Quantitatively understanding recurrent networks

Tycho van der Ouderaa
University of Amsterdam
tychovdo@gmail.com

Abstract

Recurrent Neural Networks (RNNs) and more specifically the variations with Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU) have been widely used in a wide range of natural language processing tasks. However, the high-dimensional real-valued hidden states ins these models are often hard to interpret. In this study we aim to perform a quantitative analysis on the hidden representations of recurrent networks in order to get a better understanding of the captured information. We use (log-)linear regression to test formed hypotheses about information captured by the hidden representations of multiple models. In addition, we propose a method to automatically find neurons responsible for specific subtasks.

1 Introduction

In recent years, Recurrent Neural Network based language models have shown outstanding performance in a variety of natural language processing tasks, including document embeddings (Mikolov et al. 2015), machine translation (Luong et al. 2015), language models (Sundermeyer, 2012) and speech recognition (Graves, 2013). The source of their outstanding performance is mainly due to gated architectures, such as the Long Short-Term Memory (LSTM) by Hochreiter and Schmidhuber (1997) and the Gated Recurrent Unit (GRU) by Chung, L et al. (2015).

Although the models show exceptional performance, their internal representations remain hardly understandable for humans. This lack of interpretability - and thereby explainability - is mainly a result of the fact that hidden states of a neural models are real-valued and often highly multi-dimensional.

Karpathy et al. (2016) showed that visualizations of individual hidden neurons in recurrent models can reveal clues about the inner workings of a model. For example, he found that some neurons in a language model trained on natural language keep track of the position in line. Although the method seems promising, analysis solely based on visualizations merely provides qualitative evidence about the hidden representation of models. Moreover, the proposed method only highlights responsibilities of single neurons, while information in neural networks is often captured as (linear) combinations across multiple hidden neurons.

Unlike previously conducted qualitative studies on the deeper understanding of recurrent networks, this study aims to quantitatively assess these models. To test whether certain information is captured by a model, we apply log-linear regression on the hidden states with a formulated hypotheses as target. In this context, we can refer to the log-linear model as a *diagnostic classifier* (Veldhoen S et al. 2016).

2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks specifically designed for sequential tasks. It is known that recurrent networks are hard to train because they suffer from vanishing and exploding gradients. Most notably, variants that inherit gating mechanisms, such as the Long-Short Term-Memory (LSTM) and the Gated Rectifier Unit (GRU) solve these issues and offer significant improvements to the vanilla version.

2.1 General Recurrent Neural Network (RNN)

A general (or vanilla) RNN consists of a hidden vector $\mathbf{h}$, which is updated at time step t as follows:

$$\mathbf{h}_t = f(\mathbf{W} * \mathbf{h}_{t-1} + \mathbf{I} * \mathbf{x}_t) \tag{1}$$

where f is a non-linearity, such as a sigmoid or hyperbolic tangent function, $\mathbf{W}$ is the recurrent weight matrix and I is a projection matrix. The hidden state $\mathbf{h}$ is then used to make a prediction

To learn a probability distribution for a next symbol $p(x_t|x_{t-1}, ..., x_1)$ over a sequence can be learned by applying a softmax to the network output as follows:

$$\mathbf{p}(y_{t,j} = 1) = \frac{\exp(\mathbf{W} * \mathbf{h}_{t-1})}{\sum_{j'=1}^{K} \exp(\mathbf{W} * \mathbf{h}_{t-1})} \tag{2}$$

where *softmax* provides a normalized probability distribution over the possible classes and $\mathbf{W}$ is a weight matrix. By using $\mathbf{h}$ as the input to another RNN, we can stack RNNs, creating deeper architectures.

$$\mathbf{h}_t^l = \sigma(\mathbf{W} * \mathbf{h}_{t-1}^l + \mathbf{U}\mathbf{h}_t^{l-1}). \tag{3}$$

Properly training vanilla RNNs can be difficult due to the vanishing and exploding gradient problem (Pascanu et al. 2013; Bengio et al. 1994).

