Session 4 & Assignment

Due No Due Date **Points** 0 **Submitting** a website url

•

UNET, ENCODER-DECODER ARCHITECTURE, ATTENTION MECHANISMS & LATENT SPACES

UNET ARCHITECTURE

The ***UNET*** **(https://arxiv.org/abs/1505.04597?source=post_page---------------------------)** was developed by Olaf Ronneberger et al. for Bio Medical Image Segmentation. The architecture contains two paths. First path is the contraction path (also called as the **encoder**) which is used to capture the context in the image. The encoder is just a traditional stack of convolutional and max pooling layers. The second path is the symmetric expanding path (also called as the **decoder**) which is used to enable precise localization using transposed convolutions. Thus it is an **end-to-end**

fully convolutional network (FCN), i.e. it only contains Convolutional layers and does not contain any Dense layer because of which it can accept image of any size.

In the original paper, the UNET is described as follows:

2

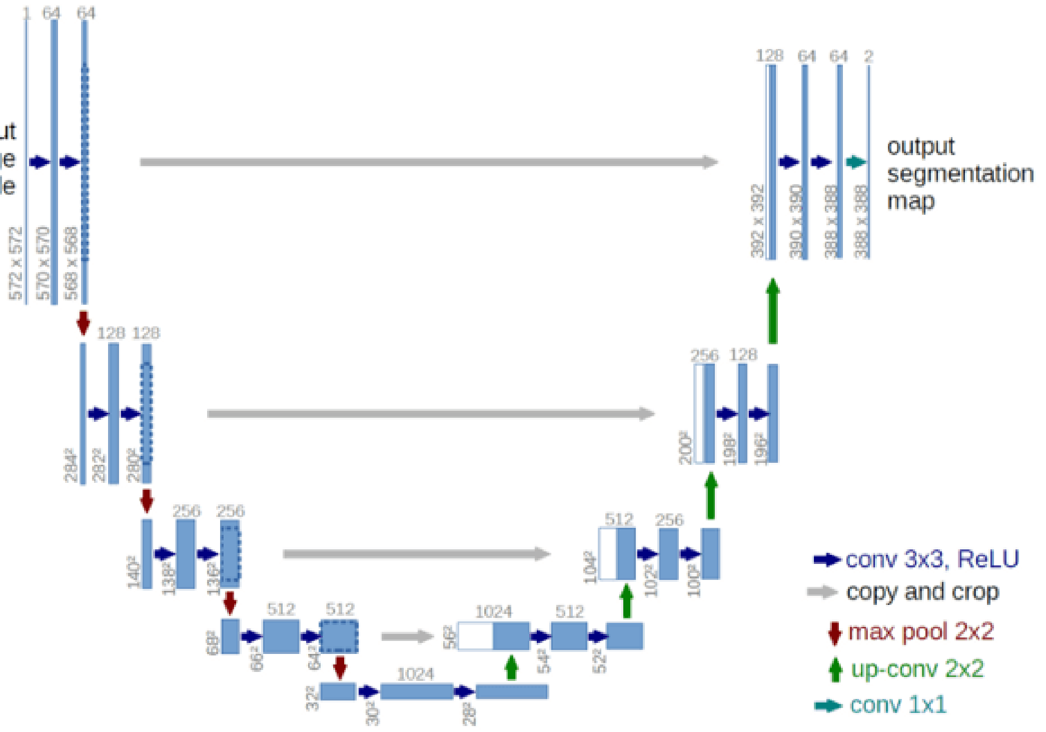
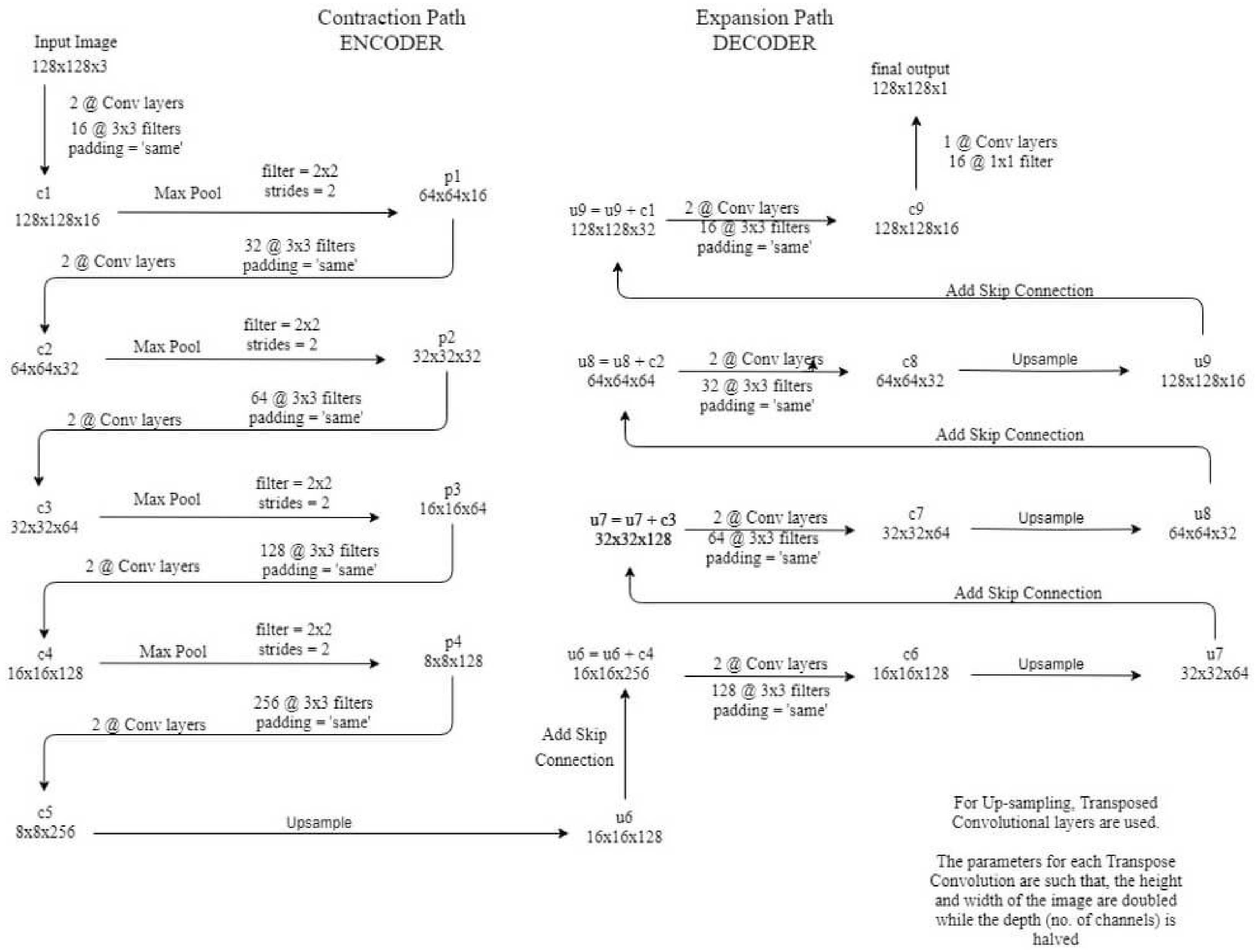


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Understanding the architecture:



Points to note:

- 2@Conv layers means that two consecutive Convolution Layers are applied
- c1, c2, …. c9 are the output tensors of Convolutional Layers
- p1, p2, p3 and p4 are the output tensors of Max Pooling Layers
- u6, u7, u8 and u9 are the output tensors of up-sampling (transposed convolutional) layers
- The left hand side is the contraction path (Encoder) where we apply regular convolutions and max pooling layers.
- In the Encoder, the size of the image gradually reduces while the depth gradually increases. Starting from 128x128x3 to 8x8x256
- This basically means the network learns the “WHAT” information in the image, however it has lost the “WHERE” information
- The right hand side is the expansion path (Decoder) where we apply transposed convolutions along with regular convolutions
- In the decoder, the size of the image gradually increases and the depth gradually decreases. Starting from 8x8x256 to 128x128x1
- Intuitively, the Decoder recovers the “WHERE” information (precise localization) by gradually applying up-sampling
- To get better precise locations, at every step of the decoder we use skip connections by concatenating the output of the transposed convolution layers with the feature maps from the Encoder at the same level:
  u6 = u6 + c4
  u7 = u7 + c3
  u8 = u8 + c2
  u9 = u9 + c1
  After every concatenation we again apply two consecutive regular convolutions so that the model can learn to assemble a more precise output
- This is what gives the architecture a symmetric U-shape, hence the name UNET
- On a high level, we have the following relationship:
  Input (128x128x1) => Encoder =>(8x8x256) => Decoder =>Ouput (128x128x1)

[**SOURCE** **(https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47)**]

ENCODER-DECODER ARCHITECTURE

An encoder is a network (FC, CNN, RNN, etc) that takes the input, and output a feature map/vector/tensor. These feature vector hold the information, the features, that represents the input. The decoder is again a network (usually the same network structure as encoder but in opposite orientation) that takes the feature vector from the encoder, and gives the best closest match to the actual input or intended output.

