Gesture Recognition from Accelerometer Data with Recurrent Neural Networks

Justin Fu
University of California, Berkeley
justinfu@berkeley.edu

Siddhartho Bhattacharya
University of California, Berkeley
siddartho_b@hotmail.com

Abstract

Gesture recognition is typically done through images, but it requires extensive effort to process image data into a form useful for gesture recognition. Our project uses accelerometer data from smartphones, so no special hardware is required. Recurrent neural networks provide a natural way to model time sequences, and the LSTM architecture in particular provides a way to model long-range dependencies. We imlemented 6 gestures, and were able to to correctly identify 53 of 60 (88%) of gestures during test time with an ensemble approach.

1 Introduction

Many gesture recognition systems use DTW (dynamic time warping) or HMMs (Hidden Markov Models) to model gestures. We wanted to see how effective recurrent neural networks could perform in such a task.

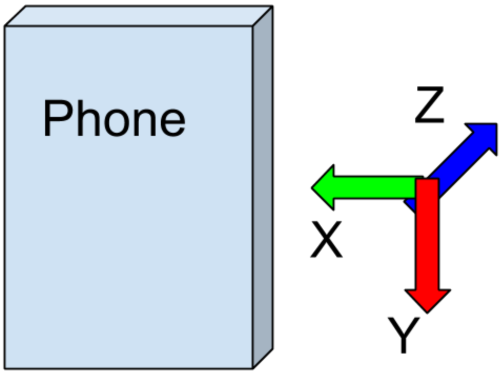
We had previously tried gesture recognition with a non-recurrent network by feeding in a fixed number of past inputs into the network. However, this approach means that the network loses all information before the time window. LSTMs were chosen to remedy this problem.

Potential applications of this project include making phones more intuitive to use. For example, imagine phones that automatically unlock and lock when you take them in and out of your pocket, picking up calls by just bringing it up to your ear, etc.

2 Data

We collected accelerometer data streamed from a smartphone. Specifically, we only considered values in the X and Y directions as seen from the perspective of the phone.

Figure 1: Directions of each axis on the phone



We implemented 6 different gestures representing some letters of the alphabet (M, Z, O, L, J, and S). Our training set contains 26 instances of each letter and our test set contains 10, for a total of 216 examples.

We introduced much more variation in size and speed of gestures in the test set to see how well the network could generalize.

3 Model

We implemented an LSTM (long short term memory) network based on the architecture proposed by Graves [1]. Specifically, our network contains the forget, input, and output gates, but lacks the peephole connections that connect the internal cell state to these gates (as done in Vinyals et al. [2]).

We then interleave layers of LSTM units with normal network layers to create a multilayer network. The output of the final layer is then fed through a softmax function.

3.1 Equations

We used cross-entropy as our objective function. O is the network output after the softmax and Y is the training label. n sums over each training exam-

Figure 2: A single LSTM cell. Red arrows are connections forward in time.

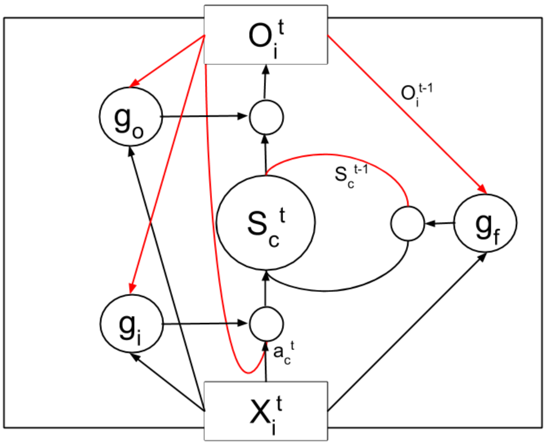
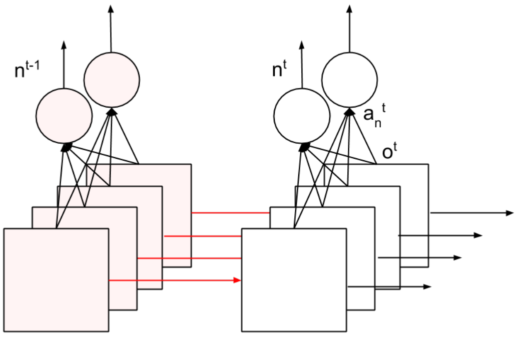


