Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Communication Theory Group
Prof. Dr. H. Bölcskei

Student Project in Information Technology
and Electrical Engineering

Fall Semester 2018

Sami Hamdan

Grounded language learning of visual-lexical color descriptions

Supervisors: Dmytro Perekrestenko

December 2018

Abstract

We design a generative neural network to create descriptions from colors and vice versa. We choose an autoencoder model which latent space is three dimension RGB space. The network learns how to encode descriptions to gaussian clusters in the RGB space and decode them after sampling these clusters. Our loss function penalizes the reconstruction error and the distance between encoded clusters and sample distributions. Once the network is trained it can be used to generate colors and descriptions which are both diverse and accurate.

Acknowledgments

I would like to thank Dmytro Perekrestenko for his help in this project.

Contents

Abstract **i**

Acknowledgments **ii**

Notation **iv**

1 Introduction **1**

1.1 Grounded language learning 1

1.2 Related work 1

2 Dataset **3**

3 Methodology **6**

3.1 Objective function 6

3.2 Encoding 7

3.3 Decoding 9

4 Training **12**

5 Experiments **14**

5.1 Generating colors 14

5.2 Generating sequences 17

6 Conclusion **20**

Notation

We denote vectors as $\vec{x}$, matrices as A, its transpose A^T and its element on the i-th column and j-th row as $A_{i,j}$. The identity matrix is written $\mathbb{I}$.
We write $\log(.)$ the napierian logarithm. The determinant operator of the matrix A is written $|A|$ and the Kullback-Leibler divergence between two discrete distributions p and q is denoted as $\mathrm{D}_{KL}(p||q)$ and computed so:

$$\mathrm{D}_{KL}(p||q) = \sum_i p(i) \log \frac{p(i)}{q(i)}$$

The multivariate gaussian distribution with mean $\vec{\mu}$ and covariance matrix Σ is written $\mathcal{N}(\vec{\mu}, \Sigma)$

Chapter 1

Introduction

1.1 Grounded language learning

Grounded language learning (GLL) is a technique for language acquisition that uses a multimodal set of inputs rather than just sets of words or symbols, e.g. it uses a combination of words and related sounds or visuals. Due to the similarity of GLL with the way humans are exposed to language, studying GLL can potentially yield insights on how language is comprehended by humans. In our task, we would like to use both visual (colors) and textual (sequences of words) elements to train a generative model that is capable of describing colors and relating colors to descriptions. However this task can be quite challenging as it is based on a dataset of labelled colors that can be quite vague and have high variance. We will in a further chapter introduce our dataset and what we mean concerning the variance.

1.2 Related work

Lazaridou et al. [1] highlight that many words are learned by humans in a context which can be then useful in other different instances. They specifically mention how text co-occurence statistics can suffice to induce meaning of unseen words. This is the idea behind the GloVe [2] word embedding. Word embedding is a technique to map words to vectors which can be than used as multidimensional inputs. The mapping relies on a co-occurence matrix and enables vector used in the same context to be close in the embedding space. Monroe et al. [3] propose a way of both generating decription from colors and vice versa. The first use the GloVe embedding described that they then input to a recurrent neural network architecture in combination with fourier transform of colors. The fourier transform components are selected because most regions of the colors space are extreme along axis of the discrete fourier transform. Their method is able to learn modifiers like *dark, dull*, etc. but relies on preprocessing that is specific to this task (a discrete fourier transform of the RGB values). Kazuya et al. [4] propose a way of infering colors from words, treating each character separately to feed a long-term short-term memory unit [5] type of recurrent neural network. However they only try to encode colors from strings and their model is not able to generate strings from colors. Barghava et al. use a architecture similar to Kingma et al. [6] Variational AutoEncoder (VAE) which is a network that maps an input to a smaller dimension (the latent space), which acts as a bottleneck. Barghava et al. identify the latent space as the RGB space. Their architecture takes description of colors as input, then encodes it in the latent RGB space using a gated recurrent unit (GRU) [7] which is a kind of neural network that gets as input its last hidden state in addition to another input (e.g. words of a sentence). The encoded RGB is then decoded to the sequence space to output a color description. The autoencoder architecture lets them sample from the latent space as well as encoding descriptions to a RGB color. They set as an objective function a combination of the reconstruction loss and a distance between the RGB mean and the encoded RGB color. Every description is mapped to a single point in the latent space, which does not reflect the dataset which pairs every sequence with multiple points in the RGB space. We will extend their work proposing an alternative architecture that captures the high variability of the task better than having a deterministic approach and try using the underlying statistics of the dataset.

