Quinn Reserach Group Fall '18 Presentation

Christian McDaniel

Developing a Start-to-Finish Pipeline for Accelerometer-Based Activity Recognition Using Long Short-Term Memory Recurrent Neural Networks

Christian McDaniel and Shannon Quinn

SciPy 2018 Conference Proceedings

Presentation Roadmap:

1. Background - Title Breakdown
   A. Accelerometer-Based Activity Recognition
      - What
      - Why
      - How
   B. Using Long Short-Term Memory Recurrent Neural Networks

Developing a Start-to-Finish Pipeline for Accelerometer-Based Activity Recognition Using Long Short-Term Memory Recurrent Neural Networks

Christian McDaniel and Shannon Quinn

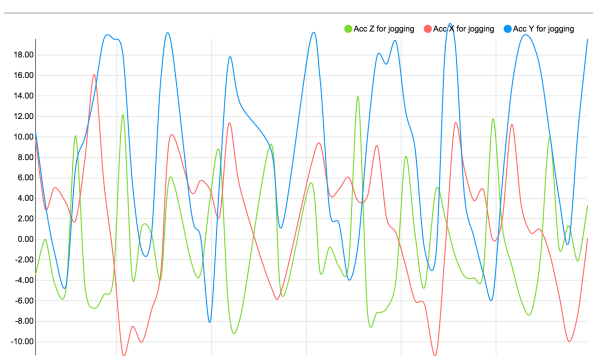
SciPy 2018 Conference Proceedings

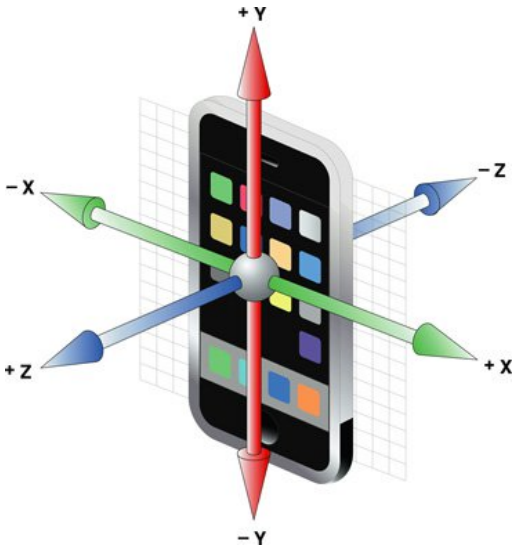
That's a loot of words... lets break it down

"Accelerometer-Based Activity Recognition"

"Accelerometer-Based Activity Recognition"

1) The "What"

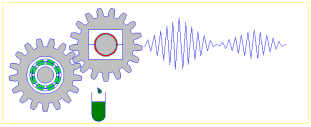




"Accelerometer-Based Activity Recognition"

2) The "Why"

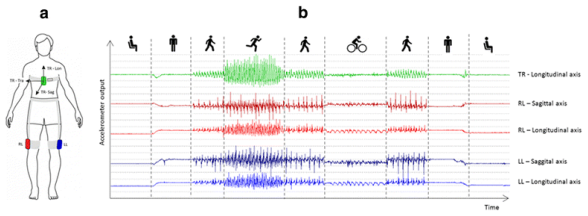
From Human Health (HAR)....

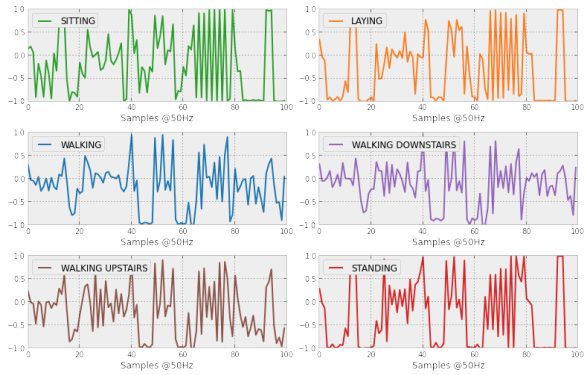


"Accelerometer-Based Activity Recognition"

2) The "Why"

Activities of Daily Life (ADL):

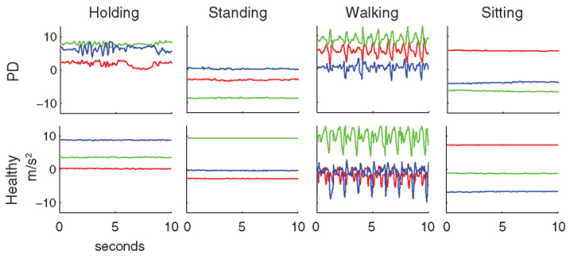




"Accelerometer-Based Activity Recognition"

2) The "Why"

Clinical Diagnosis:

Parkinson's Disease (Motor SXs)	Nighttime Scratching
	

"Accelerometer-Based Activity Recognition"

2) The "Why"

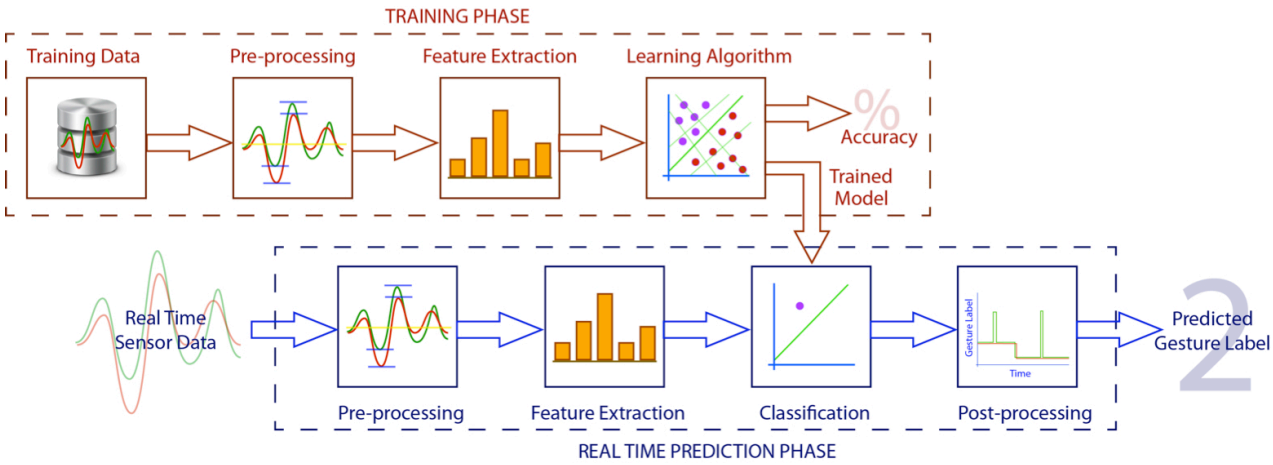
"Accelerometer-Based Activity Recognition"

3) The "How"

Key Points:

> Classical vs. AI

> Online/ On-Device vs. *post hoc* analysis



... which brings us to ...

