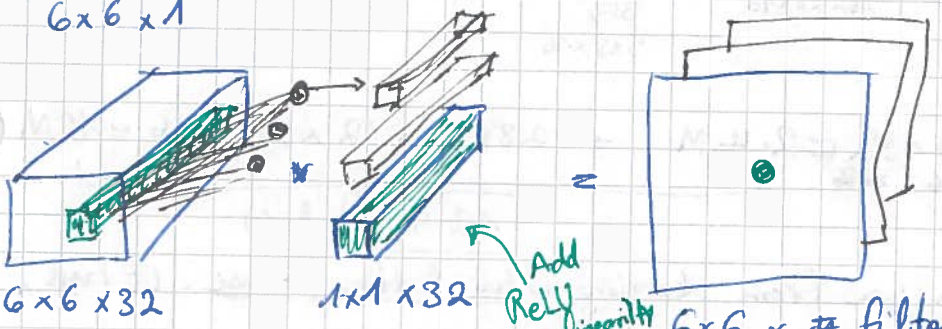
- What if $z^{[l+2]}$ and $a^{[l]}$ does not have the same dimension?
  - First, many convolutions in ResNets are "same conv" (preserves dimension)
  - if dimensions are different, you can add an extra matrix W_s to change the dimension of $a^{[l]}$ → (W_s can be learned, or be fixed)

→ See the paper (2015) by He et al.

Networks in Networks : 1x1 convolutions.

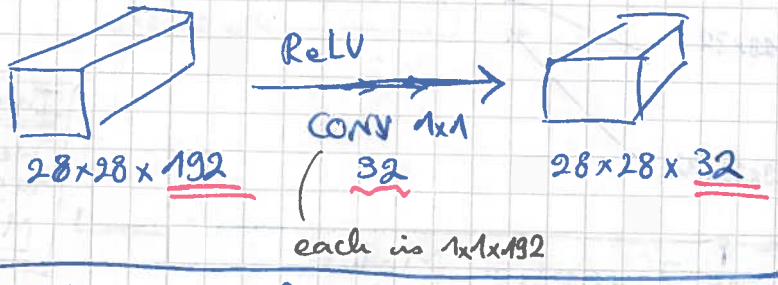
- Adds non-linearity
- Helps decrease # channels if needed.

- nothing interesting for a (6x6x1) one channel image



- now it becomes interesting
- we can now think of it as single neuron taking as input 32 numbers in different channels, and add a ReLU non-linearity to it
- It becomes even more interesting if you have more filters.

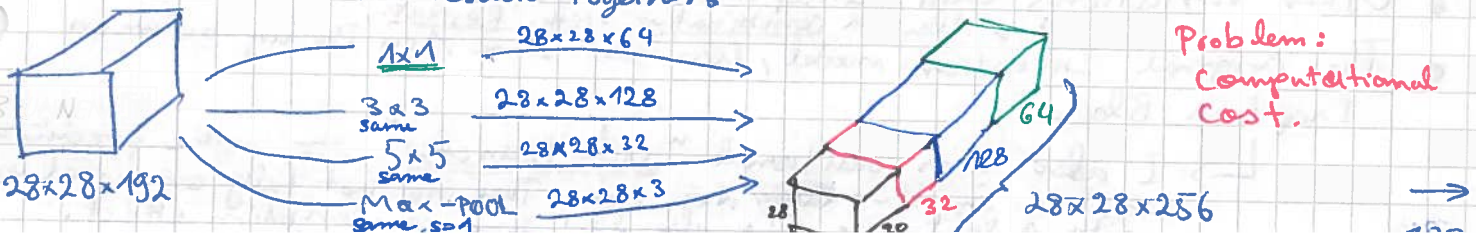
- Called "Network in Network"
- Can carry out a pretty non-trivial computation on the input volume.
- It is like a fully connected layer / network at a specific position (h, w) of your input volume.
- Is basis for many interesting innovations including the Inception.
- It can also be used to "shrink number of channels"



Networks In Network

Motivation for Inception: (2014)

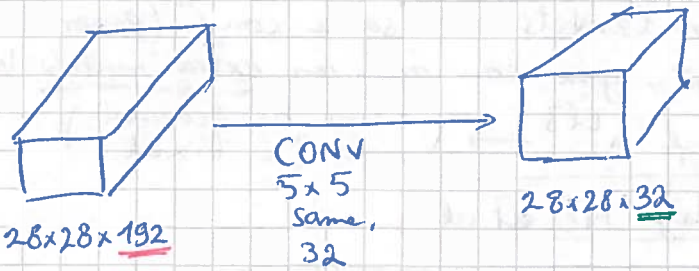
Inception layer: Applay all the possible different filters in one layer and stack up the results together:



Problem: Computational cost.

→

The problem of computational cost: Lets focus on the 5×5 CONV:

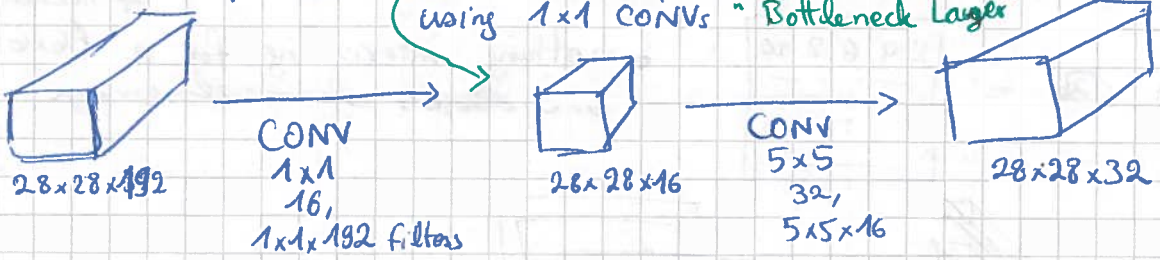


- 32 filters of size 5×5×192
- out put is 28×28×32 numbers for each we need to do 5×5×192 multiplications:

⇒ total multiplications:

$28 \times 28 \times 32 \times 5 \times 5 \times 192 = 120\,M$! :(

So, let's try this: Shrink the representation first using 1×1 CONVs "Bottleneck Layer"



total Multiplikations:

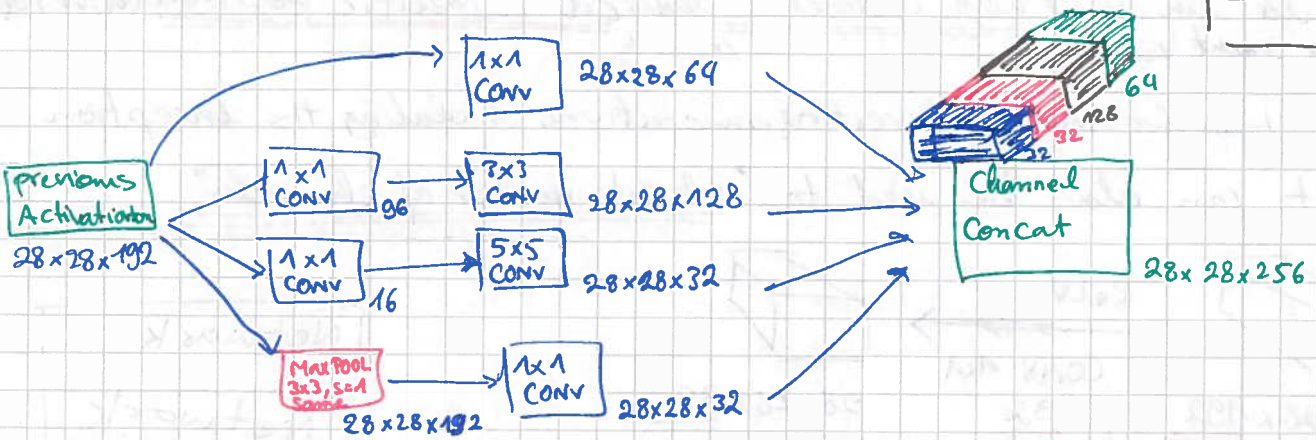
$28 \times 28 \times 192 \times 16 \approx 2.4\,M + 28 \times 28 \times 32 \times 5 \times 5 \times 16 \approx 10\,M$

12.4 M ! :)

- Inception module frees you from deciding what filter to use. It says, lets do them all.
- To reduce the comp. cost, use the "Bottleneck layer" to shrink down the represen
- Bottleneck layers do not hurt the performance as long as used reasonably.

Inception module:

Inception



Inception
"We need to go deep
Meme

One Inception Module

- An Inception Network is stacking up multiple of these modules together. (see Emerge in Video) -O-O-O-O-O→
- Other variations exist, Inception V2, V3, ... → Also a combination with ResNet.
- The original Inception model, has also some Max Pooling between Inception Blocks.
  - → It also has additional "side branches" to make predictions earlier in the Network. They are just fully connected layers with a softmax. It also has ... regularization effect.

GoogLeNet

124

Practical Advices for using ConvNets

Start with: Using Open-source Implementations: find Author's implementations to get going faster.

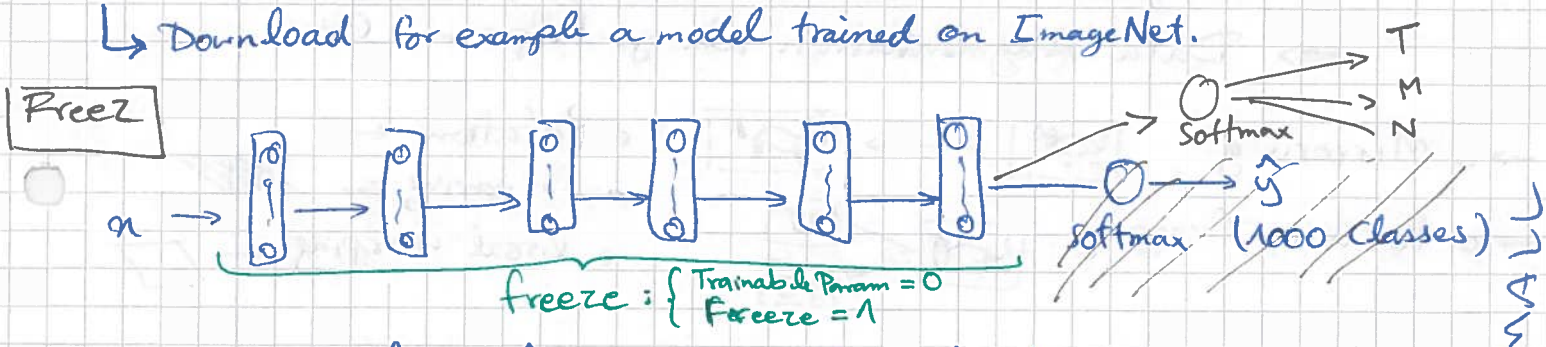
↳ Pre trained networks.

↳ Use transfer learning.

Transfer Learning

Transfer Learning:

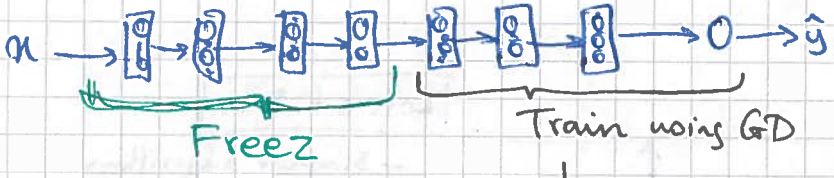
- Download pre-trained weights.
  - ↳ Data sets: Imagnet, COCO, Pascal VOC
- Imagine you want to train a classifier for your two cats (Tiger, Misty) and neither. But you don't have a large dataset of their images:
  - ↳ Download for example a model trained on ImageNet.



- remove the last softmax unit in original network and train your own softmax unit with 3 outputs.
  - ↳ freeze the weights in previous layers so that they don't get retrained.
    - ↳ All DL Frameworks support this.
- Another Option: Consider previous layers a frozen fixed function that takes an input x and maps it to some activation
  - ↳ Save to disk that activation as a feature vector
    - ↳ Train a new shallow softmax model to make the prediction.

Fixed Function

- If you have a larger training set
  - ↳ Freeze fewer layers, and train the rest.



↳ or even remove them and add your own hidden units.

Medium

The more data you have, the more later layers you can retrain

- If you have a lot of data use the pre trained weights as initialization and retrain the whole network.

Image

- Because dataset on the internet are so big, and existing networks have spend weeks on training, Almost Always.

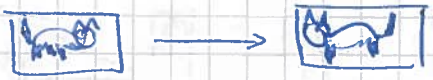
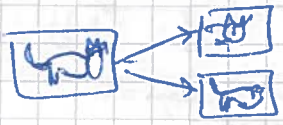
> Better off, when you start with a pre-trained Model!

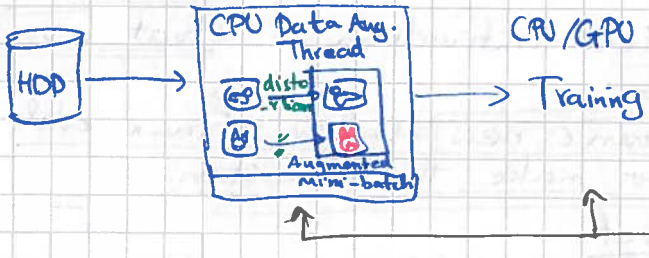
- Transfer Learning, always the first option, unless you have a exceptionally very very large dataset and huge computational budget.

Data Augmentation:

- CV involves learning pretty descent complicate functions. Image (Pixels) $\xrightarrow{f}$ What is in Image
  - In order to learn these very complicated functions, we ~~can have~~ always benefit from more data. We can't have enough data.

$\Rightarrow$ Data Augmentation always helps in CV

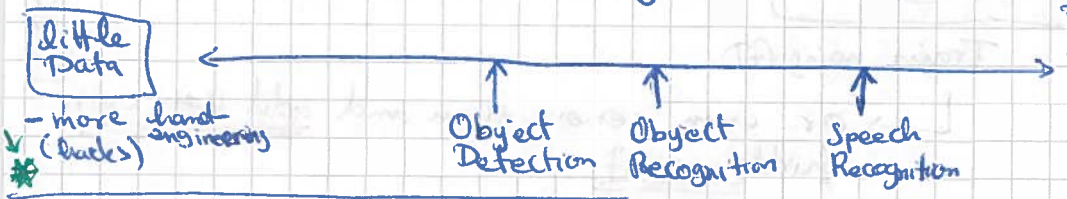
- Mirroring 
- Random Cropping 
- Rotation
- Shearing 
- local warping 
- Color shifting: R+20, G-20, B+20 (Distort RGB channels)
  - In practice the values for R, G, B are drawn from a probability dist.
  - Makes the learning more Robust to changing of color in images
  - PCA color augmentation: Change the dominant color channels more.
    - AlexNet paper.
- Online Data Augmentation: Do data augmentation right at the training time.



> Data Augmentation

State of Computer Vision: Some special aspects of DL in CV

- Data vs. Hand-engineering:



> Two Sources of Knowledge
>
> - labeled data: (x, y) pairs
> - Hand engineered features / network architecture / other components.

> Transfer learning helps a lot when you have less data

Tips for doing well on benchmarks / winning competitions:

- doing well on a benchmark normally increases the chance of the paper getting accepted.
- But the tricks are most of the time not very useful for a real system in production environment.

- **Ensembling:** Train [Needs much memory] several networks independently and average their output. → $\hat{y}$
  - 3–15 networks are typical. Gains of up to 2%.
- **Multi-crop at test time:** Run classifier on multiple versions of the test images and average the results. (e.g. 10-crop)
  - 4 corners, 1 center, flip = 10 crops

Course 4 – CNNs – Week 2 – Programming 1 – Keras

- Keras: high level NN API capable of running on top of several lower level framework like TF and CNTK

```
def model(input_shape):
    X_input = Input(input_shape)          # input placeholdr.
    X = ZeroPadding2D((3,3))(X_input)

    # CONV -> BN -> ReLU Block applied to X
    X = Conv2D(32, (7,7), strides=(1,1), name='conv0')(X)
    X = BatchNormalization(axis=3, name='bn0')(X)
    X = Activation('relu')(X)

    # MAXPOOL
    X = MaxPooling2D((2,2), name='max_pool')(X)

    # Flatten X + Fully connected
    X = Flatten()(X)
    X = Dense(1, activation='sigmoid', name='fc')(X)

    # Creat Model instance to train/test
    model = Model(inputs=X_input, outputs=X, name='HappyModel')
    return model
```

Very Cool :)

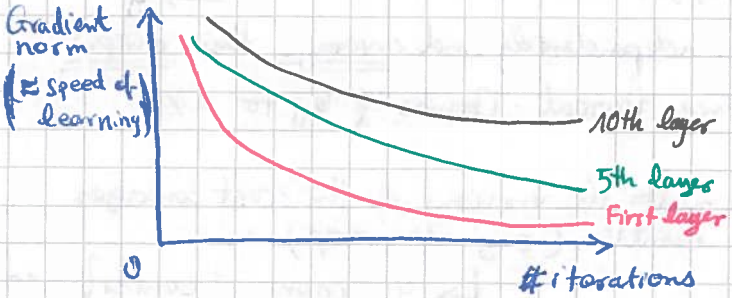
---

1. Create Model
2. Compile model: `model.compile(optimizer="...", loss="...", metrics=["accuracy"])`
3. Train the model: `model.fit(x=..., y=..., epochs=..., batch_size=...)`
4. Test the model: `model.evaluate(x=..., y=...)`

`model.summary()`, `model.plot()`

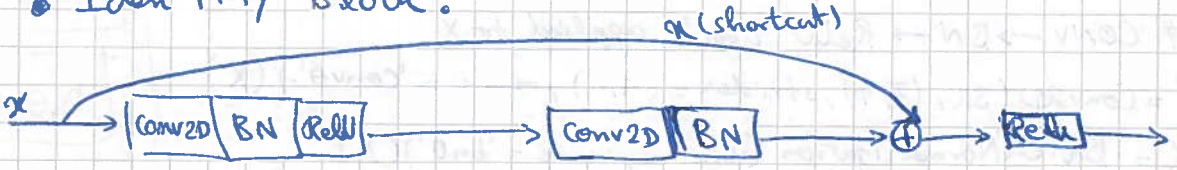
Course 4 - CNNs - Week 2 - Programming 2 - ResNets

- very deep networks are hard to train due to vanishing or exploding Gradients.
  - ↳ During training we might actually see the magnitude (or norm) of the gradients for earlier layers decreases to zero very rapidly as the training proceeds.