The encoders are trained with the decoders. There are no labels (hence unsupervised). The loss function is based on computing the delta between the actual and reconstructed input. The optimizer will try to train both encoder and decoder to lower this reconstruction loss.

Once trained, the encoder will gives feature vector for input that can be use by decoder to construct the input with the features that matter the most to make the reconstructed input recognizable as the actual input.

The same technique is being used in various different applications like in translation, generative models, etc.

It is important to know that in actual application, people do not try to reconstruct the actual input, but rather want to map/translate/associate inputs to certain outputs. For example translating french to english sentences, etc.

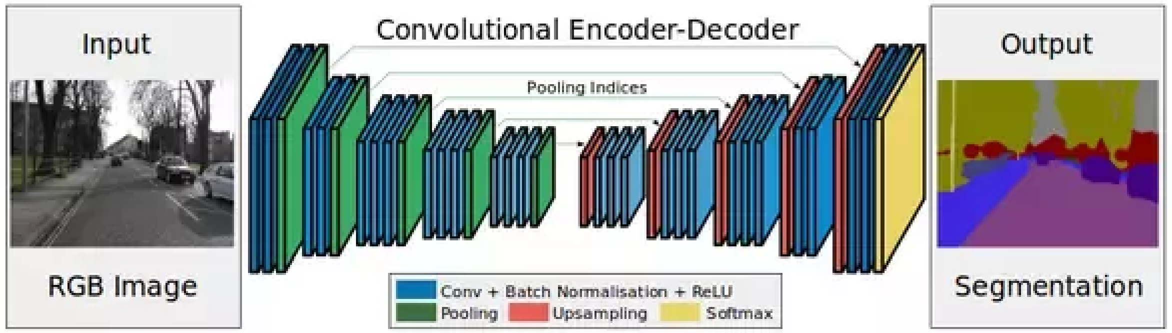
Encoder Decoder Network

Some network architectures explicitly aim to leverage this ability of neural networks to learn efficient representations. They use an encoder network to map raw inputs to feature representations, and a decoder network to take this feature representation as input, process it to make its decision, and produce an output. This is called an encoder-decoder network.

Theoretically, encoder and decoder parts can be used independent of each other. For instance, an encoder RNN can be used to encode the features of an incoming email as a “feature vector”, which is then used to predict whether the email is spam or not. However, neural encoder and decoders are often used together due to good performance on various tasks.

CNN

In a CNN, an encoder-decoder network typically looks like this (a CNN encoder and a CNN decoder):

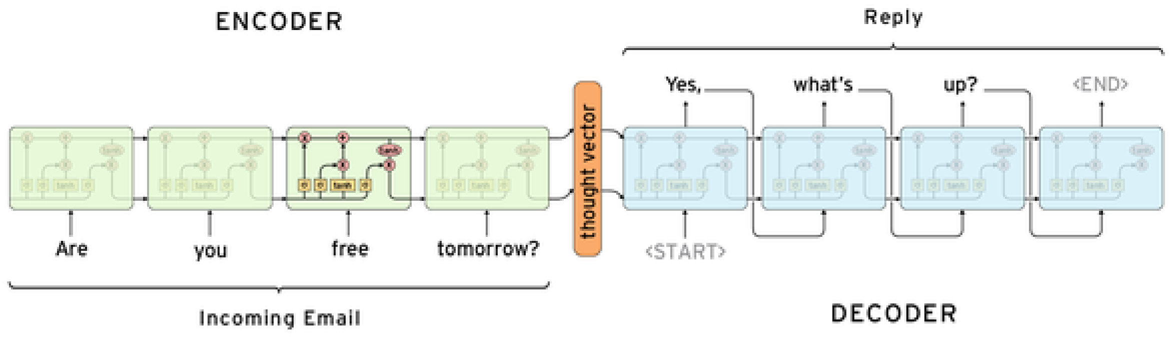


https://arxiv.org/pdf/1511.00561.pdf (https://arxiv.org/pdf/1511.00561.pdf)

This is a network to perform semantic segmentation of an image. The left half of the network maps raw image pixels to a rich representation of a collection of feature vectors. The right half of the network takes these features, produces an output and maps the output back into the “raw” format (in this case, image pixels).

RNN

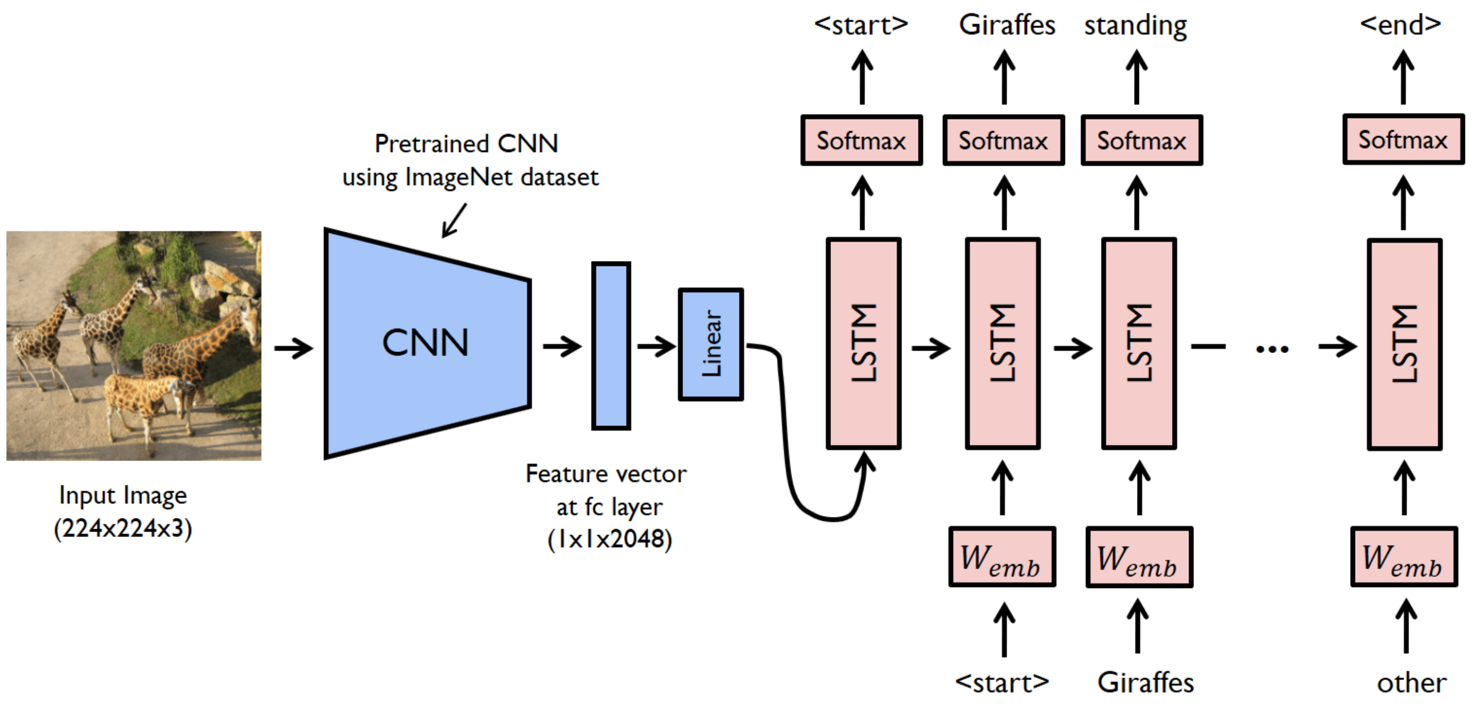
In an RNN, an encoder-decoder network typically looks like this (an RNN encoder and an RNN decoder):



This is a network to predict responses for incoming emails. The left half of the network encodes the email into a feature vector, and the right half of the network decodes the feature vector to produce word predictions.

Mixed Style

Note that it is not necessary to use just CNN or just RNN for encoder and decoder. We can mix and match and use a CNN encoder with RNN decoder, or vice versa; any combination can be used which is suitable for the given task.



Comparison with GAN

The two **architectures** (https://www.analyticsindiamag.com/google-researchers-find-out-secret-to-great-machine-translation-attention/) are used variety of tasks and both have different applications. As we mentioned above the encoder decoder architecture are mainly used to create good internal representations of the data and can be seen as great **data** (https://www.analyticsindiamag.com/the-new-nih-dataset-and-ai-may-revolutionise-lesion-detection/) compression engines. The encoder encodes the data and the decoder tries to reconstruct the data back using the internal representations and the learned weights.

