Input Switched Affine Networks: An RNN Architecture Designed for Interpretability

Jakob N. Foerster [*1] Justin Gilmer [*2] Jascha Sohl-Dickstein [3] Jan Chorowski [4] David Sussillo [3]

Abstract

There exist many problem domains where the interpretability of neural network models is essential for deployment. Here we introduce a recurrent architecture composed of input-switched affine transformations – in other words an RNN without any explicit nonlinearities, but with input-dependent recurrent weights. This simple form allows the RNN to be analyzed via straightforward linear methods: we can *exactly* characterize the linear contribution of each input to the model predictions; we can use a change-of-basis to disentangle input, output, and computational hidden unit subspaces; we can fully reverse-engineer the architecture's solution to a simple task. Despite this ease of interpretation, the input switched affine network achieves reasonable performance on a text modeling tasks, and allows greater computational efficiency than networks with standard nonlinearities.

1. Introduction

1.1. The importance of interpretable machine learning

As neural networks move into applications where the outcomes of human lives depend on their decisions, it is increasingly crucial that we are able to interpret the decisions they make. Indeed, the European Union is considering legislation with a clause that asserts that individuals have 'rights to explanation', i.e. individuals should be able to understand how algorithms make decisions about them (Council of European Union, 2016). Example problem domains requiring interpretable ML include self-driving cars (Bojarski et al., 2016), air traffic control (Katz et al., 2017), power grid control (Siano et al., 2012), hiring and promotion decisions while preventing bias (Scarborough & Somers, 2006), automated sentencing decisions in US courts (Tashea, 2017; Berk et al., 2017), and medical diagnosis (Gulshan et al., 2016). For many of these applications, practitioners will not adopt ML models without fully understanding what drives their predictions, including understanding when and how these models fail (Ching et al., 2017; Deo, 2015).

1.2. Post hoc analysis

One approach to interpreting neural networks is to train the network as normal, and then apply analysis techniques after training. Often this approach yields systems that perform extremely well, but where interpretability is challenging. For example, Sussillo & Barak (2013) used linearization and nonlinear dynamical systems theory to understand RNNs solving a set of simple but varied tasks. Karpathy et al. (2015) analyzed an LSTM (Hochreiter & Schmidhuber, 1997) trained on a character-based language modeling task. They were able to break down LSTM language model errors into classes, such as e.g., "rare word" errors. Concurrently with our submission, Murdoch & Szlam (2017) decomposed LSTM outputs using telescoping sums of statistics computed from memory cells at different RNN steps. The decomposition is exact, but not unique and the authors justify it by demonstrating good performance of decision rules formed using the computed cell statistics.

The community is also interested in post hoc interpretation of feed-forward networks. Examples include the use of linear probes in Alain & Bengio (2016), and a variety of techniques (most driven by back-propagation) to assign credit for activations to specific inputs or input patterns in feed-forward networks (Zeiler et al., 2010; Le et al., 2012; Mordvintsev et al., 2015).

1.3. Building interpretability into the architecture

A second approach is to build a neural network where interpretability is an explicit design constraint. In this approach, a typical outcome is a system that can be better understood, but at the cost of reduced performance. Model classes whose

[*]Equal contribution [1]This work was performed as an intern at Google Brain [2]Work done as a member of the Google Brain Residency program (g.co/brainresidency) [3]Google Brain, Mountain View, CA, USA [4]Work performed when author was a visiting faculty at Google Brain. Correspondence to: Jakob N. Foerster <jakob.foerster@cs.ox.ac.uk>, David Sussillo <sussillo@google.com>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

decisions are naturally interpretable include logistic regression (Freedman, 2009), decision trees (Quinlan, 1987), and support vector machines with simple (e.g. linear) kernels (Andrew, 2013).

In this work we follow this second approach and build interpretability into our network model, while maintaining good, though not always state-of-the-art, performance for the tasks we study. We focus on the commonly studied task of character based language modeling. We develop and analyze a model trained on a one-step-ahead prediction task of the Text8 dataset, which is 10 million characters of Wikipedia text (Mahoney, 2011), on the Billion Word Benchmark (Chelba et al., 2013), and finally on a toy multiple parentheses counting task which we fully reverse engineer.

1.4. Switched affine systems

The model we introduce is an Input Switched Affine Network (ISAN), where the input determines the switching behavior by selecting a transition matrix and bias as a function of that input, and there is no nonlinearity. Linear time-varying systems are standard material in undergraduate electrical engineering text books, and are closely related to our technique.

Although the ISAN is deterministic, probabilistic versions of switching linear models with discrete latent variables have a history in the context of probabilistic graphical models. A recent example is the switched linear dynamical system in (Linderman et al., 2016). Focusing on language modeling, (Belanger & Kakade, 2015) defined a probabilistic linear dynamical system (LDS) as a generative language model for creating context-dependent token embeddings and then used steady-state Kalman filtering for inference over token sequences. They used singular value decomposition and discovered that the right and left singular vectors were semantically and syntactically related. A critical difference between the ISAN and the LDS is that the ISAN weight matrices are input token dependent (while the biases of both models are input dependent).

Multiplicative neural networks (MRNNs) were proposed precisely for character based language modeling in (Sutskever et al., 2011; Martens & Sutskever, 2011). The MRNN architecture is similar to our own, in that the dynamics matrix switches as a function of the input character. However, the MRNN relied on a $\tanh$ nonlinearity, while the ISAN is explicitly linear. It is this property of our model which makes it both amenable to analysis, and computationally efficient.

The Observable Operator Model (OOM) (Jaeger, 2000) is similar to the ISAN in that the OOM updates a latent state using a separate transition matrix for each input symbol and performs probabilistic sequence modeling. Unlike the ISAN, the OOM requires that a linear projection of the hidden state corresponds to a normalized sequence probability. This imposes strong constraints on both the model parameters and the model dynamics, and restricts the choice of training algorithms. In contrast, the ISAN applies an affine readout to the hidden state to obtain logits, which are then pushed through the softmax function to obtain probabilities. Therefore no constraints need to be imposed on the ISAN's parameters and training is easy using backprop. Lastly, the ISAN is formulated as an affine, rather than linear model. While this doesn't change the class of processes that can be modeled, it stabilizes training and greatly enhances interpretability, facilitating the analysis in Section 3.3.

