Large Scale Log Analytics through ELK

Marcel den Reijer, mreijer@os3.nl

Supervised by:
Technical Supervisor/Architect: Patrick Beitsma
Functional Supervisor/Stakeholder: Tom van der Horst
Team Supervisor/Product Owner: Sjaak van Wijk
General Supervisor/UvA Liason: Sander Ruiter
Security- and Privacy Officer: Harrold van Aalderen

Vancis
Research project 2: University of Amsterdam

July 12, 2018

Abstract

Analyzing large volumes of log events without some kind of classification is undoable nowadays due to the large amount of events. Using AI to classify events make these log events usable again. With the use of the Keras Deep Learning API, which supports many Optimizing Stochastic Gradient Decent algorithms, better known as optimizers, this research project tried these algorithms in a Long Short-Term Memory (LSTM) network, which is a variant of the Recurrent Neural Networks. These algorithms have been applied to classify and update event data stored in Elastic-Search. The LSTM network consists of five layers where the output layer is a Dense layer using the Softmax function for evaluating the AI model and making the predictions. The Categorical Cross-Entropy is the algorithm used to calculate the loss. For the same AI model, different optimizers have been used to measure the accuracy and the loss. Adam was found as the best choice with an accuracy of 29,8%.

Keywords

Artificial Neural Networks, Recurrent Neural Networks, Long Short-Term Memory, Elastic-Search, Lambda, Deep-Learning, Keras, ELK-stack

Contents

1 Introduction

Nowadays a lot of hardware, software, operating systems, services, (end-user) applications, etc. generate all kind of events with different priorities, goals and intended audiences[1]. In the context of this research, an event is an occurrence recognized by software, Operating Systems, etc. All these events are relevant in their own right, but one cannot look at all these individual events, which are recorded at many different places by default[1]. Therefore, a centralized event collection environment collects all these events, coming from log files or message queues for example, but that also means an even larger amount of events to be evaluated at one place[1]. To be able to make use of all the available data and turn it into actionable information, there is a need for Artificial Intelligence (AI) that analyzes events and evaluates them.

The ELK-stack is one way that may be used for storing centralized events. The ELK-stack consists of three elements. The first element is Elastic-Search (ES), which is the noSQL/object database where every event is stored. The second element is Logstash, which is responsible for processing single events into more structured pieces and store it in ES. The third element is Kibana, which is a User Interface for displaying predefined graphs, dashboard and also allow ad-hoc querying the database.

In order to analyze the events with AI, the events should first be processed in a scalable way. One way to process a huge amount of events can be done with a Lambda architecture[2]. The lambda architecture consist of three layers. These layers are shown in figure 1. and are defined as Batch Layer, Speed Layer and Serving Layer[2].

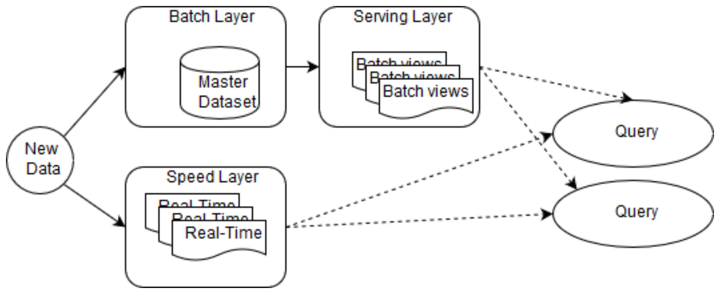

Figure 1: Lambda architecture[3]

According to Numnonda the Batch Layer is responsible for getting raw datasets and pre-computing batch views. The Speed Layer is responsible for creating real-time view from new data and the Serving Layer is responsible for responding to ad-hoc queries by returning pre-computed views[3].

1.1 Problem statement

Within the ELK-stack lots of different kinds of AI modules can be installed, but it is desirable to place an AI module on a dedicated system. Issues may arise when one wants to change the AI model in layers, algorithms or interfaces. It will be hard to accomplish this and takes too much time, because it requires diving into the code if it is open-source. When an AI module is installed on the ELK-stack, the environment is less modular. Assumable, when there is a moment the ELK-stack will be replaced by i.e. Hadoop, the model should learn again. By placing the AI module on a dedicated system, the AI module should only be interrupted and one has only to change the interfaces.

This research focuses on to design an implementation of AI.

1.2 Research Questions

Based on the introduction, the following research question is defined:

- *Which way of applying Artificial Intelligence in the form of Machine Learning (ML)/Deep Learning (DL) can be used to find relevant actionable information from event data suitable for use within the batch layer in a Lambda Architecture?*

This main research question is divided into the following sub-questions:

  - Which DL model fits for this type of data?
  - What should be the time-period to analyze, in order to optimize relevant context but also to collect events as near-realtime as possible?
  - How is the accuracy of the DL model calculated?
  - How can the model be tuned during operations?

1.3 Approach

The following steps are taken in this research. The first step will be define and discuss which algorithms that will be used in this research. The second step explain the experimental setup. This experimental setup will be proposed as implementation. During the third step, a prototype of the experimental setup will be made. The last step will be discuss the results and answer the research question.

1.3.1 Keras Deep Learning API

For this research, the Keras Deep Learning API will be used. Keras is a high-level implementation of the Artificial Neural Network (ANN), written in Python. Keras is capable to run on top of TensorFlow, CNTK or Theano (low-level DL libraries). The idea behind the Keras project is to focus on enabling fast experimentation[4]. The first step of this project is to define the AI model. The goal of this step is to define how many layers and which layers will be used in the AI model. The second step is to compile the model. This step defines the algorithm of how the learning rate in AI should be.

