Classifying EEG Spectrograms by Phalangeal Articulation using LRC Neural Networks

Robert Valencia

EE 4OJ4/EE 4OH4 Research Project

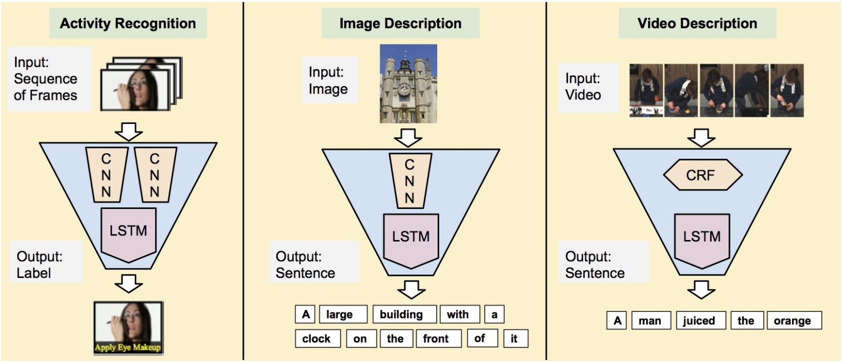
Supervisor: Dr. James P. Reilly

Problem and Approach

- Problem:
  - Given EEG data, determine which phalanges were articulated
- Approach:
  - What information can be used?
    - EEG data contains both time and frequency information
    - Spectrograms can capture frequency information over time
  - How can the data be classified?
    - Multiple machine learning algorithms exist for classification
    - Neural networks provide automated feature detection and universal approximation

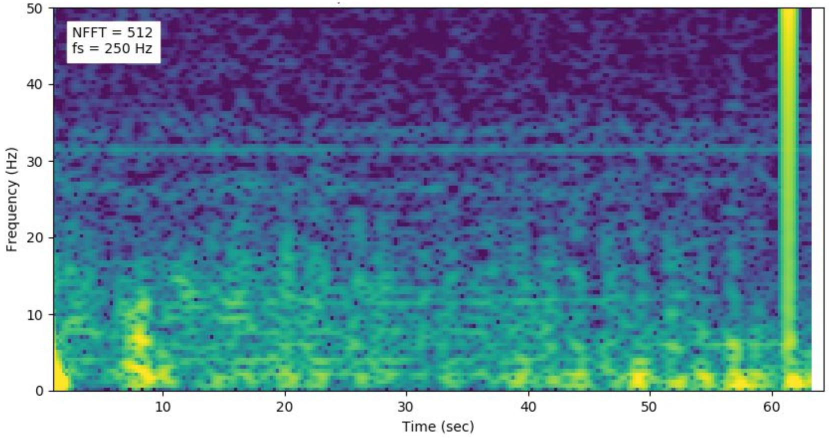
Approach (cont.)

- Various classes of neural networks
- Data involves a sequence of spectrograms
- Spectrograms similar in dimensionality as images
- Long-term Recurrent Convolutional (LRC) neural networks:
    - A class of neural networks for visual and sequence learning
    - Utilizes convolutional neural networks and long short-term memory