Whereas **GANs (https://www.analyticsindiamag.com/is-biologically-inspired-deep-learning-scalable/)** work on a generative principle and try to learn from data distributions to use a game theory approach to build great models. Here the discriminator tries to identify fake data created the generator and hence making the players job difficult and producing better results.

SEMANTIC SEGMENTATION WITH UNET

What is segmentation?

Segmentation is key to many problems. For Image you can consider it as a problem where we need to find out a feature's "relationship" to others to tag it accurately as belonging to a specific class/label.

For language translation, we need to know the "meaning" of each word and it's context to related words to translate it accurately.

Image Classification:

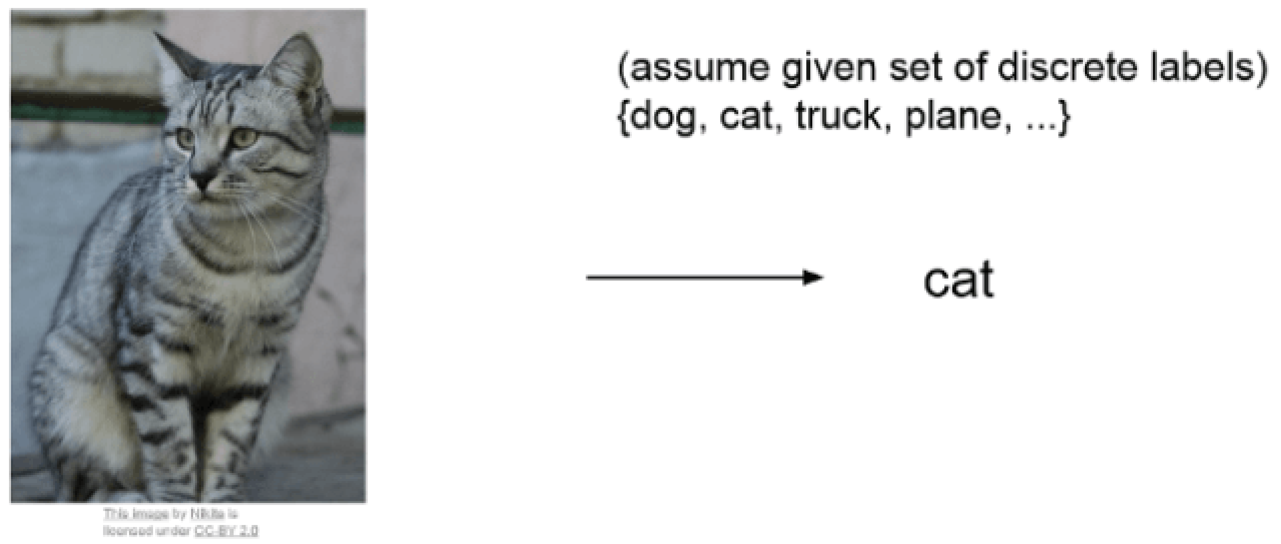
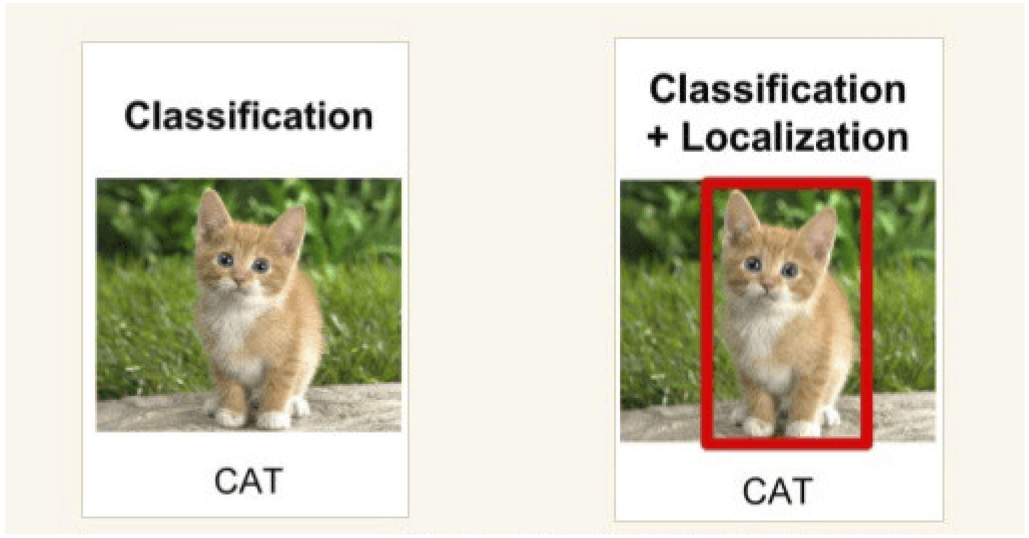
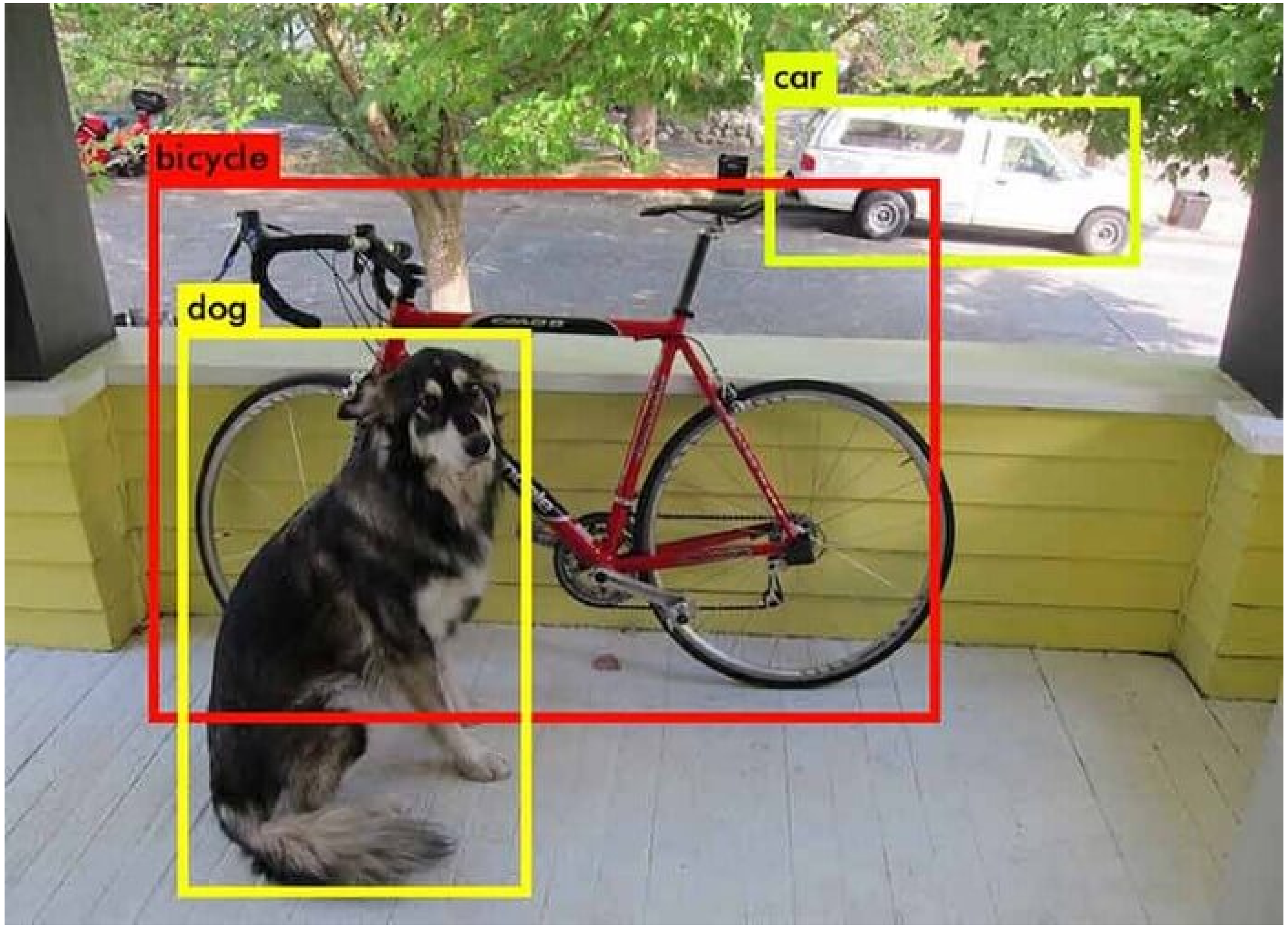

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}
cat
This image by Nikita is licensed under CC-BY 2.0

