Using Skip Connections to Eliminate Singularities

By:

Ben Parker

Tommy Hoang

Image Processing with Elements of Learning (CSCI-567 & MATH-563)

Spring 2022

Instructor: Nikolay M. Sirakov, Ph.D

Table of Contents

Abstract

Skip (Residual) connections are additional connections between nodes in different layers that “skip” one or more layers in nonlinear processing. Skip connections are implemented to create residual neural networks (RNNs), which utilize nonlinearities and batch normalization, i.e. pooling and dense layers. This paper will examine how skip connections eliminate singularities by demonstrating improvements to the testing and training data compared to a neural network that does not implement skip connections. In our test results, it is shown that the RNN displayed more staggering results whereas the CNN results were smooth. At 250 epochs and using the first 2,500 sets from the CIFAR-10 dataset, the RNN proved to be more efficient, whereas the global loss maxima is lower than the loss global loss maxima of the CNN. In accuracy, the CNN converges to 1 quicker and proved stable compared to the RNN accuracy.

Introduction

In the representation learning approach to machine learning, models can not only map representations of features to an output, but also learn the representation itself.(Goodfellow 2017) However, representation learning algorithms are challenged by the task of identifying relevant abstract features from large amounts of raw data. Deep learning methods offer a solution to this problem by first learning simpler representations and using these to build increasingly abstract representations in a hierarchical manner, similar to biological neural networks.(Pio Monti 2018)

Therefore, it is intuitive that a feedforward neural network with increased depth would have superior ability to create abstract representations and learn patterns with more complexity. The validity of this reasoning was demonstrated by Krizhevsky, et al., whose 8-layer convolutional neural network classified millions of images into thousands of categories with state-of-the-art error rates. An important conclusion from this breakthrough is that removing layers from the network architecture significantly worsened error rates.(Krizhevsky, et al. 2017)

However, this result begs the question- if an 8-layer network can achieve accuracy for such a difficult task, why not use an exponentially deep network to obtain superior "intelligence"? Although network depth is crucial for recognition tasks, improving learning capacity is more complicated than simply constructing networks with more layers. In general, the application of deep learning is beset by practical limitations which emerge when training deeper neural networks by backpropagation and gradient descent.

A major subset of artificial intelligence research is devoted to identifying and developing solutions to the difficulties associated with training deep neural networks. For example, overfitting, or excessive parameterization, results in a model that is too effective at learning features in the training data, such that accuracy is not maintained when testing with generalized data. This problem has been sufficiently addressed by regularization procedures, such as dropout. Another notorious example is the exploding/vanishing gradient problem, where the backpropagation algorithm computes very large/small gradients, resulting in dysfunctional parameter updates. This problem has also been addressed, notably by a normalized initialization procedure, which reduces the extent to which the variance of the back-propagated gradient depends on the layer.(Glorot, Bengio 2010)

In the following sections, we focus on the training difficulties associated with singularities of the parameter space of a deep neural network. We provide clarification on the significance of the problem and provide a solution by construction that has been described previously in the literature- the use of skip/residual connections. Finally, we investigate the claim that residual connections improve the rate of learning in feedforward networks by comparing experimental data from a convolutional neural network with a residual neural network. Our results show that the residual neural network outperformed the convolutional neural network in both loss and accuracy. In the experiment, it is also shown that there is no discernible improvement between both loss and accuracy in the convolutional neural network and the residual neural network past around 125 epochs.

