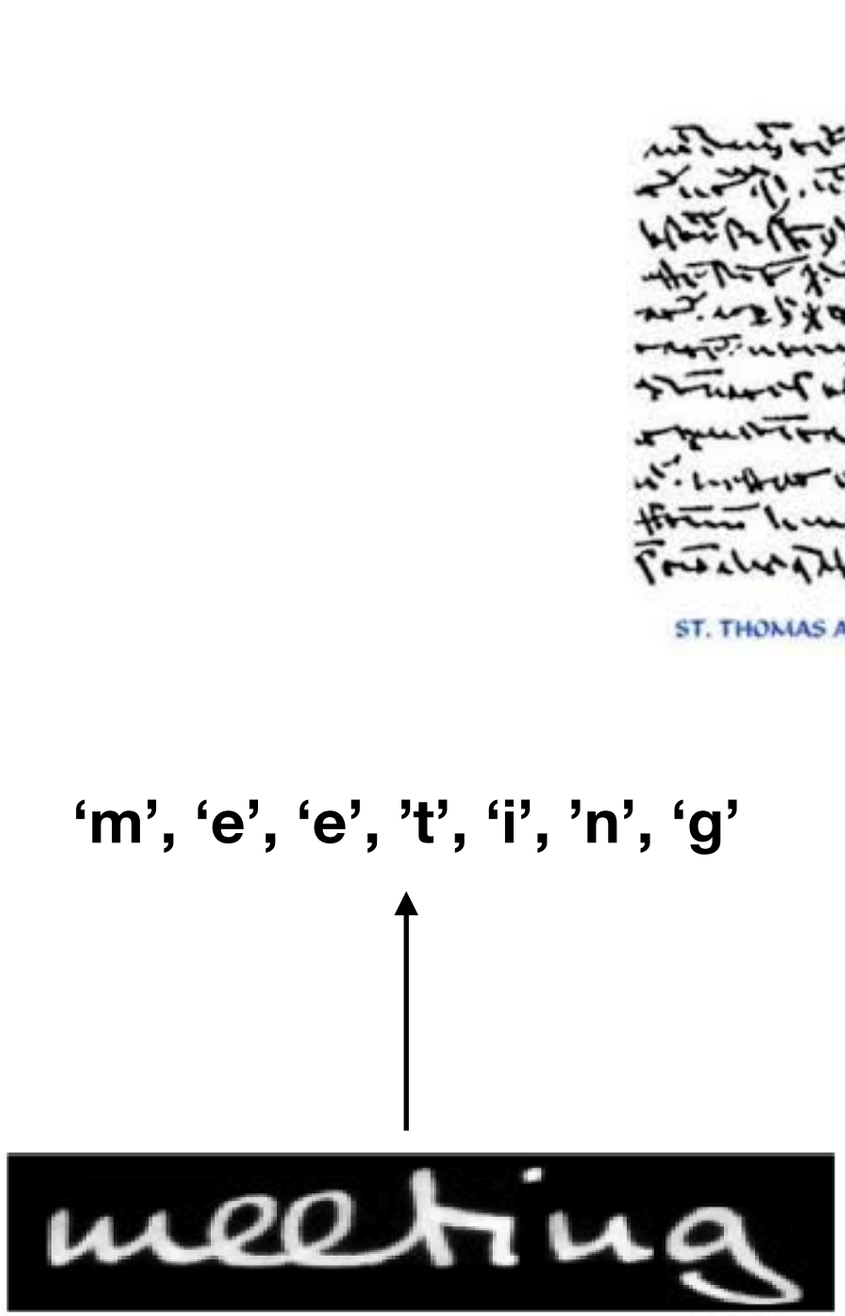
ST. THOMAS AQUINAS' HANDWRITING

‘m’, ‘e’, ‘e’, ’t’, ‘i’, ’n’, ‘g’



Handwriting recognition

Jorge Sueiras

Summary

- The handwriting recognition problem
- seq2seq solution
- Research lines
- Code examples

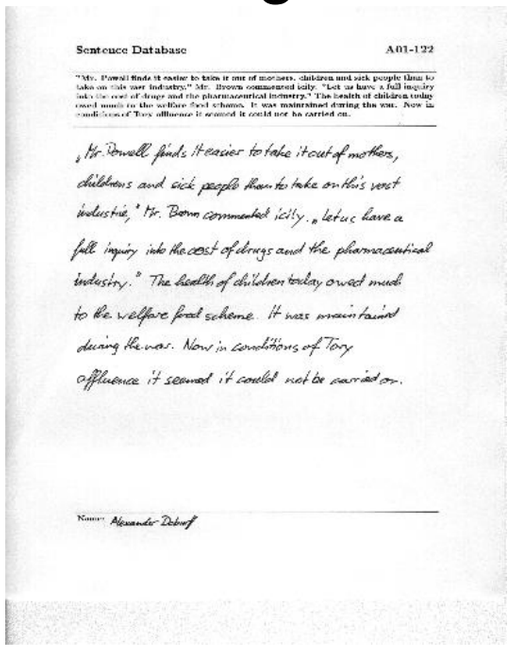
Summary

- **The handwriting recognition problem**
  - Problem definition
  - Data normalization
  - Modelling strategies
  - Language models
  - Intro to RNN
- seq2seq solution
- Research lines
- Code example

Problem definition

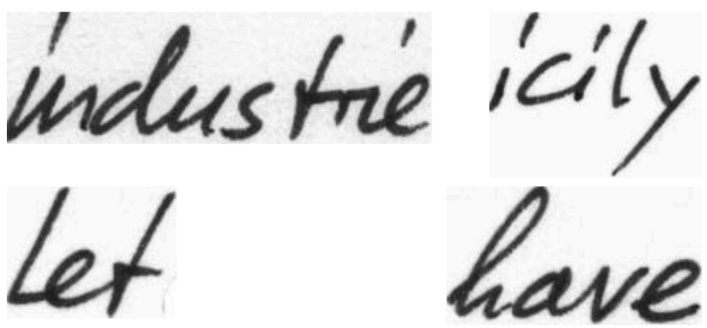
- Identify and recognize the handwriting content in digital images of documents.
- Usually the general problem is decomposed in tasks:
  - Identify handwriting parts in the document layout.
- Segment text into lines.
- Segment lines into words.
- Identify words (the main open problem).
- In the past also segment words into characters and next identify this characters.

Page



Line

Words



http://www.fki.inf.unibe.ch/databases/iam-handwriting-database

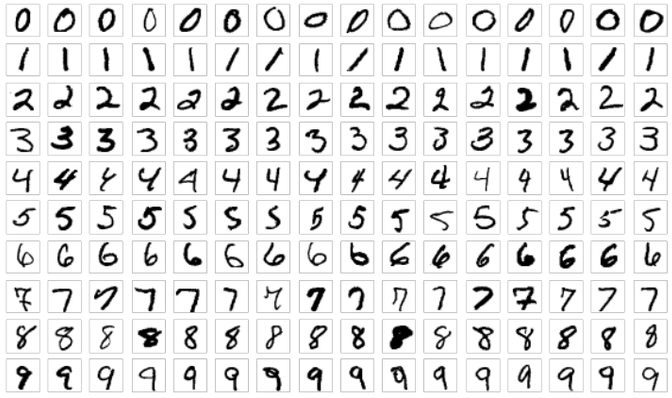
MNIST dataset and Lenet-5 architecture

- In 1998 LeCun, Y, Bengio, Y et all publish a paper with several applications of neural networks to solve some handwriting problems.

- The most relevant was the LeNet-5 architecture to recognize isolated characters. The network train simultaneously the features extractor and the classifier

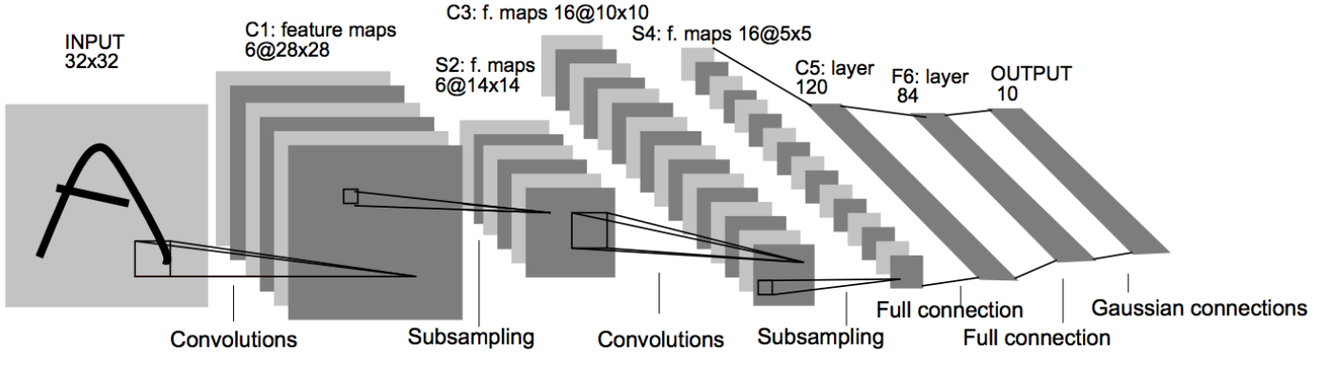
- “A new learning paradigm called Graph Transformer Networks” allows such multi-module systems to be trained globally using Gradient Based methods as to minimize overall performance measure.”

MINST dataset sample



Wikipedia

Lenet-5 architecture

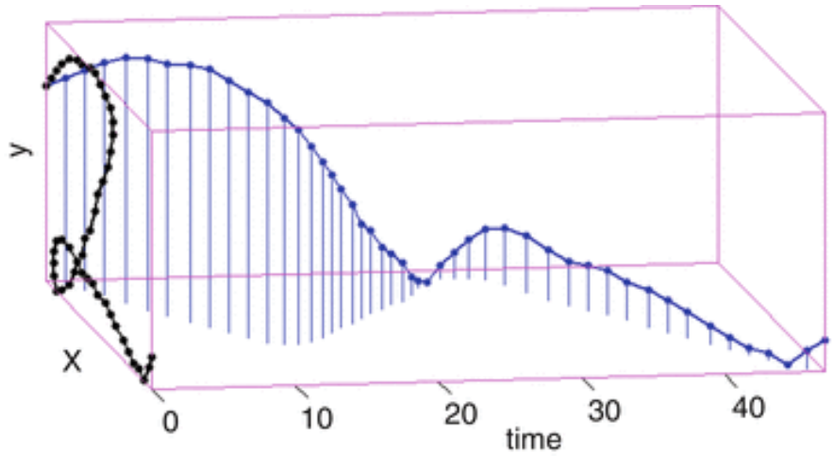


http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

Online handwriting recognition

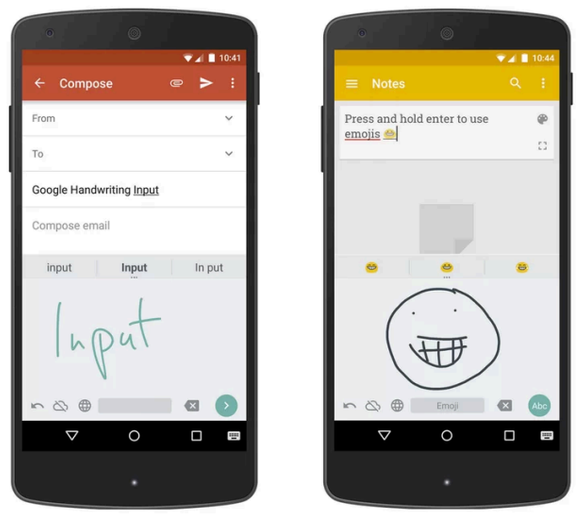
- In the online handwriting recognition we have the trajectory of the pen.
- The data structure is a sequence of:
  - X_position, Y_position, is_start_line (binary), is_end_line(binary)
- Online handwriting databases
  - IAM. http://www.fki.inf.unibe.ch/databases/iam-on-line-handwriting-database
- Online handwriting recognition is a problem solved.
  - From years exist several apps to do this and are a standard functionality in some mobile operating systems with high accuracy.

Online handwriting data



https://link.springer.com/referenceworkentry/10.1007%2F978-0-85729-859-1_29

Android online handwriting recognition

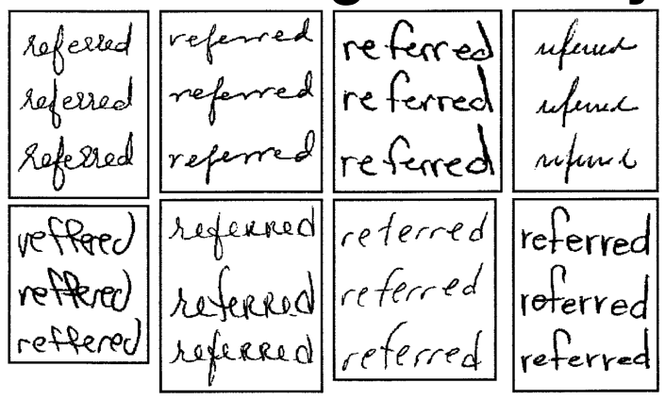


https://www.tomsguide.com/us/google-handwriting-recognition-android,news-20784.html

Data normalization

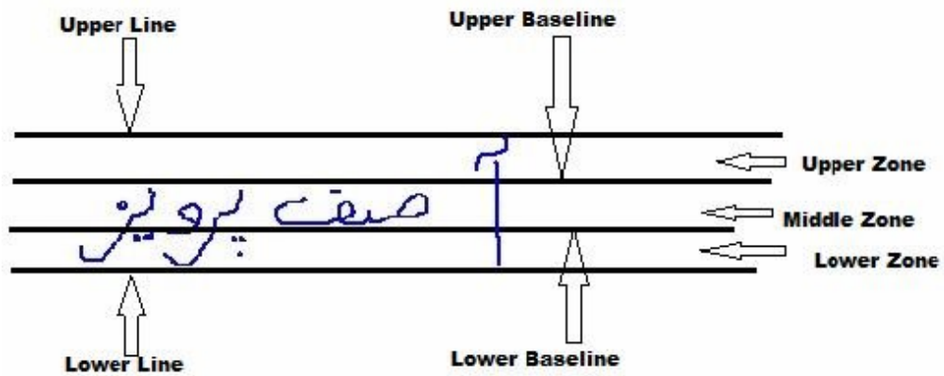
- Heterogeneity is the main problem on handwriting. There are as many different types of handwritten text as there are human.
- A good data normalization process is key to solve the problem.
- The usual normalization tasks over handwritten images are:
  - Clean noise in scanned images using thresholding techniques.
  - Detect baseline and upperline.
  - Correct skew
  - Correct slant
  - Adjust aspect ratios between baselines

Handwriting variability



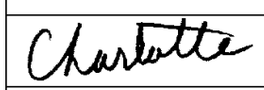
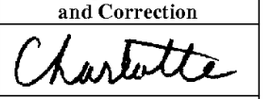
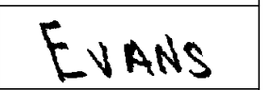
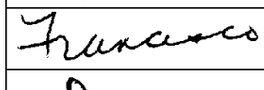
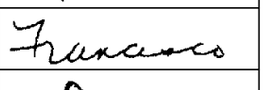
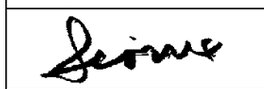
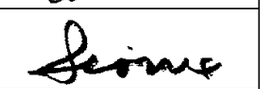
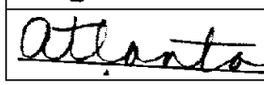
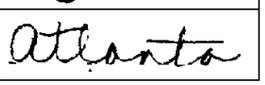
https://cedar.buffalo.edu/~srihari/papers/JFS-2002.pdf

Base and upper lines



An Empirical Evaluation of Off-line Arabic Handwriting And Printed Characters Recognition System. Parwej. F

Normalization skew example

Original Word	Following Skew Detection and Correction
	
	
	
	
	

Blumenstein et al. New Preprocessing Techniques for Handwritten Word Recognition

Modeling strategies

- Many approaches have been applied in the last years, but the main differences appear in 3 key elements of the solution:
  - The strategy to extract features from the handwriting image as inputs of the model.
  - The type of prediction model applied.
  - The way to decode the output of the classifier to predict the sequence of characters. That is a variable length sequence.
- To extract features the main approaches are:
  - Get features based on extract several computer vision techniques.
  - Use directly the pixels of images, usually by columns as model input.
- Types of models used are:
  - SVM, KNN, and others.
  - Neural networks, Full connect layers, CNNs, RNNs and variants/ combinations.
- To decode the output:
  - Hidden Markov Models
  - Connectionist Temporal Classification decoder

Examples of Computer Vision features

- Principal Component Analysis (PCA) to extract components over fixed-size frames.
- Histograms of Oriented Gradients (HOG)
- Characteristics extracted of each column pixel, such as mean and other moments, center of gravity, transitions and other aggregations.

