Neural Networks in the Wild

Handwriting Recognition

By John Liu

Motivation

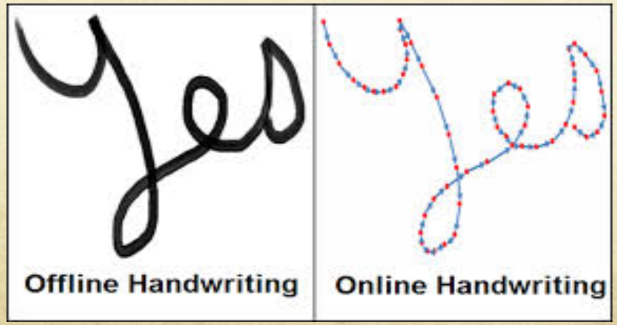
- US Post Office (700 million pieces of mail per day)
  - HWAI (Lockheed-Martin 1997)
  - Letter Recognition Improvement Program
- Electronic Health Records (*Medical scribble*)
- Banking: check processing
- Legal: signature verification
- Education: autograding

OCR is not HWR

Optical Character Recognition	Handwriting Recognition
Fixed fonts	Free flowing
No character overlap	Overlapping characters
Easy alignment id	Flowing alignment
Fixed aspect ratios	Variable aspect ratios
Low noise	Can be noisy

HWR: Online vs Offline

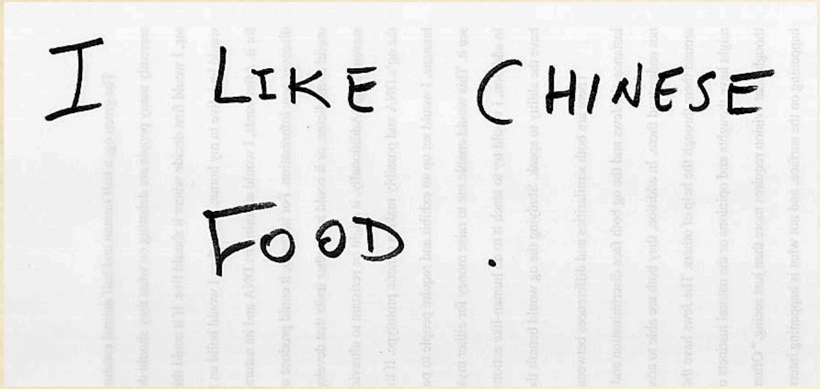
- Online recognition = conversion of text as it is written
  - Palm PDA, Google Handwrite
  - Low noise = easier classification
  - Features: stroke pressure, velocity, trajectory
- Offline recognition = image conversion of text
  - Computer-vision based methods to extract glyphs
  - Much more difficult than normal OCR



Offline HWR

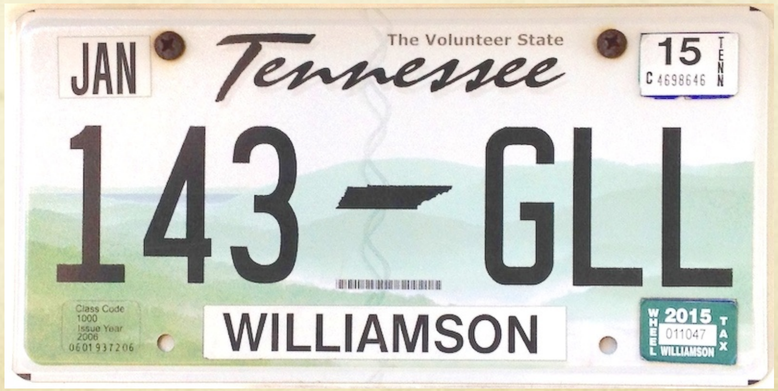
- Preprocessing
  - Discard irrelevant artifacts and remove noise
  - Smoothing, thresholding, morphological filtering
- Segmentation and Extraction
  - Find contours and bounding boxes
  - Glyph extraction
- Classification
  - Linear and ensemble methods
  - Neural Networks

HWR Examples


I LIKE CHINESE FOOD.

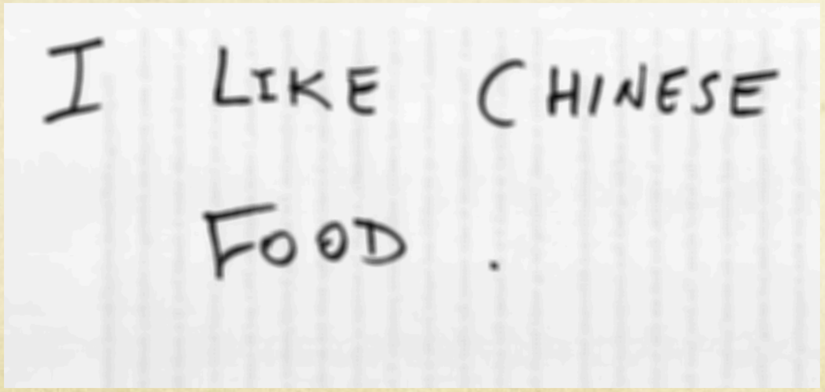
Scanned handwritten note
- noisy background
- varying character size
- biased ground truth

License Plate image
- goal: plate ID
- similar problem to HWR
- registration renewal soon



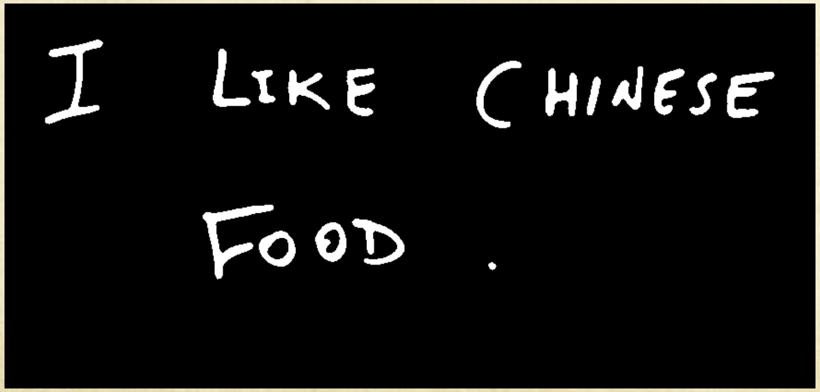
Kernel Smoothing

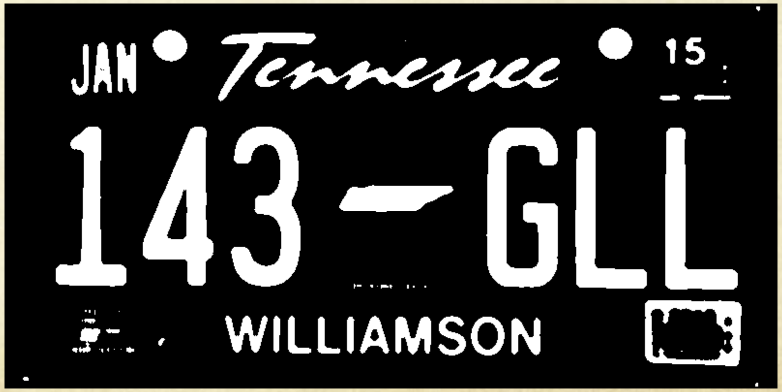
- Blurring helps eliminate noise after thresholding
  - Local kernel defines averaging area
  - Used 8x8 kernel for example images
  - OpenCV: cv2.blur()



Thresholding

- Thresholding converts to b/w image
  - Colors are inverted so (black,white) = (0,1)
  - Some noise is present, but greatly reduced due to smoothing in previous step
  - OpenCV: cv2.threshold()



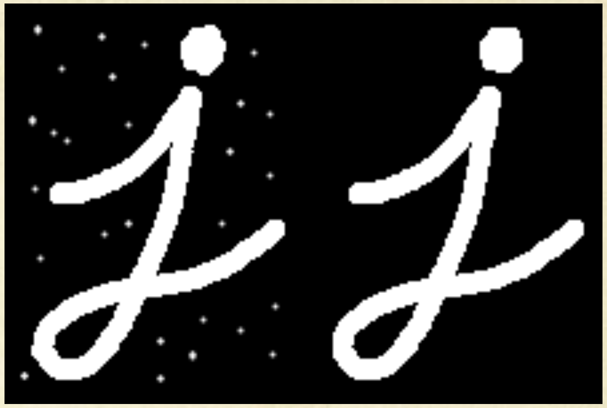


Morphological Filtering

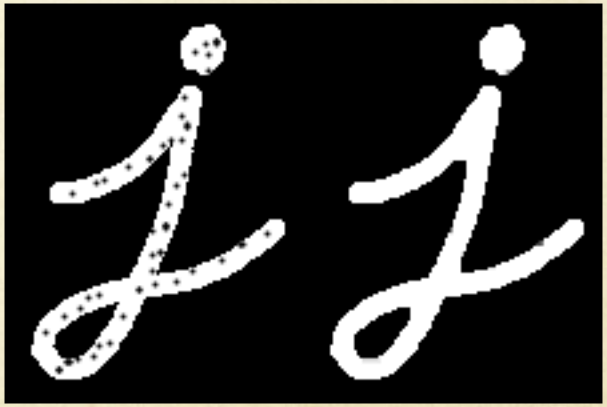
- Erosion: eat away at the boundaries of objects
  - Removes white noise and small artifacts
  - OpenCV: cv2.erode()   with 4x4 kernel
- Dilation: increases thickness and white region
  - opposite of erosion
  - Useful in joining broken parts of an object
  - OpenCV: cv2.dilate()   with 2x2 kernel