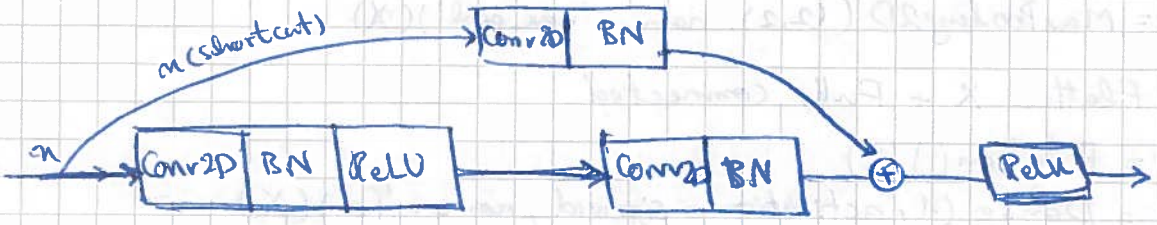


Speed of learning decreases very rapidly for the earlier layers as the network trains.

- The ResNet "shortcut" or "skip connection" allows the gradient to be directly backpropagated to earlier layers.
- Because of the skip connection they can also very easily learn the Identity function.
- Depending on wether input/output dimensions are the same, two main types of blocks are used in a ResNet:
  - Identity Block:



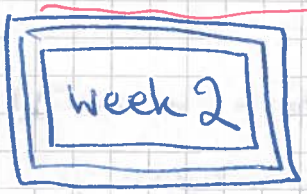
  - Convolutional Block: when input and output dimensions are different



the conv2d block in shortcut path resizes the input a to a different dimension.

Course 5 - RNNs - Week 2: Natural language processing and Word embeddings

1 h 41 m video
2 ✱ programming



- NLP with DL: An important combination.
- Word vector representations, Embedding layers
- Example Applications: sentiment analysis, named entity recognition and machine translation.

→

Word Representations:

- DL has revolutionalized NLP
- Word embedding: A way of representing words that lets your Algorithm automatically learn analogies.

So far: We had a dictionary (vocabulary) with one-hot word Representations:

$V = [a, aaron, \ldots, zulu, \langle UNK \rangle]$ $|V| = 10{,}000$

Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
O_{5391}	O_{9853}				

I want a glass of orang juice.

I want a glass of apple juice.

- Problems with dictionary and one-hot representation:
  - no generalization based on the concepts. for example, the relationship between orange and apple is no different from relationship between orang and king
  - Because the inner product of any two different one-hot vectors is 0, and euclidean distance between any pairs of these vectors is also the same

! ⇒ This representation does not allow the Algorithm to know that somehow "apple and orange" are more similar than "King and orang"

⇒ Featurized Representation: word embedding

	Man (5391)	Woman (9853)	King (4914)	Queen (7175)	Apple (456)	Orang (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Size						
Cost						
alive						
Verb						
⋮						

e.g. 300 features

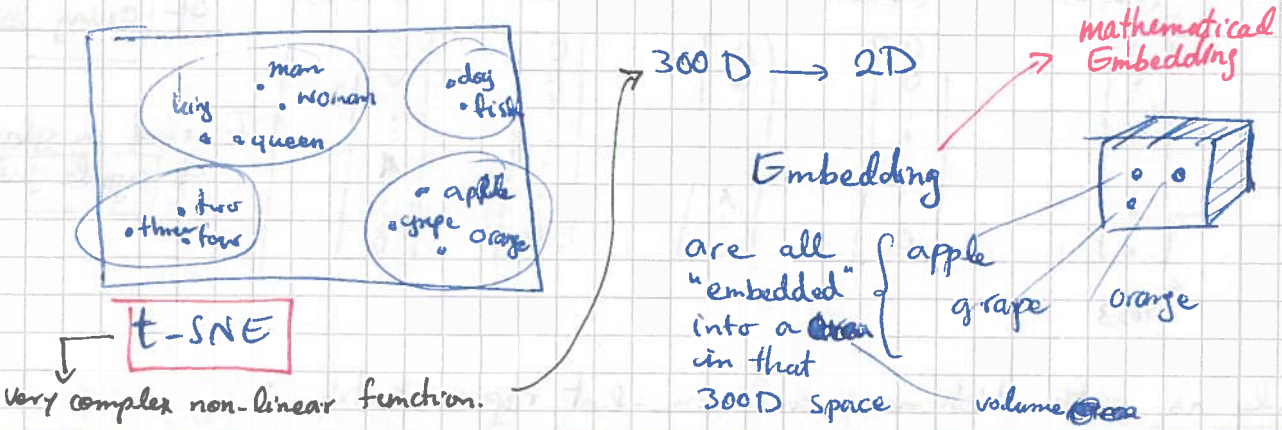
e_{5391} e_{9853}

Now: becomes easy:

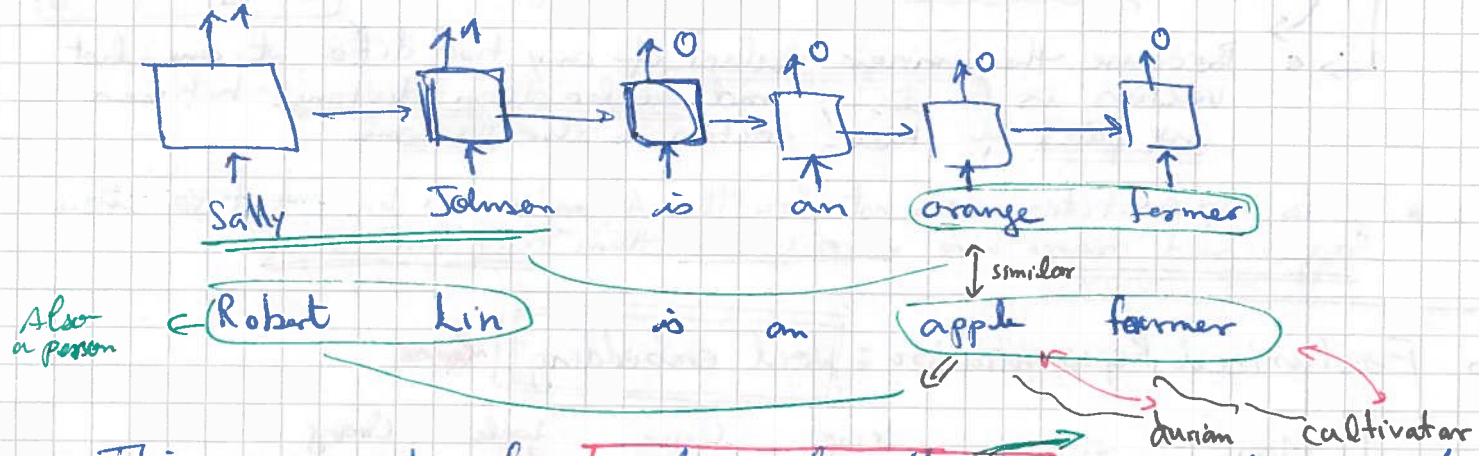
- I want a glass of orange juice
- I want a glass of apple juice.

- We want to learn word embeddings; That is, learn high dimensional feature vectors that give a better Representation than one-hot representation
- Of course the features of a WE won't be as clear interpretation like "gender", "age", "royal". What they exactly represent might be difficult to interpret, but still they capture very well the similarities like between apple and Orange.

Visualizing Word Embeddings:



Using WEs: Example: Named Entity Recognition



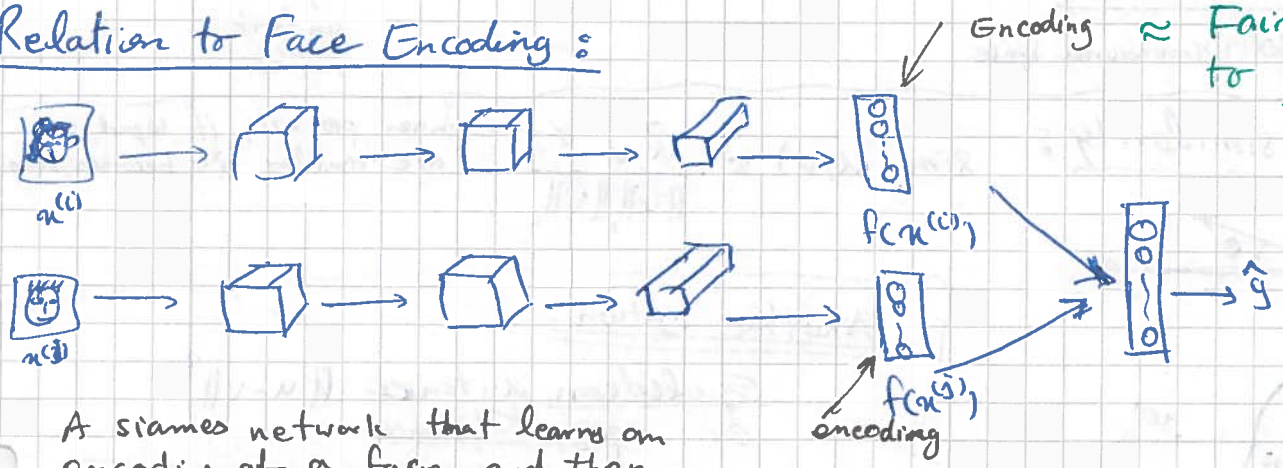
- This even works for words we haven't seen in our training set as long as we have WEs that capture the similarities of those words with words we already know.
- Transfer learning for WEs learned over very large datasets (10^8 ~ 10^{11} words) into task over much smaller Datasets (100k words)

1. Learn word embeddings from large text corpus (1-100 B words) billion (10^9) (Or download pre-trained embeddings online)
2. Transfer embeddings to new tasks with smaller training set (say, 100K words)
3. Optional: Continue to fine tune (adjust) the word embeddings with new data. → normally only if very large enough data allows it to be

⇒ Word Embeddings make a big difference when the task you are trying to carry out, has a relatively small training set.

- Named Entity Recognition
- Text summarization
- Coll reference resolution
- Parsing

Relation to Face Encoding:



A siamese network that learns an encoding of a face, and then compares the two encodings to see if they belong to the same person.

- Difference between face encoding and word embedding:
  - In face encoding we want to be able to compute the encoding for any new image, even those we have not seen before,
  - In WE we have a fixed vocabulary of say 10K and we learn only for each of them a feature vector representing it.

Properties of WEs:

Mikolov et al., 2013 "Linguistic similarities in continuous space word representations"

- WEs help with analogy reasoning

	Man	Woman	King	Queen	Apple	Orange
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97
	e_{man}	e_{woman}	e_{king}	e_{queen}		

Question (Analogy Reasoning):

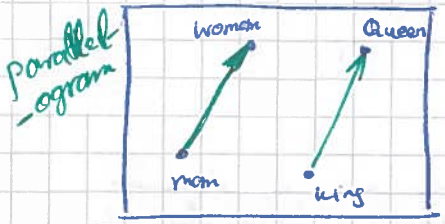
Man $\xrightarrow{\text{to}}$ Woman is like King $\xrightarrow{\text{to}}$? (what?)

$$e_{man} - e_{woman} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad e_{king} - e_{queen} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Rightarrow \boxed{e_{man} - e_{woman} \approx e_{king} - e_{queen}} \Rightarrow \text{queen}$$

difference of man and woman is like difference of king and queen.

← Analogies using Word Vector s



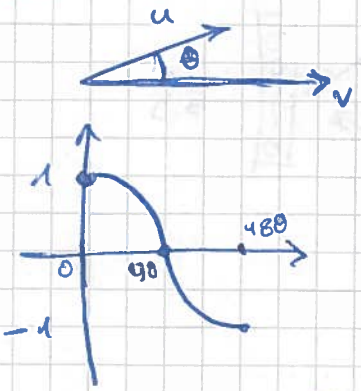
$e_{man} - e_{woman} \approx e_{king} - e_?$

→ Find word w: $\arg\max_w \text{sim}(e_w, e_{king} - e_{man} + e_{woman})$

similarity function

Cosine similarity:

$$\text{sim}(u,v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$ ← inner product, if u and v are similar $u^T v$ becomes very large



Another option:

Euclidean distance $\|u-v\|^2$ or squared distance

Cosine more often, because it normalizes better on length of $\vec{u}$ and $\vec{v}$

- Generality of analogy relationships that WEs can learn is remarkable!

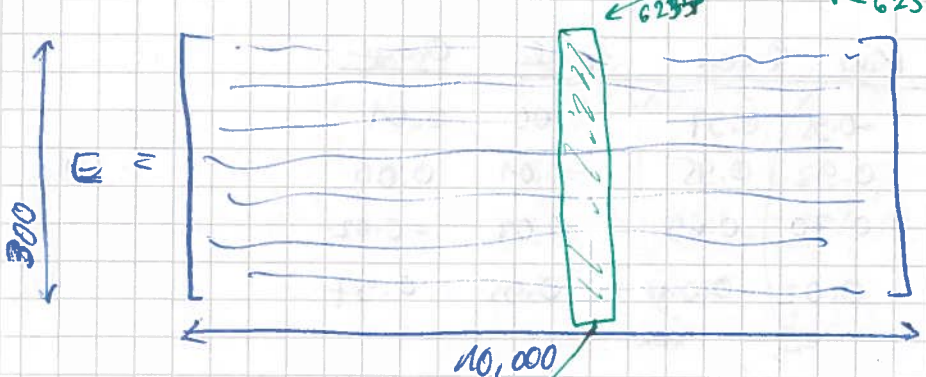
Man : Woman as Boy : Girl	Ottawa : Canada as Nairobi : Kenya
Big : Bigger as Tall : Taller	Yen : Japan as Ruble : Russia.

Embedding Matrix:

our goal is to learn the embedding matrix E

Vocabulary = [a aaron ..... orange ..... zulu <unk>] ; |V| = 10,000

6257



- E is the (300, 10,000) dim embedding matrix.
- Each column of E is the embedding for the corresponding word from the dictionary

(300, 10k) (10k, 1) (300, 1)

$E \cdot O_{6257} = e_{6257}$ ⇝ $E \cdot O_j = e_j$

one-hot representation of Orange

Embedding matrix | One-hot represent. of word j | Embedding of word j

O_{6257} (10,000, 1) = [0 0 0 ... 1 ... 0] ← 6257

Learning Word Embeddings: Neural Language Model (Example)

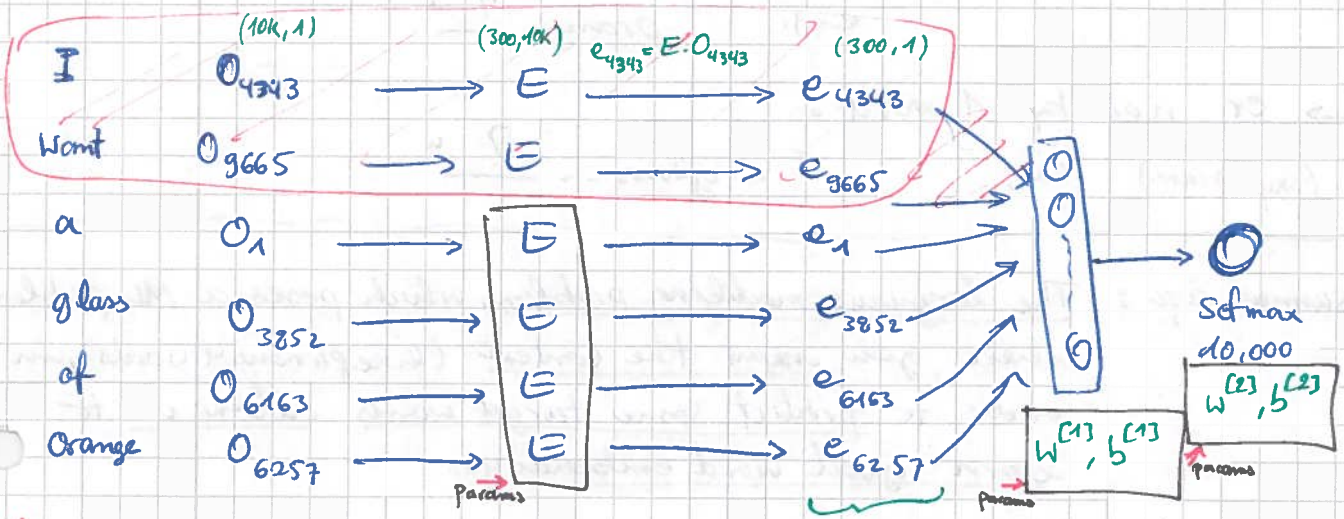
Bengio et al. 2003 "A neural probabilistic language model"

fixed historical window

I	want	a	glass	of	orange	____ .
4343	9665	1	3852	6163	6257	

We want to predict this.

- Building a Neural Language Model is a reasonable way to learn a set of embeddings



6 x 300 = ~~1800~~ 1200 inputs

e.g. 4

- It is also very commonly done to have a fixed historical window
  - => you can deal with different length sentences, because input sizes are always fixed

- Params of this model are: Matrix E (always the same for all the words)
  - $W^{[1]}, b^{[1]}$
  - $W^{[2]}, b^{[2]}$
  - Now we can use Backprop and GD to maximize the Likelihood of the training set → try to predict what is the next word given previous words in the sequence.

! - Interestingly this Algorithm will learn pretty decent word embeddings
  - Because it is in the algorithms incentive to learn pretty similar embeddings for say "orange" and "apple" for the example sentence above.

Other Context/target pairs:

> cereal

I want a glass of orange juice to go along with my

context → target

Context: 4 words before

- We can experiment with the context, given the task we have. →

← for example, if the task is to learn a language model, the context could be "the last 4 words"

↳ But the context could be: "4 words on the left and right"
eg. "a glass of orange ___ to go along with."

↳ or a simpler context: "last 1 word"
e.g. "orang ___"

↳ or nearby 1 word:
(skip gram) e.g "...glass... ___"

---

to summarize: The language modeling problem, which poses a ML problem where you input the context (like previous 4 words), in order to predict some target words allow us to learn good word embeddings.

Word2Vec: Compared to a language model, the research has shown that even simpler context → target mappings, e.g. only one word will also result in very good WE results.

Skip Gram:

I want a glass of orange juice to go along with my cereal.

Mikolov et. al. '13 "Efficient estimation of word representation in vector space"

Context	target
orange	juice
orange	glass
orange	my

- Creat a set of randomly picke context-target pairs
- select the context word first randomly
- then randomly pick another word within a given window.

- We set up a supervised learning problem that given the context word, what could be a randomly chosen word (predict) within a say ±10 word window.

↳ Not an easy problem; because the set of possible words in the vocabulary is large to choose from. But the purpose of this problem is infact to learn the WEs.

Vocab size = 10k → can even be much larger ≥1M

Learn a mapping:

context c ("orage") 6257 → to → target t ("juice") 4834

$O_c \rightarrow E \rightarrow e_c \rightarrow \text{Softmax} \rightarrow \hat{y}$

$e_c = E \cdot O_c$

Softmax: $p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$

θ_t: parameter associated with output t

Param

Params

← . The loss function for the Softmax will be the usual negative log likelihood:

$$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{10000} y_i \log \hat{y}_i$$ ⇝ y as one-hot vector $\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$ ← 4834

Parameters:

- Matrix E: all the embedding vectors e_c
- θ_t parameters of the softmax.