Define AI model:
Models in Keras are defined as a sequence of layers and two types of models are supported. The first is the sequential model and the second is the functional model. The difference according to Lin is that the sequential model allows one to create each model layer-by-layer at the time. Creating models for sharing layers, as well as having multiple inputs or outputs, is not allowed [5]. The property of the Functional model is that layers may be connected to more than just previous or next layers[5].

Compile the model:
After defining the model and before training the model, the model needs to know how to learn. With the Keras API the compiler needs two arguments, called optimizer and loss[4]. Rajarshee Mitra defines the optimizer as how the learning rate should be treated[6]. Keras support many Optimizing SGD algorithms, for example:

- Stochastic Gradient Decent (SGD)
- Root Mean Square Propagation (RMSProp)
- Adaptive Gradient Algorithm (AdaGrad)

- Adaptive Delta Algorithm (AdaDelta)
- Adaptive Moment Estimation (Adam)

These algorithms are explained in a mathematical way in section 7.

2 Related work

A lot of research has been done on applying ML/DL on large data sets. The first study discussed here is by Ruders in June 2017. In his research he made an overview of gradient descent optimization algorithms[7]. He looked into three Gradient Descent Algorithms (GDAs) in his article. These algorithms are Mini-Batch Gradient Descent (MBGD), Batch Gradient Descent (BGD) and SGD. In his article he also compared optimizing SGD algorithms. Ruder shows that the Optimizing SGD algorithms RMSprop, Adadelta and Adam are very similar. Adam adds bias-correction and momentum to RMSprop[7]. Adagrad is a gradient-based optimization algorithm with the following properties[7]: adapting the learning rate to the parameters; possibly performing smaller updates for frequent parameters; and possibly performing larger updates for infrequent parameters.

A second related work is done by Scherer in 2017. He performed a study for applying Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNN) on text for classifications[8]. He claimed that RNN are increasingly used for classifications of text. Beside applying LSTM on text for classifications, he did also compare different variants of LSTM i.e. BLSTM and GRU[8]. He claimed also that Gated Recurrent Unit (GRU) can be used for fast training because the require less Epochs (Epochs will be discussed in section 6.). LSTM and BLSTM networks are more precise, but more time is needed for an higher accuracy[8].

The third related work is done by Du, C. & Huang L. in February 2018. They say that text classification is one of the main tasks of ML[9]. In their paper a bi-directional RNN is proposed. This algorithm is based on the neural network attention mechanism[9].

3 Neural Networks

In this section information will be given of ANN and LSTM, which is a variant of RNN.

3.1 Artificial Neural Network (ANN)

The first class of ANN was Feedforward Neural Networks. This class of Neural Networks is called feedforward, because data will only send forward in the network without loops or cycles[10]. The definition of ANN accordingly to Zhang is as follows[11]:

"Artificial neural network is a structure of interconnected units of large number of neurons. Each neuron in the network is able to receive input signals, to process them, and to send an output signal. It consists of a set of the weighted synapses, an adder for summing the input data weighted by the respective synaptic strength, and an activation function for limiting the amplitude of the output of the neuron."

In other words, the neural network takes a value as input data at each layer and multiplies this input by weights[12]. The first weight which is set in the network is given by a random number generator[13]. Figure 2. shows a Multilayer Feedforward Neural Network[12]. After processing the input data of the input layer, the results are sent to other layers and these layers are called hidden layers. Hidden layers are layers between the input and output layer. Next, a non-linear function is applied, which is also known as activation function[12]. The activation function should determine the resulting values into a value between one and zero. This value can be seen as the probability of a certain layer. The calculation of the logistic function used is Softmax[12]. The math behind Softmax comes from the probability theory and is explained later on.

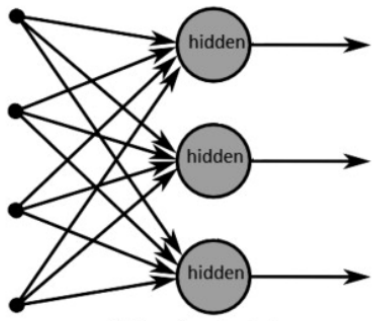

Figure 2: Artificial Neural Network; *Source: [14]*

3.2 Recurrent Neural Network (RNN)

Recurrent Neural Networks abbreviated as RNN is one of the many ANN classes. RNN was introduced to improve certain limitations of the feedforward Neural Network Language model. One of these limitations is the need of specifying the length of the context[15]. Figure 3. shows an example of RNN. As the figure shows RNN connects the hidden layers back to itself using time-delayed connections[15]. According to Hassan & Mahmood, RNN allows formulating types of short-term memories. The values of the hidden layers are based on the current input data, weights and the state values of the previously hidden layer[16]. The same input to a current layer may lead to a different result, because the weight of this hidden layer changes constantly with each new data input[17].

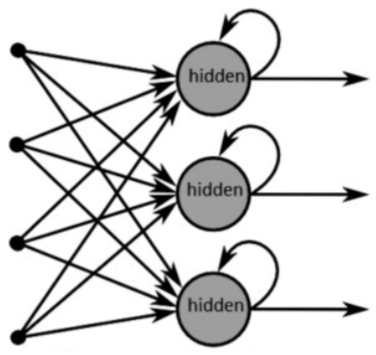

Figure 3: Recurrent Neural Network; *Source: [14]*

3.3 Long Short-Term Memory (LSTM)

RNN has many varieties. One of these varieties is the LSTM model. The predictions in RNN work in a chain (see figure 4.). The hidden layer from one prediction is the hidden layer of the next prediction. In this way a memory to the recurrent net (RNN) can be assigned and the results of earlier estimation may improve future predictions[15].