"Using Long Short-Term Memory Recurrent Neural Networks"

"Using Long Short-Term Memory Recurrent Neural Networks"

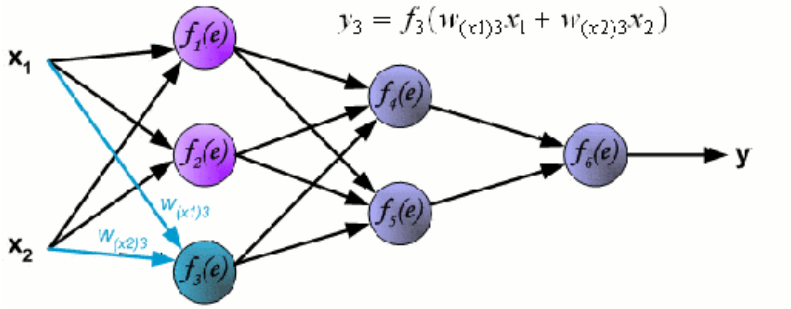
1) The "What" - Artificial Neural Networks

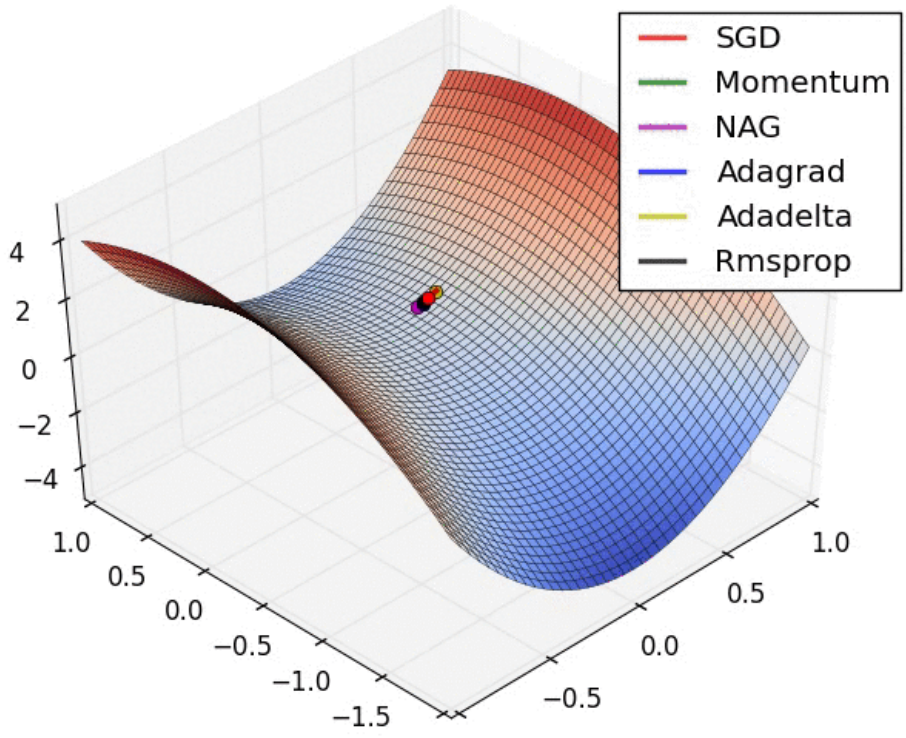
"Using Long Short-Term Memory Recurrent Neural Networks"

1) The "What" - Artificial Neural Networks

General Training Pipeline:

1. Conduct a forward pass,
   - compute $\tilde{y}$ and compare with y to generate the error term
2. Backpropagate the error regarding the correction needed for $\tilde{y}$
3. Backpropagate the correction to the hidden layer
4. update weight matrices A and B via dy and dz


$y_3 = f_3(w_{(x1)3}x_1 + w_{(x2)3}x_2)$
x1
x2
y
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
$W_{(x1)3}$
$W_{(x2)3}$


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

Bias - Variance Tradeoff

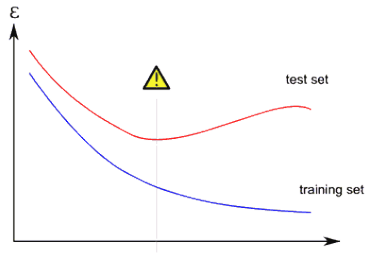
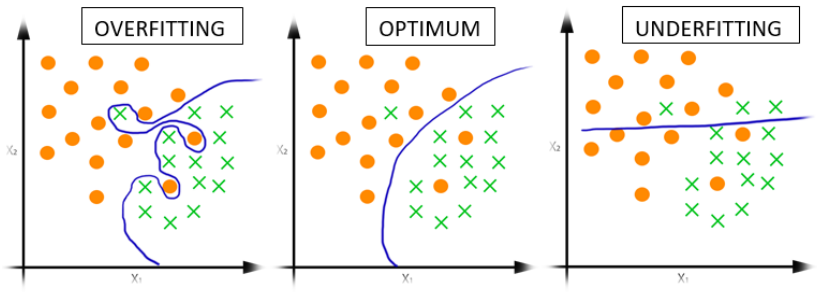
Bias - Variance Tradeoff

These networks are very flexible!

Bias - Variance Tradeoff

These networks are very flexible!

And very capable of overfitting the data

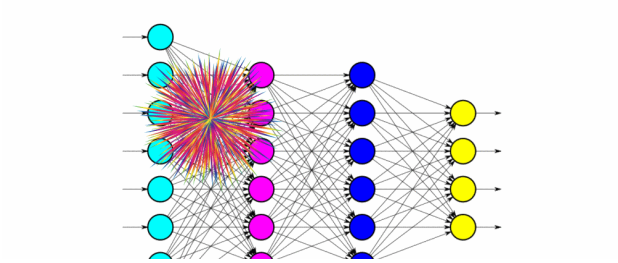
_	_
	

Bias - Variance Tradeoff

These networks are very flexible!

And very capable of overfitting the data

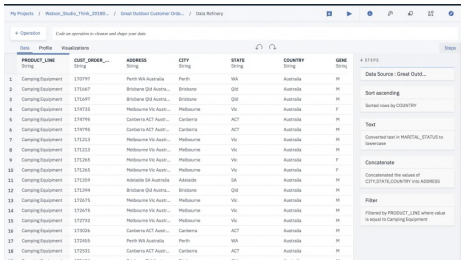
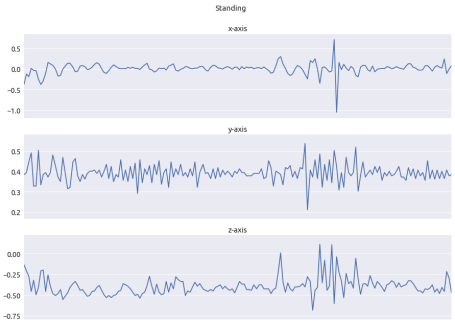
Not to mention, going deeper has led to issues (i.e., exploding and vanishing gradients)



Some methods to help with this:

- `Early Stopping | Weight Decay | Dropout | Learning Rate | Norm Regularization | Gradient Clipping | Batch Normalization`

And there's the issue of having *time series* data

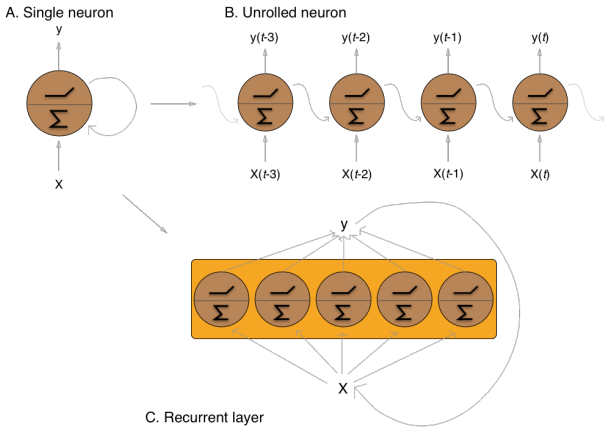
This data		This data
	... Is a bit different than ...	