Applications of LRC Neural Networks
Source: http://jeffdonahue.com/lrcn/

Data - Input



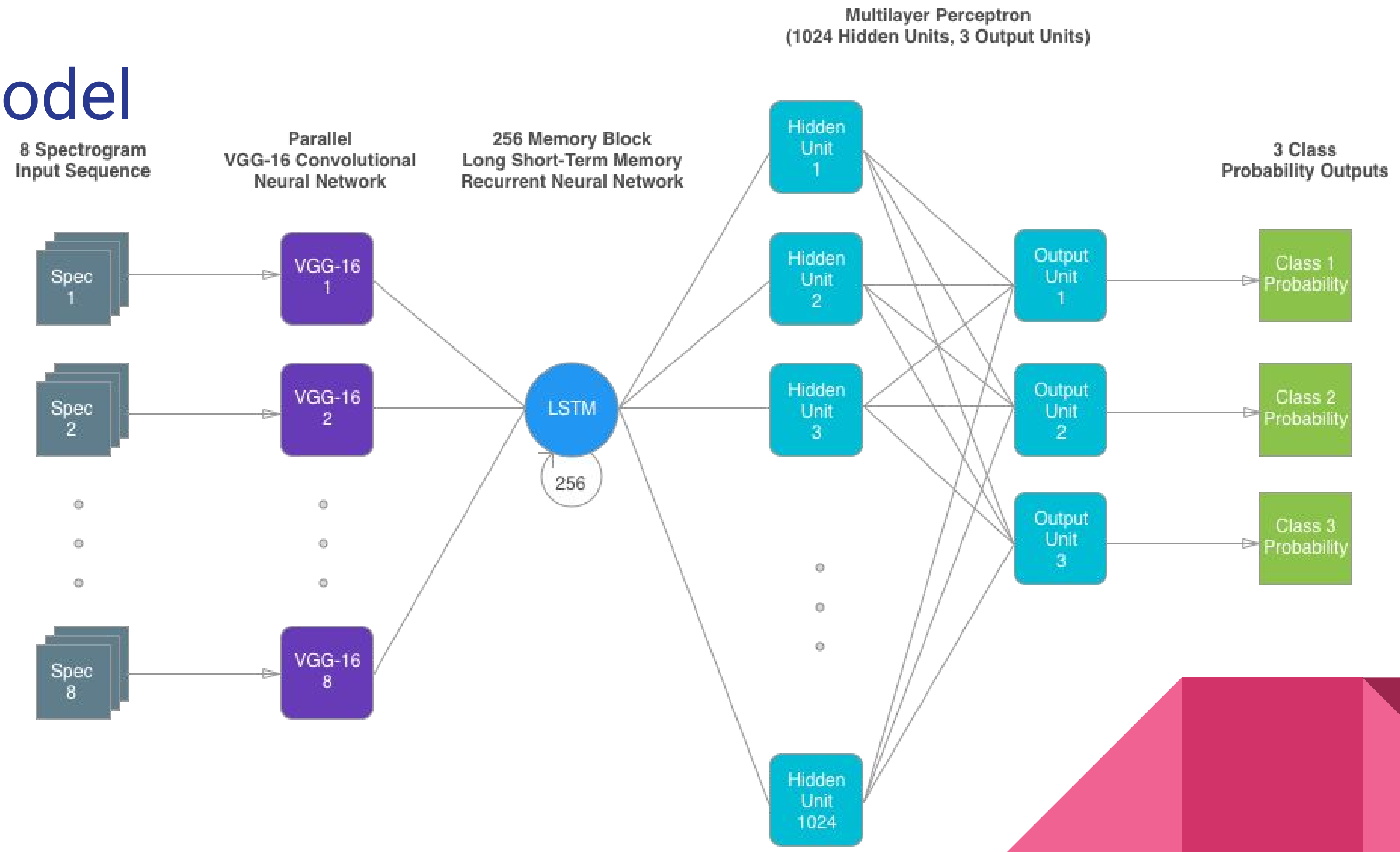
Sample spectrogram for sustained left middle finger flexion from 1 channel

- 8 EEG channels
- For each channel:
  - Set-up and teardown data trimmed
  - DC offset removed
  - Mains interference notched
  - 1-50 Hz bandpass filtered
  - Spectrograms generated
  - Spectrograms partitioned into multiple samples with dimensionality of 250 freq. points x 50 time points
- Depending on the training run configuration, samples were either replicated or augmented to increase sample size

Data - Labels and Output

- Labels:
  - One-hot encoded representation of each class
  - 3 classes: left index, left middle, and left ring finger flexion
  - e.g. [1., 0., 0.], [0., 1., 0.], [0., 0., 1.]
- Outputs:
  - Array of probabilities for each class
  - e.g. [0.50, 0.25, 0.25] would mean 50% probability for class 1, 25% probability for class 2 and 3

Model



Multilayer Perceptron
(1024 Hidden Units, 3 Output Units)
8 Spectrogram Input Sequence
Parallel VGG-16 Convolutional Neural Network
256 Memory Block Long Short-Term Memory Recurrent Neural Network
3 Class Probability Outputs
Spec 1
Spec 2
Spec 8
VGG-16 1
VGG-16 2
VGG-16 8
LSTM
256
Hidden Unit 1
Hidden Unit 2
Hidden Unit 3
Hidden Unit 1024
Output Unit 1
Output Unit 2
Output Unit 3
Class 1 Probability
Class 2 Probability
Class 3 Probability

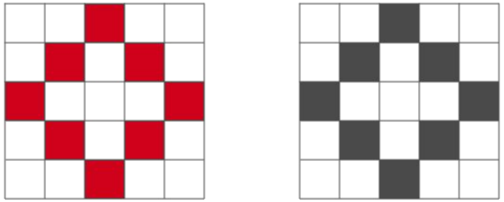
Convolutional Neural Network (CNN)

- Biologically-inspired multilayer perceptrons (MLPs)
- Mimic visual cortex:
  - Complex cell arrangement sensitive to stimuli within a restricted region (receptive field)
  - Region tiled across entire visual field
  - Cells act as localized filters for detecting spatial patterns
  - Response can be approximated by a convolution operation:

$$(f * g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f[m]\, g[n-m]$$
$$= \sum_{m=-\infty}^{\infty} f[n-m]\, g[m].$$

