Neural ODEs for Machine Translation

Enrique De Alba
Department of Computer Science
Stanford University
Stanford, CA 94305
edealba@stanford.edu

Glib Stronov
Department of Economics
Stanford University
Stanford, CA 94305
gstronov@stanford.edu

Abstract

Despite recent successes in the field of machine translation, there are still a lot of problems that exists pertaining to the quality of the translations produced. This paper is attempting to adapt the concept of Neural Ordinary Differential Equations to the task of machine translation to utilize its scalability to produce deeper neural networks. We found that this approach takes a much longer time to both train and test when compared to standard LSTM-based methods and produces comparable results as measured by BLEU scores and human evaluation.

1 Introduction

The problem of machine translation is difficult to say the least. A lot of progress has been made in recent years and we currently have neural network models that are increasingly performing close to human level. One of the biggest factors that had contributed to this success has been advances in machine learning as seen in the development of LSTMs and attention mechanisms.

At the same time, the computational requirements for these methods has grown tremendously. For example, GPT-2 is a 1.5 billion parameter model. These solutions are not always scalable and require ever-increasing amounts of computational power and memory to improve. A typical strategy to improving neural network performance is to simply increase the layer depth of the models. The goal of this paper is to show that Neural Ordinary Differential Equations (NODE) systems perform at least on par with standard LSTM-based NMT systems within the context of machine translation. Because traditional neural network models tend to either cap off or perform slightly worse after a certain number of layers, it may be possible that using Neural ODE systems to integrate over an arbitrary amount of layers can outperform traditional models given an inordinate amount of training time.

2 Related Work

The idea of modeling neural networks using partial differential equations is very recent and there is not very much research on it. Researchers seem to agree that it is possible to model discreet iterative updates using differential equations, which should result in smoother updates. The experiments that have been done were pretty small, probably because training takes longer than with more standard methods, but the results are promising. To our knowledge, this method has not been applied to the task of machine translation, so we hope that this paper may contribute to the growing field of research on this topic.

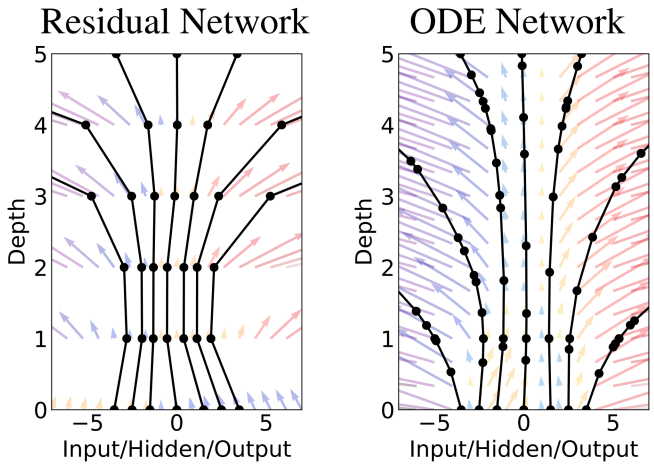
3 Approach

3.1 Neural Ordinary Differential Equations

Residual Networks and Recurrent Neural Network decoders build complex transformations on some data x by sequentially composing transformations to define the next hidden state h_t. For Residual Networks we have the following transformation: $h_t = x_t + \mathcal{F}_t(x_t)$ where $\mathcal{F}_t$ is the t^{th} hidden layer. We can differentiate h as a function of t as the step size $\epsilon \to 0$ (which was previously 1). Here t represents a continuous analog for the network's layers. We can then define the hidden layer $h(T)$ as the solution to the following integral.

$$h(T) = \int_0^T \frac{dh(t)}{dt}\, dt = \int_0^T \mathcal{F}\big(h(t)\big)\, dt$$

The advantages of this method can be seen in the figure below. On the left side, we can see that the Residual Network defines a discrete sequence of finite transformations. On the right side, we observe that the ODE Network defines a vector field, which continuously transforms the state. In both cases, the circles represent evaluation locations. We can see from these graphs that the ODE Network results in much smoother updates.



By defining neural networks as ordinary differential equations and solving them using ODE solvers we can make use of the following: better memory efficiency because of constant memory cost with respect to the neural network depth and the use of less parameters because of the discrete to continuous transformation of the layers.