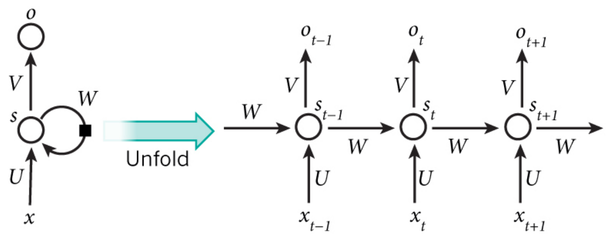

Figure 4: RNN sequence; *Source: [16]*

The issue with this chain-like fashion is that the result of older weights at the start of the chain will eventually fade; this issue is called the vanishing gradient problem[18]. LSTM tries to solve this problem by implementing gate structures[18]. Figure 5. shows that an LSTM memory cell consists of four units, namely the input gate, output gate, forget gate and memory cells[15]. The input gate is the interface of adding new data[19]. The output gate is the outgoing interface. The output gate is responsible for selecting useful information from the current cell state and showing it out as an output[19]. The forget gate is the interface of forgetting information[19].

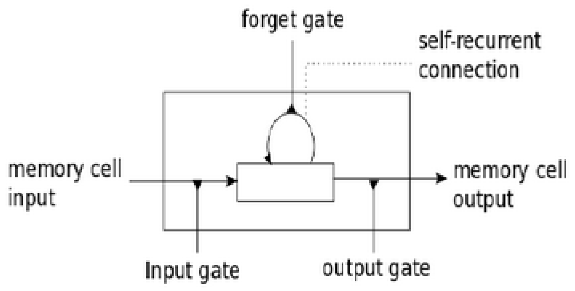

Figure 5: LSTM memory cell; *Source:[15]*

The math behind LSTM is as follows[15]: The first step is to calculate the values i_t and $\tilde{c}_t$ where i_t is the input gate and $\tilde{c}_t$ is the candidate value for the states of the memory cell at time t:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$
$$\tilde{c}_t = tanh(W_i x_t + U_i h_{t-1} + b_c)$$

The second step is to calculate the value f_t, which is the activation function of the memory cells forget gate at time t.

$$f_t = \sigma(W_f x_t + U_f h_{t-1}) + b_f$$

The third step is to compute c_t, which is the new state at time t.

$$c_t = i_t \cdot \tilde{c}_t + f_t \cdot c_{t-1}$$

The fourth step is to calculate the value of the output gates and subsequently, their outputs[15].

$$o_t = \sigma(W_o x_t + U_i h_{t-1})$$
$$h_t = o_t \cdot \tanh(c_t)$$

The resulting sequence from recurrent layer $h_1, h_2, ...h_t$ is calculated after calculating the memory cell and the output gate[15]. Where t is the length of sequence to the layer. In order to get better results, the average of the stage of the hidden layers will be calculated over a sum of sentences[15].

4 Dense layer vs Dropout layer

Besides defining a layer as LSTM, a layer could also be defined as Dense or as Dropout. A Dense layer is when all neurons in a certain layer are connected to all neurons in the next layer[20]. The left picture of Figure 6. shows an example of Dense layers. When a neuron has n inputs and m outputs, then $n \cdot m$ are the total amount of connections[21]. Dense layers may lead to many connections between neurons.
Defining a hidden layer as Dropout layer will reduce or avoid overfitting on neural networks[22]. Dropout is a technique for selecting and removing random neurons on a certain layer during training (see Fig. 6)[23]. This result in generalization, such that the certain layer has to learn the same ”concept” from different neurons[23]. The Dropout will be deactivated during the prediction of the values of a neuron[23].

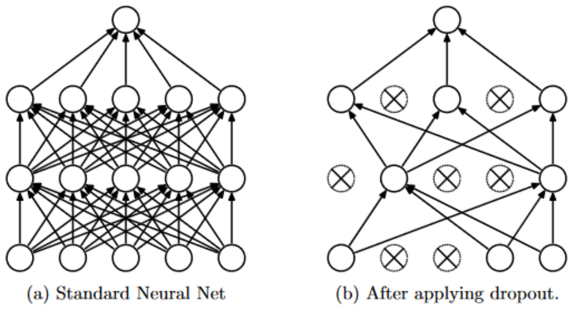

Figure 6: Dropout layer; *Source: [22]*

5 Activation function and loss function

In this section activation functions and loss functions are discussed. According to Dorst L. is an activation function an math function, which specifies the output of a neuron to a given input[24]. There exists a lot of different kind of activation functions and in this research the Softmax function is used. According to Changhau I. is the loss function a function used to measure the inconsistency between predicted value and actual label[25].

5.1 Categorical Cross-entropy

The Cross-entropy in Neural Networks is used as a loss function, which has in the output Softmax activations[26]. Cross-entropy is an alternative of Mean Squared Error (MSE). MSE tries to measure the average of the squares of the errors[26]. According to Dahal, P. Cross-entropy indicates the distance between what the model output distribution should be, and what the original distribution really is[26]. Cross-entropy is calculated with the following equation:

$$H(y,p) = -\Sigma_i y_i \text{log}(p_i)$$

A distribution or probability distribution in probability theory is a math function, which results in a list. This list shows the possible values of a random variable and the associated probability[27].

5.2 Softmax

According to Polamuri S. Softmax is an activation function for calculating the probability distribution of each target class over n different/possible target classes for the given inputs[28]. By using a "multi" classification AI model, The Softmax function will return the probabilities for each class[28]. One advantage according to Polamuri of this function is that this output probability range has a range between 0 and 1[28]. The sum of the probability range will be equal to one[28]. The math behind Softmax is as follow:

$$P_j = \frac{e^{a_i}}{\Sigma_{k=1}^{N} e_k^a} \text{for i} = 1, ..., K.$$

6 Epochs vs Batch size

An epoch is the phenomena when the entire data set went through the algorithm a number of times[29]. An epoch will be completed, each time

when the algorithm has seen all samples in a certain dataset[29].
The Batch size is the amount of test or training examples in one forward or backward pass. One property of the Batch size is that it is loaded in RAM memory. So, more RAM memory is needed, when a higher batch size is used[29]. After each batch, the weights will be updated[29].