2.2 Long Short-Term Memory (LSTM)

LSTMs address the vanishing gradient problem by introducing a recurrent architecture with gating mechanisms (Hochreiter and Schmidhuber, 1997). At each time step, an LSTM maintains a memory vector $\mathbf{m}$ and a hidden vector $\mathbf{h}$ responsible for controlling state updates and outputs. More concretely, we define the computation at time step t as follows:

$$\begin{aligned}
\mathbf{g}_i &= \sigma(\mathbf{W}^i\mathbf{h}_{t-1} + \mathbf{U}^i\mathbf{x}_t) \\
\mathbf{g}^f &= \sigma(\mathbf{W}^f\mathbf{h}_{t-1} + \mathbf{U}^f\mathbf{x}_t) \\
\mathbf{g}^o &= \sigma(\mathbf{W}^o\mathbf{h}_{t-1} + \mathbf{U}^o\mathbf{x}_t) \\
\mathbf{g}^c &= \tanh(\mathbf{W}^c\mathbf{h}_{t-1} + \mathbf{U}^c\mathbf{x}_t) \\
\mathbf{m}_t &= \mathbf{m}^f \odot \mathbf{g}^f + \mathbf{g}^c \odot \mathbf{g}^i \\
\mathbf{h}_t &= \mathbf{g}^o \tanh(\mathbf{m}_t)
\end{aligned} \tag{4}$$

Here $\mathbf{W}^i, \mathbf{W}^f, \mathbf{W}^o, \mathbf{W}^c$ are the recurrent weight matrices of the input gate, forget gate, output gate and candidate state, respectively and $\mathbf{U}^u, \mathbf{U}^f, \mathbf{U}^o, \mathbf{I}^c$ are projection matrices, σ is the sigmoid function and, finally, $\mathbf{m}$ and $\mathbf{h}$ are the memory and hidden state, respectively.

2.3 Gated Recurrent Unit (GRU)

A simpler variant of LSTM that recently gained more popularity is the Gated Recurrent Unit (GRU) by Chung, et al. (2014). We define the computation at time step t as:

$$\begin{aligned}
\mathbf{g}^z &= \sigma(\mathbf{W}^z\mathbf{h}_{t-1} + \mathbf{I}^r\mathbf{x}_t) \\
\mathbf{g}^r &= \sigma(\mathbf{W}^r\mathbf{h}_{t-1} + \mathbf{I}^z\mathbf{x}_t) \\
\tilde{\mathbf{h}}_\mathbf{t} &= \tanh(W^h(r_t \odot h_{t-1}) + U^h\mathbf{x}_t) \\
\mathbf{h}_t &= g^z \odot h_{t-1} + (1 - z_t) \odot h_t
\end{aligned} \tag{5}$$

where $\mathbf{g}^r$ and $\mathbf{g}^z$ are the reset gate and the update gate, respectively.

3 Previous Work

3.1 Generating text with character-level language models

Character-level recurrent neural language models can effectively be used to generate text sequences (Sutskever, et al. 2011; Graves, 2014). The generation process works by starting with a dummy characters and then iteratively sampling next character predictions from a language model conditioned on the previous characters. Dividing the log probabilities predicted by the model by a temperature term before applying the softmax introduces a hyper-parameter that can be used to balance exploration and exploitation of the generated text: a lower temperature causes the model to output more likely and thereby conservative estimations. However, if the temperature is set too low repetitive sequences might appear more easily in the generated text. Although the output of the network is deterministic, the probabilistic nature of the sampling process results in a probability distribution over sequences that can be generated.

3.2 Understanding Neural Networks

There are relatively few studies aimed to research the source of the high performance of modern RNNs. Most research have utilized visualization to qualitatively analyze models. For example, Karpathy, et al. (2016) showed, by visualizing, how individual neurons encode to specific high-level (linguistic) tasks. For example, one single neurons (in a character-level RNN model trained to output programming code) was encoded as a counter that holds track of the position in line, while another neuron kept of the indentation level. While this method revealed some interesting insights, an important limitation lies in the fact that the visualizations mainly offer a qualitative analysis. Moreover, the method is limited to individual neurons and is therefore unable to capture information in higher-dimensional manifolds within the representation.

Alternatively, Liwei Li et al. (2016) used t-SNE dimensionality reduction (Maaten, L, et al. 2008) and saliency heatmaps to visualize some aspects of deep recurrent language models. Taken together, visualization has shown to be an effective tool that can be used to find clues about the inner workings of recurrent models.