Theoretical Foundations

In the following subsections, we describe singularities of the Hessian, their causes, and how they affect learning through gradient descent. Then, we explain why skip connections are a useful solution to this problem. Finally, we define and provide justification for all operators, algorithms, and functions used in our implementation.

$$\mathbf{H}_f = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n} \\[2.2ex] \dfrac{\partial^2 f}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2\,\partial x_n} \\[2.2ex] \vdots & \vdots & \ddots & \vdots \\[2.2ex] \dfrac{\partial^2 f}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Figure 1: General structure of the Hessian

1. Singularities

For a function of multiple variables, the Hessian, a square matrix of second order partial derivatives, describes the local curvature of the function. (Fig. 1) A significant challenge in optimization through gradient descent is the existence of points with zero gradient that are neither local nor global minima, commonly referred to as saddle points or singular points. At these points, the Hessian is singular and its determinant is zero. Here, there may exist multiple directions in the gradient with both higher cost and lower cost- the cost is lower along eigenvectors associated with negative eigenvalues of the Hessian, and the cost is higher along eigenvectors associated with positive eigenvalues.(Goodfellow 2017)

As a result of the singularity, parameters become non-identifiable. In other words, for a subset of units, there are more than one equivalent ways of partitioning their output weights such that the network remains invariant.(Wei, et al. 2008) This creates a problem for the gradient descent algorithm, which is based on the assumption that any "step" that lowers cost is closer to the global minimum. When the model is non-identifiable, this assumption does not hold.

The magnitude of this problem is greater in deeper networks, where the cost is computed as a function of many variables. For a function of an n-dimensional space, the ratio of the number of saddle points to local minima grows exponentially with n.(Goodfellow 2017) The practical consequence of singularities is a slowdown in the rate of learning, sometimes referred to as the "plateau phenomenon." Wei, et al. generalized the dynamics of learning in the neighborhood of singularities in hierarchical systems. The authors describe the singularity region as an attractor. If the trajectory of model parameters converge to a stable part of this region, the random walk behavior of parameters gives rise to the plateau phenomenon.(Wei, et al. 2008)

Orhan and Pitkow describe three causes of singularities in fully-connected networks, where subsets of parameters become non-identifiable. (Fig. 2) With elimination singularities, zero incoming weights create zero outgoing weights, which are non-identifiable. Overlap singularities occur when the incoming weights of two nodes in the same layer are identical. Thus, the outgoing weights are also identical and non-identifiable. Linear dependence singularities, which only occur in linear networks, are caused by linear dependence of a subset of incoming weights.

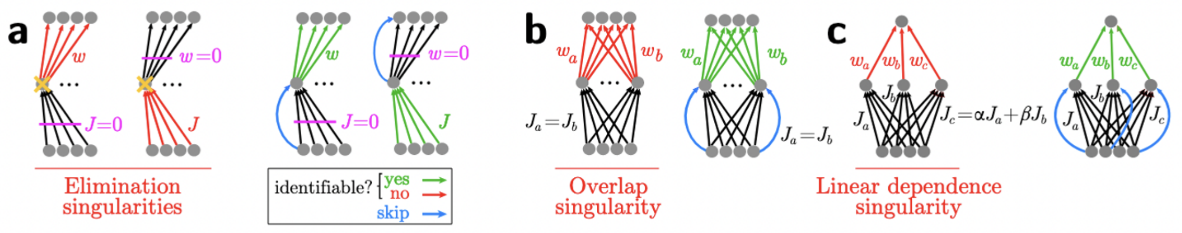

a
w
w=0
J=0
J
Elimination singularities
identifiable? yes no skip
b
w_a
w_b
$J_a=J_b$
Overlap singularity
c
w_a w_b w_c
J_a J_b J_c
$J_c=\alpha J_a+\beta J_b$
Linear dependence singularity

Figure 2: Three causes of non-identifiability and how these are overcome using skip connections. (Orhan, Pitkow 2017)

In each of these scenarios, the Hessian becomes singular. Consider equation 1, borrowed from Orhan and Pitkow, which describes the derivative of the cost function with respect to the parameter matrix, $W_{l,ij}$ between layer l and layer l+1. Here, the outputs for a given layer are defined recursively by $x_l = f(W_{l-1}x_{l-1})$ and the input vector is defined by $h_l = W_{l-1}x_{l-1}$.

