Understanding the Use of Deep Learning for Generating Captions to Describe Images

Advait Savant
Sardar Patel Institute of Technology, India

Abstract

Machine Learning is that field of Computer Science wherein we create statistical and computational models of systems and tune those models using the data at hand which acts as an experience based on which the system improves its performance at a given task with respect to some performance measure. A large amount of data that is available via computer networks, distributed systems and increase in the computing power of devices have led to a boom in the applications of machine learning today. Deep learning is that subfield of machine learning wherein our focus is on creating representations for the system through artificial neural networks. Deep learning and the use of convolutional neural networks have given us a significant performance gain in Computer Vision tasks and has been successfully used for object tracking, detection etc. Deep learning has also been used for Natural Language Processing problems like machine translation, named entity recognition, parts of speech tagging and has proven to be very useful there. Deep learning applications in Computer Vision and NLP are active areas of research. Another important avenue is the performance of tasks, which require both visual perception and linguistic abilities such as automated caption generation for a given image, which is the focus of this paper. I attempt to demonstrate and explain how deep learning can help us in this task, which is an exemplar of AI technology today.

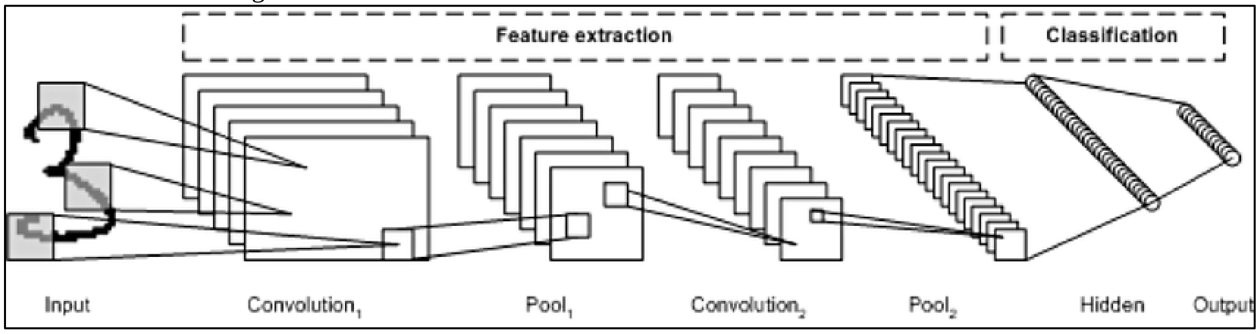
Keywords: Machine Learning, AI technology, Convolutional Neural Networks

I. Introduction

Generating a natural language description of an image is a difficult task. We need to be able to recognize the objects in an image, understand the depicted scenario and how those objects interact with and relate to each other. We also require a language model for developing a coherent sentence. A statistical language model is a probabilistic model which gives us the probability distribution over a sequence of words. These models can help us understand context and predict the next word given a sequence of words. As we shall see, recurrent neural networks will be used for language models and we will understand the use of convolutional neural networks to create useful representations of images and extract visual features. We will use transfer learning. For language models, we can learn from the techniques used in machine translation [2]. Encoder-Decoder architectures are used for sequence to sequence mappings. An encoder RNN reads the sentence as a sequence of words and transforms it into a fixed length vector which acts as a representation. A Decoder RNN is tuned on that fixed length vector to produce a sequence of words which is the translation. For caption generation, we have the merge architecture for wherein there is an RNN which acts as an encoder of prefixes of word embeddings and the output from the RNN is merged with visual features during a later stage in the model after which there is a feed-forward layer to do the predictions [1]. We can also think of the RNN as a generator which is trained to produce the next word in a sentence given the prefix. In the inject architecture, we feed in both word embeddings and image features into the RNN such that it learns a linguistic representation conditioned on images [1]. In the merge architecture, we make a separation between linguistic representations and visual features combining them together in a later stage. In what follows, we will review CNNs, RNNs, Word embeddings, representation learning and understand how these are applied to our problem. We will consider the different architectures and evaluate an implementation.