Figure 3: Diagram of one interleaved layer of our network. LSTM cells are represented as squares and normal neurons are represented as circles.



ple, t sums over time, and i sums over the dimensions of each training example at a given timestep. Note that t varies with each training example.

$$L(Y,O) = \sum_{n}^{N} \sum_{t}^{len(O^n)} \sum_{i}^{I} y_i^{n,t} * \ln(o_i^{n,t})$$

Most of the equations on the forward and backward passes are the same as those in Graves [1], but some were missing from his paper and we dropped the peepholes so we specify all for completeness.

3.2 Forward Dynamics

The forward dynamics of the network are specified as follows. Equations are specified for a whole layer (rather than per unit). Bolded values represent vectors. Note that $*$ represents a pointwise multiply, and the nonlinearities are applied element-wise to the input vector.

Gates

$$\mathbf{a}_i^t = W_{i,x}\mathbf{x}^t + W_{i,o}\mathbf{o}^{t-1}$$

$$\mathbf{g}_i^t = \sigma(\mathbf{a}_i^t)$$

$$\mathbf{a}_f^t = W_{f,x}\mathbf{x}^t + W_{f,o}\mathbf{o}^{t-1}$$

$$\mathbf{g}_f^t = \sigma(\mathbf{a}_f^t)$$

$$\mathbf{a}_o^t = W_{o,x}\mathbf{x}^t + W_{o,o}\mathbf{o}^{t-1}$$

$$\mathbf{g}_o^t = \sigma(\mathbf{a}_o^t)$$

Cell State

$$\mathbf{a}_c^t = W_{c,x}\mathbf{x}^t + W_{c,o}\mathbf{o}^{t-1}$$

$$\mathbf{s}_c^t = \mathbf{g}_i^t * \sigma(\mathbf{a}_c^t) + \mathbf{g}_f^t * \mathbf{s}_c^{t-1}$$

Cell Output (Hidden State)

$$\mathbf{o}^t = \mathbf{g}_i^t * \tanh(\mathbf{s}_c^t)$$

Normal Network Layer Output

$$\mathbf{a}_n^t = W_n\mathbf{o}^t$$

$$\mathbf{n}^t = \tanh(\mathbf{a}_n^t)$$

3.3 Backpropagation

$\delta_k^{t,l+1}$ represents the error backpropogating from the above layer in the current time step. The value of this at the top layer is the derivative of the objective with respect to the network output, which is $softmax(networkOutput) - trainingLabels$. As with before, $*$ represents a pointwise multiplication. The δ variables represent vectors for the whole layer.

Cell Output

$$\delta_n^t = \frac{\partial L}{\partial \mathbf{a}_n^t} = \delta_k^{t,l+1} * \tanh'(\mathbf{a}_n^t)$$

$$\delta_h^t = W_{i,o}^T\delta_i^{t+1} + W_{f,o}^T\delta_f^{t+1} + W_{o,o}^T\delta_o^{t+1} + W_{c,o}^T\delta_s^{t+1}$$

$$\frac{\partial L}{\partial \mathbf{o}^t} = W_n^T\delta_n^t + \delta_h^t$$

Cell State

$$\delta_c^t = \frac{\partial L}{\partial \mathbf{s}_c^t} = \mathbf{g}_o^t * \tanh'(\mathbf{s}_c^t) * \frac{\partial L}{\partial \mathbf{o}^t} + \delta_c^{t+1} * \mathbf{g}_f^{t+1}$$

$$\delta_s^t = \frac{\partial L}{\partial \mathbf{a}_c^t} = \mathbf{g}_i^t * \sigma'(\mathbf{a}_c^t) * \delta_c^t$$