"Using Long Short-Term Memory Recurrent Neural Networks"

1) The "What" - Recurrent Neural Networks

"Using Long Short-Term Memory Recurrent Neural Networks"

1) The "What" - Recurrent Neural Networks



Each neuron now has TWO sets of weights:

$$y(t) = \phi(W_x \cdot x(t) + W_y \cdot Y(t-1) + b)$$

Backpropagation is performed on the "unrolled" neuron,

i.e., "Back Propagation Through Time" (BPTT)

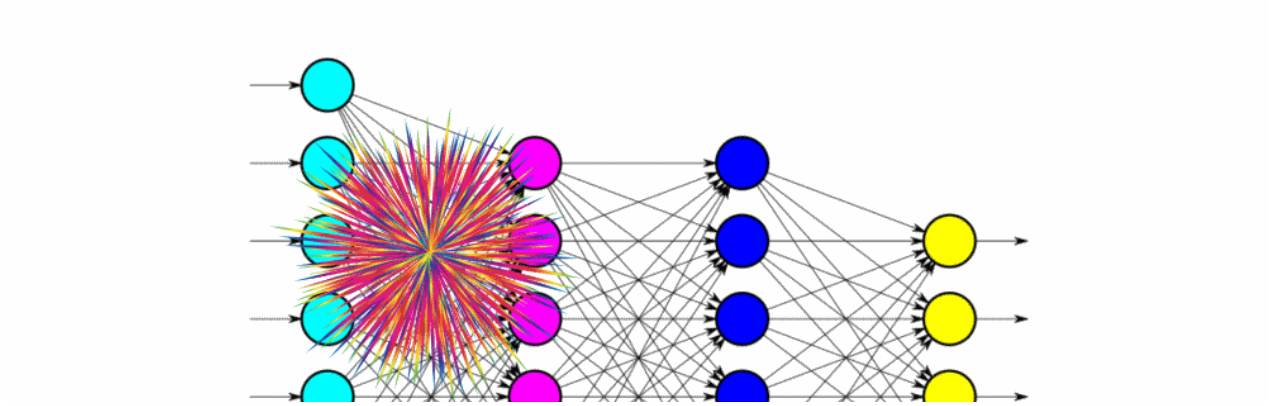
Backpropagation is performed on the "*unrolled*" neuron, i.e., "Back Propagation Through Time" (BPTT)

This causes the networks to become *very* deep *very* quickly

depth = n timesteps per input per layer

Re-enter: Exploding and Vanishing Gradients

- As the gradient is back-propagated, its influence on the weight matrix at a given layer decreases "through time"

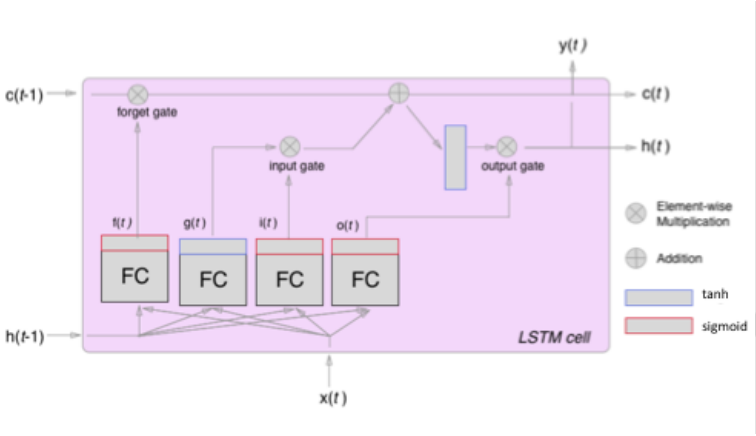


"Using Long Short-Term Memory Recurrent Neural Networks"

1) The "What" - The Long Short-Term Memory Cell

"Using Long Short-Term Memory Recurrent Neural Networks"

1) The "What" - The Long Short-Term Memory Cell



Inputs: `x(t)` = single time-step fed into 4 independent fully-connected layers
`h(t)` = short-term state `c(t)` = long-term state

Gates:

- each gate is essentially its own neuron / perceptron
  - input * weight, add a bias, activate!

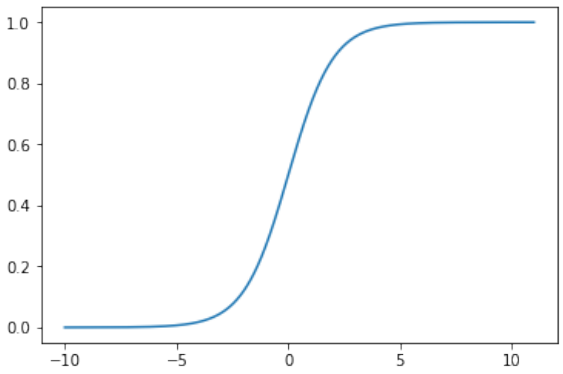
Why?

In [173]:
```
# let's look into the lstm cell using only numpy!
import numpy as np
# and matplotlib to see some of what we're doing :)
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

In [174]:
```
# first let's set up our activation functions
def sigmoid(x):
        return 1 / (1 + np.exp(-x))
```

In [175]:
```
# let's plot that to see what we've made
example = np.arange(-10,11, .01)
plt.plot(example, sigmoid(example))
```

Out[175]:
```
[<matplotlib.lines.Line2D at 0x1a25493828>]
```



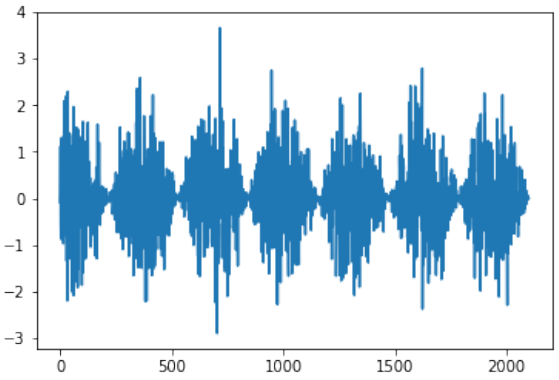
In [176]:
```
# now for tanh, another function in the exponential family with a bit different
output range
def g(x):
    return np.exp(x) / (1 + np.exp(x))

def tanh_1(x):
    return (2*g(2*x))-1

def tanh_2(x):
    return (np.exp(x)-np.exp(-x)) / (np.exp(x)+np.exp(-x))

def tanh_3(x):
    return np.tanh(x)
```

In [177]:
```
# and to plot:
plt.figure(figsize=(20,5))
plt.subplot(1,3,1)
plt.plot(example, tanh_1(example))
plt.subplot(1,3,2)
plt.plot(example, tanh_2(example))
plt.subplot(1,3,3)
plt.plot(example, tanh_3(example))
```

Out[177]:
```
[<matplotlib.lines.Line2D at 0x1a254ef668>]
```

In [193]:
```
# let's make some time series data!
data = np.cos(example)
data = np.array([t * np.random.normal() for t in data])
plt.plot(data)
data
```

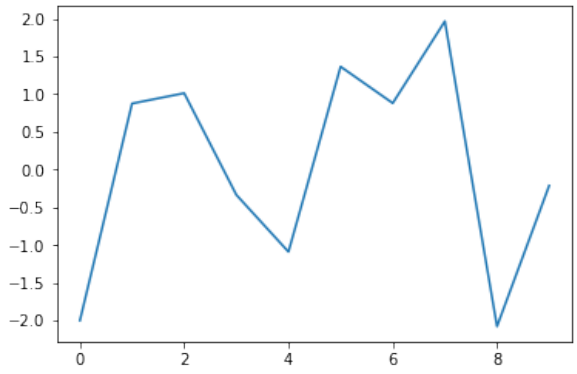
Out[193]:
```
array([-0.08557031,  0.19866981,  1.11648423, ..., -0.00834961,
       -0.00971766, -0.00373537])
```



In [194]:
```
num_classes = 6
window_size=10
```

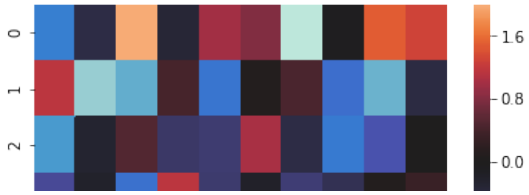
In [195]:
```
# ... and take a window of the data to represent a single input step
def window(data,win_size,n):
    start = 0
    win = (data[start+n:win_size+n])
    plt.plot(win)
    return win
```

In [196]:
```
window = window(data,window_size,53)
```



In [209]:
```
# let's initialize our weight matrix (i.e., a gate within the lstm cell)
gate = np.zeros((num_classes,len(window)))