→ $\hat{y}$ will be a 10,000 dim vector output from softmax unit, with probabilities for each of the words in in the vocab being the target.

← This model is called **Skip Gram** -model. It can learn pretty descent embeddings

- But it has some problemos:
  - Computational speed

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10k} e^{\theta_j^T e_c}}$$

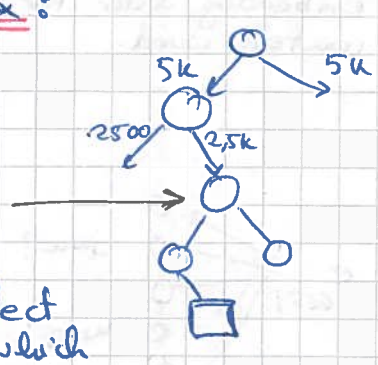
very slow :(

Sum over the whole vocab.

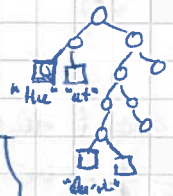
⇒ use a Hierachical Softmax:

- each interior node is a binary classifier

⇒ cost will be order $\log|V|$ rather them $|V|$



- in practice, don't use a perfect balanced tree, but a tree in which common words like "the, at, ..." are more at top



How to sample the context c?

↳ • Uniformly? → there are very frequent words "the, of, a, and, to, ..."

⇒ don't sample uniformly; use some heuristics that let you sample also less frequent words fairly.

Negative Sampling: Skip Gram can find very useful embeddings, but the down side of it is the computational cost. ↳ too slow.

"I want a glass of orang juice to go along with my cereal."

Question: Given a pair of words, is it a context-target pair?

Context	word	Target?	
orange	juice	1	← pick target like before "randomly within a given window."
orange	king	0	← pick the non-targets simply randomly from the dictionary.
orange	book	0	

Mikolov et al. '13 "Distributed representation of words and their compositionality"

to summarize How to generate the dataset:
- Pick a context word, and a target within a window. That gives us the "positive example" → label 1
- Then for k times, pick random words from the dictionary to create "negative examples" → label 0

The supervised Learning problem: Input: the word pair → Output: 1 or 0

- How to choose k? k = 5-20 for smaller DSets. k = 2-5 for larger DSets.

Model:

estimate the probability that y = 1

$$P(y=1 \mid c,t) = \sigma(\theta_t^T e_c)$$

- Similar parameters as softmax before:
  - θ_t one param vector for each possible target word
  - e_c the embedding vector for each possible context word.

X		Y
context	word	target?
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0
c	t	y

(k: the 4 rows between the first and last pair)

- K to 1: the ratio of negative to positive examples.

input word: orange (6257)

$O_{6257} \rightarrow E \rightarrow e_{6257}$ → juice? king?

- 10,000 binary logistic regression classifiers
- Instead of training all 10,000 of them, at each iteration we only train k+1 of them.

- Instead of having one giant 10,000 way softmax, which is very expensive to train ($P(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10k} e^{\theta_j^T e_c}}$), the skip gram model), we now have 10,000 binary classification problem, which are really cheap to compute and on every iteration, we are going to train only k+1 of them.
- This technique is called **Negative Sampling** because for each "positive example" we have k "negative examples"
- How to sample the negative examples?

One extreme
- Sample based on w_i frequency of each word

$P \sim f(w_i)$
- you end up learn much higher representation of frequent words: "it, of, the, a..."

In Between
- works best
- empirical

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10k} f(w_j)^{3/4}}$$

Other extreme
- Sample uniformly with probability for each word $\frac{1}{|V|}$

GloVe : (Global Vectors for word Representations)

Pennington et. al. "GloVe: Global vectors for word representation"

I want a glass of orange juice to go along with my cereal.
c, t

X_{ij} = # times i appears in context of j [counted co-appearances within a given window]

i ↑ t j ↑ c

depending on the choice of what is context and what is target, X_{ij} can be symmetric: $X_{ij} = X_{ji}$

Model:

minimize the difference $(\theta_i^T e_j - \log X_{ij})$

$$\sum_{i=1}^{10k}\sum_{j=1}^{10k} f(X_{ij})\left(\underbrace{\theta_i^T e_j}_{t\ c} + b_i + b'_j - \log X_{ij}\right)^2$$

bias: b_i (t), b'_j (c)

$f(X_{ij})$: weighting term

$\theta_i^T e_j$: how related are words t and c,

$\log X_{ij}$: which is measured by how often they appear together.

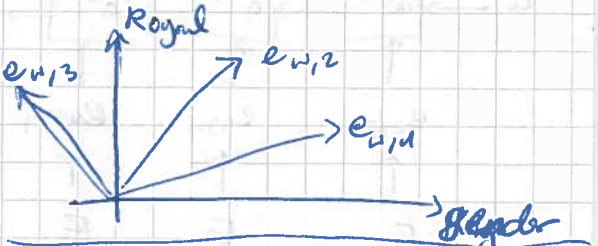
- learn two vectors (θ_i and e_j) whose inner product is a good predictor for how often these two words i, j appear together.
- $f(X_{ij})$, the weighting term, $f(X_{ij}) = 0$ if $X_{ij} = 0$ (because $\log 0$ not defined) and
  → also weighting factor can give more weight to less frequent words, and less weight to more frequent (like stopword) words.
- Now let θ's & e's be optimized using GD
  → both initialized randomly
- Interestingly, looking at math the role of θ_i and e_j are now pretty symmetric.

$$\Rightarrow e_w^{(final)} = \frac{e_w + \theta_w}{2}$$

We can calculate the final embedding as the average over learned θ and e

A note on featurization view of word embeddings:

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)
Gender	-1	1	0.95	0.97
Royal	0.01	0.02	0.93	0.95
Age	0.03	0.02	0.70	0.69
Food	0.09	0.01	0.02	0.01
⋮	⋮	⋮	⋮	⋮



- the components of the WE vector are not interpretable.
- They might not even be orthogonal to each other.
- The parallelogram math still works.

$$\text{minimize} \sum_{i=1}^{10k}\sum_{j=1}^{10k} f(X_{ij})\left(\theta_i^T e_j + b_i - b'_j - \log X_{ij}\right)^2$$

$\theta_i^T e_j$:

$$(A\theta_i)^T(A^{-T}e_j) = \theta_i^T A^T A^{-T} e_j = \theta_i^T e_j$$

Sentiment Classification:

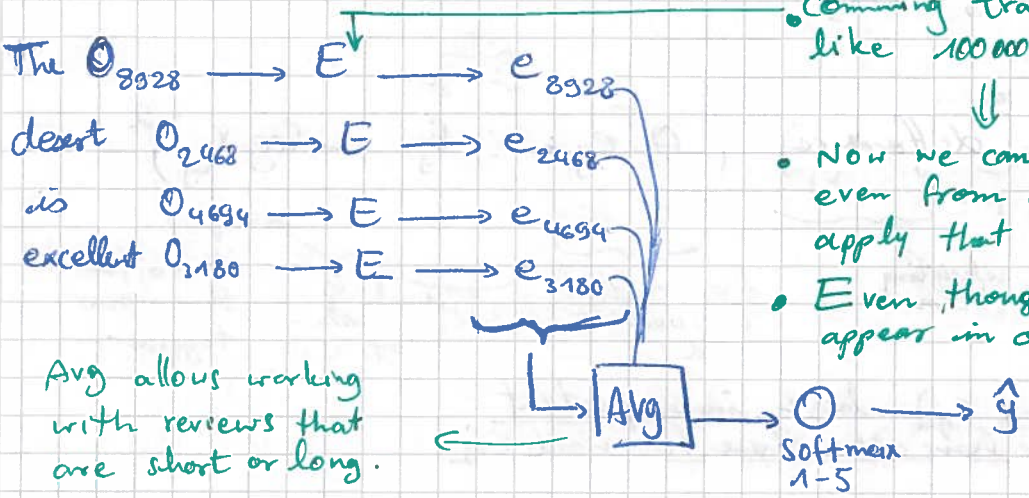
x → y ★★★★★

x: Comment text
y: rating stars

- One of challenges of Sentiment Analysis is you may not have a huge labled Data set. e.g. (10k – 100k words)

"The desert is excellent" ★★★★☆ (4 stars)

8928 2468 4694 3180



Avg allows working with reviews that are short or long.

- comming trained on a very large data set like 100 000M words. 10^9 words
- Now we can take a lot of knowledge even from infrequent words and apply that to our problem.
- Even though those word do not even appear in our training set.

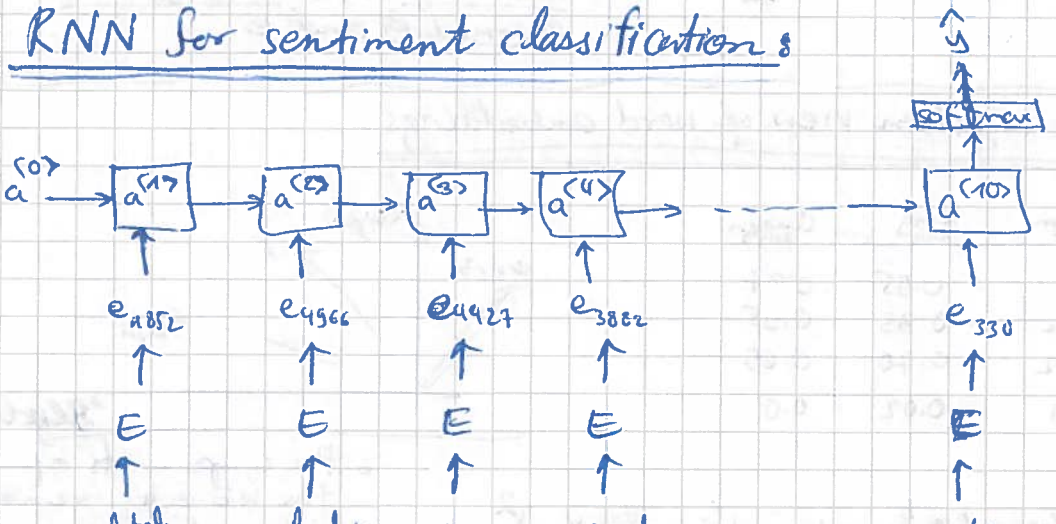
- This Algorithm works very well. It sums (averages) the meanings in our words.
- A problem is that, it ignores word order, e.g.

"Completely lacking in good taste, good service, and good ambience." :(One star

But by simply averaging the words, our algorithm would think this is a very "good" review.

⟹ Instead of just summing over all of the WEs, we can instead use a RNN for sentiment classification.

RNN for sentiment classification:



actually the one-hot representation of the words.

(Many-to-one RNN)

Debiasing word Embeddings:

Bolukbasi et al. '16
"Man is to computer as woman is to homemaker? Debiasing word embeddings."

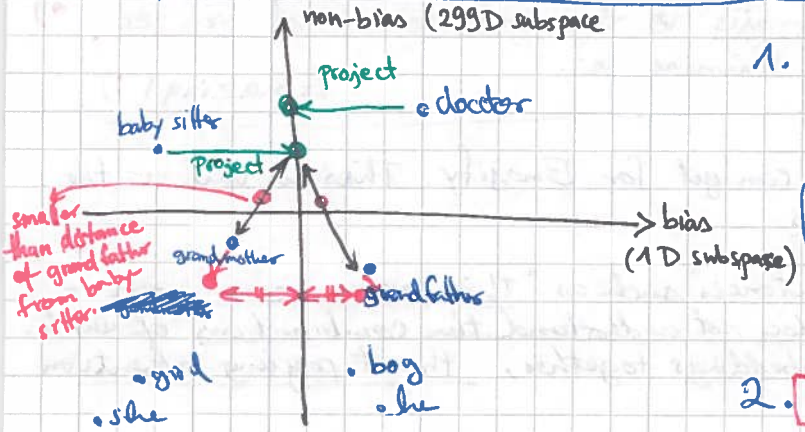
Man : Woman as King : Queen
Man : Computer_programer as Woman : Homemaker ✗ ! Gender bias!
Father : Doctor as Mother : Nurse ✗ !

Word embeddings can reflect gender, ethnicity, age, sexual orientation, and other biases of the text used to train the model.

- Learning algorithms are increasingly making very important decisions:
  - College Admissions
  - Jobs ~~assignments~~
  - Loan applications
  - Even sentencing guidelines in the criminal justice systems. ← !

Addressing bias in word Embeddings:

Example here for gender. Same procedure for any ~~bias~~ other bias.



1. **Identify the bias direction:**
   - $e_{he} - e_{she}$
   - $e_{male} - e_{female}$
   - ...
   - → Average ⇝ bias direction

   This gives the non-bias ↑ → bias

   In original paper the bias direction can be multi dim. and instead of Avg, SVD is used to find the bias direction

2. **Neutralize:** for every word that is not definitional (has gender in its definition, such as girl, boy, grandma, grandpa) project to get rid of bias.
   - ↳ eliminate their components that are in bias direction (contribute to creating bias)

3. **Equalize** pairs.

   for pairs like, grandfather – grandmother
   girl – boy
   woman – man
   ...
   that have gender, make sure they have "the same distance" from neutralized word.

For words that have gender in their definition, we want the only difference in their embedding to be the gender.
Make sure that words like grandmother and grandfather has "exactly the same" distance from words that should be gender neutral like babysitter or doctor.

How to choose words that should be Neutralized? Authors of the paper trained a classifier for that.
- most words in English language are gender neutral.
- accordingly the set of pairs we want to neutralize is also relatively small.

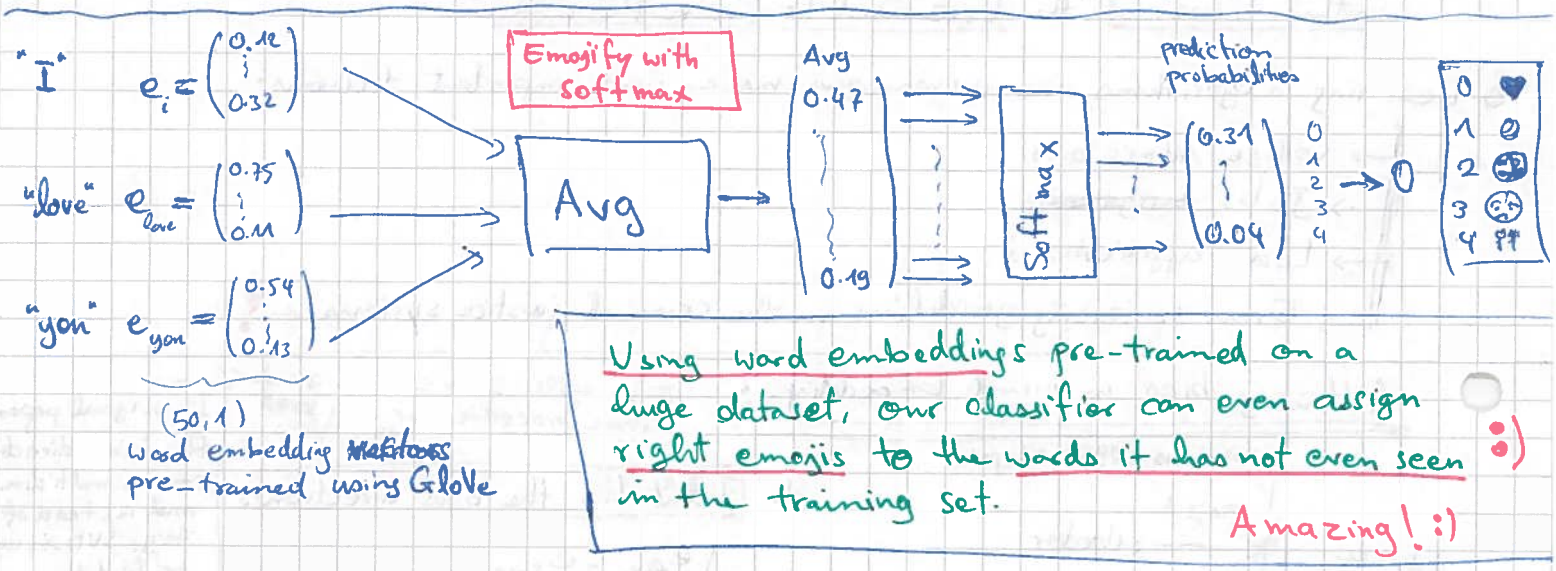
Course 5 - RNNs - Week 2 - Programming 1 - Operations on Word Vectors

- Load pre-trained word vectors and measure similarity using cosine similarity
- Use WEs to solve word analogy problems such as Man to Woman is as King to ?
- Modify WEs to reduce their gender bias

Word Analogy problem | Debiasing:

It is really amazing :) See Notebook

- Using a pre trained set of word vectors from the internet is often a good way to start a NLP application.



Using word embeddings pre-trained on a huge dataset, our classifier can even assign right emojis to the words it has not even seen in the training set. :)

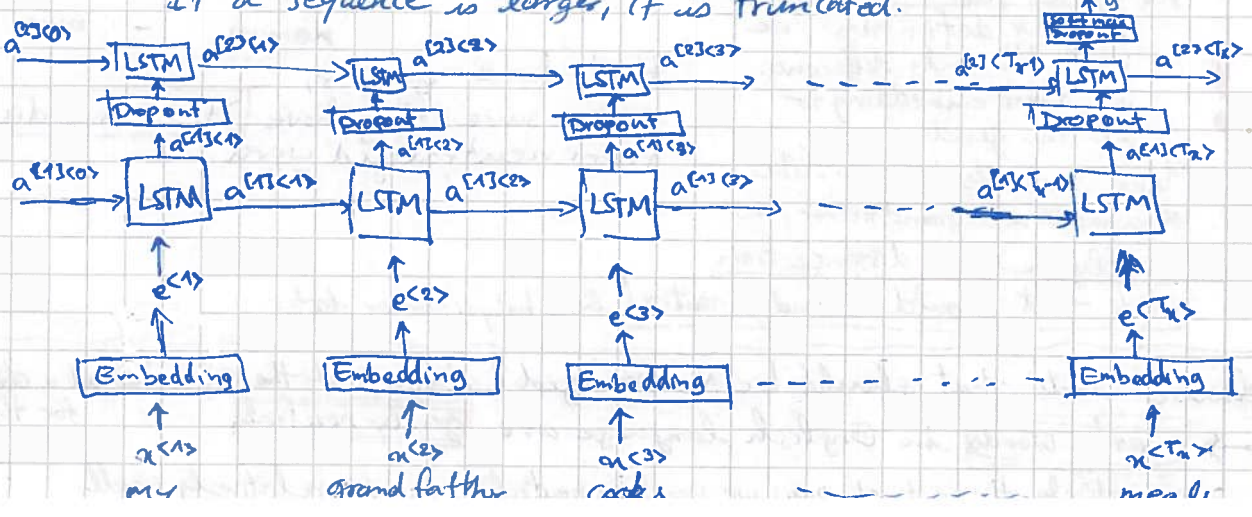
Amazing! :)

- Even with 127 training examples, you can get for Emojify. This is due to the generalization power of word vectors.
- Emojify-V1 performs poorly on sentences such as "This movie is not good - and not enjoyable", because it does not understand the combinations of words. It just averages all the words' embeddings together, without paying attention to the ordering of words.