$$\frac{\partial E}{\partial \mathbf{W}_{l,ij}} = -\begin{bmatrix} 0 \\ \vdots \\ f'(\mathbf{h}_{l+1,i})\mathbf{x}_{l,j} \\ \vdots \\ 0 \end{bmatrix}^{\top} \mathbf{W}_{l+1}^{\top}\mathrm{diag}(\mathbf{f}'_{l+2})\mathbf{W}_{l+2}^{\top}\mathrm{diag}(\mathbf{f}'_{l+3})\cdots\mathbf{W}_{L-1}^{\top}\mathrm{diag}(\mathbf{f}'_{L})\mathbf{e} \tag{1}$$

In the case of elimination singularity at unit $x_{l,j}$ the column/row of the Hessian corresponding to the parameter $W_{l,ij}$ becomes zero for all i. Hence, the Hessian is singular. The same result can be expected for the other types of singularities. Consider the example of overlap singularity- if two presynaptic units j and j’ have the same set of incoming weights, then $\frac{\vartheta E}{\vartheta W_{l,ij}} = \frac{\vartheta E}{\vartheta W_{l,ij'}}$. That is, because $x_{l,j} = x_{l,j'}$ the partial derivative of the cost function with respect to $W_{l,ij}$ is the same as the derivative with respect to $W_{l,ij'}$. Consequently, the columns/rows corresponding to $W_{l,ij}$ and $W_{l,ij'}$ in the Hessian become identical, making the Hessian singular.(Orhan, Pitkow 2017)

2. Skip Connections/Residual Neural Networks

Residual network architectures were first introduced by He et al. to alleviate the degradation problem in convolutional neural networks, where rapid degradation of accuracy counterintuitively occurs beyond a certain depth. As network depth is correlated with integration of more high level features, maintaining accuracy in deeper networks is crucial for training networks to perform more complex tasks. The authors argue that the degradation problem is neither caused by overfitting (regularization methods would have solved the problem) nor vanishing/exploding gradients (the problem persists despite the use of normalized initialization).

To address this problem, He, et al. developed a solution by including an identity mapping to the underlying function, thereby adding the output from a previous layer to the next layer.(Fig. 3) This identity mapping, commonly referred to as a “skip connection,” is an effective way to ensure that the outgoing signal is maintained when parameters are deactivated or non-identifiable.

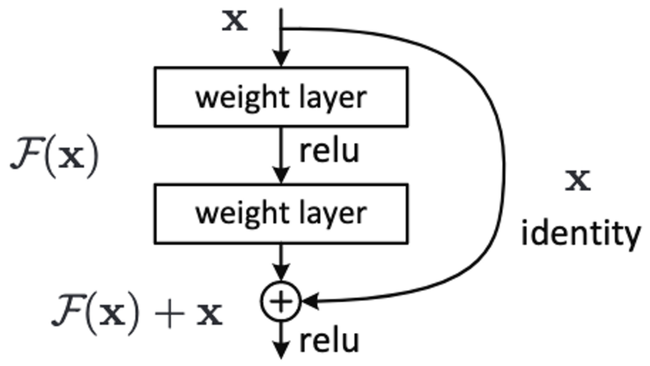

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3: A simple skip connection, showing the underlying function and added identity element (He, et al. 2015)

The authors suggest that the identity element in the residual function provides preconditioning to some extent. That is, if an optimization algorithm determines that the identity mapping is ideal, it may drive the parameter closer to zero to approach the identity.(He, et al. 2015) Thus, it may be advantageous to include a separate identity mapping to the underlying function in deeper networks.

It is relevant to note that the implementation of skip connections adds no new parameters to the network and does not affect complexity of the algorithm. Therefore, it is possible to compare the performance of networks with and without residual connections by keeping hyperparameters constant.(He, et al. 2015)

3. Convolutional Neural Networks

Convolutional networks use a convolution operator in place of matrix multiplication in at least one layer. The convolution operator is defined by the integral of the product of two functions after one is reversed and shifted. Equation 2 shows the discrete form of the convolution operation, where I is the input and K is the kernel.

