Review of auto-encoders

Piotr Mirowski, Microsoft Bing London

Deep Learning Meetup
March 26, 2014
Outline

• Deep learning concepts covered
  o Hierarchical representations
  o Sparse and/or distributed representations
  o Supervised vs. unsupervised learning

• Auto-encoder
  o Architecture
  o Inference and learning
  o Sparse coding
  o Sparse auto-encoders

• Illustration: handwritten digits
  o Stacking auto-encoders
  o Learning representations of digits
  o Impact on classification

• Applications to text
  o Semantic hashing
  o Semi-supervised learning
  o Moving away from auto-encoders

• Topics not covered in this talk
Outline

• Deep learning concepts covered
  o Hierarchical representations
  o Sparse and/or distributed representations
  o Supervised vs. unsupervised learning

• Auto-encoder
  o Architecture
  o Inference and learning
  o Sparse coding
  o Sparse auto-encoders

• Illustration: handwritten digits
  o Stacking auto-encoders
  o Learning representations of digits
  o Impact on classification

• Applications to text
  o Semantic hashing
  o Semi-supervised learning
  o Moving away from auto-encoders

• Topics not covered in this talk
Hierarchical representations

“Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. Automatically learning features at multiple levels of abstraction allows a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features.” — Yoshua Bengio

[Benio, Learning Deep Architectures for AI, 2009]
Sparse and/or distributed representations

Biological motivation: V1 visual cortex

\[
7 = 1 \square + 1 \square + 1 \square + 1 \square + 1 \square + 1 \square + 0.8 \square + 0.8 \square
\]

Example on MNIST handwritten digits
An image of size 28x28 pixels can be represented using a small combination of codes from a basis set.

Sparse and/or distributed representations

Biological motivation: V1 visual cortex (backup slides)

Example on MNIST handwritten digits
An image of size 28x28 pixels can be represented using a small combination of codes from a basis set.

At the end of this talk, you should know how to learn that basis set and how to infer the codes, in a 2-layer auto-encoder architecture. Matlab/Octave code and the MNIST dataset will be provided.

Supervised learning

\[ \hat{Y} = f(X; W, b) \]

\[ E_f(Y, \hat{Y}) \]

Input \( X \)

Prediction \( \hat{Y} \)

Error \( E_f(Y, \hat{Y}) \)

Target \( Y \)
Supervised learning

\[ \hat{Y} = W^T X + b \]

\[ \| Y - \hat{Y} \|^2 \]

Input

Prediction

Error

Target
Why not exploit unlabeled data?
Unsupervised learning

No target...

No error...

Prediction

\[ \hat{Y} = f(X; W, b) \]

Input

\[ X \]
Unsupervised learning

Code
“latent/hidden” representation

Error(s)

Prediction(s)

Input

X

Z
Unsupervised learning

We want the codes to represent the inputs in the dataset.

The code should be a compact representation of the inputs: low-dimensional and/or sparse.
Examples of unsupervised learning

- Linear decomposition of the **inputs**:
  - Principal Component Analysis and Singular Value Decomposition
  - Independent Component Analysis [Bell & Sejnowski, 1995]
  - **Sparse coding** [Olshausen & Field, 1997]
  - ...

- Fitting a distribution to the **inputs**:
  - Mixtures of Gaussians
  - Use of **Expectation-Maximization algorithm** [Dempster et al, 1977]
  - ...

- For text or discrete data:
  - Latent Semantic Indexing [Deerwester et al, 1990]
  - Probabilistic Latent Semantic Indexing [Hofmann et al, 1999]
  - **Semantic Hashing**
  - ...

Objective of this tutorial

Study a fundamental building block for deep learning, the **auto-encoder**
Outline

• Deep learning concepts covered
  o Hierarchical representations
  o Sparse and/or distributed representations
  o Supervised vs. unsupervised learning

• Auto-encoder
  o Architecture
  o Inference and learning
  o Sparse coding
  o Sparse auto-encoders

• Illustration: handwritten digits
  o Stacking auto-encoders
  o Learning representations of digits
  o Impact on classification

• Applications to text
  o Semantic hashing
  o Semi-supervised learning
  o Moving away from auto-encoders

• Topics not covered in this talk
Auto-encoder

Target = input

Code

Input

"Bottleneck" code
i.e., low-dimensional, typically dense, distributed representation