Emojify with LSTM :

Emojify with LSTM. (Keras)

- using Keras.
- Keras and most DL Frameworks need that all sequences in the same mini-batch have the same length. This allows vectorization to work.
  - Use padding to bring all the sequences to the same length. For example with zeros. We can pad to the length of the largest sequence. If a sequence is larger, it is truncated.



- In Keras, the embedings are represented as a layer that takes integers (word indices) and outputs dense embeding vectors. This layer can be initialized using existing pre-trained embeddings or be trained.

"I love you"
"I will be ready"
word_to_index
(2, max_len) [185457, 226278, 394475, 0, 0]
[185457, 387696, 71090, 302254, 0]
zero padding
Embedding layer
$[e_{185457}, e_{226278}, e_{394475}, \vec{0}, \vec{0}]$
$[e_{185457}, e_{387696}, e_{71090}, e_{302254}, 0]$
(2, max_len, 50)
↳ length of GloVe Emb. Vects

Things to remember

- If you have an NLP task where the training set is small, using word embeddings can help your algorithm significantly.
- Training sequence models in Keras (and most other DL Frameworks) requires few important details:
  - To use mini-batches, all the sequences must be padded, so that all the examples in a mini-batch have the same length.
  - The Embeddings() layer can be initialized with pre-trained values. You can further train those values (if you have a large enough Data set) or leave them fixed
  - LSTM() has a flag called "return_sequences" to decide if you want to return every hidden state or only the last one.
  - We can use Dropout() right after LSTM() to regularize the network.

Course 5 – Sequence Models – Week 3 – Sequence Models and Attention Mechanism

Week 3 Sequence Models and Attention Mechanism

video 1 h 42 m
prog. 2 h 30 m

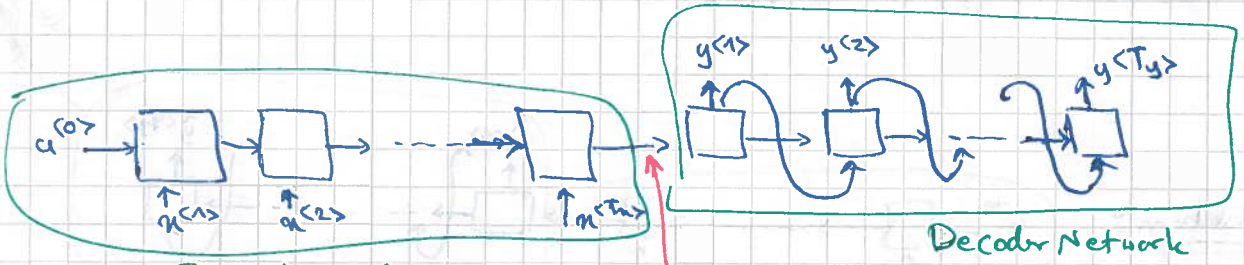
Sequence to Sequence Models : Basic Models

↳ Machine Translation, Speech Recognition, etc.

$x^{<1>}$ Jane $x^{<2>}$ visite $x^{<3>}$ l'Afrique $x^{<4>}$ en $x^{<5>}$ Septembre ← Input sequence

→ Jane is visiting Africa in September. ← Output Sequence
$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$

- Sutskever et al. '14 "Sequence to Sequence learning with neural network"
- Cho et al. '14 "Learning phrase representations using RNN encoder-decoder for statistical machine translation"



Encoder Network

- get the input sequence one word at a time and encodes it
- Can be any kind of RNN like LSTM, GRU.

Input to the decoder Network (encoded french text)

Decoder Network

- Generates the output sequence
- Similar to the way we generated text after learning a language model.

- Given enough pairs of french-english training examples, this model works pretty decently :)

← Image Captioning :

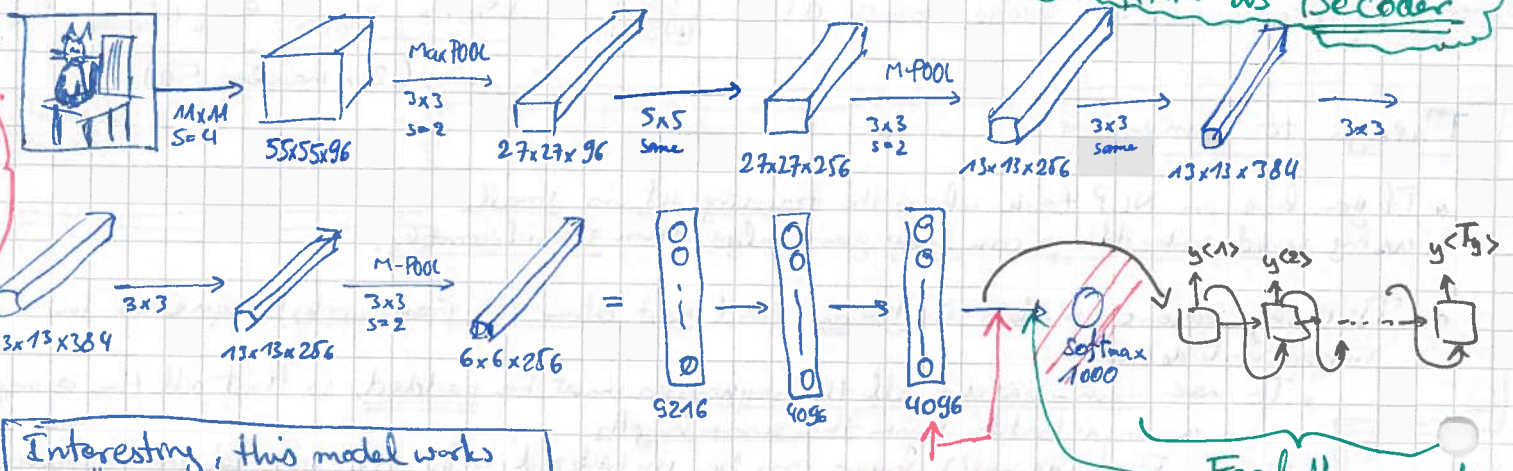
An architecture very similar to the encoder-decoder for machine translation can be used for automatic Image Captioning.

- Mao et al. '14, "Deep captioning with multi-modal recurrent neural Networks"
- Vinyals et al., '14, "Show and tell: Neural Image Captian generator"
- Karpathy and Fei Fei, '15, "Deep visual semantic alignments for generating image descriptions"

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$

A Cat sitting on a chair.

AlexNet as Encoder
RNN as Decoder



4096 D encoding of a "Cat on a chair"

Interesting, this model works pretty well.

- Feed the encoding to an RNN that generates an output sequence of words.

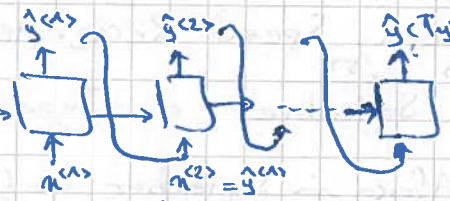
- One important difference of sequence to sequence models and the models we used for synthesizing novel text using a language model is:

→ "Here you don't want a randomly chosen translation. Here you want the most likely translation"; We don't want a randomly chosen caption, but rather the best and most likely.

Picking the Most Likely Sentence:

- We can think of machine translation as "conditional language model"

Language Model:

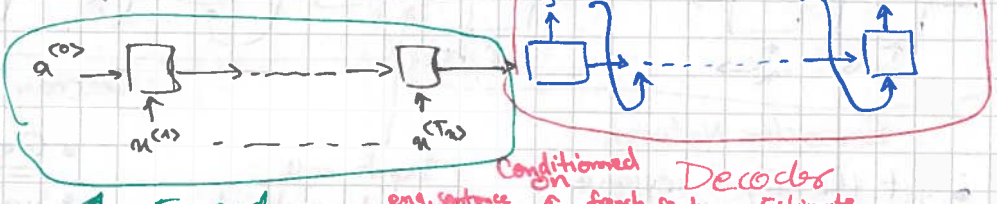


$P(\hat{y}^{<1>}, \ldots, y^{<T_y>})$

- A language model allows us to estimate the probability of a sentence
- We could also use it to generate new ~~sentences~~ novel sentences.

identical

Machine Translation:



Encoder

Conditioned Decoder

$P(y^{<1>}, \ldots, y^{<T_y>} \mid x^{<1>}, \ldots, x^{<T_x>})$

eng. sentence | french sent.

Estimate Probability of the english sentence, conditioned on the input french sentence.

- The decoder of a machine translation model is similar/identical to a language model.

Conditioned Language Model.

- Machine translation model is very similar to the language model, except that instead of starting off with the vector of all

Finding the most likely translation:

- Given the french sentence x, the model should tell us what is the conditional probability $P(\hat{y}^{<1>}, \ldots, \hat{y}^{<T_y>} \mid x)$ of different english translations.
- We don't want to sample words for the english sentence randomly
  Because we could then end up with very stupid traslations, like:
  → "Her african friend welcomed Jane in September."
- What we want is to find an english sentence y, that maximize that conditional probability:

$$\underset{y^{<1>}, \ldots, y^{<T_y>}}{\arg\max} \; P(y^{<1>}, \ldots, y^{<T_y>} \mid x)$$

Why not a greedy Search?

→ • just pick the most likely (output) word at each time-step, according to the conditional language model.

- This does not work. We want the joint probability of all words in the whole sentence to be maximized given the french sentence.
  → Greedy search does not deliver this

Example:

→ Jane is visiting Africa in September. ← Better translation, higher $P(y \mid x)$

→ Jane is going to be visiting Africa in September. ← Less good, lower $P(y \mid x)$

:(• But greedy search would generate the second one! !

→ Because $P(\text{Jane is going} \mid x) > P(\text{Jane is visitting} \mid x)$:(
(only based on maximum probability of the next word)

- So we need something that considers the joint conditional probability of the whole sentence
  → But the space of posible sentences is huge; exponentially large: $|V| = 10{,}000$

Space of posible sentences with 10 words: $10{,}000^{10}$ ↑ impossible to enumerate them all

⇒ We need an approximate search algorithm

→ • Trys to pick the sentence y that maximizes $P(y \mid x)$
  • Although it is not guaranteed to find y.

→

← Beam Search Algorithm:

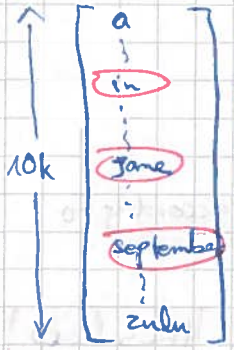
- Given a french sentence as input, we don't want to output a random english sentence. We want to output the best, the most likely sentence.
- Also given an audio signal, we want to output the most likely transcription
- **Beam Search** is a widely algorithm to achieve this.

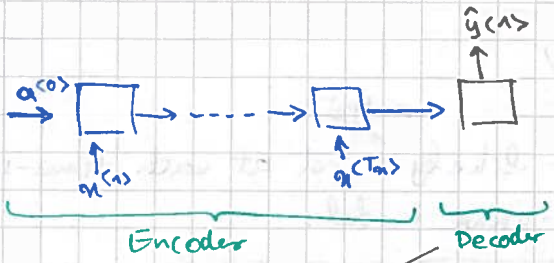
Beam Search: Instead of picking only the one most likely word (Greedy Search) at each step, pick the B (beam width) top most likely words, and build up a tree of search by picking B words at each step.

Beam Search

Step 1

Beam Width, B = 3 ⟹ Pick the three most likely words at first step.





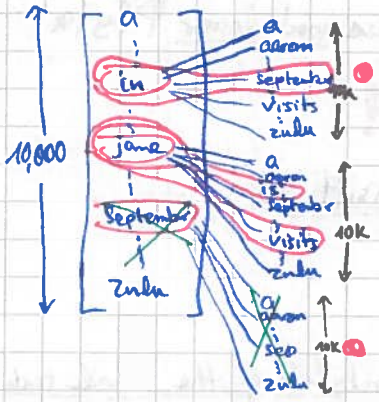
A softmax output with 10,000 possibilities, out of which we pick the three most likely.

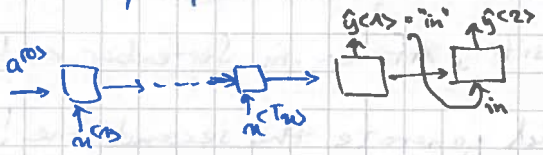
This network calculates → $P(y^{\langle 1\rangle} | x)$

Let's say evaluating $P(y^{\langle 1\rangle} | x)$ outputs the words "in", "jane", and "september" as the most likely three words.

Step 2:

Pick the second most likely words, based on the probability of the whole (2) combination $P(\hat{y}^{\langle 1\rangle}, \hat{y}^{\langle 2\rangle} | x)$





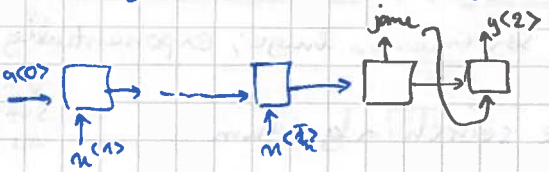
This network evaluates:

$P(\hat{y}^{\langle 2\rangle} | x, \text{"in"})$

but we are interested in

$$P(\hat{y}^{\langle 1\rangle}, \hat{y}^{\langle 2\rangle} | x) = P(y^{\langle 1\rangle} | x) \cdot P(y^{\langle 2\rangle} | x, y^{\langle 1\rangle})$$

We do the same thing for the second choice of the most likely first word i.e. "jane".



This network computes:

$P(y^{\langle 2\rangle} | x, \text{"jane"})$

we want $P(y^{\langle 1\rangle}, y^{\langle 2\rangle} | x) = P(y^{\langle 1\rangle} | x) \cdot P(y^{\langle 2\rangle} | x, y^{\langle 1\rangle})$

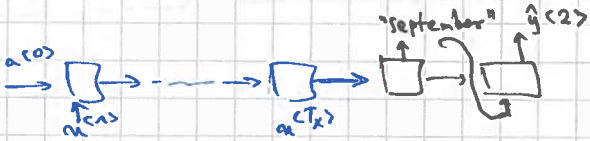
Now we evaluate all the 30,000 possibilities for the combination of the first and second word and pick the most likely top 3 again.

↳ "combinations!"

"in september"
"jane is"
"jane visits"

144

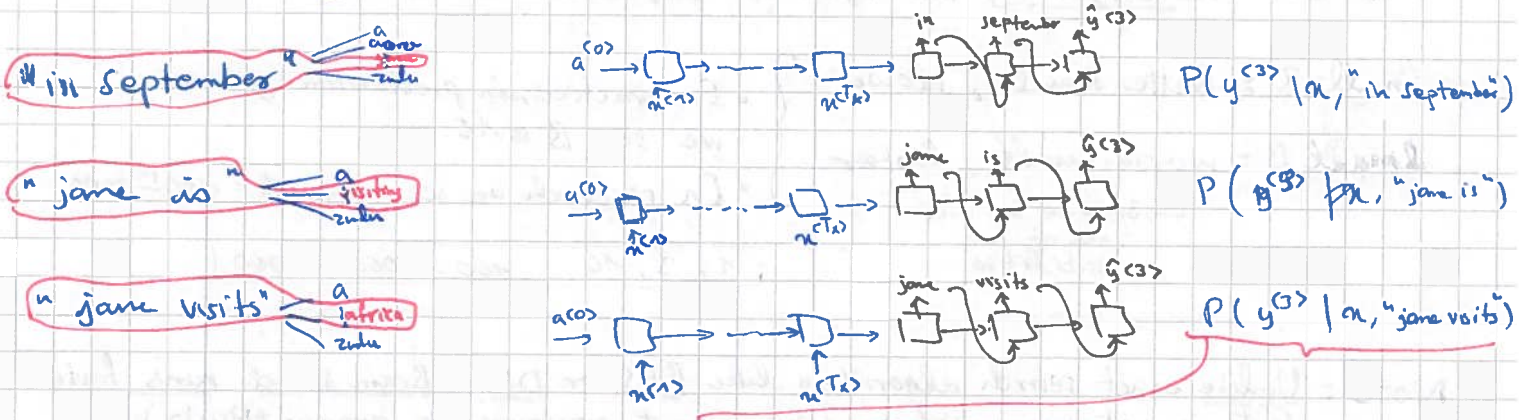
- The same thing for the third most likely first word.



⟹ We can now reject "september" for the choice of the first word. ✗

- At every step we instantiate $B=3$ copies of the network, and evaluate all the 30,000 $P(y^{<1>}, y^{<2>} | x)$ probabilities using them.

Step 3:
- So far we have saved 3 possibilities for the most likely combinations for the first 2 words.



$$P(y^{<1>}, y^{<2>}, y^{<3>} | x) = P(y^{<1>}, y^{<2>} | x) \cdot P(y^{<3>} | x, y^{<1>}, y^{<2>})$$

- **Note:** $B=1$ would be equivalent to "Greedy Search"

Beam Search Refinements:

Length Normalization:

What Beam search does is maximizing:

$$\arg\max_{y} \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

- But all these $P(\dots|\dots)$ are numbers < 1 (small numbers) and multiplying all of them together will end up in a very very small number and numerical underflow :(

⇒ **Reformulate the problem:** by adding a log before the product. [Because log is a strictly monotonic function, we know that maximizing the $\log(f(x))$ is the same as maximizing $f(x)$.]

⇒ $$\arg\max_{y} \sum_{t=1}^{T_y} \log\left(P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})\right)$$!

sum of log-probabilities

Normalization for Beam search

$\log P \leq 0$

- Another problem is that, both these objective functions (Π and $\Sigma\log$) prefer shorter sentences (Because longer sentences make Π smaller, an $\Sigma\log$ a larger negative number)

⇒ normalize the $\Sigma\log$ sum by number of words T_y in the sentence, to reduce the penalty for outputting longer sentences.

A Hyperparam to be tuned.

$$\arg\max_{y} \frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

Normalized Log Probability Objective

In practice use α to make the normalization softer.
$\alpha = 1$, completely normalize by length
$\alpha = 0$, no normalization
$\alpha = 0.7$ often used.

How to choose the final sentence?

Let's say we run Beam Search for 30 steps and $B=3$
- We see a lot of sentences with $T_y = 1, 2, \dots,$ up to 30
- We pick the best performing sentences and score them against this score. So, the best score wins.

