Deep Learning Toolbox

Stavros Petridis
Department of Computing
Imperial College London
London, UK

February 14, 2017

1 Introduction

This is a deep learning toolbox which supports Deep Belief Networks (DBNs), Stacked Denoising Autoencoders (SDAE) and Deep Neural Networks (DNNs). The toolbox is inspired by the code provided by Rasmus Berg Palm [1,4] and includes several ideas from Hinton's code [2].

2 DBN Parameters

2.1 Parameters for RBMs: dbnParams.rbmParams

- epochs: No. epochs for RBM training (default = 10)
- batchsize: Batchsize used for stochastic gradient descent (default = 100)

The default values for learning rates, weight decay L2 coefficient and momentum below are the same as in the piece of code provided by Hinton [2].

- lrW: Learning rate for weights (default = 0.1)
- lrVb: Learning rate for visible biases (default = 0.1)
- lrHb: Learning rate for hidden biases (default = 0.1)

According to [3] a lower learning rate is needed when one layer is either linear or rectified linear.

- lrW_linear: Learning rate for weights when either hidden or visible layers are linear or rectified linear (default = 0.001)
- lrVb_linear: Learning rate for visible biases when either hidden or visible layers are linear or rectified linear (default = 0.001)
- lrHb_linear: Learning rate for hidden biases when either hidden or visible layers are linear or rectified linear (default = 0.001)

According to [3] a good range for the weight decay L2 coefficient is between 0.01 and 0.00001.

- weightPenaltyL2: L2 Weight decay coefficient (default value = 0.0002)
- initMomentum: momentum value as long as epoch $\leq$ momentumEpochThres (default = 0.5)
- finalMomentum: momentum value as long as epoch $>$ momentumEpochThres (default = 0.9)
- momentumEpochThres: epoch number after which the final momentum is used. For epochs 1 to *momentumEpochThres* the initial momentum is used (default = 5)
- type: training type of contrastive divergence (default = 1)
  - 1 is what Hinton suggests in [3], i.e., "When the hidden units are being driven by data, always use stochastic binary states. When they are being driven by reconstructions, always use probabilities without sampling. Assuming the visible units use the logistic function, use real-valued probabilities for both the data and the reconstructions. When collecting the pairwise statistics for learning weights or the individual statistics for learning biases, use the probabilities, not the binary states"
  - 2 is consistent with theory. Check trainRBM for more details on how these approaches are implemented.

2.2 Parameters for DBNs: dbnParams

- type: defines the type of network that will be created
  - 1 for training an autoencoder
  - 2 for training a classifier
- inputActivationFunction: activation function of the visible layer, it should be 'sigm' for binary inputs (like MNIST) and 'linear' for continuous inputs like images.
- hiddenActivationFunctions: cell array containing the activation functions of each hidden layer, e.g. for 4 sigmoid layers it should be {'sigm', 'sigm', 'sigm', 'sigm'}. If the DBN will be used as an autoencoder then the last layer (which is usually the bottleneck layer) should be linear, i.e., {'sigm', 'sigm', 'sigm', 'linear'}

The following activation functions are available:

1. Sigmoid ('sigm')
2. Linear ('linear')
3. Rectified Linear Unit ('ReLu')

- hiddenLayers: vector array containing the size of the hidden layers, e.g. [500 500 500 200]. In case an autoencoder is used then a series of decreasing layers is usually used, e.g., [1000 500 250 50].
- weight Initialisation: Weights are initialised randomly from a gaussian distribution with mean 0 and standard deviation either 0.1 (if sigmoid neurons are used) or 0.01 (if ReLu neurons are used). Biases are set to 0 (in trainRBM) as in Hinton's code [2].

3 Unfolding DBN to an NN: unfoldDBNtoNN.m

Once a DBN has been created then it is unfolded to an NN. The function initialises a neural network using the default parameters and it also copies to the network the weights and biases returned from unfoldDBNtoAE or unfoldDBNToClsf.