Y = X

Z

X

"Overcomplete" code
i.e., high-dimensional, always sparse, distributed representation

Target = input

Code

Input
Auto-encoder

$E_g(Z, \hat{Z})$

$\hat{Z} = g(X; C, b_C)$

Input

Code prediction

Encoding “energy”

$E_h(X, \hat{X})$

Decoding “energy”

Decoding

Input decoding
Auto-encoder

\[ \hat{Z} = g(X; C, b_C) \]

\[ \|Z - \hat{Z}\|_2^2 \]

\[ \hat{X} = h(Z; D, b_D) \]

\[ \|X - \hat{X}\|_2^2 \]
Auto-encoder loss function

For one sample $t$

$$L(X(t), Z(t); W) = \alpha \|Z(t) - g(X(t); C, b_C)\|^2_2 + \|X(t) - h(Z(t); D, b_D)\|^2_2$$

Coefficient of the encoder error

Encoding energy

Decoding energy

For all $T$ samples

$$L(X, Z; W) = \sum_{t=1}^{T} \alpha \|Z(t) - g(X(t); C, b_C)\|^2_2 + \sum_{t=1}^{T} \|X(t) - h(Z(t); D, b_D)\|^2_2$$

Encoding energy

Decoding energy

How do we get the codes $Z$?

We note $W=\{C, b_C, D, b_D\}$
Learning and inference in auto-encoders

Infer the codes $Z$ given the current model parameters $W$

$$Z^* = \arg \min_Z L(X, Z; W)$$

Learn the parameters (weights) $W$ of the encoder and decoder given the current codes $Z$

$$W^* = \arg \min_W L(X, Z; W)$$

Relationship to Expectation-Maximization in graphical models (backup slides)
Learning and inference: stochastic gradient descent

**Iterated gradient descent (?)**
on the code $Z(t)$
given the current
model parameters $W$

$Z^*(t) = \arg \min_{Z(t)} L\left(X(t), Z(t); W\right)$

**Take a gradient descent step**
on the parameters (weights) $W$
of the encoder and decoder
given the current codes $Z$

$L\left(X(t), Z(t); W^*\right) < L\left(X(t), Z(t); W\right)$

Relationship to Generalized EM
in graphical models (backup slides)
Auto-encoder

Encoding energy

Code prediction

$\|Z - \hat{Z}\|_2^2$

$\hat{Z} = C^T X + b_C$

Decoding energy

$A = \sigma(Z) = \frac{1}{1 + e^{-Z}}$

$\hat{X} = D^T A + b_D$

$\|X - \hat{X}\|_2^2$
Auto-encoder: fprop

\begin{align*}
    \text{function } & \text{e} = \text{Loss}_{\text{Gaussian}}(z, z_{\text{hat}}) \\
    z_{\text{Diff}} & = z_{\text{hat}} - z; \\
    e & = 0.5 \times \text{sum}(z_{\text{Diff}}^2);
\end{align*}

\begin{align*}
    \text{function } & z_{\text{hat}} = \ldots \\
    & \text{Module}_{\text{Encode}}_{\text{FProp}}(\text{model}, x, \text{params}) \\
    & \% \text{Compute the linear encoding activation} \\
    z_{\text{hat}} & = \text{model.C}^* x + \text{model.bias}_C;
\end{align*}

\begin{align*}
    \text{function } & \text{[x}_{\text{hat}}, a_{\text{hat}}] = \ldots \\
    & \text{Module}_{\text{Decode}}_{\text{FProp}}(\text{model}, z) \\
    & \% \text{Apply the logistic to the code} \\
    a_{\text{hat}} & = 1 ./ (1 + \text{exp}(-z)); \\
    & \% \text{Linear decoding} \\
    x_{\text{hat}} & = \text{model.D}^* a_{\text{hat}} + \text{model.bias}_D;
\end{align*}

\begin{align*}
    \text{function } & e = \text{Loss}_{\text{Gaussian}}(x, x_{\text{hat}}) \\
    x_{\text{Diff}} & = x_{\text{hat}} - x; \\
    e & = 0.5 \times \text{sum}(x_{\text{Diff}}^2);
\end{align*}
Auto-encoder
backprop w.r.t. codes

\[
\frac{\partial L_{\text{enc}}}{\partial Z} = \frac{\partial}{\partial Z} \left( \| Z - \hat{Z} \|_2^2 \right)
\]