Multidimensional LSTM - MDLSTM

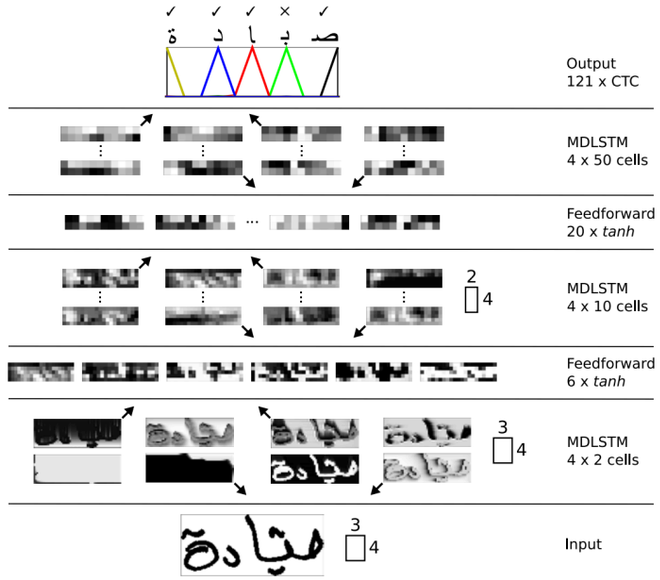
- The MDLSTM are networks that can manage recurrence in more that one dimension, but several dimensions.

- The main idea of the application of MDLSTM to handwriting are not only read the handwriting image from left to right, also read if in the right, top and bottom directions. The steps are:
  - Split the image in small square crops and convert it to a vector.
  - create 4 LSTM layer touring the cuts in 4 directions: from left-top, right-top, left-bottom, right-bottom.
  - Merge it whit dense layers.

- The first paper using MDLSTM to handwriting in Arabic are from Graves et all in 2009. In the image appears the output of intermediate layers in the MDLSTM architecture proposed.

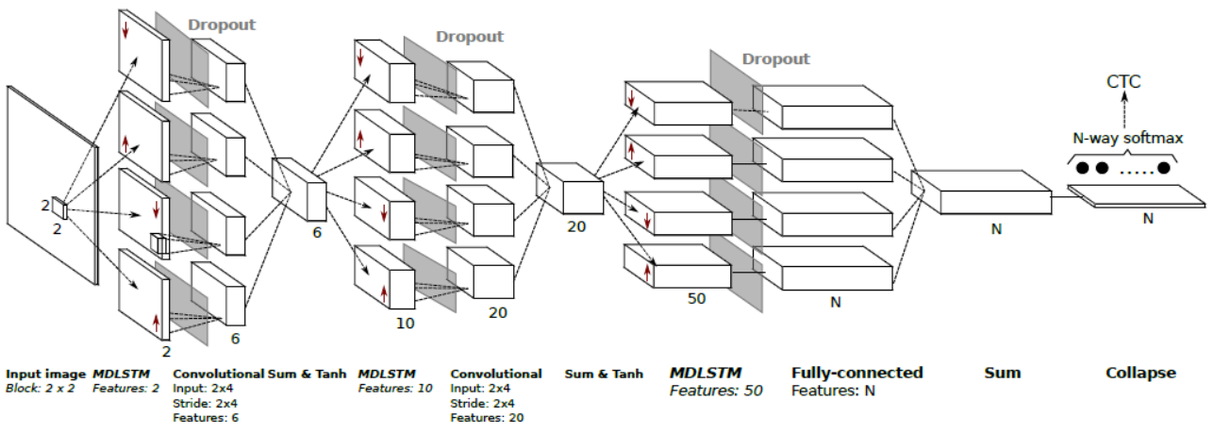
- Several authors replicate the architecture over other languages and with variations. Most of the best results in handwriting from 2009 to today are based in architectures with MDLSTM layers + CTC decoder.

MDLSTM to Arabic handwriting



http://papers.nips.cc/paper/3449-offline-handwriting-recognition-with-multidimensional-recurrent-neural-networks

MDLSTM architecture for handwriting

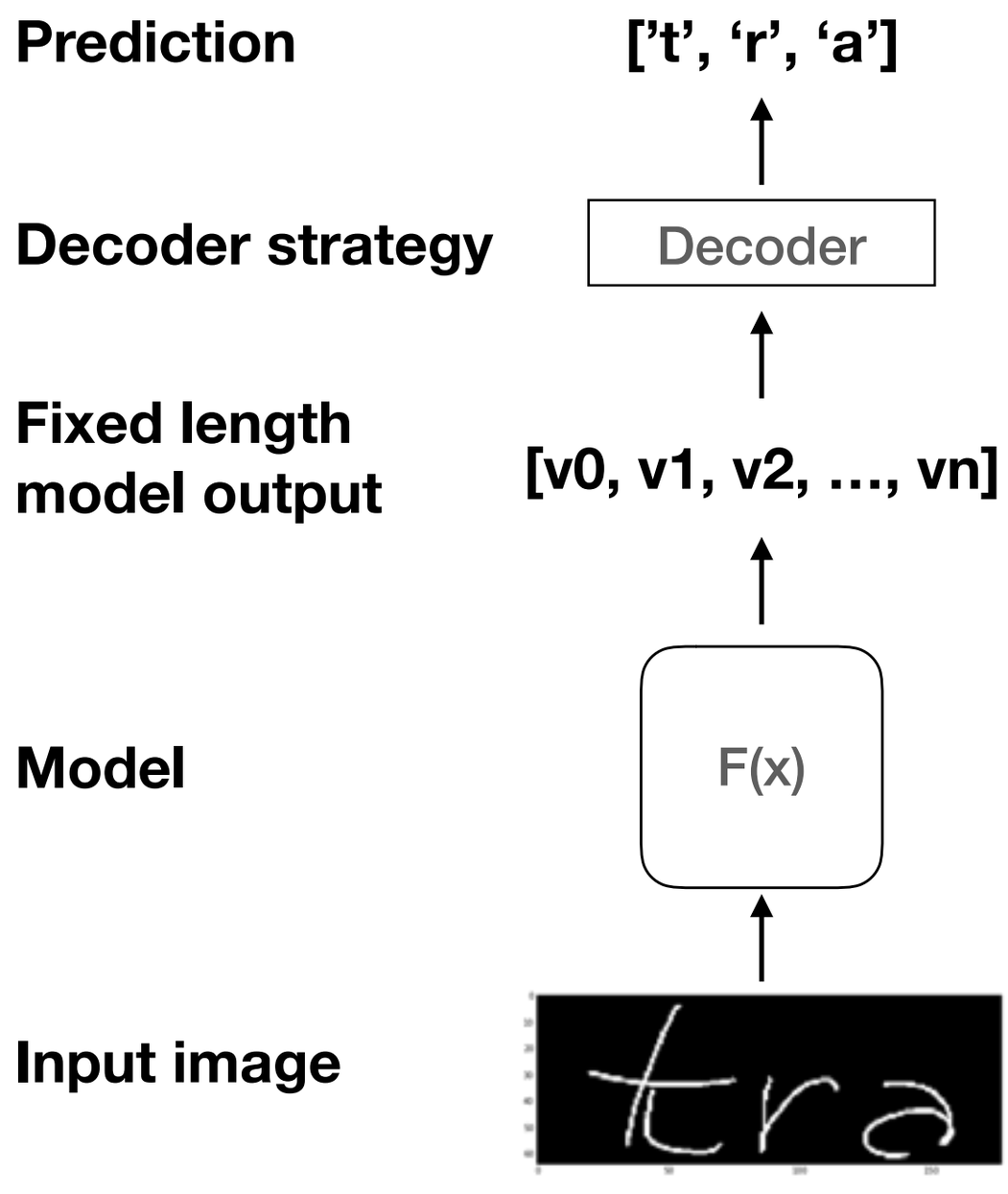


https://arxiv.org/abs/1312.4569

Modelling strategies output decoding

- One special characteristic of the online handwriting problem are that the output a re a sequence (of characters) of variable length. The usual classification models need targets of fixed size.

- options to solve this:
  - Hidden Markov Models
  - Connectionist Temporal Classification
  - Sequence to sequence decoder



Hidden Markov Models

- Process to decode the output of the model (a list of possible observations) into the sequence fo the characters of the handwriting text.

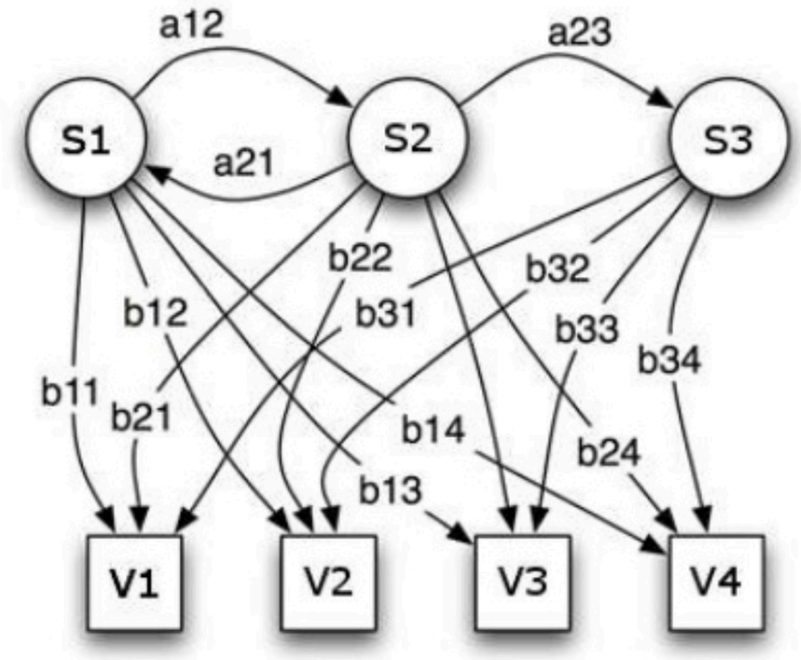
- Key elements:

  - States: the available characters.

  - States transition Matrix A: Contains the probability of the next character based into the previous characters. Trained using large text corpus.

  - Confusion (emission) matrix B: Contains the probability of the occurrence of each character into each possible observation

- The HMM generate a sequence of decoding characters based on the transitions between states driven by the A, B matrices.

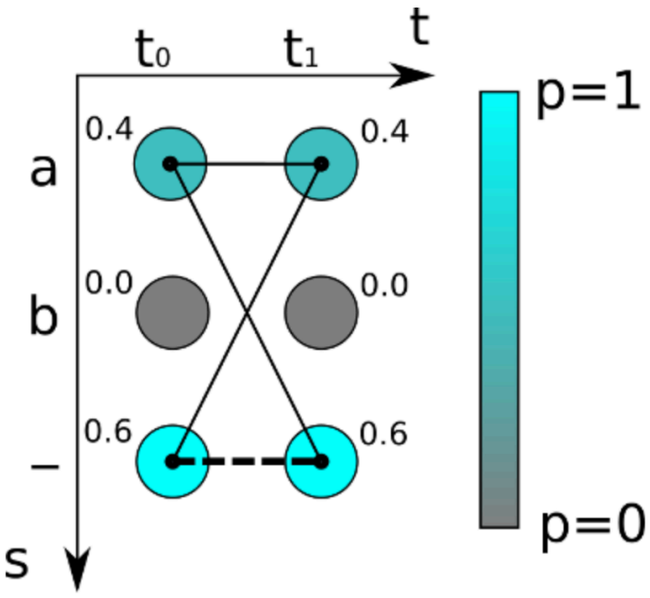


S **— states**
v **— possible observations**
a **— state transition probabilities**
b **— output probabilities**

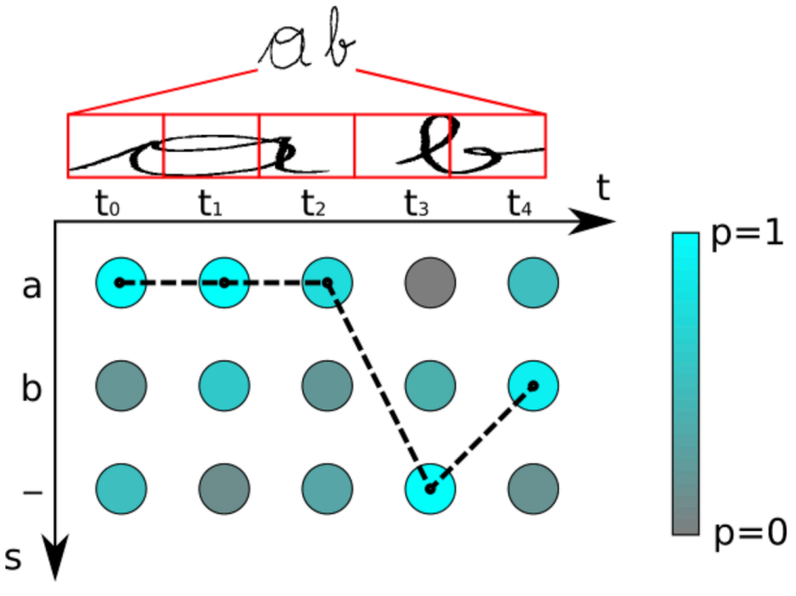
Connectionist Temporal Classification

- The CTC provide two key elements to decode the output model into the character sequence of characters of a handwritten image.
  - A CTC decoder that convert the output model in terms of a sequence of probability vectors into the final sequence of character that appears in the handwritten image.
    - Calculating the best path based on the sequence of probabilities.
    - Removing duplicates and blanks.
  - A CTC loss that measure the distance between a sequence of vectors of probabilities of occurrence of each available character and the correct target sequence of characters.
- A good explanation about CTC: https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c
- The original paper of Graves that introduce the CTC can be found here https://dl.acm.org/citation.cfm?doid=1143844.1143891

CTC loss example



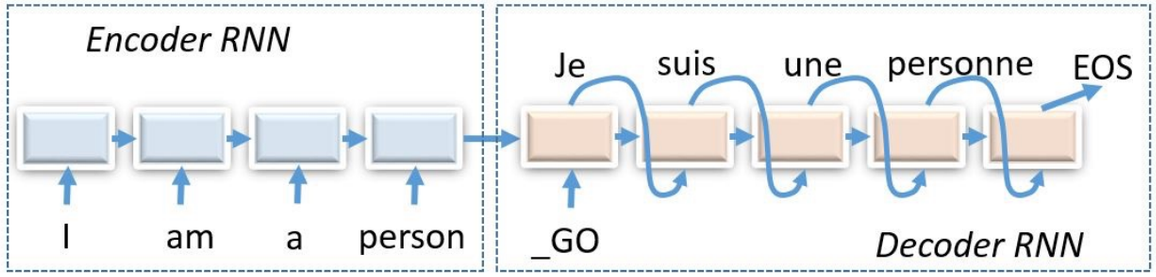
CTC decoder example



https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c

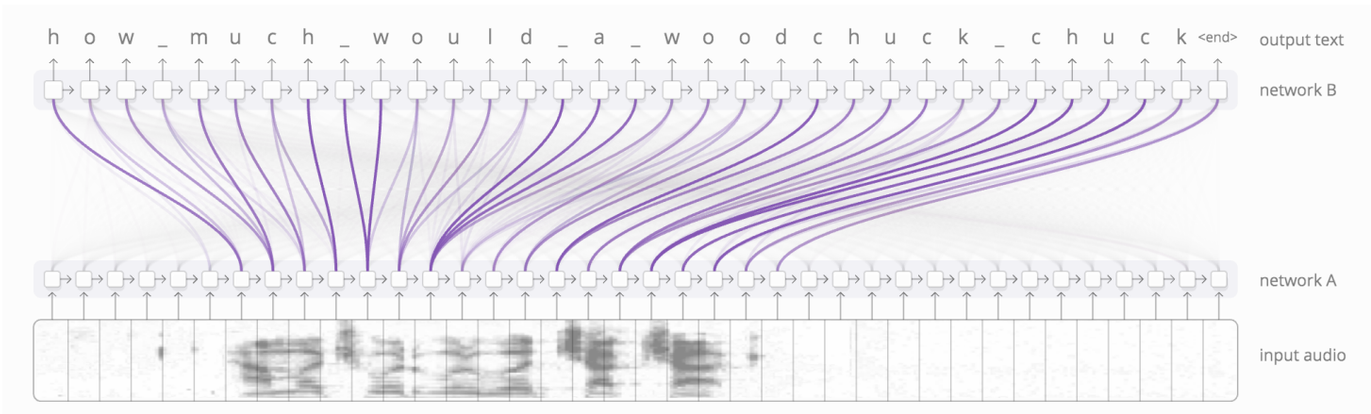
Seq2seq architecture

- Very general framework to train models that predict sequences based on sequences.
- This architecture have 3 main components:
  - The encoder: Very dependent of the input data
    - For input text can be and embedding and a RNN
    - For video can be CNN and RNN
- The decoder: A RNN (optionally bidirectional and/or stacked).
- The attention mechanism: To connect locally the encoder sequence output to the decoder sequence input.