Other methods to obtain a better understanding exists. Kaisheng Yao, et al. (2014) analyzed LSTM models by keeping only particular gates and comparing the performance.

Lastly, Veldhoen S, et al. (2016) proposed *diagnostic classifiers*, also used in this, to analyze the hidden representations of RNN models by training and evaluating a linear classifier on the hidden states with hypotheses about the state as target.

3.2.1 Diagnostic Classifiers

Diagnostic classifiers can be used to test hypotheses about the existence of certain information in hidden representations of a recurrent neural network. A diagnostic classifier can be as simple as a linear classifier on the hidden states with a hypothesis as target.

The classifier is trained to predict a formed hypothesis conditioned on hidden states to be then evaluated on different data. An hypothesis exists in the continuous or boolean domain and has the same size as the corresponding generated the sequence. Preferably, a hypothesis is described as a function as it makes it easy to automatically generate hypotheses from a given sentence.

For example, we can test whether the hidden state at a moment in time linearly capture whether a character is part of a verb, a capitalized word or whether the hidden state captures the position in line. Firstly, a RNN model generates a number of text sequences with on each step a hidden state vector. Concatenating all hidden states results in a $D \times H$ matrix, where D is the length of the sequence and H is the dimensionality of a hidden state. We then create a boolean hypothesis vector t of size D which is a boolean mask which is positive on all characters belonging to a verb , or on characters within capitalized words or linearly decays from the beginning of a line to a new line symbol. A function that creates these mappings can trivially be engineered. For linguistic aspects (such as verb recognition) a natural language toolkit, such as the NLTK toolkit (S Bird, 2006), might be needed. Then, we split the sequence vectors, hidden state matrices and hypothesis vectors into a train and a test set. We train a classifier to predict hypotheses vectors $t^{(i)}_{train}$ in the train set, based on the corresponding hidden representation $H^{(i)}_{train}$. Then, each hidden representation $H^{(i)}_{test}$ in the test set is used to predict a target hypothesis $t^{(i)}_{test}$. The performance of the predictions of this last test measure the extend to which the hidden representations capture the formed hypothesis.

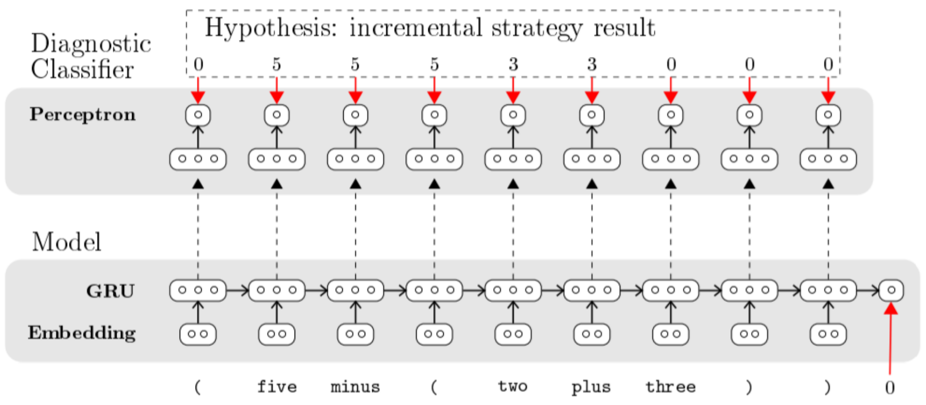

Diagnostic Classifier
Hypothesis: incremental strategy result
0 5 5 5 3 3 0 0 0
Perceptron
Model
GRU
Embedding
(five minus (two plus three)) 0

Figure 1: Testing hypothesis with diagnostic classifier (Veldhoen, S 2016)

4 Dataset and Optimization

In this study we used two datasets, namely the *Linux source kernel* ($\approx$ 474MB) and the *Tiny Shakespeare* database ($\approx$ 4.6MB). On the linux source code we trained a LSTM and GRU model with three hidden layers of 512 neurons each with a dropout rate of 0.3 resulting in a perplexity of 2.23 and 2.29, respectively, on the test set. As the tiny shakespeare dataset is smaller, we trained two layer models with 128 neurons each without dropout with resulting perplexities 2.55 and 2.68 on the test set for the LSTM and GRU model, respectively.