Image Classification

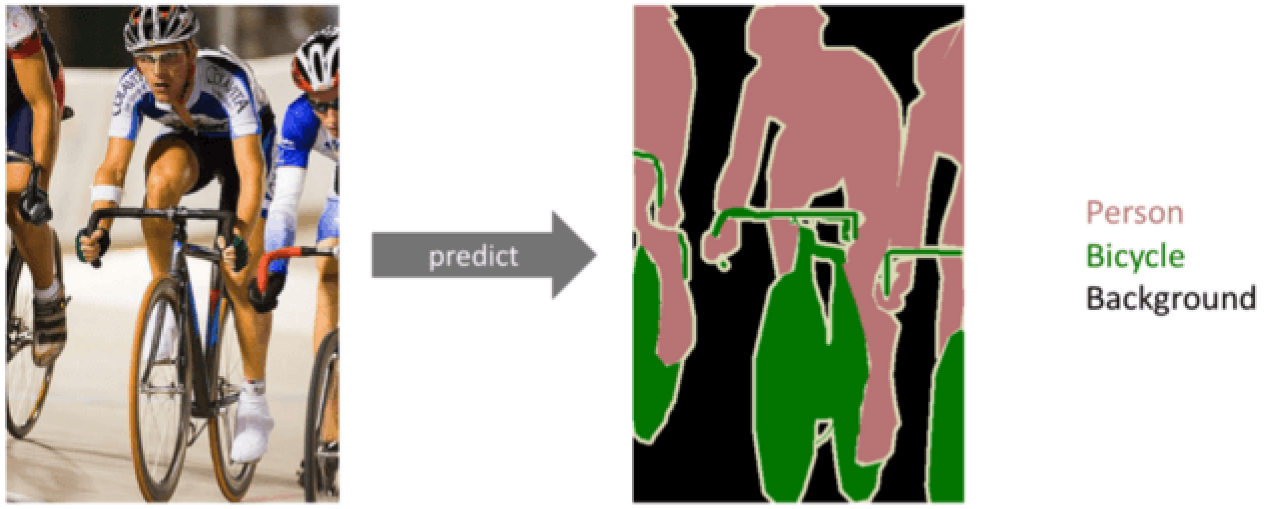
Classification with localization:



Object Detection:



Semantic Segmentation:



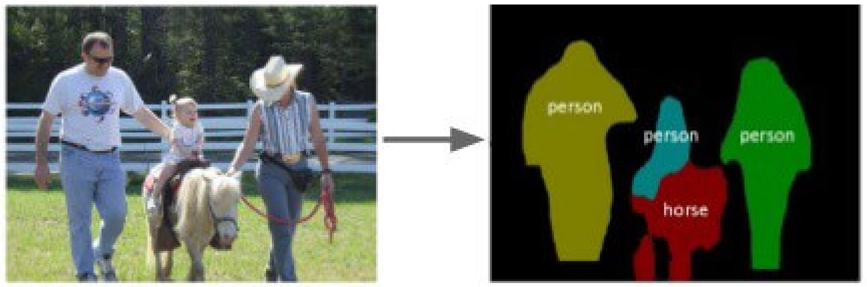
Instance Segmentation:

Instance Segmentation

Detect instances, give category, label pixels

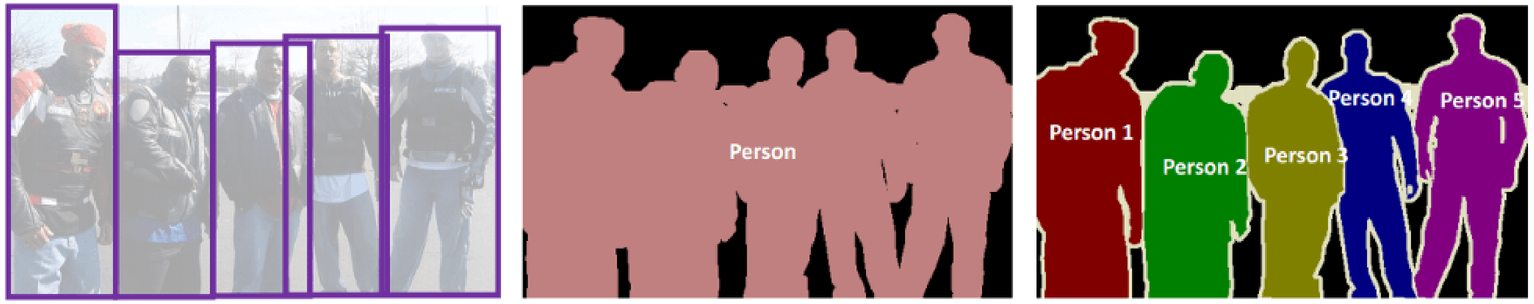
"simultaneous detection and segmentation" (SDS)

Label are class-aware and instance-aware



Slide Credit: CS231n 3

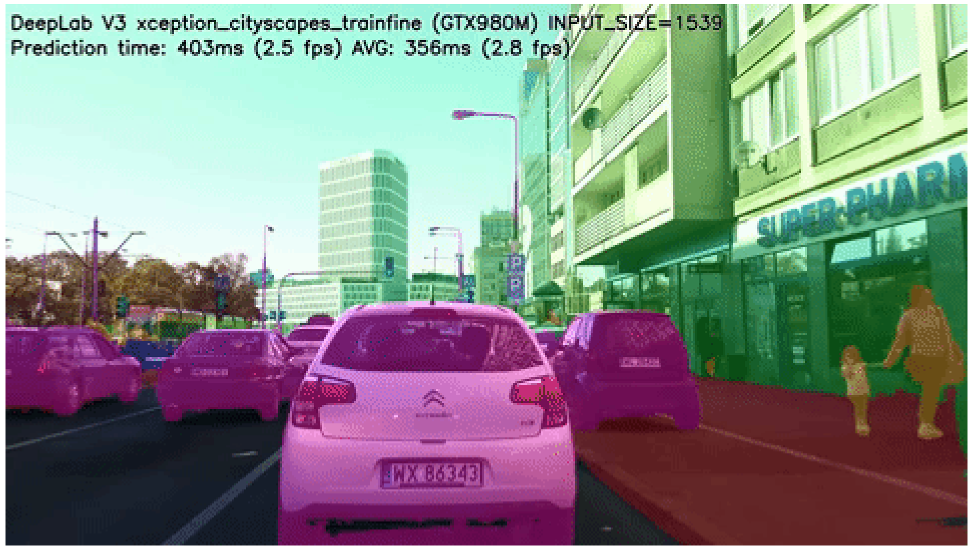
The increase in complexity and "contextual" requirement



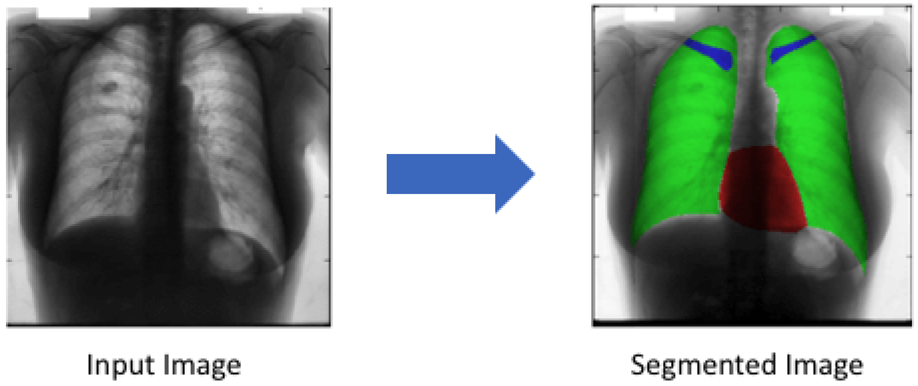
Object Detection Semantic Segmentation **Instance Segmentation**

SEMANTIC SEGMENTATION APPLICATION:

Autonomous Vehicles:


DeepLab V3 xception_cityscapes_trainfine (GTX980M) INPUT_SIZE=1539
Prediction time: 403ms (2.5 fps) AVG: 356ms (2.8 fps)
SUPER-PHARM
WX 86343