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(i-m, j-n)K(m,n) \quad (2)$$

Subsequent feature map values are calculated according to Equation 2, where the input image is denoted by I and the kernel being K. The indexes of rows and columns of the result matric are marked with i and j respectively.

Certain properties of the convolution operation are suited for application in deep learning, especially object classification. Primarily, convolutional networks have spare connectivity, meaning the kernel is smaller than the input. This reduces the number of operations required and consequently, the memory requirements of the model compared to matrix multiplication, which requires all parameters to compute the output.(Goodfellow 2017) In addition, the use of convolution results in parameter sharing within the model, or using the same parameter for more than one function. This also reduces the storage requirements of the model, since only one set of parameters is needed for each location.(Goodfellow 2017) Furthermore, parameter sharing creates equivariance of translation. In the context of images, this means that translating an object in the input equivalently translates the representation in the output.(Goodfellow 2017)

4. RMS Propagation

The Root Mean Square (RMS) Propagation algorithm is a method in which the learning rate is adapted for each of the parameters. The goal behind RMS propagation is to divide the learning rate for a weight by a running average of magnitudes of recent gradients of those weights. In the running average calculated in terms of means square:

$$E[g^2] = \beta E[g^2](t - 1) + (1 + \beta)(\frac{\partial C}{\partial \omega})^2 \qquad (3)$$

In Equation 3, E[g] is the moving average of squared gradients, dC/dω is the gradient of the cost function with respect to the weight. The parameters used in RMS propagation are updated through this equation:

$$\omega_{ij}(t) = \omega_{ij}(t - 1) - \frac{n}{\sqrt{E[g^2]}} \frac{\partial c}{\partial \omega_{ij}} \qquad (4)$$

The use of RMS propagation is not for mini-batches, as it would violate the concept of stochastic gradient descent. Instead, the RMSProp moves the average of the squared gradients for each weight and divides the gradient by the square root of the mean square.(Huang 2020)

In Equation 4, w is the moving average of the squared gradients. The partial derivative is the gradient of the cost function and the weight. n is the learning rate and β is the moving average parameter.

5. ReLU Function

The rectified linear activation function (ReLU) is a piecewise linear function that outputs the input directly if it is positive, but zero in other cases. Normally, ReLU has become the default activation function for many neural networks, including Residual Neural Networks and Convolutional Neural Networks. Rectified linear units are easy to optimize due to their near-linearity. Linear models are easier to train with gradient-based methods.(Goodfellow 2017)

An example of how ReLU activation works can be described using an if-statement.

```
if(input < 0)
        return 0;
else
        return input;
```

The function g() can be described using the max function over 0.0 and z.(Brownlee 2020)

```
        g(z) = max{0, z};
```

Using this function, any negative values will snap to 0 and the positive values will act as normal.

6. Dropout Function

The dropout function works by setting a random edge of hidden units such as neurons that make up hidden layers to 0 at each update of the training phase. The dropout layer randomly sets input units to 0 with a frequency of *rate* at each step during training time. Note that the dropout layer only applies to the training phase of a neural network.(Maklin 2019)

The purpose of the dropout function is to prevent overfitting of nodes per layer. Dropping the most experienced node forces the neural network to train the weaker nodes to produce a less chaotic result. The arguments required for a dropout function are rate, noise shape, and seed. Rate is a percentage in decimal to determine the probability of input units to drop. Noise shape is a 1 dimensional integer representing the shape of the binary dropout mask that will be multiplied with the input. Lastly, the seed is an integer that is used as a random seed.

In Python, the random function is normally predestined with a set of numbers unless a seed is implemented into the random function to force the random function to operate outside the premade integers.

7. Global Average Pooling

Global average pooling is an operation designed to replace fully connected layers in classical CNNs, where the concept is to generate one feature map for each corresponding category of the classification task in the last layer. Instead of adding fully connected layers on top of the feature maps, we take the average of each feature map, and the resulting vector is fed directly into the softmax layer.