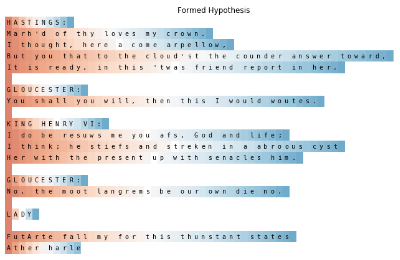
All models were trained using a variant of stochastic gradient-based optimization Adam (Kingma D et al. 2014) with a learning rate of 0.01. The weights of the models are initialized using a uniform distribution between -0.08 and 0.08. Furthermore, a batch size of 128 and a sequence length of 200 were used. Train and test sets were again splitted with according to a 95/5 ratio. All diagnostic classifiers were trained with the same ratio and on 500 character long text sequences generated with a temperature of 0.8.

5 Experiments

5.1 Hypothesis testing

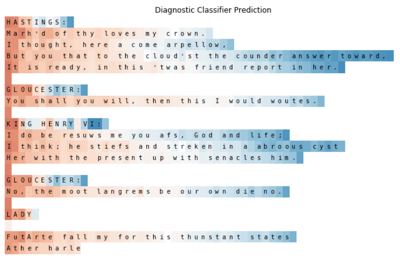
Visualizations by Karpathy et. al (2016) indicate the existence of multiple aspects in the hidden representations in a RNN trained on linux source code. Five aspects highlighted in the study are: (1) the position in line, (2) whether characters are inside quotation marks, (3) characters are inside statements, (4) to predict a new line, (5) the depth of expression and (6) whether characters are inside comments and quotes.
We test the existence of this information by applying a diagnostic classifier on the hidden weights of a character-level GRU and LSTM model.

The figures below (*Figure 2*) visualizes hypotheses and diagnostic classifier predictions on sentences from the test set.



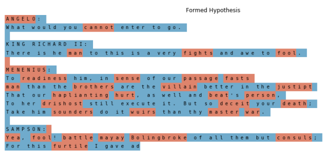
```
HASTINGS:
Marh'd of thy loves my crown.
I thought, here a come arpellow,
But you that to the cloud'st the counder answer toward,
It is ready, in this 'twas friend report in her.

GLOUCESTER:
You shall you will, then this I would woutes.

KING HENRY VI:
I do be resuws me you afs, God and life;
I think; he stiefs and streken in a abroous cyst
Her with the present up with senacles him.

GLOUCESTER:
No, the moot langrems be our own die no.

LADY

FutArte fall my for this thunstant states
Ather harle
```

(a) Hypothesis: In-line counter (Shakespeare)



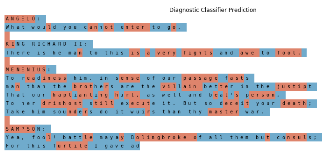
```
HASTINGS:
Marh'd of thy loves my crown.
I thought, here a come arpellow,
But you that to the cloud'st the counder answer toward,
It is ready, in this 'twas friend report in her.

GLOUCESTER:
You shall you will, then this I would woutes.

KING HENRY VI:
I do be resuws me you afs, God and life;
I think; he stiefs and streken in a abroous cyst
Her with the present up with senacles him.

GLOUCESTER:
No, the moot langrems be our own die no.

LADY

FutArte fall my for this thunstant states
Ather harle
```

(b) Prediction by Diagnostic classifier (Shakespeare)



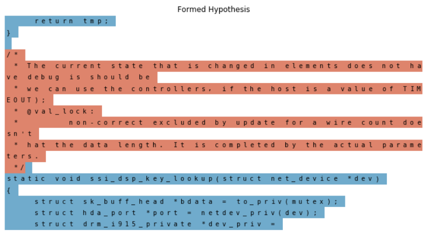
```
ANGELO:
What would you cannot enter to go.

KING RICHARD II:
There is he man to this is a very fights and awe to fool.

MENENIUS:
To readiness him, in sense of our passage fasts
man than the brothers are the villain better in the justipt
That our haplianting hurt, as well and beat's person,
To her drishost still execute it. But so deceit your death;
Take him sounders do it wuirs than thy master war.

SAMPSON:
Yea, fool! battle mayay Bolingbroke of all them but consuls;
For this furtile I gave ad
```