esciencegroup.com

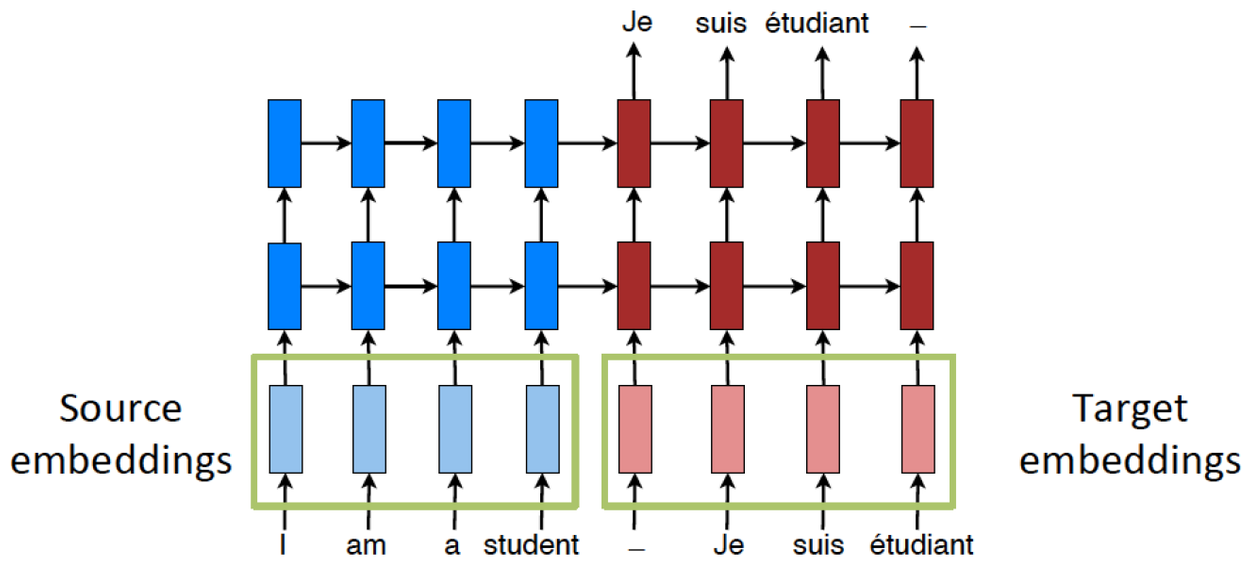
Attention mechanism



http://distill.pub/2016/augmented-rnns/#attentional-interfaces

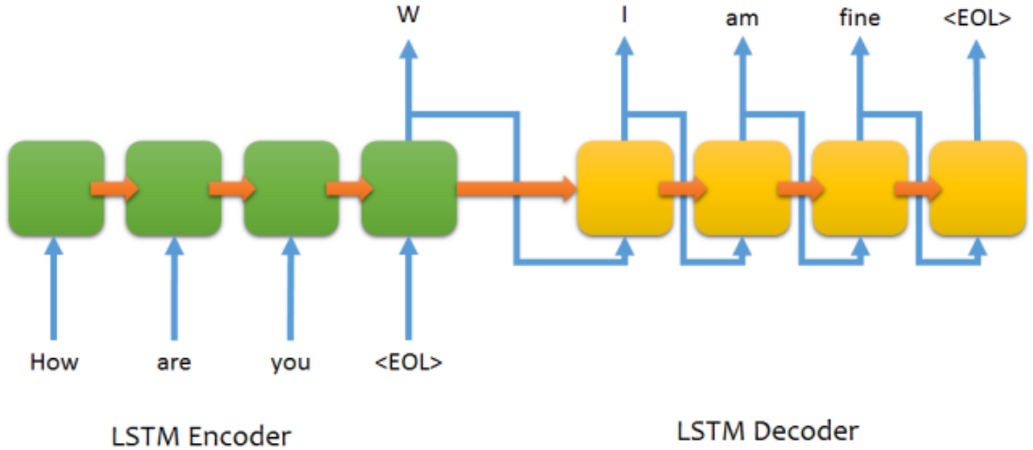
Seq2seq example: Language translation

- Machine translation:
  - Encoder with embeddings in the source language
  - Decoder with embeddings in the destination language
- The embeddings can be learned from scratch (big corpus) or can be used form word2vect or glove.
- Example: All the models necessary to build an instant speech translation in real time can be builded with the seq2seq architecture:
  - Step 1: seq2seq model to convert speech to text. Example: https://github.com/pannous/tensorflow-speech-recognition
  - Step 2: seq2seq model for machine translation of text between 2 languages. Example: https://www.tensorflow.org/versions/r0.12/tutorials/seq2seq/index.html
  - Step 3: seq2seq model to convert text to speech. Example: https://github.com/mikesj-public/rnn_spelling_bee/blob/master/spelling_bee_RNN.ipynb



Seq2seq example: Question answering

- Usually a language model at word level.
- Build model to an encoder to read the question and a decoder to propose the answer.
- In dialog systems, include the previous conversations in the input of the model.



Quoc: What is the purpose of life?
Machine: To serve the greater good.

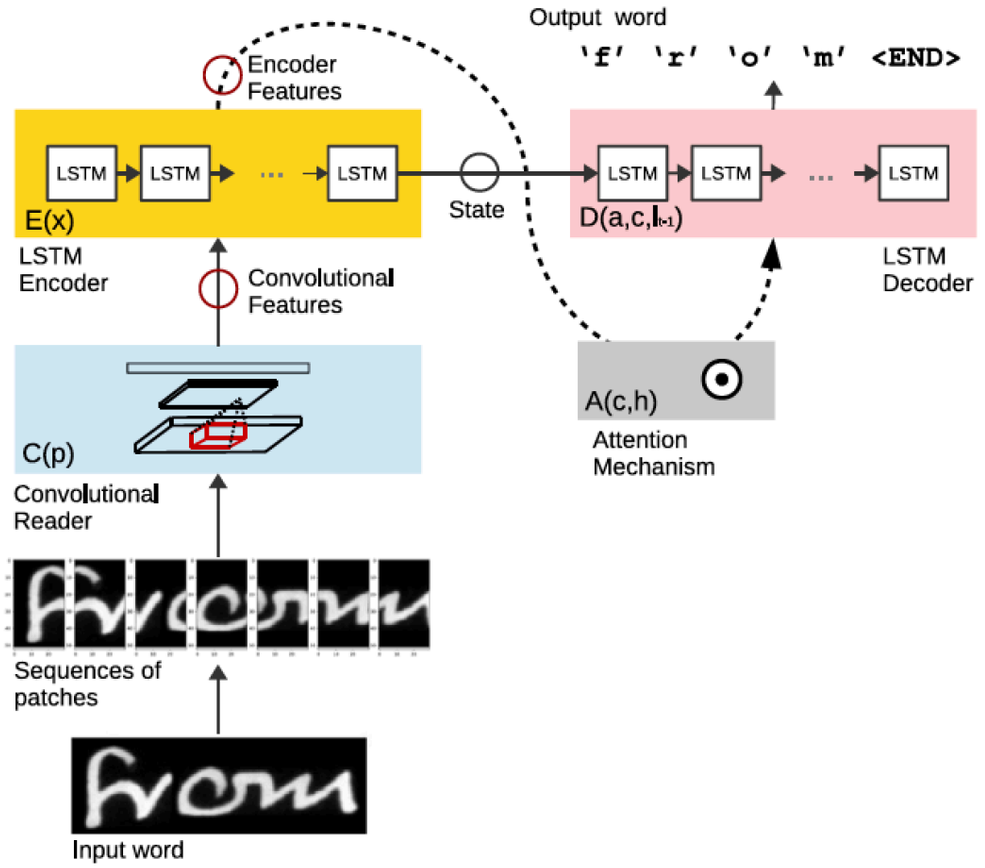
Quoc: What is the purpose of living?
Machine: To live forever.

Quoc: What is the purpose of dying?
Machine: To have a life.

Vinyals, Le. A Neural Conversational Model.
ICML Deep Learning Workshop, 2015.

seq2seq model to handwriting recognition

- Data normalization: Normalize word images.
- Patch extraction: Extract overlapped horizontal crops from word images
- Model: With seq2seq architecture
    - Convolutional features: To extract general features from crops
    - Encoder: encode convolutional features viewing it as sequence.
    - Decoder: Generate the output word character by character.
    - Attention mechanism: Connect decoder and convolutional features to put the attention in specific parts.
- Experiments
    - Datasets: IAM and RIMES databases.
    - patch extraction strategy: Size and step
    - Encoder size: Num of LSTM layers and size
    - Results: Interms of CER and WER
    - Error analysis: by word size and decoding dictionary.

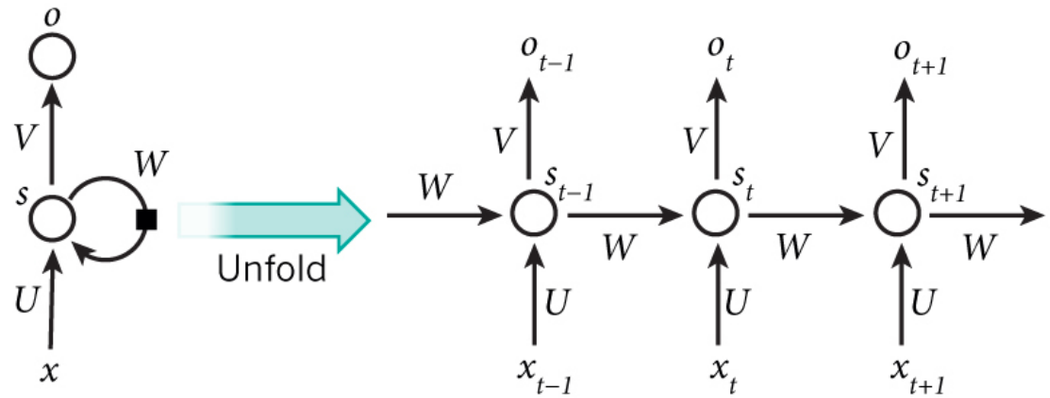


Intro to Recurrent neural networks

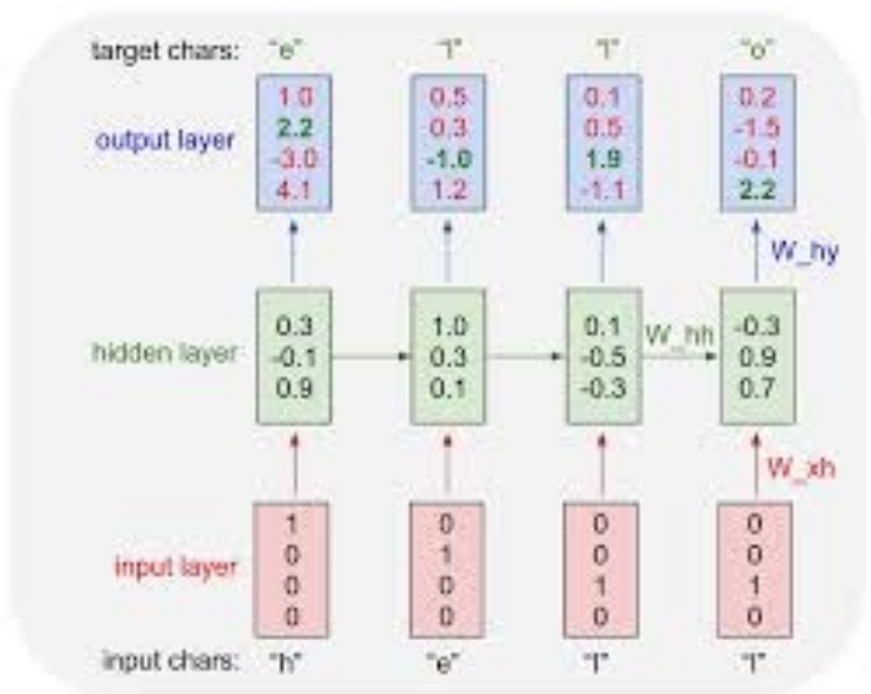
- Sequential data: Text, video, time series, events, …

- Recurren neural networks: Models with memory that can store the relevant aspects of the sequence to predict the target.

- Is possible to combine to itself to create deep architectures and with other types, for example, with convolutions to combine text and image o sequences of images as video.