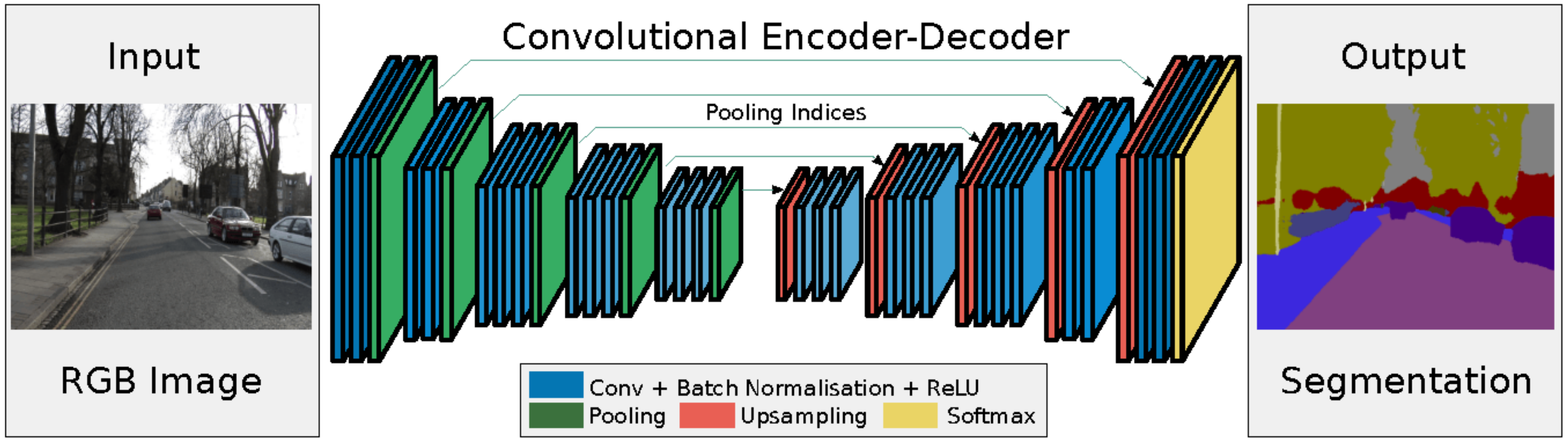
Bio-medical image diagnosis



Geo Sensing

Precision Agriculture

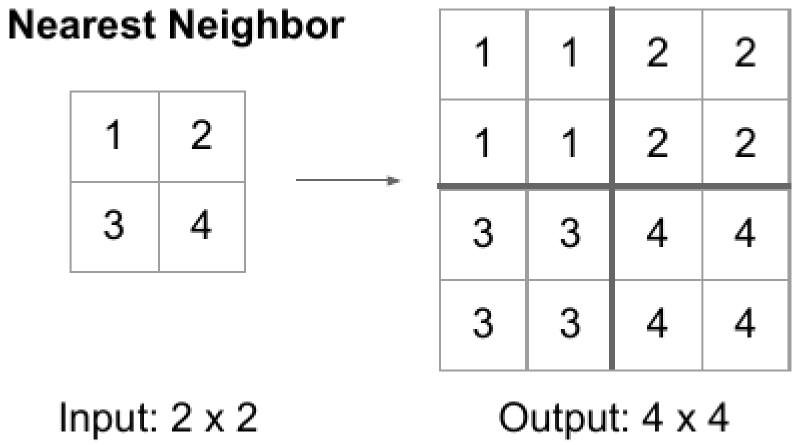
SegNET



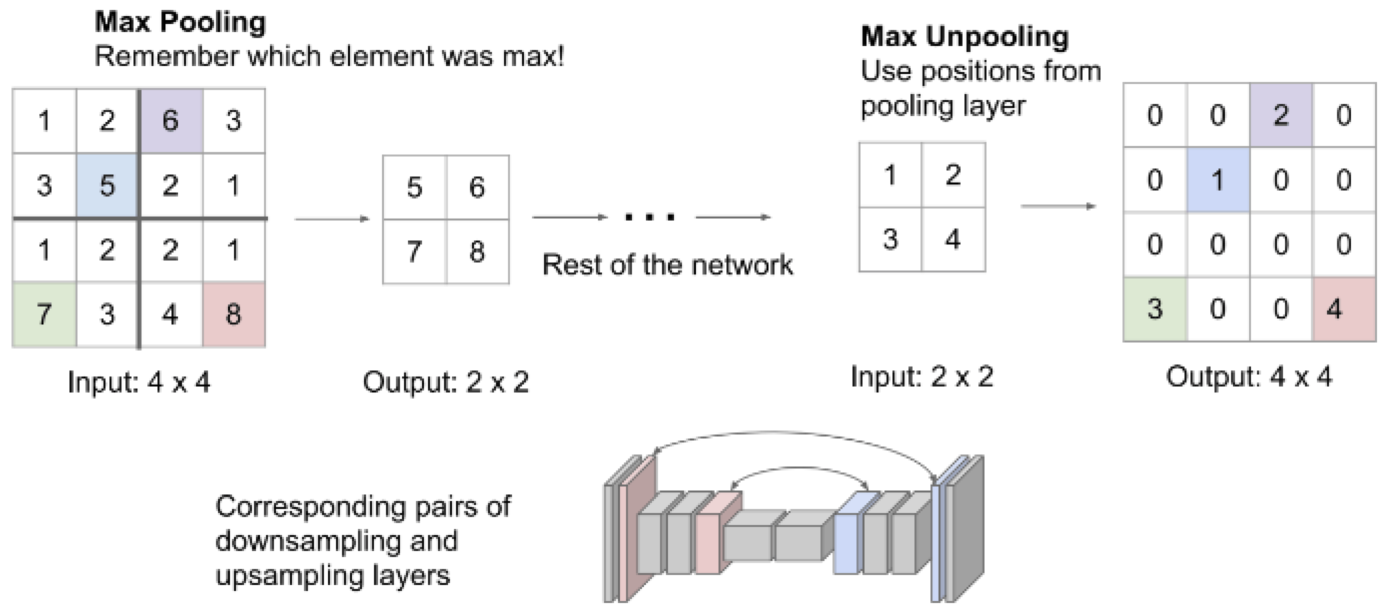
- compared to U-NET, SegNet does not use the entire feature map, but reuse the pooling indices, hence, it saves memory
- uses all of the pre-trained VGG net for it's pre-trained weights, so is faster to train.

UnPooling Operation

Normal Upsampling:



Unpooling & Switch Variables



The encoder-decoder architecture for recurrent neural networks is the standard **neural machine translation (https://machinelearningmastery.com/introduction-neural-machine-translation/)** method that rivals and in some cases outperforms classical statistical machine translation methods.

This architecture is very new, having only been pioneered in 2014, although, has been adopted as the core technology inside **Google's translate service (https://translate.google.com/)**.

Encoder-Decoder Architecture for NMT

The Encoder-Decoder architecture with recurrent neural networks has become an effective and standard approach for both neural machine translation (NMT) and sequence-to-sequence (seq2seq) prediction in general.

The key benefits of the approach are the ability to train a single end-to-end model directly on source and target sentences and the ability to handle variable length input and output sequences of text.

As evidence of the success of the method, the architecture is the core of the **Google translation service** **(https://translate.google.com/)**.

> Our model follows the common sequence-to-sequence learning framework with attention. It has three components: an encoder network, a decoder network, and an attention network.

— **Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation** **(https://arxiv.org/abs/1609.08144)**, 2016

We will take a closer look at one of the two different research projects that developed the same Encoder-Decoder architecture at the same time in 2014 and achieved results that put the spotlight on the approach. They are:

- **Sutskever NMT Model**
- Cho NMT Model

For more on the architecture, see the post:

- **Encoder-Decoder Long Short-Term Memory Networks (https://machinelearningmastery.com/encoder-decoder-long-short-term-memory-networks/)**

Sutskever NMT Model

In this section, we will look at the neural machine translation model developed by **Ilya Sutskever (http://www.cs.toronto.edu/~ilya/)**, et al. as described in their 2014 paper "**Sequence to Sequence Learning with Neural Networks (https://arxiv.org/abs/1409.3215)**". We will refer to it as the "*Sutskever NMT Model*", for lack of a better name.

This is an important paper as it was one of the first to introduce the Encoder-Decoder model for machine translation and more generally sequence-to-sequence learning.

It is an important model in the field of machine translation as it was one of the first neural machine translation systems to outperform a baseline statistical machine learning model on a large translation task.

Problem

The model was applied to English to French translation, specifically the **WMT 2014 translation task (http://www.statmt.org/wmt14/translation-task.html)**.

The translation task was processed one sentence at a time, and an end-of-sequence (<EOS>) token was added to the end of output sequences during training to signify the end of the translated sequence. This allowed the model to be capable of predicting variable length output sequences.

> Note that we require that each sentence ends with a special end-of-sentence symbol "<EOS>", which enables the model to define a distribution over sequences of all possible lengths.

The model was trained on a subset of the 12 Million sentences in the dataset, comprised of 348 Million French words and 304 Million English words. This set was chosen because it was pre-tokenized.

The source vocabulary was reduced to the 160,000 most frequent source English words and 80,000 of the most frequent target French words. All out-of-vocabulary words were replaced with the "UNK" token.