Chapter 2

Dataset

We will now introduce our dataset in more details. To be able to compare our approach to the methods cited in the related work section we will use the same online color survey dataset gathered and filtered by Monroe et al. [3] This dataset is a list of colors descriptions in the form of sequence of one to four english words. It is based on the survey of Munroe [8]. For each description, a list of colors is encoded in the Hue Saturation Value (HSV) format. This reduces it to more than two million color-description pairs. 829 different captions are attributed to colors. The filtering done by Monroe et al. in their paper gets rid of very rarely occuring descriptions (most of the time typos). One should note that the way people labelled colors was quite subjective and very similar colors could be attributed different descriptions. This is what we described as *high variance* in the introduction as illustrated in figure 2.1.

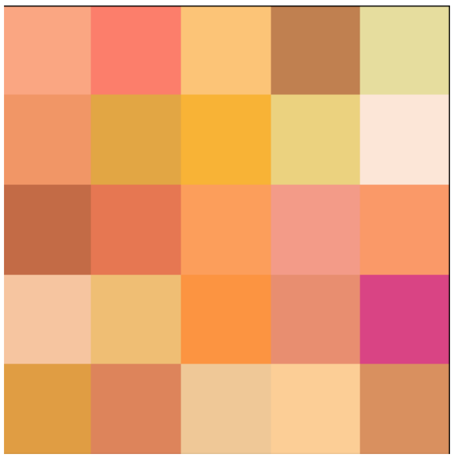

Figure 2.1: Labels for the description *peach*

We notably observe the presence of outliers that may not correspond to the color *peach* for some people. Variance is also present with labels. Some colors are being attributed the same description as can be seen on figure 2.2. The variance of the labelling in both directions is inherent to the taskbecause a color can be percieved very differently.

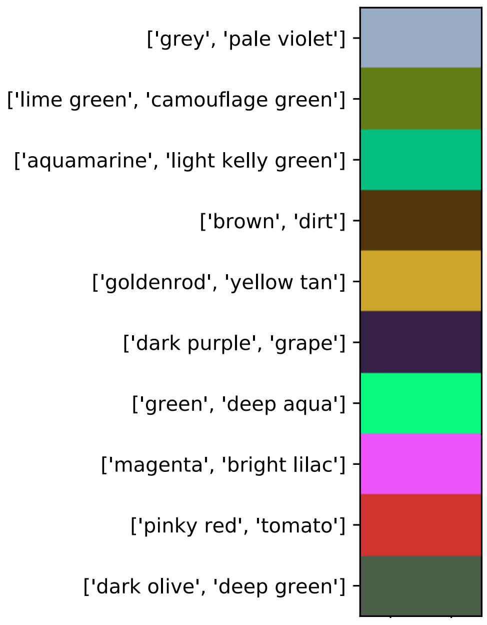

Figure 2.2: Colors being labeled with two different descriptions

We would now like to visualize the statistics of this dataset. Assuming a gaussian distribution for the set of colors corresponding to a description, we can plot gaussian clusters in the RGB space as 3D ellipsoids. We note non-negligeable non-diagonal elements in the covariance matrix and also that some outliers heavily influence it especially when a description is paired with few samples. On figure 2.3 we display three clusters colorized with their mean RGB value.