Opening vs Closing

- Erosion/Dilation = Opening
  - Eliminates noise outside objects

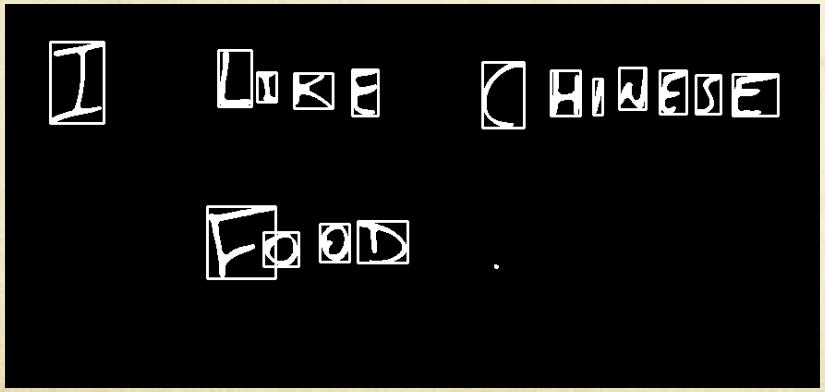


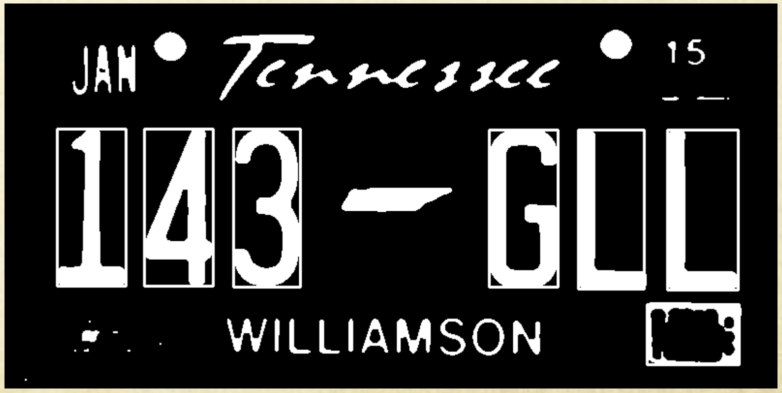
- Dilation/Erosion = Closing
  - Eliminates noise within objects



Contour/Bounding Box

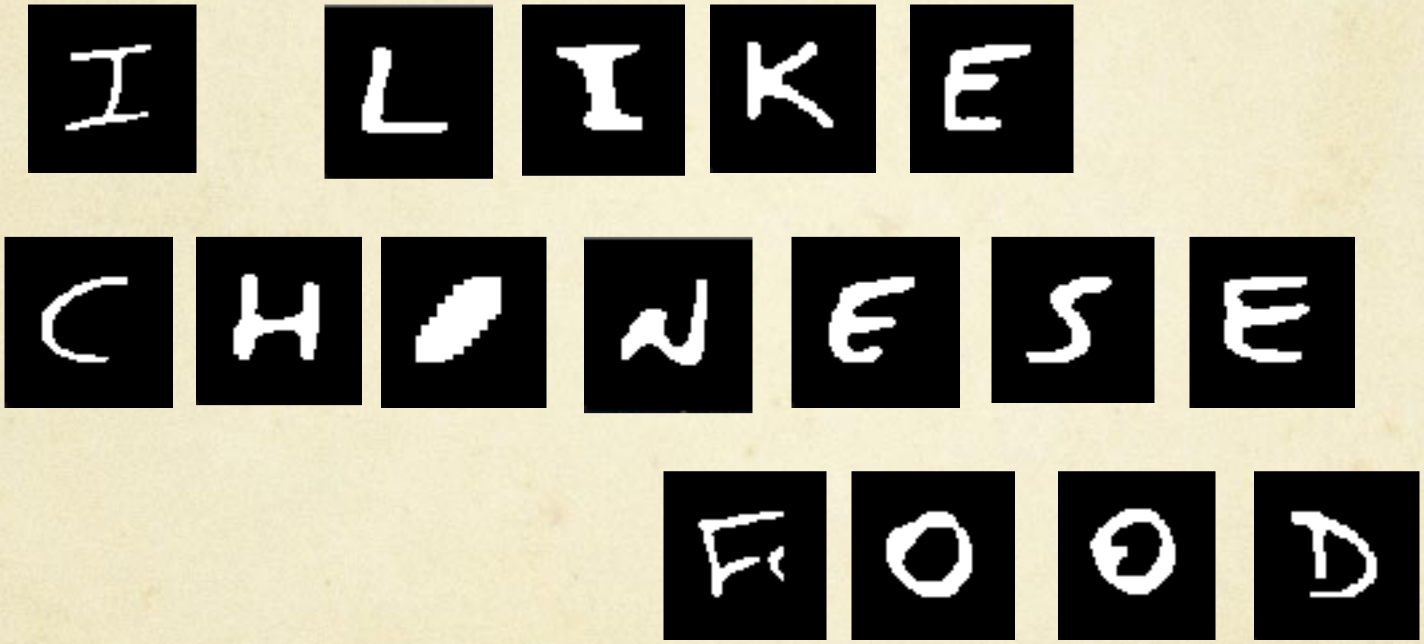
- Contours = set of all outer contiguous points
  - Approximate contours as a reduced polygon
  - Calculate the bounding rectangle
  - OpenCV: cv2.findContours(), cv2.approxPolyDP(), cv2.boundingRect()





Glyph Extraction

- Results from bounding boxes



NIST Special DB 19

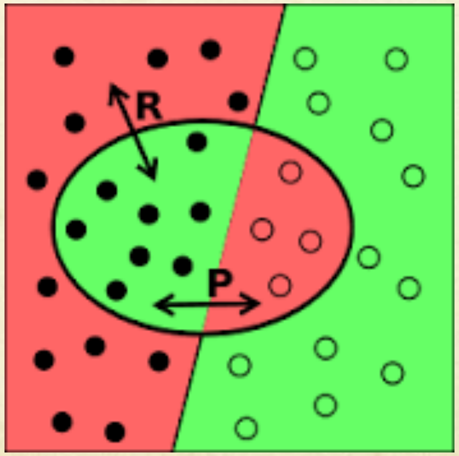
- Contains 814,255 segmented handwritten characters
- Superset of MNIST that includes alphabetic characters
- 62 character classes [A-Z], [a-z], [0-9], 128x128 pixels
- We down-sample to 32x32 and use only a subsample of 90,000 characters (train=70,000, test&valid=10,000)

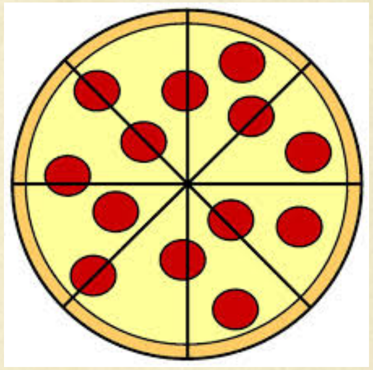
MNIST vs SD19

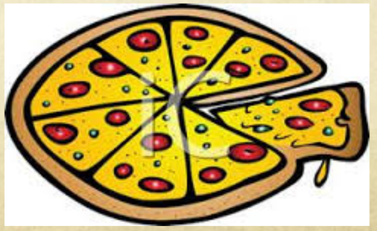
MNIST (LeCun)	SD19
10 classes	62 classes
Digits	Upper & Lower case + Digits
28x28 pixel	128x128 pixel
60,000 samples	814,255 samples
boring	GROOVY

Classification

- Goal: correctly classify character with highest:
  - Accuracy
  - F1 = geometric mean of precision & recall
- Typical Methods
  - Linear
  - SVM
  - Ensemble
  - Neural Networks

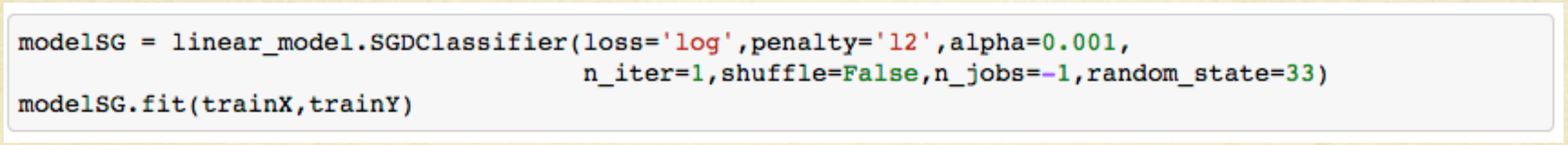






Logistic Regression

- a.k.a. Softmax, MaxEnt, logit regression
- We use multinomial LR with classes = 62
- Implemented with scikit-learn