1.5. Paper structure

In what follows, we define the ISAN architecture, demonstrate its performance on the one-step-ahead prediction task, and then analyze the model in a multitude of ways, most of which would be currently difficult or impossible to accomplish with modern nonlinear recurrent architectures.

2. Methods

2.1. Model definition

In what follows $\mathbf{W_x}$ and $\mathbf{b_x}$ respectively denote a transition matrix and a bias vector for a specific input $\mathbf{x}$, the symbol $\mathbf{x}_t$ is the input at time t, and $\mathbf{h}_t$ is the hidden state at time t. Our ISAN model is defined as

$$\mathbf{h}_t = \mathbf{W}_{\mathbf{x_t}}\ \mathbf{h}_{t-1} + \mathbf{b}_{\mathbf{x_t}}. \tag{1}$$

The network also learns an initial hidden state $\mathbf{h}_0$. We emphasize the intentional absence of any nonlinear activation function.

2.2. Character level language modeling with ISAN

We trained RNNs on the Text8 Wikipedia dataset and the billion word benchmark (BWB), for one-step-ahead character prediction. The Text8 dataset consists only of the 27 characters 'a'-'z' and '_' (space). The BWB dataset consist of Unicode text and was modelled as a sequence of bytes (256 discrete tokens) that formed the UTF8-encoded data. Given a character sequence of $\mathbf{x}_1, ..., \mathbf{x}_t$, the RNNs are trained to minimize the cross-entropy between the true next character, and the output prediction. We map from the hidden state, $\mathbf{h}_t$, into a logit space via an affine map. The probabilities are computed as

$$p\left(\mathbf{x}_{t+1}\right) = \text{softmax}\left(\mathbf{l}_t\right) \tag{2}$$

$$\mathbf{l}_t = \mathbf{W}_{ro}\ \mathbf{h}_t + \mathbf{b}_{ro}, \tag{3}$$

where $\mathbf{W}_{ro}$ and $\mathbf{b}_{ro}$ are the readout weights and biases, and $\mathbf{l}_t$ is the logit vector. For the Text8 dataset, we split

Table 1. The ISAN has similar performance to other RNN architectures on the Text8 dataset. Performance of RNN architectures on Text8 one-step-ahead prediction, measured as cross-entropy loss on a held-out test set, in bits per character. The loss is shown as a function of the maximum number of parameters a model is allowed. The values reported for all other architectures are taken from (Collins et al., 2016).

Parameter count	8e4	3.2e5	1.28e6
RNN	1.88	1.69	1.59
IRNN	1.89	1.71	1.58
GRU	1.83	1.66	1.59
LSTM	1.85	1.68	1.59
ISAN	1.92	1.71	1.58

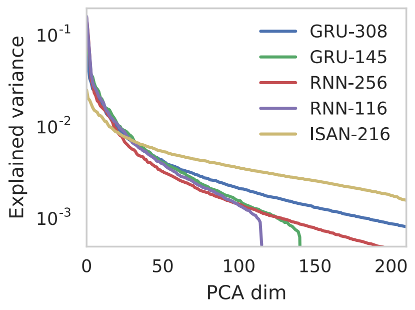

Figure 1. The ISAN makes fuller and more uniform use of its latent space than vanilla RNNs or GRUs. Figure shows explained variance ratio of the first 210 most significant PCA dimensions of the hidden states across several architectures for the Text8 dataset. The legend provides the number of latent units for each architecture.

the data into 90%, 5%, and 5% for train, validation, and test respectively, in line with (Mikolov et al., 2012). The network was trained with the same hyperparameter tuning infrastructure as in (Collins et al., 2016). For the BWB dataset, we used data splits and evaluation setup identical to (Józefowicz et al., 2016). Due to long experiment running times, we manually tuned the hyperparameters.

3. Results and analysis

3.1. ISAN performance on Text8 prediction

The results on Text8 are shown in Table 1. For the largest parameter count, the ISAN matches almost exactly the performance of all other nonlinear models with the same number of maximum parameters: RNN, IRNN, GRU, LSTM. However, we note that for small numbers of parameters the ISAN performs considerably worse than other architectures. All analyses use ISAN trained with 1.28e6 maximum parameters (1.58 bpc cross entropy). Samples of generated text from this model are relatively coherent. We show two examples, after priming with "annual reve", at inverse temperature of 1.5, and 2.0, respectively:

- *"annual revenue and producer of the telecommunications and former communist action and saving its new state house of replicas and many practical persons"*
- *"annual revenue seven five three million one nine nine eight the rest of the country in the united states and south africa new"*.

As a preliminary, comparative analysis, we performed PCA on the state sequence over a large set of sequences for the vanilla RNN, GRU of varying sizes, and ISAN. This is shown in Figure 1. The eigenvalue spectra, in log of variance explained, was significantly flatter for the ISAN than the other architectures.

We compared the ISAN performance to a fully linear RNN without input switched dynamics. This achieves a cross-entropy of 3.1 bits / char, independent of network size. This perplexity is only slightly better than that of a Naive Bayes model on the task, at 3.3 bits / char. The output probability of the fully linear network is a product of contributions from each previous character, as in Naive Bayes. Those factorial contributions are learned however, giving the non-switched affine network a slight advantage. We also trained a fully linear network with a nonlinear readout. This achieves 2.15 bits / char, independent of network size. Both of these comparisons illustrate the importance of the input switched dynamics for achieving good results.

Lastly we also test to what extent the ISAN can deal with large dictionaries by running it on a byte-pair encoding of the text8 task, where the input dictionary consists of the 27^2 different possible character combinations. We find that in this setup the LSTM consistently outperforms the ISAN for the same number of parameters. At $1.3m$ parameters the LSTM achieves a cross entropy of 3.4 bits / char-pair, while ISAN achieves 3.55. One explanation for this finding is that the matrices in ISAN are 27 times smaller than the matrices of the LSTMs. For very large numbers of parameters the performance of any architecture saturates in the number of parameters, at which point the ISAN can 'catch-up' with more parameter efficient architectures like LSTMs.