8. Categorical Cross-Entropy & Log Loss function

Categorical Cross-Entropy is a loss function used in multi-class classification tasks. This function is mostly found in recognition programs (image recognition, voice recognition). The equation behind categorical cross-entropy employs:

$$Loss = -\sum_{i-1}^{m} y_i \cdot log(\widehat{y_i}) \qquad (5)$$

$\widehat{y_i}$ is the i-th scalar value in the model output, y_i is the corresponding target value, and *m* is the output size, which is the number of scalar values in the model output.

9. CIFAR-10 Dataset vs. CIFAR-100

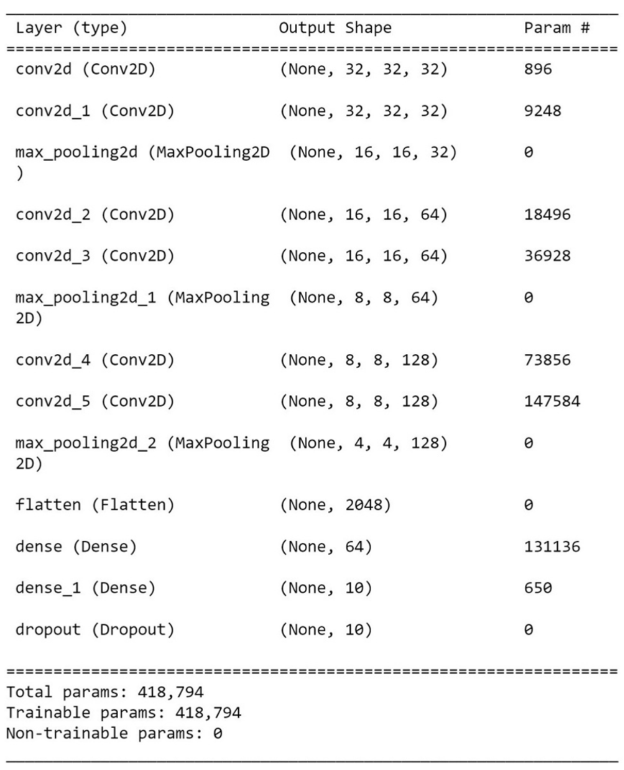
The CIFAR-10 dataset is a collection of 60,000 color images with size of 32x32 pixels, 50,000 of which are training images and 10,000 are testing images. The CIFAR-10 dataset consists of 10 classifications of the pictures within the dataset. In the CIFAR-100 dataset, there are 100 classes with 20 superclasses, to describe the classification of the classes.

Experimental Results

Comparing the CNN Sequential model and the RNN Functional model results using 250 epochs, and the first 2,500 samples in the CIFAR-10 dataset, the difference between the CNN and RNN is the architecture of the neural network. However, all functions used between the CNN and RNN remained the same, i.e. ReLU activation, RMS propagation, categorical cross-entropy, global average pooling, and dropout functions.

The reason behind the size restriction on the number of images is for time constraints. In both the CNN sequential model and the RNN functional model, there are spikes in results. This is normal as the dropout function will inherently cause inexperienced nodes to be forcibly trained. The CNN model does not show significant spikes in accuracy, but in the RNN, spikes are shown in consistent loss in accuracy at certain intervals. Significant loss in accuracy and rises in loss is due to dropout functions coinciding with skip connections in which nodes that are taken out of an destined layer causes inexperienced nodes to fill the position. Thus causing incorrect answers from the neural network.