CNN (cont.): Simplified Illustration

- Sample 5x5 image
- Converted to grayscale
- Simplified digital representation



Sample Image

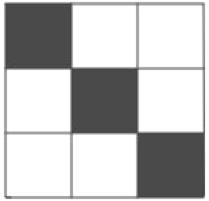
Grayscale Representation

0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
0	1	0	1	0
0	0	1	0	0

Simplified Digital Representation

CNN (cont.): Simplified Illustration

- Sample 3x3 filter
- Simplified digital representation



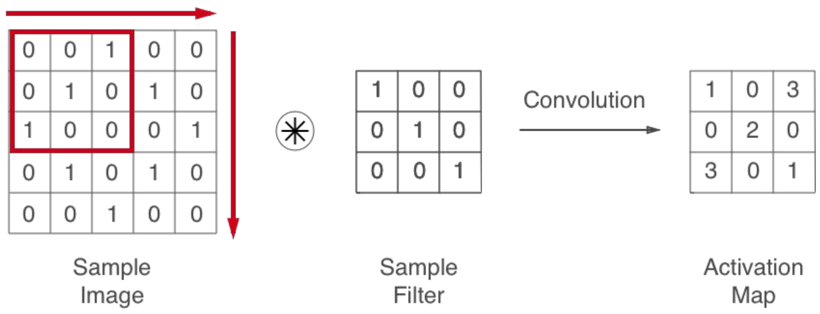
Sample
Filter

1	0	0
0	1	0
0	0	1

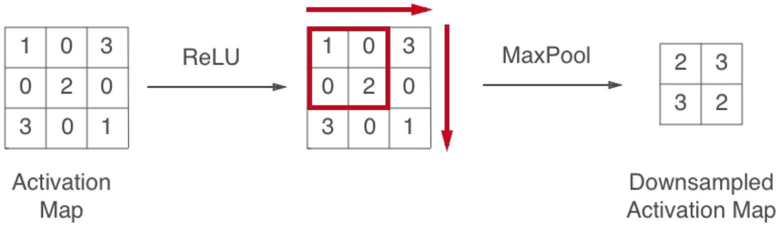
Simplified
Digital Representation

CNN (cont.): Simplified Illustration

- Convolutional filters tiled across an image
- Receptive fields convolved with visual field regions, generating an activation map
- Activation map:
    - Regions with spatial patterns with a high correlation with the filter pattern have high activation values, and vice versa
- Filter stride of 1 pixel, generating a 3x3 activation map



CNN (cont.): Simplified Illustration



$$f(x) = \max(0, x)$$

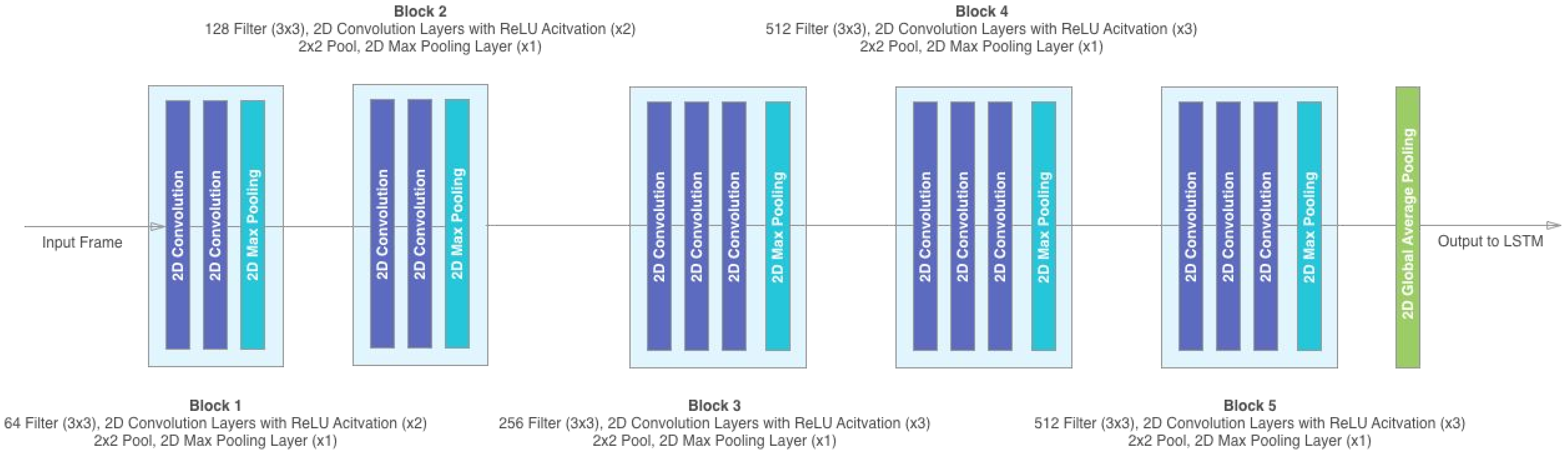
Rectified Linear Unit (ReLU)

- Activation map passed through a layer of rectified linear units (ReLUs), an activation function to improve network nonlinearity:

$$f(x) = \max(0, x)$$

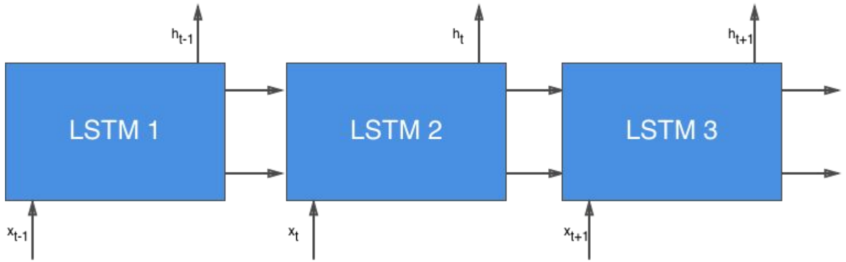
- Pooling layer downsamples activation map, reducing number of parameters
- MaxPool layer used, replacing a pool of values with its maximum value
- 2x2 pool size, stride of 1 pixel