(c) Hypothesis: Nouns (Shakespeare)



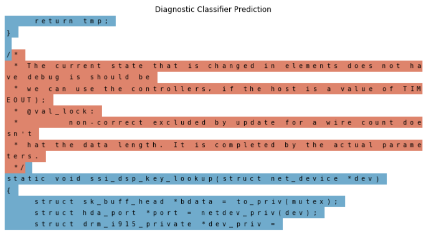
```
ANGELO:
What would you cannot enter to go.

KING RICHARD II:
There is he man to this is a very fights and awe to fool.

MENENIUS:
To readiness him, in sense of our passage fasts
man than the brothers are the villain better in the justipt
That our haplianting hurt, as well and beat's person,
To her drishost still execute it. But so deceit your death;
Take him sounders do it wuirs than thy master war.

SAMPSON:
Yea, fool! battle mayay Bolingbroke of all them but consuls;
For this furtile I gave ad
```

(d) Prediction by Diagnostic classifier (Shakespeare)



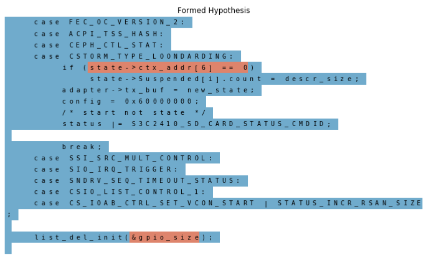
```
    return tmp;
}

/*
 * The current state that is changed in elements does not ha
ve debug is should be
 * we can use the controllers, if the host is a value of TIM
EOUT);
 * @val_lock:
 *      non-correct excluded by update for a wire count doe
sn't
 * hat the data length. It is completed by the actual parame
ters.
 */
static void ssi_dsp_key_lookup(struct net_device *dev)
{
    struct sk_buff_head *bdata = to_priv(mutex);
    struct hda_port *port = netdev_priv(dev);
    struct drm_i915_private *dev_priv =
```

(e) Hypothesis: Inside comments (Linux code)



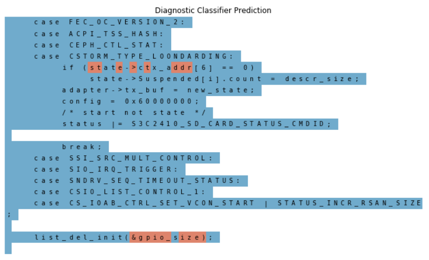
```
    return tmp;
}

/*
 * The current state that is changed in elements does not ha
ve debug is should be
 * we can use the controllers, if the host is a value of TIM
EOUT);
 * @val_lock:
 *      non-correct excluded by update for a wire count doe
sn't
 * hat the data length. It is completed by the actual parame
ters.
 */
static void ssi_dsp_key_lookup(struct net_device *dev)
{
    struct sk_buff_head *bdata = to_priv(mutex);
    struct hda_port *port = netdev_priv(dev);
    struct drm_i915_private *dev_priv =
```

(f) Prediction by Diagnostic classifier (Linux code)



```
    case FEC_OC_VERSION_2:
    case ACPI_TSS_HASH:
    case CEPH_CTL_STAT:
    case CSTORM_TYPE_LOONDARDING:
        if (state->ctx_addr[6] == 0)
            state->Suspended[i].count = descr_size;
        adapter->tx_buf = new_state;
        config = 0x60000000;
        /* start not state */
        status |= S3C2410_SD_CARD_STATUS_CMDID;

        break;
    case SSI_SRC_MULT_CONTROL:
    case SIO_IRQ_TRIGGER:
    case SNDRV_SEQ_TIMEOUT_STATUS:
    case CSIO_LIST_CONTROL_1:
    case CS_IOAB_CTRL_SET_VCON_START | STATUS_INCR_RSAN_SIZE
;

    list_del_init(&gpio_size);
```