\[
\frac{\partial A}{\partial Z} = \frac{\partial \sigma(Z)}{\partial Z}
\]

\[
\frac{\partial \hat{X}}{\partial A} = \frac{\partial}{\partial A} \left( D^T A + b_D \right)
\]

\[
\frac{\partial L}{\partial \hat{X}} = \frac{\partial}{\partial \hat{X}} \left( \| X - \hat{X} \|_2^2 \right)
\]

\[
\frac{\partial L_{\text{dec}}}{\partial Z} = \frac{\partial L}{\partial \hat{X}} \frac{\partial \hat{X}}{\partial \hat{A}} \frac{\partial \hat{A}}{\partial Z}
\]

Auto-encoder: backprop w.r.t. codes

Code

% Gradient of the loss w.r.t. activations
dL_da = model.D' * dL_dx;
% Gradient of the loss w.r.t. latent codes
% a_hat = 1 ./ (1 + exp(-z_hat))
dL_dz = dL_da .* a_hat .* (1 - a_hat);
% Add the gradient w.r.t. the encoder's outputs
dL_dz = z_star - z_hat;
% Gradient of the loss w.r.t. the decoder prediction
dL_dx_star = x_star - x;

Input

X

Code

function dL_dz = ...
    Module_Decode_BackProp_Codes(model, dL_dx, a_hat)
    dL_da = model.D' * dL_dx;
    a_hat = 1 ./ (1 + exp(-z_hat))
    dL_dz = dL_da .* a_hat .* (1 - a_hat);
    z_star = z + dL_dz;

function [z_star, z_hat, loss_star, loss_hat] = Layer_Infer(model, x, params)
% Encode the current input and initialize the latent code
z_hat = Module_Encode_FProp(model, x, params);
% Decode the current latent code
[x_hat, a_hat] = Module_Decode_FProp(model, z_hat);
% Compute the current loss term due to decoding (encoding loss is 0)
loss_hat = Loss_Gaussian(x, x_hat);
% Relaxation on the latent code: loop until convergence
x_star = x_hat; a_star = a_hat; z_star = z_hat; loss_star = loss_hat;
while (true)
    % Gradient of the loss function w.r.t. decoder prediction
    dL_dx_star = x_star - x;
    % Back-propagate the gradient of the loss onto the codes
    dL_dz = Module_Decode_BackProp_Codes(model, dL_dx_star, a_star, params);
    % Add the gradient w.r.t. the encoder's outputs
    dL_dz = dL_dz + params.alpha_c * (z_star - z_hat);
    % Perform one step of gradient descent on the codes
    z_star = z_star - params.eta_z * dL_dz;
    % Decode the current latent code
    [x_star, a_star] = Module_Decode_FProp(model, z_star);
    % Compute the current loss and convergence criteria
    loss_star = Loss_Gaussian(x, x_star) + ...
        params.alpha_c * Loss_Gaussian(z_star, z_hat);
    % Stopping criteria
    [...]
end

Auto-encoder backprop w.r.t. codes

\[
\frac{\partial L}{\partial C} = \frac{\partial L}{\partial \hat{Z}} \cdot \frac{\partial \hat{Z}}{\partial C}
\]

Code prediction

\[
\frac{\partial L}{\partial Z^*} = \frac{\partial}{\partial Z^*} \left( \| \hat{Z} - Z^* \|_2^2 \right)
\]

\[
\frac{\partial \hat{Z}}{\partial C} = \frac{\partial}{\partial C} \left( C^T X + b_D \right)
\]

Encoding energy

\[
A^* = \sigma \left( Z^* \right)
\]

\[
\frac{\partial X^*}{\partial D} = \frac{\partial}{\partial D} \left( D^T A^* + b_D \right)
\]

\[
\frac{\partial L}{\partial X^*} = \frac{\partial}{\partial X^*} \left( \| X - X^* \|_2^2 \right)
\]

Auto-encoder: backprop w.r.t. weights

% Gradient of the loss w.r.t. codes
dL_dz = z_hat - z_star;

% Gradient of the loss w.r.t. reconstruction
dL_dx_star = x_star - x;

% Jacobian of the loss w.r.t. decoder bias
model.dL_dbias_D = dL_dx_star;

% Jacobian of the loss w.r.t. decoder matrix
model.dL_dD = dL_dx_star * a_star';

