Course 395: Machine Learning - Lectures

Lecture 1-2: Concept Learning (M. Pantic)

Lecture 3-4: Decision Trees & CBC Intro (M. Pantic & S. Petridis)

Lecture 5-6: Evaluating Hypotheses (S. Petridis)

Lecture 7-8: Artificial Neural Networks I (S. Petridis)

➤ Lecture 9-10: Artificial Neural Networks II (S. Petridis)

Lecture 11-12: Artificial Neural Networks III (S. Petridis)

Lecture 13-14: Genetic Algorithms (M. Pantic)

Stochastic Gradient Descent

- Stochastic/Incremental/On-line: One example at a time is fed to the network.
- Weights are updated after each example is presented to the network

Batch Gradient Descent

- Batch: All examples are fed to the network. Weights are updated only after all examples have been presented to the network
- For each weight the corresponding gradient (or Δw) is computed (for each example).
- The weights are updated based on the average gradient over all examples.Type equation here.
- $\Delta w_{allExamples} = \frac{1}{D} \sum_{d=1}^{D} \Delta w_{oneExample}$

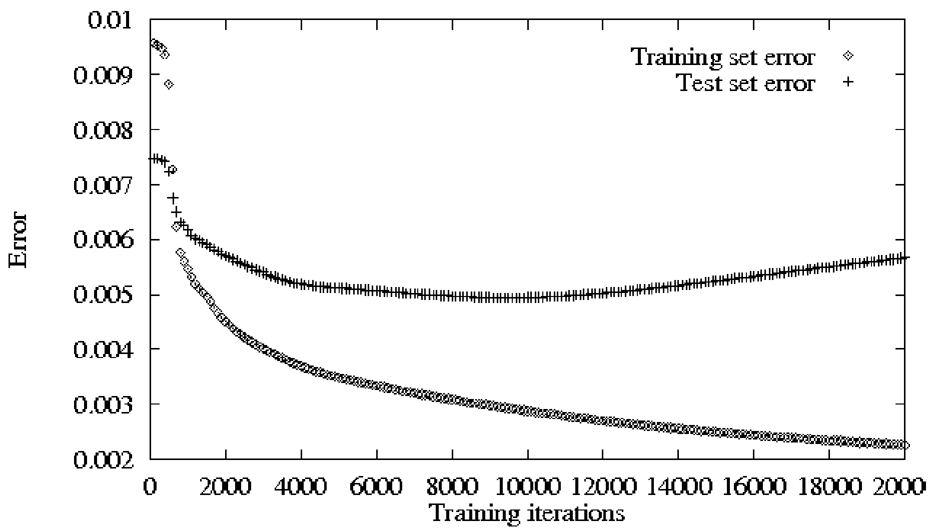
Mini-batch Gradient Descent

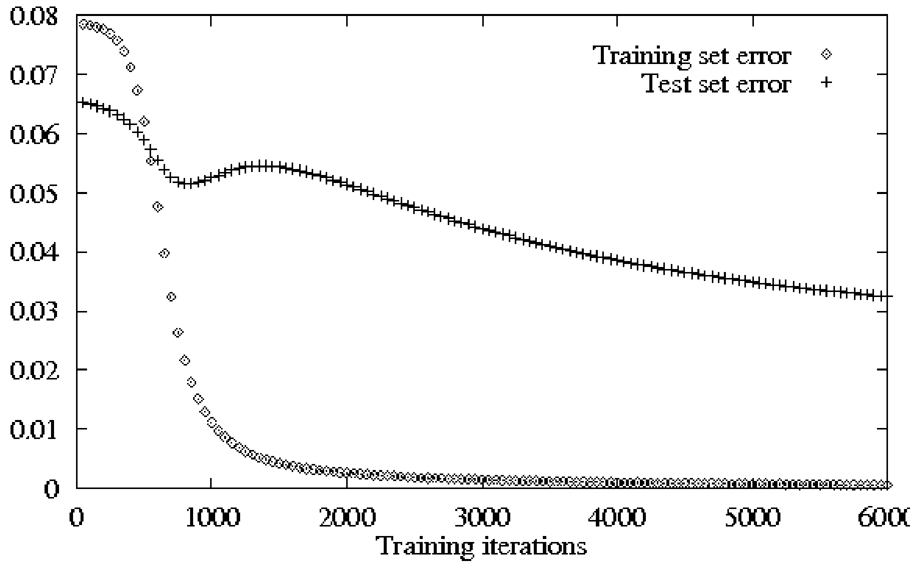
- Mini-Batch: M randomly examples are fed to the network.
  - M = usually 32…128
- For each weight the corresponding gradient (or Δw) is computed (for each example).
- The weights are updated based on the average gradient over all M examples.
- Set of M examples is called mini-batch.
- Popular approach in deep neural networks.
- Sometimes called stochastic gradient descent (NOT to be confused with online/incremental gradient descent).