1. CNN Sequential Model Results



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 64)                131136

 dense_1 (Dense)             (None, 10)                650

 dropout (Dropout)           (None, 10)                0

=================================================================
Total params: 418,794
Trainable params: 418,794
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: Summary of CNN sequential model

The Convolutional Neural Network, after 250 epochs and using the first 2,500 files from the CIFAR-10 dataset, and a batch size of 64, shows that the CNN's training loss in Figure 5 converges to zero very quickly. Comparing the testing data with the training data, the testing data increases in loss. In the Accuracy Model, the training data converges to 0.9 and not 1. This may be a consequence of a singularity, as the Accuracy Model suffers from the "Plateau Phenomenon". In the testing data, the accuracy stagnates quickly between 0.4 and 0.5, meaning that the test data is around 45% accurate.

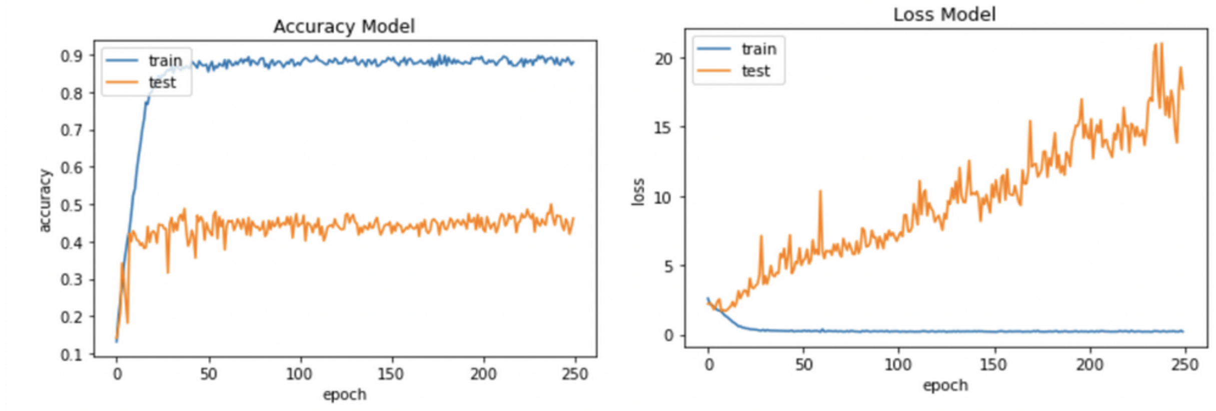
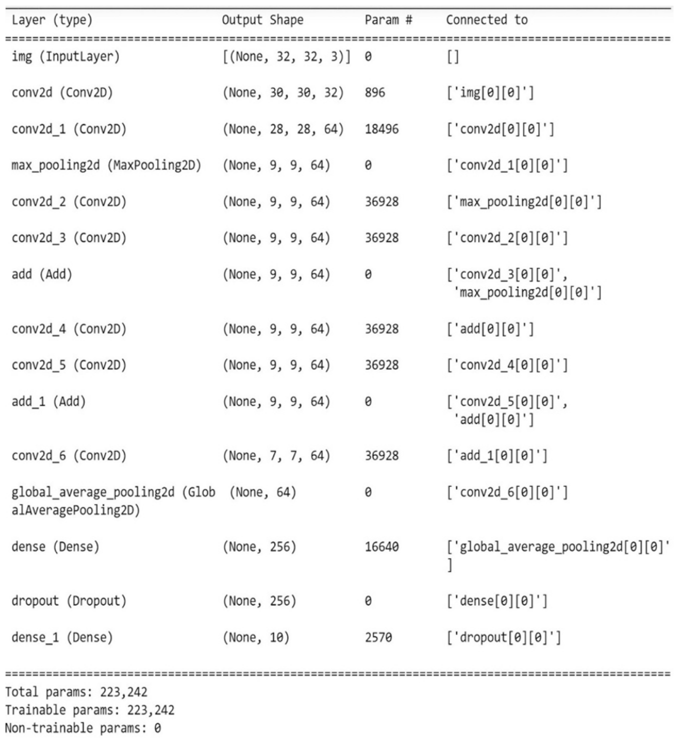


Figure 5: Accuracy and loss of the convolutional network during training and testing