% Gradient of the loss w.r.t. encoder matrix
model.dL_dC = dL_dz * x';

% Gradient of the loss w.r.t. encoding bias
model.dL_dbias_C = dL_dz;

function model = ...
    Module_Encode_BackProp_Weights(model, ...
    dL_dz, x, params)
    % Jacobian of the loss w.r.t. encoder matrix
    model.dL_dC = dL_dz * x';
    % Gradient of the loss w.r.t. encoding bias
    model.dL_dbias_C = dL_dz;

function model = ...
    Module_Decode_BackProp_Weights(model, ...
    dL_dx_star, a_star, params)
    % Jacobian of the loss w.r.t. decoder matrix
    model.dL_dD = dL_dx_star * a_star';
    % Gradient of the loss w.r.t. decoder bias
    model.dL_dbias_D = dL_dx_star;

Usual tricks about classical SGD

- Regularization (L1-norm or L2-norm) of the parameters?
- Learning rate?
- Learning rate decay?
- Momentum term on the parameters?
- Choice of the learning hyperparameters
  - Cross-validation?
Sparse coding

\[ L_s(Z) \]

\[ L(X,Z;W) = \|X - h(Z;W,b)\|_2^2 + \lambda L_s(Z) \]

Sparse coding

Sparsity constraint
\[ \sum_{d=1}^{M} \log(1 + z_d^2) \]

Overcomplete code
\[ \hat{X} = W^T Z + b \]

Decoding error
\[ \| X - \hat{X} \|_2^2 + \lambda L_s(Z) \]

Limitations of sparse coding

• At runtime, assuming a trained model $W$, inferring the code $Z$ given an input sample $X$ is expensive

• Need a tweak on the model weights $W$: normalize the columns of $W$ to unit length after each learning step

• Otherwise:

$$\hat{X} = W^T Z + b$$

code pulled to 0 by sparsity constraint

weights go to infinity to compensate
Sparse auto-encoder

\[ L_s(Z) \]

\[ E_g(Z, \hat{Z}) \]

\[ \hat{Z} = g(X; W_g, b_g) \]

\[ \hat{X} = h(Z; W_h, b_h) \]

\[ E_h(X, \hat{X}) \]

Code

Code prediction

Sparsity constraint

Code error

Decoding error

Decoding

Input decoding

Input


Symmetric sparse auto-encoder

Encoder matrix $W$ is symmetric to decoder matrix $W^T$

Sparsity constraint

$$\sum_{d=1}^{M} \log \left(1 + \sigma(z_d)^2\right)$$

Code prediction

$$\hat{Z} = WX + b_c$$

Code error

$$\|Z - \hat{Z}\|_2^2$$

Decoding error

$$\|X - \hat{X}\|_2^2$$

Decoding

$$\hat{X} = W^T \sigma(Z) + b_D$$

Input

$$X$$


Once the encoder $g$ is properly trained, the code $Z$ can be directly predicted from input $X$.
Outline

• Deep learning concepts covered
  o Hierarchical representations
  o Sparse and/or distributed representations
  o Supervised vs. unsupervised learning

• Auto-encoder
  o Architecture
  o Inference and learning
  o Sparse coding
  o Sparse auto-encoders

• Illustration: handwritten digits
  o Stacking auto-encoders
  o Learning representations of digits
  o Impact on classification

• Applications to text
  o Semantic hashing
  o Semi-supervised learning
  o Moving away from auto-encoders

• Topics not covered in this talk
Stacking auto-encoders

[Sparsity constraint]

- Code energy
- Code prediction
- Code

- Input decoding
- Decoding energy
- Input

- Z
- X

[Ranzato, Bourou& LeCun, "Sparse Feature Learning for Deep Belief Networks", NIPS, 2007]
MNIST handwritten digits

- Database of 70k handwritten digits
  - Training set: 60k
  - Test set: 10k
- 28 x 28 pixels
- Best performing classifiers:
  - Linear classifier: 12% error
  - Gaussian SVM 1.4% error
  - ConvNets <1% error

[http://yann.lecun.com/exdb/mnist/]
Stacked auto-encoders

Layer 1: Matrix $W_1$ of size $192 \times 784$
192 sparse bases of 28 x 28 pixels

Layer 2: Matrix $W_2$ of size $10 \times 192$
10 sparse bases of 192 units

Sparsity constraint

Code energy

Code prediction

Input

Decoding energy

Input decoding
Our results: bases learned on layer 1
Our results:
back-projecting layer 2
Sparse representations
Training “converges” in one pass over data
Outline