(g) Hypothesis: Inside parentheses (Linux code)



```
    case FEC_OC_VERSION_2:
    case ACPI_TSS_HASH:
    case CEPH_CTL_STAT:
    case CSTORM_TYPE_LOONDARDING:
        if (state->ctx_addr[6] == 0)
            state->Suspended[i].count = descr_size;
        adapter->tx_buf = new_state;
        config = 0x60000000;
        /* start not state */
        status |= S3C2410_SD_CARD_STATUS_CMDID;

        break;
    case SSI_SRC_MULT_CONTROL:
    case SIO_IRQ_TRIGGER:
    case SNDRV_SEQ_TIMEOUT_STATUS:
    case CSIO_LIST_CONTROL_1:
    case CS_IOAB_CTRL_SET_VCON_START | STATUS_INCR_RSAN_SIZE
;

    list_del_init(&gpio_size);
```

(h) Prediction by Diagnostic classifier (Linux code)

Figure 2: Hypothesis and Diagnosis of Diagnostic classifier

From the visualizations, we can see that the predictions by the diagnostic classifier closely resemble formed hypotheses. What stands out is that the diagnostic classifier is able to predict many tasks almost perfectly. Interestingly, we also find some correlation between hypotheses and predictions by the diagnostic classifier of seemingly difficult tasks, such as predicting whether a character belonging to a verb.

5.2 Diagnostic Classifier Quantification

To evaluate the extent to which neural models capture specific subtasks encoded by individual neurons found by Karpathy et. al (2016), we predict a fitted diagnostic classifier on a test set of generated sequences.

It is important to bear in mind that the accuracy of this prediction might not be a relevant metric as there might be a class imbalance in some hypotheses. For example, very little characters belong to text inside quotation marks and therefore (very wrongly) predicting that no characters belong inside quotation marks yields high accuracy. Therefore, the table provides additional metrics, namely precision, recall and f1-score. To further tackle class imbalance we, we train a log-linear classifier with balanced class weights in addition to a regular log-linear model as diagnostic classifier.

The figures below show the results of the hypothesis tests on a character-level RNN-LMs trained on Linux source code: LSTM based (*Figure 3*) and GRU-based (*Figure 4*).

Hypothesis	Accuracy	Precision	Recall	F1-score
Inside parentheses	0.98/0.97	0.95/0.95	0.96/0.98	0.96/0.96
Inside quotation	0.99/0.99	0.84/0.82	0.91/0.93	0.87/0.87
Comments	0.99/0.98	0.97/0.90	0.95/0.93	0.96/0.91
Indent Level > 1	0.94/0.98	0.98/0.99	0.98/0.97	0.98/0.98
Indent Level > 2	0.97/0.98	0.97/0.93	0.94/0.97	0.95/0.95
Indent Level > 3	0.98/98	0.93/0.84	0.90/0.93	0.91/0.88

Figure 3: Hypothesis tests on Linux source code LSTM model. Each cell: [linear fit/balanced linear fit]

Hypothesis	Accuracy	Precision	Recall	F1-score
Inside parentheses	0.96/0.94	0.93/0.88	0.96/0.91	0.94/0.90
Comments	0.95/0.93	0.92/0.94	0.84/0.75	0.88/0.83
Inside quotation	0.99/0.99	0.73/0.84	0.81/0.99	0.77/0.91
Indent Level > 1	0.95/0.92	0.97/0.93	0.97/0.97	0.97/0.95
Indent Level > 2	0.96/0.93	0.93/0.95	0.96/0.88	0.95/0.91
Indent Level > 3	0.96/0.94	0.67/0.83	0.61/0.84	0.63/0.84

Figure 4: Hypothesis tests on Linux source code GRU model. Each cell: [linear fit/balanced linear fit]

From the tables above we can conclude that both the LSTM and the GRU model capture the information tested with the diagnostic classifier with high precision and high recall. Although all tasks are predicted by the diagnostic classifier with high accuracy, a small decrease in precision and recall is visible for higher level indentations.

The figures below shows hypothesis tests on character-level RNN-LMs trained on the tiny Shakespeare dataset: LSTM-based (*Figure* **??**) and GRU-based (*Figure* **??**).