Backpropagation Stopping Criteria

- When the gradient magnitude (or Δw_i) is small, i.e. $\frac{\partial E}{\partial w_i} < \delta \; or \; \Delta w_i < \delta$
- When the maximum number of epochs has been reached
- When the error on the validation set does not improve for *n* consecutive times (this implies that we monitor the error on the validation set). This is called early stopping.

Early stopping





- Stop when the error in the validation does not impove.
- Error might decrease in the training set but increase in the ‘validation’ set (overfitting!)
- It is also a way to avoid overfitting.

Backpropagation Summary

1. Initialise weights randomly
2. For each input training example *x* compute the outputs (**forward pass**)
3. Compute the output neurons errors and then compute the update rule for output layer weights (**backward pass**)

$$\Delta \boldsymbol{w_{kj}} = -\boldsymbol{\eta} \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial w_{kj}}} = -\boldsymbol{\eta \delta_k y_j} \quad \textit{where} \quad \boldsymbol{\delta_k} = \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial o_k}} \frac{\boldsymbol{\partial \sigma(net_k)}}{\boldsymbol{\partial net_k}}$$

4. Compute hidden neurons errors and then compute the update rule for hidden layer weights **(backward pass)**

$$\Delta \boldsymbol{w_{ji}} = -\boldsymbol{\eta} \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial w_{ji}}} = -\boldsymbol{\eta \delta_j x_i} \; \textit{where} \; \boldsymbol{\delta_j} = \sum_{\boldsymbol{k=1}}^{\boldsymbol{K}} (\boldsymbol{\delta_k w_{kj}}) \frac{\boldsymbol{\partial \sigma(net_j)}}{\boldsymbol{\partial net_j}}$$

