Commands:

========

python3 -m spacy download en\_core\_web\_md

Jupyter notebook

<https://www.udemy.com/course/chatbot/learn/lecture/8807818#overview> ( **UDEMY  PAID Course )**

<https://medium.com/swlh/end-to-end-chatbot-using-sequence-to-sequence-architecture-e24d137f9c78>

**Recommendation project**

**=======================**

[**https://github.com/saikrithik/JanataHack-Recommendation-Systems/blob/master/FinalLSTM(0.271).ipynb**](https://github.com/saikrithik/JanataHack-Recommendation-Systems/blob/master/FinalLSTM(0.271).ipynb)

[**https://github.com/saikrithik/JanataHack-Recommendation-Systems**](https://github.com/saikrithik/JanataHack-Recommendation-Systems)

**CHATBOT:**

1.ENTITIES & NER

2.INTENT of the text

3.Normalization:   The Chatbot program model processes the text in an effort to find common spelling mistakes or typographical errors

                   that might the user intent to convey. This gives more human like effect of the Chatbot to the users.

4.Dependency Parsing: The Chatbot looks for the objects and subjects- verbs, nouns and common phrases in the user’s text

                    to find dependent and related phrases that users might be trying to convey.

5.CONTEXT-Next query if user asks it should remember the chain of questions asked

6. Learn to combine Moneky Learn,Spacy,Chatbot (Rasa)

7.Doc2vec, LDA2VEC, Seq2Seq  model

<https://chatbotslife.com/text-classification-using-algorithms-e4d50dcba45>

<https://github.com/Priya997/SKF-Chatbot/blob/master/Basic_Approach/Scripts_to_prepare_dataset/data.json>

search for   "NLP services and applications programming interfaces"

**BagofWords models**

**=================**

It constructs  a bag of vocab (say 20k english words)

then for  each document who calculate the COUNT of words present into vectorized input format

Drawback:

          Fixed size input

                                  Here it doesnt catch the context or semantic of the entire document

                                        ( As words jumbled also results in same Vectorized input format ... coz it doesnt consider wat word follows this word or prev word...

                                                                only the frequency of its occurence is captured)

                                  Fixed size output

                                  (In real world for conversations (like for CHATBOT building) there wont fixed length of sentences we speak)

Image classification (CNN ) : It takes a fixed size image and predicts whether it is  DOG/CAT

Context between series of images ( RNN ) :If you need to find the  context of sequence of images (like a small video frames to image of dog you provide) and you can  classify it has WALKING/RUNNING (Here is where  it learns from a sequence of images)

Usually  Sequence is mostly implemented for NLP like problems ( Where to  understand the CONTEXT or INTENT of the sentence)

**Seq2Seq**

=======

It helps to mimic real life conversation and best  suited for chatbot (Input  & Output lengths can vary and its accomdated with help of SOS/EOS tags)

since the order of the words is crucial for understanding the sentence. i.e context

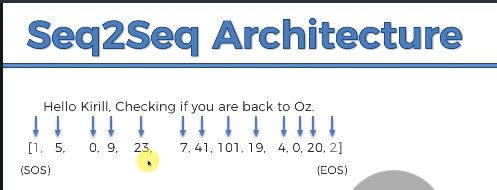
**Here instead of the FREQUENCY/COUNT\_OF\_THE\_WORDS:**

we shall consider the POSITION\_OF\_WORD in the taken sentence

    - sentence can be of varying length

    - position of the word recorded is w.r.t that sentence only

                - Same word present in diff sentence have no relation to the same word  in current sentence (Only there respective position are recorded within that sentence)



Or

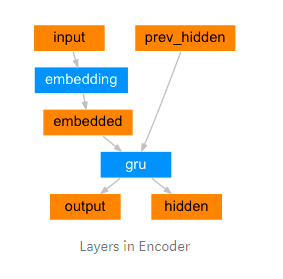
ANN is used for Classification & Regression

CNN is supervised more apt for Images (But can be used for Text too)

**RNN is called Seq2Seq model**….

And its applications is for (Time\_Series/Video Analysis /Seq2Seq chatbot conversations/Language Translation)