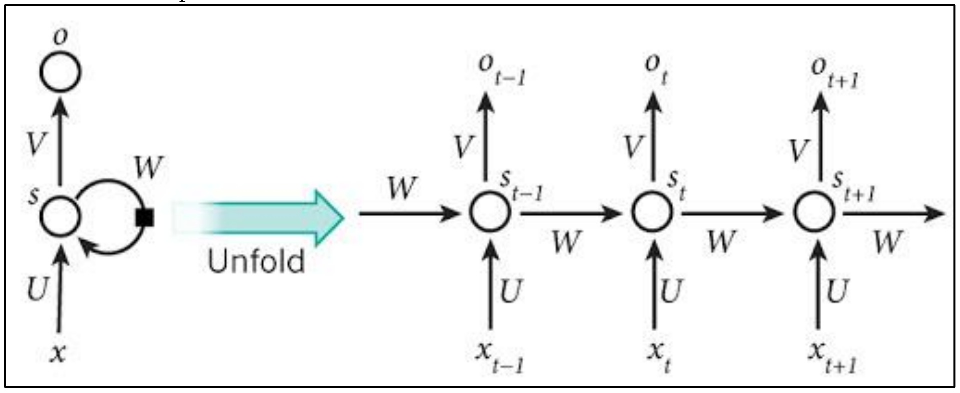
II. Convolutional neural networks:

Convolutional neural networks are neural networks made for data which has a spatial topology like images. It takes advantage of the inherent structure of the data in this sense. Let us say we have an image which has 3 RGB channels(C=3) and each channel having height H and width W. The elements of the input image tensor give us the intensity of the corresponding pixels. This is the input to the first layer. Further, we express the input in each layer as having dimensions H*W*C. We are dealing with tensors. Back to the first layer, we have multiple filters which perform the convolution operation on the input image and produce a feature map. To elaborate, each filter has a height and a width F which is the filter size and a depth which is the same as the depth of the input image. In the convolution operation, we superimpose our filter (3D Tensor) on a starting positon of the input image (3D Tensor) such that there is an element to element correspondence between the filter parameters and positions in our input image. In effect, we have selected a subset of the input image. We then perform an element wise multiplication of those values in the two tensors and place the sum on the first position of our feature map corresponding to that filter. This is done for all coordinate positions across the height and width of our image with that same filter and we obtain a 2D feature map. (We have to take care of the padding, strides chosen which will affect the size of the 2D feature map obtained [4]). This was for one filter, we repeat the same procedure for other filters in the convolutional layers and add each feature map obtained to the depth of the output of that layer (we stack the obtained feature maps for each filter and form another 3D tensor). We pass the obtained feature maps through the RELU activation function. We can add a pooling layer which replaces the output at a location with a summary statistic of the nearby locations. This makes the representation invariant to small translations of the input. We can then have another layer of convolution with another set of filters whose depth is the same as the number of feature maps coming from the previous layer. The same process can continue. Once these convolution, activation and pooling operations are done, we flatten the result obtained in the later layers and add a dense fully connected layer or layers which can be used to produce the output. The point is that in the training process, we can learn meaningful filters across the layers which give us useful spatial feature representations. Training is done using backpropagation with gradient descent. CNNs using the convolution operations give us sparse interactions (gives us faster computation), parameter sharing (a fewer number of parameters) and translation equivariance [4]. They are proven to be better suited for images/spatial data as compared to ANNs and as we shall see, they will be used as feature extractors for our images. We conclude our section on CNNs.