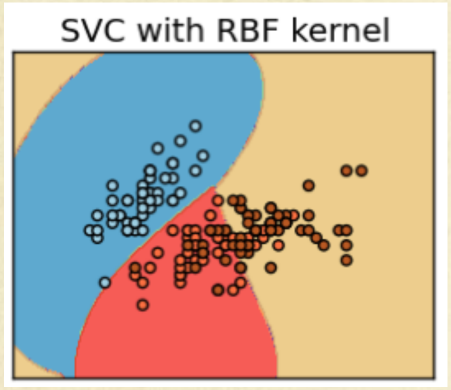


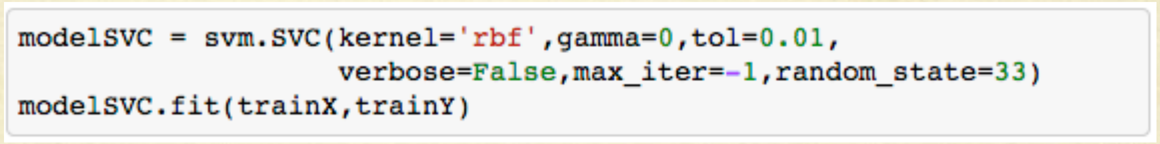
```
modelSG = linear_model.SGDClassifier(loss='log',penalty='l2',alpha=0.001,
                                     n_iter=1,shuffle=False,n_jobs=-1,random_state=33)
modelSG.fit(trainX,trainY)
```

Accuracy score = 59%, avg F1 score = 0.56 (baseline)

SVM - RBF

- We use a Gaussian kernel (a.k.a. Radial Basis Function)
- Implemented with scikit-learn (slow!)



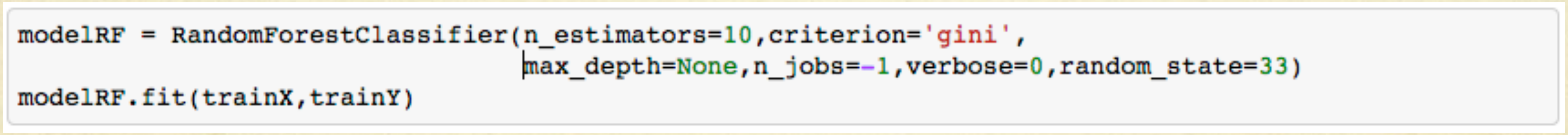


```
modelSVC = svm.SVC(kernel='rbf',gamma=0,tol=0.01,
                   verbose=False,max_iter=-1,random_state=33)
modelSVC.fit(trainX,trainY)
```

Accuracy score = 65%, avg F1 score = 0.61

Random Forest

- Ensemble estimator that builds a random forest of decision trees and combines their estimations
- Crowd Intelligence: "the wisdom of Crowds"
- Implemented with scikit-learn (fast!)

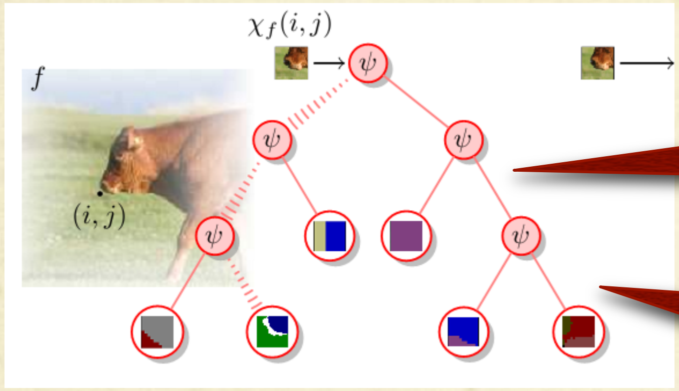


```
modelRF = RandomForestClassifier(n_estimators=10,criterion='gini',
                                 max_depth=None,n_jobs=-1,verbose=0,random_state=33)
modelRF.fit(trainX,trainY)
```

Accuracy score = 69%, avg F1 score = 0.66

Extra Trees

- a.k.a. Extremely Randomized Trees, similar to Random Forest except splits are also randomized



RF picks best from random subset of features

Extra Trees picks randomly from random subset of features

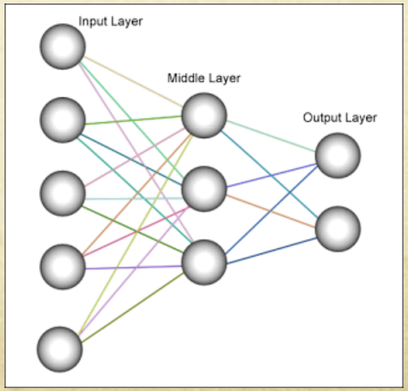
```
modelET = ExtraTreesClassifier(n_estimators=10,
                               max_depth=None,random_state=33)
modelET.fit(trainX,trainY)
```

Accuracy score = 73%, avg F1 score = 0.71

Neural Networks

- HWR inherently a computer vision problem, apt for neural networks given recent advances
- Inputs: image reshaped as a binary vector
- Outputs: one-hot representation = 62 bit vector
- Question: How do you actually go about building a (deep) neural network model?

WARNING: don't try this in Excel



Theano

- Python library that implements Tensor objects that leverage GPUs for calculation
  - Plays well with numpy.ndarray datatypes
  - Transparent use of GPU (float32) = 140X faster
  - Dynamic C code generation
- Installation is not fun:
  - Update GPU driver
  - Install CUDA 6.5 + Toolkit
  - Install CUDAMat

Pylearn2

- Python machine learning library toolbox built upon Theano (GPU speed)
- Provides flexibility to build customized neural network models with full access to hyper-params (nodes per layer, learning rate, etc...)
- Uses pickle for file I/O
- Two methods to implement models:
    - YAML
    - ipython notebook

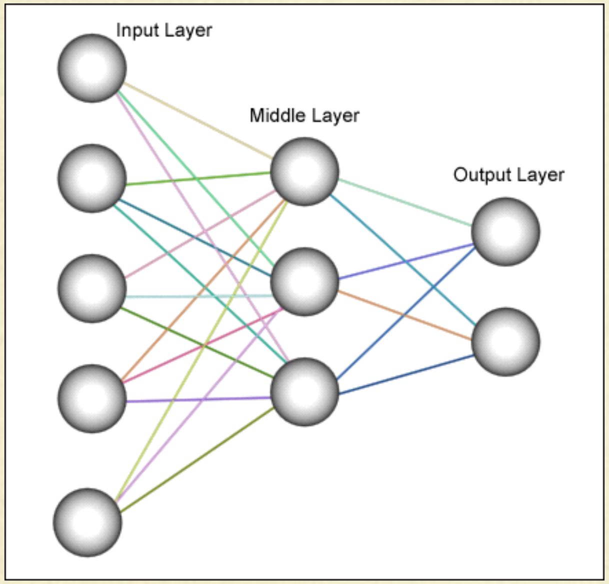
Pylearn2 Code Structure

- Dataset specification
  - Uses DenseDesignMatrix class
- Model configuration
  - LR, Kmeans, Softmax, RectLinear, Autoencoder, RBM, DBM, Cuda-convnet, more…
- Training Algorithm choice
  - SGD, BGD, Dropout, Corruption
- Training execution
  - Train class
- Prediction and results