CNN (cont.): VGG-16 Architecture

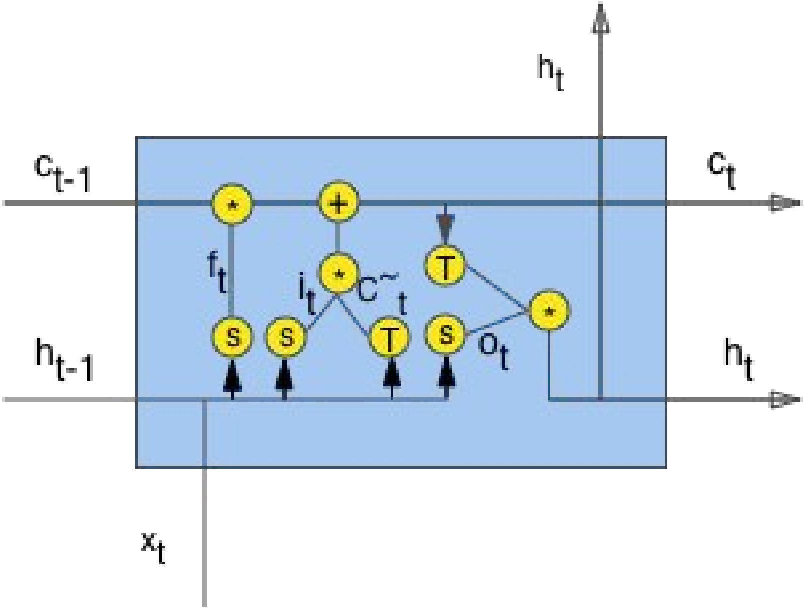


Long short-term Memory (LSTM)

- A type of recurrent neural networks (RNNs)
  - Artificial neural networks
  - Consists of stateful memory units that are cyclically connected
  - Detect sequential patterns
- LSTMs, unlike regular RNNs, are well suited for data with variable gaps between events
  - E.g. variations observed in speech due to demographic and biological variability
- LSTMs consist of chains of repeated LSTM units



LSTM (cont.)



- Within each LSTM unit, a series of operations occur:
  - S, logistic sigmoid:

$$S(t) = \frac{1}{1 + e^{-t}}$$

  - T, hyperbolic tangent:

$$T(t) = \frac{e^{2t} - 1}{e^{2t} + 1}$$

  - +, element-wise addition
  - *, element-wise multiplication
- Also utilizes weight matrices W, U, and V, and bias vector b

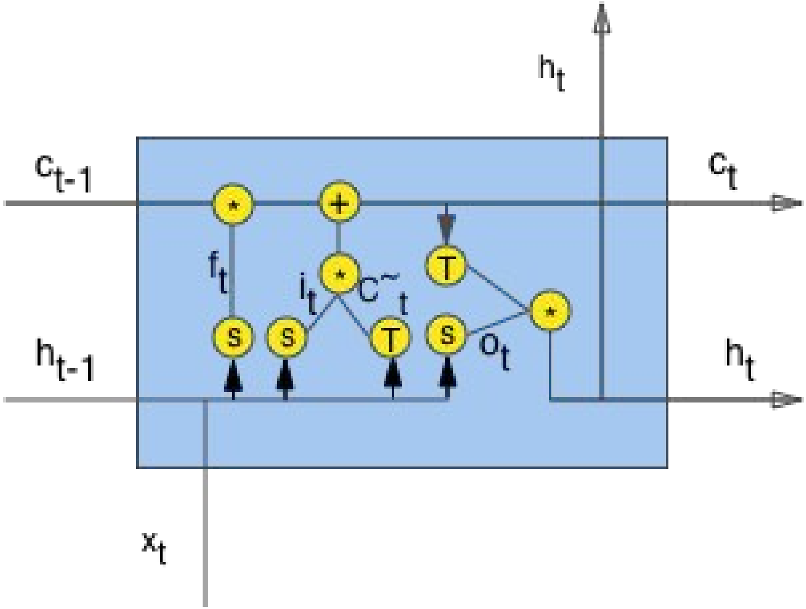
LSTM (cont.)