7 Optimizers

The optimizers as mentioned in the Approach of this paper will be described in this sub-section. This section focuses on the math of each optimizer.

7.1 SGD

SGD is an algorithm, which updates each parameters for each training example $x^{(i)}$ and label $y^{(i)}$. The math behind this optimizer as follow[7]:

$$0 = 0 - n \cdot \Delta_0 J(0; x^{(i)}; y^{(i)})$$

7.2 Adam

According to Ruder RMSprop, Adadelta and Adam are very similar, but differ in a way such that Adam adds bias-correction and momentum to RMSprop. According to Kingma bias-correction helps Adam to slightly outperform RMSprop towards the end of optimization as gradients become sparser[7]. The math according to Ruder behind Adam is[7]:

We compute the decaying averages of past and past squared gradients m_t and v_t respectively as follows[7]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

They counteract these biases by computing bias-corrected first and second moment estimates[7]:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

They then use these to update the parameters just as we have seen in Adadelta and RMSprop, which yields the Adam update rule[7]:

$$0_{t+1} = 0_t - \frac{n}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

7.3 RMSprop

RMSprop is, according to Ruder, an unpublished adaptive learning rate method proposed by Geoff Hinton[7]. The math behind this optimizer is as follows[7]:

$$E[g^2]_t = 0.9E[g^2]_{t1} + 0.1g_t^2$$
$$0_{t+1} = 0_t - \frac{n}{\sqrt{E[g^2]_{t+\epsilon}}} g_t$$

Ruder claims that Hinton has suggested that the learning rate value should be set to 0.9, while Ruder claims that the learning rate value should be set to 0.001, without any explanation[7].

7.4 Adagrad

Adagrad is just like the other optimizers an Optimizing SGD algorithm with three properties[7]. The first property is that Adagrad adapts the learning rate to parameters. The second property of Adagrad is that it may perform smaller updates for parameters associated with frequently occurring features. the third property is that Adagrad may perform larger updates for parameters associated with infrequent features[7]. The math is as follow:

$$g_{t,i} = \Delta_0 J(0_{t,i})$$

The SGD update every parameter 0_i at each time step t, where n is the learning rate.

$$0_{t+1,i} = 0_{t,i} - n \cdot g_{t,i}$$

According to Ruder is $G_t \in R^{d \cdot d}$ a diagonal matrix, where each element i, i is the sum of the squares of the gradients w.r.t. $_i$ up to time step t[7].

$$0_{t+1,i} = 0_{t,i} - \frac{n}{\sqrt{G_{t,ii+\epsilon}}} \cdot g_{t,i}$$

8 Improve performance

In order to tune the AI model during operations, Brownlee has described four methods in his article[30]. Tuning the model is needed when the accuracy is lower than it should be. The four methods to improve the performance are: add data, improve algorithms, tune algorithms and improve with ensembles. Improve algorithms, tune algorithms and improve with ensembles cannot be done during operations, because the module should be interrupted. Add new data into the model can, because new data are added in a certain period. Brownlee describes three relevant methods to add new data[30].
The first way is by getting more data. The quality of the models may be in proportions to the quality of the training data, but it is not for sure that it will improve the model[30].
When getting more data is not possible, then getting more data is a limited way to train the model. In this case, one can invent data; this is the second way to improve the performance of the model. An example is generating random "fictional" data. This is also called data augmentation or data generation[30].
The third way is rescaling the data to the boundaries of the activation function. This is a way to improve the performance by normalizing the value in to output layer[30].

9 Experimental setup

In this section, the experimental setup will be as architecture proposed and will be explained in two subsections. The first subsection gives a more detailed explanation about the ELK-stack. The second subsection is the most important part. This part gives an explanation about the implementation of the Machine Learning MOdule (MLM) and AI is placed. The third part is an explanation about the implemented layers in the second part.

9.1 Implementation ELK-stack

The first part of Figure 7. shows a "high-level" implementation of the ELK-stack added with an AI module called MLM. All events from all nodes are sent to Logstash over the Syslog protocol. Logstash understands the message format of this protocol with regular expressions and cuts the event into smaller pieces. The message format of Syslog according to RFC-5424 is as follow:

priority|version|timestamp|hostname|program|pid|message

One can decide how many pieces the message should be cut. It will not change the result, because the AI model will filter for the right information. After cutting the message into smaller pieces, the data is sent to ES. is nothing more than an object-based database and every component may query for information and ES gives a response. Kibana queries for information and plots the response of the query to a graph in a Dashboard. The same holds true for Nagios, only the difference is that Nagios triggers on certain events.