Hypothesis	Accuracy	Precision	Recall	F1-score
Capitalized word	1.00/0.99	0.97/0.90	0.99/1.0	0.98/0.95
Verbs	0.80/0.74	0.68/0.26	0.29/0.66	0.40/0.38
Nouns	0.81/0.80	0.72/0.68	0.63/0.75	0.67/0.72

Figure 5: Hypothesis tests on Shakespeare dataset on LSTM model. Each cell: [linear fit/balanced linear fit]

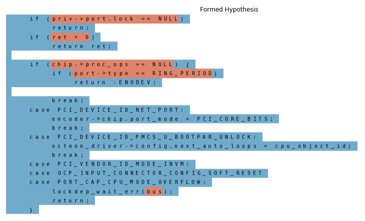
Hypothesis	Accuracy	Precision	Recall	F1-score
Capitalized words	1.00/0.99	0.99/0.99	1.0/1.0	0.99/1.00
Verbs	0.78/0.76	0.68/0.59	0.59/0.73	0.63/0.65
Nouns	0.91/0.77	0.49/0.24	0.18/0.81	0.26/0.37

Figure 6: Hypothesis tests on Shakespeare dataset on GRU model. Each cell: [linear fit/balanced linear fit]

The most obvious finding to emerge from the analysis is the a near perfect linear relationship between hidden states and character that belong to capitalized words. However, there is no clear evidence that the model is able to accurately tell the difference between characters that are part of a verb or noun or part of an other word.

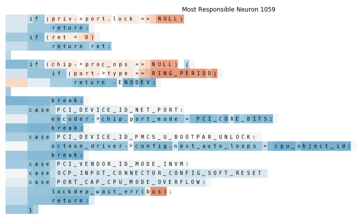
5.3 Visualization of Most Responsible Neuron

To automatically find the index of the neuron most responsible for the activations of a formulated hypothesis we simply take the index of the maximum absolute weight of a (log-)linear diagnostic classifier. The activations of the found single neurons are visualized in the figure below.



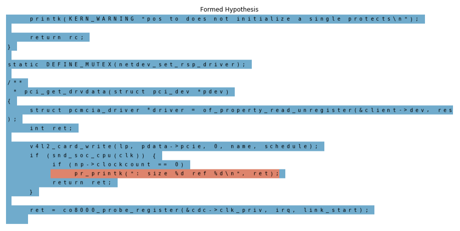
```
    if (priv->port.lock == NULL)
        return;
    if (ret < 0)
        return ret;

    if (chip->proc_ops == NULL) {
        if (port->type == RING_PERIOD)
            return -ENODEV;

        break;
    case PCI_DEVICE_ID_NET_PORT:
        encoder->chip.port_mode = PCI_CORE_BITS;
        break;
    case PCI_DEVICE_ID_PMCS_U_BOOTPAR_UNLOCK:
        octeon_driver->config.next_auto_loops = cpu_object_id;
        break;
    case PCI_VENDOR_ID_MODE_INVM:
    case OCP_INPUT_CONNECTOR_CONFIG_SOFT_RESET
    case PORT_CAP_CPU_MODE_OVERFLOW:
        lockdep_wait_err(bus);
        return;
    }
```

(a) Hypothesis: Inside parentheses



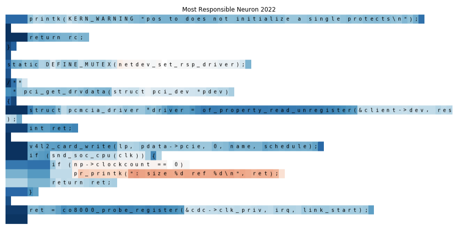
```
    if (priv->port.lock == NULL)
        return;
    if (ret < 0)
        return ret;

    if (chip->proc_ops == NULL) {
        if (port->type == RING_PERIOD)
            return -ENODEV;

        break;
    case PCI_DEVICE_ID_NET_PORT:
        encoder->chip.port_mode = PCI_CORE_BITS;
        break;
    case PCI_DEVICE_ID_PMCS_U_BOOTPAR_UNLOCK:
        octeon_driver->config.next_auto_loops = cpu_object_id;
        break;
    case PCI_VENDOR_ID_MODE_INVM:
    case OCP_INPUT_CONNECTOR_CONFIG_SOFT_RESET
    case PORT_CAP_CPU_MODE_OVERFLOW:
        lockdep_wait_err(bus);
        return;
    }
```