• Deep learning concepts covered
  o **Hierarchical** representations
  o **Sparse** and/or **distributed** representations
  o Supervised vs. **unsupervised** learning

• Auto-encoder
  o Architecture
  o **Inference** and **learning**
  o Sparse coding
  o Sparse auto-encoders

• Illustration: handwritten digits
  o **Stacking** auto-encoders
  o Learning representations of digits
  o Impact on **classification**

• Applications to text
  o Semantic hashing
  o Semi-supervised learning
  o Moving away from auto-encoders

• Topics not covered in this talk
Semantic Hashing

Semi-supervised learning of auto-encoders

- Add classifier module to the codes
- When a input $X(t)$ has a label $Y(t)$, back-propagate the prediction error on $Y(t)$ to the code $Z(t)$
- Stack the encoders
- Train layer-wise

Semi-supervised learning of auto-encoders

2000w-TFIDF logistic regression $F_1=0.83$
2000w-TFIDF SVM $F_1=0.84$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>no dynamics</td>
<td>L1 dynamics</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>0.81</td>
<td>0.86</td>
<td>0.85</td>
</tr>
<tr>
<td>30</td>
<td>0.70</td>
<td>0.86</td>
<td>0.85</td>
</tr>
<tr>
<td>10</td>
<td>0.40</td>
<td>0.86</td>
<td>0.85</td>
</tr>
<tr>
<td>2</td>
<td>0.09</td>
<td>0.54</td>
<td>0.51</td>
</tr>
</tbody>
</table>

Performance on document retrieval task: **Reuters-21k** dataset (9.6k training, 4k test), vocabulary 2k words, 10-class classification

Comparison with:
- unsupervised techniques
  (DBN: Semantic Hashing, LSA) + SVM
- traditional technique: word TF-IDF + SVM

Beyond auto-encoders for web search (MSR)

Compute Cosine similarity between semantic vectors

\[
\cos(s, t_1) \quad \cos(s, t_2)
\]

Semantic vector

\[
d = 300
\]

W₁ \[
d = 500
\]

W₂ \[
d = 500
\]

W₃ \[
d = 500
\]

W₄ \[
d = 500
\]

Letter-tri-gram embedding matrix

\[
\text{dim} = 50K
\]

Letter-tri-gram coeff. matrix (fixed)

Bag-of-words vector

\[
\text{dim} = 5M
\]

Input word/phrase

s: “racing car”

\[
t_1: \text{“formula one”}
\]

\[
t_2: \text{“ford model t”}
\]

Beyond auto-encoders for web search (MSR)

Results on a web ranking task (16k queries)
Normalized discounted cumulative gains

<table>
<thead>
<tr>
<th>#</th>
<th>Models</th>
<th>NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TF-IDF</td>
<td>0.319</td>
<td>0.382</td>
<td>0.462</td>
</tr>
<tr>
<td>2</td>
<td>BM25</td>
<td>0.308</td>
<td>0.373</td>
<td>0.455</td>
</tr>
<tr>
<td>3</td>
<td>WTM</td>
<td>0.332</td>
<td>0.400</td>
<td>0.478</td>
</tr>
<tr>
<td>4</td>
<td>LSA</td>
<td>0.298</td>
<td>0.372</td>
<td>0.455</td>
</tr>
<tr>
<td>5</td>
<td>PLSA</td>
<td>0.295</td>
<td>0.371</td>
<td>0.456</td>
</tr>
<tr>
<td>6</td>
<td>DAE</td>
<td>0.310</td>
<td>0.377</td>
<td>0.459</td>
</tr>
<tr>
<td>7</td>
<td>BLTM-PR</td>
<td>0.337</td>
<td>0.403</td>
<td>0.480</td>
</tr>
<tr>
<td>8</td>
<td>DPM</td>
<td>0.329</td>
<td>0.401</td>
<td>0.479</td>
</tr>
<tr>
<td>9</td>
<td>DNN</td>
<td>0.342</td>
<td>0.410</td>
<td>0.486</td>
</tr>
<tr>
<td>10</td>
<td>L-WH linear</td>
<td>0.357</td>
<td>0.422</td>
<td>0.495</td>
</tr>
<tr>
<td>11</td>
<td>L-WH non-linear</td>
<td>0.357</td>
<td>0.421</td>
<td>0.494</td>
</tr>
<tr>
<td>12</td>
<td>L-WH DNN</td>
<td>0.362</td>
<td>0.425</td>
<td>0.498</td>
</tr>
</tbody>
</table>

