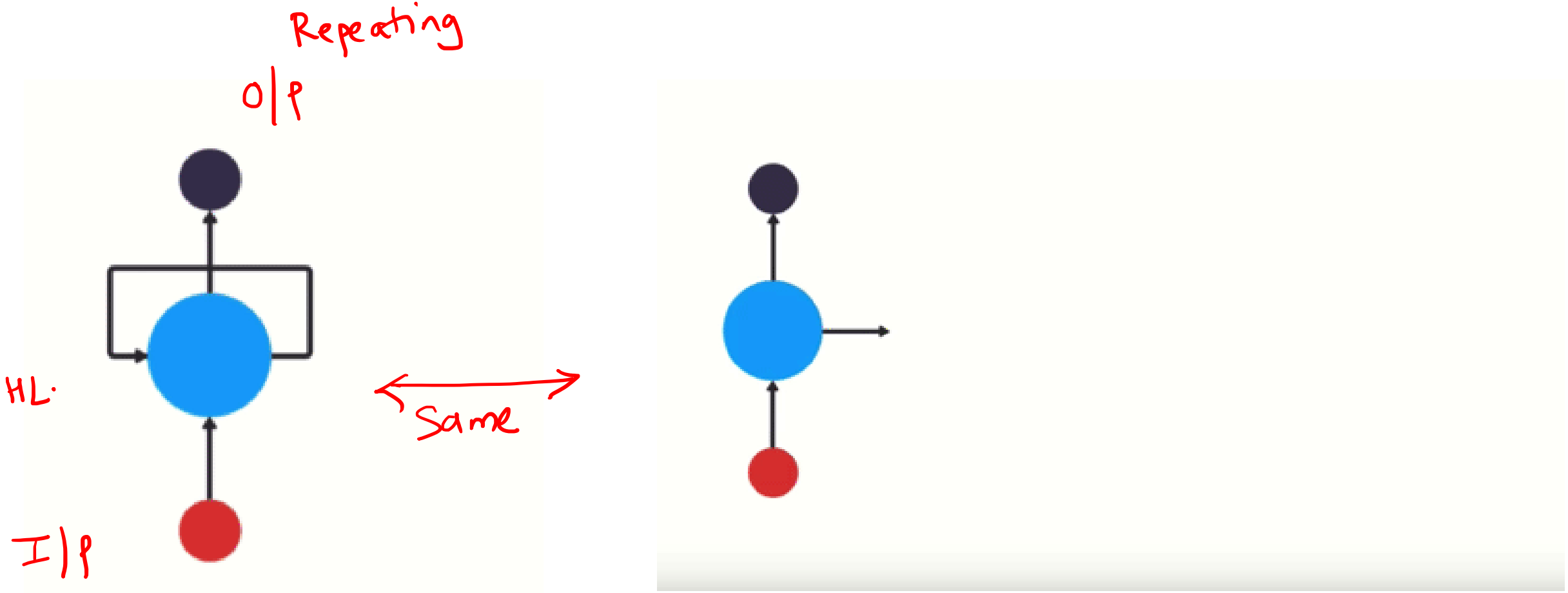
Refer NB “7 CNN_CIFAR_Improved.ipynb”

CNN Completed!!!

DEEP LEARNING – Recurrent Neural Network

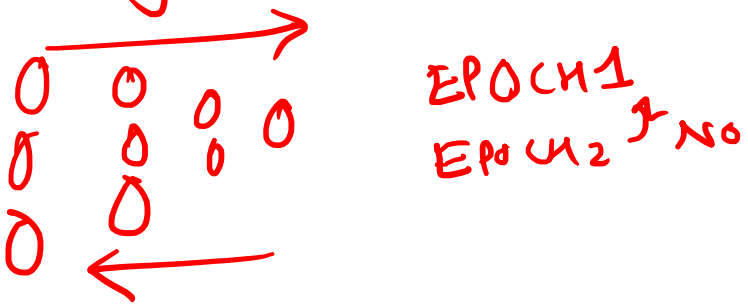
Trainer: Dr. Darshan Ingle.

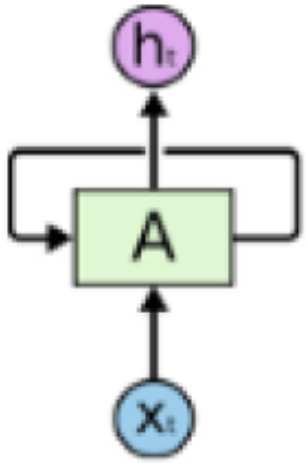
Recurrent Neural Network



Recurrent Neural Network (RNN)

Classify what kind of event is happening at every point in the Movie.