Backpropagation Summary

5. Compute the sum of all Δw, once all training examples have been presented to the network

6. Update weights $w_i \leftarrow w_i + \Delta w_i$

7. Repeat steps 2-6 until the stopping criterion is met

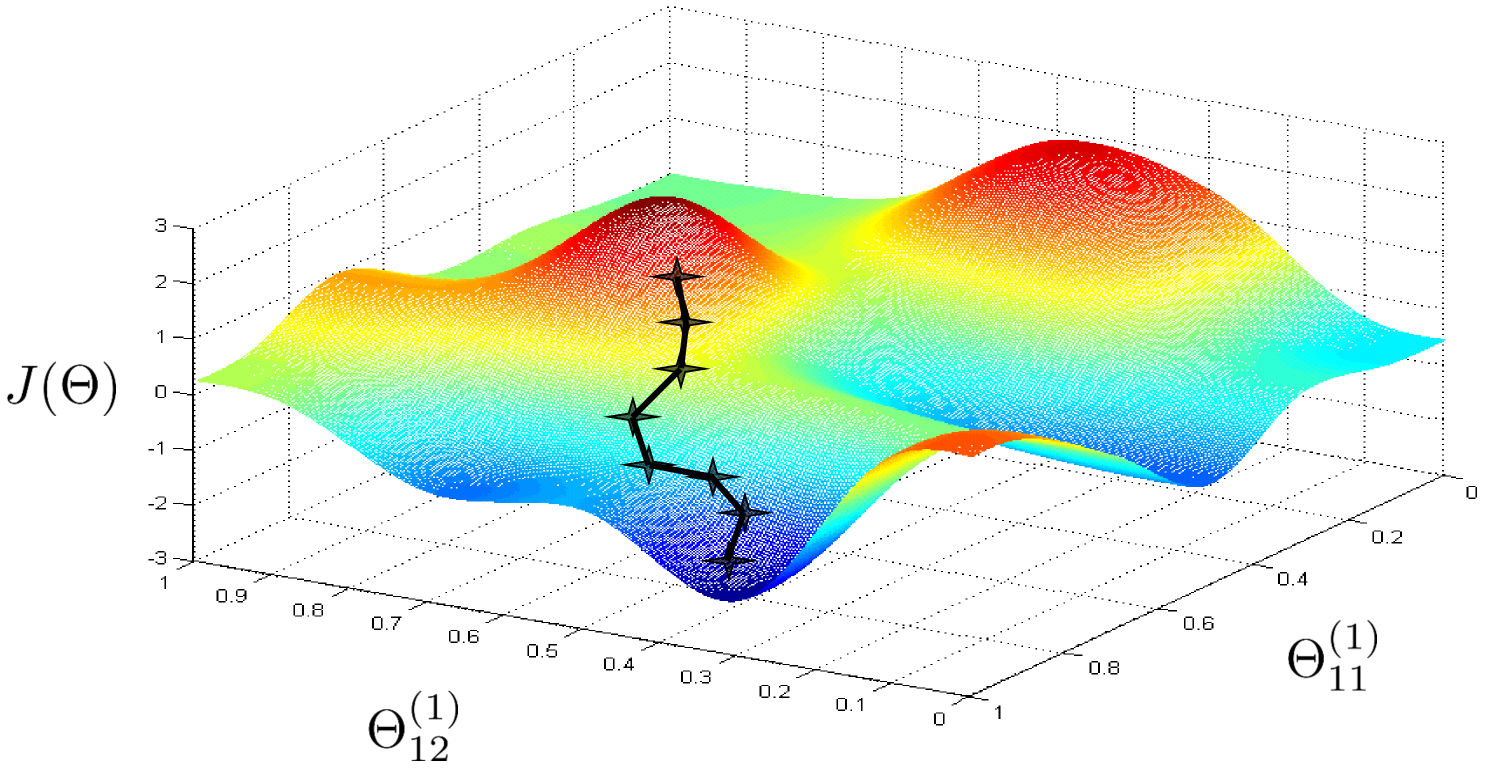
- The algorithm will converge to a weight vector with minimum error, given that the learning rate is sufficiently small

Backpropagation: Convergence

- Converges to a local minimum of the error function
  - … can be retrained a number of times
- Minimises the error over the training examples
  - …will it generalise well over unknown examples?

- Training requires thousands of iterations (slow)
  - … but once trained it can rapidly evaluate output

Backpropagation: Error Surface



Output Weights Update Rule: Example

- Update rule for output units: $\Delta w_{kj} = -\eta \frac{\partial E}{\partial o_k} \frac{\partial \sigma(net_k)}{\partial net_k} y_j$
- Error function $E = \frac{1}{2} \sum_{k=1}^{K} (t_k - o_k)^2$
- $\frac{\partial E}{\partial o_k} = -(t_k - o_k)$
- $\frac{\partial \sigma(net_k)}{\partial net_k} = \sigma(net_k)(1 - \sigma(net_k)\ ) = o_k(1 - o_k)$

  when σ is sigmoid

Output Weights Update Rule: Example

- $\Delta w_{kj} = -\eta \frac{\partial E}{\partial o_k} \frac{\partial \sigma(net_k)}{\partial net_k} y_j = \eta(t_k - o_k)o_k(1 - o_k)y_j$
- When the output is 0 or 1 then Δw is 0 as well
- No matter if our prediction is right or wrong Δw will be 0 if the output is either 0 or 1
- When the output activation function is sigmoid it is not a good idea to use the quadratic error function
- See http://neuralnetworksanddeeplearning.com/chap3.html