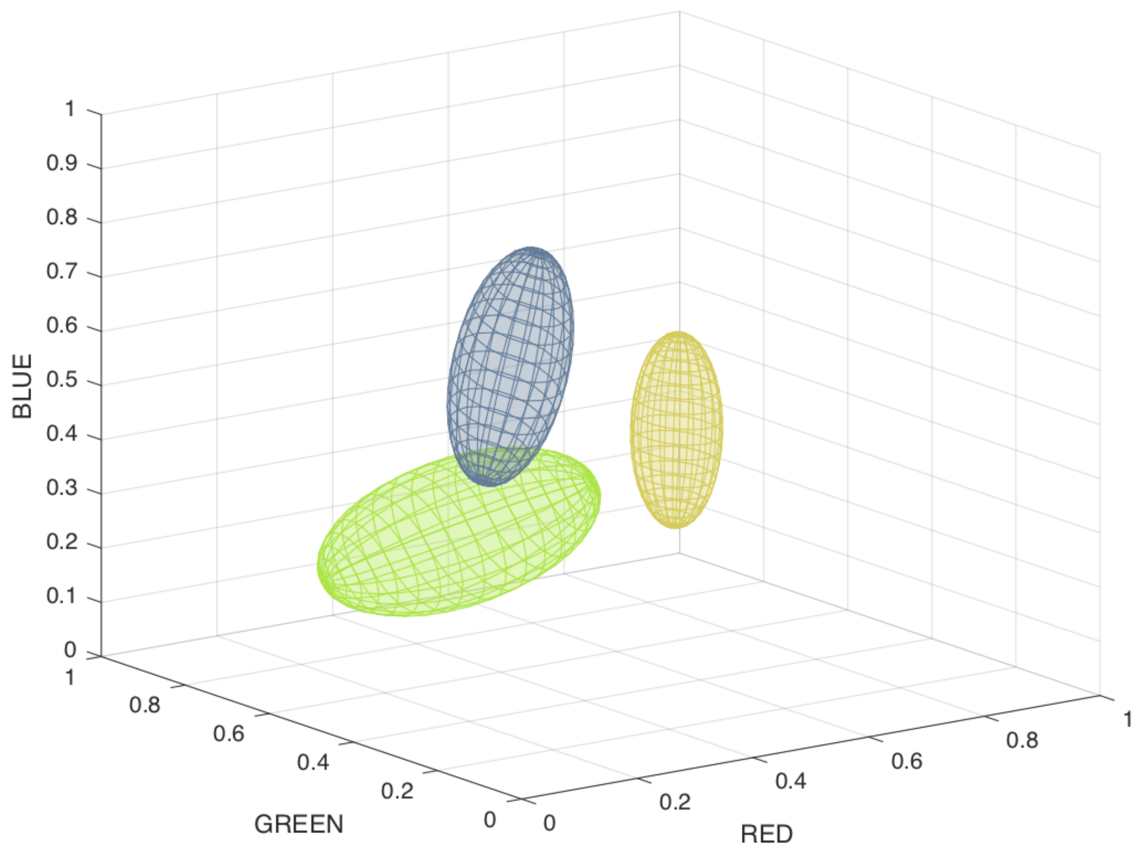

Figure 2.3: Clusters in the RGB space, the color corresponds to the mean point

Chapter 3

Methodology

3.1 Objective function

As stated in the introduction, our objective is to be able to generate descriptions from colors, and sample the RGB space to form descriptions. To adress this in spite of the high variance of the dataset we would like to somehow use the information given by the statistics of the dataset. We will first start with the autoencoder of Bhargava et al. [9] as it allows to generate sequences and colors with ease. First they convert all HSV values to RGB to avoid difficulties dealing with the circular coordinates. They normalize the lengths of the input sequences by adding padding token at the end (after using a special end-of-sequence token), embed them based on pre-trained GloVe embeddings to then feed to a gated-recurrent units network, word by word. The interest of using gated-recurrent units, or any kind of reccurent neural network is its ability to retain information that was previously given, by getting as input both the next word (in our case) and the last hidden state. The first hidden state is initialized randomly. They then output a vector in the RGB space ($[0, 1]^3$). This vector is the latent space representation of the sequence. It is then compared with the real value of the description (based on all associated representations) to make up for the first element of the loss function, namely the distance between real RGB and encoded RGB. Finally they reconstruct the sequence and compute the ability to decode the RGB point to a sequence using a loss function for every word up to the end of sequence token. For network parameters θ, encoding function $enc_\theta(.)$ and input sequence x, the objective function can be written as so [9]:

$$\mathcal{L}(x|\theta) = \mathcal{L}_{\text{reconstruction}}(x|\theta) + \mu \cdot D(enc_\theta(x), v(x))$$

where μ is a scalar to fine-tune the impact of the first term relative to the second term, D is some distance function between the encoded RGB and $v(x)$ the RGB values corresponding to the input sequence x in the train set. The reconstruction function $\mathcal{L}_{\text{reconstruction}}(.|\theta)$ penalizes low prediction of the correct words when decoding the RGB point $enc_\theta(x)$. We will refer to this method as being deterministic since it maps a sequence to a unique point and the training of the decoder is done using this encoded point deterministically.
We introduce now the objective function which we seek to optimize in our method which is composed of two terms similar to the equation above. The main difference is that we would like our autoencoder to generate gaussian clusters for each input sequence which are close to the clusters of the training set. The decoder is then trained to reconstruct points sampled from those clusters. The loss of the input sequence x given the corresponding RGB color label y and the encoder parameters θ and decoder parameters ϕ is :

$$\mathcal{L}(x|y,\theta,\phi) = \mathrm{D}_{KL}(\mathcal{N}(\vec{\mu}_{x|\theta}, \Sigma_{x|\theta})||\mathcal{N}(\vec{\mu}_y, \Sigma_y)) + \mathbb{E}_{\vec{z}\sim\mathcal{N}\vec{\mu}_{x|\theta},\Sigma_{x|\theta})}\left[l(\mathrm{dec}_\phi(\vec{z}), x)\right] \tag{3.1}$$

The loss for the input sequence x and the encoder parameters θ and the decoder parameters ϕ depends on the latent space values corresponding to x in the training set. Because this could be asssimilated as labels, we denote these values which are colors in the RGB space as y and the statistics of those points as the mean μ_y and the covariance matrix Σ_y. The output distribution parameters $\vec{\mu}_{x|\theta}, \Sigma_{x|\theta}$ are depending on the input sequence x and on the encoder parameters θ which we seek to optimize. In the second term we look at the expectancy of getting back our input sequence given the estimated distribution parameters and a sampled point $\vec{z}$. We write $l(.)$ a mesure of the distance between our reconstructed sequence $\mathrm{dec}_\phi(\vec{z})$ and the original sequence x. The details of each term will be explained in the following subsections.