* **RNN** (Encoder,Decoder with  no gates) just one tanh activation.
* **LSTM** (It’s a advanced form of RNN**)…. It has 3 gates** (Input gate, Output gate and Memory/Forget gates)
* **Bi Directional-LSTM’s –** It uses both the left/Right sequence knowledge and that sum of knowledge is used as final context vector
* **GRU**(It’s a variation of LSTM**)….** Gated Recurrent unit… **Only 2 gates** (Update/Reset gates) it doesn’t use Memory Gate …. GRUs or Gated Recurrent Unit are simpler than the LSTM model. GRU came after LSTM and the former is simpler, faster, computationally less expensive and scalable as compared to latter
* **BI-DIRECTIONAL GRU’s**
* **Transformers (** ENCODER,ATTENTION,DECODER) …. **It Uses self Attention Mechanism ….**Along with order of the words **it  includes the  POSITION of the words too for additional learning**
* **MULTI-HEAD Transformers (It uses Masking concept)**
* **BI-Transformer called BERT**



**RNN** (Stateful algo) It has state vector (for short term memory)

===

In ANN it goes has Input==>Hidden==>Output (Hidden layer only gives the data to OUTPUT LAYER)

but in **RNN the hidden layer gives data to (OUTPUT LAYER + Some STATE\_VECTOR to its successor Hidden layer)** -[ Thats why RNN is called  SHORT TERM memory ]

One to Many (One picture) to (providing the caption for the  picture)

Many to one (Sentiment analysis) so many text to predict the POSITIVE/NEGATIVE

Many to Many (Language translation)

**Problems of RNN:**

RNN fails for longer sentences ….coz the size of  input sentence is directly proportional to Vanishing/Exploding Gradient problem.

                . Vanishing Graident Problems

                                                                 (Over sequence of neuron layers ...due to DOT VECTOR MULTIPLICATION the weights tend to ZERO out and NO LEARNING happens

                                                                  or weights tend  to EXPLODE and NO LEARNING happens)

                                                                 . Vanishing Problem ( Intial Weight Intialization takes care or LSTM takes care)

                                                                . Exploding Problem ( Truncated BackProgation, Penalties, Gradient clipping)

**LSTM:**  (Only ENCODER and DECODER )

====

Advantage: Vanishing Gradient is taken care ….. by using selective Forget & Memory(To store longterm info) units.

           It learns  from the data automatically over several iterations that what to remember or wat to forget

                                   Sent1: Boy wants to learn (Subject is  Boy)

                                   Sent2: Girl wants to dance (Subject is Girl)

                                   when while reading sentences if it  encounters comma (it might still keep the MEMORY GATE activated and continue collecting context)

                                   When the subject changes (boy to girl) then it activate FORGET GATE (So it flushes  the  info and freshly collect context)

**Problems of LSTM:**

* Although an LSTM is supposed to capture the long-range dependency better than the RNN, it tends to become forgetful in specific cases.
* Another problem is that there is no way to give more importance to some of the input words compared to others while translating the sentence.

*Gated Recurrent Unit(GRU)….* ***(It is LSTM variation):*** Here instead of FORGET/MEMORY mechanism it has  **UPDATE/RESET Gates**  its called **GRU**

GRU’s are the altered version of LSTM and are computationally less expensive when compared to them. GRU comprises two gates, update gate and reset gate

***update gate*** is to decide whether to pass Previous O/P to next Cell or not.

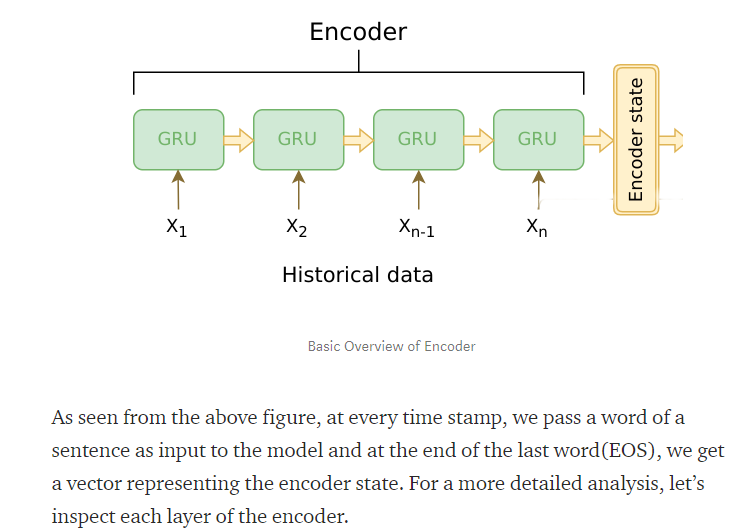
We can use ***update gate*** to decide what information to throw away and what new information to add.