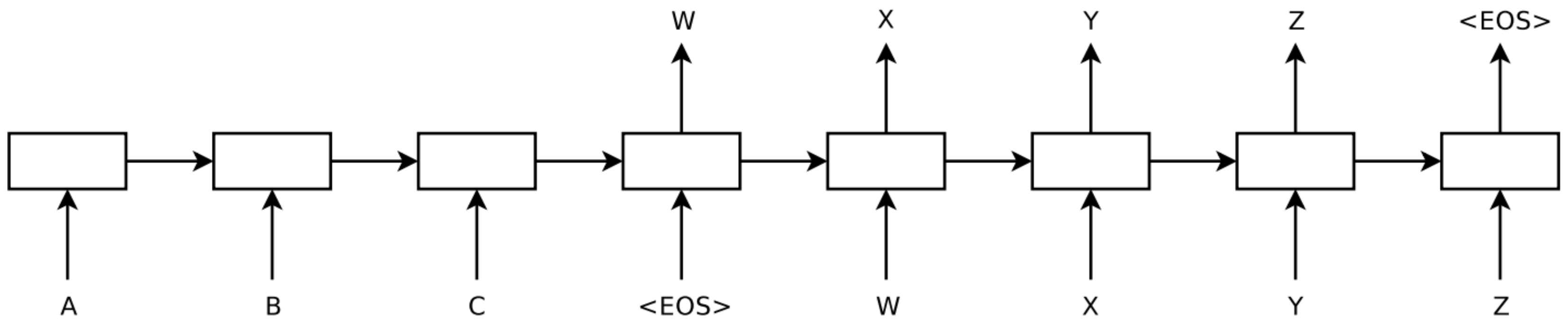
Model

An Encoder-Decoder architecture was developed where an input sequence was read in entirety and encoded to a fixed-length internal representation.

A decoder network then used this internal representation to output words until the end of sequence token was reached. LSTM networks were used for both the encoder and decoder.

> The idea is to use one LSTM to read the input sequence, one timestep at a time, to obtain large fixed-dimensional vector representation, and then to use another LSTM to extract the output sequence from that vector

The final model was an ensemble of 5 deep learning models. A left-to-right beam search was used during the inference of the translations.



Model Configuration

- Input sequences were reversed.
- A 1000-dimensional word embedding layer was used to represent the input words.
- Softmax was used on the output layer.
- The input and output models had 4 layers with 1,000 units per layer.
- The model was fit for 7.5 epochs where some learning rate decay was performed.

- A batch-size of 128 sequences was used during training.
- Gradient clipping was used during training to mitigate the chance of gradient explosions.
- Batches were comprised of sentences with roughly the same length to speed-up computation.

The model was fit on an 8-GPU machine where each layer was run on a different GPU. Training took 10 days.

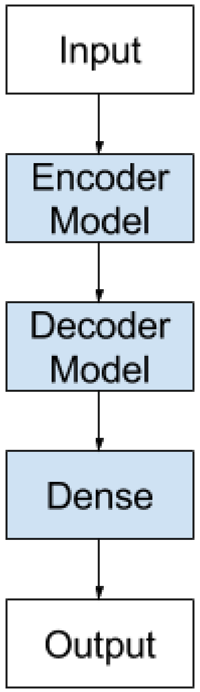
> The resulting implementation achieved a speed of 6,300 (both English and French) words per second with a minibatch size of 128. Training took about ten days with this implementation.

Result

The system achieved a BLEU score of 34.81, which is a good score compared to the baseline score developed with a statistical machine translation system of 33.30. Importantly, this is the first example of a neural machine translation system that outperformed a phrase-based statistical machine translation baseline on a large scale problem.

> … we obtained a BLEU score of 34.81 […] This is by far the best result achieved by direct translation with large neural networks. For comparison, the BLEU score of an SMT baseline on this dataset is 33.30

The final model was used t ore-score the list of **best translations** (http://www-lium.univ-lemans.fr/~schwenk/cslm_joint_paper/) and improved the score to 36.5 which brings it close to the best result at the time of 37.0.



https://machinelearningmastery.com/encoder-decoder-long-short-term-memory-networks/ (https://machinelearningmastery.com/encoder-decoder-long-short-term-memory-networks/)

Sub-Topic BLEU SCORE:

Applications of Encoder-Decoder LSTMs

The list below highlights some interesting applications of the Encoder-Decoder LSTM architecture.

- Machine Translation, e.g. English to French translation of phrases.
- Learning to Execute, e.g. calculate the outcome of small programs.
- Image Captioning, e.g. generating a text description for images.
- Conversational Modeling, e.g. generating answers to textual questions.
- Movement Classification, e.g. generating a sequence of commands from a sequence of gestures.

REFER THIS: **https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346 (https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346)**

Attention Mechanisms

A recent trend in Deep Learning are Attention Mechanisms.

In an **interview** (https://re-work.co/blog/deep-learning-ilya-sutskever-google-openai), Ilya Sutskever, now the research director of OpenAI, mentioned that Attention Mechanisms are one of the most exciting advancements, and that they are here to stay. That sounds exciting. But what are Attention Mechanisms?

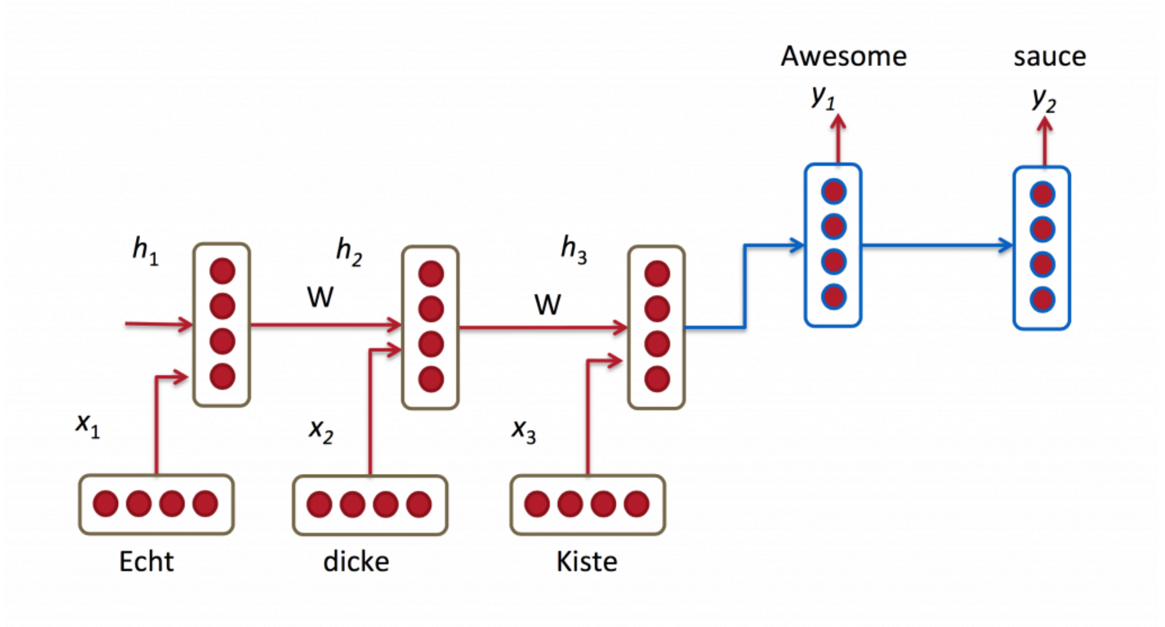
Attention Mechanisms in Neural Networks are (very) loosely based on the visual attention mechanism found in humans. Human visual attention is well-studied and while there exist different models, all of them essentially come down to being able to focus on a certain region of an image with "high resolution" while perceiving the surrounding image in "low resolution", and then adjusting the focal point over time.

Attention in Neural Networks has a long history, particularly in image recognition. Examples include **Learning to combine foveal glimpses with a third-order Boltzmann machine** (http://papers.nips.cc/paper/4089-learning-to-combine-foveal-glimpses-with-a-third-order-boltzmann-machine) or **Learning where to Attend with Deep Architectures for Image Tracking** (http://arxiv.org/abs/1109.3737). But only recently have attention mechanisms made their way into recurrent neural networks architectures that are typically used in NLP (and increasingly also in vision).

What problem does Attention solve?

To understand what attention can do for us, let's use Neural Machine Translation (NMT) as an example. Traditional Machine Learning systems typically rely on sophisticated feature engineering based on the statistical properties of text. In short, these systems are complex, and a lot of engineering goes into building them. NMT systems work a bit differently. In NMT, we map the meaning of a sentence into a fixed-length vector representation and then generate a translation based on that vector. By not relying on things like n-gram counts and instead trying to capture the higher-level meaning of a text, NMT systems generalize to new sentences better than many other approaches. Perhaps more importantly, NMT systems are much easier to build and train, and they don't require any manual feature engineering. In fact, , **a simple implementation in Tensorflow (https://www.tensorflow.org/versions/master/tutorials/seq2seq/index.html#sequence-to-sequence-models)** is no more than a few hundred lines of code.

Most NMT systems work by encoding the source sentence (e.g. a German sentence) into a vector using a Recurrent Neural Network, and then decoding an English sentence based on that vector, also using a RNN.