3.2 Encoding

As we said in the introduction of this chapter, we will extend the architecture of Barghava et al. [9] to reflect the distribution of the RGB values associated to each description by having a stochastic process. We also describe the way we penalize our first loss term. In this section we elaborate on how a sequence is attributed a distribution in the RGB space. Since we want to express the variance of the task, a first approach could be to generate colors with a small noise, namely, to encode a color and then sample from a small ball around it in the RGB space. This approach is not satisfactory as the sample covariance matrix has non-zero elements outside of the diagonal as mentioned in the previous section. To adress this, we associate each description of the training set to a gaussian cluster. After gathering statistics for each sequence, we associate every word in the input sequence x to an index in our dictionnary. The dictionnary is composed of all unique words with the addition of special tokens for padding, and to indicate the beginning of a sequence and the end of it. The indices are then mapped to the embedding space based on the GloVe embedding trained on the Wikipedia 2014 dataset. These pre-trained weights map a list of words into an embedding space of 300 dimensions. We assign unknown words of our sequences to random points between the furthest apart points in the GloVe embedding spaces. However because of the GloVe embeddings, colors tend to occur in similar context, and therefore are close in the embedding space. To adress this issue, we let the embedding layer weights be part of the parameters and let them be trained. Feeding the list of embedded tokens to the GRU unit is done iteratively: it is given the first embedded token and an initial hidden state (initialized with a default value of only zeros) and then the second token with the last hidden state and so on an so forth. Our network outputs the parameters of a gaussian distribution in the latent space instead of a RGB points but we are only interested in the last output since it contains the information of the whole sequence. This effectively triples the number of output of the encoder (the mean in the RGB space (three elements) and the symmetric covariance matrix which has six different elements that add up to a total of nine parameters instead of the three in the deterministic case where the encoder outputs a single value in the RGB space). To ensure that our generated covariance matrix is positive semi-definite (which is always the case for a valid covariance matrix), we output a matrix C which we multiplied with its transposed, i.e. $\Sigma := CC^T$. Σ is positive semi-definite, because CC^T is symmetric since $(CC^T)^T = (C^T)^T C^T = CC^T$ and because the product :

$$\vec{x}^T \Sigma \vec{x} = \vec{x}^T C C^T \vec{x} = \|C^T \vec{x}\|^2$$

will be zero only if $\vec{x}$ is zero which makes Σ positive semi-definite. The choice of having C a lower triangular matrix instead of any regular matrix is the lower number of parameters (six instead of nine) and because it enables a fast computation of the determinant. For C lower triangular, it is simply

$$|\Sigma| = |CC^T| = |C||C^T| = |C|^2 = [\prod_{i=1}^{3}(C)_{i,i}]^2$$

since C is diagonal. The lower triangular C matrix is actually the Cholesky decimposition of a positive semi-definite matrix. To compute the distance between our output gaussian parameters and the ground truth we use the Kullback-Leibler divergence, which is a measure of distance between two distributions. It can be computed as follows for two arbitrary multi-variate gaussian distribution: for $P(x) = \mathcal{N}(\vec{\mu}_1, \Sigma_1)$, $Q(x) = \mathcal{N}(\vec{\mu}_2, \Sigma_2)$ and d the number of dimensions:

$$\mathrm{D}_{KL}(P\|Q) = \frac{1}{2}\left[\log\frac{|\Sigma_2|}{|\Sigma_1|} - d + \mathrm{tr}\{\Sigma_2^{-1}\Sigma_1\} + (\vec{\mu_2} - \vec{\mu_1})^T\Sigma_2^{-1}(\vec{\mu_2} - \vec{\mu_1})\right]$$

where tr$\{.\}$ is the trace operator i.e. tr$\{A\} = \sum_{i=1}^{3} A_{i,i}$. We compute the first term of the loss function in (3.1) between $\mathcal{N}(\vec{\mu}_{x|\theta}, \Sigma_{x|\theta})$ and $\mathcal{N}(\vec{\mu}_y, \Sigma_y)$ to ensure that the output distribution for this sequence is close to the training set distribution which is our first objective.