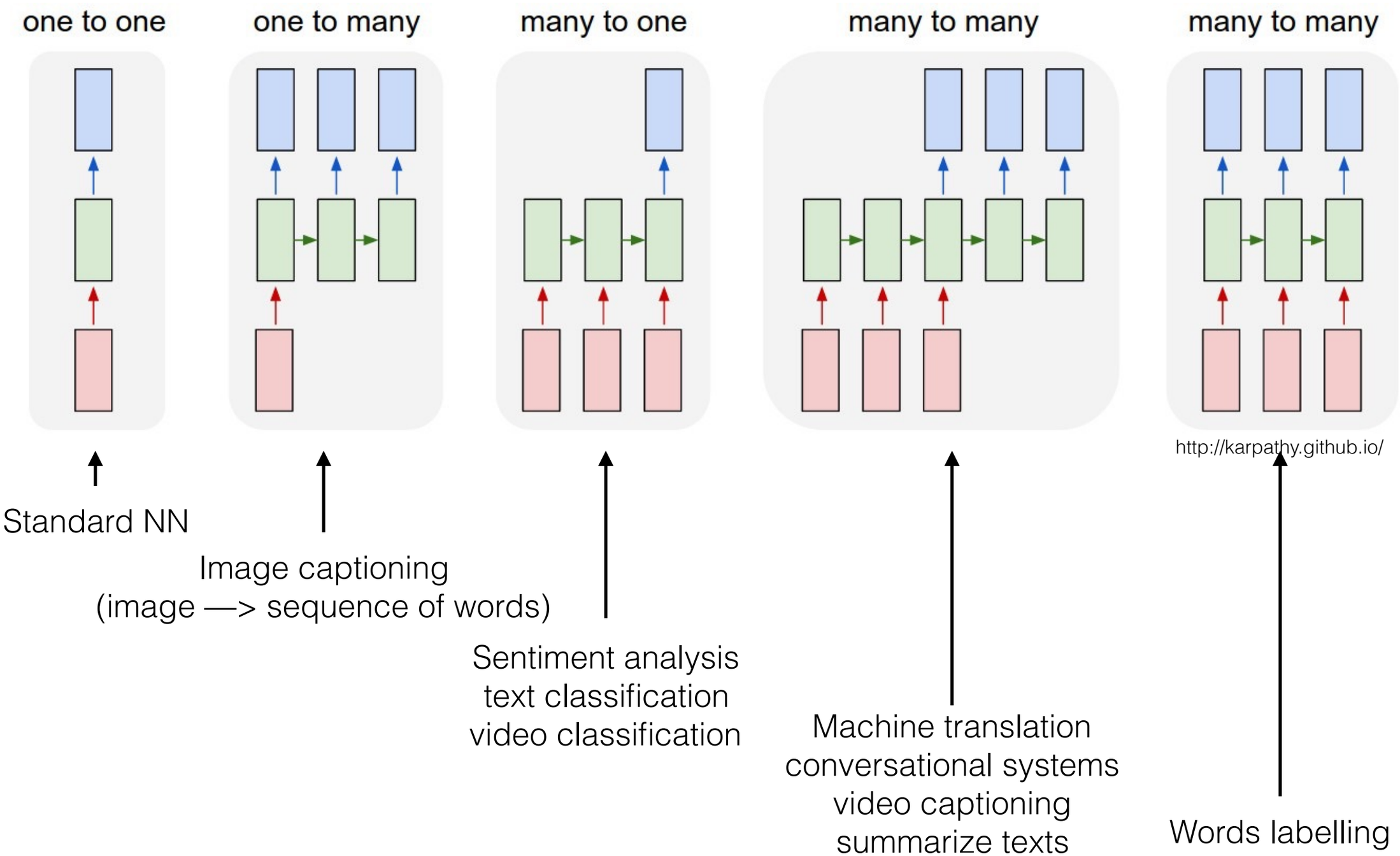


wildml.com

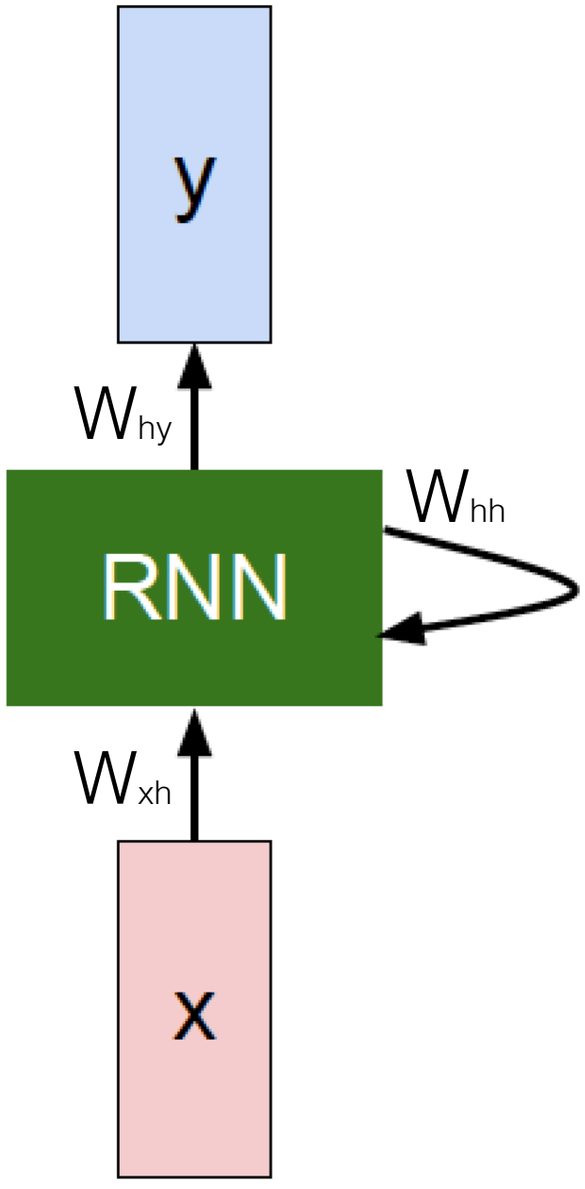


http://karpathy.github.io/

RNN - Flexibility



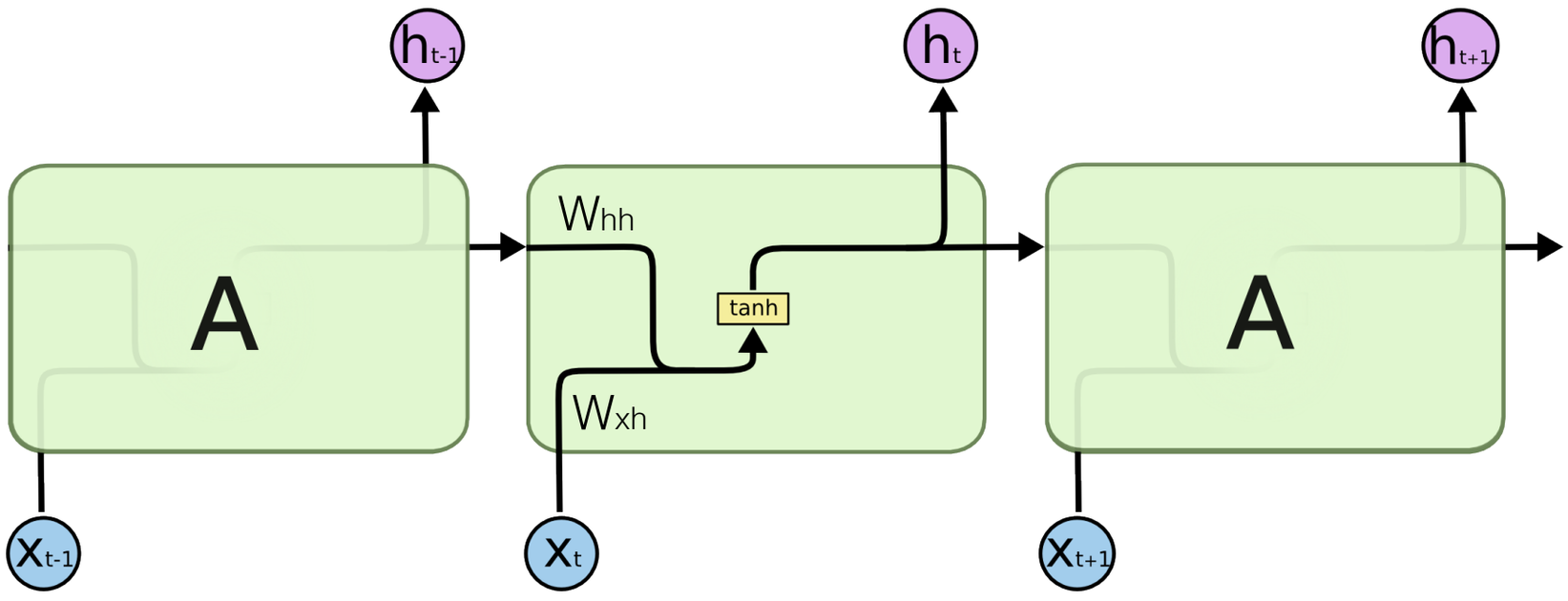
RNN - Basic equations



$$h_t = f_W(h_{t-1}, x_t)$$

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

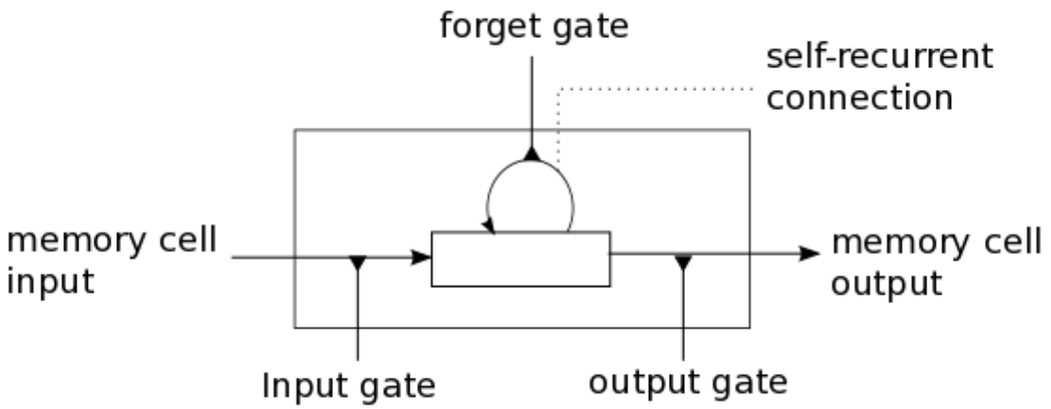
$$y_t = W_{hy} h_t$$



LSTM

- Basic RNNs have problems to store info about long sequences (vanishing gradient problem).
- The layers LSTM and GRU solve this problem incorporating to the layer sobe memory cells that can store long sequences.
- A LSTM layer have 4 cells:
  - The standard hidden cell
  - An input cell that control how the new input affect to the internal cells
  - An output cell that control how the internal cell affect to the output.
  - A memory cell that control if we need to forgot something in the hidden cell or not.

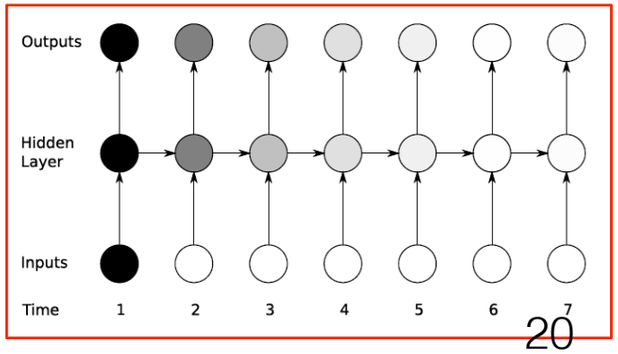
LSTM cell



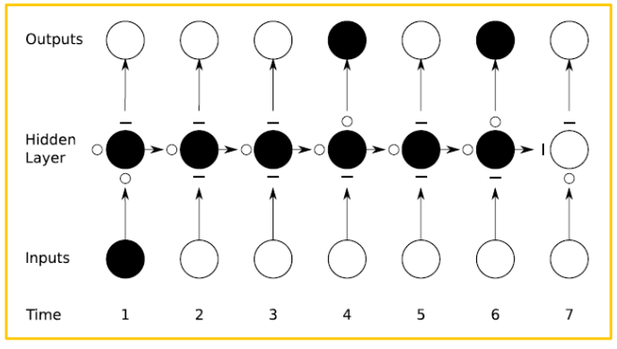
http://deeplearning.net/

Vanishing gradient problem

Standar RNN



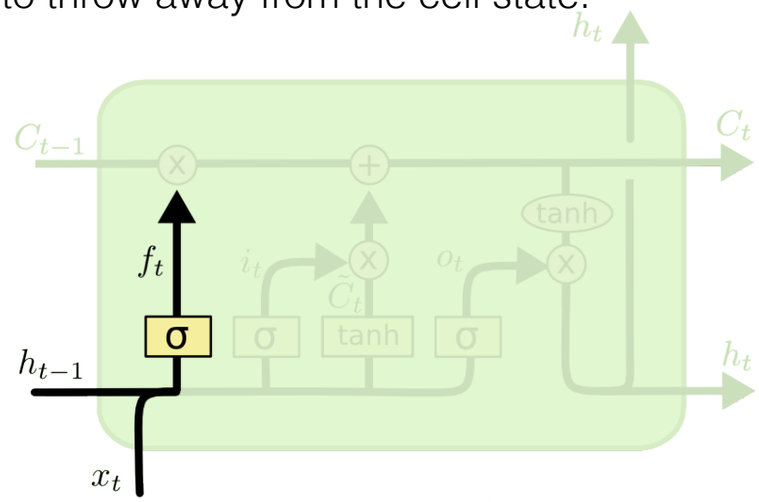
LSTM



LSTM Steps

The **Forget gate** decide what information we're going to throw away from the cell state.

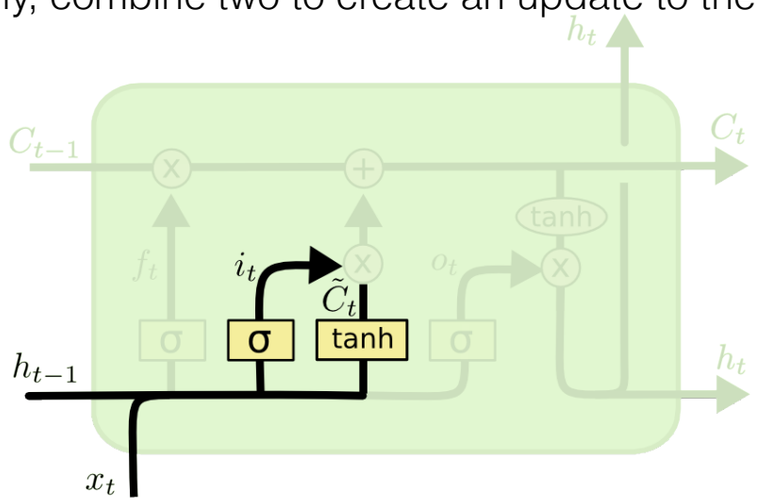
1



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Decide the new information to store in the cell state
First, **Input gate** decide which values we'll update
Next, **tanh layer** creates a vector of new candidate values
Finally, combine two to create an update to the state

2

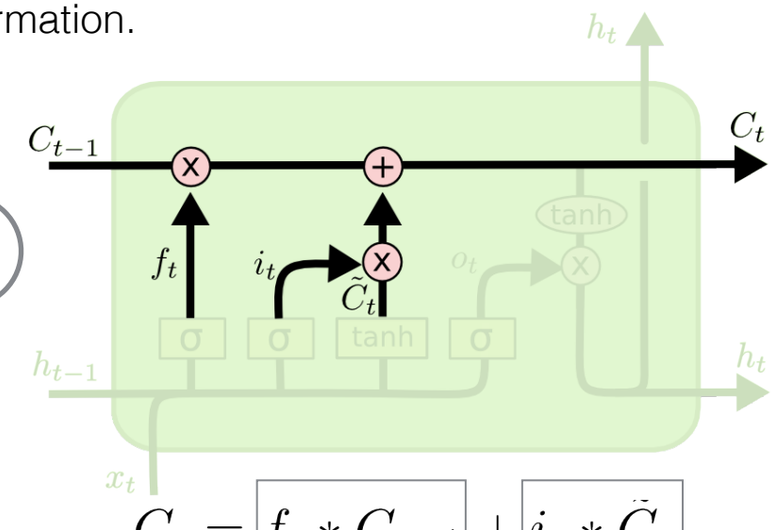


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Update the internal state whit the forget and input gates information.

3

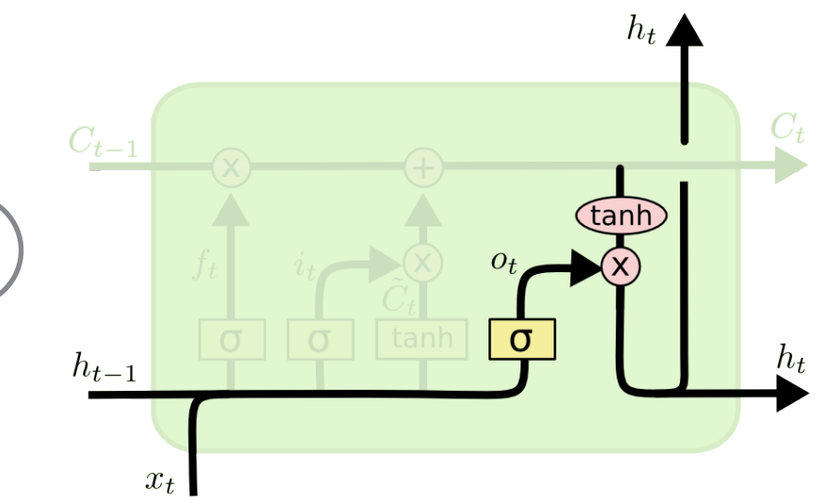


$$C_t = \boxed{f_t * C_{t-1}} + \boxed{i_t * \tilde{C}_t}$$

Forget the old Add the new

Output will be based on cell state

4



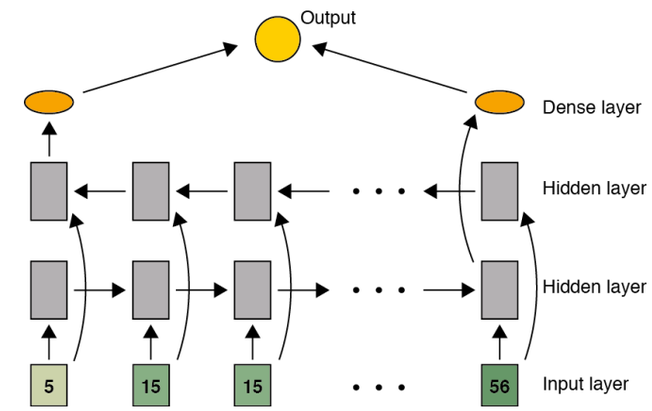
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

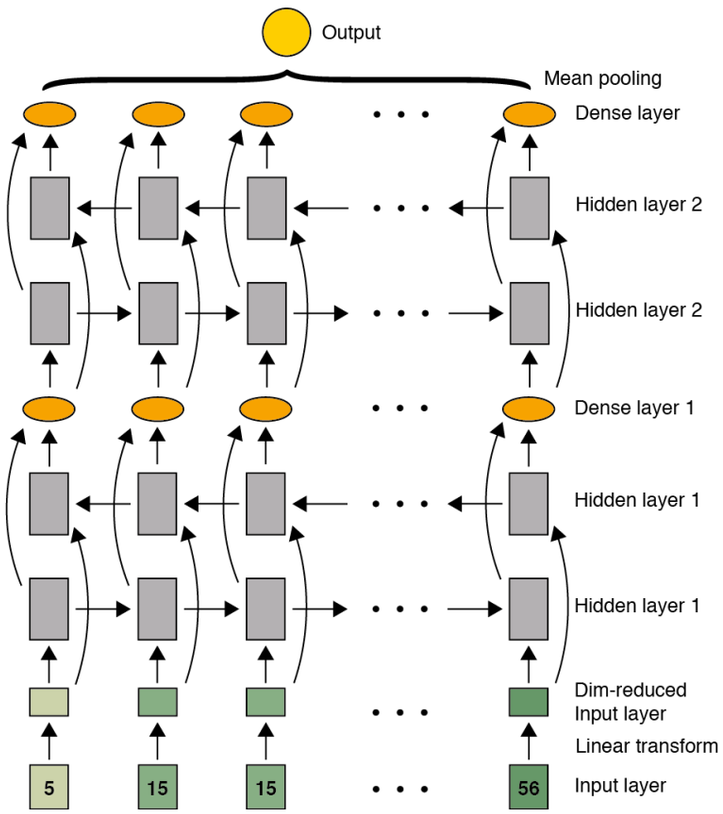
Bidirectional LSTM & Stacked LSTM

- To increase the complexity of the RNN network
- Bidirectional LSTM (BLSTM): Model that ‘Read’ the input sequence in both directions and combine the 2 outputs
- Stacked LSTM: Add additional RNN layers over the output of the previous RNN layers
- Stacked BLSTM: Combination of the 2 previous strategies.