3.1 Unfolding DBN to an AE: unfoldDBNtoAE.m

The DBN is used as the encoding layer and then the hidden layers are mirrored and added to the existing network as a decoding layer, e.g., if the hidden layers of a DBN are [1000 500 250 50] then the AE layers are [1000 500 250 50 250 500 1000 inputSize]. The same applies to activation functions. The function returns:

- weightsAE: cell array containing the weights of the AE
- biasesAE: cell array containing the biases of the AE
- newActivationFunctions: cell array containing the activation functions of all the layers
- newLayers: vector containing the size of the hidden and output layers

3.2 Unfolding DBN to a classifier: unfoldDBNToClsf.m

Adds a softmax output layer on top of the DBN's hidden layers. The weights and biases for the added layer are initialised randomly from a gaussian distribution with 0 mean and standard deviation 0.1, in the same way as in Hinton's code [2]. The function returns:

- weightsClsf: cell array containing the weights of the classifier
- biasesClsf: cell array containing the biases of the classifier
- newActivationFunctions: cell array containing the activation functions of all the layers
- newLayers: vector containing the size of the hidden and output layers

4 SDAE Parameters

Each denoising autoencoder is initialised using the same parameters as a standard NN (see section 6). The parameters type, inputActivationFunction, hiddenActivationFunctions and hiddenLayers are the same as those in section 2.2.

5 Unfolding SDAE to an NN: unfoldSDAEtoNN.m

Once a SDAE has been created then it is unfolded to an NN. The function initialises a neural network using the default parameters and it also copies to the network the weights and biases returned from unfoldSDAEtoAE or unfoldDBNToClsf (the same function as before is used since we just need to copy the weights/biases and add a softmax layer).

5.1 Unfolding SDAE to an AE: unfoldSDAEtoAE.m

The SDAE is used as the encoding layer and then the hidden layers are mirrored and added to the existing network as a decoding layer, e.g., if the hidden layers of a SDAE are [1000 500 250 50] then the AE layers are [1000 500 250 50 250 500 1000 inputSize]. The same applies to activation functions. The main difference with unfoldDBNToAE is that we add random biases (drawn from a gaussian distribution with mean 0 and standard deviation 0.1) in the final layer (same size as input) since unlike DBNs there are no biases for the input. The function returns:

- weightsAE: cell array containing the weights of the AE
- biasesAE: cell array containing the biases of the AE
- newActivationFunctions: cell array containing the activation functions of all the layers
- newLayers: vector containing the size of the hidden and output layers

5.2 Unfolding SDAE to a classifier: unfoldDBNToClsf.m

The same function as before is used (unfoldDBNToClsf) since we just we just need to copy the weights/biases and add a softmax layer. See unfoldDBNToClsf in section 3.2.

6 NN Params

- layersSize: vector array containing the size of the hidden and output layers
- noLayers: number of layers
- epochs: No. Epochs for NN training (default = 1000)
- batchsize = batchsize used for stochastic gradient descent (default = 100)
- activation_functions: cell array containing the activation functions of hidden and output layers. If the NN will be used as an autoencoder then the middle layer should be linear, e.g., {'sigm', 'sigm', 'linear','sigm', 'sigm', 'same as input layer' }. If it is used a classifier then the last layer will be softmax, e.g., {'sigm', 'sigm', 'sigm', 'softmax'}

The available activation functions are the following:

1. Sigmoid ('sigm')
2. Linear ('linear')
3. Rectified Linear Unit ('ReLu')
4. Hyperbolic tangent ('tanh')
5. Leaky Rectified Linear Unit ('leakyReLu')
6. Softmax ('Softmax') - Can only be used as an output layer

6.1 Learning Rate Parameters: nn.trParams.lrParams

- lr: learning rate used during training. It is updated after each epoch according to the scheduling type. The user should define the initial learning rate and the scheduling type.
- initialLR: initial learning rate (default = 0.01)
- schedulingType: scheduling type for learning rate (default = 1)

  - If 1 then the learning rate remains constant (= *initialLR*) for epochs 1 to *lrEpochThres* and then decreases as follows lr = $\frac{initialLR \times lrEpochThres}{max(currentEpoch, lrEpochThres)}$.
  - If 2 then the learning rate remains constant (= *initialLR*) for epochs 1 to *lrEpochThres* and then is multiplied by the *scalingFactor* after each epoch, $lr_t = lr_{t-1} \times scalingFactor$
  - If 3 then the learning rate constantly decreases starting from the initial value (*initialLR*) and becomes half after *lrEpochThres* epochs, lr = $\frac{initialLR}{1+\frac{currentEpoch}{lrEpochThres}}$

- scalingFactor: Scaling factor for the learning rate, used only when schedulingType = 2. If it's set to 1 then the learning rate remains constant throughout training. Learning rate usually decreases during training so a value less than 1 should be used (default = 0.999)

- lrEpochThres:
  - schedulingType = 1, 2. Epoch number after which the learning rate begins to decrease (default = 50)
  - scheduling type = 3. The epoch number where the learning rate is half of the initial learning rate (default = 50)

6.2 Momentum Parameters: nn.trParams.momParams

- momentum: momentum used during training. It is updated after each epoch according to the momentum scehduling type.