for r in range(len(gate[:,0])):
    for c in range(len(gate[0,:])):
        gate[r,c] = gate[r,c]+np.random.normal()

sns.heatmap(gate, center=0.0)
```

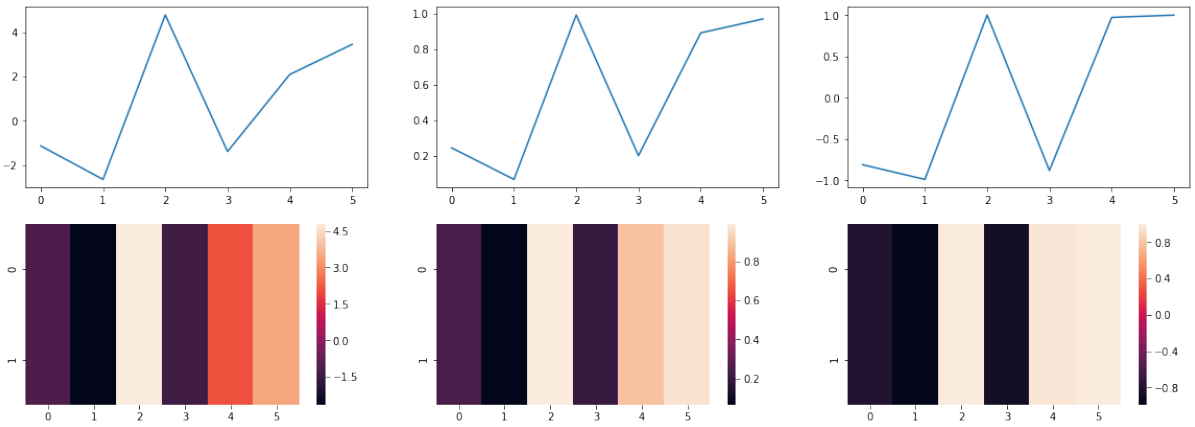
Out[209]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2752dd30>
```



In [211]:
```
# combine our input time step with the weight matrix
weighted_input = np.dot(gate, window)
# activate the matrix via sigmoid
sig_input = sigmoid(weighted_input)
# activate the matrix via tanh
tanh_input = tanh_3(weighted_input)
```

In [212]:
```
# plot for comparison
plt.figure(figsize=(20,7))
plt.subplot(2,3,1)
plt.plot(weighted_input)
plt.subplot(2,3,4)
sns.heatmap((weighted_input,weighted_input))
plt.subplot(2,3,2)
plt.plot(sig_input)
plt.subplot(2,3,5)
sns.heatmap((sig_input,sig_input))
plt.subplot(2,3,3)
plt.plot(tanh_input)
plt.subplot(2,3,6)
sns.heatmap((tanh_input,tanh_input))
```

Out[212]: <matplotlib.axes._subplots.AxesSubplot at 0x1a28392668>



_	_
	Four gates (each with its own pair of sets of weights):¶ `i(t)` = σ(W)xi•x(t) +Whi •h(t−1) +bi

Learn to code an LSTM cell from scratch:

See Siraj Rival's LSTM YouTube video and the associated links (https://www.youtube.com/watch?v=9zhrxE5PQgY)!

And check out LSTMVis (http://lstm.seas.harvard.edu) from harvardnlp and Harvard's Visual Computing Group,

As well as Andrej Karpathy's LSTM blog post (http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

"Using Long Short-Term Memory Recurrent Neural Networks"

1) The "What"

2) The "Why"

"Using Long Short-Term Memory Recurrent Neural Networks"

2) The "Why"

	Classical methods	**DL/AI**
Pre-processing	+ Smooth; Remove noise, gravity component, etc. + Use many sensors and channels (e.g., gyroscope, magnetometer)	+ Keep to a minimum + Standardize data
Feature Extraction	+ Generate many hand-crafted features (e.g., 551 for the HAR dataset)	+ Let the network do this for you
Learning Algorithm	+ Many to chose from + Depedent on parameter choice	+ Optimize hyperparams + Parameters are learned from the data

"Using Long Short-Term Memory Recurrent Neural Networks"

2) The "Why"

Both Classical and NN/AI methods have been shown to perform well,

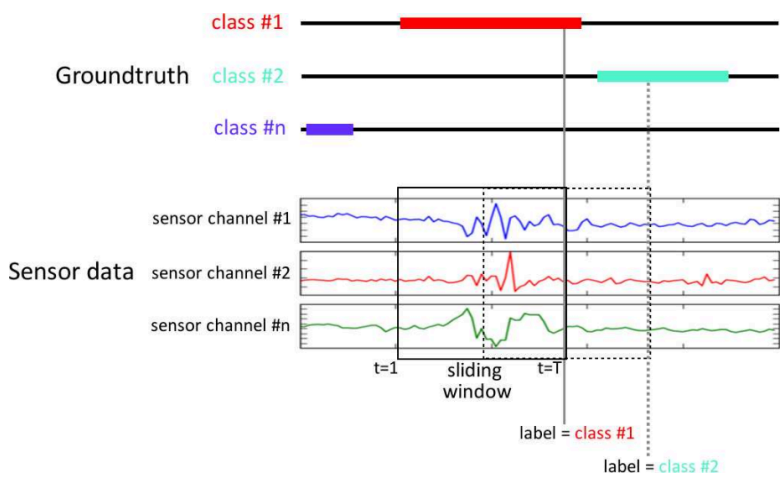
But, given the trending goal of online and on-device classification, can we agree that NNs deserve some attention?!

"Using Long Short-Term Memory Recurrent Neural Networks"

2) The "What"

"Using Long Short-Term Memory Recurrent Neural Networks"

3) The "How" - Data Processing

–	–
+ Sliding window + Batch size +Step size	

"Using Long Short-Term Memory Recurrent Neural Networks"

3) The "How" - Hyperparameters, Architectures

Category	Hyperparameter
Architecture	Units Layers
Forward processing	Activation Function Bias Weight Init
Regularization	Gradient clipping Dropout Batch Normalization

... And now to put it all together ...