← How to choose the Beam width B?

- The larger B, the more possibilities we are considering and so higher the chance of finding a good sentence.
- But the larger B, the higher the computational cost., and memory need.

- larger B : Better Results, slower
- smaller B : worse results, faster
  ↳ Because we keep track of less possibilities

- In practice, in production systems we see $B \approx 10$
- In research we see even $B = 100$ to 3000

1, 3, 10, 100, 1000, 3000

Note : Unlike exact search algorithms like BFS or DFS, Beam Search runs faster but is not guaranteed to find exact maximum for $\arg\max_y P(y|x)$.

Error Analysis in Beam Search : Error analysis for finding best params and tunning BS

- BS is an approximate search Alg; Also called a heuristics search app.
- Error Analysis also helps us in case of low performance to figure out whether it is the beam search that causes the problem, or it is the RNN models that need more work.

Example :

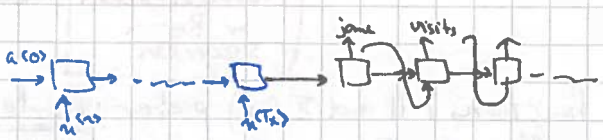
Error Analysis for RNN with Beam Search

input : Jane visite l'Afrique en Septembre.

human : Jane visits Africa in September (y^*) ← Better

Algorithm : Jane visited Africa last September ($\hat{y}$) ← worse ← ?

- Our model has two main components { RNN (encoder and decoder), Beam Search. } ← Which is responsible RNN or BS ??
  ↳ => • Should we collect more training data?
  • Should we increase the beam width?



- RNN computes $P(y|x)$

=> • let RNN compute $P(y^*|x)$ and $P(\hat{y}|x)$ ($>$? $\leq$?)

- depending on the result we can decide what to do next

Human : Jane visits Africa in September (y^*)
Alg. : Jane visited Africa last September ($\hat{y}$)

- let RNN compute $P(y^*|x)$ and $P(\hat{y}|x)$

Case 1 : $P(y^*|x) > P(\hat{y}|x)$

↳ • Beam search chose $\hat{y}$. But RNN gave a higher probability to y^*
=> Beam search is a fault. Because if fails to find a value of y that maximizes $\arg\max_y P(y|x)$.

Case 2 : $P(y^*|x) \leq P(\hat{y}|x)$

↳ • y^* is a better translation than $\hat{y}$. But RNN predicted $P(y^*|x) < P(\hat{y}|x)$ →

Note: In case we use normalization, we should evaluate the optimization Objective instead of the probabilities.

Error Analysis Process:

- Go through the development set, and find the mistakes the algorithm did.

Human	Algorithm	$P(y^*\|x)$	$P(\hat{y}\|x)$	At fault?
Jane visits Africa in Septembers.	Jane visited Africa in September.	2×10^{-10}	1×10^{-10}	Beam Search / RNN

=> We figure out what fraction of errors are due to BS and what due to RNN.

↳ We can attribute the error to either the search algorithm or to the Objective function of RNN that generates the objective function that BS is supposed to be maximizing.

- If RNN is at fault, then use other network improvement techniques (Course 3) to improve it (change architecture, add more data, add regularization, ...)
- Such error analysis is very useful, whenever we have an optimization algorithm (e.g. BS) that wants to optimize the objective function output by a learning algorithm.

Bleu Score:

Bleu Score

Papineni et al. '02; "Bleu: A method for automatic evaluation of machine translation"

- A difficulty in evaluating machine translation systems is that there are multiple correct answers possible. Unlike CV and Image recognition, where only one correct answer exists and hence accuracy is a good measure. How do you measure accuracy if there are multiple correct answers?

=> The **Bleu Score**

Example:

French: le chat est sur le tapis.

Both translations good and correct.
- Reference 1: The cat is on the mat.
- Reference 2: There is a cat on the mat.

Bleu Score: automatically measures a score that tells how good is a piece of machine translation.

Intuition: every translation that is near to the given reference translations, gets a high Bleu score.

Bleu: Bilingual evaluation understudy.
↳ (means substitute for an actor in theatre)

- Intuition: Look at machine generated output and see if types of words it generates appear in at least one of the human generated references.

Imagine: MT output: the the the the the the the.

precision: 1 if a word from output appears in the references.

↳ $\frac{7}{7}$ ← each word appears also in references / ← # words in output

=> Precision is not a useful measure here.

Credit & Modified Precision

Lets define Modified Precision:

- define credit as the max number of times a word appears in references.

=> "the" get credit 2 => Modified Precision = $\frac{2}{7}$ ← max count("the") in refs.

Bleu Score in Bigrams : Bigram : pairs of words appearing next to eachother.

- We don't want to look at words only in isolation. Let's look at pairs of words as well.

Ref 1: The cat is on the mat.
Ref 2: There is a cat on the mat.
MT output: The cat the cat on the mat.

Bigrams:	Count	$Count_{clip}$
the cat	2	1
cat the	1	0
cat on	1	1
on the	1	1
the mat	1	1
	6	4

Modified Bigram Precesion (P_2) = $\frac{4}{6}$

$$P_1 = \frac{\sum_{unigrams \in \hat{y}} Count_{clip}(unigram)}{\sum_{unigram \in \hat{y}} Count(unigram)}$$

(precesion, unigram)

for n-grams

$$P_n = \frac{\sum_{n\text{-grams} \in \hat{y}} Count_{clip}(n\text{-gram})}{\sum_{n\text{-grams} \in \hat{y}} Count(n\text{-gram})}$$

- Modified Precesnon Scores allow to measure the degree of how much is the MT similar (in other words "overlaps") with the reference translations.
  - ⇒ If MT output is exactly equal to one of the references, then all the values P_1 to P_n will be equal to 1.0 ($P_1 = P_2 = \dots = P_n = 1.0$)

Bleu Details : P_n = Bleu score on n-grams only

to compute the one Bleu score, by convention, compute P_1 to P_4 :

Combined Bleu Score : $BP \exp\left(\frac{1}{4}\sum_{n=1}^{4} P_n\right)$

BP : Bravity Penality : is an adjustment factor that penalizes too short translations.

(Because: It is easier to get high precesion if we output very short translations. Because most of the words we output appear in the references. But we don't want translations that are too short.)

$$BP = \begin{cases} 1 & \text{if } MT\_output\_length > reference\_output\_length \\ \exp(1 - MT\_output\_length / reference\_output\_length) & \end{cases}$$

see paper for details.

- Refer to course 3 on the importance of having one single real number evaluation metric.
  The Bleu score provided the field of MT with such a metric and it was revolutionary and accelerated the advancement in this field a lot.
  - It is also used to evaluate image captioning systems, and generally any kind of system that generates text. (Not always the ground truth)

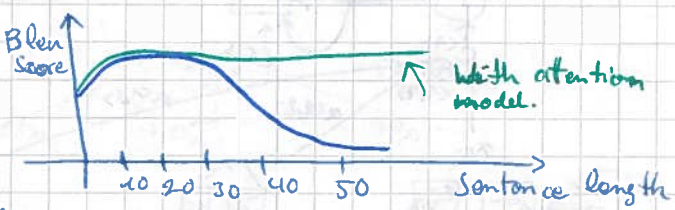
Attention Model Intuition: one of the most influencial ideas in DL that makes the encoder-decoder Architectures work much better.

Attention Model

The problem of very long sequences:

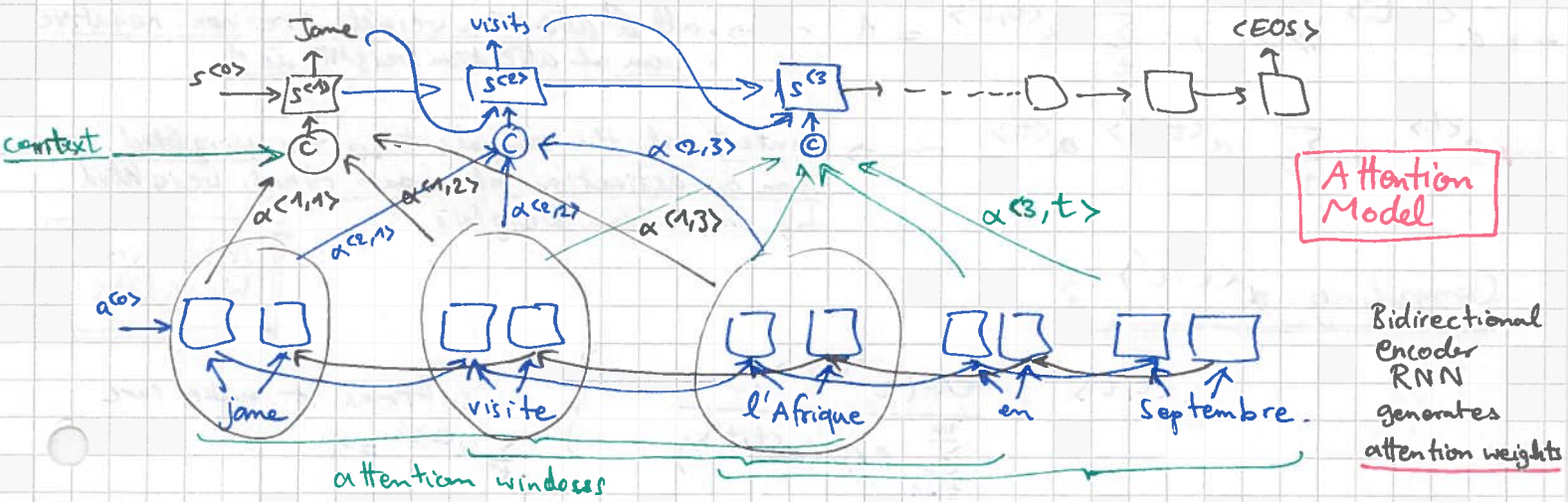
Bahdanau et al '14 "Neural machine translation by jointly learning to align and translate" Very seminal

- the NN cannot memorize the whole sentence in the encoder and decoder.
- Performance degrades when sentences become too long.
- A human translator would not memorize the whole sentence first before translating. He would translate a little part of sentence first, then another little part, Then another little part and so on. Attention Model does something similar.



- Attention Model (AM) originally developed for MT, but is also very successfully applied to other domains Very influencial. Also applied to image captioning.

- We the concept of "Attention Weights" to denote how much should you be paying attention to input words, and define a Window of attention in the input sequence.



- $\alpha^{<1,2>}$: how much attention should we pay to input word 2 in order to generate output word 1

- The amount of attention to pay at generating the 3rd output to the input french word at time t depends on the activations of bidirectional RNN at time t, and on the state of the previous step.

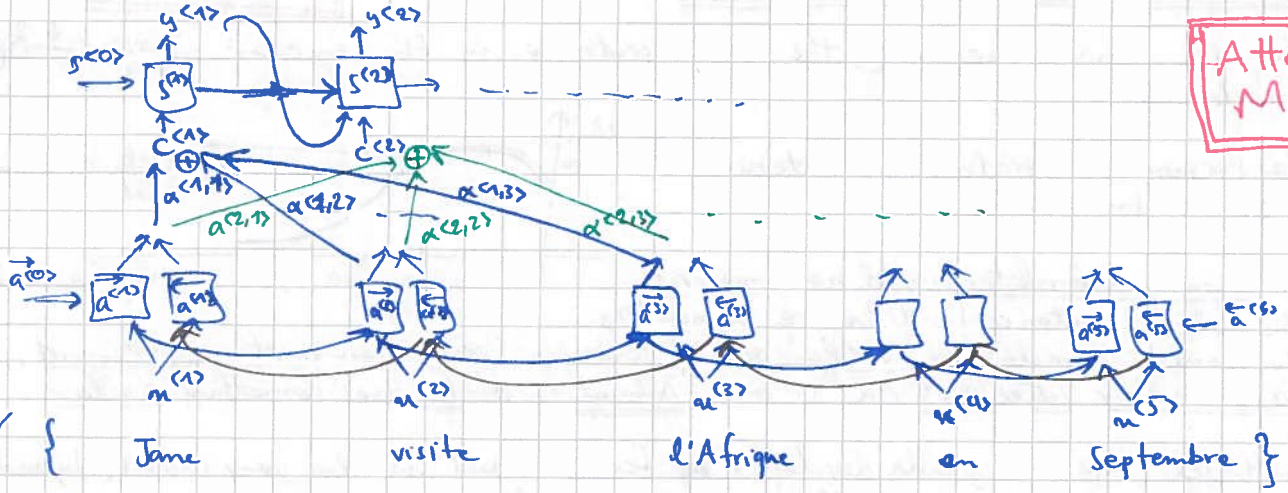
$\alpha^{<3,t>}$ depends on $\{ s^{<2>}, \vec{a}^{<t>}, \overleftarrow{a}^{<t>} \}$

- The RNN marches forward generating one word at the time governed by $\alpha^{<t,t'>}$ at each step, that tell how much attention to pay to the t'-th input word.

→

Attention Model:

- Attention model allows the NN to only pay attention to parts of the input sentence when generating the output sequence (much as a human translator would do)



→ $a^{<t'>} = (\vec{a}^{<t'>}, \overleftarrow{a}^{<t'>})$ activations of the BRNN at timestep t' of input

→ $\alpha^{<t,t'>}$: amount of attention $y^{<t>}$ should pay to $a^{<t'>}$

→ $\alpha^{<t,t'>} \geq 0$, $\sum_{t'} \alpha^{<t,t'>} = 1$ ⟶ • all attention weights are non-negative
• sum of attention weights is 1.

→ $c^{<t>} = \sum_{t'} \alpha^{<t,t'>} a^{<t'>}$ ⟶ Context of the output t is the weighted sum of activations of input steps, weighted by attention weights.

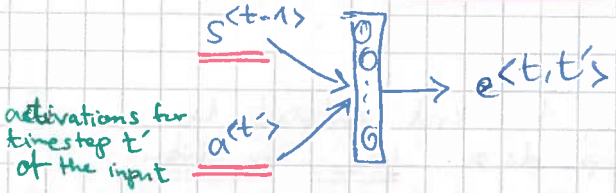
Attention Weights

Computing $\alpha^{<t,t'>}$:

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$

} a softmax to make sure $\sum_{t'} \alpha^{<t,t'>} = 1$

$e^{<t,t'>}$: use a small neural network to compute it based on $a^{<t'>}$ and $s^{<t-1>}$



→ a very small NN usually 1 hidden layer NN.

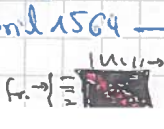
→ Intuition behind a small NN:

- Based on our architecture, we know $s^{<t>}$ depends on $s^{<t-1>}$ and $a^{<t'>}$
- But we don't know what is the function that computes each attention weight based on these values.
- So we use a small NN to compute this function.

- One downside of this algorithm is that it takes quadratic cost

total # $\alpha^{<t,t'>}$ $= T_x \times T_y$

Examples: Date Normalization:
July 20th 1969 ⟶ 1969-07-20
23 April 1564 ⟶ 1564-04-23



- Also visualizing attention weights shows attention weights are high :)

Speech Recognition :

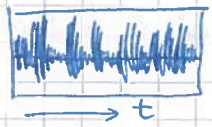
- The advances in sequance to sequence modelling has dramatically improved the speech Recognition systems.

Speech Recognition

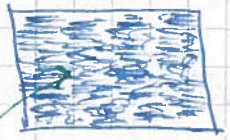
x → y

audio clip → transcript

changes in air pressure → "the quick brown fox"



preprocessing



color intensities showing the energy at that frequency in time

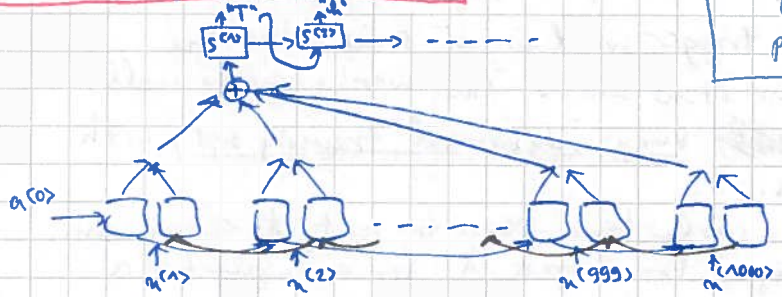
Spectogram (filter bank outputs)

(how loud is each frequency at each time)

- A very common preprocessing step, that very much corresponds to how human ear processes the audio signal (spectrogram, how loud or energyful are different frequencies at different times)

- Phonems used to be used largely for speech recognition
- But if we have enough data, now an E2E DL would perform better than hand-engineered phonems as input.

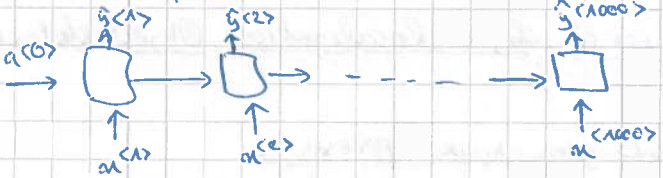
Attention Model for SR:



- horizontal axis: single time frames of the input signal
- Attention model outputs the transcript.

CTC cost for SR: (connectionist temporal classification)

"the quick brown fox"



- RNN with equal number of input & output.
- in practice a bidirectional RNN with LSTM or GRU and usually deeper

- Usually in SR, number of input time-steps is much bigger than the number of output time-steps. For example 10s of audio with 100 Hz ⇝ 1000 input features.
- The CTC allows the NN to output a sequence like:

ttt_ _h_eee____␣____q_qq___ } ⇝ the␣q

blank character

space character

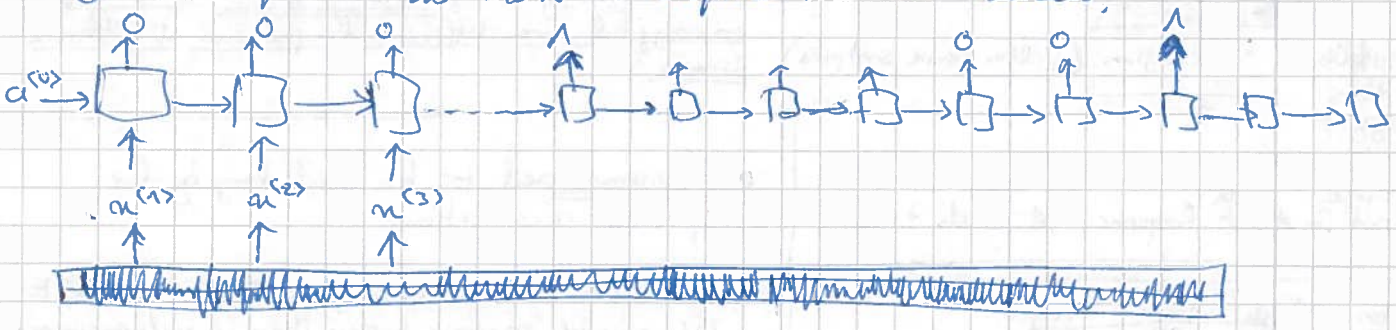
collapse repeated characters not seperated by blank.