#1 Feedforward NN

- Hidden layer = sigmoid, Output layer = softmax

Input = 1024 (=32x32) bit vector

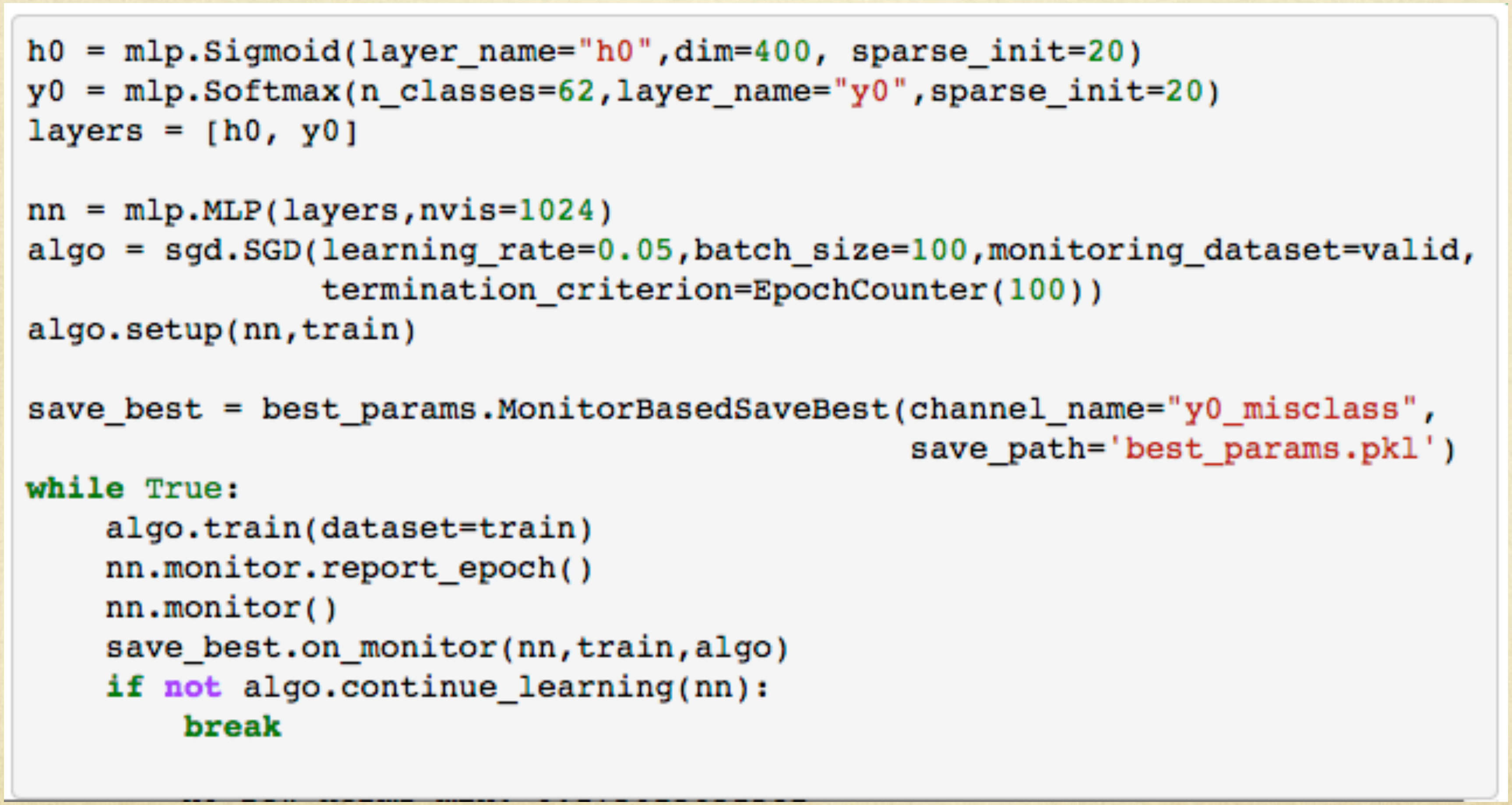


Prediction= 62 bit vector (one-hot rep)

Hyperparams

- Hidden layer = 400 neurons
- Output layer = 62 neurons
- Random initialization (symmetry breaking)
- SGD algorithm (mini-batch stochastic gradient descent)
  - Fixed learning rate = 0.05
  - Batch size = 100
- Termination = after 100 epochs

#1 pylearn2 code



```python
h0 = mlp.Sigmoid(layer_name="h0",dim=400, sparse_init=20)
y0 = mlp.Softmax(n_classes=62,layer_name="y0",sparse_init=20)
layers = [h0, y0]

nn = mlp.MLP(layers,nvis=1024)
algo = sgd.SGD(learning_rate=0.05,batch_size=100,monitoring_dataset=valid,
               termination_criterion=EpochCounter(100))
algo.setup(nn,train)

save_best = best_params.MonitorBasedSaveBest(channel_name="y0_misclass",
                                             save_path='best_params.pkl')
while True:
    algo.train(dataset=train)
    nn.monitor.report_epoch()
    nn.monitor()
    save_best.on_monitor(nn,train,algo)
    if not algo.continue_learning(nn):
        break
```

#1 Running in ipython

```
Epochs seen: 100
Batches seen: 70000
Examples seen: 7000000
learning_rate: 0.0500000119209
objective: 0.623726010323
y0_col_norms_max: 11.5188903809
y0_col_norms_mean: 6.90935611725
y0_col_norms_min: 4.80359125137
y0_max_max_class: 0.996758043766
y0_mean_max_class: 0.759563267231
y0_min_max_class: 0.221193775535
y0_misclass: 0.173800006509
y0_nll: 0.623726010323
y0_row_norms_max: 6.41655635834
y0_row_norms_mean: 2.55642676353
y0_row_norms_min: 0.48346811533
```

Accuracy = 83%

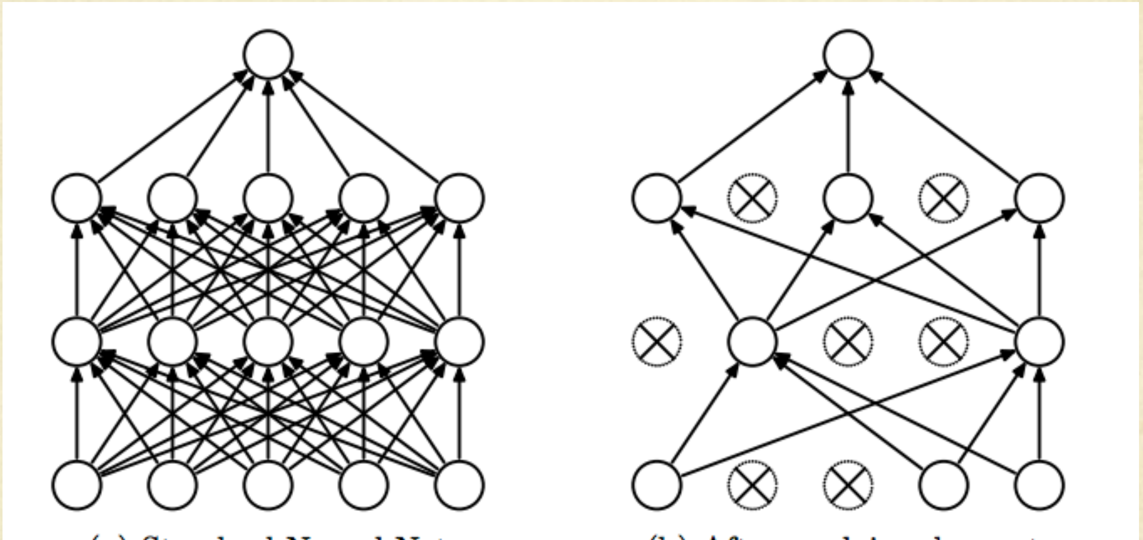
Problem: Overfitting

- It’s easy to overfit using neural networks
  - How many neurons per layer? 400? 800?
- Methods to deal with it include:
  - L1, L2, ElasticNet regularization
  - Early stopping
  - Model averaging
  - **DROPOUT**

NO. . . PLEASE, NOT DROP-OUTS!
THEY WILL KICK YOUR AS-SUMPTIONS.
5-23-09 ©2009 Scott Adams, Inc./Dist. by UFS, Inc.

Remedy: Dropout

- Dropout invented by G. Hinton to address overfitting
  - Automatically provides ensemble boosting

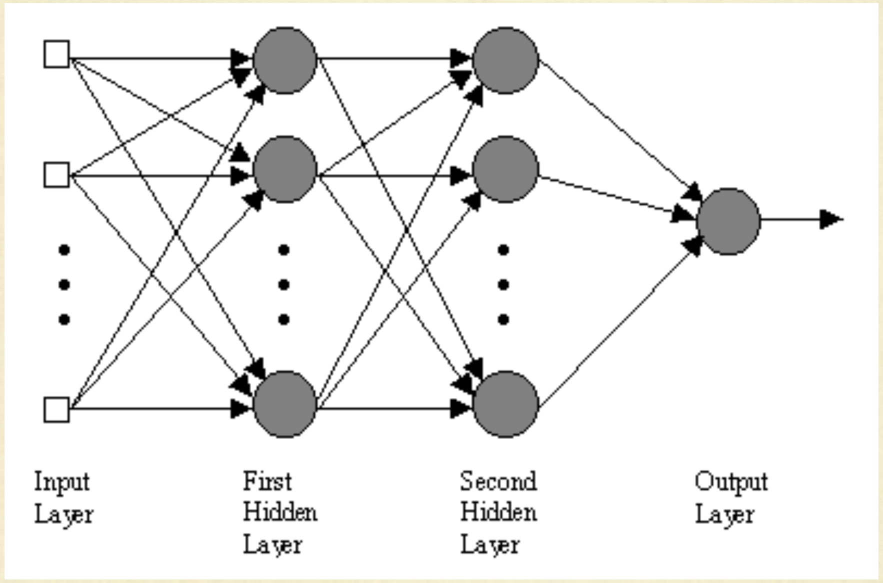

(a) Standard Neural Net (b) After applying dropout.