3.3 Decoding

Now that we have a distribution in the RGB space for the input description, we can sample from this distribution a RGB value and decode it, i.e. assign a description to it. The idea behind the sampling of the RGB space is a form of regularization: the decoder network should be able to recognize RGB points likely to come from the sample distribution of the training set. We randomly sample from the gaussian multivariate distribution to get a RGB color. This color,which we write $\vec{z}$, is then mapped to the embedding space using a linear layer. The parameters of this layer are considered part of the decoder parameter and are trainable. We then input to the GRU unit the token signaling the beginning of a word and the first hidden state (initialized with a default value of only zeros) to which we add our embedded sample point. The GRU unit of the decoder then ouputs a probability for every word. We then feed the most likely token, after embedding it back to the embedded space, and the last initial state to which we add the result of the linear layer embedding our sampled value. This addition reinforce the effect of the color $\vec{z}$ to create a description following [9]. We do this until the decoder outputs an end-of-sequence token. Since every word is given a probability with the softmax we can select the highest prediction and compute the *negative log likelihood* loss that penalizes low confidence to predict the correct word like so : $\mathcal{L}(p) = -\log(p)$ where p is the probability attributed to the ground truth. We compare every word in the input string with the sequence we decoded, except for padding tokens. This second loss term correspond to the second term in 3.1. We describe on figure 3.1 the complete pipleine with light grey captions indicating the dimensions, in purple the two terms of the loss function 3.1. The broken line after the $\mathcal{N}(\vec{\mu}_{x|\theta}, \Sigma_{x|\theta})$ term signifies the sampling of one point of this distribution, $\vec{z}$. The $\oplus$ symbol signifies the element-wise addition.