→

- Building an effective and production scale SR system is a pretty significant effort and requires a lot of data
- Building a "Trigger word detection" System is much easier and requires much less amount of data.

Trigger Word Detection: "Alexa"; "Hey Siri"; "Okay Google"; "xiaodunihao" ↑Baidu.

- Still an evolving field of research. Many different approaches. No consensus yet on which is the best.
- We just use a normal RNN (eg. maybe bidirectional, with GRU, with LSTM, ...) as we have learned, with the input audio signal (maybe in form of an spectrogram.
- The question is how to define the labels?



- We can use output 0 when trigger word is not said, set the output to one 1 as soon as it is said. This works pretty well, but has the problem of a very imbalanced training set, with a lot more zeros than 1's.
  - → Instead of having one single time-step to output 1, we can have several timesteps to output 1, for example for a defined period of time.

Course 4 – CNNs – Week 3 – Object Detection

1 h 34 m video
2 h prog.

Week 3

Learning Objectives:
- Understand the challenges of object localization, Object detection and Landmark Finding.
- Understand and implement non-max suppression
- Understand and implement intersection over union
- Understand how to label a dataset for object detection task.
- Learn the vocabulary of Object detection (landmark, anchor, bounding box, grid, ...)

Object Localization: (OL)

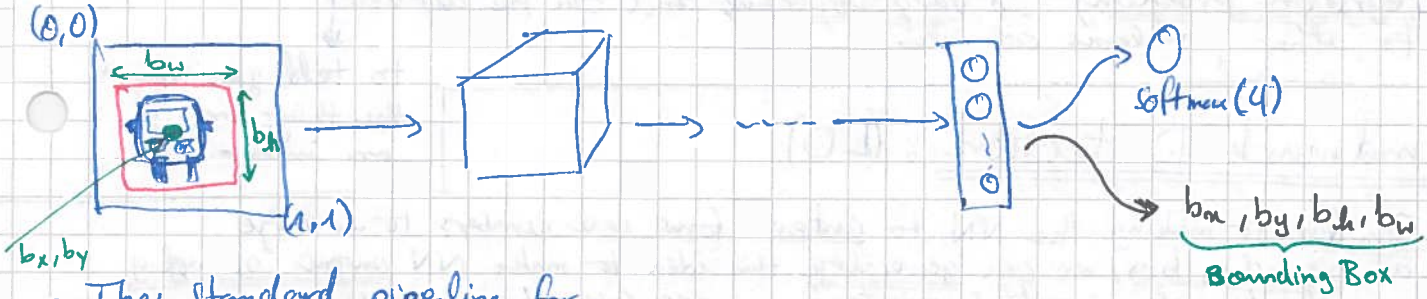
What are OL and OD? OD = Object detection

Classification: 1 Object → "Car"

Classification with localization: 1 Object and its location → "Car"

Detection: multiple Objects and their location. →

Classification with Localization: At most only 1 object in image



- The standard pipeline for classification, say with 4 classes: 1. pedestrain, 2. car, 3. Motorcycle, 4. Background.
- In order to do OL (object Localization) we extend this pipeline with few more output units that output a bounding box.
- Now the training set need to include the bounding box.
  In the example: e.g. $b_x = 0.5$, $b_y = 0.6$, $b_h = 0.3$, $b_w = 0.4$. These are not coordinates, but relative ratios.

Defining the Target label y:

Object Localization

1 - pedestrain
2 - Car
3 - Motorcycle
4 - Background ← $P_c = 0$

($P_c = 1$ for 1-3)

Output: b_x, b_y, b_h, b_w, Class label (1-4)

- We define output y as follows:

$$y = \begin{bmatrix} P_c \leftarrow \text{tells if there is any object} \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$$

- The training set needs to have bounding boxes now (b_x, b_y, b_h, b_w)

Example:

$x =$, $y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$

$x =$, $y = \begin{bmatrix} ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$? = "don't care"

Loss Function:

Loss Function

$$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \dots + (\hat{y}_8 - y_8)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \end{cases}$$

In this case only $\hat{y}_1$ matters, because all other elements of $\hat{y}$ are "don't care"

$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}$ — P_c: logistic reg. loss; b_x, b_y, b_h, b_w: squared error; C_1, C_2, C_3: log likelihood of softmax

- Note that if $y_1 = 1$, we penalize deviations in all 8 components; whereas if $y_1 = 0$ we only care about $\hat{y}_1$ (the P_c), and don't care about the rest.
- In this example we used squared error for all 8 components of y, just for simplification. In practice we normally use a log likelihood loss for C_1, C_2, C_3 of the softmax output, squared error for bounding box, and for P_c something like Logistic Regression loss.

- The idea of having the NN output a bunch of real numbers (like a regression problem) is very powerful and can be applied to other problems as well.
  - to tell you where the things are in an image

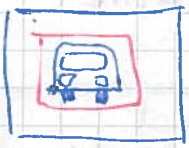
Landmark Detection: (LD)

- Similar to making the NN to output four real numbers to localize a bounding box, we can generalize this idea to make NN output x and y coordinates of important points in image (the Landmarks)

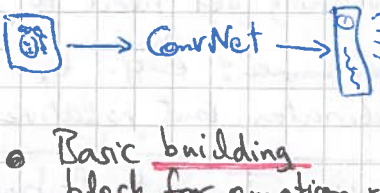
Landmarks

- Make NN automatically output (x,y) coordinates of important points in image

Example: Landmarks of a face

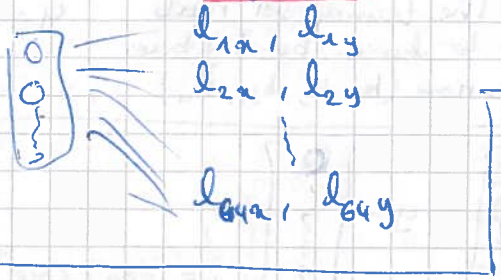


Bounding Box
b_x, b_y, b_w, b_h



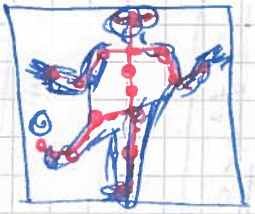
- Basic building block for emotion recognition and AR filters like Snapchat.

- We define let's say 64 keypoints in face



- An train the NN to output the (x,y) coordinates of those points.

- We can even use it for Pose detection



Amazing :)

Landmark Detection

Object Detection: Sliding Windows

- First train a ConvNet with a data set of closely cropped images of car

- Then crop the input image using a sliding window, and feed every time the small cropped portion of the input to the trained classifier and Hope that it will predict the cars.
- Change the window size and repeat the process.

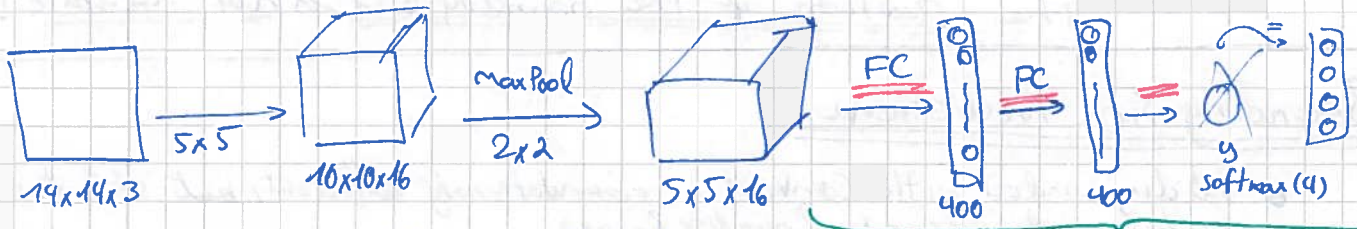
Sliding Windows

- Problem: Computational cost. For a single image we have to repeat the classification task several times. Specially in order to have good performance we have to choose a small stride

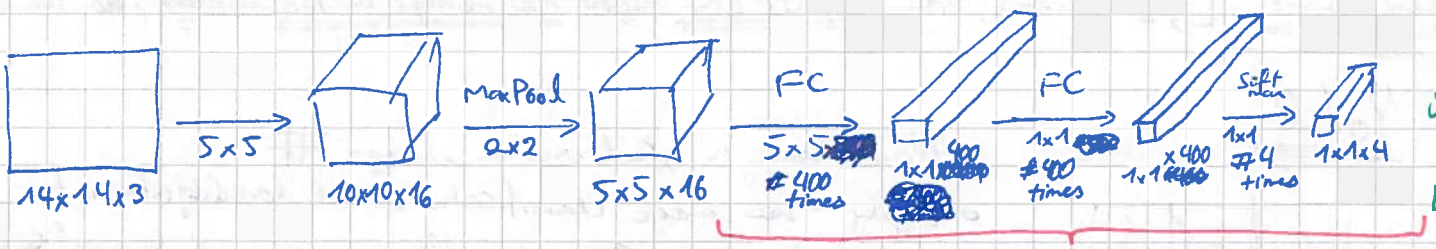
Solution

Convolutional Implementation of Sliding Windows:

Turning FC Layers into Convolutional Layers:



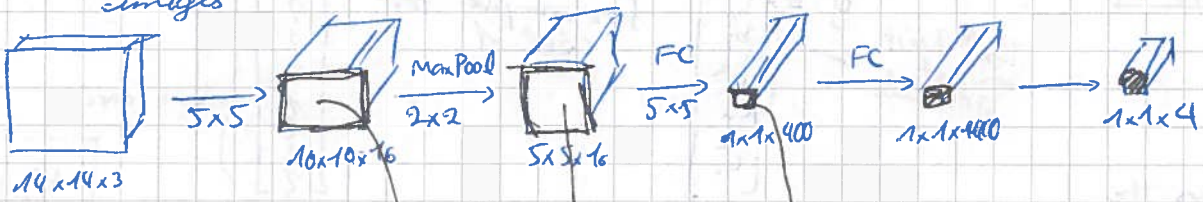
- We can turn the fully connected layers above into convolutional layers by using equivalent convolutional filters:



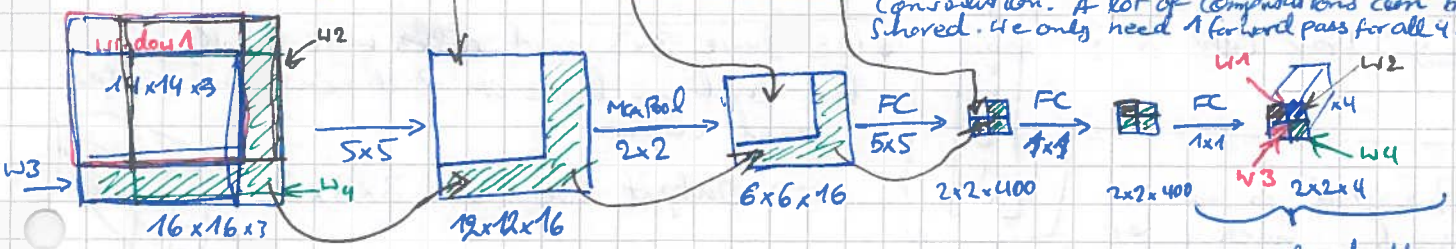
Convolutional Imple of Sliding Windows:

Sermanet et al '14
"OverFeat: Integrated recognition, localization, and detection using convolutional networks"

- Let's assume we have trained our ConvNet for 14x14x3 images



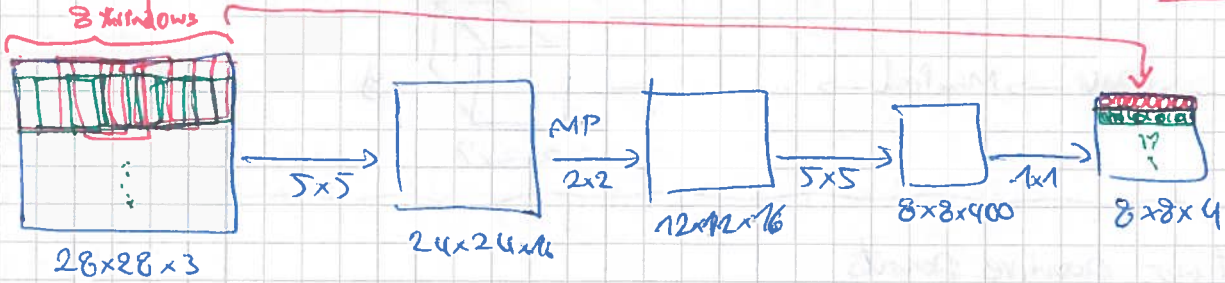
- Now we input a 16x16x3 Test image. Cropping and feed is the same as applying Convolution. A lot of computations can be shared. We only need 1 forward pass for all 4 windows.



- now each of the 1 pixel subsets of the 2x2x4 output volume corresponds to one of the input windows

~~Another example~~

- Another example: 28x28x3 input:



- Using Convolutional filters, we can do all the predictions of all the windows of an input image at the same time in one single forward pass.

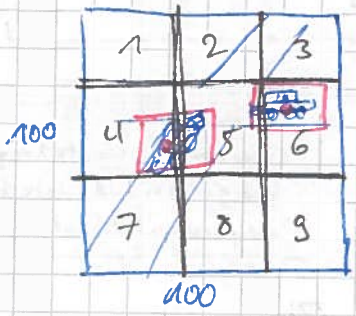
- Sliding window with ConvNets has still a problem:
  The position of the bounding Box is not accurate!

Bounding Box Predictions:

- Sliding window with ConvNet is computationaly efficient, but still has the problem of inacurate Bounding Boxes:
  - first, depending on the window size and stride, often, none of the Boxes really matches up with the car (object)
  - second, the objects BB might not always be rectangular, as the windows are

Redmon et. al '15 "You only look once: Unified real-time Object detection"

YoLO:



- Place down a grid on the image
- Apply the image classification and localization to each of those grids cells (in example 9 grid cells)
- labales for training for each of the grid cells:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

BB of object in that grid cell

for example for cell 1

$$\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$ ← no object; ? = don't care

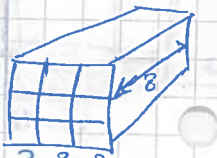
- Yolo considers the center point of the BB and assigns the object to the BB that contains the center point

YoLo

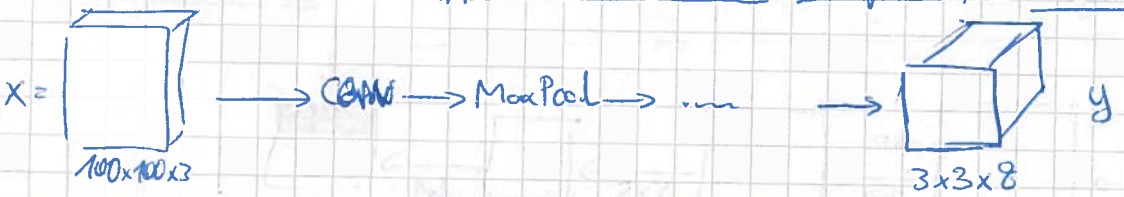
⇒ for cell 3:

$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

- We have 3x3 grid with 9 cells for each we have an 8 dimensional output target vector
  ⇒ Output volume: 3x3x8



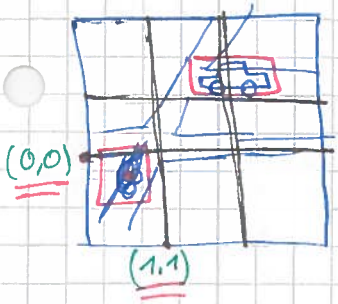
- The grid is in practice much finer than 3x3. ⇒ It is not probable that a cell contains two objects.
- Now we implement a typical ConvNet to output the desired out volume



Yolo has two positive points:

- Yolo is very much like the object localization algorithm, and outputs the BB coordinates explicitly ⇒ It can output BBs of any aspect ration and is not restricted to a forces window size.
  ↳ with very precise
- This is a convolutional implementation. The result for all grid cells is computed in one single forward pass.

How do we encode the BBs (b_x, b_y, b_H, b_W) in a cell grid?



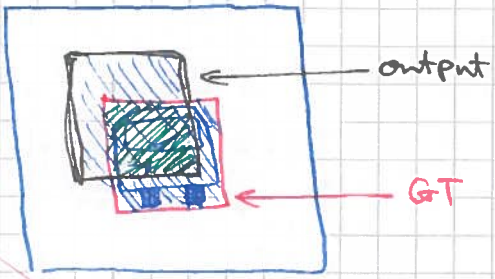
- b_x, b_y, b_W, b_H are defined relative to the grid cell.
- example:

$$\begin{bmatrix} 1 \\ 0.3 \\ 0.6 \\ 0.8 \\ 0.7 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

b_x and b_y must be $0 \leq b_x, b_y \leq 1$
can be ≥ 1, for the case the BB crosses the cell boundaries. !

How to measure the performance of an Object detection Algorithm?

Intersection over Union: IoU



IoU = Intersection over union of the two BBs

$$= \frac{\text{size of intersection}}{\text{size of Union}}$$

- in most CV tasks, by convention,
  cf $IoU \geq 0.5 \Longrightarrow$ "Correct detection"

IOU ≥ 1
=> Perfect Match

See the Rest in the attached pages ⟹ from 158

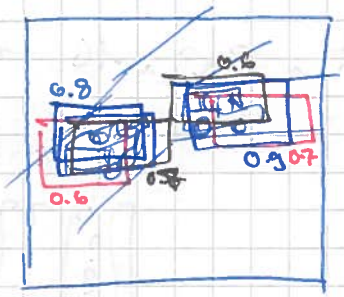
Next topic: Non-max Suppression ⟹ see attached Pages

Non-Max Suppression:

- So far our OD algorithm have the problem of multiple detection of the same object.

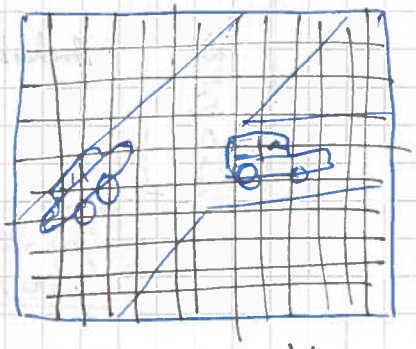
- Multiple cells might think that the center of the object is contained in them. I.e., multiple cells might have a P_c high.

Example:



- => Take the BB with the highest probability, and suppress all others with I_oU with it (high)
  - Suppres all overlapping rectangles.

Details:



19x19 grid

Each output prediction is $\begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$

- Assume we have only 1 class!