Just like with traditional neural networks, we would need to figure out a way to perform backpropagation to improve our model. In the continuous case, however, it is not as clear-cut, since we no longer have discrete transitions between layers. In order to address this issue, we present a way to perform a continuous method for backpropagation.

We have a loss function, L, that takes in $h(t_N)$ as input. Let $N = 1$ for simplicity, so we have:

$$L\big(h(t_1)\big) = L\left(h(t_0) + \int_{t_0}^{t_1} \mathcal{F}\big(h(t)\big)\, dt\right) = L\Big(\text{ODESolver}\big(h(t_0), \mathcal{F}, t_0, t_1, \theta\big)\Big)$$

where θ are the parameters of the model. We want to find $\frac{\partial L}{\partial \theta}$ to optimize the parameters like in other neural network models. By defining an adjoint function $a(t) = \frac{\partial L}{\partial h(t)}$ such that $\frac{da(t)}{dt} = -a(t)^\top \frac{\partial \mathcal{F}(h(t))}{\partial h(t)}$ and

$$\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} -a(t)^\top \frac{\partial \mathcal{F}\big(h(t)\big)}{\partial \theta}\, dt$$

$$\frac{\partial L}{\partial h(t_0)} = \frac{\partial L}{\partial h(t_1)} + \int_{t_1}^{t_0} -a(t)^\top \frac{\partial \mathcal{F}\big(h(t)\big)}{\partial h}\, dt = a(t_0)$$

we can compute the gradients of L with respect to all layers given the final layer t_N and θ using the ODESolver with $\mathbf{0}_{|\theta|}$, $-a(t)^\top \frac{\partial \mathcal{F}}{\partial \theta}$ and $\frac{\partial L}{\partial h(t_1)}$, $-a(t)^\top \frac{\partial \mathcal{F}}{\partial h}$ as inputs respectively.

However, this method has not been tested on a big scale. The original Neural ODE paper only presents experiments on small datasets and assumes that the same results will be observed on a larger scale. This paper tries to adapt this concept to the task of machine translation and implements it on a much larger scale in order to verify these results. We demonstrate that this new paradigm of using ODESolvers as a method to integrate differential forward functions is effective.

3.2 Adapting Neural ODEBlock in Machine Translation System

For the purpose of this paper, we decided to use a character-based convolutional encoder that is based on Kim et al.'s work described in *Character-Aware Neural Language Models*. First, we convert a batch of words to character indices, pad them, and perform embedding lookups for the characters. We then pass these character embeddings into a 1D convolutional layer with max-pooling and ReLU activation. After that, we utilize an ODEBlock and input the convolutional network's output x_{convout} into it to output the tuple (x_{convout}, ODESolver($\mathcal{F}, x_{\text{convout}}, t_{\text{int}}$)) where $\mathcal{F}$ is the ODEFunction of the ODEBlock and takes x_{convout} as its input and t_{int} is the integration time used in the ODESolver. The ODEBlock outputs ODESolver($\mathcal{F}, x_{\text{convout}}, t_{\text{int}}$) which goes through a dropout layer and outputs the final word embeddings subsequently used in a Seq2Seq translation network.

The forward functions in our ODEBlocks take t and x as inputs, where x is the output of the convolutional layer and t is a float value input by the ODESolver. Whenever we're training a Neural ODE network we have the choice to use the adjoint function for the ODESolver. In this paper we denote models that use the adjoint function with **adj**. Typically, utilizing the adjoint function results in increased model performance. We illustrate the difference between using the adjoint function and not using by observing the different values of t. Given an integration time of $[0, 1]$, i.e. we're "integrating" from 0 to 1, then t values, for example, could range from values between 0 and 1.87 (depending on the error tolerance of the ODESolver) if we're not using the adjoint function. If we are using the adjoint then t could range from -0.35 to 1.87 and include $-\infty$.

4 Experiments

4.1 Data

We decided to use the data provided to us in HW5. The reason for this decision is that we are already familiar with this data and we know what to expect from the baseline neural machine translation system similar to the one we implemented in class for this data. Having this knowledge helped us set up our Neural ODE system.

4.2 Evaluation method

We compare a vanilla Seq2Seq machine translation network (**Baseline**) to Neural ODEs (**NODE**) Seq2Seq machine translation networks with different forward functions. To do that, we primarily rely on metrics such as the BLEU score and the time it takes to perform the translation. We also compare the time it takes to fully train the models. Some further analysis includes observing how rapidly the loss and perplexity decrease as we train our models. Finally, since BLEU scores can be misleading we have done some limited human evaluation of the translations and made one-to-one comparisons between networks.