3.2. ISAN performance on Billion Word Benchmark prediction

We trained ISAN and LSTM models on the BWB dataset. All networks were trained using asynchronous gradient descent using the Adagrad learning rule. Our best LSTM model reached 1.1 bits per character, which matches published results (Hwang & Sung, 2016). The LSTM model had one layer of 8192 LSTM units whose outputs were projected onto 1024 dimensions (44e6 parameters). Our best ISAN models reached 1.4 bits per character and used

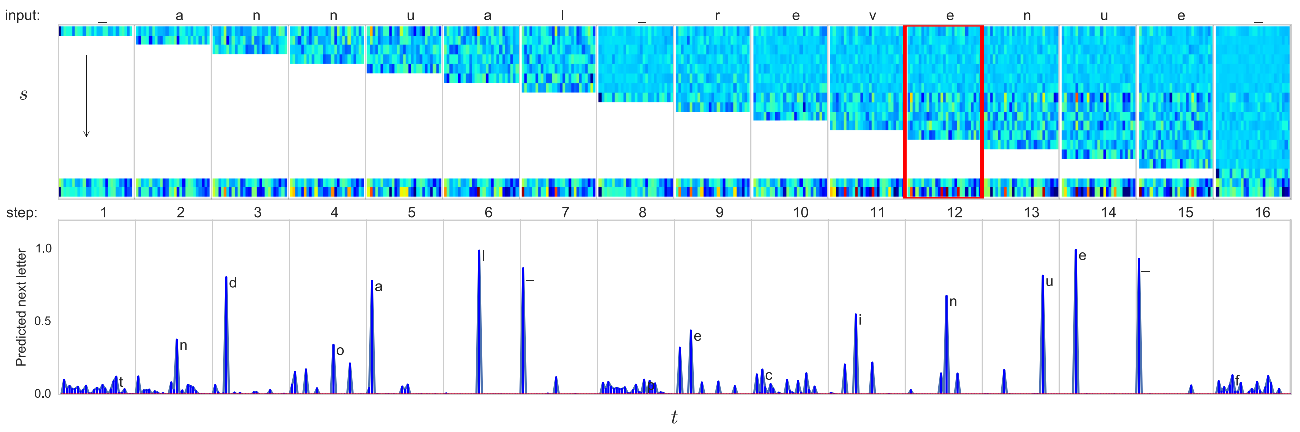

Figure 2. Using the linearity of the hidden state dynamics, predictions at step t can be broken out into contributions, $\boldsymbol{\kappa}_s^t$, from previous steps. Accordingly, each row of the top panel corresponds to the propagated contribution ($\boldsymbol{\kappa}_s^t$) of the input character at time s, to the prediction at time t (summed to create the logit at time t). The penultimate row contains the output bias vector replicated at every time step. The last row contains the logits of the predicted next character, which is the sum of all rows above. The bottom panel contains the corresponding softmax probabilities at each time t for all characters (time is separated by gray lines). Labeled is the character with the maximum predicted probability. The time step boxed in red is examined in more detail in Figure 3.

512 hidden units, a reduced set of most common 70 input tokens and 256 output tokens (18e6 parameters). Increasing ISAN's hidden layer size to 768 units (41e6 parameters) yielded a perplexity improvement to 1.36 bits/char. Investigation of generated samples shows that the ISAN learned the distinction between lower- and upper-cased letters and is able to generate text which is coherent over short segments. To demonstrate sample variability we show continuations of the prompt "The [Pp]ol" generated using the ISAN:

- *The Pol|ish pilgrims are as angry over the holiday trip*
- *The Pol|ice Department subsequently slipped toward*
- *The Pol|ice Federation has sought Helix also investors*
- *The Pol|itico is in a tight crowd ever to moderated the*
- *The pol|itical scientist in the Red Shirt Romance cannot*
- *The pol|icy for all Balanchine had formed when it set a*
- *The pol|l conducted when a suspected among Hispanic*
- *The pol|itical frenzy sparked primary care programs*

3.3. Decomposition of current predictions based on previous time steps

Analysis in this paper is carried out on the best-performing Text8 ISAN model, which has $1,271,619$ parameters, corresponding to 216 hidden units, and 27 dynamics matrices $\mathbf{W}_\mathbf{x}$ and biases $\mathbf{b}_\mathbf{x}$.

With ISAN we can analyze which factors were important in the past for determining the current character prediction. Taking advantage of the linearity of the hidden state dynamics for any sequence of inputs, we decompose the current latent state $\mathbf{h}_t$ into contributions originating from different time points s in the history of the input:

$$\mathbf{h}_t = \sum_{s=0}^{t} \left(\prod_{s'=s+1}^{t} \mathbf{W}_{\mathbf{x}_{s'}} \right) \mathbf{b}_{\mathbf{x}_s}, \tag{4}$$

where the empty product when $s+1 > t$ is 1 by convention, and $\mathbf{b}_{\mathbf{x}_0} = \mathbf{h}_0$ is the learned initial hidden state.

Using this decomposition and the fact that matrix multiplication is a linear transformation we can also write the unnormalized logit-vector, $\mathbf{l}_t$, as a sum of terms linear in the biases,

$$\mathbf{l}_t = \mathbf{b}_{ro} + \sum_{s=0}^{t} \boldsymbol{\kappa}_s^t \tag{5}$$

$$\boldsymbol{\kappa}_s^t = \mathbf{W}_{ro} \left(\prod_{s'=s+1}^{t} \mathbf{W}_{\mathbf{x}_{s'}} \right) \mathbf{b}_{\mathbf{x}_s}, \tag{6}$$

where $\boldsymbol{\kappa}_s^t$ is the contribution from time step s to the logits at time step t, and $\boldsymbol{\kappa}_t^t = \mathbf{b}_{\mathbf{x}_t}$. For notational convenience we will sometimes replace the subscript s with the corresponding input character $\mathbf{x}_s$ at step s when referring to $\boldsymbol{\kappa}_s^t$. For example, $\boldsymbol{\kappa}_{\text{‘q’}}^t$ refers to the contribution from the character ‘q’ in a string. Similarly, when discussing the summed contributions from a word or substring we will sometimes write $\boldsymbol{\kappa}_{word}^t$ to mean the summed contributions of all the $\boldsymbol{\kappa}_s^t$ from that source word. For example, $\sum_{s \in \text{word}} \boldsymbol{\kappa}_s^t - \boldsymbol{\kappa}_{\text{‘the’}}^t$ refers to the total contribution from the word ‘the’ to the logit.