- Discard all boxes with $P_c \leq 0.6$ (Get rid of all low probability output boxes)
- While there are any remaining BBs:
  - pick the Box with the largest P_c and output that as prediction.
  - Discard any remainning BB with $I_oU \geq 0.5$ with the output box.

Non-Max Suppression

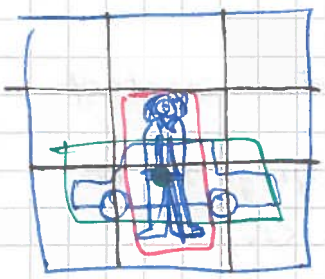
- For multiple classes, run separate Non-max suppression once for each of the classes

---

Anchor Boxes:

- So far we have the problem that each of the grid cells can detect only one objet.

Overlapping Objects:



- previously: each cell only one object => $y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$

Anchor box 1:

Anchor box 2:

- now each cell multiple objects each corresponding to one anchor box

$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ Anchor box 1 / Anchor box 2

Anchor Box Algorithm:

- Previously:
  Each object in training image is assigned to grid cell that contains that object's midpoint

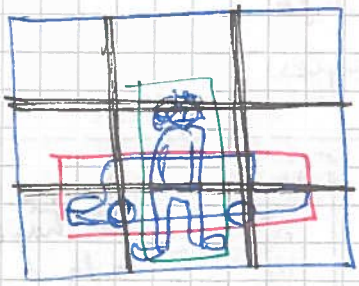
=> Output y: 3x3x8

- With two anchor boxes:
  Each object in the training set is assigned to grid cell that contains objects midpoint and anchor box for the grid cell with highest IoU

=> (grid cell, anchor box)

=> Output: 3x3x16 ~> (3x3x2x8)

Example



$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$

Car only?

$\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$

Anchor box 1

Anchor box 2

Anchor box 1: Anchor box 2:

Anchor Boxes

Note: Two cases are not handled well with this algorithm:

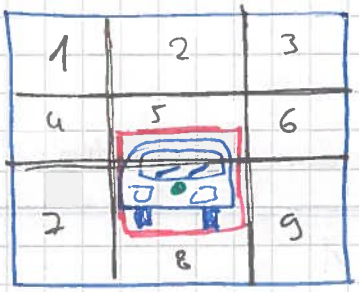
- If we have, say 2 Anchorboxes, but three overlapping objects.
- If we have two objects, that correspond both to the same anchor boxshape

- Generally, in practice, with a fine grid (say 19x19) the chance of two objects overlapping is not that high.
- Anchor boxes allow your algorithm to specialize better. Particulary if your Dataset has tall objects and wide objects.

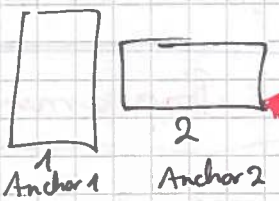
How to choose Anchor boxes?

- normally chosen by hand, 5 or 10 shapes, that represent a variety of shapes of objects.
- Even better is to use a k-Means algorithm to group together different shapes of objects and choose anchor box shapes to cover a wide range of object groups.

Yolo Algorithm:



3 classes
1 - Pedestrain
2 - Car
3 - Motorcycle



$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$

$y_{cell\,1} = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$ ← p_{c_1} ← p_{c_2}

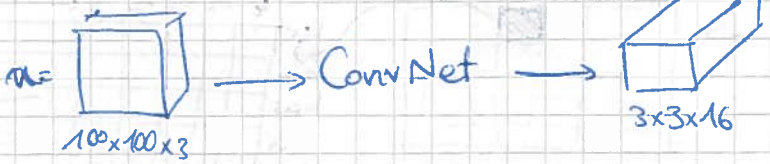
$y_{cell\,8} = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$

y is $3 \times 3 \times 2 \times 8$
- 3×3: grid
- 2: # anchors
- 8: $5 + \#\,classes$ → p_c, b_x, b_y, b_h, b_w

Anchor box is higher IoU with the GT.

Training Set:

- Now lets train a ConvNet:



- Making Predictions: as you expect
- Outputting the non-max suppressed outputs:
  - for each grid cell, get 2 predicted bounding boxes (for two anchors)
  - Then get rid of low probability predictions.
  - for each of the classes, independently run the non-max suppression.

Region Proposal:

↳ quite influential in CV.

Girshik et al. '13;
"Rich feature hierarchies for accurate object detection and semantic segmentation"

Region Proposal
R-CNN

- Many areas of an input image are often empty. There is nothing interesting in them.

  ⇒ Run the convnet only on selected regions

1. Use a segmentation algorithm to find proposed regions (blobs)
2. Run the classification on the proposed regions one at a time outputing label + BB

} R-CNN

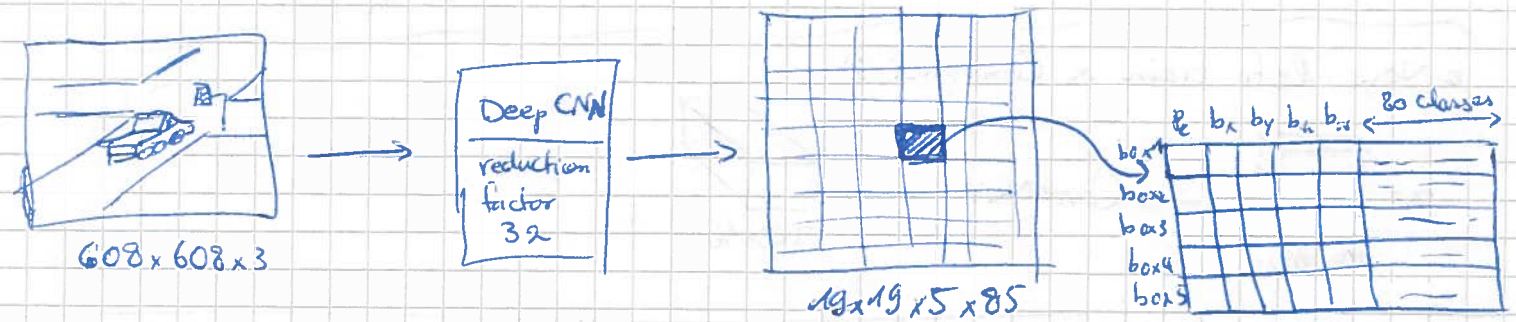
→

- Fast R-CNN:
  - Propose regions. Use convolution implementation of sliding windows to classify all the proposed regions.
- Faster R-CNN:
  - Use conv nets to propose regions instead of a classical algorithm.

Course 4 - CNNs - Week 3 - Programming 1

Car Detection with YOLO: 80 classes, 5 Anchors

- Input: Batch of images of shape $(m, 608, 608, 3)$
- Output: list of BBs with recognized classes. Each BB has is represented by 6 numbers $(p_c, b_x, b_y, b_h, b_w, c)$. Classes if expanded into a 80 dim one-hot vector $\Rightarrow$ Each BB is a 85 dim vector.



- We flatten the last two dimension (for simplicity)
  $\Rightarrow$ output shape $19 \times 19 \times 425$

Summary of YOLO:

- Input image (608, 608, 3)
- The input image goes through a CNN, resulting in a (19, 19, 5, 85) dimensional output.
- flattening the last two dimensions of the output $\rightarrow$ (19, 19, 425)
  - Each cell in a 19x19 grid over the input image gives 425 numbers.
  - $425 = 5 \times 85$. Each cell contains predictions for 5 boxes, corresponding to 5 Anchorboxes
  - $85 = 5 + 80$, 5 because of $(p_c, b_x, b_y, b_h, b_w)$ and 80 then number of classes.
- We then select only few boxes based on:
  - score thresholding: throw away boxes that have detected a class with a score less than the threshold
  - Non-max Suppression: Compute the intersection over union and avoid selecting over lapping boxes.
- This gives us YOLO's final output.

Course 4 - CNNs - Week 4 - Special Applications

Week 4 — Face Recognition & Neural Style transfer

What is Face Recognition?

- A Demo video from Baidu's HQ in China.
  - Face Recognition
  - liveness detection

  } opens entrance gate

Face Verification vs. Face Recognition:

Verification:

1:1 Problem

- Input: Image, name/ID
- Output: Whether the input image is that of the claimed person.

We build a very accurate face verification system, and use that for face recognition.

Recognition:

1:K Problem

- Has a Database of K persons
- gets an input image
- Outputs ID if the image is any of the K persons (or "not recognized")

↳ much harder than verification problem.

↳ • Imagine you have a FV system with 99% accuracy
• Now you apply it to a dataset with 100 persons ⇒ you have 100 times 1% chance of making a mistake ⇒

! → • Your Face Verification System needs to be very very (99,9 ≥) accurate to be used for Face Recognition

- One of the main problems that needs to be solved is "One-Shot Learning"

One Shot Learning:

- You need to be able to recognize a person given only one sample or one example of that person's face!
- Historically DL algorithms don't work well, if you only have one training example.

→

One-Shot learning :

• Learning from one example to recognize the person again.

↳ Because we might have only one picture of the person in our database.

• Not working solution

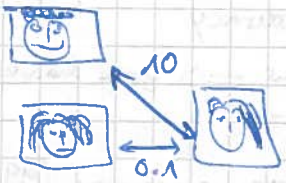
• It does not work:
  • we don't have enough data to train it robustly.
  • What if a new person is added to the DB?
    ↳ Do we have to train the CNN over again?

⇒ Let's learn a "Similarity function"

d(img1, img2) = degree of difference between two images

• d small for two images of one person.

If $d(img^1, img^2) \leq \tau$ ⟶ "same person"
$> \tau$ ⟶ "different"
} Verification.

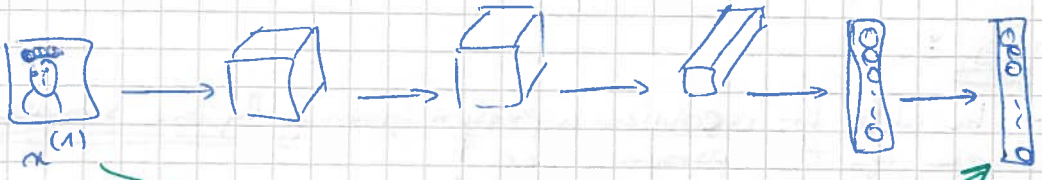


• Compare the new input image with all the images of the DB and compute the d
• If now a new person is added to the DB, we don't need to learn anything new.

Siamese Network :

• A good way to calculate the d() function

Taigman et al' 14, "DeepFace: closing the gap to human-level performance"



• If we believe that $f(x^{(1)})$ and $f(x^{(2)})$ are good representations of $x^{(1)}$ and $x^{(2)}$, we can define $d(x^{(1)}, x^{(2)}) = \| f(x^{(1)}) - f(x^{(2)}) \|_2^2$

→

← • The idea of running two identical CNNs on two different inputs and then comparing them, is called a Siamese Network.

@ How to train Siamese NNs?

↳ The two NNs in previous page have same parameters

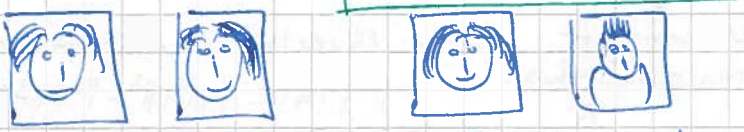
→ So we want to train a neural network, so that the encoding it computes results in a function d that tells us when two pictures are of the same person.

- Parameters of NN define an encoding $f(x^{(i)})$ ← say a 128 dim vector.
- Learn Parameters so that:
  - If $x^{(i)}, x^{(j)}$ are the same person, $\| f(x^{(i)}) - f(x^{(j)}) \|^2$ is small
  - If $x^{(i)}, x^{(j)}$ are different persons, $\| f(x^{(i)}) - f(x^{(j)}) \|^2$ is large.

Schroff et al. '15, "FaceNet: A unified embedding for face recognition and clustering"

Triplet Loss :

Learning Objective :



Anchor (A) Positive (P) Anchor (A) Negative (N)

- We want the encoding between Anchor and positive to be very similar
- Whereas the encoding of anchor and negative should be far from eachother.

⇒ Term, Triplet loss, because at any time we are looking at three images (A, P, N)

What we want:

$$\underbrace{\| f(A) - f(P) \|^2}_{d(A,P)} \leq \underbrace{\| f(A) - f(N) \|^2}_{d(A,N)}$$

$$\Rightarrow \| f(A) - f(P) \|^2 - \| f(A) - f(N) \|^2 \leq 0$$

Problem : $f(img) = \vec{0}$ will trivally satisfy this equation; or also $f(img_1) = f(img_2)$ $\forall img_1, img_2$

↳ ⇒ To make sure our NN does not learn these trivial useless encodings modify the objective function:

$$\| f(A) - f(P) \|^2 - \| f(A) - f(P) \|^2 + \alpha \leq 0$$

α is now the margin

↓ similar to margin in SVM.

← • now let say the margin is $\alpha = 0.2$

$$\|f(A) - f(P)\|^2 + \alpha \leq \|f(A) - f(N)\|^2$$

$$\Rightarrow |d(A,P) - d(A,N)| \geq 0.2$$

$\Rightarrow$ Margin defines the gap between distances of the anchor and the Positive and the distance of the anchor and the negative.

• The margin pushes the (A,P) pair and the (A,N) pair, further away from eachother.

Loss function:

Given 3 images A, P, N:

$$\mathcal{L}(A,P,N) = \max\left(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0\right)$$

We wont to minimize this. | The Objective function was: $\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha \leq 0$

Cost function:

$$J = \sum_{i=1}^{m} \mathcal{L}(A^{(i)}, P^{(i)}, N^{(i)})$$

• So, if we have a Traning set of 10,000 pictures of 1000 persons, we generate triplets from the 10000 pictures and then train the algorithm to minimize that cost function.

• for the purpose of training the system, we do need pairs of A and P. Pairs of pictures of the same person.

• But after training, we can use the network for doing one-shot learning.

Choosing the Triplets, A, P, N

• During training if A, P, N are chosen randomly, $d(A,P) + \alpha \leq d(A,N)$ is easily satisfied. :(

↳ Because if we choose A and N randomely, chances are high that they are very different and so the $\|f(A) - f(P)\|^2 + \alpha \leq \|f(A) - f(N)\|^2$ is easily satisfied and the NN does not learn much from it

⇒ We want to choose triplets that are "hard" to train on.

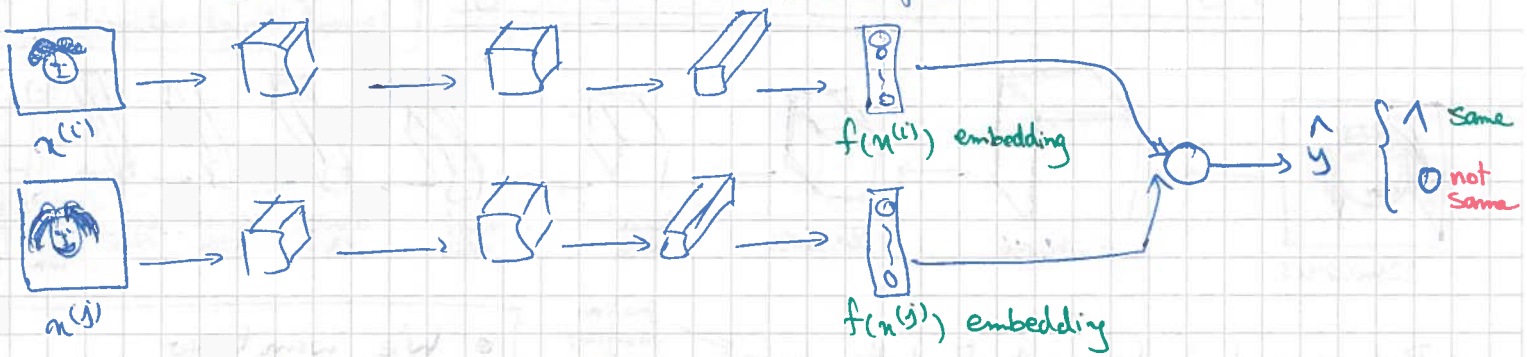
⇒ That is choosing triplets for which $d(A,P) \approx d(A,N)$

↳ so that the NN learns to learn to push down $d(A,P)$ push up $d(A,N)$.

- If the problem is too easy, the GD does not have much to do, and the algorithm does not learn well.

Face Verification and Binary Classification:

- Posing FR problem as a binary classification problem.
  - This is an alternative to the triplet loss



- Feed the $f(x^{(i)})$ and $f(x^{(j)})$ embedings into a logistic regression unit to make the prediction.
  The target output = 1 if same person, and = 0 if ~~one~~ different person.

$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_i \left| f(x^{(i)})_k - f(x^{(j)})_k \right| + b\right)$$

↳ element-wise difference of embedding vectors → other formulas also possible:

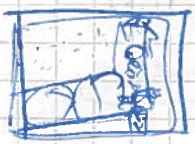
$$\chi^2 \text{ similarity}$$

$$\frac{\left(f(x^{(i)})_k - f(x^{(j)})_k\right)^2}{f(x^{(i)})_k + f(x^{(j)})_k}$$

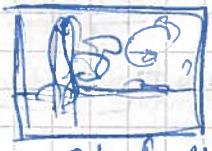
- The input is a pair of images → The output is 0 or 1
- We train a Siamese Network again

- Computational Tip:
  - for the images in the DB, we can have already the embedding computed, and saved.
    ↳ Then we only need to compute the embedding for the new input image.

Neural Style Transfer:



Content Image (C)



Style (Van Gogh) (S)

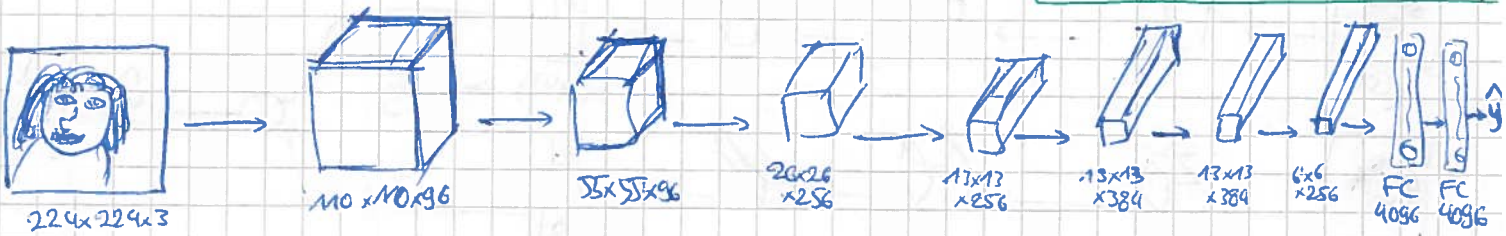
generated Image. (G)

Important:

- Neural Style transfer is a specific application of supervised learning in which
  - The network weights does not change. They don't get optimized.
- We use a pre-trained network, trained on a dataset of images. This network is capable of detecting features. We use the activations of this network (i.e. the features detected) to optimize our generate image

Visualizing what a Deep network is learning

Zeiler et al '13; "Visualizing and understanding convolutional networks"



- Pick a unit in layer 1. Find the nine (← just an arbitrary number.) image patches that maximize the unit's activation.
  - ↳ Pass your training set through the NN and find out what is the image that maximizes that particular unit's activation.