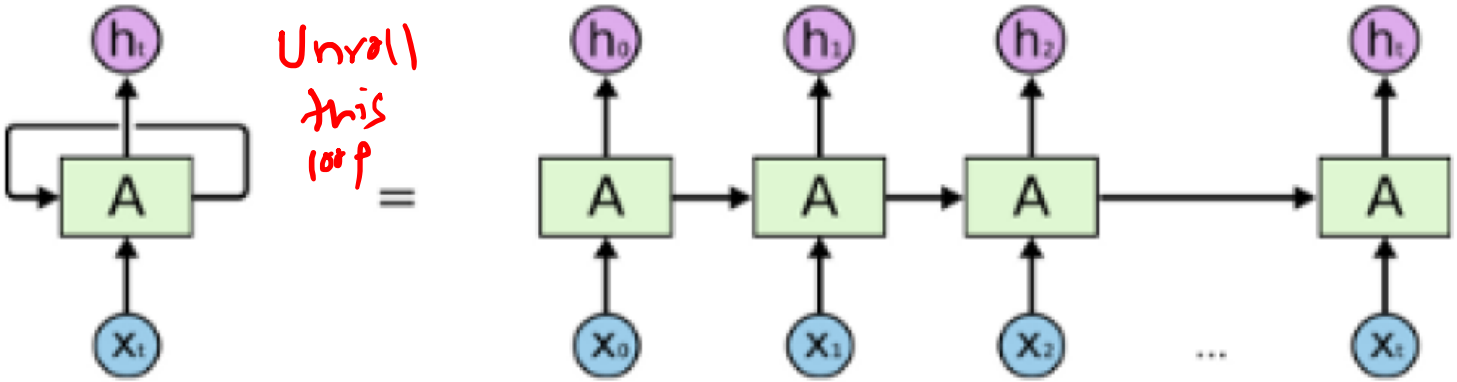




RNN have loops.

I/p

RNN



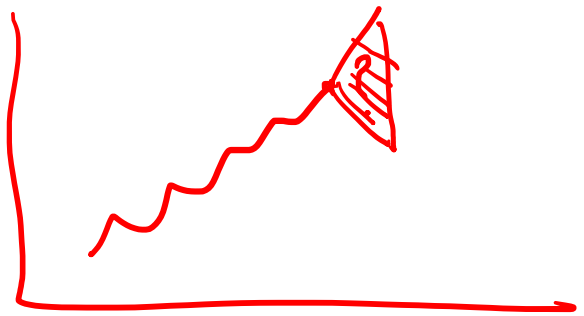
An unrolled recurrent neural network.

Why RNN?

Text Processing

Sequence: Stream of data which are independent.

eg: Time Series, Pieces of strings, Conversation, etc.



CNN dont perform well when the i/p data is interdependent in a sequential pattern.

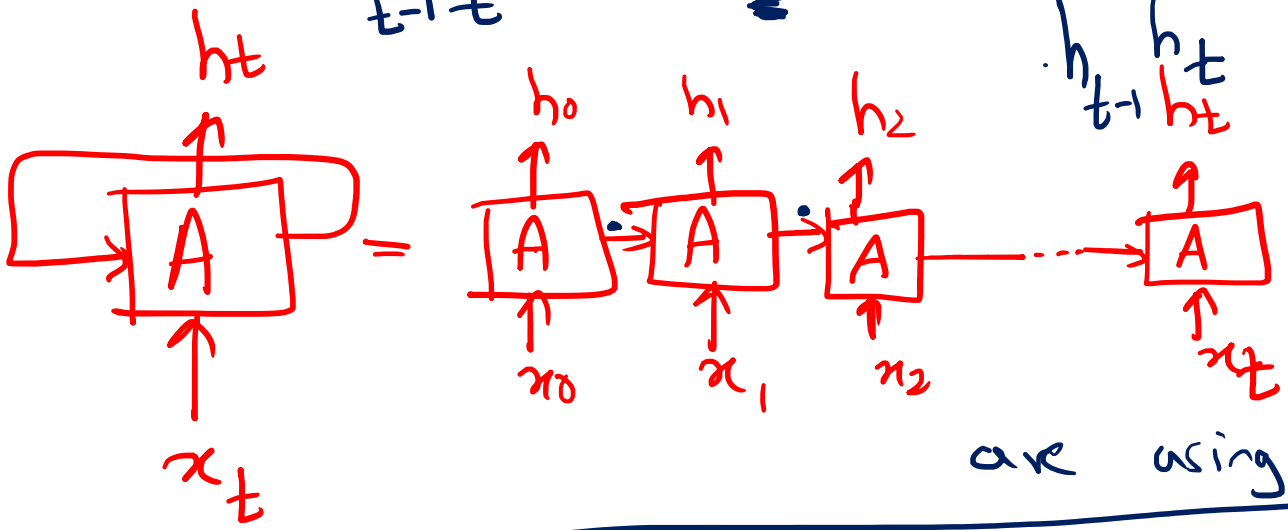
Example

Predict what is the next letter ?

Word : namaste

namaste

t-1 t



$\Rightarrow$ Formula : $h_t = f(h_{t-1}, x_t)$

where h_t = new state

h_{t-1} = previous state

x_t = i/p at time t.

Let us assume that we are using AF tanh

$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t)$$

Final o/p: $y_t = W_{hy} \cdot h_t$