Gates

$$\delta_o^t = \frac{\partial L}{\partial \mathbf{a}_o^t} = \sigma'(\mathbf{a}_o^t) * \tanh(\mathbf{s}_c^t) * \frac{\partial L}{\partial \mathbf{o}^t}$$

$$\delta_f^t = \frac{\partial L}{\partial \mathbf{a}_f^t} = \sigma'(\mathbf{a}_f^t) * \mathbf{s}_c^{t-1} * \delta_c^t$$

$$\delta_i^t = \frac{\partial L}{\partial \mathbf{a}_i^t} = \sigma'(\mathbf{a}_i^t) * \sigma(\mathbf{a}_c^t) * \delta_c^t$$

Input

$$\delta_k^{t,l} = W_{i,x}^T \delta_i^t + W_{f,x}^T \delta_f^t + W_{o,x}^T \delta_o^t + W_{c,x}^T \delta_s^t$$

3.4 Gradient

The gradient of a matrix W is calculated by taking the outer product of its input and the delta of its output.

For example, here is the derivative of the matrix of the normal layer output W_n. The input to this matrix are the cell outputs $\mathbf{o}^t$ and its output, $\mathbf{a}_n$ has the delta δ_n^t

$$\frac{\partial L}{\partial W_n} = outer(\mathbf{o}^t, \delta_n^t)$$

These gradients are summed over all time steps and all training examples per weight update.

3.5 Training

We trained our network using BPTT (backpropogation through time) and gradient descent with momentum, using the equations described in the previous section. All weights are randomly initialized in the range [-0.1, 0.1].

Training inputs (acceleration values) are directly presented to the network on the lowest layer. We present training labels to the network as a one-hot vector representing the gesture to the network at all time steps.

3.6 Decoding

To retrieve a single gesture prediction from the network, we run a forward pass and collect the network outputs. Then, we average the network outputs across time, use a softmax, and select the class c with the maximum average response.

$$c = \arg\max_i(softmax(\sum_t o_i^t))$$

Since we also tried an ensemble approach, we summed the outputs of each network and then selected the argmax.

$$c = \arg\max_i(\sum_{net} softmax(\sum_t o_i^{t,net}))$$

4 Results

As we collected data ourselves, there were no avilable datasets online to compare our architecture against. As mentioned before, out training set consisted of 26 instances of each letter, and our test set contains 10.

We trained 3 identical networks. Each network consists of 4 layers - 10 LSTM units, 10 normal units, 6 LSTM units, and 6 normal units.

Gesture	Net 1	Net 2	Net 3	Ensemble
M	0.0	0.0	0.0	0.0
Z	0.0	1.0	0.0	0.0
O	0.2	0.0	0.4	0.1
L	0.9	0.1	0.0	0.3
S	0.2	0.7	0.2	0.3
J	0.1	0.0	1.0	0.0
Total	0.233	0.3	0.267	.117

Table 1: Gesture error rates.

Commonly confused characters included the pairs (L, J), (O,S), and (S,Z). In fact, each network had a character pair it could not identify. Since each network got trapped in a different local minimum, they made different errors, but the ensemble approach allowed other networks to ”outvote” the mistakes made by one network.

It is interesting that the character M was the only one that all networks correctly identified 100% of the time. This could be due to the fact that there were no other characters similar to it in our experiments.

5 Conclusion

In general, we were very surprised at how accurate the results we obtained were - the networks were able to generalize very well to variations in speed and size of gestures. We were able to obtain very reasonable results even with the small amount of training data we had and the limited computing resources we had for training.

One difficulty we encountered was that the model had trouble differentiating certain pairs of gestures as we scaled up the number of gestures; 4 gestures (M, Z, O, L) were easily learned by a 2-layer network, but we stopped at 6 gestures because it was getting very difficult. We coped with this by adding more neurons, more layers, and moving to an ensemble approach to average outputs from individually inaccurate networks.

Additional Materials

Our code and datasets are hosted on https://github.com/justinjfu/lstm

Acknowledgments

We would like to thank Brian Cheung for giving us some key pointers that were extremely helpful in implementing backpropogation for our architecture.

References

[1] Alex Graves. 2008. *Supervised Sequence Labelling with Recurrent Neural Networks*. PhD Thesis.

[2] Orial Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan. 2014. *Show and Tell: A Neural Image Caption Generator*.