- Prevents neurons from co-adapting

#2 NN w/Dropout

- 2 Softplus hidden layers, Softmax output layer

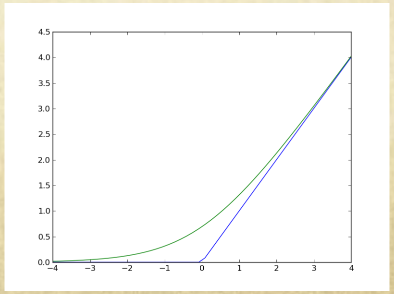
Input = 1024 (=32x32) bit vector



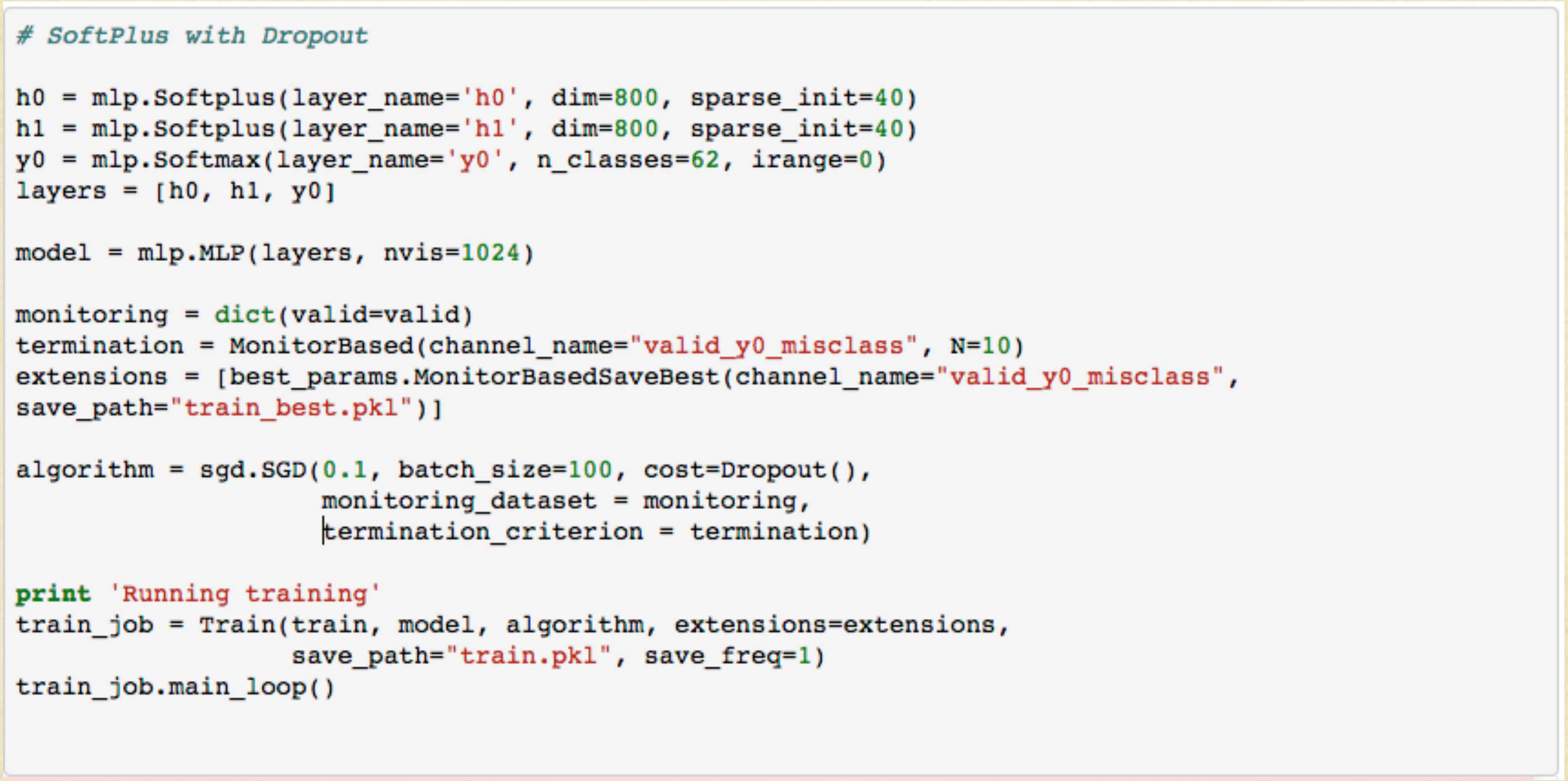
S

Prediction= 62 bit vector (one-hot rep)

Softplus $f(x) = \log(1+e^{x})$



#2 pylearn2 code



```
# SoftPlus with Dropout

h0 = mlp.Softplus(layer_name='h0', dim=800, sparse_init=40)
h1 = mlp.Softplus(layer_name='h1', dim=800, sparse_init=40)
y0 = mlp.Softmax(layer_name='y0', n_classes=62, irange=0)
layers = [h0, h1, y0]

model = mlp.MLP(layers, nvis=1024)

monitoring = dict(valid=valid)
termination = MonitorBased(channel_name="valid_y0_misclass", N=10)
extensions = [best_params.MonitorBasedSaveBest(channel_name="valid_y0_misclass",
save_path="train_best.pkl")]

algorithm = sgd.SGD(0.1, batch_size=100, cost=Dropout(),
                    monitoring_dataset = monitoring,
                    termination_criterion = termination)

print 'Running training'
train_job = Train(train, model, algorithm, extensions=extensions,
                  save_path="train.pkl", save_freq=1)
train_job.main_loop()
```

Termination Condition = stop if no improvement after N=10 epochs

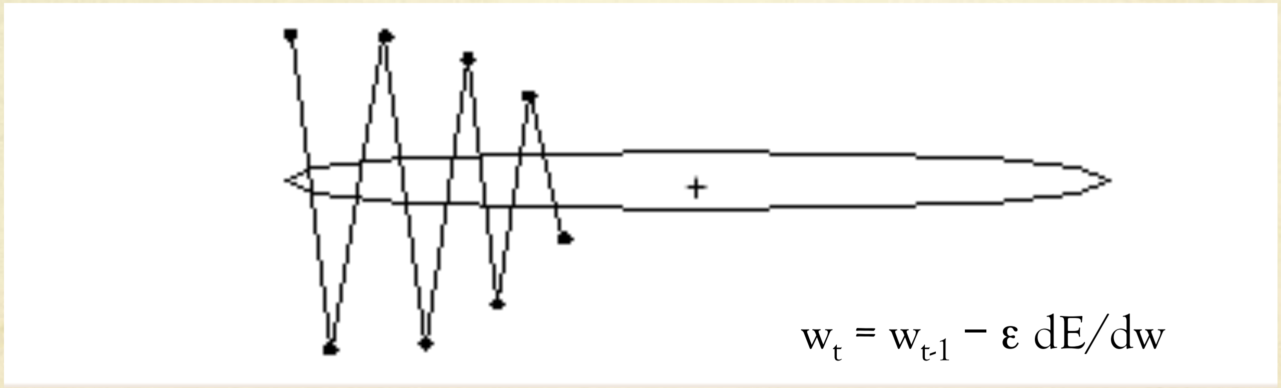
#2 running in ipython

```
Epochs seen: 93
Batches seen: 65100
Examples seen: 6510000
learning_rate: 0.100000023842
total_seconds_last_epoch: 10.1289653778
training_seconds_this_epoch: 6.20919704437
valid_objective: 1.4562972784
valid_y0_col_norms_max: 5.99511814117
valid_y0_col_norms_mean: 2.90585327148
valid_y0_col_norms_min: 2.15357899666
valid_y0_max_max_class: 0.967477440834
valid_y0_mean_max_class: 0.583315730095
valid_y0_min_max_class: 0.12644392252
valid_y0_misclass: 0.253300011158
valid_y0_nll: 0.946564733982
valid_y0_row_norms_max: 2.06650829315
valid_y0_row_norms_mean: 0.773801326752
valid_y0_row_norms_min: 0.347277522087
```

Accuracy = 76%

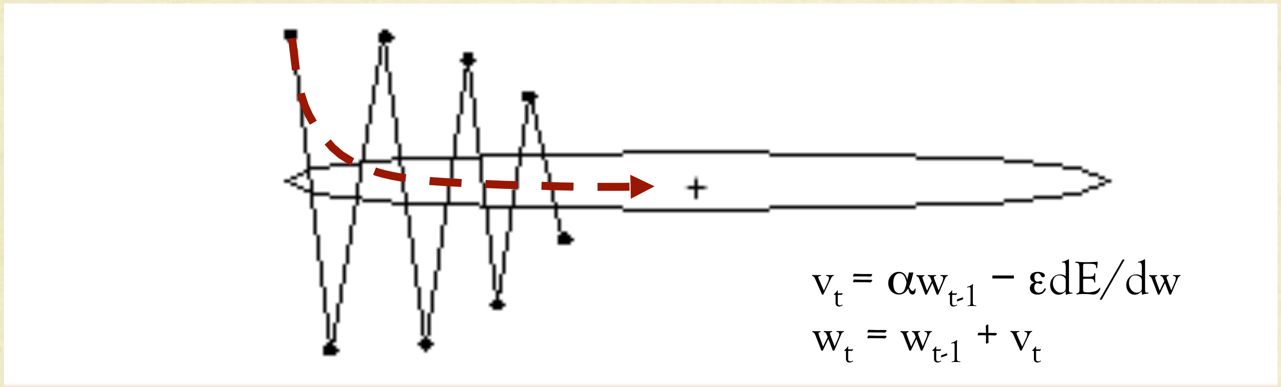
Problem: SGD speed

- SGD learning tends to be slow when curvature differs among features
- Errors change trajectory of gradient, but always perpendicular to feature surface