- We want to visualize what the hidden ~~layers~~ units in different layers are computing

Earlier layers look at smaller patches of the input image and find more fine grained features like lines.
Later layers look at larger patches of the input image and find more complex features, like faces and dogs.

Important lecture, but hard to understand → See video

Neural Style Transfer Cost function:

Gatys et al '15; "A neural Algorithm of artistic style"

$J(G)$: measure how good is a generate image G

↳ We separate $J(G)$ into two components

$$J(G) = \alpha J_{content}(C,G) + \beta J_{style}(S,G)$$

We also define two Hyperparameters α and β to specify the relative weighting between the content and style costs.

$J_{content}(C,G)$: How "similar" is G to the input content image.

$J_{style}(S,G)$: How "similar" is G to the input Style image.

Finding the Generated Image G :

1. Initialize G randomly

   $G : 100 \times 100 \times 3$

2. Use Gradient descent to minimize $J(G)$

   $G := G - \frac{\partial}{\partial G} J(G)$ } ⟶ updating the pixel values of the image G

Content cost Function :

$$J(G) = \alpha\, J_{content}(C, G) + \beta\, J_{Style}(S, G)$$

- Say we use hidden layer l to compute content cost.
  - if l is small, say first layer, then the generated image will be very much like at pixel level like the input content image
  - If l is very deep, then the layer figures out there is a dog in your input content image and says, well, let's put a dog somewhere in the output generated image.
  - So we want l to be somewhere in between. Neither too shallow nor too deep.

- Use a pre-trained ConvNet. (Eg. VGG network)
  - ↳ We want to measure, given a content image C and a generated image G How similar are their content.

- Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar both images have similar content.

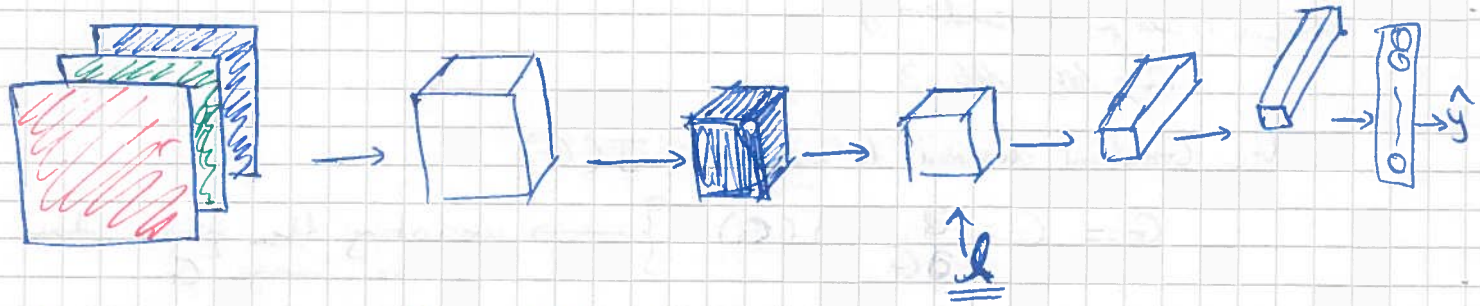
$$J_{content}(C, G) = \| a^{[l](C)} - a^{[l](G)} \|^2$$

We can also have a normalization factor, say $1/2$ there. But it does not matter

unroll both activations into vectors and calculate their difference.

Style Cost Function :

Meaning of the "Style" of an Image:



- Say we are using layer l's activation to measure "Style"
- We define Style as correlation between activations across different channels.
  - Correlation between activations means, whenever part of the image that fires that activation is present, another part of the image that fires the other activation is also present
    - feature

See Video

- The correlations tells us, which of the high level features/ textures / forms / lines / shapes / of the image occur together. (like, whenever you have vertical narrow lines, you have an orange like colored texture)

"Gram matrix"

Style Matrix: Captures all the correlations between activations across different channels.

- Let $a^{[l]}_{i,j,k}$ = activation at (i,j,k). $G^{[l]}$ is $n_c^{[l]} \times n_c^{[l]}$ → Style Matrix
  - i: H, j: W, k: C channel; (i,j): position
- $G^{[l]}_{kk'}$ measures how correlated are the activations in channels k and k'; $k, k' = 1, \ldots, n_c$

Because we want to measure how correlated are each pair of channels (pairwise correlations between channels)

$$G^{[l]}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a^{[l]}_{ijk} \cdot a^{[l]}_{ijk'}$$

Technically this is not correlation, but the unnormalized cross-covariance. But fulfills the same purpose

- If both activations are large together, then $G^{[l]}_{kk'}$ is large for these two channels, and
- If activations are not both large i.e. uncorrelated, G will be small.

i.e. correlated

- We compute $G^{[l]}$ for both the input & Style image and the Generate image G

$$J^{[l]}_{style}(S,G) = \frac{1}{(2 n_H^{[l]} n_W^{[l]} n_c^{[l]})^2} \left\| G^{[l](S)} - G^{[l](G)} \right\|_F^2$$

(normalization factor (does not matter); Frobinus norm: of element-wise diff.)

$$= \frac{1}{(\cdots)} \sum_{k} \sum_{k'} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'} \right)^2$$

← Style Cost Function:

$$\| G^{[l](S)} - G^{[l](G)} \|_F^2$$

$$J^{[l]}_{style}(S,G) = \frac{1}{(2 n_H^{[l]} n_w^{[l]} n_c^{[l]})^2} \sum_k \sum_{k'} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'} \right)^2$$

- It turns out that we get more visually pleasing results, if we use the style cost function from multiple different layers.

So the overall Style Cost Function can be defined as:

$$J_{style}(S,G) = \sum_l \lambda^{[l]} J^{[l]}_{style}(S,G)$$

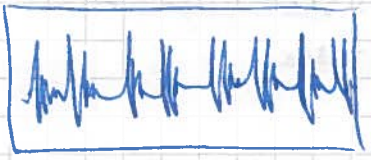
$\lambda^{[l]}$ ← weighting factor for style cost function of different layers.

- This allows us to take to account style cost of different layers in CNN
  - → erlier ones, considering lower level features like edges
  - → and later ones, considering some higher level feaures.

Convolutions on 1D and 3D Data:

- Exactly as we apply Convolutions on 2D data (like images), we can also apply a convolution on 1D or 3D data

Example: EKG

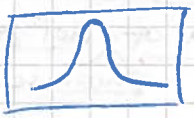


1 20 15 3 18 14 4 17 19

1D signal

Say 14 ← size

*



1 3 10 3 1

1D filter

filter size 5

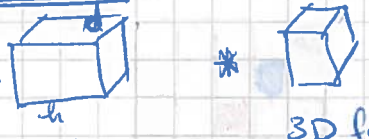
$\rightsquigarrow$ $14 \times 1 \ * \ 5 \times 1$ (16 filters)

$\rightarrow 10 \times 16$

$10 \times 16 \ * \ 5 \times 16$ (32 filters)

$\rightarrow 6 \times 32$

3D data:



3D data Volume

3D filter

$\#n_c \rightarrow \times 1$

$14 \times 14 \times 14 \ * \ 5 \times 5 \times 5 \times 1$ (16 #filters)

$\rightarrow 10 \times 10 \times 10 \times 16$

(32 filters)

$\rightarrow 6 \times 6 \times 6 \times 32$

- CT scan, CAD 3D models
- Video, motion data.

All the same ideas as we saw in 2D data also apply here in 1D data

Course 4 - CNNs - Week 4 - Programming 1

Art Generation with Neural Style Transfer :

- In Neural Style Transfer, we optimize a cost function to get pixel values. (as opposed to other approches, where we optimized a cost function to get parameter values)
- NST can be considered as a transfer learning task. It uses another pre-trained network and builds on top of that.
- We use VGG-19 (19 layers) trained on the very large ImageNet database.
  - ↳ This model has already learned to recognize a variety of low-level features (at earlier layers) and high-level features (at later layers)
- We load the weights in a dictionary. Each layer.

`model["input"].assign(image)` ← feed the image through network

After that
- ↳ To get the activation of a layer : `sess.run(model["conv4_2"])`

- NST in three steps :
  - ↳ Build the content cost $J_{content}(C,G)$
  - Build the style cost $J_{style}(S,G)$
  - Overall cost $J(G) = \alpha J_{content}(C,G) + \beta J_{style}(S,G)$

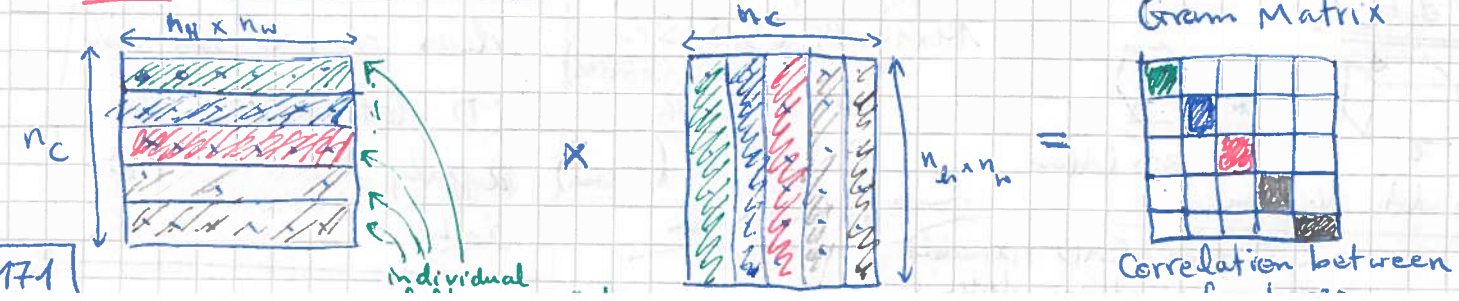
`tf.transpose()`, `tf.reshape()`, `tf.reduce_sum()`, `tf.square()`

Gram Matrix : Gram matrix of a set of vectors $(v_1, \dots, v_n)$ is the matrix of dot products of vectors (pair-wise)

$$G_{ij} = v_i^T v_j = \text{np.dot}(v_i, v_j)$$

In other words G_{ij} computes how similar are v_i and v_j. If v_i and v_j are similar, then their dot product would be high.

In NST we can compute the Style matrix by multiplying the "unrolled" filter matrix with its transpose (refer to notebook)



The resulting

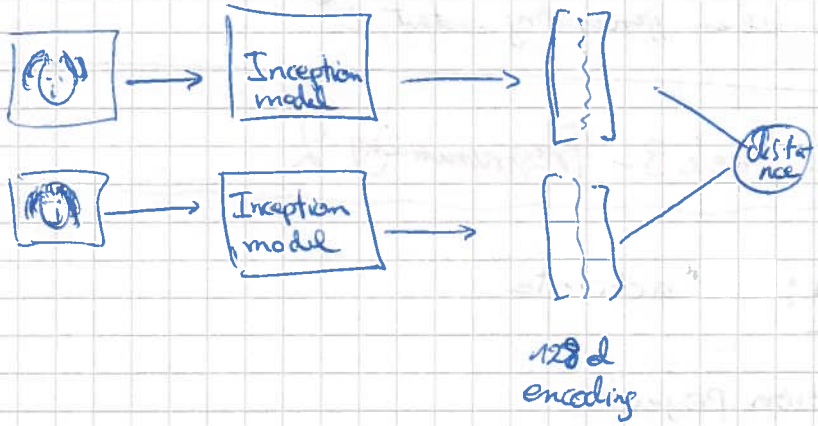
- Gram matrix is a $n_c \times n_c$ matrix. The G_{ij}'s measure how similar the activation of filter i is with the activation of filter j
- The diagonal of the G matrix (G_{ii}'s) is a measure of how active filter i is. For example, let's say filter i is detecting vertical textures in the image; Then G_{ii} measures how common are vertical textures in the image as a whole.
  If G_{ii} is large, this means the image has a lot of vertical texture.
- By capturing the prevelance of different types of features (G_{ii}) as well as how much different features occur together (G_{ij}) the style matrix G measures the style of an image

Course 4 - CNNs - Week 4 - Programming 2

Face Recognition

- Implement the Triplet loss function
- Use a pretrained model to map face images into 128-dimensional encodings
- Use encodings to perform face verification and face recognition.

- We use Inception model and channel first for input data.



Triplet loss:

$$J = \sum_{i=1}^{m} \left[\left\| f(A^{(i)}) - f(P^{(i)}) \right\|_2^2 - \left\| f(A^{(i)}) - f(N^{(i)}) \right\|_2^2 + \alpha \right]_+$$

$[z]_+ = \max(z, 0)$.

Cours 5 - Sequence models - Week 3 - Programming 1

Neural Machine Translation (NMT) with Attention Model:

- Translating human readable dates into machine readable dates:
  - "the 29th of August 1958" ⟶ 1958-08-29
  - "03/30/1968" ⟶ 1968-03-30
  - "24 JUNE 1987" ⟶ 1987-06-24

  YYYY-MM-DD

- Dataset 10 000 pairs ("9 May 1998", "1998-05-09")
- human vocab : Python dict mapping all characters of human readable dates to int.
- machine vocab : Python dict = all characters of machine readable dates to int.

$T_x = 30$ max length of human readable date

$T_y = 10$ (YYYY-MM-DD)

- Machine translation models can map one sequence to another. They can be used for tasks like date format translation too
- We can visualize attention weights $\alpha(t,t')$ to know to what input t' is the network paying attention when generating output t.

Course 5 - Sequence Models - Week 3 - Programming 2

Trigger Word Detection: "activate"

- Structure an Speech recognition project
- Synthesize and process audio Recordings to create train/dev datasets.
- Train a trigger word detection model.

- Data set ⟶
  - mixture of positive / negative words
  - different background noise
  - different accents.

  } Synthesize DS.

- Synthesizing data also allows us to easily set the ground truth labels. ⟶

← • Input: 5511 = T_x (spectogram output

• Output: 1375 = T_y

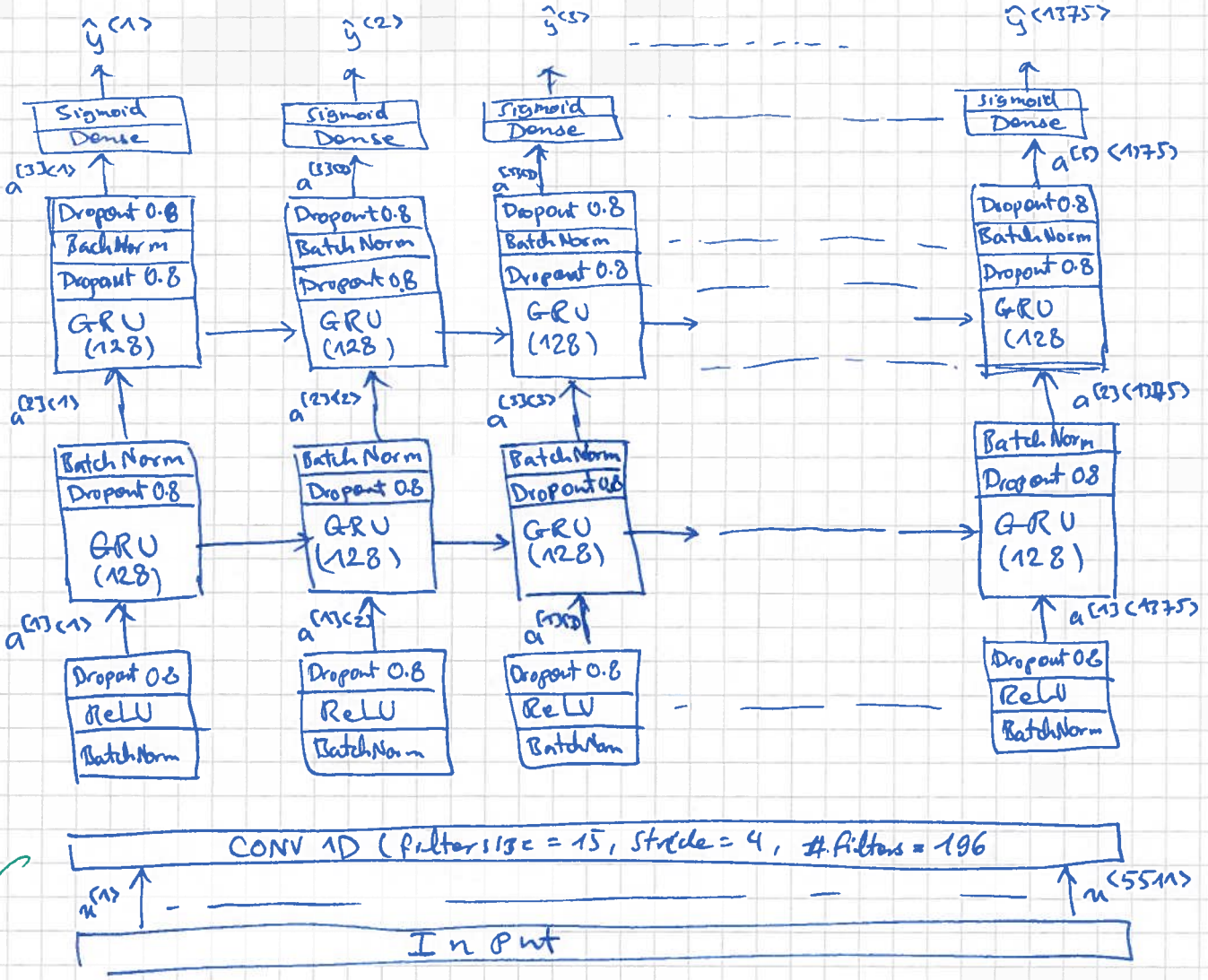
raw audio: 441000 samples / 10 sec.

• output = 1 exactly right after someone finished saying "activate"
time-step + 50 further timesteps.

• Synthesizing data:
  • Pick a random 10 sec background noise audio clip.
  • randomly insert 0-4 audio clips of "activate" into it
  • randomly insert 0-2 audio clips of negative words into it

---

Model:
• 1D conv layers; GRU layers; dense layers.



• Uni-directional GRU.

• make input smaller (5511 ⇝ 1375) ⇒ speed up the network

• detect / extract first set of low-level features

Fantastic!

- Data Synthesis is an effective way to create a large training set for speech problems, specifically trigger word detection.
- Using a Spectrogram and optionally a 1D convolutional layer is a common pre-processing step prior to passing audio data to an RNN, GRU, or LSTM.
- An end-to-end deep learning approach can be used to build a very effective trigger word detection system.