III. Recurrent neural networks

RNNs are a class of neural networks designed for sequence modelling and sequence labelling. We are taking inspiration from dynamical systems, the state space models of the systems, how they evolve over time and have feedback mechanisms. Time (a variable with respect to which our system is changing) comes in as an important parameter in the learning process. Consider that the input space consists of N sequences. Each sequence has T time steps and each time step has a feature vector of dimensionality D. We are thus dealing with N * T * D data. Successive feature vectors across time-steps in a sequence are correlated have dependencies with each other and cannot be treated as independent data points. This is the inherent structure of the data which the RNN capitalizes on. Suppose we want to predict the price of a certain stock (y(t)) based on the current financial parameters (x(t)). The problem with using a feed-forward neural network for (x,y) pairs is that we cannot capture the dependencies of the current y values on previous x values. Here is what works and what we can do so that our model considers an arbitrary window of time in the past: If we focus on one time-step and take the activations of the hidden layer at that time-step and feed them back as inputs to the hidden layer at the next time-step along with the corresponding input of the next time-step, we get a recurrence relation where the feedback mechanism between the hidden layers at consecutive time-steps is able to capture the information of the history of the input. At any time-step, we have inputs from the feedback to the hidden layer which represent earlier time-steps. Since the hidden state at one time-step depends not only on the current input but also on the hidden state at the previous time-step, summarizing information from earlier values of x is possible. This feedback mechanism leads to cycles in the computational graph which shows us the computations performed by an RNN. We can think of an unfolding of the computational graph as depicted in the picture in order to simplify our calculations in the forward pass and the backward pass. We share parameters across time-steps.



The vector s represents the state of the system which is the hidden layer post-activation. We can write this as a dynamical system-

$$s_t = f(s_{t-1}, x_t ; w)$$

x represents the input vector, U represents the weights between the input layer and the hidden layer, W represents the hidden to hidden recurrence relations, a represents the hidden layer pre-activations at each time-step and o represents the output vector pre-activations at each time-step. y represents the output vector. Consider the use of the softmax function for the output (this is the case if our target is a probability distribution) and the tanh activation function at the hidden layer (adds non linearity varying between -1 and 1). For the forward pass, we know that the hidden layer at each time-step depends on the hidden layer at the previous time-step and the input at the time-step. Let b be the bias vector (one bias value for each hidden state) and c be the bias vector for the output. We have-

$$a_t = Ux_t + Ws_{t-1} + b$$
$$s_t = \tanh(a_t)$$
$$o_t = Vs_t + c$$
$$y_t = \mathrm{softmax}(o_t)$$

We start with s_0 and x_1.

Consider the following - The loss will be the sum of losses at each time-step. The prediction at each time-step is dependent on the history of the inputs. Maximizing the likelihood is equivalent to minimizing the negative log likelihood as the logarithm is a monotonically increasing function. Since our output is based on softmax we will be using a categorical cross entropy loss. We thus have the following:

$$L = \sum_{t=1}^{T} L_t$$

$$L = -\sum_{t=1}^{T} \log(p(y_t/\{x_1, \ldots x_t\}))$$

In the backward pass, we apply the back-propagation algorithm to the unrolled computational graph and move from right to left. We start with the end of the sequence and work our way backwards as we calculate the gradients of the loss with respect to the hidden states at different time-steps and subsequently we calculate the gradients of the loss with respect to the weights which are shared across time-steps (We sum them up as we use the chain rule). This is known as back-propagation through time[4]. In practice we do a vectorized implementation.

We have the following gradients for the last time-step-

$$\nabla_{s_T} L = V^T \nabla_{o_T} L$$

For a time-step t<T:

$$\nabla_{s_t} L = (\partial s_{t+1} | \partial s_t)^T \nabla_{s_{t+1}} L + (\partial o_t | \partial s_t)^T \nabla_{o_t} L$$
$$\nabla_{s_t} L = W^T \nabla_{s_{t+1}} L * \mathrm{diag}\big(1 - (s_{t+1})\big)^2 + V^T \nabla_{o_t} L$$

From here, as we said before, we can calculate the gradients with respect to the weights for each time step using the chain rule of multivariate calculus and then sum them up across time-steps to find the composite gradient. We can then use a weight update algorithm like stochastic gradient descent or its improvisations like adam, adagrad in order to update our parameters and then find the parameters which optimally fit the training set after training across a certain number of epochs. The nature of the feedback mechanism in an RNN may vary, like we can have output to hidden layer connections in the unrolled computational graph. We conclude our section on RNNs.