Remedy: Momentum

- Errors change velocity of gradient, not gradient itself



$v_t = \alpha w_{t-1} - \varepsilon dE/dw$

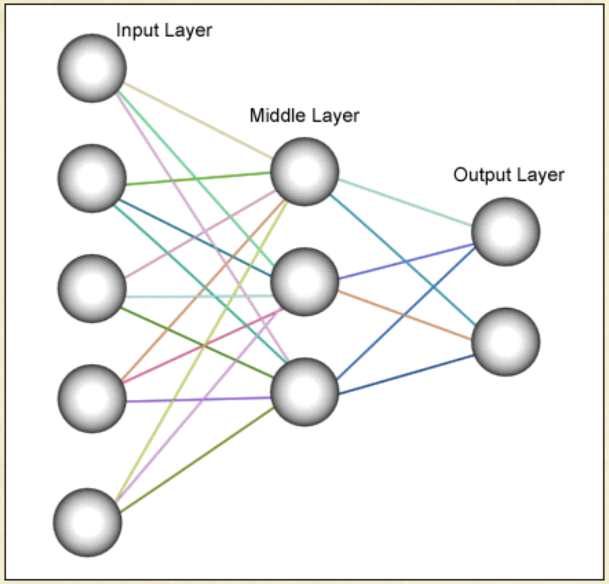
$w_t = w_{t-1} + v_t$

- Start with small momentum to dampen oscillations
- Fully implemented in Pylearn2
- Other methods (conjugate gradient, Hessian-free)

#3 NN w/Momentum

- Rectified Linear hidden layer, Softmax output layer

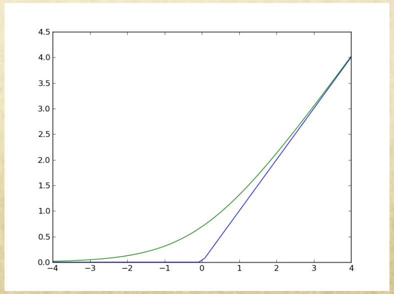
Input = 1024 (=32x32) bit vector



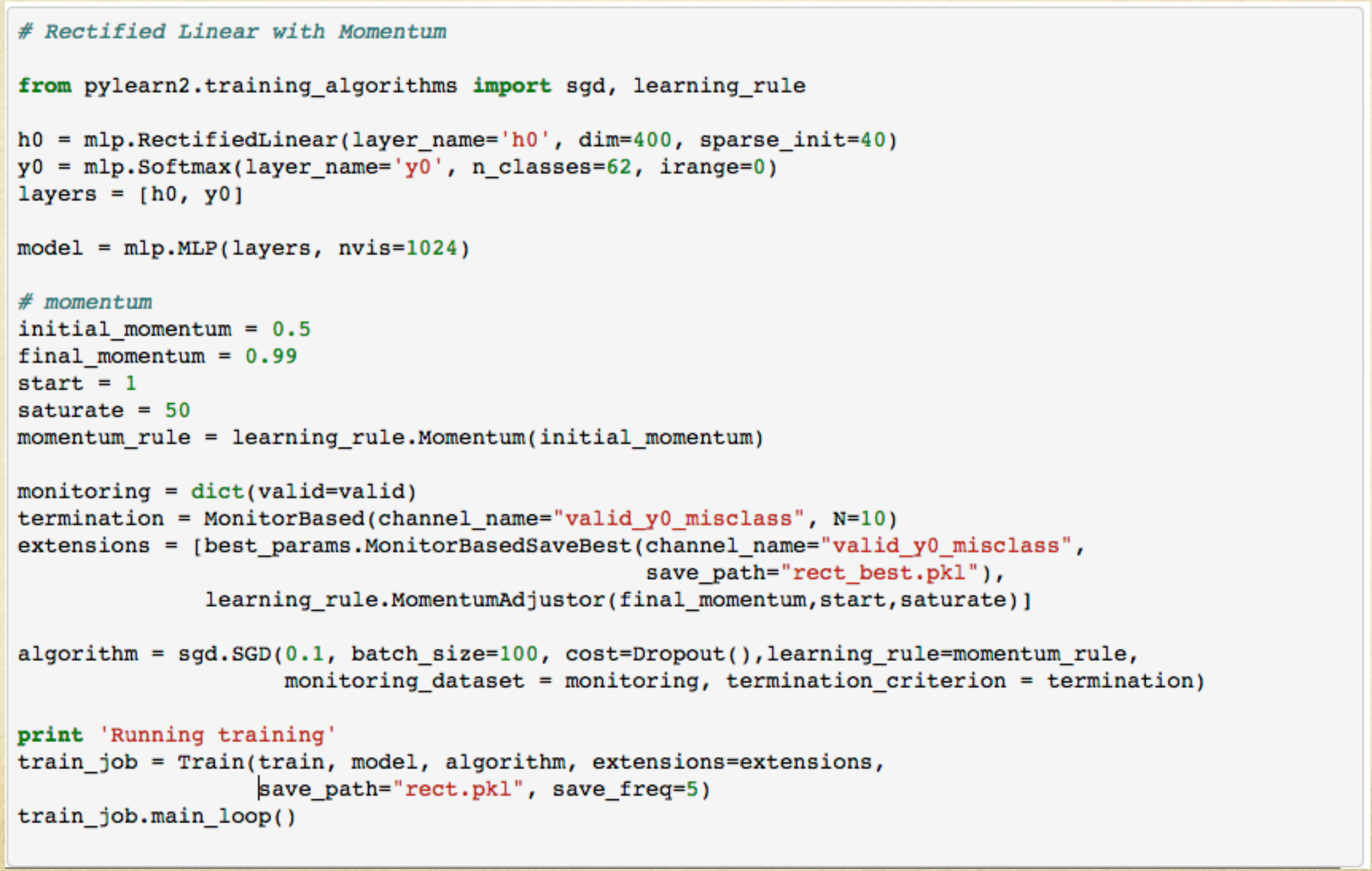
Prediction= 62 bit vector (one-hot rep)

Hidden = 400 neurons
Initial Momentum = 0.5
Final Momentum = 0.99

Rectified Linear
f(x) = max(x,0)



#3 pylearn2 code



```
# Rectified Linear with Momentum

from pylearn2.training_algorithms import sgd, learning_rule

h0 = mlp.RectifiedLinear(layer_name='h0', dim=400, sparse_init=40)
y0 = mlp.Softmax(layer_name='y0', n_classes=62, irange=0)
layers = [h0, y0]

model = mlp.MLP(layers, nvis=1024)

# momentum
initial_momentum = 0.5
final_momentum = 0.99
start = 1
saturate = 50
momentum_rule = learning_rule.Momentum(initial_momentum)

monitoring = dict(valid=valid)
termination = MonitorBased(channel_name="valid_y0_misclass", N=10)
extensions = [best_params.MonitorBasedSaveBest(channel_name="valid_y0_misclass",
                                               save_path="rect_best.pkl"),
              learning_rule.MomentumAdjustor(final_momentum,start,saturate)]

algorithm = sgd.SGD(0.1, batch_size=100, cost=Dropout(),learning_rule=momentum_rule,
                    monitoring_dataset = monitoring, termination_criterion = termination)

print 'Running training'
train_job = Train(train, model, algorithm, extensions=extensions,
                  save_path="rect.pkl", save_freq=5)
train_job.main_loop()
```

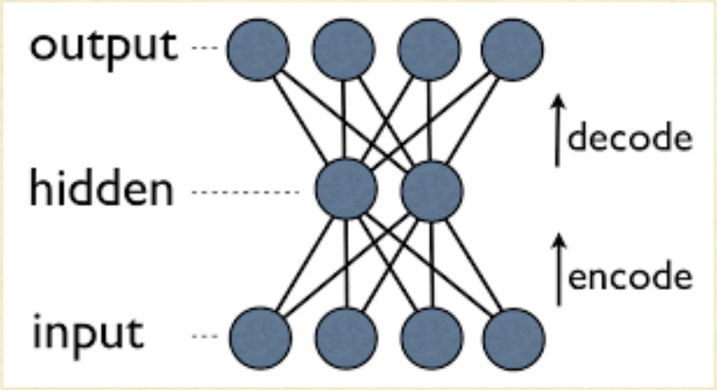
#3 running in ipython

```
Epochs seen: 27
Batches seen: 18900
Examples seen: 1890000
learning_rate: 0.100000023842
momentum: 0.760000526905
total_seconds_last_epoch: 4.66804122925
training_seconds_this_epoch: 2.96171355247
valid_objective: 1.42965459824
valid_y0_col_norms_max: 4.5757818222
valid_y0_col_norms_mean: 3.47835850716
valid_y0_col_norms_min: 2.6680624485
valid_y0_max_max_class: 0.962530076504
valid_y0_mean_max_class: 0.555216372013
valid_y0_min_max_class: 0.12590457499
valid_y0_misclass: 0.249799996614
valid_y0_nll: 0.971761405468
valid_y0_row_norms_max: 2.2137401104
valid_y0_row_norms_mean: 1.3540699482
valid_y0_row_norms_min: 0.666570603848
```