Cross Entropy Error as Error Function

- A good error function when the output activation functions are sigmoid is the binary cross entropy defined as follows:

$$E = -\sum_{k=1}^{K} \left(t_k \ln o_k + (1 - t_k) \ln(1 - o_k) \right)$$

- $\Delta w_{kj} = -\eta \frac{\partial E}{\partial o_k} \frac{\partial \sigma(net_k)}{\partial net_k} y_j$
- $\frac{\partial E}{\partial o_k} = \frac{o_k - t_k}{o_k(1 - o_k)}$
- $\frac{\partial \sigma(net_k)}{\partial net_k} = \sigma(net_k)(1 - \sigma(net_k)) = o_k(1 - o_k)$

Cross Entropy Error as Error Function

- $\Delta w_{kj} = -\eta \frac{\partial E}{\partial o_k} \frac{\partial \sigma(net_k)}{\partial net_k} y_j$

- $\Delta w_{kj} = -\eta \frac{o_k - t_k}{o_k(1-o_k)} o_k(1 - o_k) y_j = \eta(t_k - o_k)\, y_j$

- The higher the error the higher the weight update

Softmax output activation functions

- A popular output activation function for classification is softmax $o_k = \frac{e^{net_k}}{\sum_k e^{net_k}}$
- The output can be interpreted as a discrete probability distribution
- The right error function is the negative log likelihood cost E = $-\sum_k t_k ln o_k$
- Target vectors = [0 0 1 … 0] → E = $-ln o_L$ where L is the position of the active target, i.e., it is 1.

Output activation functions: Summary

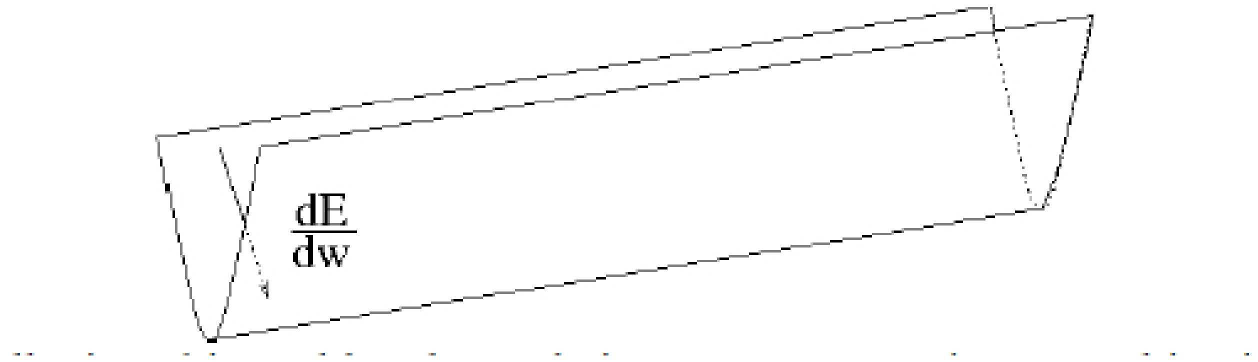
- For each output activation function the right error function should be selected
- Sigmoid → Cross entropy error (useful for classification)
- Softmax → negative log likelihood cost (useful for classification)
- Both combinations work well for classification problems, Softmax has the advantage of producing a discrete probability distribution over the outputs
- Linear → Quadratic loss (useful for regression)

SGD with momentum

- Standard backpropagation

$$w_i \leftarrow w_i + \Delta w_i \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

- If the error surface is a long and narrow valley, gradient descent goes quickly down the valley walls, but very slowly along the valley floor.