4.3 Forward Function Experiments

Experimented with various forward functions: (Note that $\mathcal{F}$ is the differential function we're solving with the ODESolver)

- **NODE** Vanilla ResBlock Forward Function:
  Baseline for the forward function of ODEFunction which is not dependent on t, which is a float value input into the forward function by the ODESolver. We want to find a forward function that incorporates t into the (differential) function $\mathcal{F}$, which is be input into the

ODESolver to be solved, such that it outperforms this baseline.

$$a \leftarrow \text{ReLU}\left(W_i x + b_i\right) \text{ where } W_i \in \mathbb{R}^{e_{\text{word}} \times e_{\text{word}}}, b_i \in \mathbb{R}^{e_{\text{word}}}$$
$$\mathcal{F}_{\text{baseline}}(t, x) = x + \text{ReLU}\left(W_j a + b_j\right) + a$$

- $t-$Baseline Forward Function:
  Here we restricted $t \in [-2.5, 2.5]$ since we found it would sporadically output $-\infty$. We found that this forward function was strictly worse than the baseline because it's basically a re-scaling of $\mathcal{F}_{\text{baseline}}$.

$$a \leftarrow \text{ReLU}\left(t * W_i x + t * b_i\right) \text{ where } W_i \in \mathbb{R}^{e_{\text{word}} \times e_{\text{word}}}, b_i \in \mathbb{R}^{e_{\text{word}}}$$
$$\mathcal{F}_{\text{t-baseline}}(t, x) = x + \text{ReLU}\left(t * W_j a + t * b_j\right) + t * a$$

- **NODE**$-t$ Concatenated Forward Function:
  We found this forward function to outperform both $\mathcal{F}_{\text{baseline}}$ and $\mathcal{F}_{\text{t-baseline}}$. Since t is incorporated into this forward function then there may be some time differences when using the adjoint ODESolver versus the non-adjoint ODESolver. We have experimented with both ODESolvers as seen with NODE$-t$ and NODE$-t$ adj.

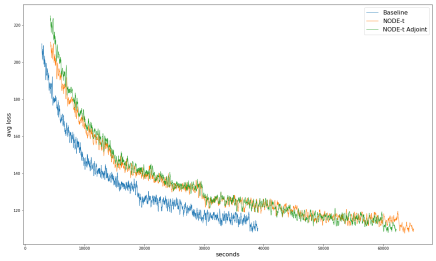
$$X^* \leftarrow \text{ReLU}(x)$$
$$A^* \leftarrow \text{ReLU}\left(W_{t(1)} T + b_{t(1)}\right) \text{ where } W_{t(1)} \in \mathbb{R}^{e_{\text{word}}+1 \times e_{\text{word}}}, b_{t(1)} \in \mathbb{R}^{e_{\text{word}}}$$
$$\mathcal{F}_{\text{t-concat}}(t, x) = x + W_{t(2)} \widetilde{T} + b_{t(2)} \text{ where } W_{t(2)} \in \mathbb{R}^{e_{\text{word}}+1 \times e_{\text{word}}}, b_{t(2)} \in \mathbb{R}^{e_{\text{word}}}$$
$$\text{where } T = \left[\begin{array}{c|c} t & \\ \vdots & \mathbf{X}^* \\ t & \end{array}\right] \text{ and } \widetilde{T} = \left[\begin{array}{c|c} t & \\ \vdots & \mathbf{A}^* \\ t & \end{array}\right]$$

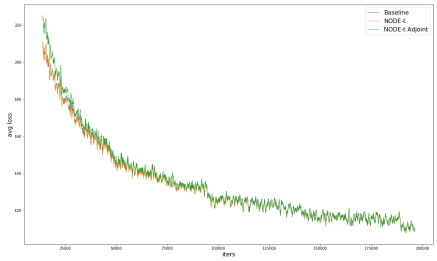
4.4 Results

- Summary Results:

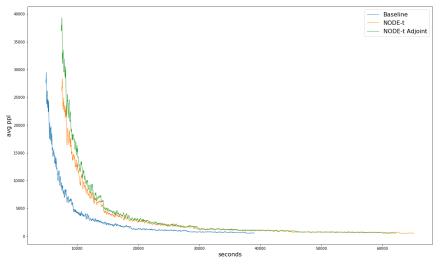
	BLEU	Train Time (hrs)	Decoding Time (mins)	Avg Words/Sec
Baseline	24.45	10.8	9.2	2854
NODE Vanilla adj	24.05	25.2	58.3	
NODE$-t$	24.25	18.1	96.6	1716
NODE$-t$ **adj**	24.45	17.3	71.4	1791

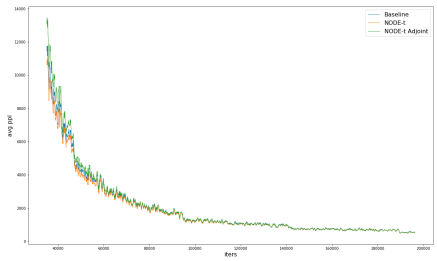
- Loss and Perplexity:
  - Average Loss:





  - Average Perplexity:





If we look at the loss and perplexity versus iterations graphs, we can see that they decrease at a very similar rate for our NODE$-t$ models and our baseline. At the same time, we can clearly observe the difference between them when we look at the loss and perplexity versus time graphs. According to those graphs, our baseline clearly outperforms NODE$-t$, since it is able to achieve lower loss and lower perplexity levels in a shorter period of time. What this means to us is that even though NODE$-t$ and our baseline produce relatively similar results (as further indicated by their close BLEU scores), NODE$-t$ takes a much longer time to achieve them. NODE$-t$ took 18.1 hours to train and 96.6 minutes to decode, while Baseline took only 10.8 hours to train and 9.2 minutes to decode.

While we expected the longer training time, we did not expect it to be almost twice the time it takes to train the baseline. Decoding time difference is tremendous as well - more than a tenfold increase when comparing the baseline and the NODE$-t$ performance. Our best guess as to why this is the case is that traditional LSTMs have already been optimized for speed, while our method does not have that optimization. This means that the comparison is not exactly fair and baseline does have an advantage, but it does illustrate the current state of things.

5 Analysis

Since we are dealing with language models and the quality of translation cannot be perfectly captured with just the BLEU score and some quantitative metrics, we also provide some qualitative analysis of how our models perform. Below we include four cases, which we believe are illustrative of the differences between the models.

Translation Cases:

- Original: *A medida que se derrite un tmpano libera agua dulce rica en minerales que alimenta a muchas formas de vida.*

Human Reference	As the iceberg melts, it is releasing mineral-rich fresh water that nourishes many forms of life.
Baseline	As you melt a iceberg free water fresh water in minerals that feeds many forms of life.
NODE Vanilla adj	As a taxes would melt a train that releases rich water in minerals that feed many forms of life.
NODE$-t$	As a iceberg releases a rich fresh water in minerals that feed many forms of life.
NODE$-t$ adj	As a ties is breaking a little rich fresh water in minerals that feed many forms of life.

In the first example, we can see that all models are able to translate the last part of the sentence correctly and most of the differences come in at the beginning. For some reason, our NODE Vanilla model is translating, "as the iceberg melts" to be "as a taxes would melt" and our NODE$-t$ adjoint is translating that part as "as a ties is breaking." At the same time, the baseline outputs "slide" instead of "release" and doesn't describe the "mineral-rich fresh water" correctly. NODE$-t$ produces the best translation, outperforming the baseline in this case, with a more valid sentence structure, only misplacing the word "rich."

- Original: *Ella salv mi vida al permitirme entrar al bao de la sala de profesores.*

Human Reference	She saved my life by letting me go to the bathroom in the teachers' lounge.
Baseline	She saved my life by allowing me to enter the teacher's bathroom.
NODE Vanilla adj	She saved my life when he allowed me to go to the teacher room.
NODE$-t$	She saved my life when she allowed me to go into the teacher's room.
NODE$-t$ adj	She saved my life when she allowed me to go into the teacher's bathroom.

In the second example, all models output similar translations. One notable difference is that only the NODE Vanilla model is unable to correctly infer the gender in both cases with "she saved... he allowed..." Another difference is that only NODE$-t$ and baseline methods were able to pick up that we are talking about bathrooms, not just rooms. While the NODE$-t$ adjoint and the baseline translations are very close in this case, NODE$-t$ adjoint produces a slightly more natural sentence, since we don't usually say that we enter bathrooms.