Bidirectional LSTM



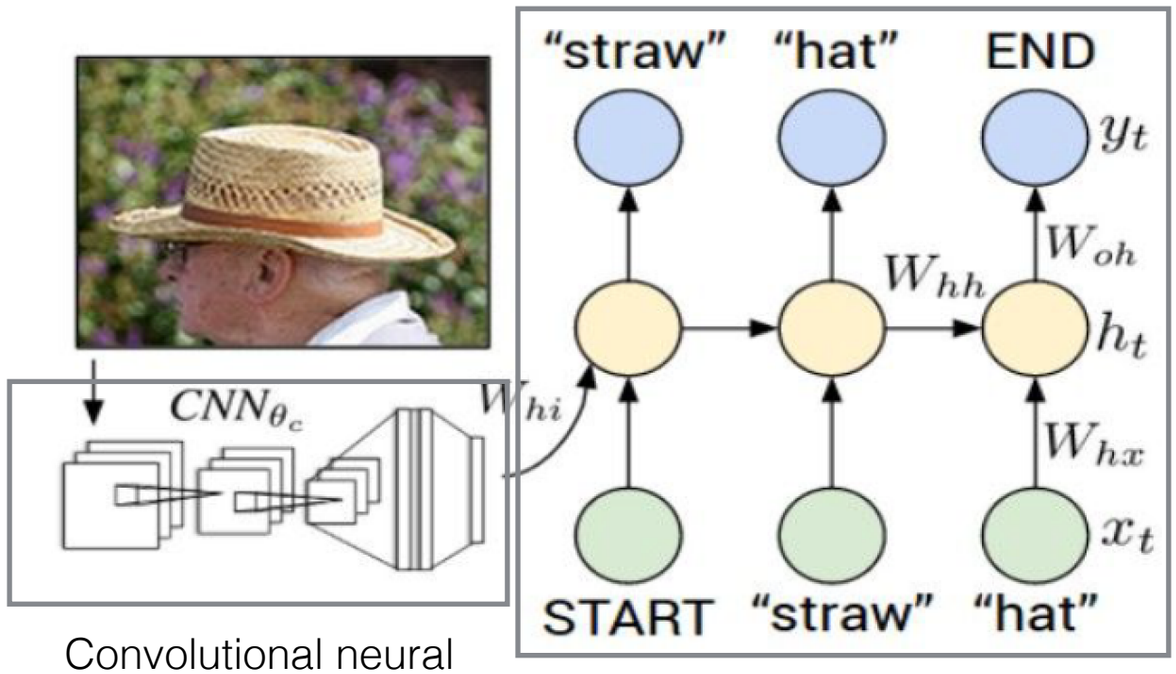
Stacked BLSTM



http://simaaron.github.io

Example of usage of RNN Image captioning

- Describe images in natural language.
  - CNN to extract visual features.
  - RNN to generate description using the last CNN layer as input.
- Online demo: http://deeplearning.cs.toronto.edu/i2t
- Fun video: https://vimeo.com/146492001



Convolutional neural network

Recurrent neural network

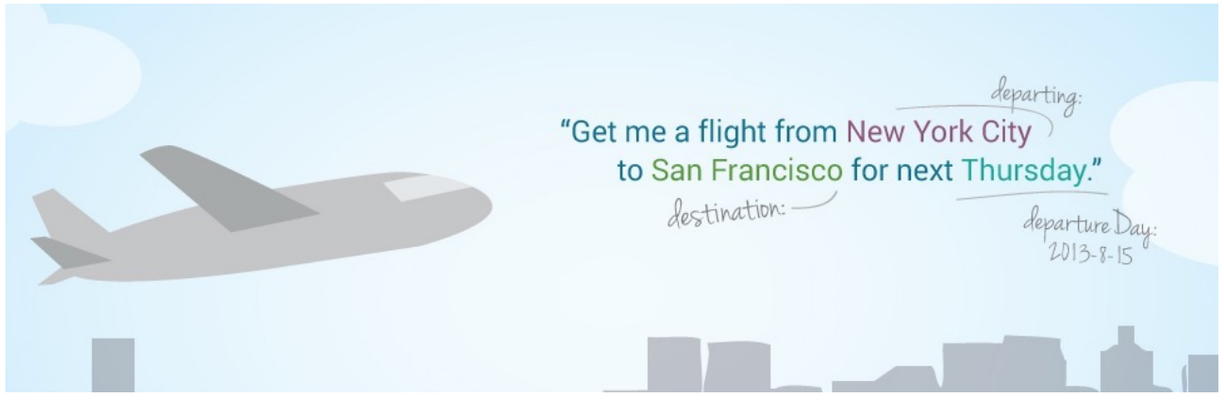
Is possible to use pertained image classification models

Example of usage of RNN Words labelling

- Many-to-many Recurrent neural network.
- Label name entities or other words/sequences classification criteria.

In 1917, Einstein applied the general theory of relativity to model the large-scale structure of the universe. He was visiting the United States when Adolf Hitler came to power in 1933 and did not go back to Germany, where he had been a professor at the Berlin Academy of Sciences. He settled in the U.S., becoming an American citizen in 1940. On the eve of World War II, he endorsed a letter to President Franklin D. Roosevelt alerting him to the potential development of "extremely powerful bombs of a new type" and recommending that the U.S. begin similar research. This eventually led to what would become the Manhattan Project. Einstein supported defending the Allied forces, but largely denounced using the new discovery of nuclear fission as a weapon. Later, with the British philosopher Bertrand Russell, Einstein signed the Russell-Einstein Manifesto, which highlighted the danger of nuclear weapons. Einstein was affiliated with the Institute for Advanced Study in Princeton, New Jersey, until his death in 1955.

Tag colours:
LOCATION TIME PERSON ORGANIZATION MONEY PERCENT DATE



The	committee	,	as	usual	,	has
O	O	O	B_ESE	I_ESE	O	B_DSE

refused	to	make	any	statements	.
I_DSE	I_DSE	I_DSE	I_DSE	I_DSE	O

direct subjective expressions **(I_DSE)**
expressive subjective expressions **(I_ESE).**

Summary

- The handwriting recognition problem
- **seq2seq solution**
    - Preprocessing data
    - Model details
    - Experiments, results and error analysis
- Research lines
- Code example

Preprocessing: Normalization

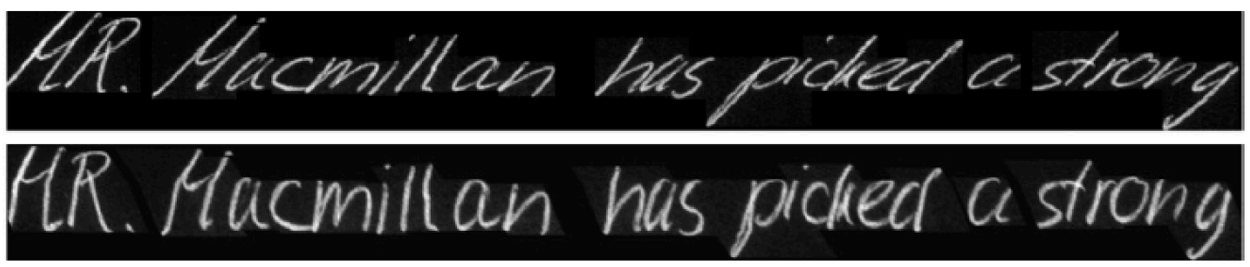
- Normalization at line level with this steps:
  - Identified the baseline: Use Random Sample Consensus (RANSAC) regression over the lower points of each column in the image of a given line
  - corpus line: Same as baseline over the higher points
  - corrected the line skew: computed the baseline angle to apply an affine transformation
- corrected the slant
  - first identified the slant angle.
  - Correct the slant using the affine transformation matrix
- normalize the height of the characters based on baseline and corpus line: resizing it over the corpus line and under the baseline to a half of the height of the main part of the line
- Finally we resize the image to 48 pixels of heigh and maximum of 250 pixels of with keeping the aspect ratio unless the word are too long, in this case we modify the aspect ratio (only very few cases).

Baseline and corpus line examples

Affine matrix to slant correction

$$\begin{pmatrix} 1 & -\alpha & 0.5 \cdot w \cdot \alpha \\ 0 & 1 & 0 \end{pmatrix}$$

Example of normalization

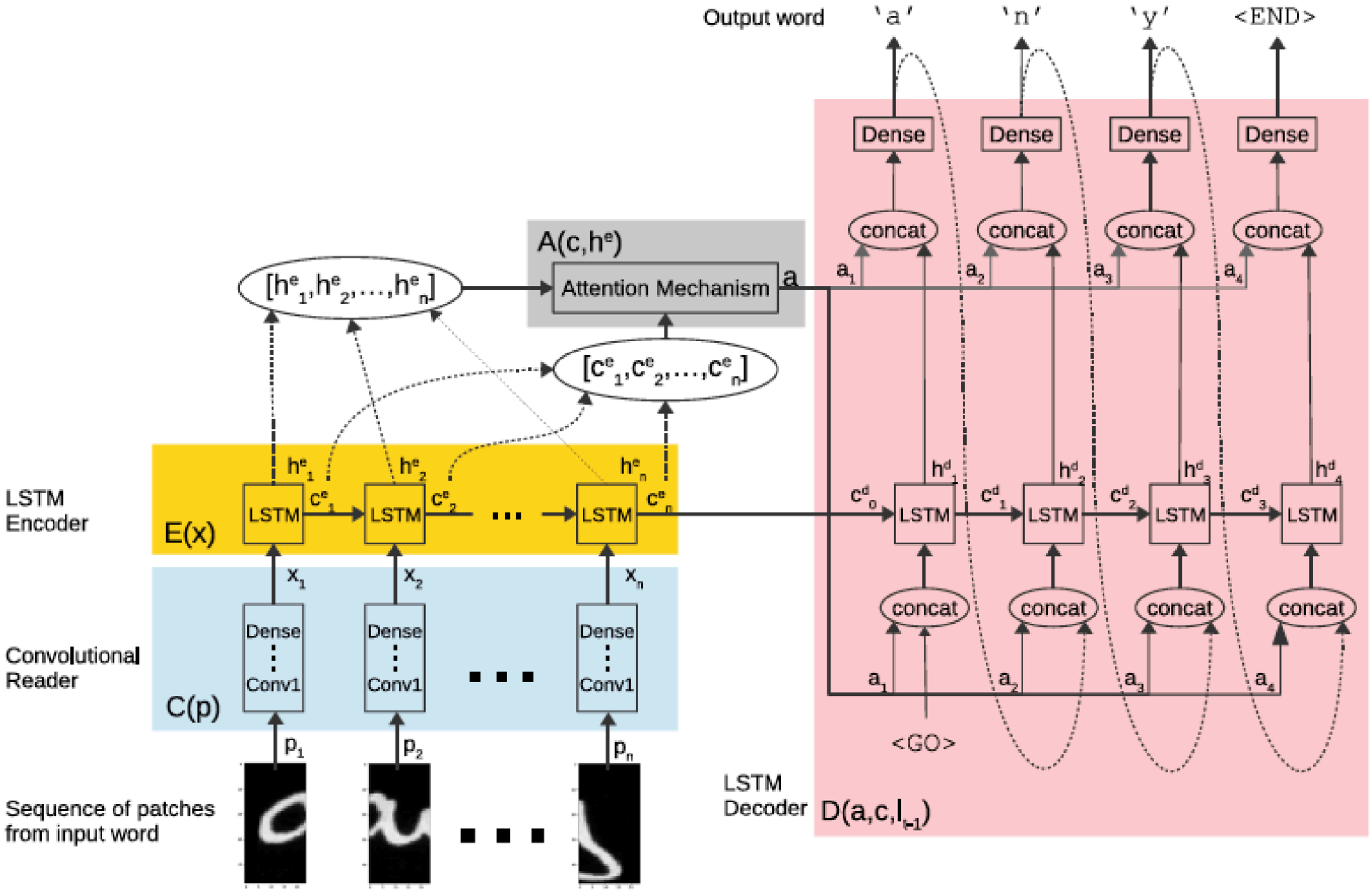


Preprocessing
patch extraction

- The input model is't the normalized word image, but a sequence of horizontal overlapped slides extracted from it. These slides can include a part of a letter, one letter or more than one letters and are the inputs of the model and they enable the model to view the word image not globally but as a sequence of patches. These patches are characterized by two parameters:
  - The patch size (i.e. its horizontal size if the patch)
  - The patch step (i.e. the number of pixels that we move to the right to extract the next patch).
- Strategy inspired by two of the main cognitive psychology models relative to handwriting word recognition:
  - The Serial Letter Recognition (SLR) propose that words are read letter by letter, serially from left to right.
  - The Parallel Letter Recognition (PLR) propose that letters within a word are recognized simultaneously, and this joint information is used to recognize the word.

Example of patch extraction

Detailed architecture



Model: Convolutional part

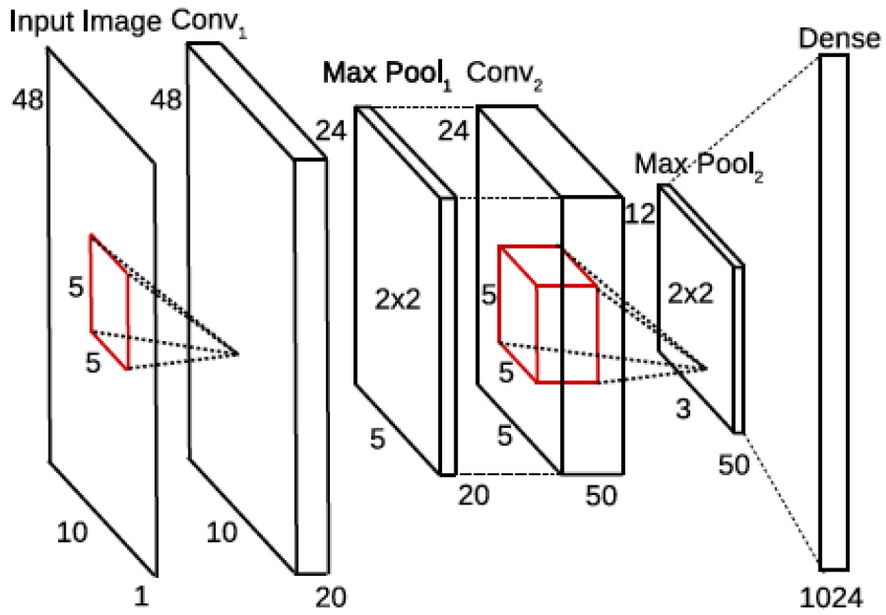
- This component of the network use a deep convolutional architecture (CNN) stacking two groups of one convolutional and one max pooling layers and finishing with one dense layer. This architecture was inspired in the LeNet-5 architecture to isolate handwriting recognition.

- In the image we use input patches of size 48x10 in grayscale and a dense size of 1024. We experiment with different dense sizes to optimize this parameter.

- The same convolutional architecture was applied in each patch sharing weights between these components of the model.

- the output of this CNN is a sequence of vector features [x_1, …, x_n] that encapsulates the visual features of the patches sequence extracted from a handwritten word image.

Convolutional architecture



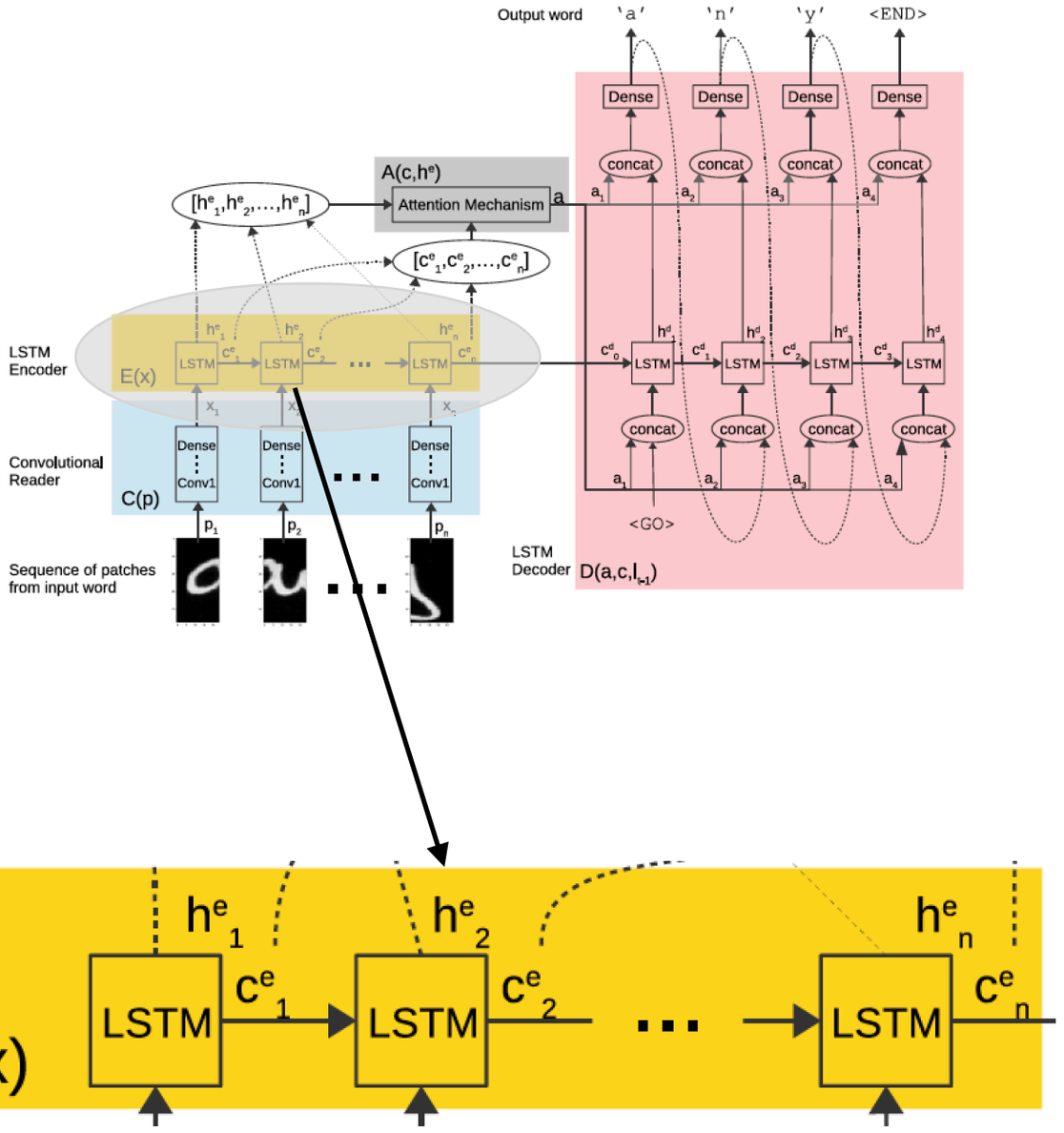
Experiments with different dense sizes

Comparative of WER results for different dense layer sizes of the convolutional reader.