While in standard RNNs the nonlinearity causes interdependence of the bias terms across time steps, in the ISAN the bias terms contribute to the state as independent linear

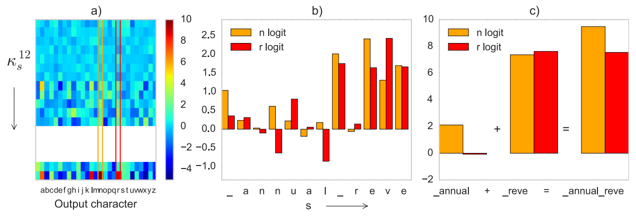

Figure 3. Detailed view of the prediction stack for the final 'n' in '_annual_reve<u>n</u>ue'. In *a)* all κ_s^t are shown, in *b)* only the contributions to the 'n' logit and 'r' logits are shown, in orange and red respectively, from each earlier character in the string. This corresponds to a zoom in view of the columns highlighted in orange and red in a). In *c)* we show how the sum of the contributions from the string '_annual', $\kappa^t_{\text{'\_annual'}}$, pushes the prediction at '_annual_reve' from 'r' to 'n'. Without this contribution the model decodes based only on $\kappa^t_{\text{'\_reve'}}$, leading to a MAP prediction of 'reverse'. With the contribution from $\kappa^t_{\text{'\_annual'}}$ it instead predicts 'revenue'. The contribution of $\kappa^t_{\text{'\_annual'}}$ to the 'n' and 'r' logits is linear and exact.

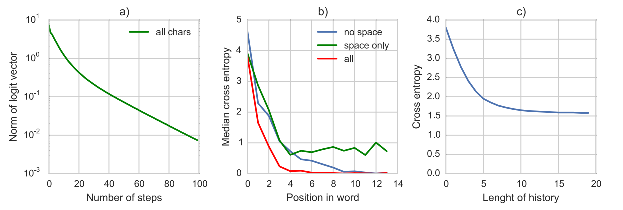

Figure 4. The time decay of the contributions from each character to prediction. *a)* Average norm of κ_s^t across training text, $\mathbb{E}\left[\left|\left|\kappa_s^t\right|\right|_2\right]$, plotted as a function of $t - s$, and averaged across all source characters. The norm appears to decay exponentially at two rates, a faster rate for the first ten or so characters, and then a slower rate for more long term contributions. *b)* The median cross entropy as a function of the position in the word under three different circumstances: the red line uses all of the κ_s^t (baseline), the green line sets all κ_s^t apart from $\kappa^t_{\text{'\_'}}$ to zero, while the blue line only sets $\kappa^t_{\text{'\_'}}$ to zero. The results from panel b demonstrate the disproportionately large importance of '_' in decoding, especially at the onset of a word. *c)* The cross-entropy as a function of history when artificially limiting the number of characters available for prediction. This corresponds to only considering the most recent n of the κ, where n is the length of the history.

terms that are propagated and transformed through time. We emphasize that κ_s^t includes the multiplicative contributions from the $\mathbf{W}_{\mathbf{x}_{s'}}$ for $s < s' \leq t$. It is however independent of prior inputs, $\mathbf{x}_{s'}$ for $s' < s$. This is the main difference between the analysis we can carry out with the ISAN compared to a nonlinear RNN. In a general recurrent network the contribution of a specific character sequence will depend on the hidden state at the start of the sequence. Due to the linearity of the dynamics, this dependency does not exist in the ISAN.

In Figure 2 we show an example of how this decomposition allows us to understand why a particular prediction is made at a given point in time, and how previous characters influence the decoding. For example, the sequence '_annual_revenue_' is processed by the ISAN: Starting with an all-zero hidden state, we use equation (6) to accumulate a sequence of $\kappa^t_{\text{'\_'}}, \kappa^t_{\text{'a'}}, \kappa^t_{\text{'n'}}, \kappa^t_{\text{'n'}}, \ldots$. We then used these values to understand the prediction of the network at some time t, by simple addition across the s index.

We provide a detailed view of how past characters contribute to the logits predicting the next character in Figure 3. There are two competing options for the next letter in the word stem 'reve': either 'reve**n**ue' or 'reve**r**se'. We show that without the contributions from '_annual' the most likely decoding of the character after the second 'e' is 'r' (to form 'reverse'), while the contributions from '_annual' tip the balance in favor of 'n', decoding to 'revenue'.

Using ISAN, we can investigate information timescales in the network. For example, we investigated how quickly the contributions of κ_s^t decay as a function of $t - s$ on average. Figure 4a shows that this contribution decays on two different exponential timescales. We hypothesize that the first time scale corresponds to the decay within a word, while the next corresponds to the decay of information across words and sentences. We also show the relevance of the κ_s^t contributions to the decoding of characters at different positions in the word (Figure 4b). For example, we observe that $\kappa^t_{\text{'\_'}}$ makes important contributions to the prediction of the next character at time t. We show that using only the $\kappa^t_{\text{'\_'}}$, the model can achieve a cross entropy of less than 1 bit / char when the position of the character is more than 3 letters from the beginning of the word. Finally, we link the norm-decay of κ_s^t to the importance of past characters for the decoding quality (Figure 4c). By artificially limiting the number of past κ available for prediction we show that the prediction quality improves rapidly when extending the history from 0 to 10 characters and then saturates. This rapid improvement aligns with the range of faster decay in Figure 4a.

3.4. From characters to words

The ISAN provides a natural means of moving from character level representation to word level. Using the linearity of the hidden state dynamics we can aggregate all of the κ_s^t belonging to a given word and visualize them as a single contribution to the prediction of the letters in the next word. This allows us to understand how each preceding word impacts the decoding for the letters of later words. In Figure 5 we show that the words 'was' and 'higher' make large contributions to the prediction of the characters in 'than' as measured by the norm of the $\kappa^t_{\text{'\_was'}}$ and $\kappa^t_{\text{'\_higher'}}$.