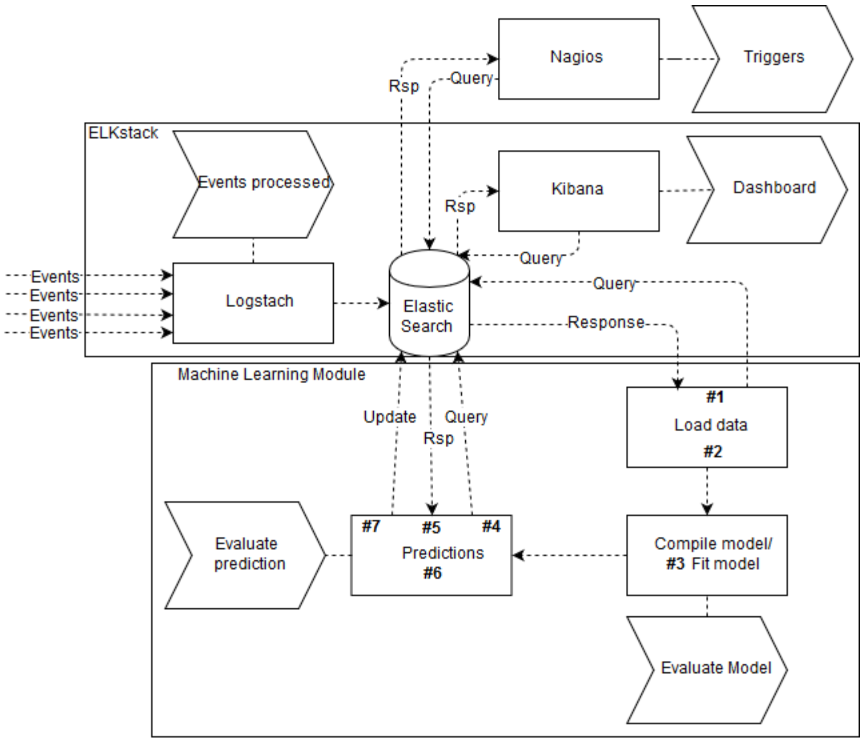

Figure 7: Machine learning architecture on ES

9.2 Implementation MLM

This subsection described the MLM. This module consists of three components called, Load data, Compile/Fit model and Predictions. The component Load data takes care of getting "test or train" data and parse it to numerical values. The component Compile/Fit model takes care of creating the AI model. The last component is called predictions and this component takes care of the predictions of unlabeled data and the implementation of the Lambda architecture. In such a way, a batch of data will be retrieved in a certain period. The prediction of this batch of data will be evaluated. Figure 7. shows numbers in each component, referring to a section in Appendix A, which shows the python script of the Proof of Concept.

9.2.1 Load data

As briefly described in the introduction of this subsection, the component Load data takes care of retrieving labeled data with ”AI-test” (to get test data) or ”AI-trained” (to get trained data). This component consists of two parts. The first part queries for ”test/training” data. The second part parsed the data into numerical values; in such a way Keras can compile and fit the model. Every row in the logs is filtered on Hostname, Syslog message and the labeled tag. This filtered information is put into an empty array separated by a new line. The queried data is parsed to numerical values over an X and Y axes. At this stage, unique words, the length of the data and the amount of patterns are calculated.

9.2.2 Compile/ fit model

The second component defines the layers, optimizers and other algorithms used in the model. Figure 8. shows the layers. The reason to put a Dropout layer after each LSTM layer is to prevent overfitting, because in a live production environment there will be a huge amount of data. The Softmax function is placed in the Dense layer, for the highest calculation of the accuracy per Epoch and the model as a whole. The whole AI model will be equipped with all the supported optimizers each one at a time. Every LSTM layers will get 256 neurons. The Dropout layers will use only 20% of them. The loss function algorithm will be categorical Cross-entropy. The whole model will be tested on 20 and 40 Epochs and the batch size stays on 256 batch.

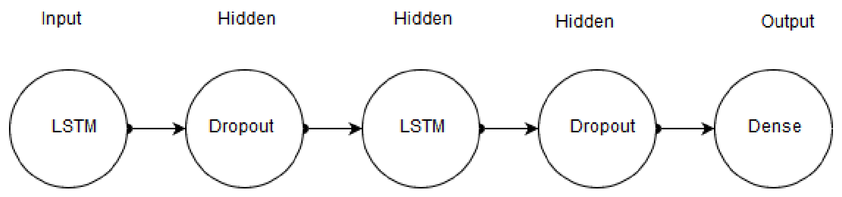

Figure 8: AI layers

9.2.3 Predictions and Lambda implementation

The last component takes care of implementing Lambda and making a prediction about unlabeled data. After a model has been created, this component queries for unlabeled data. Unlabeled data are events, at the moment they are processed by Logstash and stored in ES. After querying unlabeled data, this component parses the data to numerical values and puts it through the prediction function. The predicted value will be between 0 and 1. By putting the predicted value to a variable and matching to a threshold, an update query will be sent to ES with an "AI-test" or "AI-train" label.

10 Results

The results of the learning rates applied on the proposed architecture are shown in the figures below. Figure 9. shows the accuracy of the optimizing SGD algorithms, which Keras supports. It seems that with 20 Epochs in a model Adam and RMSprop have the best accuracy per Epoch. The higher the accuracy the better. Figure 10. shows the loss per Epoch. The lower the loss the better. It seems that SGD, Adadelta and Adagrad have the highest loss. This means that Adam and RMSprop are the best choices to use in an AI model with 20 Epochs, both in terms of accuracy and of loss.