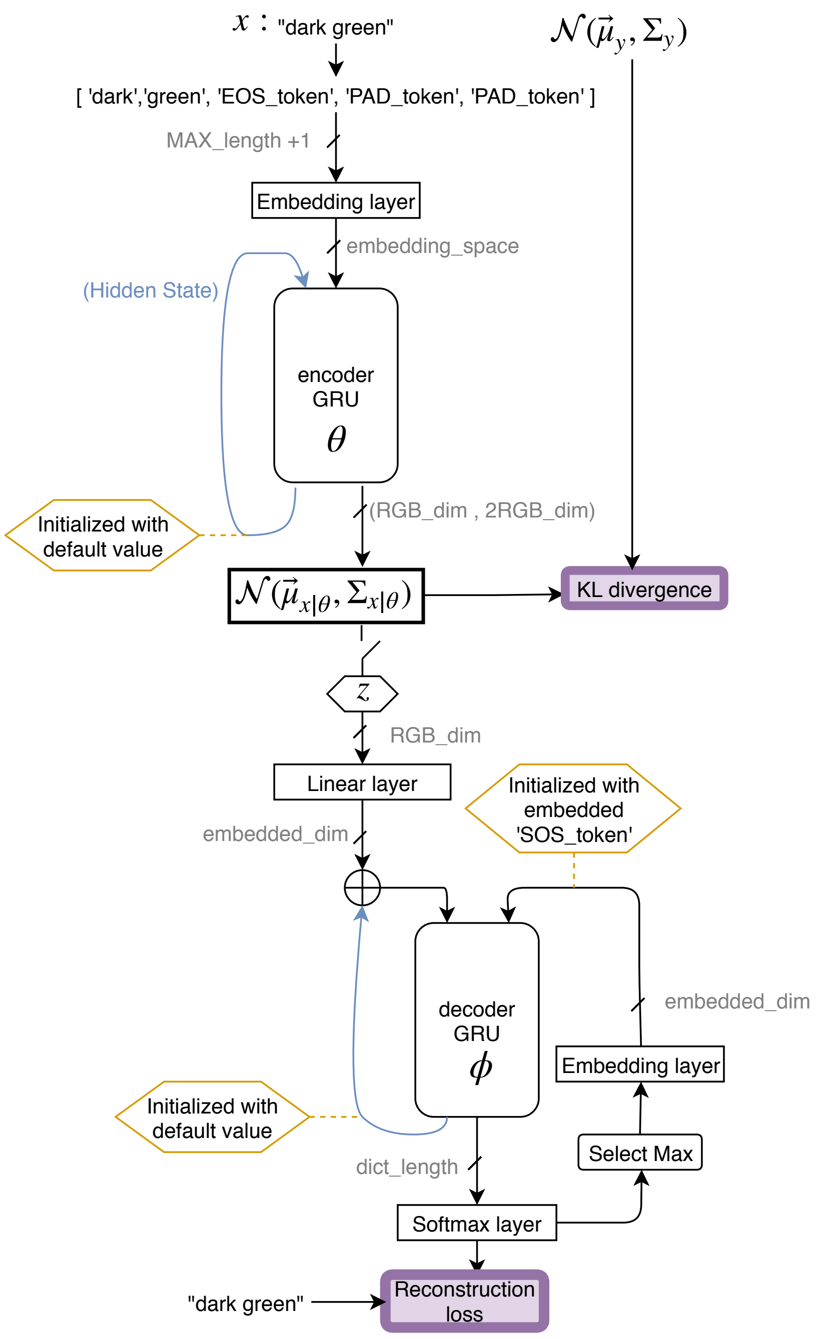

Figure 3.1: The complete pipeline to train the autoencoder

Chapter 4

Training

The two terms of the loss function could be seen as separate goals that could be optimized separately. However, those objectives are contradictory since higher variance means higher reconstruction loss in general as it will likely sample from values far away from the mean which are harder to reconstruct. This means that those two terms balance each others to have both a robust and versatile generative model. We balance the two losses by penalizing the KL divergence loss with a scalar, which we fine-tune with a parameter search, to equilibrate the two losses.

As we saw on figure 2.1, many outliers may have an effect on the mean and covariance matrix of the colors associated with each decription. This outlier effect is naturally reduced with the second term of equation 3.1 which is higher if the variance increases. We also note a shift of the mean which occurs when training our network. Because of this regularization, our network does not have a tendency to overfit and having number of epochs orders of magnitude higher have a small effect on the ability of the autoencoder to generalize. We trained our encoder using Adam optimizer [10] with learning rate 0.00005 and the decoder part of the network with stochastic gradient descent and learning rate of 0.05, and with momentum. Those hyperparameters, and the number of epochs were selected using grid search to determine the optimal factors with regard to our experiences. We point out that to be able to back-propagate the gradient, we use the reparametrization trick in [6] which we extend to the 3-dimensional case in the following equation:

$$\vec{z} = \vec{\mu} + C_\Sigma \cdot \vec{n}, \qquad \vec{n} = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix}, \quad n_i \overset{i.i.d.}{\sim} \mathcal{N}(0,1), \quad i \in \{1,2,3\}$$

We verify the mean and covariance matrix of $\vec{z}$:

$$\begin{aligned}
\mathbb{E}\left[\vec{z}\right] &= \mathbb{E}\left[\vec{\mu} + C_\Sigma \vec{n}\right] = \vec{\mu} \\
\text{Var}(\vec{z}) &= \mathbb{E}\left[(\vec{z} - \mathbb{E}\left[\vec{z}\right])(\vec{z} - \mathbb{E}\left[\vec{z}\right])^T\right] = \mathbb{E}\left[(\vec{z} - \vec{\mu})(\vec{z} - \vec{\mu})^T\right] = \mathbb{E}\left[\vec{z}\vec{z}^T\right] - \mathbb{E}\left[\vec{\mu}\vec{\mu}^T\right] \\
&= \mathbb{E}\left[\vec{\mu}\vec{\mu}^T\right] + \mathbb{E}\left[\vec{\mu}(C_\Sigma \vec{n})^T\right] + \mathbb{E}\left[C_\Sigma \vec{n}\vec{\mu}^T\right] + \mathbb{E}\left[C_\Sigma \vec{n}(C_\Sigma \vec{n})^T\right] - \mathbb{E}\left[\vec{\mu}\vec{\mu}^T\right] \\
&= \mathbb{E}\left[C_\Sigma \vec{n}\vec{n}^T C_\Sigma^T\right] = C_\Sigma \mathbb{I} C_\Sigma^T = \Sigma
\end{aligned}$$

We trained for a total of 500'000 epochs. The process we follow is the one decribed in the methodology applied for each epoch to a different sequence. The sequences are selected randomly at each pass. Some methods needed a rewrite to allow back-propagation using *pytorch* libraries [11] such as the trace, which cannot use *in-place* type of operations, which do not use auxillary space to store data. A typical workaround was the use of copies in customized functions. We also used gradient clipping to tackle the problem of exploding gradient, which can appear when the reccurent unit is getting unstable and cannot update its weights anymore. Gradient-clipping simply enforce that the absolute value of the gradient norm cannot be larger than a specified value. The shortness of the sentences and the relatively small size of the matrix multiplications led to a bottleneck not allowing for a speedup using the GPU. The code is available on `https://github.com/hamdans-eth/colors` as well as the data to replicate results.