- First, select new data to store:
  - Logistic sigmoid (input gate) layer selects which values to update:

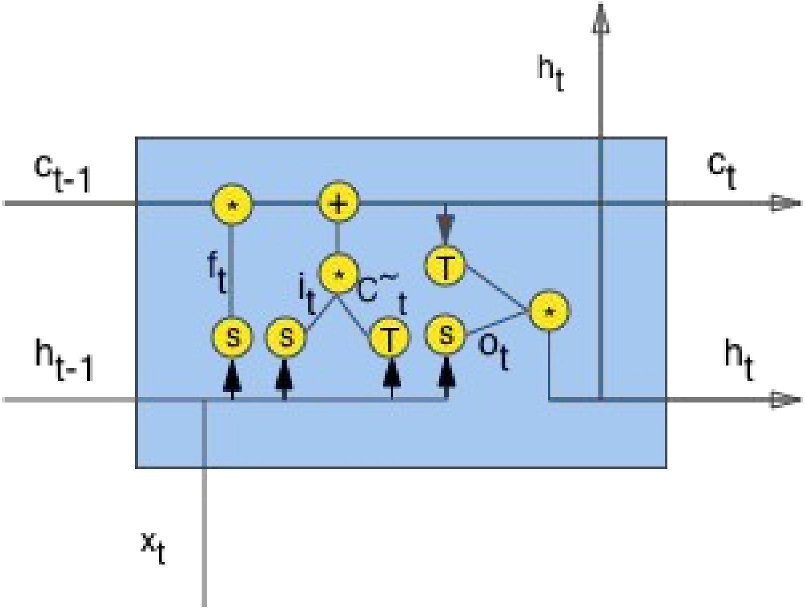
$$i_t = s(W_i x_t + U_i h_{t-1} + b_i)$$

  - Hyperbolic tangent layer generates new candidate values:

$$C^{\sim}_t = t(W_c x_t + U_c h_{t-1} + b_c)$$



LSTM (cont.)



- Next, another logistic sigmoid layer selects data to forget:

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

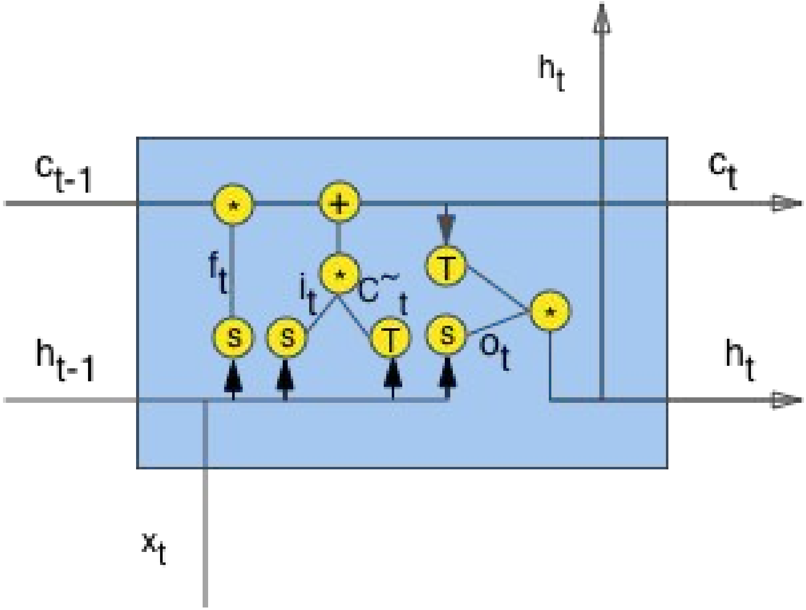
- Takes h_{t-1} and x_t, then returns either 0 or 1 for each value in cell state c_{t-1}:
  - 0 for “forget”
  - 1 for “remember”

LSTM (cont.)

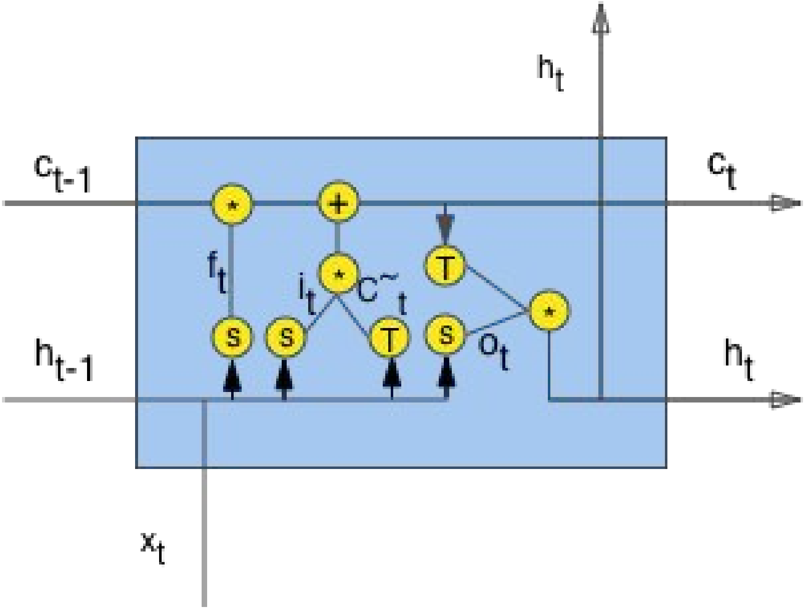
- Then, the cell state is updated from the old state C_{t-1} to the new state C_t:

$$C_t = i_t * C^{\sim}_t + f_t * C^{\sim}_{t-1}$$

- Forgets what has to be forgotten by multiplying the old state C_{t-1} with the output of the forget gate f_t
- Adds new candidate values scaled by update weights by multiplying the new state C_t with the output of the input gate i_t



LSTM (cont.)