"Developing a Start-to-Finish Pipeline"

(for the aforementioned data and models)

"Developing a Start-to-Finish Pipeline"

(for the aforementioned data and models)

GOAL 1: UNIFY previous works and STANDARDIZE future work

GOAL 2: Provide POC for optimized Baseline LSTM on *only*

Breakdown of Related Works:

Over 30 studies analyzed

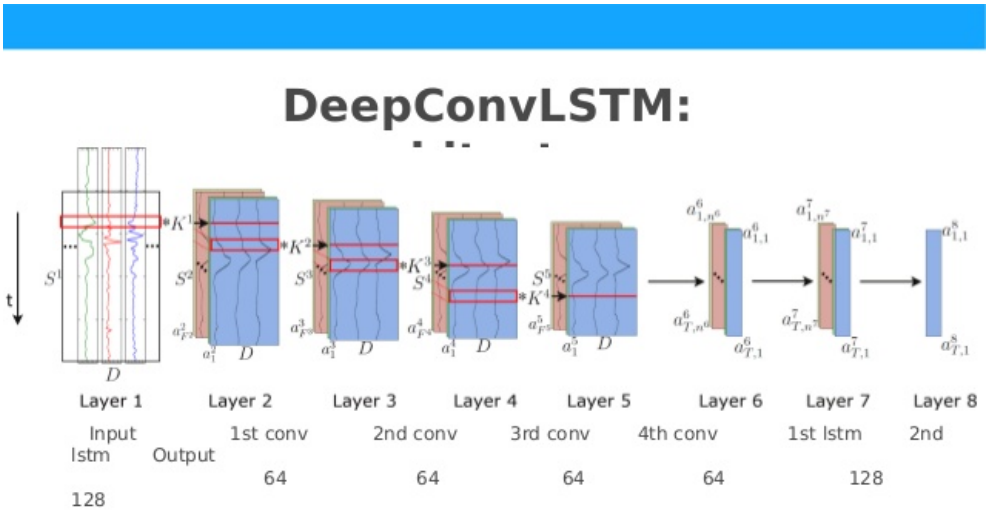
Breakdown of Related Works:

Each used a **unique implementation** of LSTMs for HAR using Accelerometer data

Breakdown of Related Works:

Breakdown of Related Works:

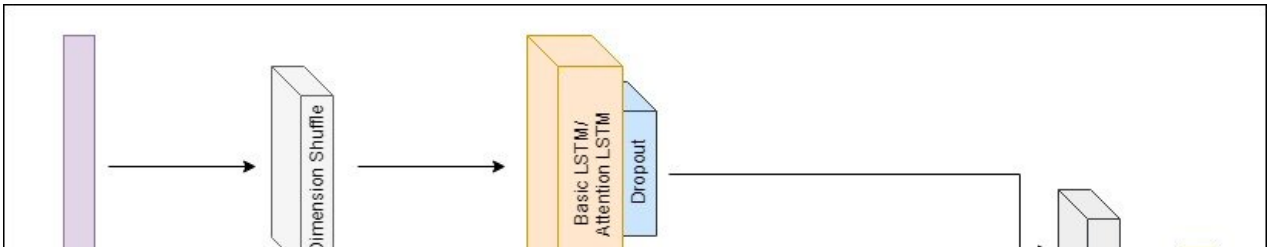
Each used a unique implementation of LSTMs for HAR using Accelerometer data



Breakdown of Related Works:

Each used a unique implementation of LSTMs for HAR using Accelerometer data

Multivariate Fully Convolutional LSTM



Breakdown of Related Works:

Each used a unique implementation of LSTMs for HAR using Accelerometer data

BLSTM

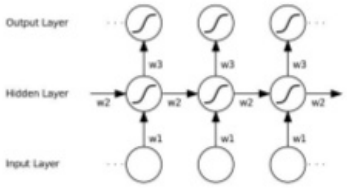


Figure 3.4: **An unfolded recurrent network.** Each node represents a layer of network units at a single timestep. The weighted connections from the input layer to hidden layer are labelled 'w1', those from the hidden layer to itself (i.e. the recurrent weights) are labelled 'w2' and the hidden to output weights are labelled 'w3'. Note that the same weights are reused at every timestep. Bias weights are omitted for clarity.

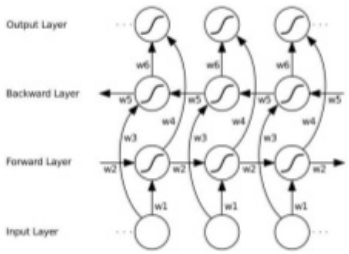
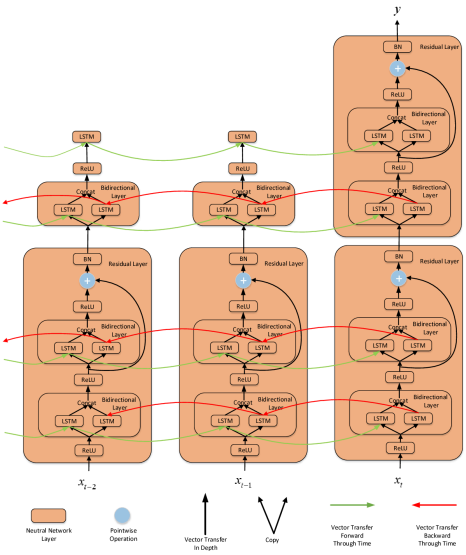


Figure 3.5: **An unfolded bidirectional network.** Six distinct sets of weights are reused at every timestep, corresponding to the input-to-hidden, hidden-to-hidden and hidden-to-output connections of the two hidden layers. Note that no information flows between the forward and backward hidden layers; this ensures that the unfolded graph is acyclic.

Breakdown of Related Works:

Each used a unique implementation of LSTMs for HAR using Accelerometer data

Deep Residual Bidirectional LSTMs



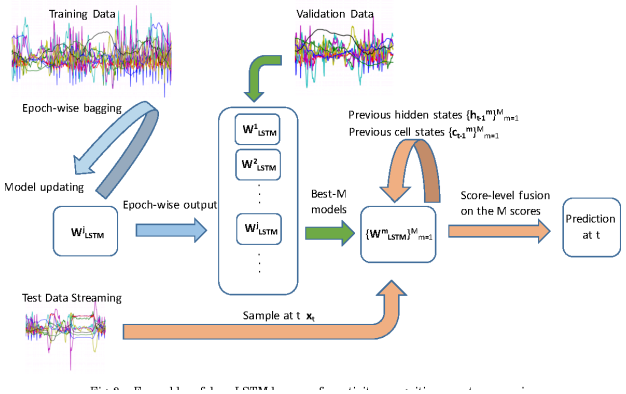
Breakdown of Related Works:

Each used a unique implementation of LSTMs for HAR using Accelerometer data

Breakdown of Related Works:

Each used a unique implementation of LSTMs for HAR using Accelerometer data

Ensemble LSTMs



Breakdown of Related Works:

Each used a unique implementation of LSTMs for HAR using Accelerometer data

Meanwhile, some are still asking, do LSTMs even make sense to use?

Breakdown of Related Works:

Research into the *functioning* of LSTMs for classifying accelerometer data is very sparse

This sort of research for LSTMs has mainly been done for text processing

Cell sensitive to position in line:

```
The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.
```

Cell that turns on inside quotes:

```
"You mean to imply that I have nothing to eat out of..... On the
contrary, I can supply you with everything even if you want to give
dinner parties," warmly replied Chichagov, who tried by every word he
spoke to prove his own rectitude and therefore imagined Kutuzov to be
animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating
smile: "I meant merely to say what I said."
```

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

A large portion of cells are not easily interpretable. Here is a typical example:

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX
   * defines the longest valid length.
   */
```

Breakdown of Related Works:

Within the LSTM layers, the hyperparameter settings were all across the board

- Many provided little to no justification for there choice
- Those who did test a range of hyperparameters used small ranges that did

Breakdown of Related Works:

Within the LSTM layers, the hyperparameter settings were all across the board

- Meanwhile, one study used fANOVA to find that certain hyperparameter settings *do* affect the outcomes
- Many of the settings matched those optimized for text processing, but might they be different for accelerometer-based HAR?

Breakdown of Related Works:

Within the LSTM layers, the hyperparameter settings were all across the board

Hyperparam	Range found in Lit
n Layers	1, 2, 3, 4
units per layer	3 - 512
recurrent activation fnxn	sigmoid*
cell output activation fnxn	tanh* relu sigmoid softmax linear
weight init	random orthogonal, fixed random seed, Glorot uniform, random uniform on [-1, 1], random normal distribution
mini batch sizes	32 - 450
loss fnxn	categorical cross entropy, F1 score loss, MSE, MAE
optimization	RMSprop, Adam, Adagrad, Nadam, Adadelta
learning rate	1e−7,1e−6,1e−5,1e−4,1e−3,1e−2,1e−1
regularization schemes	weight decay, dropout, momentum, gradient clipping, batch normalization
epochs	10-10000 with early stopping
sliding winow size	32 - 5000 ms

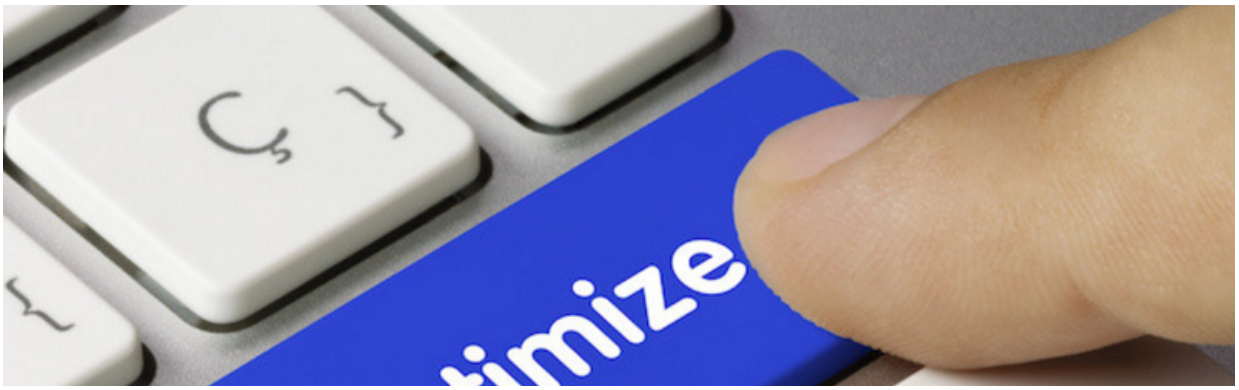
Breakdown of Related Works:

Within the LSTM layers, the hyperparameter settings were all across the board

Hyperparam	Range found in Lit
n Layers	1, 2, 3, 4
units per layer	3 - 512
recurrent activation fnxn	sigmoid*
cell output activation fnxn	tanh* relu sigmoid softmax linear
weight init	random orthogonal, fixed random seed, Glorot uniform, random uniform on [-1, 1], random normal distribution
mini batch sizes	32 - 450
loss fnxn	categorical cross entropy, F1 score loss, MSE, MAE
optimization	RMSprop, Adam, Adagrad, Nadam, Adadelta
learning rate	1e−7,1e−6,1e−5,1e−4,1e−3,1e−2,1e−1
regularization schemes	weight decay, dropout, momentum, gradient clipping, batch normalization
epochs	10-10000 with early stopping
sliding winow size	32 - 5000 ms

Even if we discretize each of these Hyperparameters into, e.g., 5 options per hyperparameter, that's still 5^11 = tens of millions of possible combinations

Unifying the Field with Bayesian Optimization



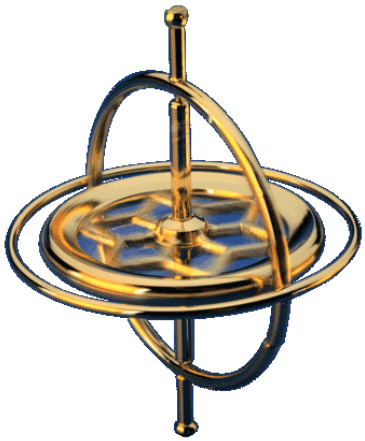
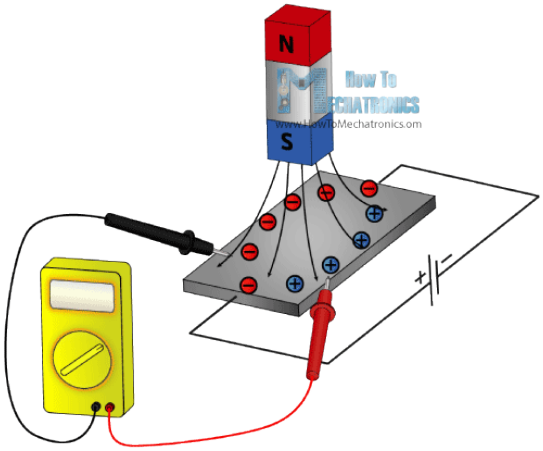
Breakdown of Related Works:

All kinds of preprocessing methods were used

- Nadaraya-Watson kernel weighted avg with Epanachnikov quadratic kernel and 40 nearest-neight window size
- remove gravity acceleration by applying median filter of length 2sec to each dimension and subratcting result