The ***reset gate*** is another gate that decides how much past information to forget

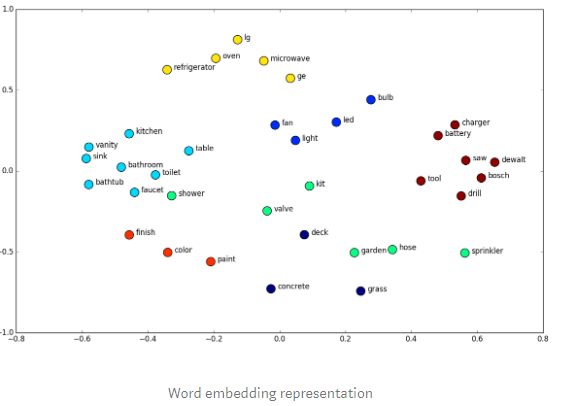
**Bi-directional GRU-**

The idea behind Bidirectional RNN is to feed in the input sequence in normal order for the first GRU and in reverse order for the second GRU. The output from both the GRUs can either be concatenated or summed up… Besides this, we also take the last hidden state  that will be passed to the next layers.

**Encoder-** We can see an encoder as a simple RNN that takes the words in question as input and returns a single vector representing all the words of the question without losing the sequential information. Lets deep dive into how is this going to work and the layers that are associated with it.



**Embedding-** Why do we need to use an embedding layer in Neural Network? Why can’t we directly send the tokens inside the model? The obvious reason is that we are implicitly giving a natural order by tokenizing the words which are not a correct way. So the next idea would be to perform a one-hot encoding of words. We have a very large vocabulary of words and converting these into dimensions using **one hot encoding would lead us to a massive sparse dataset** which would obviously not work well with our neural network. So overcoming this problem, we embed the words into an N-dimensional point such that what is semantically closer to each other are grouped together as shown in the figure.



**Decoder:** It is similar to the encoder. It takes as input the hidden vector generated by encoder(**Usually Attention info**), its own hidden states and current word to **produce the next hidden vector and finally predict the next word.**

**Working of Seq2Seq :**

**(SOS usually not needed coz its anyways the start of sentence)**

**Encoding:**

**While training we give the input “ Hello Krill Checking” follow by EOS.**

**Once the EOS is encountered, it understands that  now Encoding should be  stopped**

**And Decoding sud start i.e it should  start Predicting.**

**Decoding:**

**So it takes all the Encoder Knowledge and predicts the  next word “YES” as output èThen it takes “The hidden vector knowledge gained from Yes and predicts the “I’m”**

**And so in till “EOS” is predicted.**

***Inner Detail of Decoder:* The “YES”  is predicted by  using the probability of**

**(for example among the 20K English  words or  knowledge learnt from training of  corpus )**

**And then next word  “I’m” Is predicted based on same logic**

**And so on it will understand that no good probability threshold score is obtained**

**Then predicted it as “EOD” or *< if you had defined the max limit of the words to be predicted as output>***

**Approaches of Attention:**

* **Attention:** The input to the decoder is a single vector which has to store all the information about the context (I.e H4 in below example)

. This becomes a problem with large sequences

. Hence the attention mechanism is applied which allows the decoder to look at the input sequence selectively.

Inother words,

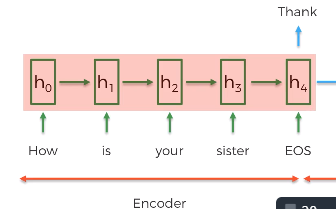
Attention mechanism while predicting every single OUTPUT\_WORD during decoding

It not only uses the usual STATE\_VECTOR from previous predicted word.