(b) Single most responsible neuron



```
    printk(KERN_WARNING "pos to does not initialize a single protects\n");

    return rc;
}

static DEFINE_MUTEX(netdev_set_rsp_driver);

/**
 * pci_get_drvdata(struct pci_dev *pdev)
{
    struct pcmcia_driver *driver = of_property_read_unregister(&client->dev, res
);
    int ret;

    v4l2_card_write(lp, pdata->pcie, 0, name, schedule);
    if (snd_soc_cpu(clk)) {
        if (np->clockcount == 0)
            pr_printk(": size %d ref %d\n", ret);
        return ret;
    }

    ret = co8000_probe_register(&cdc->clk_priv, irq, link_start);
```

(c) Hypothesis: Indentation level ≥ 3



```
    printk(KERN_WARNING "pos to does not initialize a single protects\n");

    return rc;
}

static DEFINE_MUTEX(netdev_set_rsp_driver);

/**
 * pci_get_drvdata(struct pci_dev *pdev)
{
    struct pcmcia_driver *driver = of_property_read_unregister(&client->dev, res
);
    int ret;

    v4l2_card_write(lp, pdata->pcie, 0, name, schedule);
    if (snd_soc_cpu(clk)) {
        if (np->clockcount == 0)
            pr_printk(": size %d ref %d\n", ret);
        return ret;
    }

    ret = co8000_probe_register(&cdc->clk_priv, irq, link_start);
```

(d) Single most responsible neuron

Figure 7: Automatically found single most responsible neurons for different tasks

Besides clear responsibilities of some individual neurons, including the instances shown above, we did not always find a single neuron responsible for a certain task. On average, we found it harder to find neurons responsible for single tasks in lower dimensional models. Interestingly, the diagnostic classifier is often able to verify a hypothesis with high precision and high recall without a single individual neuron being responsible. From this, we can conclude that models often tend to encode essential information for specific tasks as (linear) combination across multiple neurons instead of using individual neurons of specific tasks.

6 Conclusion

Due to the high dimensional and real-valued hidden states, recurrent neural models are hard to interpret. Previous research has shown that qualitative methods, such as visualizations of individual hidden neurons, can give clues about the inner workings of such models. However, these approaches are often not automated and limited to single neurons, while features are often captured as combinations over multiple weights.

The present study confirms findings from previous research and contributes additional quantitative evidence. We used regular and class-balanced log-linear regression as a diagnostic classifier to test a number of hypotheses about the hidden states in recurrent networks. Multiple hypotheses were formulated and evaluated on LSTM and GRU character-level language models trained on both Linux source code and natural language.

The study has quantitatively shown that a range of subtasks, such as keeping track of the position in a line or whether characters are inside quotation, are captured in the hidden states of recurrent language models. Furthermore, we proposed a method to automatically find individual neurons encoded for specific tasks. However, we also found that a single responsible neuron does not always exists as a tasks is often distributed over multiple nodes.

An important limitation of the study lies in the fact that we performed our experiments on a limited amount of hypotheses and models. However, the results do suggest that diagnostic classifiers are an effective way to analyze recurrent models.

A natural progression of this work is to test a wide range of hypotheses on a number of language models. Moreover, it would be interesting to explore the differences between gates.

References

[1] Sara Veldhoen, Dieuwke Hupkes, and Willem Zuidema. Diagnostic classifiers revealing how neural networks process hierarchical structure. In *CoCo@ NIPS*, 2016.

[2] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[4] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[5] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[6] Kaisheng Yao, Geoffrey Zweig, Mei-Yuh Hwang, Yangyang Shi, and Dong Yu. Recurrent neural networks for language understanding. In *Interspeech*, pages 2524–2528, 2013.

[7] Andrew M Dai, Christopher Olah, and Quoc V Le. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*, 2015.

[8] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[9] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[10] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[11] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[12] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, pages 2067–2075, 2015.

[13] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[14] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[15] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.