Database	Dense size		
	512	1024	2048
IAM	21,1	**18,9**	Not convergence
RIMES	13,6	**12,8**	13,2

Model: Encoder

- Component model to capture the horizontal relationships existing among the sequence of overlapped horizontal patches (summarized as dense features by the convolutional part)
- The encoder is implemented using LSTM layers. Experiments with:
  - LSTM layer size
  - Unidirectional or bidirectional LSTM
  - Number of stacked LSTM layers
- The encoder receives the sequence of vector features *x*, that represent each input image patch *p* generated by the convolutional reader, and generates two outputs:
  - The sequence of output vectors *h*
  - The sequence of state vectors *c.* The last output state was used as the initial state of the decoder layer.
- The final state is not enough to pass all the relevant encoder information to the decoder. To do this we add an attention mechanism to the model.

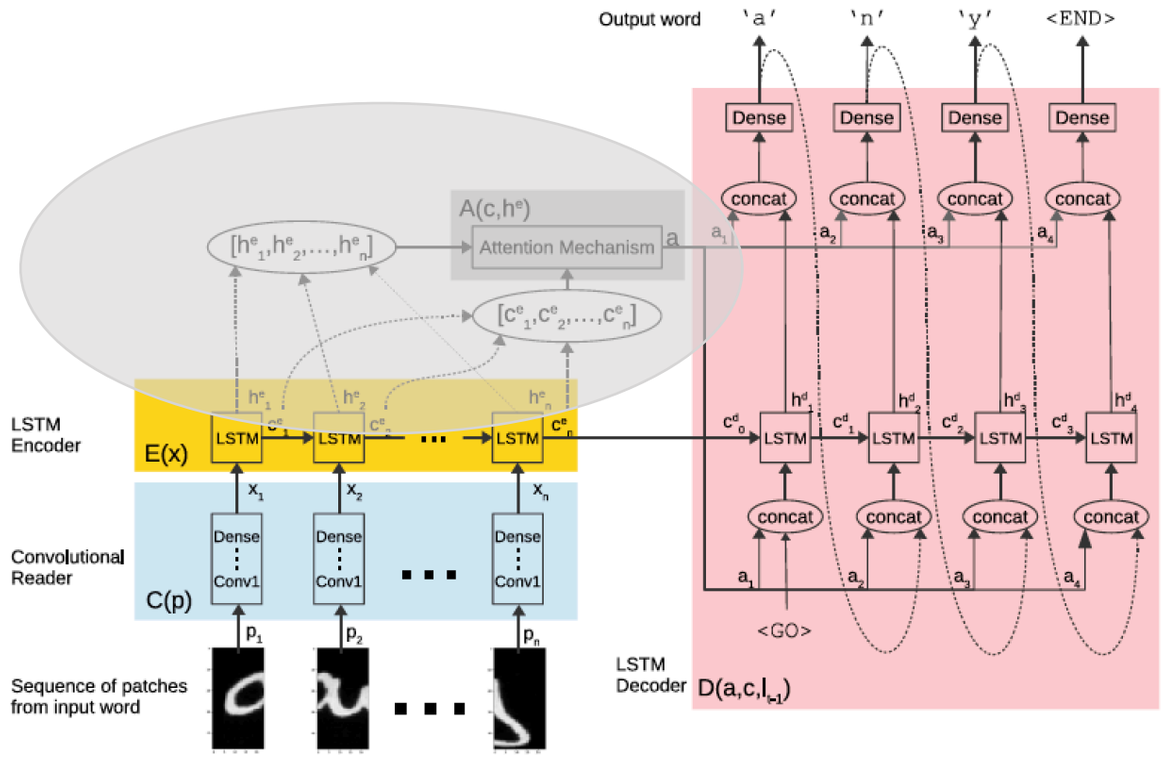


Model: Attention

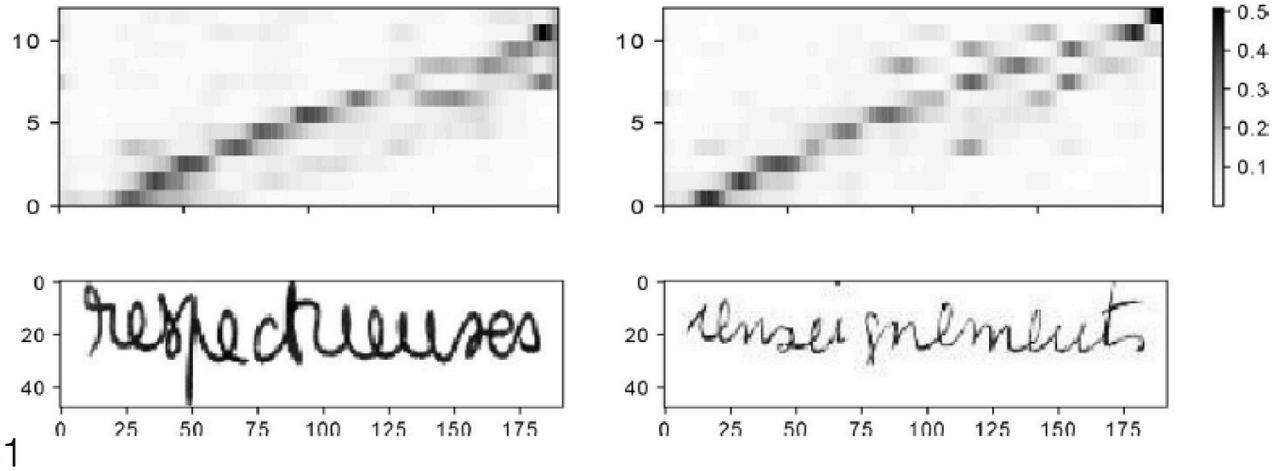
- This mechanism allows the decoder to access to the features of the LSTM encoder at every decoding step.

- The attention mechanism is implemented as a neural network layer that combines the states and the outputs of the encoder to use this data when each character are decoded.

- We select an attention mechanism without monotonicity restrictions because reading letters of a word from left to right is a conventionalism but one algorithm to read are more optimal if can access to all the info about the word in each step.

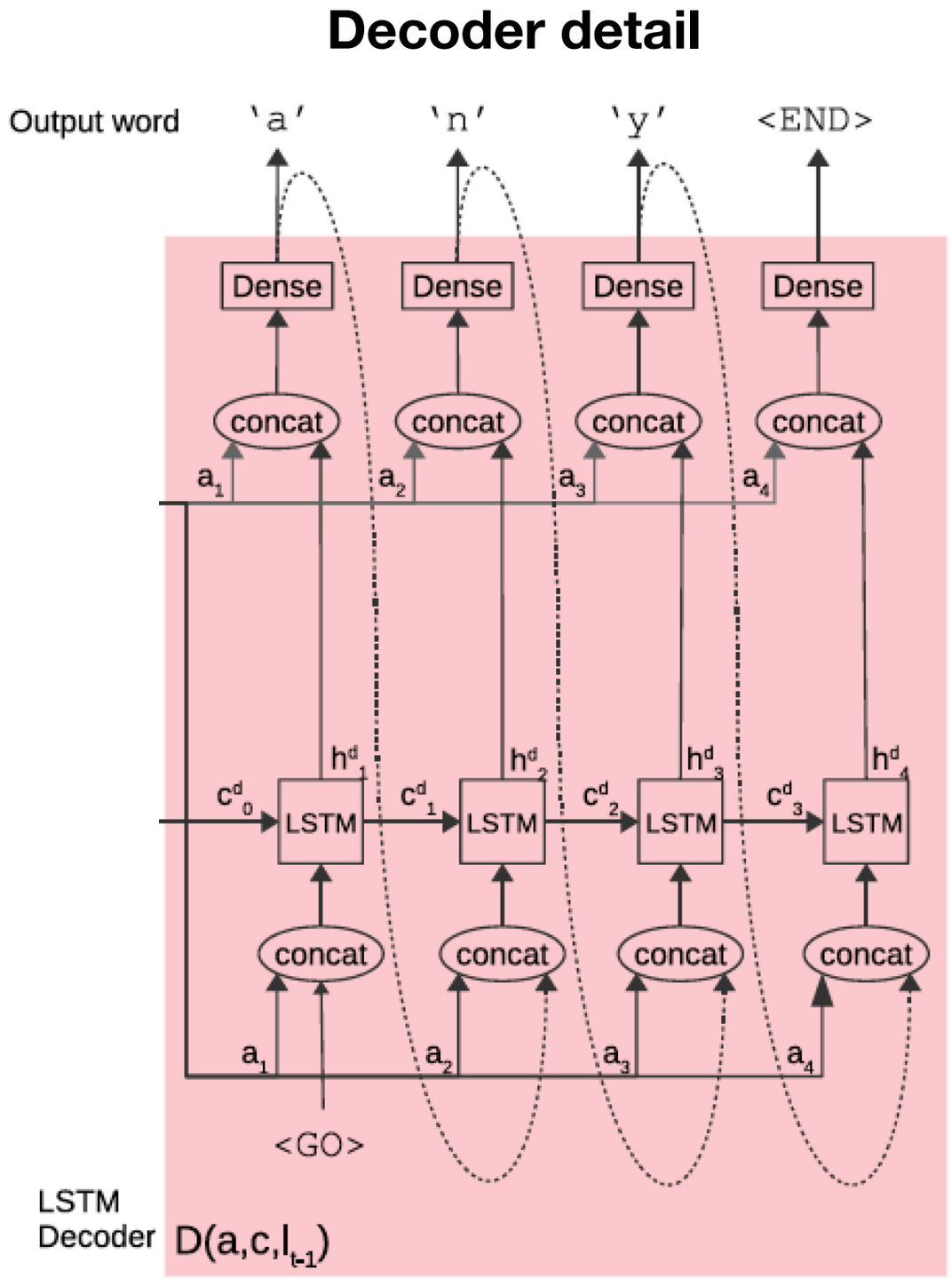


Examples of attention outputs



Model: Decoder

- Builded with a LSTM layer follow to a dense softmax layer do decode each character of the word. The initial stare of the LSTM layer was the end state of the encoder.
- We use two special characters into the decoding process:
  - <GO>: Input for the first step of the decoder.
  - <END>: The decoder predict this at the end of the current word.
- In training we have 2 possible strategies for the inputs of the decoder:
  - Use the real previous character.
  - Use the previous output, a vector of probabilities (image).
- In test the input of each step must be the output vector of the previous step.
- As input of the LSTM and dense layers we concat the attention output a to allow this layers to access to the full output sequence of the encoder.



Model training details

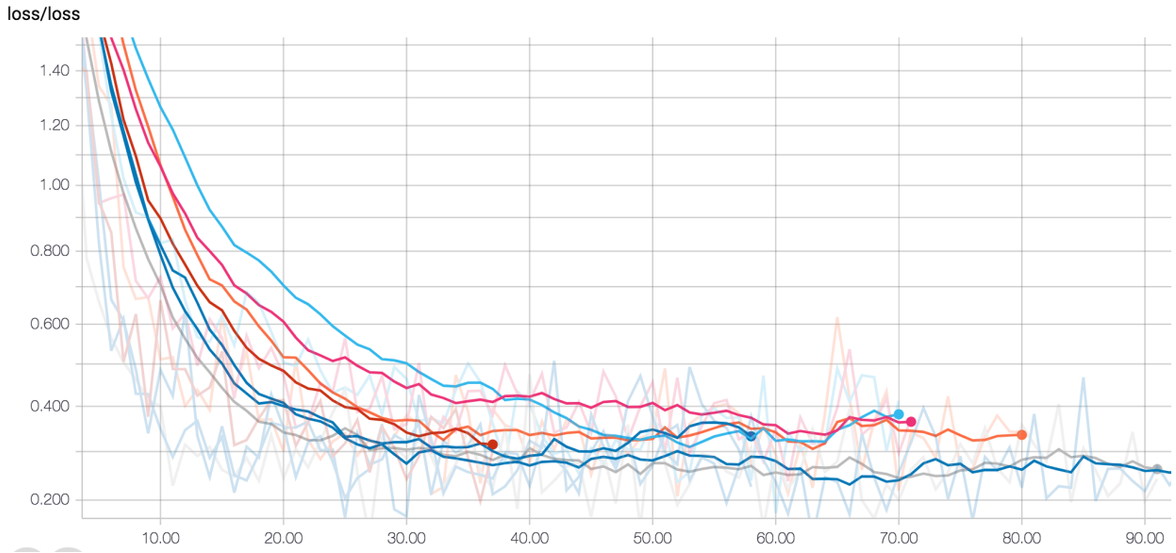
- The model was Implemented in Tensorflow. The input images was re-size to 48x192 completing the left with blanks to keep the aspect ratio.

- Also we implemented an online randomized data augmentation process in the training step that included:

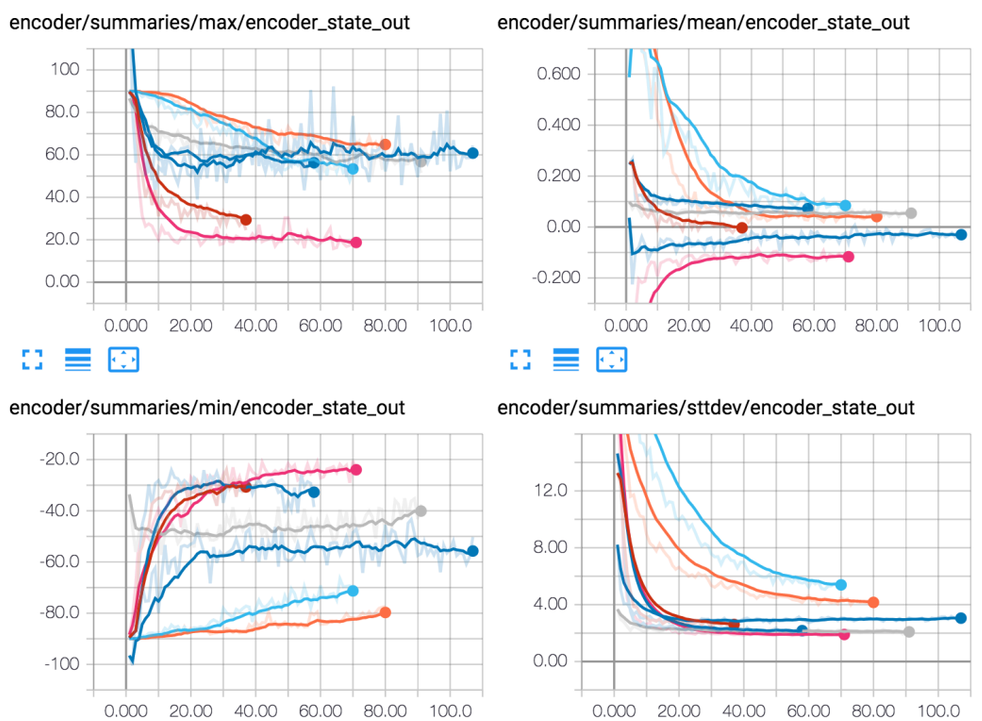
  - ±10 percent of zoom,
  - ±10 percent of displacement,
  - ±10 degrees of skew,
  - 3 ×3 dilation and erosion transformations.

- An initial learning rate of 0.001 was used and this rate was decreased by 2% in each epoch. We applied a stopping criteria of 10 epochs without any improvement in the Word Error Rate (WER) on the validation set. The different experiments took between 60 and 120 epochs (between 10h and 20h of execution) on a NVIDIA Titan X Pascal GPU.