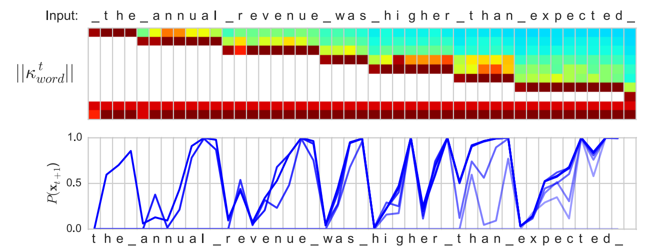

Figure 5. The ISAN architecture can be used to precisely characterize the relationship between words and characters. The top panel shows how exploiting the linearity of the network's operation we can combine the $\boldsymbol{\kappa}^t_{s_1}..\boldsymbol{\kappa}^t_{s_n}$ in a word to a single contribution, $\boldsymbol{\kappa}^t_{word}$, for each word. Shown is the norm of $\boldsymbol{\kappa}^t_{word}$, a measure of the magnitude of the effect of the previous word on the selection of the current character (red corresponds to a norm of 10, blue to 0). The bottom panel shows the probabilities assigned by the network to the next sequence character. Lighter lines show predictions conditioned on a decreasing number of preceding words. For example, when predicting the characters of 'than' there is a large contribution from both $\boldsymbol{\kappa}^t_{\text{'was'}}$ and $\boldsymbol{\kappa}^t_{\text{'higher'}}$, as shown in the top pane. The effect on the log probabilities can be seen in the bottom panel as the model becomes less confident when excluding $\boldsymbol{\kappa}^t_{\text{'was'}}$ and significantly less confident when excluding both $\boldsymbol{\kappa}^t_{\text{'was'}}$ and $\boldsymbol{\kappa}^t_{\text{'higher'}}$. This word based representation clearly shows that the system leverages contextual information across multiple words.

3.5. Change of basis

We are free to perform a change of basis on the hidden state, and then to run the affine ISAN dynamics in that new basis. Note that this change of basis is not possible for other RNN architectures, since the action of the nonlinearity depends on the choice of basis.

In particular we can construct a 'readout basis' that explicitly divides the latent space into a subspace $\mathbf{P}^{ro}_{\parallel}$ spanned by the rows of the readout matrix $\mathbf{W}_{ro}$, and its orthogonal complement $\mathbf{P}^{ro}_{\perp}$. This representation explicitly divides the hidden state dynamics into a 27-dimensional 'readout' subspace that is accessed by the readout matrix to make predictions, and a 'computational' subspace comprising the remaining $216 - 27$ dimensions that are orthogonal to the readout matrix.

We apply this change of basis to analyze an intriguing observation about the hidden offsets $\mathbf{b}_{\mathbf{x}}$. As shown in Figure 6, the norm of the $\mathbf{b}_{\mathbf{x}}$ is strongly correlated to the log-probability of the unigram $\mathbf{x}$ in the training data. Re-expressing network parameters using the 'readout basis' shows that this correlation is not related to reading out the next-step prediction. This is because the norm of the projection of $\mathbf{b}_{\mathbf{x}}$ into $\mathbf{P}^{ro}_{\perp}$ remains strongly correlated with character frequency, while the projection into $\mathbf{P}^{ro}_{\parallel}$ shows little correlation. This indicates that the information content or 'surprise' of a letter is encoded through the norm of the

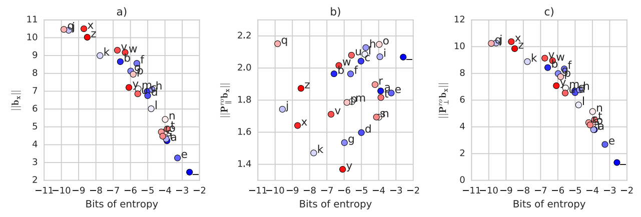

Figure 6. By transforming the ISAN dynamics into a new basis, we can better understand the action of the input-dependent biases. *a)* We observe a strong correlation between the norms of the input dependent biases, $\mathbf{b}_{\mathbf{x}}$, and the log-probability of the unigram $\mathbf{x}$ in the training data. We can begin to understand this correlation structure using a basis transform into the 'readout basis'. Breaking out the norm into its components in $\mathbf{P}^{ro}_{\parallel}$ and $\mathbf{P}^{ro}_{\perp}$ in *b)* and *c)* respectively, shows that the correlation is due to the component orthogonal to $\mathbf{W}_{ro}$. This implies a connection between information or 'surprise' and distance in the 'computational' subspace of state space.

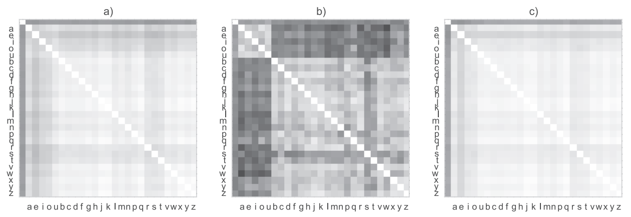

Figure 7. By transforming ISAN dynamics into a new basis, we can better interpret structure in the input-dependent biases. In *a)* we show the cosine distance between the input dependent bias vectors, split between vowels and consonants (' ' is first). In *b)* we show the correlation only considering the components in the subspace $\mathbf{P}^{ro}_{\parallel}$ spanned by the rows of the readout matrix $\mathbf{W}_{ro}$. *c)* shows the correlation of the components in the orthogonal complement $\mathbf{P}^{ro}_{\perp}$. In all plots white corresponds to 0 (aligned) and black to 2.

component of $\mathbf{b}_{\mathbf{x}}$ in the computational space, rather than in the readout space.

Similarly, in Figure 7 we illustrate that the structure in the correlations between the biases $\mathbf{b}_{\mathbf{x}}$ (across all $\mathbf{x}$) is due to their components in $\mathbf{P}^{ro}_{\parallel}$, while the correlation in $\mathbf{P}^{ro}_{\perp}$ is relatively uniform. We can clearly see two blocks of high correlations between the vowels and consonants respectively, while $\mathbf{b}_{\text{'\_'}}$ is uncorrelated to either.