Accuracy = 76%

Autoencoders

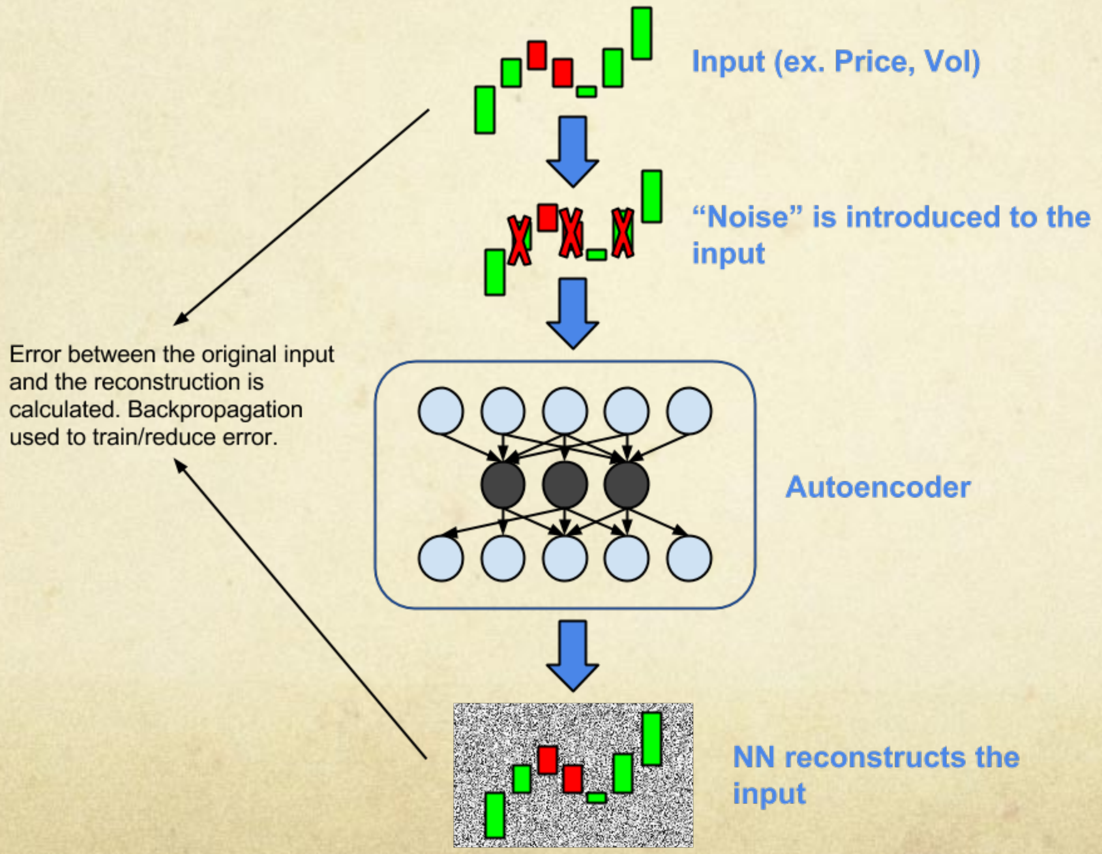
- Learns efficient codings (dimensionality reduction)
  - Linear units = similar to PCA (+rotation)
  - Nonlinear units = manifold learning



- Autoencoders are deterministic (not good at predicting)

Denoising Autoencoders

- Corrupt input with noise (randomly set to 0), then force hidden layer to predict input

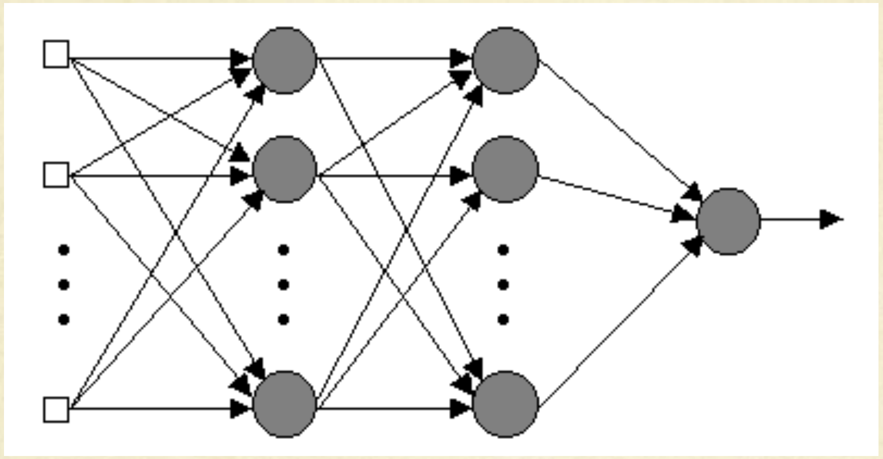

Input (ex. Price, Vol)
"Noise" is introduced to the input
Error between the original input and the reconstruction is calculated. Backpropagation used to train/reduce error.
Autoencoder
NN reconstructs the input

source: neuraltip

#4 Stacked DAE

- Autoencoders for hidden layers 1 & 2, softmax output layer

Unsupervised
Training for
Autoencoder
Hidden Layers

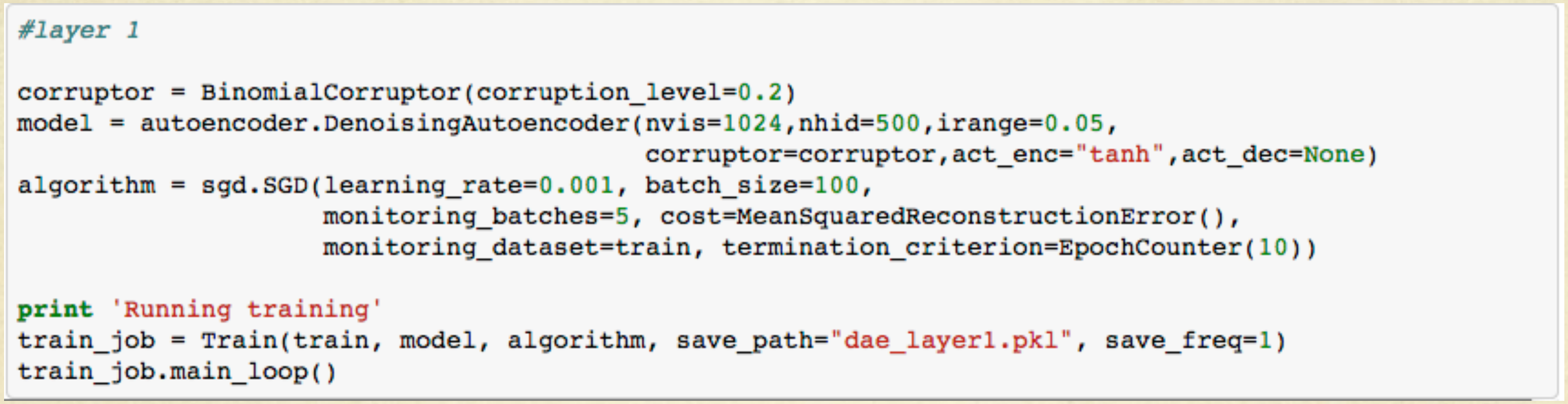


Supervised
Training to
fine-tune
Stacked DAE

- Pre-train Hidden Layers 1 & 2 sequentially, stack a softmax output layer on top and fine tune