In the picture above, “Echt”, “Dicke” and “Kiste” words are fed into an encoder, and after a special signal (not shown) the decoder starts producing a translated sentence. The decoder keeps generating words until a special end of sentence token is produced. Here, the *h* vectors represent the internal state of the encoder.

If you look closely, you can see that the decoder is supposed to generate a translation solely based on the last hidden state (h_3 above) from the encoder. This h_3 vector must encode everything we know about the source sentence. It must fully capture its meaning. In more technical terms, that vector is a sentence embedding. In fact, if you plot the embeddings of different sentences in a low dimension space using PFA or t-SNE for dimensionality reduction, you can see (https://arxiv.org/pdf/1409.3215.pdf) that semantically similar phrases end up close to each other, and that's pretty amazing!

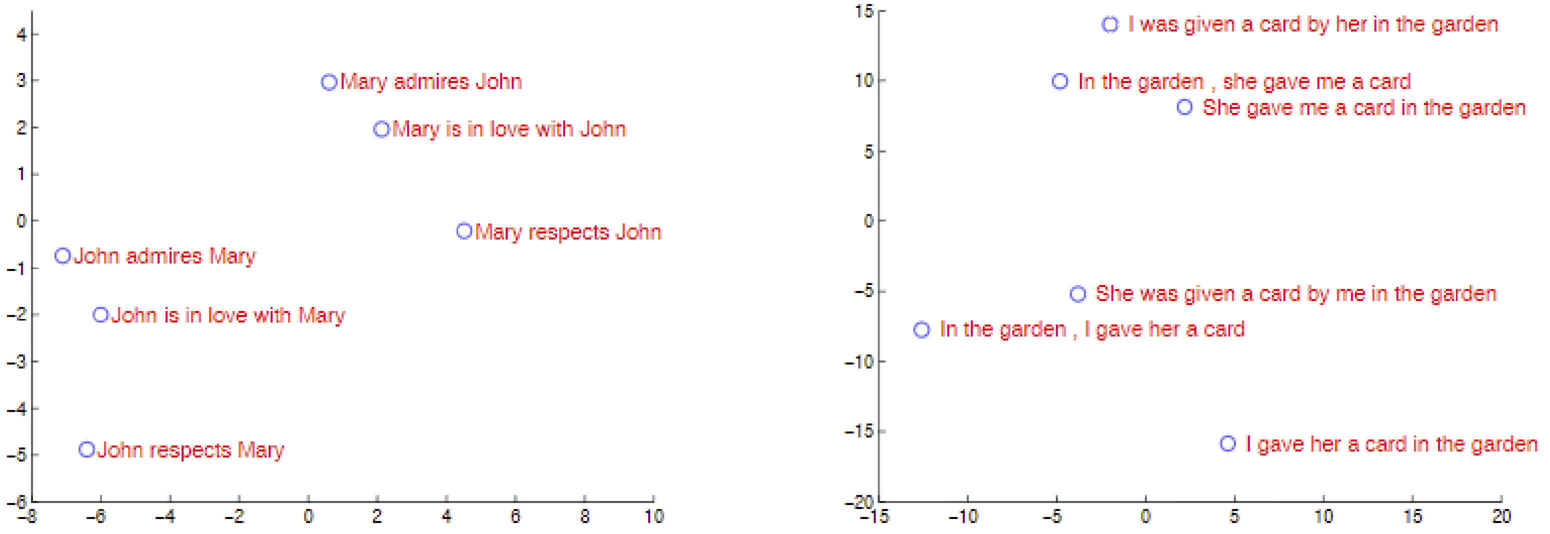
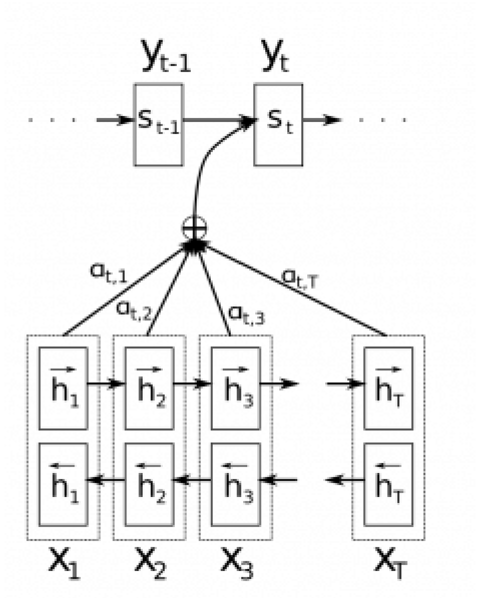


Figure 2: The figure shows a 2-dimensional PCA projection of the LSTM hidden states that are obtained after processing the phrases in the figures. The phrases are clustered by meaning, which in these examples is primarily a function of word order, which would be difficult to capture with a bag-of-words model. Notice that both clusters have similar internal structure.

Still, it seems somewhat unreasonable to assume that we can encode all information about a potentially very long sentence into a good translation based on only that. Let's say your sentence is 50 words long. The first word of the English translation is probably highly correlated with the first word of the course sentence. But that means the decode has to consider information from 50 steps ago, and that information needs to be somehow encoded in the vector. RNN are known to have problems dealing with such long-range dependencies. In theory, LSTM should be able to deal with this, but in practice long-range dependencies are still problematic. For example, researchers have found that **reversing the source sequence** (feeding it backwards into the encoder) produces significantly better results because it shortens the path from the decoder to the relevant parts of the encoder. Similarly, **feeding an input twice (https://arxiv.org/abs/1410.4615)** also seems to help a network to better memorize things.

The approach of reversing a sentence is a "hack". It makes things work better in practice, but it's not a principled approach. Most translation benchmarks are done on languages like French and German, which are quite similar to English (even Chinese word order is quite similar to English). But there are languages (like Japanese) where the last word of a sentence could be highly predicative of the first word in an English translation. In that case, reversing the input would make things worse. So what's an alternative? **Attention Mechanism.**

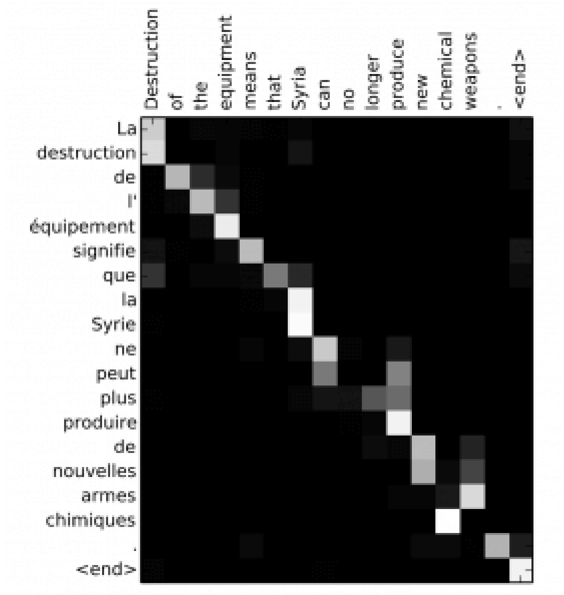
With an attention mechanism we no longer try encode the full source sentence into a fixed-length vector. Rather, we allow the decoder to "attend" to different parts of the source sentence at each step of the output generation. Importantly, we let the model **learn** what to attend to based on the input sentence and what it has produced so far. So, in langages tht are pretty well aligned (like English and German) that decoder would probably choose to attend to things seuentially. Attending to the first word when producing the first English word, and so on. That's what was done in **Neural Machine Translation by Jointly Learning to Align and Translate (http://arxiv.org/abs/1409.0473)** and look as follows:


y_{t-1}
y_t
s_{t-1}
s_t
$a_{t,1}$
$a_{t,2}$
$a_{t,3}$
$a_{t,T}$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
X_1 X_2 X_3 X_T

Here, the y's are the translated words produced by the decoder, and the x's are our course sentence words. The above illustration uses a bidirectional RNN.

Each decoder output word y_t now depends on a **weighted combination of all the input states**, not just last state. The a's are weights that define in how much of each input state should be considered for each output. So if $a_{3,2}$ is a large number, this would mean that the decoder pays a lot of attention to the second state in the source sentence while producing the third word of the target sentence. The a's are typically normalized to sum to 1 (o they are distributed over the input states).

A big advantage of attention is that it gives us the ability to interpret and visualize what the model is doing. For example, by visualizing the attention weight matrix **a** when a sentence is translated, we can understand how the model is translating:



Here we see that while translating from French to English, the network attends sequentially to two words at time when producing an output, as in translation "la Syrie" to "Syria" for example.

The cost of Attention

If we look a bit more closely at the equation for attention we can see that attention comes at a cost. We need to calculate an attention value for each combination of input and output word. If you have a 50-word input sequence and generate a 50-word output sequence that would be 2500 attention values. That's not too bad, but if you do character-level computations, and deal with sequences consisting of hundreds of tokens the above attention mechanisms can become prohibitively expensive.