3.6. Comparison with n-gram model with back-off

We compared the computation performed by n-gram language models and those performed by the ISAN. An n-gram model with back-off weights expresses the conditional probability $p\left(\mathbf{x}_t|\mathbf{x}_1...\mathbf{x}_{t-1}\right)$ as a sum of smoothed count ratios of n-grams of different lengths, with the contribution of shorter n-grams down-weighted by back-off weights. On the other hand, the computations performed by the ISAN start with the contribution of $\mathbf{b}_{ro}$ to the logits, which as shown in Figure 8a, corresponds to the unigram log-probabilities. The logits are then additively updated with contributions from longer n-grams, represented by $\boldsymbol{\kappa}_s^t$. This additive contribution to the logits corresponds to a multiplicative modification of the emission probabilities from histories of different length. For long time lags, the additive correction to log-probabilities becomes small (Figure 2), which corresponds to multiplication by a uniform distribution. Despite these differences in how n-gram history is incorporated, we nevertheless observe an agreement between empirical models estimated on the training set and model predictions for unigrams and bigrams. Figure 8 shows that the bias term $\mathbf{b}_{ro}$ gives the unigram probabilities of letters, while the addition of the offset terms $\mathbf{b}_\mathbf{x}$ accurately predict the bigram distribution of $P\left(\mathbf{x}_{t+1}|\mathbf{x}_t\right)$. Shown in panel b is an example, $P\left(\mathbf{x}|\text{'\_'}\right)$, and in panel c, a summary plot for all 27 letters.

We further explore the n-gram comparison by artificially limiting the length of the character history that is available to the ISAN for making predictions, as shown in Figure 4c).

4. Analyses of a parentheses counting task

To show the possibility of complete interpretability of the ISAN we train a model on a parenthesis counting task. Bringing together ideas from section 3.5 we re-express the transition dynamics in a new basis that fully reveals performed computations.

We analyze the task of counting the nesting levels of multiple parentheses types, a simplified version of a task defined in (Collins et al., 2016). Briefly, a 35-unit ISAN is required to keep track of the nesting level of 2 different types of parentheses independently. The inputs are the one-hot encoding of the different opening and closing parentheses (e.g. '(', ')', '[', ']') as well as a noise character ('a'). The output is the one-hot encoding of the nesting level between (0-5), one set of counts for each parenthesis type (so the complete output vector is a 12 dimensional 2-hot vector). Furthermore, the target output is the nesting level *at the previous time step*. This artificial delay requires the model to develop a memory. One change from (Collins et al., 2016) is that we exchange the cross-entropy error with an $L2$ error. This leads to slightly cleaner figures, but does not qualitatively change the results.

To elucidate the mechanism of ISAN's operation we first re-express the affine transitions $\mathbf{h}_{t+1} = \mathbf{W}\mathbf{h}_t + \mathbf{b}$ by their linear equivalents $\mathbf{h}'_{t+1} = \mathbf{W}'\mathbf{h}'_t$, where $\mathbf{W}' = [\mathbf{W}\ \mathbf{b}; \mathbf{0}^T\ 1]$ and $\mathbf{h}'_t = [\mathbf{h}_t; 1]$. Next, we used linear regression to find a change of basis for which all augmented character matrices and the hidden states are sparse. To do this we construct the 'readout' ($\mathbf{P}_{\parallel}^{ro}$) and 'computational' ($\mathbf{P}_{\perp}^{ro}$) subspace decomposition as discussed in Section 3.5. We choose a basis for $\mathbf{P}_{\perp}^{ro}$ which makes the projections of the hidden

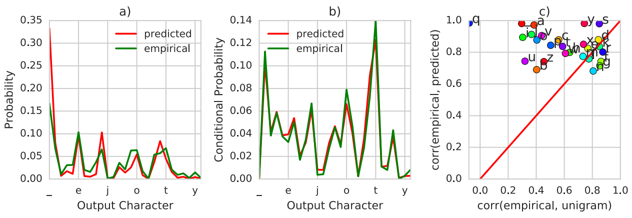

Figure 8. The predictions of ISAN for one and two characters well approximate the predictions of unigram and bigram models. In *a)* we compare softmax($\mathbf{b}_{ro}$) to the empirical unigram distribution $P(\mathbf{x})$. In *b)* we compare softmax($\mathbf{W}_{ro}\mathbf{b}_{\text{'\_'}} + \mathbf{b}_{ro}$) with the empirical distribution $P(\mathbf{x}_{t+1}|\text{'\_'})$. In *c)* we show the correlation of softmax($\mathbf{W}_{ro}\mathbf{b}_\mathbf{x} + \mathbf{b}_{ro}$) with $P(\mathbf{x}_{t+1}|\mathbf{x}_t)$ for all 27 characters (y-axis), and compare this to the correlation between the empirical unigram probabilities $P(\mathbf{x})$ to $P(\mathbf{x}_{t+1}|\mathbf{x}_t)$ (x-axis). The plot shows that the readout of the bias vector is a better predictor of the conditional distribution than the unigram probability.

states into this computational subspace 2-hot vectors. With this subspace decomposition, the hidden states and character matrices have the form

$$\mathbf{W}'_x = \begin{bmatrix} \mathbf{W}_x^{rr} & \mathbf{W}_x^{rc} & \mathbf{b}_x^r \\ \mathbf{W}_x^{cr} & \mathbf{W}_x^{cc} & \mathbf{b}_x^c \\ \mathbf{0}^T & \mathbf{0}^T & 1 \end{bmatrix} \qquad \mathbf{h}'_t = \begin{bmatrix} \mathbf{h}_t^r \\ \mathbf{h}_t^c \\ 1 \end{bmatrix} \tag{7}$$

and the update equation can be written as

$$\mathbf{h}'_{t+1} = \mathbf{W}'_x\mathbf{h}'_t = \begin{bmatrix} \mathbf{W}_x^{rr}\mathbf{h}_t^r + \mathbf{W}_x^{rc}\mathbf{h}_t^c + \mathbf{b}_x^r \\ \mathbf{W}_x^{cr}\mathbf{h}_t^r + \mathbf{W}_x^{cc}\mathbf{h}_t^c + \mathbf{b}_x^c \\ 1 \end{bmatrix}. \tag{8}$$

Here $\mathbf{h}_t^r$ and $\mathbf{h}_t^c$ denote the readout and computational portions of $\mathbf{h}_t$, and $\mathbf{W}_x^{rr}, \mathbf{W}_x^{cr}, \mathbf{W}_x^{rc}, \mathbf{W}_x^{cc}$ denote the readout to readout, readout to computation, computation to readout, and computation to computation blocks of the character matrix for character x, respectively.