- Finally, the output is generated:

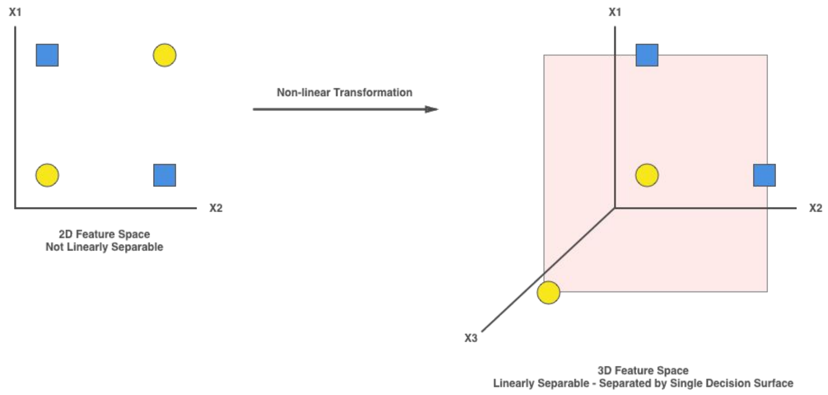
$$o_t = s(W_o x_t + U_o h_{t-1} + V_o C_t + b_o)$$

$$h_t = o_t * t(C_t)$$

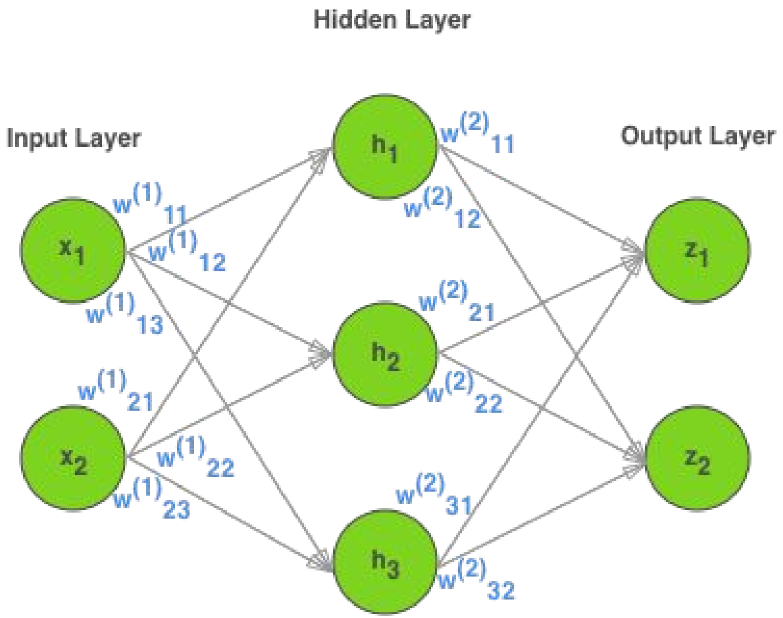
- First, a logistic sigmoid layer selects which values of the cell state to output
- Next, the cell state values pass through a hyperbolic tangent layer, scaling it to values between -1 and 1
- Finally, the outputs of the first and second steps are multiplied, resulting in a filtered cell state

Multilayer Perceptron (MLP)

- Artificial neural networks
- Consists of fully connected layers of nodes
- Map input data into outputs via a learned non-linear transformation
    - Projects input data into a space where they become linearly separable, enabling classification



MLP (cont.)



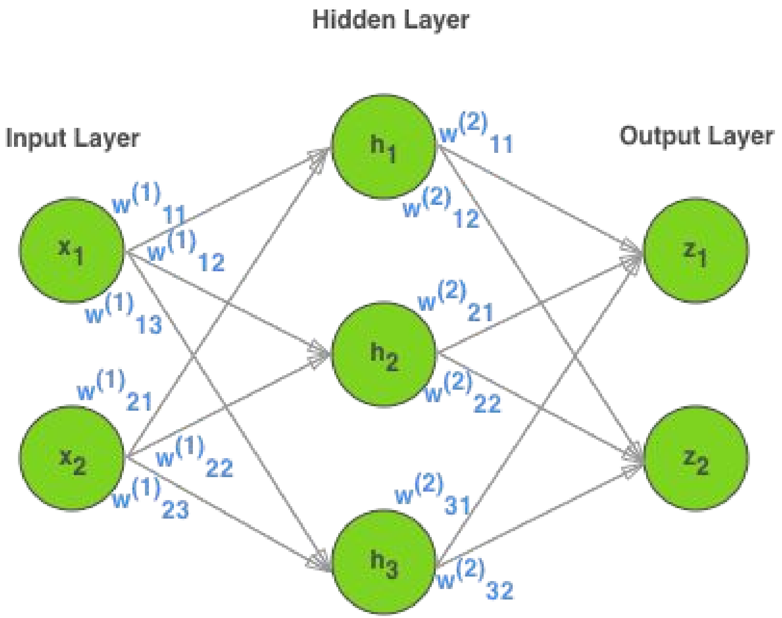
- MLPs consist of 3 primary stages:
  - Input layer
  - Hidden layer/s
  - Output layer
- With at least 1 hidden layer, MLP becomes a universal approximator
- Sample MLP with a 2-node input layer, 3-node hidden layer, and 2-node output layer
- In practical deep learning applications, multiple hidden layers are utilized to generate more features

MLP (cont.)