Semantic hashing
[Salakhutdinov & Hinton, 2007]

Deep Structured Semantic Model
[Huang, He, Gao et al, 2013]

Outline

• Deep learning concepts covered
  o Hierarchical representations
  o Sparse and/or distributed representations
  o Supervised vs. unsupervised learning
• Auto-encoder
  o Architecture
  o Inference and learning
  o Sparse coding
  o Sparse auto-encoders
• Illustration: handwritten digits
  o Stacking auto-encoders
  o Learning representations of digits
  o Impact on classification
• Applications to text
  o Semantic hashing
  o Semi-supervised learning
  o Moving away from auto-encoders
• Topics not covered in this talk
Topics not covered in this talk

- Other variations of auto-encoders
  - Restricted Boltzmann Machines (work in Geoff Hinton’s lab)
  - Denoising Auto-Encoders (work in Yoshua Bengio’s lab)
- Invariance to shifts in input and feature space
  - Convolutional kernels
  - Sliding windows over input
  - Max-pooling over codes

Thank you!

• Tutorial code: https://github.com/piotrmirowski http://piotrmirowski.wordpress.com

• Contact: piotr.mirowski@computer.org

• Acknowledgements: Marc’Aurelio Ranzato (FB) Yann LeCun (FB/NYU)
Auto-encoders and Expectation-Maximization

Energy of inputs and codes

\[ E(X, Z; W) = \sum_{t=1}^{T} \alpha \| Z_t - g(X_t; C, b_C) \|^2_2 + \sum_{t=1}^{T} \| X_t - h(Z_t; D, b_D) \|^2_2 \]

Input data likelihood

\[ P(X|W) = \int P(X, Z|W) = \frac{\int e^{-\beta E(X, Z; W)}}{\int e^{-\beta E(X, Z; W)}} \]

Do not marginalize over: take maximum likelihood latent code instead

\[ -\log P(X|W) = -\frac{1}{\beta} \log \int e^{-\beta E(X, Z; W)} + \frac{1}{\beta} \log \int e^{-\beta E(X, Z; W)} \]

Maximum A Posteriori: take minimal energy code \( Z \)

\[ -\log P(X|W) = E(X, Z^*; W) + \frac{1}{\beta} \log \int e^{-\beta E(X, Z^*; W)} \]

Enforce sparsity on \( Z \) to constrain \( Z \) and avoid computing partition function
Stochastic gradient descent

Dataset #1

Examples of each class are drawn from a Gaussian distribution centered at (-0.4, -0.8), and (0.4, 0.8).

Eigenvalues of covariance matrix: 0.83 and 0.036

Batch gradient descent

- **Data set:** set-1 (100 examples, 2 gaussians)
- **Network:** 1 linear unit, 2 inputs, 1 output.
  - 2 weights, 1 bias.

**Learning rate:**
\[ \eta = 1.5 \]

**Hessian largest eigenvalue:**
\[ \lambda_{\text{max}} = 0.84 \]

**Maximum admissible Learning rate:**
\[ \eta_{\text{max}} = 2.38 \]
Stochastic gradient descent

Batch gradient descent

data set: set-1 (100 examples, 2 gaussians)
network: 1 linear unit, 2 inputs, 1 output.
2 weights, 1 bias.

Learning rate:
$\eta = 2.5$

Hessian largest eigenvalue:
$\lambda_{\text{max}} = 0.84$

Maximum admissible Learning rate:
$\eta_{\text{max}} = 2.38$

Log MSE (dB)

epochs

Stochastic gradient descent

data set: set-1 (100 examples, 2 gaussians)
network: 1 linear unit, 2 inputs, 1 output.
2 weights, 1 bias.

Learning rate:
$\eta = 0.2$

( equivalent to a batch learning rate of 20 )

Hessian largest eigenvalue:
$\lambda_{\text{max}} = 0.84$

Maximum admissible Learning rate
(for batch):
$\eta_{\text{max}} = 2.38$

Log MSE (dB)

epochs

Dimensionality reduction and invariant mapping

- Similarly labelled samples
- Dissimilar codes