But also it considers the *WEIGHTED\_CONTEXT\_VECTOR\_of\_ENTIRE\_INPUT\_SENTENCE* **every single time it predicts the subsequent successive predicted words**

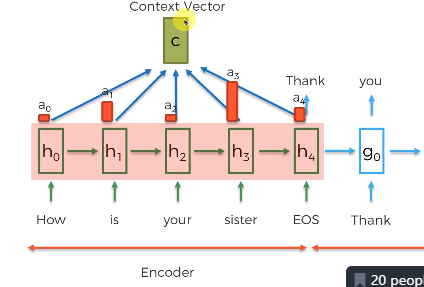
         i.e *WEIGHTED\_CONTEXT\_VECTOR\_of\_ENTIRE\_INPUT\_SENTENCE* <==> you can call it has *“Paying\_Attention” only to the highest weighted input word.*

**To predict “Thank” *only the context vector of EOS* *i.e ‘h4’* is enuf** **(**coz ‘h4’ containsfull knowledge of input sentence**)**

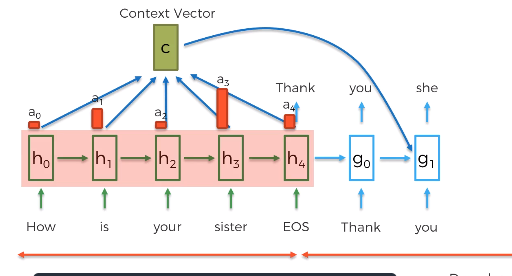
****

**To predict “ You”  it takes the  (STATE\_VECTOR of ‘Thank’ + the entire input sentence ka CONTEXT VECTOR) …notice it the weightage of the input words is also learned from training**

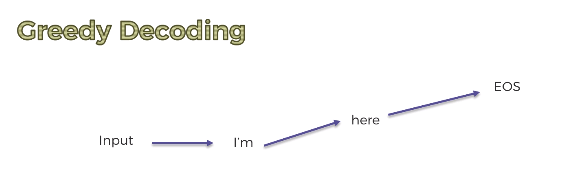
**You è *< State\_Vector>* of ‘Thank’   +  *<Context\_Vector> of entire input sentence (with ‘a3’ word given more weightage) i.e Sister***

****

**Similarly the  ‘SHE’ word is also calculated with all the WEIGHTED INPUT SENTENCE + ‘YOU’ ka *State\_Vector***

****

* **Greedy Decoding:** The highest probability word is selected as the output by the decoder.



* **Beam Decoding:** Greedy decoding is not always yield the best results, because of the basic problem of greedy algorithms. Hence beam search is applied which suggests possible Top\_Best\_3\_words at each step. This is done making a tree of top k-results.

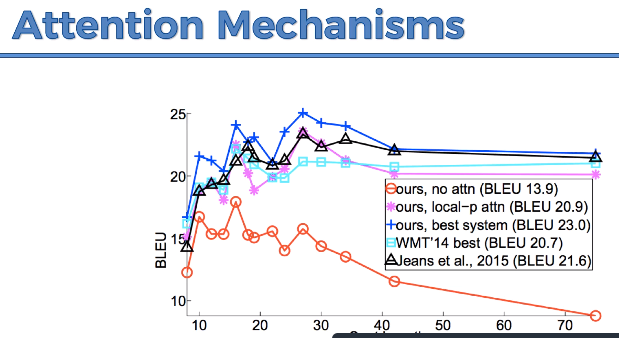


* **Bucketing:** Variable-length sequences are possible in a seq2seq model because of the padding of 0’s which is done to both input and output. However, if the max length set by us is 100 and the sentence is just 3 words long it causes huge wastage of space. So we use the concept of bucketing. We make buckets of different sizes like (4, 8) (8, 15) and so on, where 4 is the max input length defined by us and 8 is the max output length defined.

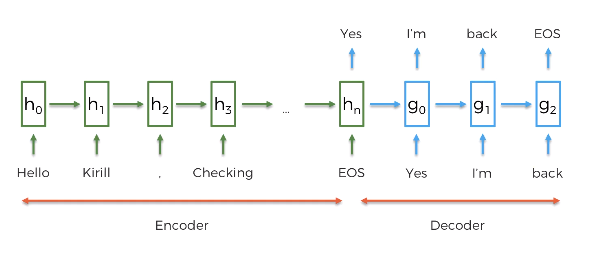
**BLEU metric:**

**Metrics for analysing the PAYING ATTENTION  vs  LENTH OF SENTENCE**

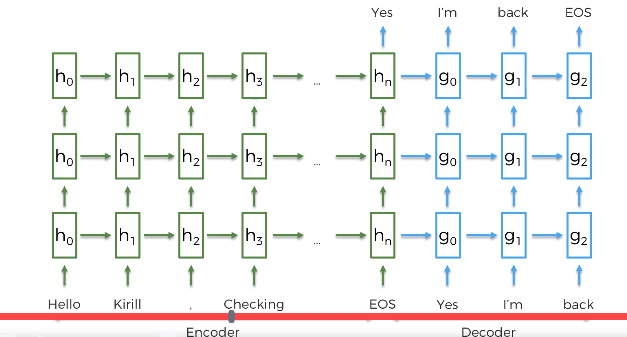
**(Here the red line is not using attention mechanism…. Hence its accuracy is decreasing)**