Validation loss by epoch



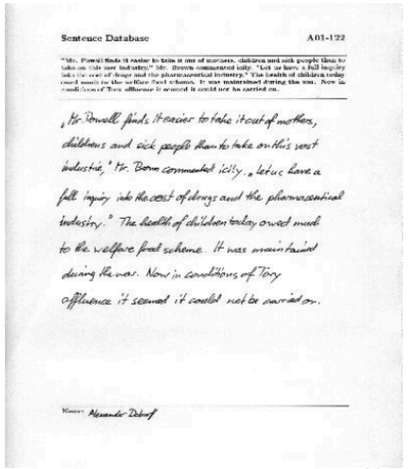
Output encoder stats by epoch

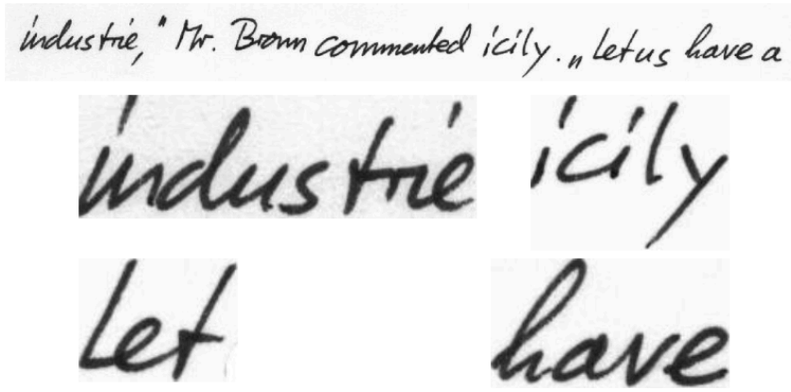


Experiments: Datasets

- IAM offline handwriting database.
  - Composed by 1,066 forms produced by approximately 400 different writers based in the LOB corpus. A total of 82,227 word instances out of a vocabulary of 10,841 words.
  - The standard partition include 55,081 words in train, 8895 in validation and 25,920 in test.
  - http://www.fki.inf.unibe.ch/databases/iam-handwriting-database
- RIMES database
  - Handwritten letters from simulated French mail. 1,300 people who have participated to the RIMES database creation by writing up to 5 mails. The RIMES database thus obtained contains 12,723 pages corresponding to 5605 mails of two to three pages.
  - The partition used includes 51,739 words to train, 7464 words to validation and 7776 words to test,
  - Several versions available at http://www.a2ialab.com/doku.php?id=rimes_database:start. We used the ICDAR 2011 version provided by the A2IA Lab.

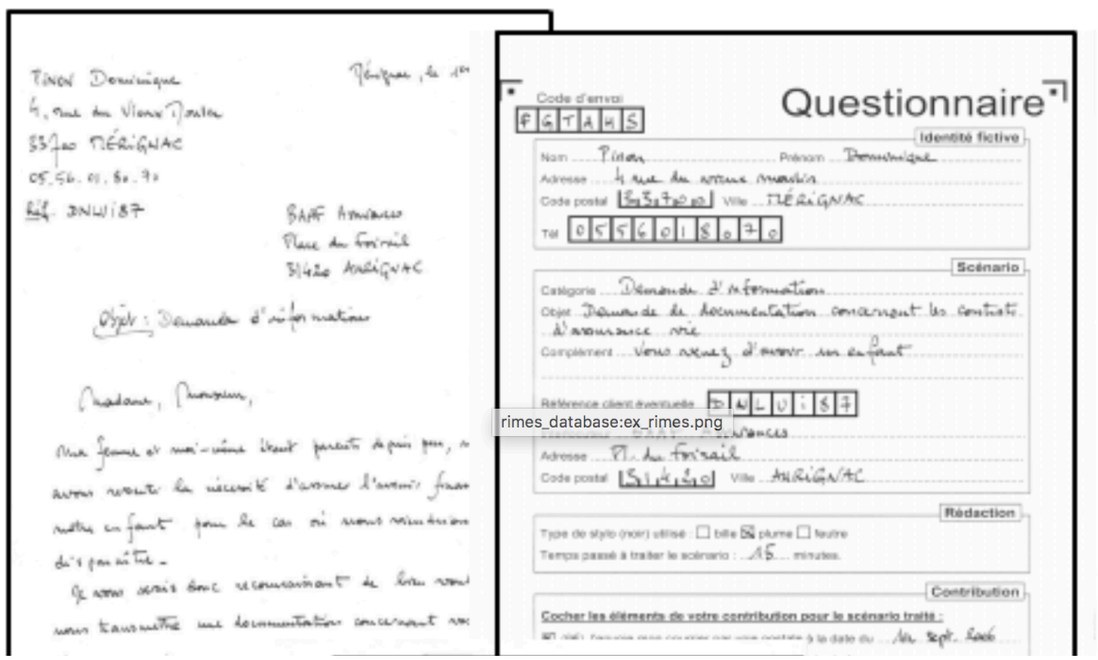
IAM dataset images





http://www.fki.inf.unibe.ch/databases/iam-handwriting-database

RIMES dataset images



http://www.a2ialab.com/doku.php?id=rimes_database:start

Experiments: patches partition strategy

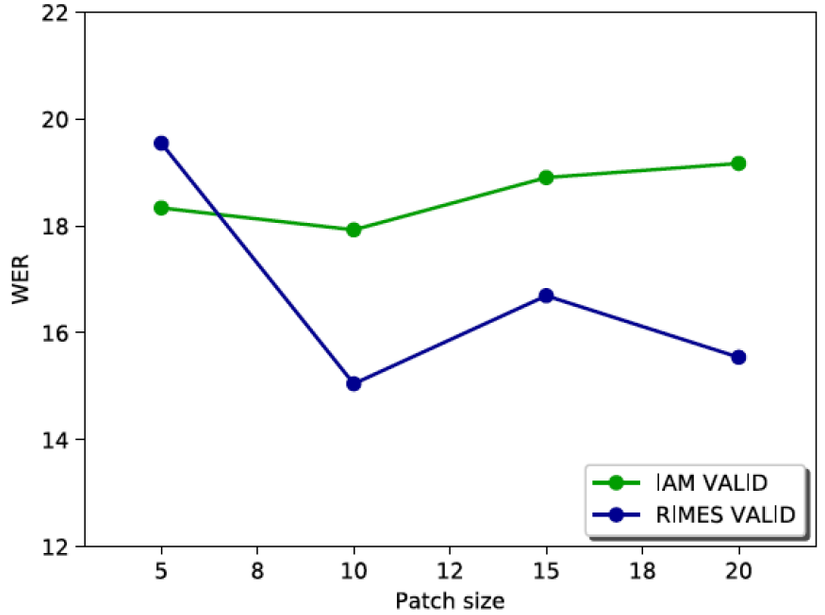
- The first set of experiments find optimal parameters for the creation of horizontal patches of the input word images.

- Two main parameters to control the patch strategy optimized sequentially:

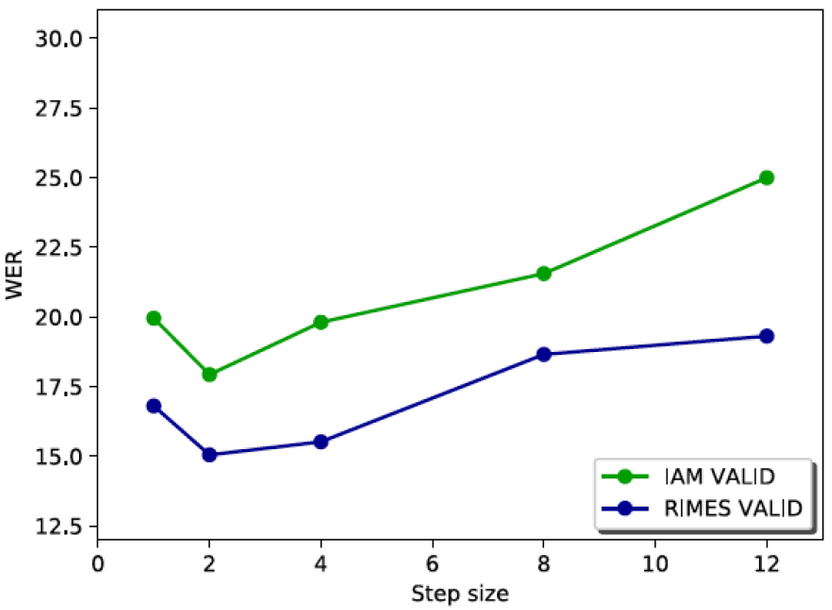
  - Patch size: Width of the patches in pixels. We experiment with: 5, 10, 15 and 20 pixels. Optimal value achieved in size 10

  - Step size: Left pixels to move to extract the next patch. Experimentes with 1, 2, 4, 8 and 12 pixels. Optimal value achieved in one step of 2 pixels.

Validation WER by patch size



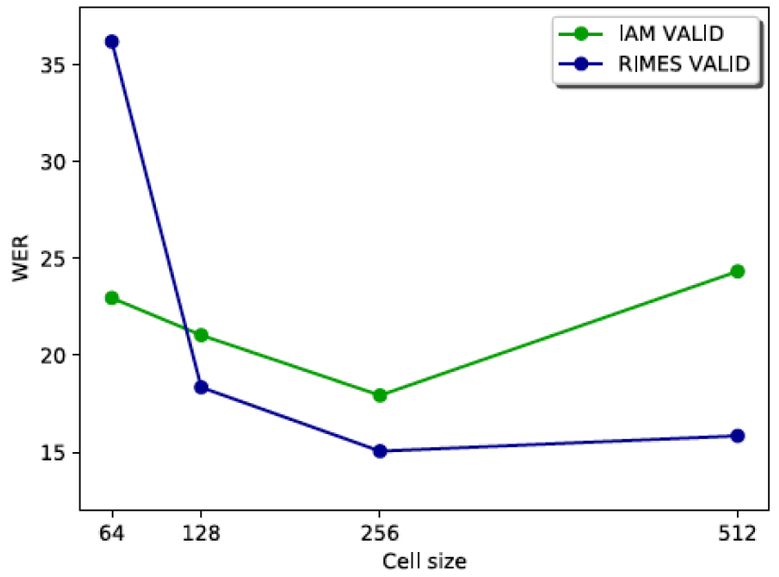
Validation WER by step size



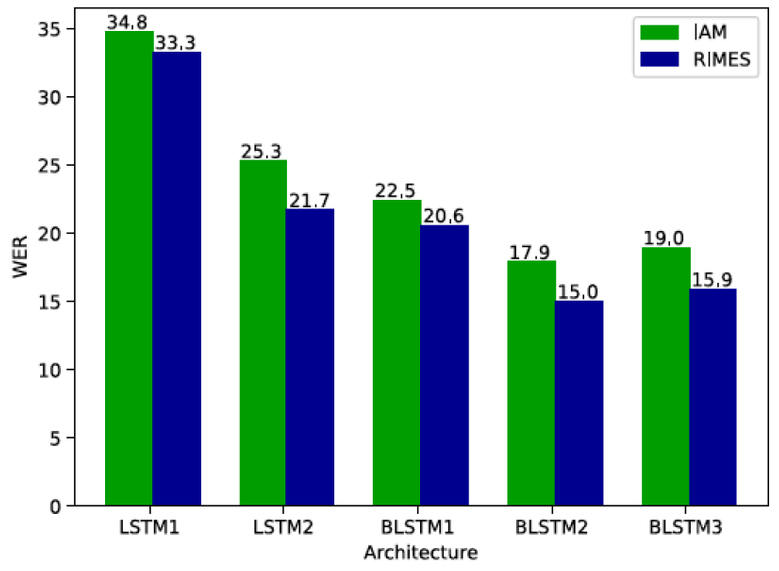
Experiments: encoder size

- Over an optimal patch partition strategy we experiment with different options to the encoder size and architecture.
  - Cell size of LSTM encoder layers: Experiments with sizes of 64, 128, 256 and 512. optimal in 256.
- Architecture of encoder, we experiment with One directional and bidirectional architectures and with one, 2 or 3 layers. Optimal achieved with a bidirectional LSTM wit 2 stacked layers.

Validation WER by encoder cell size



Validation WER by encoder architecture



Experiments: results

- In the problem of off-line handwritting recognition, the databases and the partitions are common.
- The main problem to comprare the current results with other publications are the variability of post processing types performed over the model output to finally identify the correct word in the test set.
    - Different lexicons
    - Different language models over different corpus
    - Test augmentation or not.
    - Etc.
- In the paper we propose 3 of the more standard comparatives easy to reply exactly and user by other authors
    - Compare the direct results of the model, with any post processing.
    - Find the output model in a standard lexicon for each dataset using Levenshtein distance metric.
        - 50000 more frequent words of the LOB+Brown corpus for IAM
        - train + validation corpus in RIMES case.
    - Find the output model in the test lexicon for each dataset using Levenshtein distance metric. This metric help to measure the out of vocabulary case.

Results

Table 2
Comparative of results of the visual model without lexicon.

Database	IAM		RIMES	
	WER	CER	WER	CER
Our method	**23.8 ± 0.05**	8.8 ± 0.02	15.9 ± 0.4	4.8 ± 0.1
Pham et al. [33]	35.1	10.8	28.5	6.8
Bluche [37] MLP	54.2	15.6	59.5	17.8
Bluche [37] RNN	24.7	**7.3**	20.9	5.6
Bluche [36]	24.6	7.9	**12.6**	**2.9**

Table 3
Comparative of results of the visual model with a standard lexicon.

Database	IAM		RIMES	
	WER	CER	WER	CER
Our method	19.7 ± 0.03	9.5 ± 0.03	**13.1 ± 0.2**	5.7 ± 0.1
Bluche [37] MLP	25.5	8.0	26.1	7.2
Bluche [37] RNN	**16.7**	**5.3**	16.4	**4.3**

Table 4
Comparative of results of the visual model with the test lexicon.

Database	IAM		RIMES	
	WER	CER	WER	CER
Our method	12.7 ± 0.03	6.2 ± 0.02	6.6 ± 0.2	2.6 ± 0.1
Almazan et al. [39]	20.1	11.2		
Bluche et al. [40]	20.5		9.2	

Experiments: Error analysis

- In the firs graph the errors increases with the word length. This is related with the OOV influence, because the words with less letters typically appears in the lexicons.

- The second graph clean the influence of the OOV. We show that the most of the errors in long words are originated by the absence of the correct word in the lexicon.

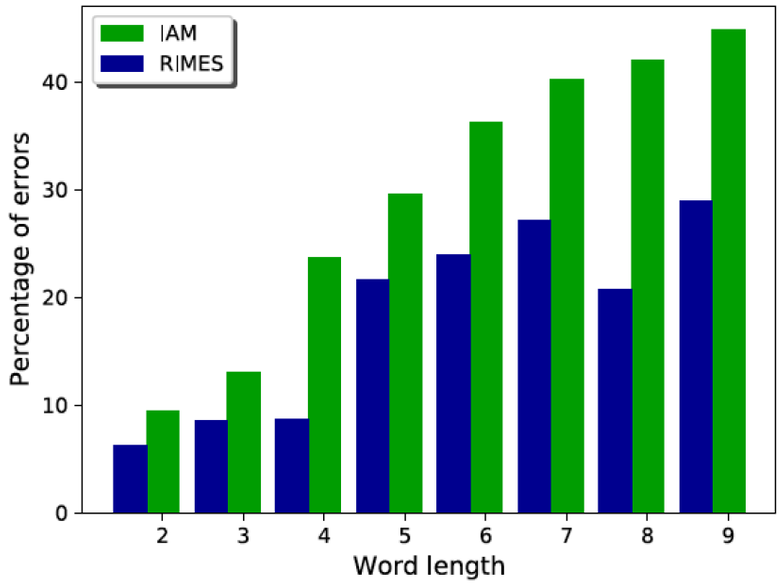
- The errors by the replacement of only one character represent the 68.6% of errors of the visual model in IAM and the 69.3% in RIMES.

- The errors by the confusion between the same character but one in lowercase and the other in uppercase.

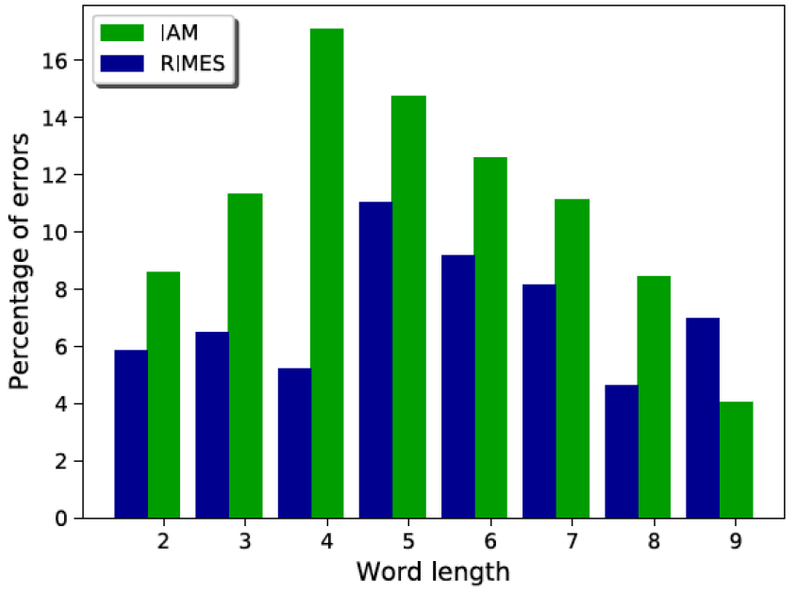
  - The 5% of errors in IAM and the 4.7% in RIMES are of this type.