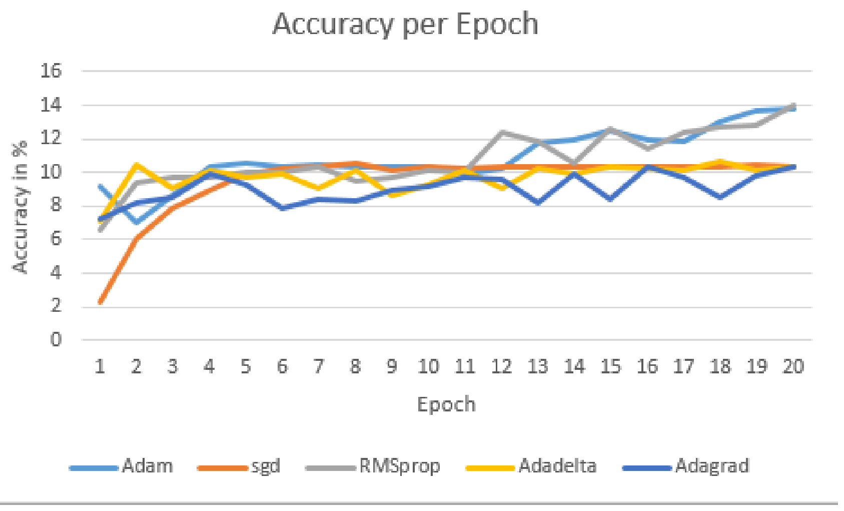

Figure 9: Accuracy per 20 Epochs

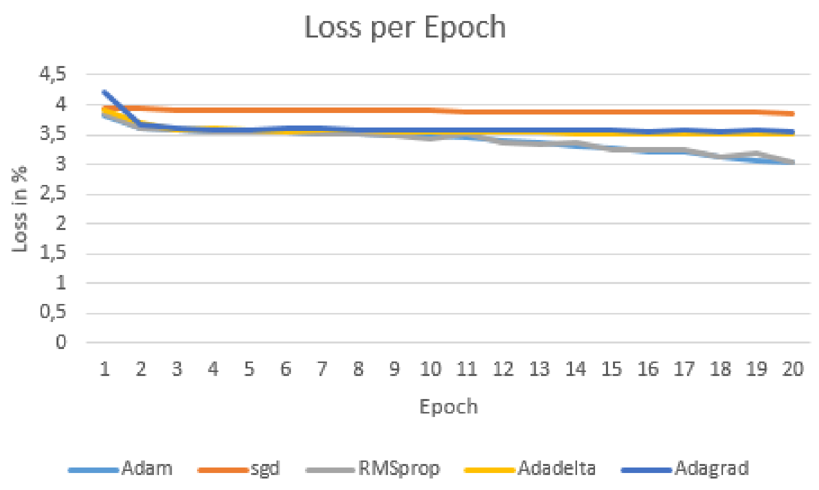

Figure 10: Loss per 20 Epochs

When 40 Epochs are used inside a model, it seems that Adam and Adagrad are the best choices to use. Figure 11. shows that Adadelta and SGD have to lowest accuracy. Figure 12 shows that SGD has the highest loss. Adam, Adagrad and RMSprop have the lowest. In this situation, Adam or Adagrad is the best choice to use.

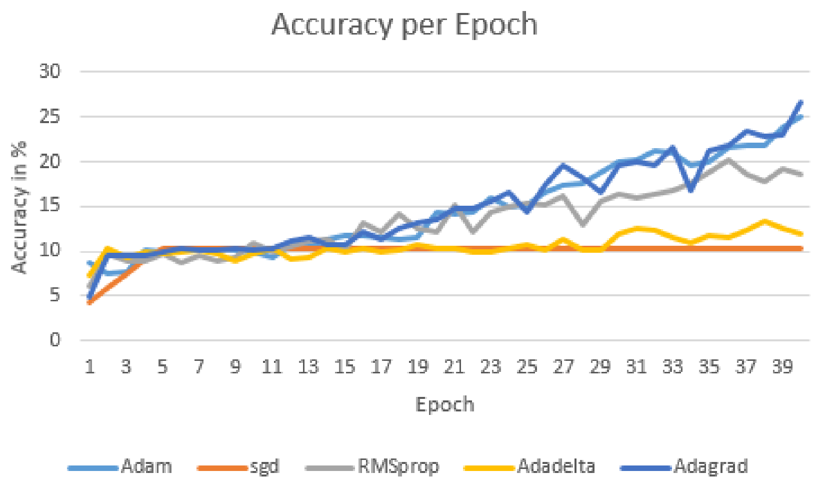

Figure 11: Accuracy per 40 Epochs

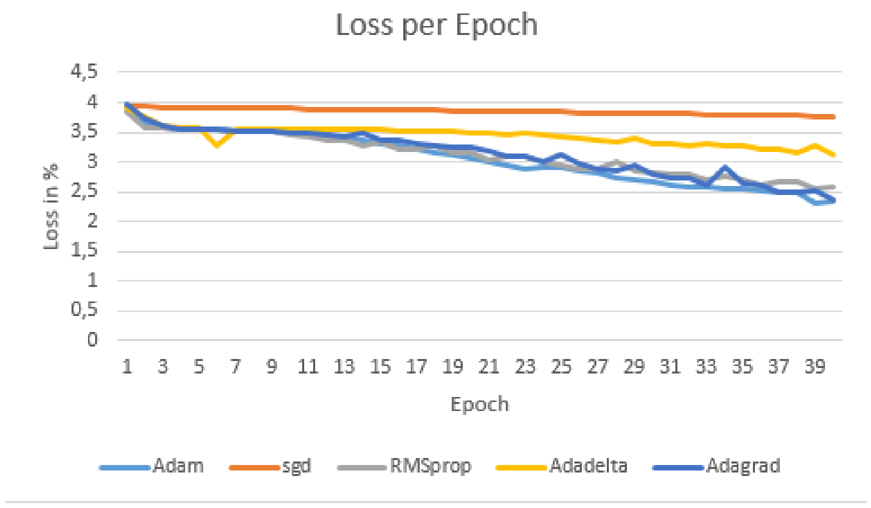

Figure 12: Loss per 40 Epochs

Table 1. shows an overview of the accuracy and loss of each model as a whole. From this table, Adam will be the best choice when 20 or 40 Epochs are chosen in a model. The accuracy is calculated with the Softmax algorithm as described in technical background and proposed architecture.