- schedulingType: scheduling type for momentum, at the moment only one type is supported (default = 1)
  - If 1 then linear increase between initial (*initialMomentum*)and final value (*finalMomentum*). Final value is reached after *momentumEpochThres* epochs.

- initialMomentum: initial momentum (default = 0.5)

- finalMomentum: final momentum, used when scheduling type = 1. It is the momentum value used after epoch ≥ *momentumEpochThres* (default = 0.9)

- momentumEpochLowerThres: the epoch number after which momentum begins to increase. Momentum is constant to initialMomentum value for epochs 1 to momentumEpochLowerThres. (default = 50)
- momentumEpochUpperThres: the epoch number after which momentum becomes constant. Momentum increases linearly from momentumEpochLowerThres to momentumEpochUpperThres when it reaches the finalMomentum value. (default = 100)
- scaleLR: If 1 then the update rule for SGD+momentum (nn.trainingMethod = 2) is the following $\Delta W(t) = \mu \Delta W(t-1) + (1-\mu)(-lr\frac{\partial E}{\partial W})$.

  If 0 then the update rule is $\Delta W(t) = \mu \Delta W(t-1) + (-lr\frac{\partial E}{\partial W})$. (default = 0)

6.3 Weight Initialisation Parameters: nn.weightInitParams

- type: Weight Initialisation Type (default = 8)
  - If 1 then weights are initialised from a gaussian distribution with mean *mean* and standard deviation *sigma*
  - If 2 then weights are initialised uniformly in the range [lowerLimit, upperLimit]
  - If 3 then weights are initialised uniformly in the range [-r, r] where $r = \frac{sqrt(3)}{\sqrt{sizeOfPreviousLayer}}$. From Efficient BackProp by LeCun et al., 1998
  - If 4 then weights are initialised from a gaussian distribution with 0 mean and standard deviation $\frac{1}{\sqrt{sizeOfPreviousLayer}}$. From Efficient BackProp by LeCun et al., 1998
  - If 5 then weights are initialised uniformly in the range [-r, r] where $r = sqrt(3) \times g \times \sqrt{\frac{2}{sizeOfPreviousLayer+sizeOfCurrentLayer}}$. g is a parameter which depends on the activation function used. From Understanding the difficulty of training deep feedforward neural networks by Glorot and Bengion, JMRL, 2010
  - If 6 then weights are initialised from a gaussian distribution with 0 mean and standard deviation $g \times \sqrt{\frac{2}{sizeOfPreviousLayer+sizeOfCurrentLayer}}$ g is a parameter which depends on the activation function used.

From Understanding the difficulty of training deep feedforward neural networks by Glorot and Bengion, JMRL, 2010

  - If 7 then weights are initialised uniformly in the range [-r, r] where $r = sqrt(3) \times \sqrt{\frac{2}{sizeOfPreviousLayer}}$. From Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, He et al., ICCV 2015
  - If 8 then weights are initialised from a gaussian distribution with 0 mean and standard deviation $\sqrt{\frac{2}{sizeOfPreviousLayer}}$. From Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, He et al., ICCV 2015
  - If 9 then sparse initialisation is used where most of the weights are set to 0 and just a small set of weights is initialised from a gaussian distribution with mean *mean* and standard deviation *sigma*. The percentage of zero weights is specified by the parameter *sparsity*. From Deep Learning via Hessian-free optimisation by Martens, ICML 2010
  - If 10 then orthogonal initialisation is used, see Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by Saxe et al.

- mean: mean of gaussian used to initialise weights, applies to types 1 and 9 only (default = 0)
- sigma: standard deviation of gaussian used to initialse weights, applies to type 1 and 9 only (default = 0.1)
- lowerLimit: min value of the range used for uniform initialisation, applies to type 2 only (default = -1)
- upperLimit: max value of the range used for uniform initialisation, applies to type 2 only (default = 1)
- sparsity: defines the percentage of zero weights in a given layer. Applies to type 9 only (default 0.8)
- biasType: Bias Initialisation Type (default = 2)
  - If 1 then biases initialised from a gaussian with mean *mean* and standard deviation *sigma*

  - If 2 then biases are constant (= *biasConstant*).