**Simple Layer**

****

**Complex Layers:**

****

**Problems of RNN:**

                . Vanishing Graident Problems

                                                                 (Over sequence of neuron layers ...due to DOT VECTOR MULTIPLICATION the weights tend to ZERO out and NO LEARNING happens

                                                                  or weights tend  to EXPLODE and NO LEARNING happens)

                                                                 . Vanishing Problem ( Intial Weight Intialization takes care or LSTM takes care)

                                                                . Exploding Problem ( Truncated BackProgation, Penalties, Gradient clipping)

**LSTM:** (Vanishing Gradient is taken care)

====

It has the GATES concept ....Forget/Memory(remember) Gates

           It learns  from the data automatically over several iterations that what to remember or wat to forget

                                   Sent1: Boy wants to learn (Subject is  Boy)

                                   Sent2: Girl wants to dance (Subject is Girl)

                                   when while reading sentences if it  encounters comma (it might still keep the MEMORY GATE activated and continue collecting context)

                                   When the subject changes (boy to girl) then it activate FORGET GATE (So it flushes  the  info and freshly collect context)

**LSTM Variations:** Here the variations are done by considering the combinations for MEMORY +FORGET gate rather than either MEMORY or FORGET

                               its called **GRU**

**CHATBOT HANDSON:**

**Decoder**

For decoder we need not define EOS…. Coz (EOS is not needed… its taken care by the **threshold score** of predicting or a upper a limit)

You give the  data in batches

Each data  you give will have a SOS token and last column(EOS is removed)

And then the batch of the data rows are concatenated.

**LSA**

**===**

In natural language understanding (NLU) tasks, there is a hierarchy of lenses through which we can extract meaning —

from words to sentences to paragraphs to documents.

At the DOCUMENT LEVEL, one of the most useful ways to understand text is by analyzing its TOPICS. (usually it is Co-Occuring Keywords)

The process of learning, recognizing, and extracting these topics across a collection of documents is called topic modeling.

LSA uses bag of word(BoW) model, which results in a term-document matrix

LSA learns latent topics by performing a matrix decomposition on the document-term matrix using Singular value decomposition(SVD)

LSA (Latent semantic analysis) is for finding the topics in the corpus, or inshort for summarizing of the text.

SVD Decomposition

=================

M=USV

i.e M(m\*m)          ==> U(m\*n) \* S(n\*n) \* v(n\*m)

i.e Term\_doc\_matrix ==> Term\_vs\_Topics   \*   Topic\_Vs\_Topic(co-occurence of topics)   \*  Topic\_vs\_document matrix

LSA is typically used as a dimension reduction or noise reducing technique.

The core idea is to take a matrix of what we have — documents and terms — and decompose it into a separate document-topic matrix and a topic-term matrix.

load the corpus into dataframe

==>make list of documents present

 ==> preprocess the documents(Tokenize the doc,stemming,lemma)

==> create dictionary of all\_words (every word is given index value)

==> Using the dictionary created ...create the Document-Term matrix

 ==> fit into the LSI model (By telling how many TOPICS you want)

==> There is a cohersion method to find out optimum  no of TOPICS for your corpus

==> with the obtained num\_of\_topics train the LSI model

==> It predicts each Topics (with the words associated)

==>With these document vectors and term vectors, we can now easily apply measures such as cosine similarity to evaluate:

the similarity of different documents

the similarity of different words

LDA – Latent Dirichlet Allocation –  Its foundations are Probabilistic Models.

from the LSA wat we get is the list of topics and how similar are the documents to each other

LDA provides the probability factor, by telling each document probabilty score for the topics out there.