- construct movement borders (re hand orientation) via derivative of previously extracted gravity signals and calculating vector magnitude)

Breakdown of Related Works:

And many used additional data

Gyroscope	Magnetometer
	

But wouldn't it be neat if we could just use triaxial accelerometer data?

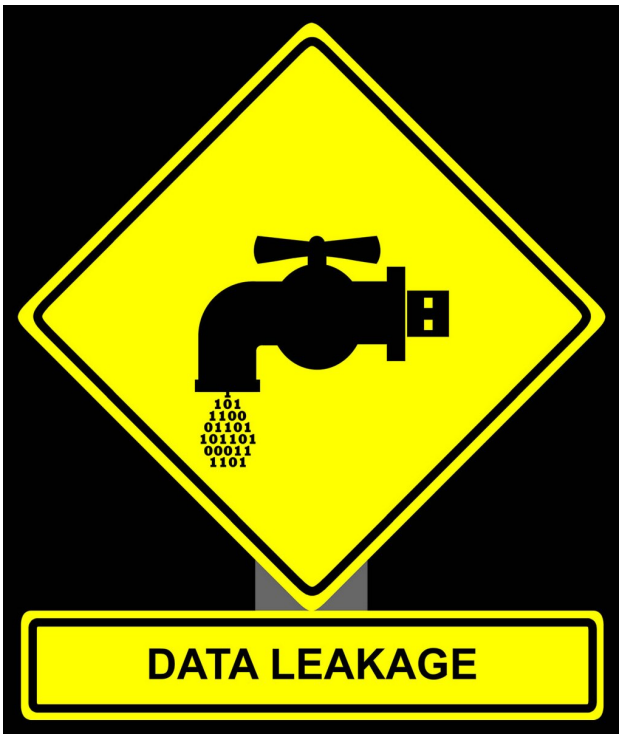
Less data to store, transfer, process...

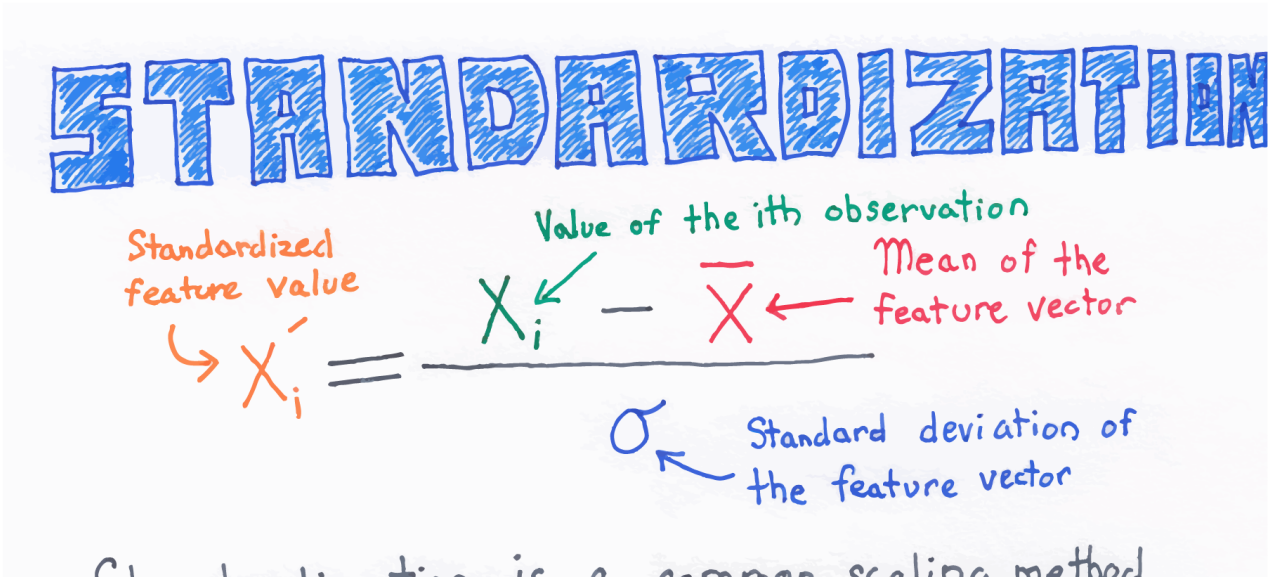
Puts the burden on the model !

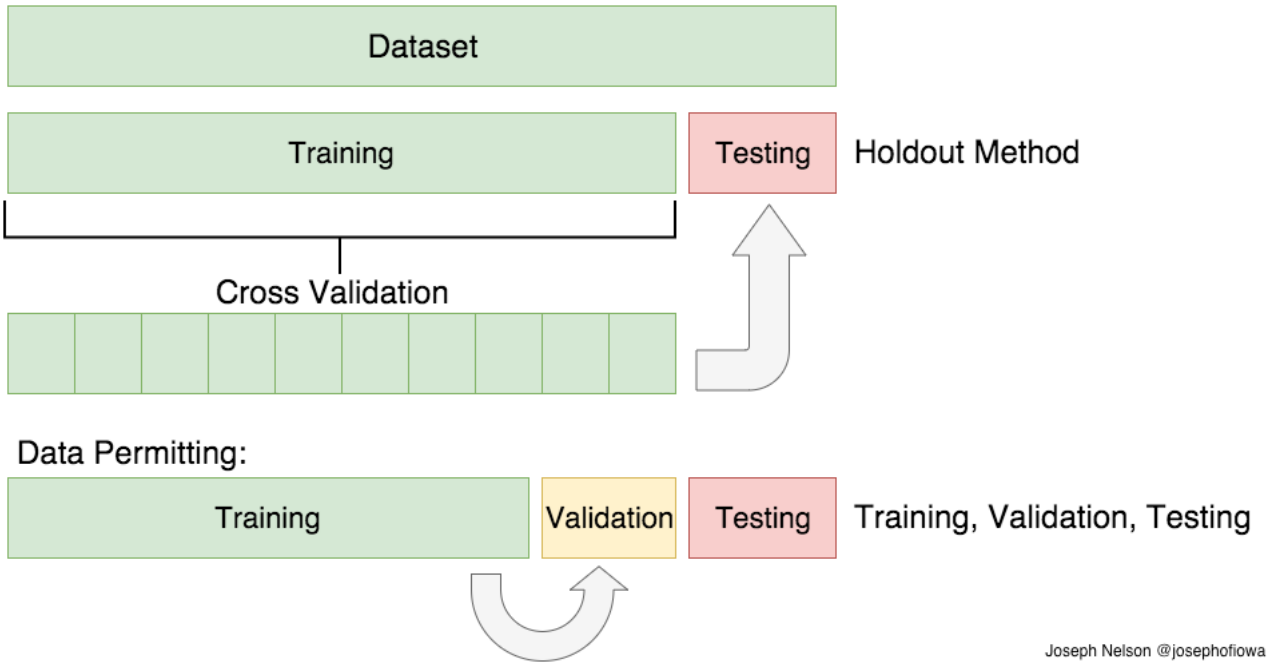
More on this later...

Breakdown of Related Works:

HAZARD: Data Leakage




STANDARDIZATION
Standardized feature value
Value of the ith observation
Mean of the feature vector
Standard deviation of the feature vector

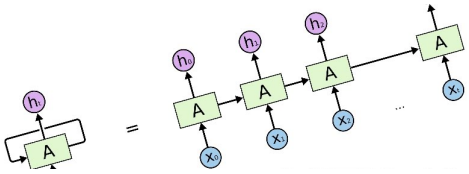

Dataset
Training
Testing
Holdout Method
Cross Validation
Data Permitting:
Training
Validation
Testing
Training, Validation, Testing
Joseph Nelson @josephofiowa

Breakdown of Related Works:

Results: The Strong vs. The Weak

CV, Repeated Experiments ... vs. ... Single Best Result

Methods



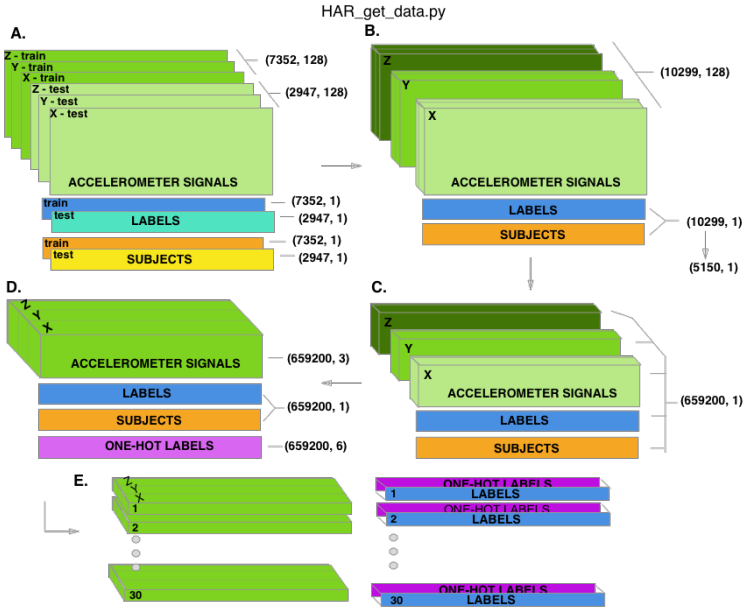
The data ...

- UCI HAR Dataset
  - Used only the triaxial accelerometer data
  - 30 volunteers
  - Each performed 6 motor activities
    - Walking, Upstairs, Downstairs, Sittinng, Standing, Layinng
    - Recorded in a single time series with ~650,000 time steps
  - Accelerometer: embedded accelerometer in iPhone, worn on hip
  - https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartp (https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smart

First off, let's fix this formatting ...

- The data is provided pre-windowed and pre-split into test/train sets

First off, let's fix this formatting ...

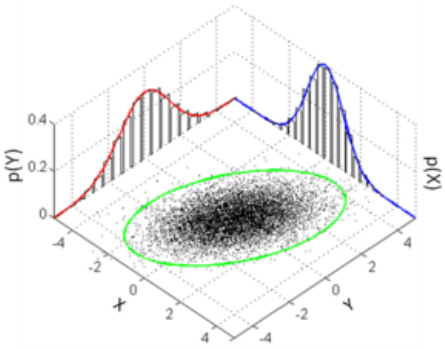


Now let's optimize this network

Now let's optimize this network

- With millions of possible hyperparameter configurations, grid search with packages like `GridSearchCV` from `scikit-learn` won't do ...
- We need some **heuristics** to help speed the search!

Modeling the search space (i.e., data science)



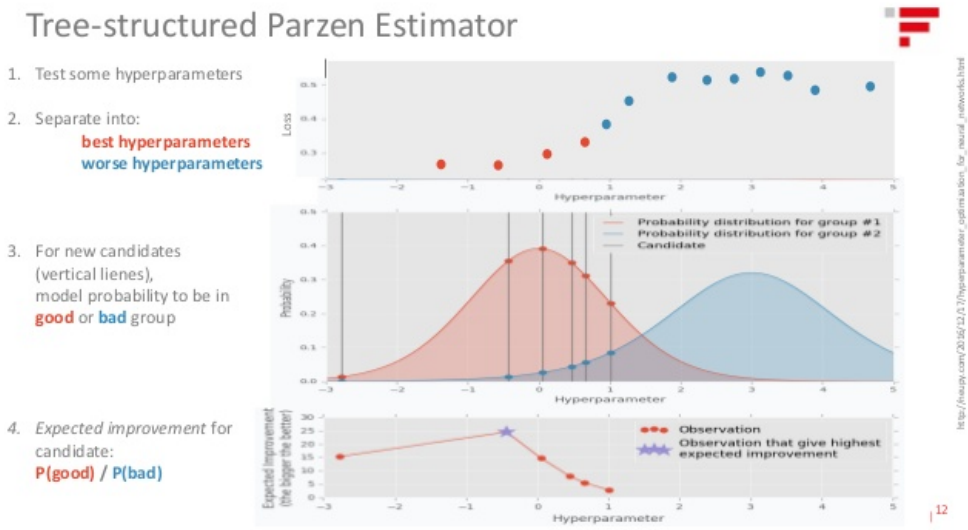
Now let's optimize this network

Tree-Structured Parzen Estimators, implemented via `Hyperas` from `Hyperopt`

An Expected Improvement algorithm

Now let's optimize this network

Tree-Structured Parzen Estimators



Tree-Structured Parzen Estimators

- Builds a probability model of the objective (loss) function:

$$P(\,score \mid hyperparameters)$$

+ so as to approximate

$$x^\star = \arg\min f(x)$$

Now let's optimize this network

Tree-Structured Parzen Estimators, implemented via `Hyperas` from `Hyperopt`

Basic Components of Sequential Model-Based Optimization (SMBO)

- **domain** of hyperparameters (search space)
  - probability distributions of each hyperparameter