- biasConstant = value used to initialise the biases to a constant value when biasType = 2 (default = 0)

6.4 Weight Constraint Parameters: nn.weightConstraints

- weightPenaltyL2: L2 regularization coefficient. When 0 this constraint is not active (default = 0)
- weightPenaltyL1: L1 regularisation coefficient. When 0 this constraint is not active (default = 0)
- maxNormConstraint: Max norm regularisation coefficient (c). It constraints the vector of weights of each each hidden unit to have a maximum norm of c, i.e., $\|w\| \leq$ c .When 0 this constraint is not active. Suggested values in [5] are 3 or 4 (default = 0)

6.5 Dropout Parameters: nn.dropoutParams

- dropoutType: dropout type (default = 0)
  - If 0 then no dropout is used
  - If 1 then Bernoulli dropout is used
- dropoutPresentProbVis: present probability for a visible node, use 1 if no input layer dropout is needed (default = 0.8)
- dropoutPresentProbHid: present probability for a hidden node, use 1 if no hidden layer dropout is needed (default = 0.5)

6.6 Training Algorithms: nn.trParams

- adagrad.epsilon: ϵ value for adagrad (default = 10^{-6})
- adadelta.epsilon: ϵ value for adadelta (default = 10^{-6})
- adadelta.gamma: γ value for adadelta (default = 0.95)
- rmsprop.epsilon: ϵ value for rmsprop (default = 10^{-6})

- rmsprop.gamma: γ value for rmsprop (default = 0.9)
- adam.b1: b_1 value for adam (default = 0.9)
- adam.b2: b_2 value for adam (default = 0.999)
- adam.epsilon: ϵ value for adam (default = 10^{-8})

If adam is used then the recommended value for initial learning rate is 0.001.

6.7 Other Parameters: nn

- inputZeroMaskedFraction: percentage of input units that will be set to zero, used for denoising autoencoders (default = 0)
- earlyStopping: If 1 then early stopping is used, if 0 then it's disabled (default = 0)
- max_fail: maximum number of increases in the validation set until training stops, used in early stopping (default = 10)
- trainingMethod: 1 = SGD, 2 = SGD with momentum, 3 = SGD with Nesterov momentum, 4 = adagrad, 5 = adadelta, 6 = rmsprop, 7 = adam
- pretraining: flag indicating if pretraining has been used (if yes then 1 otherwise 0), it is automatically set to 1 by unfoldDBNtoNN or unfoldSDAEtoNN (default = 0)
- testing: variable which determines if the network is in training (=0) or testing (=1) mode. The network is initialised by nnsetup in training mode, therefore its value is 0, and before it can be used for testing it should be changed to 1.
- diagnostics: If 1 then the mean / st. dev. of neuron activations is computed in each layer together with the ratio norm(ΔW) / norm(w). (default = 1)
- showDiagnostics: The above statistics are computed and displayed every *showDiagnostics* epochs (default = 10)

- showPlot: If 1 then plots the training and validation loss in every epoch. If 0 then the plot is not shown (default = 1)

7 Visualisation Parameters: visParams

- noExamplesPerSubplot: defines how many images per subplot will be shown, see below
- noSubplots: $= \text{floor}(\frac{hiddenLayers(1)}{noExamplesPerSubplot})$, it automatically computes how many subplots are needed
- col: the number image columns
- row: the number image rows

8 Use of default parameters when dropout is used

- If dropout is used then:
  - If pretraining ($nn.pretraining = 1$) then use low learning rate and no weight constraints. In addition, the weights should be scaled by $\frac{1}{p}$.
  - If no pretraining ($nn.pretraining = 0$) then use high learning rate and momentum and max-norm regularisation (suggested values 3,4 [5]).
- If dropout is used then a larger network should be used than what would be used without dropout [5] and trained for more epochs.

References

[1] `https://github.com/rasmusbergpalm/DeepLearnToolbox`, 2012.

[2] `http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html`, 2016.

[3] G. Hinton. A practical guide to training RBMs. https://www.cs.toronto.edu/ hinton/absps/guidetr.pdf, 2010.

[4] R. B. Palm. Prediction as a candidate for learning deep hierarchical models of data, 2012.

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.