In Figure 9d we show the hidden states in the rotated basis as a sequence of column vectors. The 35 dimensional hidden states are all 4-hot. We can treat them as a concatenation of a readout $\mathbf{h}_t^r$ and a computation $\mathbf{h}_t^c$ part. The 12-dimensional readout $\mathbf{h}_t^r$ corresponds to network's output at time step t and encodes the counts from time step $t-1$ as a 2-hot vector (one count per parenthesis type). The computational space $\mathbf{h}_t^c$ is $35 - 12 = 23$ dimensional, and encodes the current counts as another 2-hot vector. Note that in this basis the ISAN effectively uses only 24 dimensions and the remaining 11 dimensions have no noticeable effect on the computation. In Figure 9c we show $\mathbf{W}'_{[}$ in the rotated basis. We see from the leftmost 12 columns that $\mathbf{W}_{[}^{rr}$ and $\mathbf{W}_{[}^{cr}$ are both nearly 0. This means that $\mathbf{h}_t^r$ has no influence on $\mathbf{h}_{t+1}$. Furthermore, the computation to readout block, $\mathbf{W}_{[}^{rc}$, is identity on the first 12 dimensions, effectively implementing the lagging output $\mathbf{h}^r_t = \mathbf{h}^c_{t-1}$. The current counts are implemented as delay lines and identity sub-matrices in $\mathbf{W}^{cc}_{[}$, which respectively has the effect of incrementing the count of '[' by one, saturating at 5, and leaving the count of '()' parentheses fixed. The matrices $\mathbf{W}_{]}, \mathbf{W}_{(}, \mathbf{W}_{)}$ behave analogously. It is clear that this solution is general, in that retraining for increased numbers of parentheses types or an increased counting maximum, would have the analogous solution.

5. Discussion

In this paper we motivated an input-switched affine recurrent network for the purpose of interpretability. We showed that a switched affine architecture achieves the same performance as LSTMs on the Text8 dataset for the same number of maximum parameters, and reasonable performance on the BWB. We performed a series of analyses, demonstrating the ability to understand how inputs at one point in the input sequence affect the outputs later in the output sequence. We showed further in the multiple parentheses counting task that the ISAN dynamics can be completely reverse engineered. In summary, this work provides evidence that the ISAN is able to express complex dynamical systems, yet its operation can in principle be fully understood, a prospect that remains out of reach for many popular recurrent architectures.

5.1. Computational benefits

Switched affine networks hold the potential to be massively more computationally and memory efficient for text processing than other recurrent architectures. First, input-dependent affine transitions reduce the number of parameters used at every step. For K possible inputs and N parameters, the computational cost per update step is $O\left(\frac{N}{K}\right)$, a factor of K speedup over non-switched architectures. Similarly, the number of hidden units is $O\left(\sqrt{\frac{N}{K}}\right)$, a factor of $K^{\frac{1}{2}}$ memory improvement for storage of the latent state.

Furthermore, the ISAN is unique in its ability to precompute affine transformations corresponding to input strings. This is possible because the composition of affine transformations is also an affine transformation. This property is used in Section 3.4 to evaluate the linear contributions of words, rather than characters. This means that the hidden state update corresponding to an entire input sequence can be computed with identical cost to the update for a single character (plus the dictionary look-up cost for the composed transformation). ISAN can therefore achieve very large speedups on input processing, at the cost of increased memory use, by accumulating large look-up tables of the $\mathbf{W}_\mathbf{x}$ and $\mathbf{b}_\mathbf{x}$ corresponding to common input sequences. Of course, practical implementations will have to incorporate complexities of memory management, batching, etc.

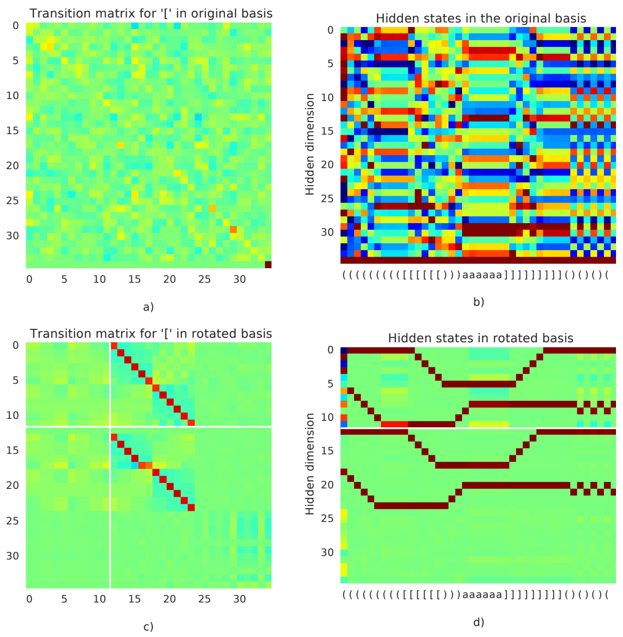

Figure 9. A visualization of the dynamics of an ISAN for the two parentheses counting task with 1 time lag (count either '()' or '[]' nesting levels with a one-step readout delay). In *a)* the weight matrix for '[' is shown in the original basis. In *c)* it is shown transformed to highlight the delay-line dynamics. The activations of the hidden units are shown *b)* in the original basis, and *d)* rotated to the same basis as in c), to highlight the delay-line dynamics in a more intelligible way. The white line delineates the transition matrix elements and hidden state dimensions that directly contribute to the output. All matrices for parentheses types appear similarly, with closing parentheses, e.g. ']', changing the direction of the delay line.