Chapter 5

Experiments

We look at both the encoding and decoding to test the ability of our network to generate colors from description, and vice versa, respectively. We would also like to investigate how the stochastic version of the autoencoder improves in comparision with a deterministic one. We tried training both networks and fine-tuning them to make for a fair comparision however we do not exclude that better hyper-parameters could improve performances of both cases.

5.1 Generating colors

Our model is able to generate colors from descriptions using only the first part of our model, i.e. the encoding part. We ran an experience to see how the trained model could generate the color *peach* we saw on figure 2.1. Since our model outputs a distribution in the RGB space, we can sample from it like on figure 5.1.

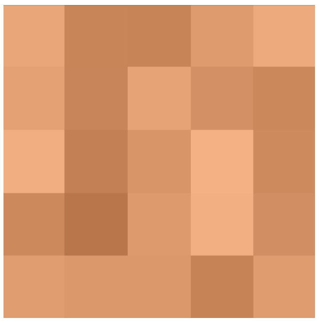

Figure 5.1: RGB samples from the *peach* color distribution of our trained model

The result of this sampling shows less variability as expected from our loss function 3.1 that reduces the variance to allow for an easier reconstruction. We can also try to create sequences generating some pairs of adjective and colors that are not part of our original sequences. On the following figures, we simply output the most likely decoded description by taking the mean of the output distribution for each sequence. We first do a sanity check of our method on figure 5.1 and compare it to the compositionality figure 4 from Barghava et al. [9]. We display the colors on the y-axis and the modifiers on the x-axis (the first entry is read *dark blue*).

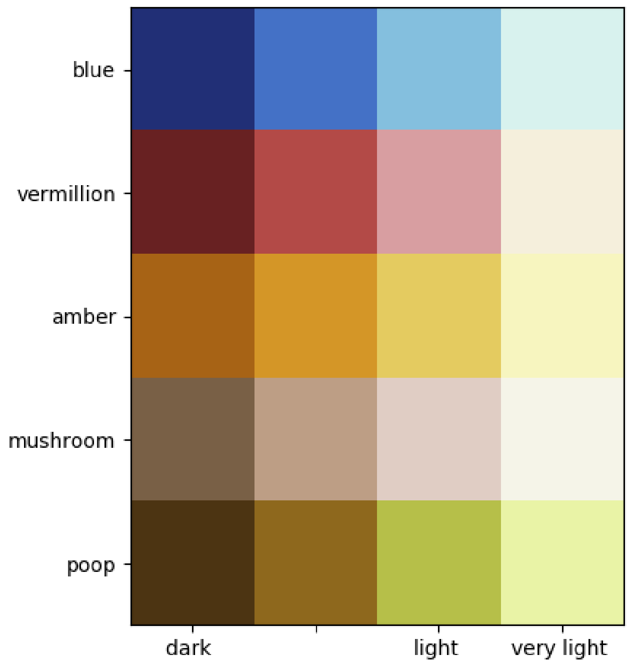

Figure 5.2: How modifiers are learnt (only the first line and the second column are part of the original dataset)

On figure 5.1, we generate further color-modifier pairs to see their effects on colors. Those modifiers are more complicated than simple range of luminosity displayed in 5.1 but are acceptable. It is however hard to quantify the quality of the color generation part as it requires human labelling which is, as discussed in the introduction, quite subjective and variable from one person to the other.

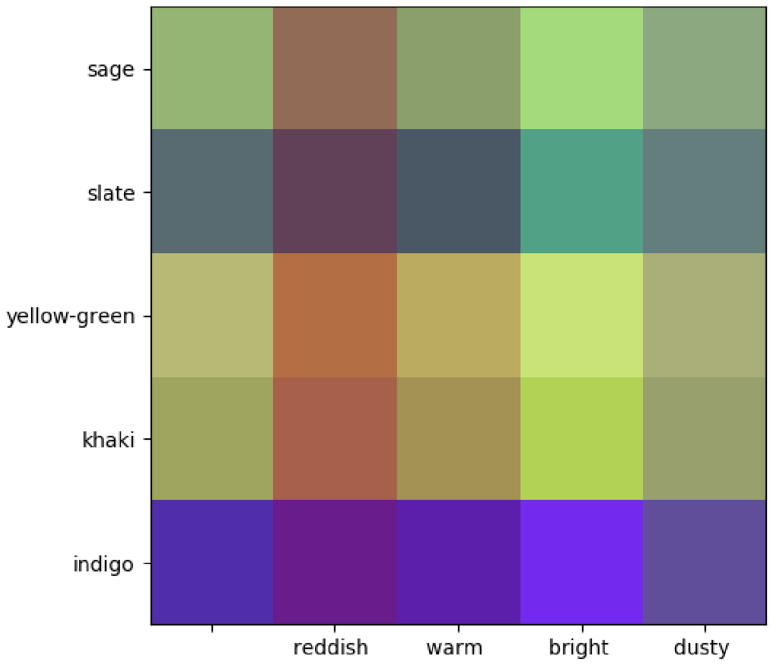

Figure 5.3: Randomly assigning adjectives and colors. Note the subtle differences between variations of yellow-green colors.