- Original: *En educacin, la nica cosa que sabemos cmo medir de mejor manera*

Human Reference	In education, the one thing we know how to measure best is IQ.
Baseline	In education.
NODE Vanilla adj	In education, the only thing we know how to measure a better way.
NODE$-t$	In education, the only thing we know how to measure in better ways.
NODE$-t$ **adj**	In education, the only thing we know how to measure the best way.

Looking purely at the Spanish original sentence there is no reference to "IQ", so it should be incredibly difficult for any MT system to translate this sentence in line with the human reference. Here we see that the baseline NMT model simply outputs "In education." which results in a relatively low BLEU score because of its brevity. We see that all the NODE MT models produce longer sentences that are much closer to the human reference, and given the original Spanish sentence, as close as the translations could get to the human reference, with the NODE$-t$ models being slightly more accurate than NODE vanilla.

- Original: *Primero, botnicos... de los que podrn hacerse una idea.*

Human Reference	First, botanical – which you can kind of get a sense of.
Baseline	First of all, bottles.
NODE Vanilla adj	First, biologists – that you can get a idea.
NODE$-t$	First, buttons that will be able to make an idea.
NODE$-t$ **adj**	First, bottoms, you can make an idea.

This sentence has a very unusual structure, which is probably what confuses our models. The baseline is producing a very short and very incorrect translation - translating "botanical" as "bottles" and choosing to end there without mentioning anything else. Our models perform better, as all of them get the second part of the sentence right and our NODE Vanilla is producing the closest translation to "botanical" by outputting "biologists."

- Original: *A su sobrina? A su amiga?*

Human Reference	Your niece?
Baseline	Is it the one to your owner?
NODE Vanilla adj	Her supernova? Is her friend?
NODE$-t$	Is it your support? Is your friend of friend?
NODE$-t$ **adj**	Her supply to his friend?

Here we see all models having difficulty with the original Spanish sentence, producing largely nonsensical translations. The human reference is not exactly correct since it leaves out the "A su amiga?" section in the original sentence. The NODE models capture some basic quality of this section, as seen with the vanilla model capturing it in the second part of its output with, "...Is her friend?" Moreover we see both NODE$-t$ models' attempts in "...Is your friend of friend?" and "...his friend?" Of course, these translations are examples of erroneous outputs as we see references to "supernova" and other illogical aspects to the sentences.

This shows that even though the BLEU scores are relatively close the models are indeed outputting different things in our few handpicked translation examples.

6 Conclusion

We show that using ODESolvers to solve differential equations as a new proxy for forward functions in an NMT context performs on par with traditional LSTM-based NMT systems. These NODE machine

translation models display significantly worse training/testing speeds compared to the traditional baseline NMT model because it is difficult to compete with respect to speed due to the rigorous optimization that has gone into LSTMs compared to ODESolvers. Integrating forward differential equations is an exciting new paradigm that may prove to be more effective when increasing the depth of ResBlock layers and integrating these layers as functions of t. NODE models should in theory be able to produce arbitrarily deep ResBlock layers, so despite of their speed inefficiencies, may still end up producing better models given enough time.

7 Future Work

- Experiment with increasing depth of ResBlock layers, including experimenting with setting the depth as a function of t.
- We want to explore how changing the error tolerance of the ODESolver could speed up training, etc. Additionally, we would want to see how changing other hyperparameters may affect our performance.
- We would also like to test our systems on other language pairs. A particularly interesting area for experiments can be language pairs that don't contain English.

8 Acknowledgements

We would like to thank the entire teaching team of CS224N for their continued support and guidance with this project. Especially we would like to thank Chris Manning, who was our mentor for this project and who provided numerous valuable suggestions.

References

[1] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. *Neural Ordinary Differential Equations*. University of Toronto, Vector Institute, 2019

[2] Marco Ciccone, Marco Gallieri, Jonathan Masci, Christian Osendorfer, and Faustino Gomez. *NAIS-NET: Stable Deep Networks from Non-Autonomous Differential Equations*. NNAISENSE SA, 2018

[3] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. *Character-Aware Neural Language Models*. Cornell University, 2015

[4] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. *Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations*. Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden

[5] Lars Ruthotto and Eldad Haber. *Deep Neural Networks Motivated by Partial Differential Equations*. arXiv preprint arXiv:1804.04272, 2018