2. RNN Functional Model Results



```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
img (InputLayer)                [(None, 32, 32, 3)]  0           []

conv2d (Conv2D)                 (None, 30, 30, 32)   896         ['img[0][0]']

conv2d_1 (Conv2D)               (None, 28, 28, 64)   18496       ['conv2d[0][0]']

max_pooling2d (MaxPooling2D)    (None, 9, 9, 64)     0           ['conv2d_1[0][0]']

conv2d_2 (Conv2D)               (None, 9, 9, 64)     36928       ['max_pooling2d[0][0]']

conv2d_3 (Conv2D)               (None, 9, 9, 64)     36928       ['conv2d_2[0][0]']

add (Add)                       (None, 9, 9, 64)     0           ['conv2d_3[0][0]',
                                                                  'max_pooling2d[0][0]']

conv2d_4 (Conv2D)               (None, 9, 9, 64)     36928       ['add[0][0]']

conv2d_5 (Conv2D)               (None, 9, 9, 64)     36928       ['conv2d_4[0][0]']

add_1 (Add)                     (None, 9, 9, 64)     0           ['conv2d_5[0][0]',
                                                                  'add[0][0]']

conv2d_6 (Conv2D)               (None, 7, 7, 64)     36928       ['add_1[0][0]']

global_average_pooling2d (Glob  (None, 64)           0           ['conv2d_6[0][0]']
alAveragePooling2D)

dense (Dense)                   (None, 256)          16640       ['global_average_pooling2d[0][0]'
                                                                 ]

dropout (Dropout)               (None, 256)          0           ['dense[0][0]']

dense_1 (Dense)                 (None, 10)           2570        ['dropout[0][0]']

==================================================================================================
Total params: 223,242
Trainable params: 223,242
Non-trainable params: 0
```

Figure 6: Summary of RNN functional model

The Residual Neural Network displayed staggering results with the same amount of epochs. the same 2,500 sets of data that the CNN used, and a batch size of 64. Unlike the loss model in the CNN, the RNN loss model is shown to utilize skip connections to prevent overfitting. In the accuracy model, the accuracy varies between .9 and 1 in the training data, thus outperforming the CNN in both Accuracy and Loss. In the Accuracy Model, testing data shows to perform in the high 40% to low 50%. While the average is also somewhere around 45%, the RNN does outperform in test data compared to the CNN by a negligible margin.

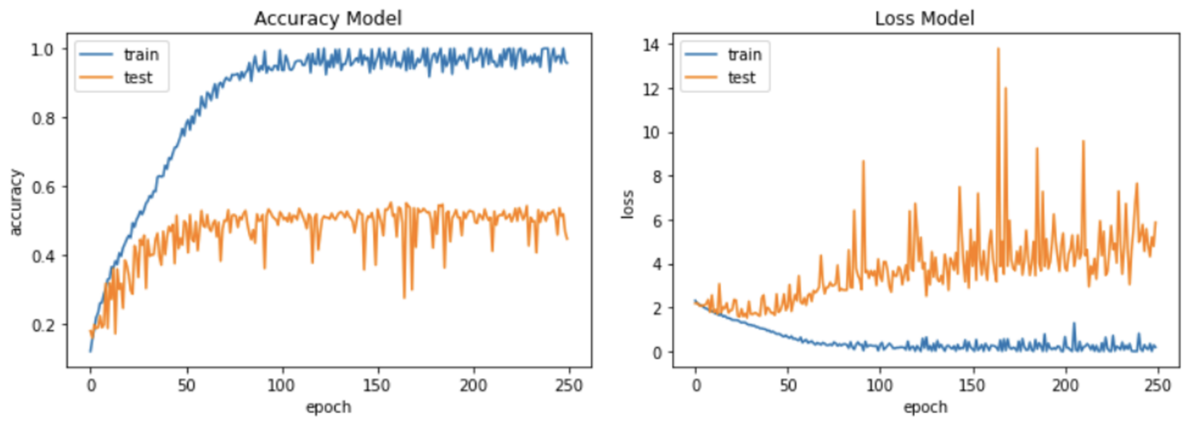


Figure 7: Accuracy and loss of the residual network during training and testing.