  - Usually in characters with similar shape in user and lowercase as: ‘f’, ‘c’, ‘v’, ‘l’, ‘k’ or ‘w

WER in test vs word length



WER in test vs word length (test lexicon)



Summary

- The handwriting recognition problem
- seq2seq solution
- **Research lines**
  - Synthetic generation of handwriting text
  - Other architecture examples: Convolutions + CTC
- Code example

Synthetic handwriting generation using typography

- The most important problem in handwriting recognition are the high variability of the data. And the non existence of datasets that cover this variability.

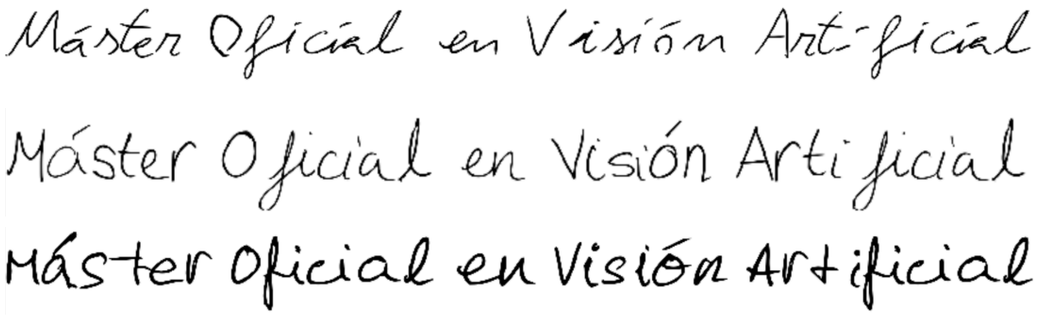
- One way to avoid this are generate synthetic data that can include more variability into the models.

- We experiment using typographies based on the standard of OpenType https://en.wikipedia.org/wiki/OpenType . That include random selection of the glyphs between several cases and connections inclusion.

- In any case the synthetic generation methods tested have also limitations in variability because the seed that these methods need to start have also variability limitations:

    - In the case of the typography the seed are the samples of characters used to generate each typography.

Examples of synthetic handwriting text



Template example to build one typography

Generating handwritting with Recurrent Neural Networks

- Graves (2014) use online handwriting data (x,y trajectories of) to train a model to generate handwriting trajectories using as seed the sequence of characters to draw.

- Use network based on RNNs to predict the next position (x,y) based of the sequence of previous positions and the character sequence to draw.

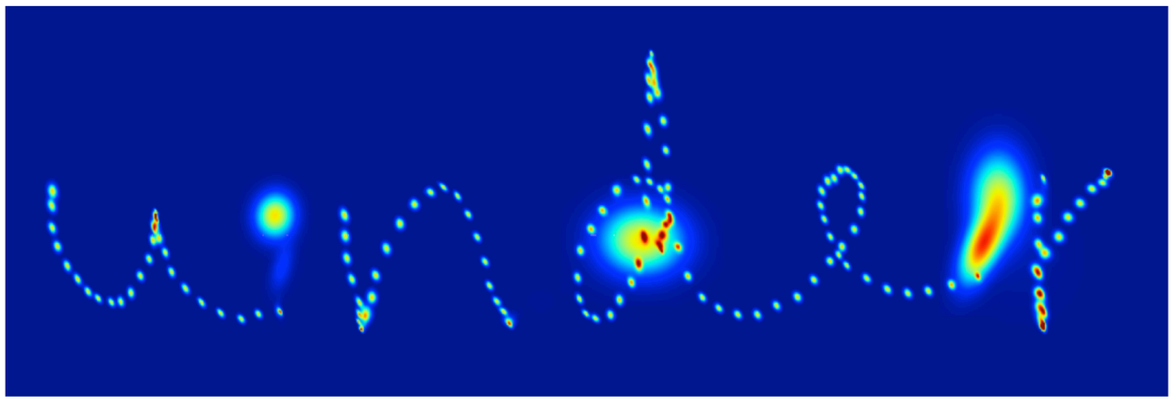
- Use mixture density networks to generate the trace (the output of the model was the parameters of probability density functions).

  - Mixture of several bivariate gaussians to predict (x, y)

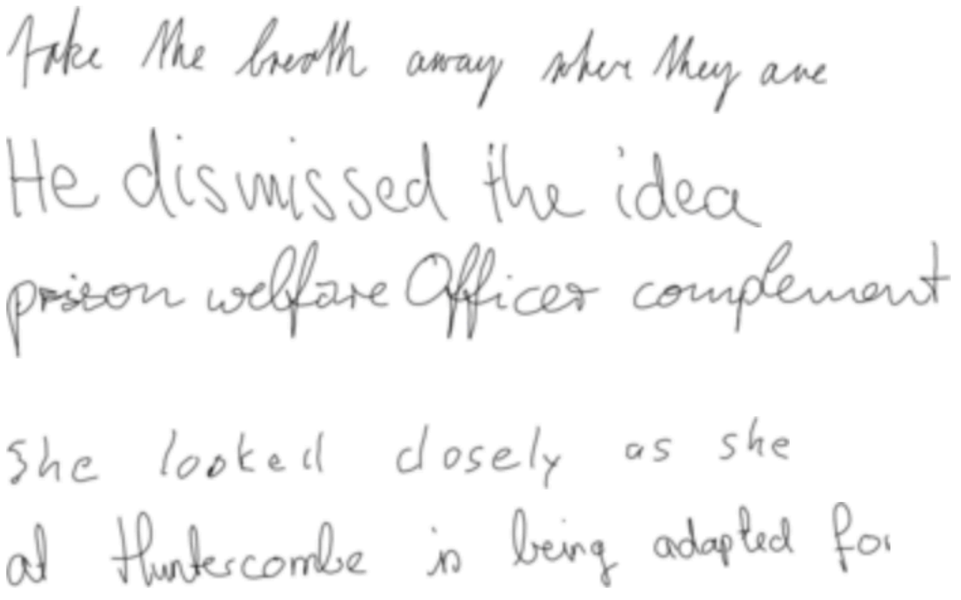
  - One Bernoulli for the pen_up, pen_down events.

- Demo an paper in: https://www.cs.toronto.edu/~graves/handwriting.html

Mixture density outputs



Samples of text generated



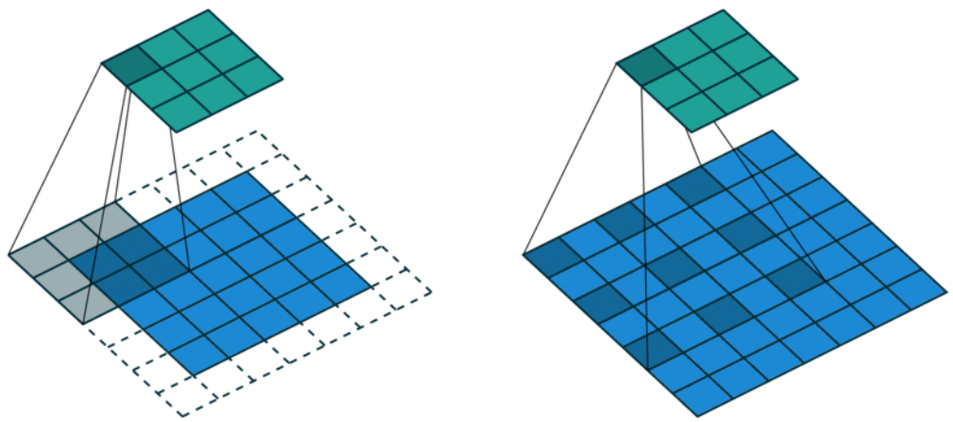
New architectures

- Use dilated convolutions to get features at different resolutions form handwriting text images.

- Use it also to capture long dependencies in sequences of features with significant less parameters that the RNN.

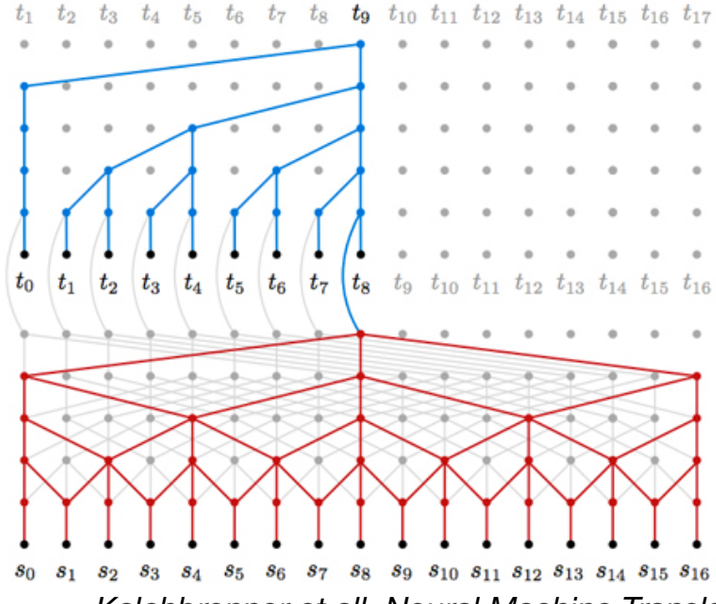
- The dilated convolutions have success application to encoding and decoding complex sequences of audio and text. Kalchbrenner et all. Neural Machine Translation in Linear Time. https://arxiv.org/pdf/1610.10099.pdf

Normal convolution **Dilated convolution**



https://towardsdatascience.com/review-dilated-convolution-semantic-segmentation-9d5a5bd768f5

Dilated convolutions to model sequences



Kalchbrenner et all. Neural Machine Translation in Linear Time

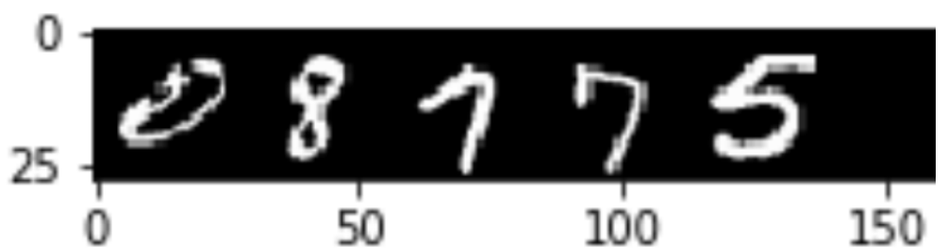
Summary

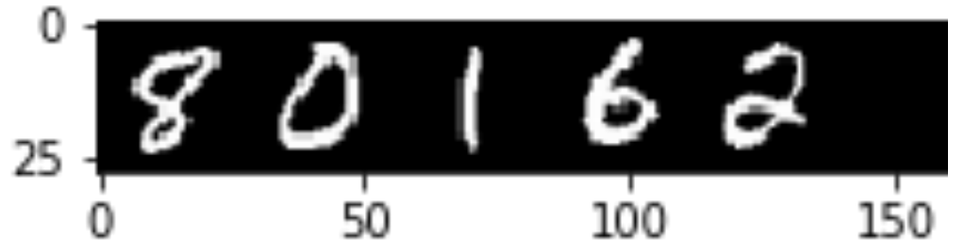
- The handwriting recognition problem
- seq2seq solution
- Research lines
- Code example
  - Handwriting recognition with seq2seq models

Data generation

- Create images of NMIST sequences
  - Concat MNIST digits. The image size are fixed, but the sequence length are variable.
- Input data to the model:
  - Sequence of slices of the images generated
  - Number of slices no blank. Length of the previous sequence
  - Target: As a sequence of chars with fixed size, completed with ‘zeros’ at the end
  - Target length: Number of elements of the target sequence.
  - Weights sequence: Different weights from each decoder output to the loss calculation (The error in the firsts characters are more relevant)
- Epoch generator:
  - Return batches from the previous data
  - Because we are generating synthetic sequences, we need to define the epoch size.

Example of image sequences generated





Code available on

https://github.com/sueiras/handwriting_recognition_seq2seq

Model building

- Convolutional part function
  - Lenet without the final softmax layer.
  - Input: list of images
  - Output: list of features fo each input image.
- Encoder: Bidirectional LSTM of one layer
- Decoder: LSTM wrapped by dropout and by the TensorFlow seq2seq.attention_decoder function
  - The initial state are the final backward state of the encoder
  - The attention inputs are the sequence of outputs of the encoder.
  - Loop function to use the previous char estimation as input of the next step in the decoder RNN network
- Loss: TensorFlow seq2seq.sequence_loss function
  - Example of usage of one learning rate in the encoder and other in the rest of the network and access to the gradients.

Decoder seq2seq function

```
tf.contrib.legacy_seq2seq.attention_decoder(
  input_decoder, enc_state_b,
  attention_states, cell_dec,
  num_heads=1, loop_function=loop_f)
```

Loop function

```
def loop_function(prev, _):
  relu_prev = tf.nn.relu(tf.matmul(prev,
      W_decoder) + b_decoder
  return tf.nn.softmax(relu_prev)
```

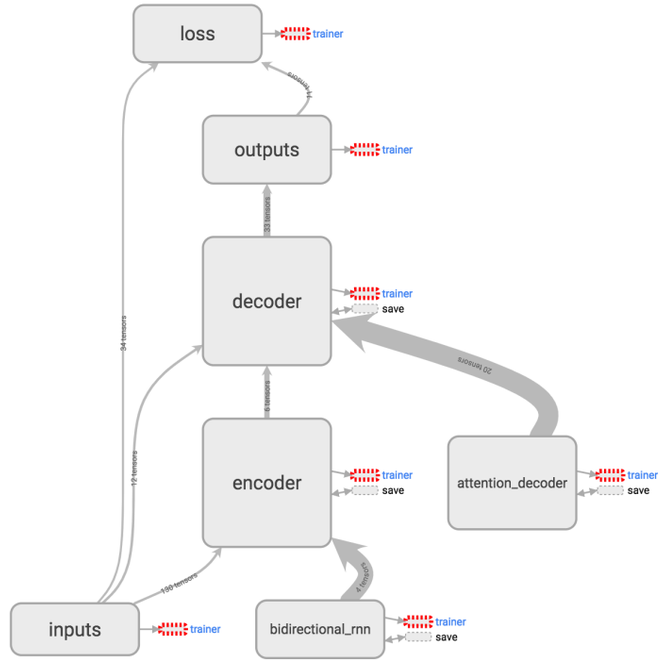
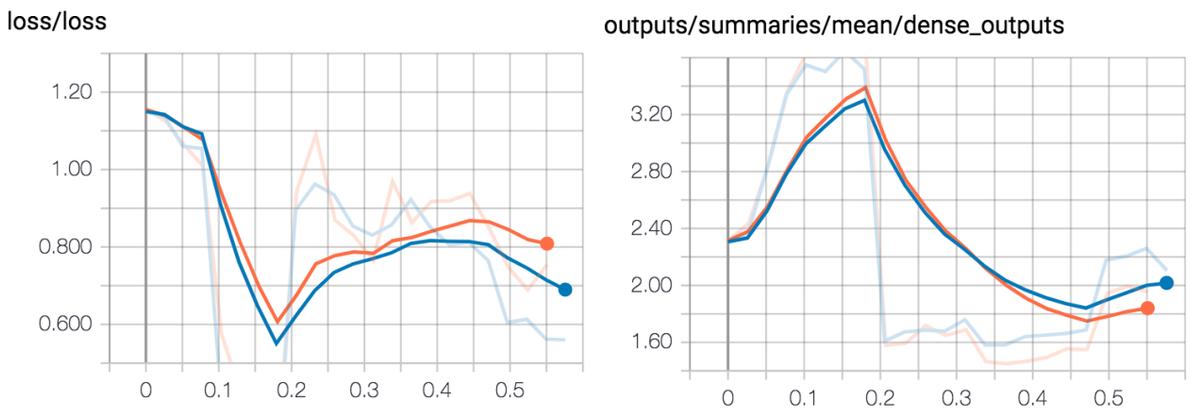
Loss seq2seq function

```
loss = tf.contrib.legacy_seq2seq.sequence_loss(
  dense_outputs, targets, weights)
```

Train model

- Train epoch
  - Generate batches of one epoch of train
  - Execute the train_op and evaluate train loss.
  - Generate batches for validation data (validation MNIST characters)
  - Evaluate loss and plot some examples
- Training strategy
  - learning rate: initial and evolution strategy across the epochs.
  - Curriculum learning
    - increment complexity of the data: In this case the maximum size of the sequences
    - decrement learning rate simultaneously

Tensorboard model structure



Tensorboard training evolution