- Input nodes represent input features
- Hidden nodes represent generated features
- Output nodes represent class probabilities
- To make predictions, forward propagation is performed:

$$h(x) = s(b^{(1)} + w^{(1)}x)$$

$$z(h(x)) = G(b^{(2)} + w^{(2)}h(x))$$



MLP (cont.)

$$h(x) = s(b^{(1)} + w^{(1)}x)$$

$$z(h(x)) = G(b^{(2)} + w^{(2)}h(x))$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Softmax function

- **x** is the input layer vector, which contains input features
- **h** is the hidden layer vector, which contains generated features
- **z** is the output layer vector, which contains class probabilities
- **b** are bias vectors, **w** are weight matrices, which are learned parameters
- **s** is the hidden layer activation function , set to ReLU for this model
- **G** is the output layer activation function, set to the softmax function for multi-class classification

MLP (cont.)

- Initially, parameters are randomized, resulting in low prediction accuracies
- To improve accuracies, parameters are learned via the backpropagation algorithm:
  - Trains the model on labelled data
  - Updates parameters until a cost function is minimized

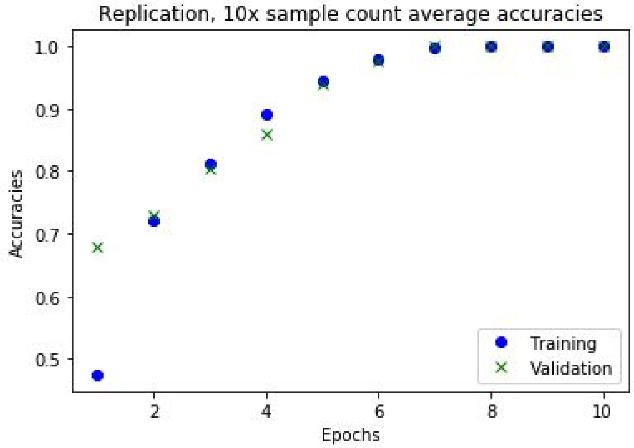
Training

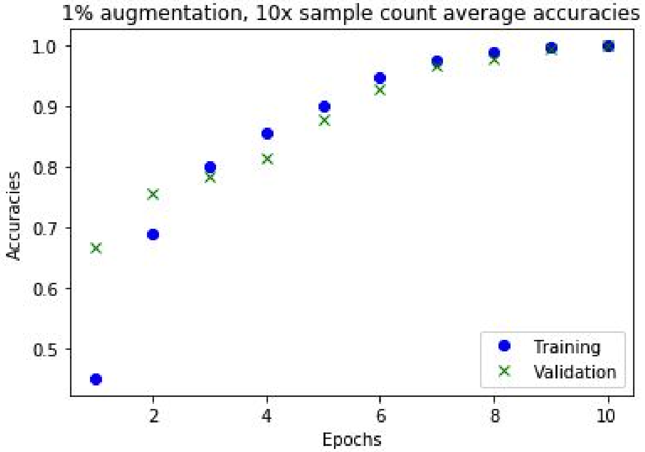
- Model was trained on 300 labelled samples
- Samples partitioned into training and validation sets
  - 70% (210) for training, 30% (90) for validation
- Training trials ran for 10 epochs
- Epochs processed 300 samples in 32-sample batches

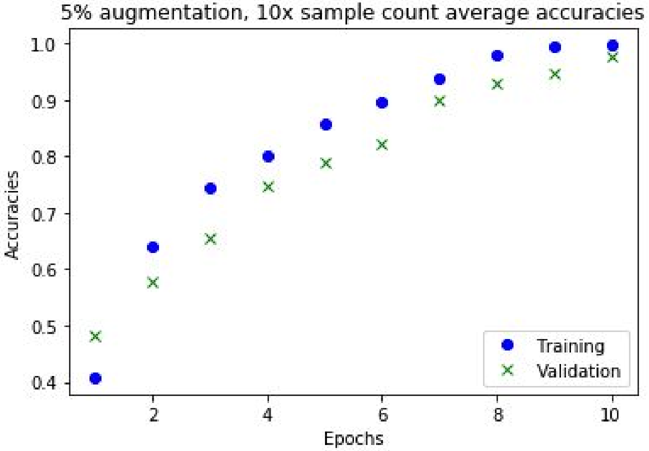
Training (cont.)