IV. WORD EMBEDDING

We want to create a vectorial representation of words which are categorical objects such that the representation will be useful as we feed it to the RNN based language model. We have a vocabulary of size V (V is very large) and we would like to do something better than one hot encoding in order to capture the interdependencies and relationships between words in D dimensional vector space(D is smaller than V). Deep learning in NLP started with training for bigrams, a pair of consecutive words which appear in our sentences. We could make a model like the following:

$$p\left({}^{y}/_{x}\right) = \mathrm{softmax}(Wx)$$

For each pair of consecutive words (x,y) in our sentences, we can one hot encode them, feed them as outputs and inputs respectively in our model and train the model with the softmax activation function which is used for predicting a probability distribution. This will help us obtain a useful representation in the weights of W. Note that the size of W is V*V.

In the Skipgram model, we predict the context words given an input word. For example, in the sentence ‘A dog ran in the garden’ the word ‘ran’ will have ‘dog’, ’the’, ’garden’ in its context.

We have two matrices: U of size V*D and V of size D*V.

We have the following neural network:

$$y = \mathrm{softmax}(U(Vx))$$

Notice that the vector x is one hot encoded column vector with 1 in a particular row (for each word) and zeros everywhere else, if we consider the column picture of matrix multiplication [5], we see that one column of V (let it be v) will go in forward to

be multiplied by U(This corresponds to the input word). And for every output word y, one row of U (let it be u) will be multiplied by that column of V. We thus have a dot product $v^T u$ for each input word, context word pair.
In the technique of negative sampling, we model the problem as multiple binary classifications (correct word, context word pairs or not). We use the sigmoid function to mark the probability that the concerned word pair is in the set of the correct word, context word pairs.

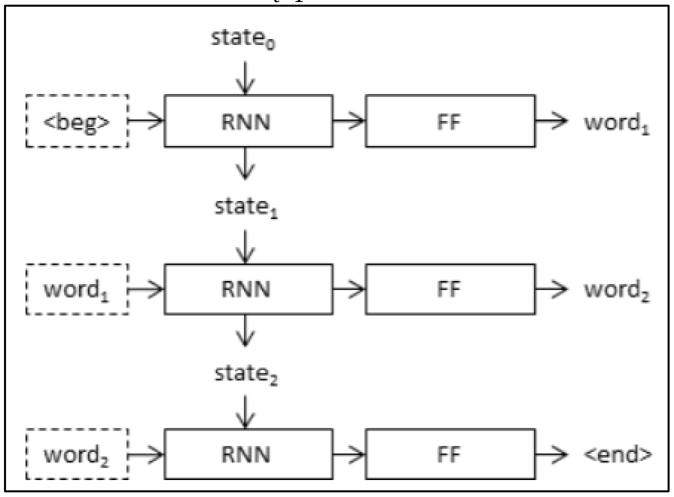
$$p\left(z = 1 / w, c\right) = \sigma(\, v_w{}^T u_c)$$

We have a set of correct word, context word pairs and incorrect word, context word pairs, we use maximize the probability that z equals 1 for correct pairs (product of probabilities for all pairs) and also maximize the probability that z equals 0 for the incorrect pairs. Taking the negative logarithm here we get a cost function for minimization equivalent to that of the logistic regression problem. We than further compute the gradients of the loss with respect to our U,V and update these matrices using gradient descent. The Skipgram model used along with negative sampling gives us the standard Word2Vec representation. Each column of V is correlated with the one hot vector corresponding to each word in vocabulary and can be used as a representation of that word.