Discussion

In the experiments, the dropout functions wildly stagger the accuracy and loss. In our research, there were some considerations on which functions to use for the neural network, such as using Adam instead of RMSProp, or creating the CNN in a non-sequential format. We have decided that due to the computational strain of Adam, RMS propagation was the better choice, as the loss convergence in RMSProp is faster than Adam. CNN is normally built with a sequential model, thus we used the sequential model as a basic comparison to a modified CNN in which it implements skip connections (RNN).

While the convergence rate of the CNN is much faster than the RNN, the CNN suffers from the "plateau phenomenon" in which the improvements in learning do not occur in later epochs. The plateau phenomenon is shown in Fig. 5 Loss Model. The training data does not show any movement of loss after a sudden decrease at the beginning of the run. The RNN in Fig. 4 shows a similar result, but does show spikes of increase in the loss model.

Both neural networks show to have unusual spikes in both accuracy and loss for both the testing and training data. It is most apparent in the test model. Due to training in dropout functions combined with skip connections, probabilities with skipping both a trained node in an unskipped layer can show that an untrained node will provide a serious detriment to accuracy and training.

References

Bohra, Yash. (2021, June 18). The Challenge of Vanishing/Exploding Gradients in Deep Neural Networks. Analytics Vidhya. Retrieved March 2022 from https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks

Brownlee, Jason. “A Gentle Introduction to the Rectified Linear Unit (ReLU).” *Machine Learning Mastery*, 20 Aug. 2020, https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Chollet, Francois. “Keras Documentation: The Functional Api.” *Keras*, 12 April. 2020, https://keras.io/guides/functional_api/

Chollet, Francois. “Keras Documentation: The Sequential Model.” *Keras*, 12 April. 2020, https://keras.io/guides/sequential_model/

Glorot, Xavier, Bengio, Yoshua. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, PMLR 9:249-256. Retrieved March 2022 from https://proceedings.mlr.press/v9/glorot10a.html

Goodfellow, I., Bengio, Y., & Courville, A. (2017) *Deep Learning*. MIT Press.

He, Kaiming, et. al. (2015, December 10). Deep Residual Learning for Image Recognition. Retrieved March 2022 from https://arxiv.org/abs/1512.0338

Huang, Jason. “RMSPROP.” *RMSProp - Cornell University Computational Optimization Open Textbook - Optimization Wiki*, 12 Dec. 2020, https://optimization.cbe.cornell.edu/index.php?title=RMSProp#RMSProp

Krizhevsky, et. al. (2017, June). ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the ACM,* 60(6): 84-90. Retrieved March 2022 from https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Maklin, Cory. “Dropout Neural Network Layer in Keras Explained.” *Medium*, Towards Data Science, 22 July 2019, https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab

Monti, R.P., Tootoonian, S., Cao, R. (2018). Avoiding Degradation in Deep Feed-Forward Networks by Phasing Out Skip-Connections. In: Kůrková, V., Manolopoulos, Y., Hammer, B., Iliadis, L., Maglogiannis, I. (eds) Artificial Neural Networks and Machine Learning – ICANN 2018. ICANN 2018. Lecture Notes in Computer Science(), vol 11141. Springer, Cham. https://doi.org/10.1007/978-3-030-01424-7_44

Orhan, Ermin A., Pitkow, Xaq. (2017, January 31). Skip Connections Eliminate Singularities. *arXiv.* Retrieved March 2022 from https://arxiv.org/pdf/1701.09175v8.pdf

Wei, Haikun, et al. (2008). Dynamics of Learning Near Singularities in Layered Networks. *Neural Computation* 20, 813–843. Retrieved from https://bsi-ni.brain.riken.jp/database/file/285/291.pdf