5.2. Future work

There are some obvious future directions to this work. Currently, we define switching behavior using an input set with finite and manageable cardinality. Studying word-level language models with enormous vocabularies may require some additional logic to scale. Another idea is to build a language model that switches on bigrams or trigrams, rather than characters or words, targeting an intermediate number of affine transformations. Adapting this model to continuous-valued inputs is another important direction. One approach is to use a tensor factorization similar to that employed by the MRNN (Sutskever et al., 2014) or defining weights via additional networks, as in HyperNetworks (Ha et al., 2016). Finally, we expect that automated methods for changing bases to enable sparse representations in the hidden state and dynamics matrices will be a particularly fruitful direction to pursue.

Acknowledgements

We would like to thank Jasmine Collins for her help and advice, and Quoc Le, David Ha and Mohammad Norouzi for helpful discussions. We would also like to thank Herbert Jaeger for insightful discussions regarding the Observable-Operator-Model.

References

Alain, Guillaume and Bengio, Yoshua. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

Andrew, Alex M. An introduction to support vector machines and other kernel-based learning methods. *Kybernetes*, 2013.

Belanger, David and Kakade, Sham. A linear dynamical system model for text. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 833–842, 2015.

Berk, Richard, Heidari, Hoda, Jabbari, Shahin, Kearns, Michael, and Roth, Aaron. Fairness in criminal justice risk assessments: The state of the art. *arXiv preprint arXiv:1703.09207*, 2017.

Bojarski, Mariusz, Del Testa, Davide, Dworakowski, Daniel, Firner, Bernhard, Flepp, Beat, Goyal, Prasoon, Jackel, Lawrence D, Monfort, Mathew, Muller, Urs, Zhang, Jiakai, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling. *ArXiv e-prints*, December 2013.

Ching, Travers, Himmelstein, Daniel S, Beaulieu-Jones, Brett K, Kalinin, Alexandr A, Do, Brian T, Way, Gregory P, Ferrero, Enrico, Agapow, Paul-Michael, Xie, Wei, Rosen, Gail L, et al. Opportunities and obstacles for deep learning in biology and medicine. *bioRxiv*, pp. 142760, 2017.

Collins, Jasmine, Sohl-Dickstein, Jascha, and Sussillo, David. Capacity and trainability in recurrent neural networks. *ICLR 2017 submission*, 2016.

Council of European Union. General Data Protection Regulation, Article 22 (Regulation (EU) 2016/679), 2016. URL `http://www.privacy-regulation.eu/en/22.htm`.

Deo, Rahul C. Machine learning in medicine. *Circulation*, 132(20):1920–1930, 2015.

Freedman, David A. *Statistical models: theory and practice*. cambridge university press, 2009.

Gulshan, Varun, Peng, Lily, Coram, Marc, Stumpe, Martin C, Wu, Derek, Narayanaswamy, Arunachalam, Venugopalan, Subhashini, Widner, Kasumi, Madams, Tom, Cuadros, Jorge, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *JAMA*, 316(22):2402–2410, 2016.

Ha, David, Dai, Andrew, and Le, Quoc V. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

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

Hwang, Kyuyeon and Sung, Wonyong. Character-level language modeling with hierarchical recurrent neural networks. *CoRR*, abs/1609.03777, 2016. URL `http://arxiv.org/abs/1609.03777`.

Jaeger, Herbert. Observable operator models for discrete stochastic time series. *Neural Computation*, 12(6):1371–1398, 2000.

Józefowicz, Rafal, Vinyals, Oriol, Schuster, Mike, Shazeer, Noam, and Wu, Yonghui. Exploring the limits of language modeling. *CoRR*, abs/1602.02410, 2016. URL `http://arxiv.org/abs/1602.02410`.

Karpathy, Andrej, Johnson, Justin, and Li, Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Katz, Guy, Barrett, Clark, Dill, David, Julian, Kyle, and Kochenderfer, Mykel. Reluplex: An efficient smt solver for verifying deep neural networks. *arXiv preprint arXiv:1702.01135*, 2017.

Le, Quoc V, Ranzato, Marc A., Monga, Rajat, Devin, Matthieu, Chen, Kai, Corrado, Greg S., Dean, J, and Ng, Andrew Y. Building high-level features using large scale unsupervised learning. In *International Conference on Machine Learning*, 2012.

Linderman, Scott W, Miller, Andrew C, Adams, Ryan P, Blei, David M, Paninski, Liam, and Johnson, Matthew J. Recurrent switching linear dynamical systems. *arXiv preprint arXiv:1610.08466*, 2016.

Mahoney, Matt. Large text compression benchmark: About the test data, 2011. URL `http://mattmahoney.net/dc/textdata`. [Online; accessed 15-November-2016].

Martens, James and Sutskever, Ilya. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1033–1040, 2011.

Mikolov, Tomáš, Sutskever, Ilya, Deoras, Anoop, Le, Hai-Son, and Kombrink, Stefan. Subword language modeling with neural networks. *preprint*, 2012.

Mordvintsev, Alexander, Olah, Christopher, and Tyka, Mike. Inceptionism: Going deeper into neural networks. *Google Research Blog. Retrieved June*, 20:14, 2015.

Murdoch, W. James and Szlam, Arthur. Automatic rule extraction from long short term memory networks. In *ICLR*, 2017.

Quinlan, J. Ross. Simplifying decision trees. *International journal of man-machine studies*, 27(3):221–234, 1987.

Scarborough, David and Somers, Mark John. *Neural networks in organizational research: Applying pattern recognition to the analysis of organizational behavior.* American Psychological Association, 2006.

Siano, Pierluigi, Cecati, Carlo, Yu, Hao, and Kolbusz, Janusz. Real time operation of smart grids via fcn networks and optimal power flow. *IEEE Transactions on Industrial Informatics*, 8(4):944–952, 2012.

Sussillo, David and Barak, Omri. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation*, 25(3):626–649, 2013.

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

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Tashea, Jason. Courts are using ai to sentence criminals. that must stop now. *WIRED magazine*, 2017.

Zeiler, Matthew D, Krishnan, Dilip, Taylor, Graham W, and Fergus, Rob. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2528–2535. IEEE, 2010.