==> LSI method sometimes predict multiple overlapping topics

     (To better it ...we can use LSA- Latent dirichtlet allocation..which gives us which overlapping topic has more probabi

Additional Info

===============

Options for topic modelling (Tokenize the  corpus)- LSA/LDI/pip install lexrank (<https://pypi.org/project/lexrank/>)

Text Summarization (You can consider sentences)-pip install lexrank

                                   You  can form sentences using SENTENCE\_TOKENIZATION with NLTK or SPACY

Aspect based analysis :

                        Printer is robust and  have long shell life (Positive) but Ink cartidge is pricey and frequent fills needed (Negative)

                                                                                                The process for ascept based  approach  would be convert the paragraphs into smaller units  (Called Opinion units)

                                                                                                You can create your own CUSTOM\_EXTRACTOR (Monkey learn is a tool) - <http://help.monkeylearn.com/en/articles/2174073-how-to-build-a-custom-extractor>

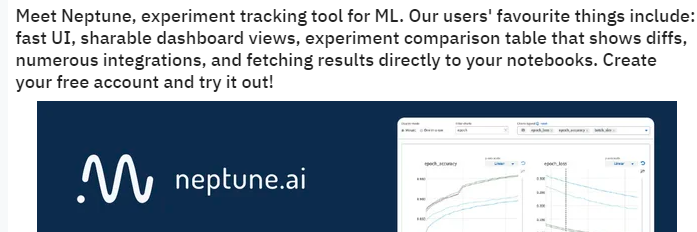
                                                                                                Google for alternatives for 'Monkey Learn' who will get list of tools or Github: <https://github.com/monkeylearn>

Learn about "Text mining" to prepare your data.

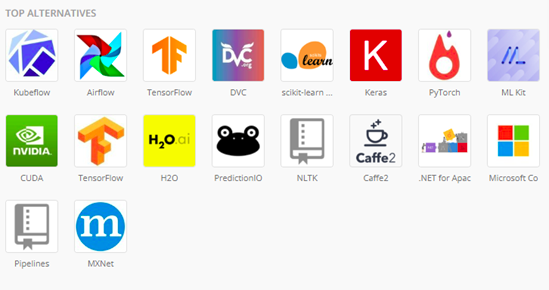
                             google about "Annotations for Textual Data"

mlflow ui

<https://colab.research.google.com/drive/1WiGoLBO5hi63zgpcFmWbWGVOdDNzxicx#scrollTo=TPcE-cf03x1G>



**MLFLOW is used for MachineLearning lifecycle and training n all or try  TENSORFLOW TFX for deployment**

****

**FEMCARE PROJECT (Yolo,SSD, Faster RCNN, fast.ai(research), resnet(research) ,**

**Regularization – L2 Regularization**

**EarlyStopping -For every Epoch -Its called as CheckPoinitng actually**

**(Save the separate** pkl **file and check TRAINING vs VALIDATION loss…and choose the EPOCH which gives the**

**LESS VALIDATION LOSS…… notion is usually TRAINING LOSS WOULD BE LESS but VALIDATION LOSS is more**

**Whole idea is to have LESS VALIDATION LOSS  )**

**Dropout  (Try one model with 0.3 see accuracy…do the same for 0.4 dropout…same for 0.5 ) è more the dropout …rem to do more EPOCH**

**BatchNormalization**

**Data Augumentation**

**Blur,**

**sharpen,**

**Histogram(HOG)  images (On full image or on parts of the image),**

**crop the data è(**RandomResizedCrop**way of cropping and rescaling in** pytorch**) PADHAIà CNN architecture à Image transform**

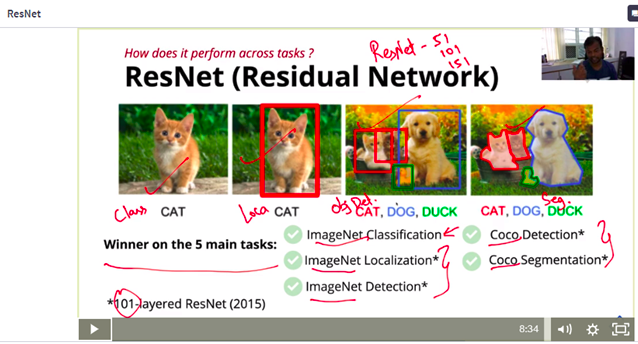
* **Transform.RandomResizedCrop() and then Normalize the data (advised always for image** kinda **data)** Transform.Normalize**()**