#4 training layer 1

- Pre-train Layer 1, save weights

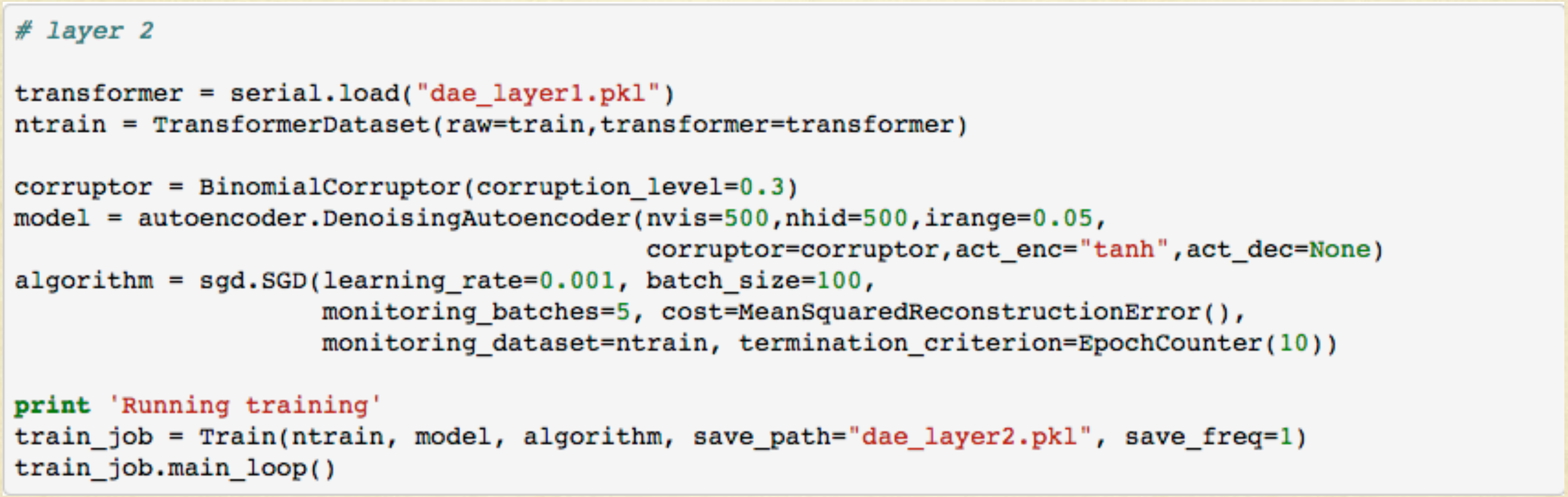


```
#layer 1

corruptor = BinomialCorruptor(corruption_level=0.2)
model = autoencoder.DenoisingAutoencoder(nvis=1024,nhid=500,irange=0.05,
                                         corruptor=corruptor,act_enc="tanh",act_dec=None)
algorithm = sgd.SGD(learning_rate=0.001, batch_size=100,
                    monitoring_batches=5, cost=MeanSquaredReconstructionError(),
                    monitoring_dataset=train, termination_criterion=EpochCounter(10))

print 'Running training'
train_job = Train(train, model, algorithm, save_path="dae_layer1.pkl", save_freq=1)
train_job.main_loop()
```

```
Epochs seen: 10
Batches seen: 7000
Examples seen: 700000
learning_rate: 0.0010000000475
objective: 9.47059440613
total_seconds_last_epoch: 5.54894971848
training_seconds_this_epoch: 3.88439941406
```

#4 training layer 2

- Pre-train layer 2, using outputs of layer 1 as inputs

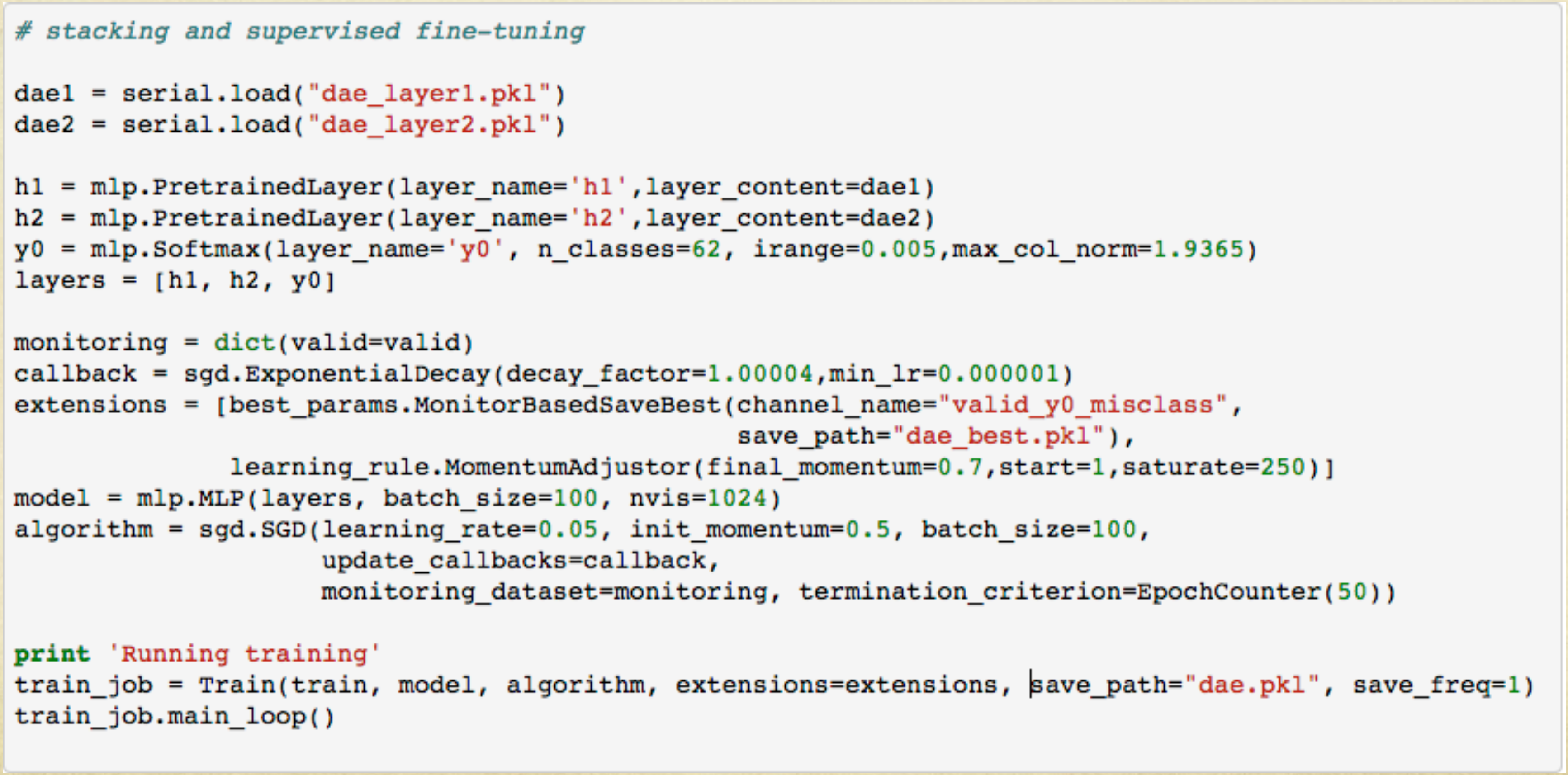


```
# layer 2

transformer = serial.load("dae_layer1.pkl")
ntrain = TransformerDataset(raw=train,transformer=transformer)

corruptor = BinomialCorruptor(corruption_level=0.3)
model = autoencoder.DenoisingAutoencoder(nvis=500,nhid=500,irange=0.05,
                                         corruptor=corruptor,act_enc="tanh",act_dec=None)
algorithm = sgd.SGD(learning_rate=0.001, batch_size=100,
                    monitoring_batches=5, cost=MeanSquaredReconstructionError(),
                    monitoring_dataset=ntrain, termination_criterion=EpochCounter(10))

print 'Running training'
train_job = Train(ntrain, model, algorithm, save_path="dae_layer2.pkl", save_freq=1)
train_job.main_loop()
```

```
Epochs seen: 10
Batches seen: 7000
Examples seen: 700000
learning_rate: 0.0010000000475
objective: 2.91170930862
total_seconds_last_epoch: 4.94157600403
training_seconds_this_epoch: 3.63193702698
```

#4 Fine-tuning

- SGD with momentum for 50 Epochs



```
# stacking and supervised fine-tuning

dae1 = serial.load("dae_layer1.pkl")
dae2 = serial.load("dae_layer2.pkl")

h1 = mlp.PretrainedLayer(layer_name='h1',layer_content=dae1)
h2 = mlp.PretrainedLayer(layer_name='h2',layer_content=dae2)
y0 = mlp.Softmax(layer_name='y0', n_classes=62, irange=0.005,max_col_norm=1.9365)
layers = [h1, h2, y0]

monitoring = dict(valid=valid)
callback = sgd.ExponentialDecay(decay_factor=1.00004,min_lr=0.000001)
extensions = [best_params.MonitorBasedSaveBest(channel_name="valid_y0_misclass",
                                               save_path="dae_best.pkl"),
              learning_rule.MomentumAdjustor(final_momentum=0.7,start=1,saturate=250)]
model = mlp.MLP(layers, batch_size=100, nvis=1024)
algorithm = sgd.SGD(learning_rate=0.05, init_momentum=0.5, batch_size=100,
                    update_callbacks=callback,
                    monitoring_dataset=monitoring, termination_criterion=EpochCounter(50))

print 'Running training'
train_job = Train(train, model, algorithm, extensions=extensions, save_path="dae.pkl", save_freq=1)
train_job.main_loop()
```

Classification Results

Model	Accuracy
Logistic Regression	59%
SVM w/RBF	65%
Random Forest	69%
Extremely Random Forest	73%
2-layer Sigmoid	83%
3-layer Softplus w/dropout	76%
2-layer RectLin w/momentum	76%
Stacked DAE w/momentum	89%

Tools

- OpenCV – computer vision library
- Scikit-image – image processing library
- Scikit-learn – ML on python
- Theano – fast tensor math library
- Pylearn2 – neural networks on python
- Nolearn – deep learning on python
- DL4J – deep learning on Java
- Torch7 – ML on LUA
- MLlib – ML on Spark

LAST WORDS

I LIKE CHINESE FOOD

- STACKED DAE