V. Models and Architectures

We can treat a sentence as a sequence of words with the words corresponding to time-steps, take the word embedding of each word and consider it as a feature vector. We can feed the embeddings into the RNN and use it as a language model. We start with a model wherein we generate the sequence of words using a RNN conditioned on an image, the RNN encodes the prefix and generates a probability distribution for the next word in the sentence [2]. There is a token which represents the start of a sentence and also one token for the end of a sentence. Given a prefix, we predict the next word. We maximize the log of the probability that S (N terms) is the sequence, given our image (for our training data) with respect to our RNN model parameters.

$$\log p(S|I) = \sum_{t=1}^{N} \log p(S_t|I, S_1, \ldots, S_{t-1})$$



An important question arises about how we introduce image features into our model. We use the VGG Convnet(or any other benchmark model like ResNet etc) which is trained on a large number of images and has many layers in its architecture. We see here that we can apply the concept of transfer learning. Basically, transfer learning allows us to utilize the knowledge acquired in performing one task in order to better perform another task. The nature of image data is such that the VGG must have captured useful patterns and abstractions about the wide variety of images it has seen across various layers in a representational hierarchy as is with a CNN [7]. If we feed an image into the VGG, we see that the second last layer will give us a representation which can lead us to useful regions in the parameter space and we can forward the output of that second last layer into our RNN model and interpret it as the VGG extracting the features from our image.

In the init-inject architecture, we feed the visual features as a starting state of the RNN, this requires the RNN hidden state vector and the image feature vector be of the same length. After this we can go about feeding in the words and training the RNN language model. We have seen earlier how an RNN is trained. In the pre-inject architecture, the image feature vector is used as first input to the RNN during training and later followed by the word embeddings. In the par-inject architecture, we feed the image representation to the RNN at each time-step along with the word embeddings, we either feed them in parallel as two separate inputs or combine the two vectors [3].

In the merge architecture, we feed the visual feature inputs later into a feed-forward network which also receives an encoded representation of the prefix from our RNN and the feed-forward network is used to predict a probability distribution for the next word. The RNN here primarily acts as an encoder rather than a generator [1].

VI. IMPLEMENTATION AND ANALYSIS

MSCOCO, Flickr8k and Flickr30k are the main datasets available for this problem. Using the MSCOCO datasets, the use of RNNs and VGG with a merge architecture gives us decent and diverse results. We can evaluate using the BLEU score [6]. The literature shows that LSTMs which are improvisations of RNNs solving the problem of long term dependencies and the problem of exploding and vanishing gradients perform better [3]. Literature suggests that merge outperforms inject by a narrow margin across datasets, layers and vocabularies [1]. The difference might be in the handling of variations by each architecture. For example, in the par-inject architecture, we combine the image with the words in order to create new composite vectors which increases the vocabulary size for the RNN. The encoding task in an inject architecture becomes more complex as we include image feature vectors into the RNN. In the merge architecture, the RNN is for linguistic representations only and acts as an encoder for the prefix (the output of which is passed ahead to the feedforward layer). We have thus seen the architectures and methods that can be used to implement our problem.

VII. CONCLUSION

We have seen and understood the successful use of deep learning for generating image captions. We have been through convolutional neural networks, recurrent neural networks, word embeddings, transfer learning and language models in order to grasp the technicalities of this task. We reviewed the inject and merge architectures used and analyzed the two of them. We have thus seen the theoretical and practical aspects of this task. Deep learning is proving to be a promising and fast evolving field in computer science and artificial intelligence and it is the ability of deep neural networks to solve problems like these which makes us optimistic about their potential.

REFERENCES

[1] What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator? Marc Tanti et al, Institute of Linguistics and Language Technology, University of Malta 2017.
[2] Show and Tell: A Neural Image Caption Generator, Oriol Vinyals et al, Google 2015.
[3] Where to put the Image in an Image Caption Generator, Marc Tanti Albert Gatt , Institute of Linguistics and Language Technology, University of Malta 2018.
[4] Goodfellow, Bengio and Courville, Deep Learning, MIT press,2016.
[5] Gilbert Strang,Linear Algebra and its applications, MIT press,2006.
[6] BLEU: a Method for Automatic Evaluation of Machine Translation Kishore Papineni et al, IBM, T J Watson research centre, 2002.
[7] From Generic to Specific Deep Representations for Visual Recognition. Azizpour, Hossein & Razavian, Ali & Sullivan, Josephine & Maki, Atsuto & Carlsson, Stefan, Royal Institute of Technology, Stockholm 2014.