  - start as uniform, log uniform, Gaussian, etc.
- **objective function** to minimize (loss function)
- **surrogate model** of the objective function
  - response surface
  - TPE: Baye's Theorem!
- **criteria for selecting** hyperparameters (selection function)

$$\mathrm{EI}_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y) p(y|x) dy$$

- **history** of previous trials

Now let's optimize this network

Tree-Structured Parzen Estimators, implemented via `Hyperas` from `Hyperopt`

An Expected Improvement algorithm

TPE:

- group the hyperparameters into **tree-like structure** to represent commonality (category)
  - contruct probability distribution for each
    - e.g., one distr. over each layer
      - individual hyperparams within each layer
- TPE uses a **threshold** "best" performance
- and constructs **two different probability distributions** for a given hyperparameter combination:
  - one that is worse than this threshold
  - one that is better

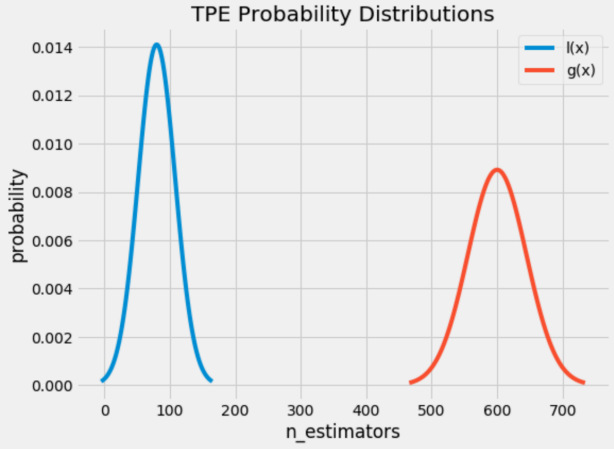
Tree-Structured Parzen Estimators, implemented via `Hyperas` from `Hyperopt`

-	-

Now let's optimize this network

Tree-Structured Parzen Estimators, implemented via `Hyperas` from `Hyperopt`

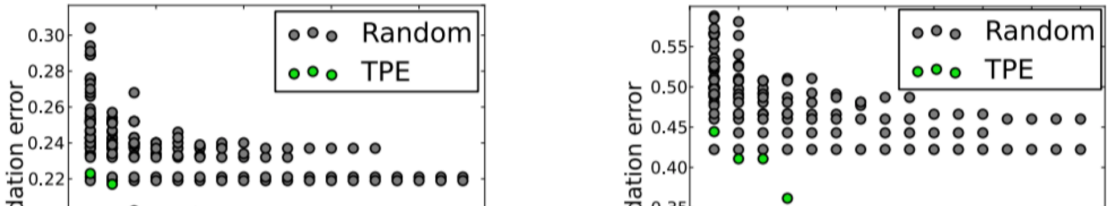
An Expected Improvement algorithm



Now let's optimize this network

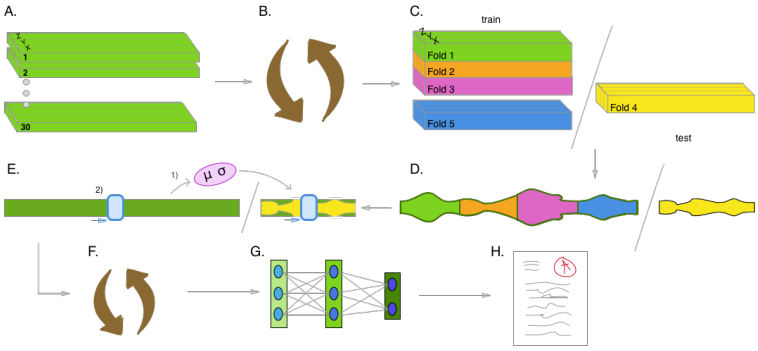
Tree-Structured Parzen Estimators, implemented via `Hyperas` from `Hyperopt`

An Expected Improvement algorithm



Now that we've prepared our data and optimized our network, let's throw it all into a reproducible Pipeline

Now that we've prepared our data and optimized our network, let's throw it all into a reproducible Pipeline



A Note on Performance Measures: Don't be fooled by the Accuracy Paradox!

- Multiclass predictions:
  - Each class has its TP, TN, FP, FN
- Micro- vs. Macro- performance measures
  - Macro- computes the PM for each class and averages

Results:

Model	Performance	Features
Baseline LSTM 1	90.77%	9 (T,B,G)
Baseline LSTM 2	85.35%	3-9 (?)
Pipeline P1 (Best)	**93.47%**	**3**
Pipeline P2 (CV)	**90.97%**	**3**
	0.90968	**3**
Pipeline P2 (Best)	**95.25%**	**3**
	0.9572	**3**

Key Takeaways:

- LSTM's have great potential for online on-device Human Activity Recognition from raw accelerometer data alone
- We still have a lot to learn about LSTM cells' specific functioning when processing accelerometer data
- As such, case-wise model optimization is important!
- Additionally, we should be reporting the parameters we use, the ranges we test over, our reasoning and justifications thereof
- CV yields robust results; micro-F1 Score is more robust than Accuracy
- Pipelines allow reproducibility

The code: https://github.com/xtianmcd/accelstm (https://github.com/xtianmcd/accelstm)