5.2 Generating sequences

In this section we look at the effectiveness of predicting sequence of words. The first metric we will use is the absolute accuracy, namely counting how many colors from the test set could be decoded with an exact match to the corresponding sequence on table 5.1. The second metric is the percentage of exact matching of the last word - e.g. *bright green* and *acid green* would be considered a correct prediction. We use first a set of colors sampled from the test set distribution *sampling* and then do the same test using values from the test set *test set* column.

absolute accuracy	sampling	test set
deterministic	0.5%	3.9%
stochastic	**18.1%**	**4.5%**

Table 5.1: Absolute accuracy

If stochasticity only marginally improves the absolute accuracy of our model when using points of the test set, using samples from the gaussian cluster with the test set statistics (the *sampling* column) makes a significant difference between the two models. This can be explained by the fact that in the case of going through RGB colors, some points may have been seen by our network in the training phase, maybe associated with another description. But when confronted with points never seen before, the deterministic autoencoder struggles to generalize whereas the stochastic version performs better. The difference between the *sampling* and the *test set* colum is due to the fact that using points sampled from a gaussian distribution may lead to less outliers which could be hard to decode.

However the absolute accuracy metric may not be really useful as many generated descriptions can be very close to an acceptable sequence and be counted wrong For example, absolute accuracy penalizes *light light* as much as *red orange* when the label is *reddish orange*. For this reason we also use the second measure of accuracy mentioned, the last word match accuracy. We report on table 5.2 the results for this experience and notice similar tendency as in table 5.1. We finally show some actual generated descriptions from both models in figures 5.3, and 5.4, where we select labels with a high error to look in details at what is exactly output. We note that in both deterministic and stochastic cases the description are close. However, we see less artifacts in the stochastic model, and the descriptions can be closer to what a human could label. Also it is quite diverse in terms of words used in the sequences.

last word match	sampling	test set
deterministic	12.2%	15.4%
stochastic	**44.1%**	**31.5%**

Table 5.2: Match percentage for the last word

stochastic	deterministic
"blood red "	"orange orange "
"rust red "	"dark "
"orange red "	"orange orange "
"bright orange "	"salmon "
"orange red "	"light "
"grapefruit "	"light salmon "
"orangey red "	"orange orange "
"rust red "	"orange orange "
"dusty red "	"salmon "

Table 5.3: A set of colors labelled *orange red* decoded by the two models

stochastic	deterministic
"yellowish "	"pale "
"canary "	"light cream "
"light "	"light peach "
"butter "	"light cream "
"canary yellow "	"pale "
"burnt yellow "	"yellow yellow "
"goldenrod "	"pale "
"light yellow "	"light "
"yellowish "	"pale "

Table 5.4: A set of colors labelled *yellow tan* decoded by the two models

Chapter 6

Conclusion

In this project we designed a stochastic version of the deterministic autoencoder method to make it more flexible and more robust in generating sequences and RGB colors. In conclusion we think that having an autoencoder that learns gaussian clusters in the latent space can generalize better than one that outputs points deterministically. We think that our method could be extended directly to setups similar, namely a dataset of sequences with associated points in a latent space. We could for example think of other grounded learning models, such as sound-sequence pairs. However this would require some more data. We think our metrics could be improved to reflect better on how the models perform (maybe having humans evaluating them). Finally we could mix the colors with visual representations such as the MNIST dataset.

Bibliography

[1] A. Lazaridou, M. Marelli, and M. Baroni, "Multimodal word meaning induction from minimal exposure to natural text," *Cognitive Science*, vol. 41, no. S4, pp. 677–705. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cogs.12481

[2] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[3] W. Monroe, N. D. Goodman, and C. Potts, "Learning to generate compositional color descriptions," *CoRR*, vol. abs/1606.03821, 2016. [Online]. Available: http://arxiv.org/abs/1606.03821

[4] K. Kawakami, C. Dyer, B. R. Routledge, and N. A. Smith, "Character sequence models for colorfulwords," *CoRR*, vol. abs/1609.08777, 2016. [Online]. Available: http://arxiv.org/abs/1609.08777

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[6] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[7] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[8] R. Munroe, "Color survey," https://blog.xkcd.com/2010/05/03/color-survey-results/.

[9] D. V. Bhargava, “Grounded learning of color semantics with autoencoders,” 2017.

[10] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[11] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” 2017.