**rotation(all 8** anagles**) ,**

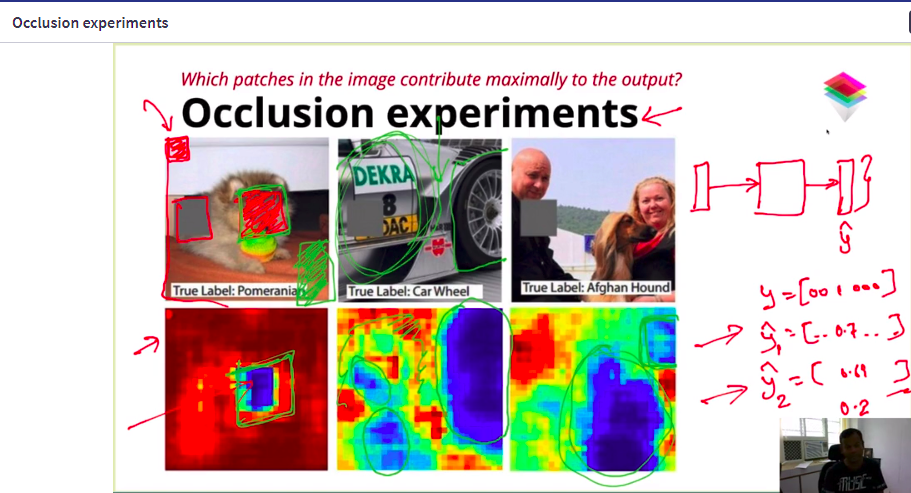
**zoom IN/OUT on the defect ,**

**online see for black & white pad images**

**Find out which algorithm or approach is good for Black&White images (for intuition which is good for RGB images)**

****

**You Can do occlusion experiment to figure out whether it learn something at the defected spot**

****

**When to reduce epochs**

* If your train error is very low, yet your test/validation is very high, then you have over-fit the model with too many epochs.
* The best way to find the right balance is to use early-stopping with a validation test set. Here you can specify when to stop training, and save the weights for the network that gives you the best validation loss. (I highly recommend using this always)

**Implications of a larger batch-size**

* Too large of a batch\_size can produce memory problems, especially if you are using a GPU. Once you exceed the limit, dial it back until it works. This will help you find the max batch-size that your system can work with.
* Too large of a batch size can get you stuck in a local minima, so if your training get stuck, I would reduce it some. Imagine here you are over-correcting the jumping-around and it's not jumping around enough to further minimize the loss function.
* BIGGER the BATCH SIZE (execution faster but it wont be generalized i.e overfits)...hence its advised the batch-size needs to be medium.... so that in calculating the avg gradients across the NO\_OF\_STEPS\_INSIDE\_ONE\_EPOCH makes it more robust & generalized rather than training entire DATASET at once in memory

**Implications of a smaller batch-size:**

**It** does not guarntee it will hit the global minima (it might get stuck in some local minima)

**When to adjust steps-per-epoch**

* Traditionally, the steps per epoch is calculated as train\_length // batch\_size, since this will use all of the data points, one batch size worth at a time.
* If you are augmenting the data, then you can stretch this a tad (sometimes I multiply that function above by 2 or 3 etc. But, if it's already training for too long, then I would just stick with the traditional approach.

**Imp takeaways:**

We can increase the  Learning rate in case of LARGE BATCH SIZE (it will be better)

***Varying batchsize every epoch***

train using a small batch size for a single epoch then switch to a large batch size, train using a small batch size for many epochs then switch to a larger batch size, and train using a large batch size then switch to a higher learning rate with the same batch size.

In conclusion, starting with a large batch size doesn’t “get the model stuck” in some neighbourhood of bad local optimums. The model can switch to a lower batch size or higher learning rate anytime to achieve better test accuracy.

<https://towardsdatascience.com/an-overview-of-time-series-forecasting-models-a2fa7a358fcb>

<http://localhost:5000/#/>

KERAS

#####

<https://www.guru99.com/keras-tutorial.html>

|  |
| --- |
| [Keras Tutorial for Beginners with Python: Deep Learning EXAMPLE](https://www.guru99.com/keras-tutorial.html)  What is Keras? Keras is an Open Source Neural Network library written in Python that runs on top of Theano or Tensorflow. It is designed to be modular, fast and easy to use. It was developed by Franço  [www.guru99.com](http://www.guru99.com) |

<https://www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in-keras/>

|  |
| --- |
| [Quick Notes on How to choose Optimizer In Keras | DLology](https://www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in-keras/)  What is Nesterov momentum?. Nesterov accelerated gradient (NAG) Intuition how it works to accelerate gradient descent. We’d like to have a smarter ball, a ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again.. Here is an animated gradient descent with multiple optimizers.  [www.dlology.com](http://www.dlology.com) |