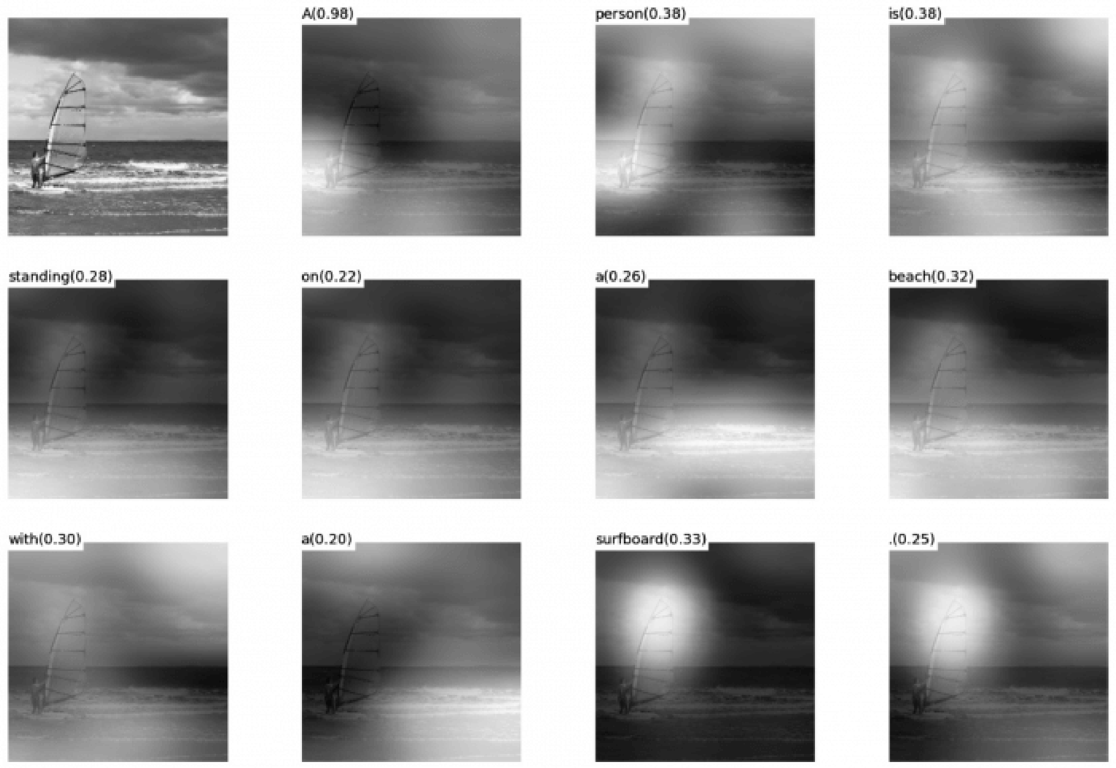
And it is quite counter-intuitive as well. Human attention is something that's supposed to **save** computation resources. By focusing on one thing, we can neglect many other things. But that's not really what we're doing in the above model. We're essentially looking at everything in detail before deciding what to focus on. Intuitively that's equivalent outputting a translated word, and then going back through all of your internal memory of the text in order to decide which word to produce next. That seems like a waste, and not at all what humans are doing. In fact, it's more akin to memory access, but that hasn't stopped attention mechanisms from becoming quite popular and performing well on many tasks.

An alternative approach to attention is to use Reinforcement Learning to predict an approximate location to focus on. That sounds a lot more like human attention, and that’s what’s done in **Recurrent Models of Visual Attention** (http://arxiv.org/abs/1406.6247).

Attention beyond Machine Translation

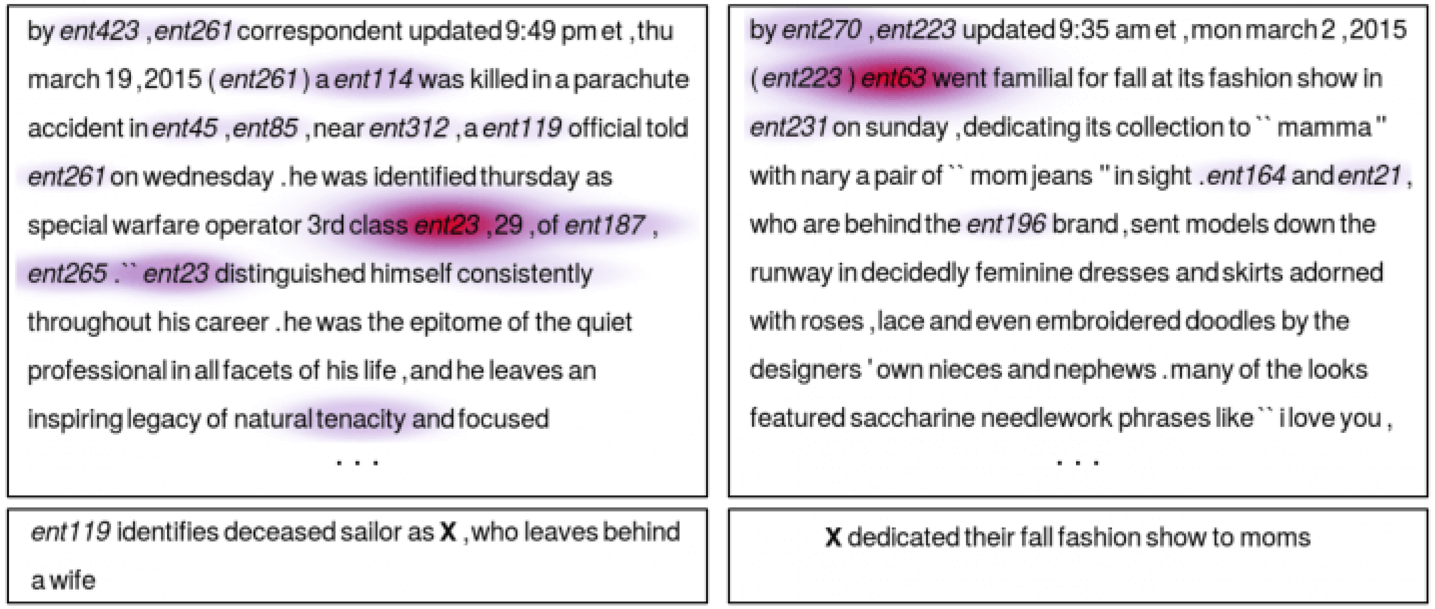
So far we've looked at attention applied to Machine Translation. But the same attention mechanism from above can be applied to any recurrent model. So let's look at a few more examples.

In **Show, Attend and Tell** (http://arxiv.org/abs/1502.03044) the authors apply attention mechanisms to the problem of generating image descriptions. They use Convolutional Neural Network to "encode" the image, and a Recurrent Neural Network with attention mechanism to generate a description. By visualizing the attention weights (just like in the translation example), we interpret what the model is looking at while generating a word:



(b) A person is standing on a beach with a surfboard.

In [Teaching Machines to Read and Comprehend](http://arxiv.org/abs/1506.03340) (http://arxiv.org/abs/1506.03340), the authors use a RNN to read a text, read a (synthetically generated) question, and then produce an answer. By visualizing the attention matrix we can see where the networks "looks" while it tries to find the answer to the question:



by *ent423* ,*ent261* correspondent updated 9:49 pm et ,thu march 19 ,2015 (*ent261*) a *ent114* was killed in a parachute accident in *ent45* ,*ent85* ,near *ent312* ,a *ent119* official told *ent261* on wednesday .he was identified thursday as special warfare operator 3rd class *ent23* ,29 ,of *ent187* , *ent265* .`` *ent23* distinguished himself consistently throughout his career .he was the epitome of the quiet professional in all facets of his life ,and he leaves an inspiring legacy of natural tenacity and focused

. . .

ent119 identifies deceased sailor as **X** ,who leaves behind a wife

by *ent270* ,*ent223* updated 9:35 am et ,mon march 2 ,2015 (*ent223*) *ent63* went familial for fall at its fashion show in *ent231* on sunday ,dedicating its collection to `` mamma " with nary a pair of `` mom jeans " in sight .*ent164* and *ent21* , who are behind the *ent196* brand ,sent models down the runway in decidedly feminine dresses and skirts adorned with roses ,lace and even embroidered doodles by the designers ' own nieces and nephews .many of the looks featured saccharine needlework phrases like `` i love you ,

. . .

X dedicated their fall fashion show to moms

Attention = (Fuzzy) Memory?

The basic problem that the attention mechanism solves is that it allows the network to refer back to the input sequence, instead of forcing it to encode all information into one fixed-length vector. With attention mechanisms, we are there, but not there yet. [SOURCE (http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/?source=post_page-----------------------------)]

SESSION VIDEO:

EIP Phase 2 Session 4