	20 Epochs		40 Epochs	
	Loss	Accuracy	Loss	Accuracy
Adam	2.97%	16.36%	2.22%	29.82%
SGD	3.86%	10.40%	3.76%	10.40%
RMSprop	3.00%	5.13%	2.74%	19.42%
Adadelta	3.45%	10.55%	3.12%	13.82%
Adagrad	3.53%	10.40%	2.29%	27.27%

Table 1: Total Accuracy and Loss per model as a whole

11 Discussion

In order to select the best optimizer, the accuracy and loss were considered. Based on these criteria, the optimizer Adam performed well for both 20 and 40 Epochs. Therefore, Adam is the recommended choice for optimization, however it will not guarantee that the output result of each event is correct. The model needs to be trained. The accuracy is very low in contrast with the research of Du, C. & Huang L.[9], but the model can be tuned by the methods as discussed in section 8. This research has not investigated or shown any results of the predictions itself per new event, which is initially unlabelled. Ruder claimed that the following optimizers Adam, RMSprop and Adadelta perform well in similar circumstances, however the results of this research shows that Adam, RMSprop and Adagrad perform similar circumstances.

12 Conclusion

The results are based on the AI model described in section 10 Architecture with different Optimizing SGD algorithms. It seems that RMSprop, Adagrad and Adam are the best choices to use, to achieve the highest accuracy in the experimental setup. Adam is preferred, because this learning rate algorithm uses bias-correction and momentum. Adam also has the highest model accuracy of 29,8% over 40 Epochs. The second best choice to use is Adagrad, because this learning rate algorithm has a model accuracy of 27.3% over 40 Epochs. SGD and Adadelta are not recommended, because the accuracy does not increase and the loss does not decrease fast over time. RMSprThe Softmax function is used in order to calculate the accuracy and it may be used for predicting a classification of an event. The Softmax function is placed in the lowest layer of this model. Keras supports Softmax and Sigmoid. A comparison of these functions cannot be made, because as previously mentioned Sigmoid is used for binary classifications. In order to increase the accuracy of each model, Brownlee has described methods, but these are not evaluated in this research.

12.1 Future work

This study would benefit from follow-up research to improve the applicability of the AI model. For example, it would be interesting to increase the accuracy during operations and to measure it. Higher accuracy would make the model more robust. The AI model used in this research focuses on the

batch layer, but progress can be made by applying this model on the speed layer. So far, real-time evaluation of events by using this speed layer has not been done yet, but it would give new insights into how quick real-time will be in the proposed implementation.
Furthermore, different algorithms could be tested in combination with the AI model and these algorithms could be tuned increase the applicability. According to the supervisors of this project, an AI model may need months of test and training data to increase the accuracy, which was not performed in view of time limitations.

References

[1] S. Beaver, D. & Hutchison. Elasticsearch, Logstash and Kibana(ELK). https://resources.sei.cmu.edu/asset_files/Presentation/2015_017_001_431205.pdf, 2015.

[2] Anderson, K.M. Lambda architecture. https://pdfs.semanticscholar.org/presentation/48dd/94cb5c7c0f3134ea9af1ac8797354938fcd8.pdf, 2014.

[3] T. Numnonda. A real-time recommendation engine using lambda architecture. https://link-springer-com.proxy.uba.uva.nl:2443/article/10.1007/s10015-017-0424-8, 2017.

[4] Keras. Keras Deep Learning API Documantation. https://keras.io, 2018.

[5] J. Lin. Keras Models: Sequential vs. Functional. https://jovianlin.io/keras-models-sequential-vs-functional/, 2017.

[6] Rajarshee Mitra, B. What are differences between update rules like adadelta, rmsprop, adagrad, and adam? https://www.quora.com/What-are-differences-between-update-rules-like-AdaDelta-RMSProp-AdaGrad-and-AdaM 2017.

[7] S. Ruder. An overview of gradient descent optimization algorithms. http://ruder.io/optimizing-gradient-descent/index.html#parallelizinganddistributingsgd, 2018.

[8] Scherer, R. Lstm recurrent neural networks for short text and sentiment classication. https://www.researchgate.net/publication/

318018787_LSTM_Recurrent_Neural_Networks_for_Short_Text_and_Sentiment_Classification, 2017.

[9] Du, C. & Huang, L. Text classification research with attention-based recurrent neural networks. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&ved=0ahUKEwjpxai615bcAhWGWhQKHfKqCBMQFghLMAQ&url=http%3A%2F%2Funivagora.ro%2Fjour%2Findex.php%2Fijccc%2Farticle%2Fdownload%2F3142%2Fpdf&usg=AOvVaw02sgX96hAwVlOEbe14etjm, 2018.

[10] McGonagle, J. Feedforward neural networks. https://brilliant.org/wiki/feedforward-neural-networks/, 2018.

[11] Zhang, Z. Artificial neural networks. https://link.springer.com/chapter/10.1007/978-3-319-67340-0_1, 2017.

[12] Schmatz, S. How does softmax function work in AI field? https://www.quora.com/How-does-softmax-function-work-in-AI-field, year = 2014.

[13] Miller, S. Mind: How to Build a Neural Network (Part One). https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/, 2015.

[14] De Mulder, W., Bethard, S. & Moens, M.F. Deep learning applications and challenges in big data analytics. https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/, 2015.

[15] A. Hassan, A. & Mahmood. Deep learning for sentence classification. https://www.researchgate.net/publication/318975052_Deep_learning_for_sentence_classification, year = 2017.

[16] C. Godbout. Recurrent Neural Networks for Beginners. https://medium.com/@camrongodbout/recurrent-neural-networks-for-beginners-7aca4e933b82#.q5otnbm2i, year = 2016.

[17] Siraj Raval. How to Predict Stock Prices Easily - Intro to Deep Learning #7, 2017.