<https://www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in-keras/>

Use SGD+Nesterov for shallow networks,

sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

and either Adam or RMSprop for FULL CONNECTED NETWORKS

**GRADIENT DESCENT:** Entire dataset  it traverses and the  calculate the LOSS FUNCTION(Ypred-Yactual) and then  it adjusts the WEIGHTS/BIASES

                  It will encounter many Local Minima (Before  finding the  Global  Minimum )

**STOCHASTIC GRADIENT DESCENT:** It takes just one input record on random basis.Calculate the LOSS FUNCTIONYpred-Yactual) ==> and then  it adjusts the WEIGHTS/BIASES

                            Its bit faster coz it adjusts weight/Biases for every record

                                                                                                                                               It finds Global minma faster but  fluctates a lot

**BATCH GRADIENT DESCENT:** It takes a small batch of records and does the LOSS FUNCTION

                        ( Little less fluctuate compared to stochastic)

**DIFF OPTIMIZERS:**

Adam **-** fully connected networks

RMS Prop fully connected networks

Stochiasted Gradient Descent - for shallow networks,

NestrovAccelerated Gradient - Big leap on correct direction & later slow down

ADAGrad - It makes big updates for infrequent parameters and small updates for frequent parameters. For this reason, it is **well-suited for dealing with sparse data**.

ADADelta **-**

It is an extension of AdaGrad which tends to remove the decaying learning Rate problem of it.

Another thing with AdaDelta is that we don’t even need to set a default learning rate.

**DIFF LOSS FUNCTIONS:**

[**https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/**](https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/)

|  |  |
| --- | --- |
|  | [How to Choose Loss Functions When Training Deep Learning Neural Networks](https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/)  Neural networks generally perform better when the real-valued input and output variables are to be scaled to a sensible range. For this problem, each of the input variables and the target variable have a Gaussian distribution; therefore, standardizing the data in this case is desirable.  machinelearningmastery.com |

* How to configure a model for mean squared error and variants for regression problems.
* How to configure a model for cross-entropy and hinge loss functions for binary classification.
* How to configure a model for cross-entropy and KL divergence loss functions for multi-class classification.

       1. **Regression Loss Functions:**

             Mean Squared Error Loss

             Mean Squared Logarithmic Error Loss - **when huge numbers involved.**

             Mean Absolute Error Loss - ***used if lot of outliers is there*** (t is calculated as the average of the absolute difference between the actual and predicted values)

1. **Binary Classification Loss Functions** 
   1. Binary Cross-Entropy (if output range is 0 to 1...like sigmoid)
   2. Hinge Loss (if output range is -1 to 1...with tanh function)
   3. Squared Hinge Loss
2. **Multi-Class Classification Loss Functions** 
   1. Multi-Class Cross-Entropy Loss ==>(for output classes like 0,1,2,3,...10)
   2. Sparse Multiclass Cross-Entropy Loss ==> (for output classes if done **one hot encoding** it will have lot of sparse data)
   3. Kullback Leibler Divergence Loss ==>used when using models that learn to approximate a more complex function than simply multi-class classification, such as in the case of an autoencoder used for learning a dense feature representation

**ACTIVATION FUNCTION:**

1. **Sigmoid - It's not zero centred...so it makes gradient go to extreme either 0 or 1 (suffers vanishing graident)...makes optimization difficult**
2. **Tanh - suffers vanishing gradient problem too ...but its zero centred (-1,0,1) so better for gradient calculation and optimization is lot easier ...hence used more often than sigmoid**
3. **ReLu - handles vanishing graidient (Sud be used only in HIDDEN LAYERS)...coz it tends to create some dead neurons sometimes ..so its best used only in HIDDEN layers**
4. **LeakyRelu -**

**Another problem with ReLu is that some gradients can be fragile during training and can die. It can cause a weight update which will makes it never activate on any data point again. Simply saying that ReLu could result in Dead Neurons.**

**To fix this problem another modification was introduced called Leaky ReLu to fix the problem of dying neurons. It introduces a small slope to keep the updates alive**

1. **SOFTMAX - only output layer (for classification problems)**
2. **Linear- only output layer​ (for regression problems)**
3. **Maxout: Its a hybrid of RELU & LeakyRelu**

BERT

=====

<https://medium.com/@ranko.mosic/googles-bert-nlp-5b2bb1236d78>