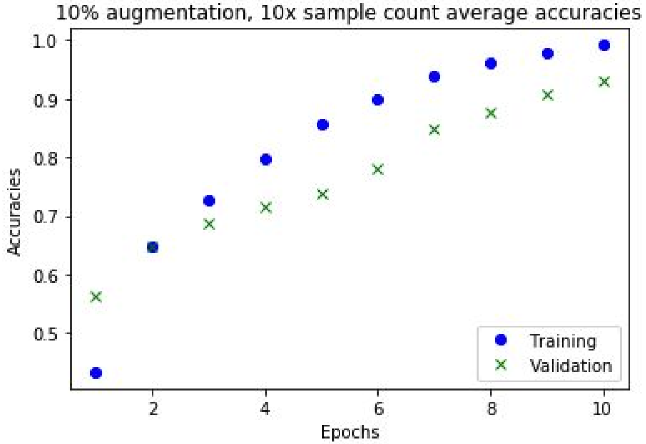
- Loss function was categorical cross entropy
- Optimizer was Adaptive Moment Estimation (Adam) with Nesterov Momentum:
  - Learning rate: 0.00015
  - Beta 1: 0.9
  - Beta 2: 0.999,
  - Epsilon: 1e-08,
  - Schedule Decay: 0.004
- 4 training set-ups, 3 trials each:
  - Replication, 10x sample count
  - 1% augmentation, 10x sample count
  - 5% augmentation, 10x sample count
  - 10% augmentation, 10x sample count

Results


Replication, 10x sample count average accuracies
Accuracies
Epochs
Training
Validation


1% augmentation, 10x sample count average accuracies
Accuracies
Epochs
Training
Validation


5% augmentation, 10x sample count average accuracies
Accuracies
Epochs
Training
Validation


10% augmentation, 10x sample count average accuracies
Accuracies
Epochs
Training
Validation

Recommendations

- Multiple changes can be implemented, which could improve the performance of the model
- Some potential changes:
  - Expand model to work with more classes: classify for finer phalangeal articulations
    - Requires more training data and more complex models
  - Use wider, deeper networks to improve performance
    - More computationally expensive and prone to overfitting
  - Use other CNN architectures:
    - ResNet, Inception
    - Apparently have higher accuracies and less expensive than VGG
  - Use other RNN architectures
    - Gated Recurrent Unit (GRU)
    - Unlike LSTMs, no memory unit, uses 2 gates (reset, update) instead of 3
    - On par performance, less complex, more efficient

References

[1] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko, “Long-term recurrent convolutional Networks for visual recognition and description,” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.
[2] “Convolutional Neural Networks (LeNet),” Convolutional Neural Networks (LeNet) — DeepLearning 0.1 documentation. [Online]. Available: http://deeplearning.net/tutorial/lenet.html. [Accessed: 22-Apr-2017].
[3] S. B. Damelin and W. Miller, The mathematics of signal processing. Cambridge: Cambridge University Press, 2012.
[4] A. Sachan, “Tensorflow Tutorial 2: image classifier using convolutional neural network,” CV-Tricks.com, 09-Apr-2017. [Online]. Available: http://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/. [Accessed: 22-Apr-2017].
[5] R Hahnloser, R. Sarpeshkar, M A Mahowald, R. J. Douglas, H.S. Seung (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature. 405. pp. 947–951.
[6] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” [1409.1556] Very Deep Convolutional Networks for Large-Scale Image Recognition, 10-Apr-2015. [Online]. Available: https://arxiv.org/abs/1409.1556. [Accessed: 22-Apr-2017].
[7] “LSTM Networks for Sentiment Analysis,” LSTM Networks for Sentiment Analysis — DeepLearning 0.1 documentation. [Online]. Available: http://deeplearning.net/tutorial/lstm.html. [Accessed: 22-Apr-2017].
[11] C. Olah, “Understanding LSTM Networks,” Understanding LSTM Networks -- colah's blog. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 24-Apr-2017].
[12] Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation. 9 (8): 1735–1780.
[13] J. Han and C. Moraga, “The influence of the sigmoid function parameters on the speed of backpropagation learning,” Lecture Notes in Computer Science From Natural to Artificial Neural Computation, pp. 195–201, 1995.
[14] “Multilayer Perceptron,” Multilayer Perceptron — DeepLearning 0.1 documentation. [Online]. Available: http://deeplearning.net/tutorial/mlp.html. [Accessed: 22-Apr-2017].
[15] Haykin, Simon (1998). Neural Networks: A Comprehensive Foundation (2 ed.). Prentice Hall.
[16] C. M. Bishop, Pattern recognition and machine learning. New Delhi: Springer, 2013.
[17] 2017. [Online]. Available: http://yann.lecun.com/exdb/publis/pdf/boser-92a.pdf. [Accessed: 02- May- 2017].
[18] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf. [Accessed: 22-Apr-2017]
[19] T. Dozat, “Incorporating Nesterov Momentum into Adam,” [Online]. Available: http://cs229.stanford.edu/proj2015/054_report.pdf. [Accessed: 02-May-2017]