[18] Christopher Olah. Understanding LSTM Networks – colah's blog. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[19] Srivastava, P. Essentials of deep learning : Introduction to long short term memory. https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/, 2017.

[20] Raschka, S. In keras, what is a "dense" and a "dropout" layer? https://www.quora.com/In-Keras-what-is-a-dense-and-a-dropout-layer, 2016.

[21] Howard, J. Dense vs convolutional vs fully connected layers. http://forums.fast.ai/t/dense-vs-convolutional-vs-fully-connected-layers/191/2, 2016.

[22] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. . Dropout: A simple way to prevent neural networks from overfitting. https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf, 2014.

[23] dos Santos, L.A. Artificial inteligence. https://legacy.gitbook.com/@leonardoaraujosantos, 2018.

[24] Dorst, L. Neural activation functions. https://staff.science.uva.nl/l.dorst/math/sigma.pdf, unknown.

[25] Changhau, I. Loss functions in neural networks. https://isaacchanghau.github.io/post/loss_functions/, 2017.

[26] Dahal, P. Classification and loss evaluation - softmax and cross entropy loss. https://deepnotes.io/softmax-crossentropy, 2016.

[27] Tryfos, P. Probability and probability distributions. http://www.yorku.ca/ptryfos/ch2000.pdf, 2009.

[28] Polamuri, S. Difference between softmax function and sigmoid function. http://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/, 2017.

[29] Dernoncourt, F. & Hon, K. Epoch vs iteration when training neural networks. https://stackoverflow.com/questions/4752626/epoch-vs-iteration-when-training-neural-networks, 2015.

[30] Brownlee, J. How to improve deep learning performance? https://machinelearningmastery.com/improve-deep-learning-performance/, 2016.

13 Appendix A: PoC script

#1. Load training data from Elastic-Search

```
es = Elasticsearch([{'host': 'velk', 'port': 9200}])
vara=(es.search(index="*", body={"query": {"match" : {"AI-Test" : "1"}}}))
arrayt=[]
for row in vara["hits"]["hits"]:
    #b=(row["_id"])
    c=(row["_source"]["host"])
    d=(row["_source"]["message"])
    e=(str(row["_source"]["AI-Test"]))
    raw = (c + d + " AI-test " + e)
    arrayt.append(raw)
    #print (raw)
str1 = '\n'.join(arrayt)
print (str1)
```

#2. Enumerate characters from training data

```
raw_text = str1.lower()
chars = sorted(list(set(raw_text)))
char_to_int = dict((c, i) for i, c in enumerate(chars))
n_chars = len(raw_text)
n_vocab = len(chars)
print ("Total Characters: ", n_chars)
print ("Total Vocab: ", n_vocab)
seq_length = 100
dataX = []
dataY = []
for i in range(0, n_chars - seq_length, 1):
seq_in = raw_text[i:i + seq_length]
seq_out = raw_text[i + seq_length]
dataX.append([char_to_int[char] for char in seq_in])
dataY.append(char_to_int[seq_out])
n_patterns = len(dataX)
print ("Total Patterns: ", n_patterns)
X = numpy.reshape(dataX, (n_patterns, seq_length, 1))
X = X / float(n_vocab)
y = np_utils.to_categorical(dataY)
```

#3. Create AI model

```
model = Sequential()
model.add(LSTM(256, input_shape=(X.shape, X.shape[2]), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(256))
model.add(Dropout(0.2))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']
model.save('model.h5')
model.fit(X, y, epochs=20, batch_size=256)
score, acc = model.evaluate(X, y)
print('Test score:', score)
print('Test accuracy:', acc*100)
```

#4. New input data

```
while True:
    varb=(es.search(index="*", body={"query": {"match" : {"syslog_program" : "sshd"}}
    for rowa in varb["hits"]["hits"]:
        xa=(rowa["_index"])
        xb=(rowa["_id"])
        xc=(rowa["_source"]["host"])
        xd=(rowa["_source"]["message"])
        rawa = (xb + " " + xc + " " + xd)
        print (rawa)
```

#5. Enumerate characters from new input data

```
        rawa = rawa.lower()
        chars = sorted(list(set(rawa)))
        char_to_int = dict((c, i) for i, c in enumerate(chars))
        n_chars = len(rawa)
        n_vocab = len(chars)
        print ("Total Characters: ", n_chars)
        print ("Total Vocab: ", n_vocab)
        seq_length = 100
        dataX = []
        dataY = []
        for i in range(0, n_chars - seq_length, 1):
            seq_in = rawa[i:i + seq_length]
            seq_out = rawa[i + seq_length]
            dataX.append([char_to_int[char] for char in seq_in])
            dataY.append(char_to_int[seq_out])
        n_patterns = len(dataX)
        print ("Total Patterns: ", n_patterns)
        X = numpy.reshape(dataX, (n_patterns, seq_length, 1))
        X = X / float(n_vocab)
        y = np_utils.to_categorical(dataY)
```

#6. Create predictions

```
        #predict
        pred = model.predict(X)
        q, w = 0, 0
        preda= pred[q][w]*100
        predb= str(preda)
        print('Predicted:', "%", predb)
```

#7. Update Elastic-Search data

```
        #update ES field
        if preda < 80:
            print ("this might not be a threat")
            es.update(index=xa,doc_type='syslog',id=xb, body={"doc": {"AI-test": "0"}
            es.update(index=xa,doc_type='syslog',id=xb, body={"doc": {"AI-test-pred"
            predb }})
        else:
            print ("this this might be a threat")
            es.update(index=xa,doc_type='syslog',id=xb, body={"doc": {"AI-test": "1"
            es.update(index=xa,doc_type='syslog',id=xb, body={"doc": {"AI-test-pred"
            predb }})
    time.sleep(900) #15 min
```