From https://www.cs.toronto.edu/~hinton/csc2515/notes/lec6tutorial.pdf

SGD with momentum

- Standard backpropagation

$$w_i \leftarrow w_i + \Delta w_i \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

- Backpropagation with momentum

$$\Delta w_i(t) = \mu\, \Delta w_i(t-1) + (1-\mu)\left(-\eta \frac{\partial E}{\partial w_i(t)}\right) \textbf{ OR}$$

$$\Delta w_i(t) = \mu\, \Delta w_i(t-1) + \left(-\eta \frac{\partial E}{\partial w_i(t)}\right)$$

- $\mu = \text{momentum constant, usually } 0.9, 0.95$

- It is like giving momentum to the weights

- We do not take into account only the local gradient but also recent trends in the error surface

Other Training Algorithms

- Adam (usually works quite well)
- Adagrad
- Adadelta
- RMSprop
- Nesterov momentum
- …and others

Learning Rate Decay

- In the beginning weights are random so we need large weight updates, then as training progresses we need smaller and smaller updates.
- It’s a good idea to start with a “high” (depends on the problem/dataset) learning rate and decay it slowly.
- Typical values for initial learning rate, 0.1, 0.01. It’s problem dependent
- Step decay: Reduce the learning rate by some factor every few epochs, e.g., divide by 2 every 50 epochs

Learning Rate Decay

- Keep learning rate constant for T epochs and then decrease as follows:$lr_t = \frac{lr_0 * T}{\max(t,T)}$
- Keep learning rate constant for T epochs and then decrease as follows: $lr_t = lr_{t-1} * scalingFactor$ (e.g. 0.99)
- Decrease as follows: $lr_t = \frac{lr_0}{1+\frac{t}{T}}$, T is the epoch where the learning rate is halved
- You can think of many other ways to decay the learning rate

Momentum

- It’s usually a good practice to increase the momentum during training.
- Typically the initial value is 0.5 and the final value is 0.9, 0.95
- Increase is usually linear
- It’s also common to start increasing the momentum when the learning rate starts decreasing.

Weight Initialisation

- We said we start with random weights…but how?
- Some of the most common weight initialisation techniques are the following:

1. Sample from a gaussian distribution, we need to define mean (usually 0) and standard deviation (e.g. 0.1 or 0.01)
2. Sample from a uniform distribution, we need to define the range [-b,b]
3. Sparse initialisation: Use gaussian/uniform distributions to initialise weights and then set most of them to 0. You need to define sparsity level, e.g. 0.8 (80% weights in each layer are set to 0).

Weight Initialisation

4. Glorot Initialisation: Sample from a gaussian distribution with 0 mean and st. dev. = $\sqrt{2/(n1 + n2)}$

   - n1, n2 are the number of neurons in the previous and next layers, respectively.

   - Glorot, Bengio, Understanding the difficulty of training deep feedforward neural networks, JMLR, 2010

Weight Initialisation

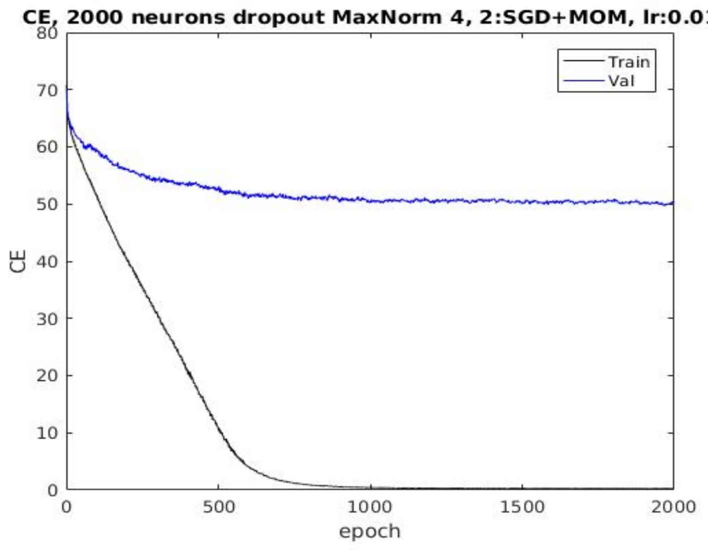
5. He Initialisation: Sample from a gaussian distribution with 0 mean and st. dev. = $\sqrt{2/n1}$
   - n1 is the number of inputs to the neuron (i.e. the size of the previous layer).
   - Designed for neurons which use ReLu as activation functions.
   - He et al., Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, ICCV 2015

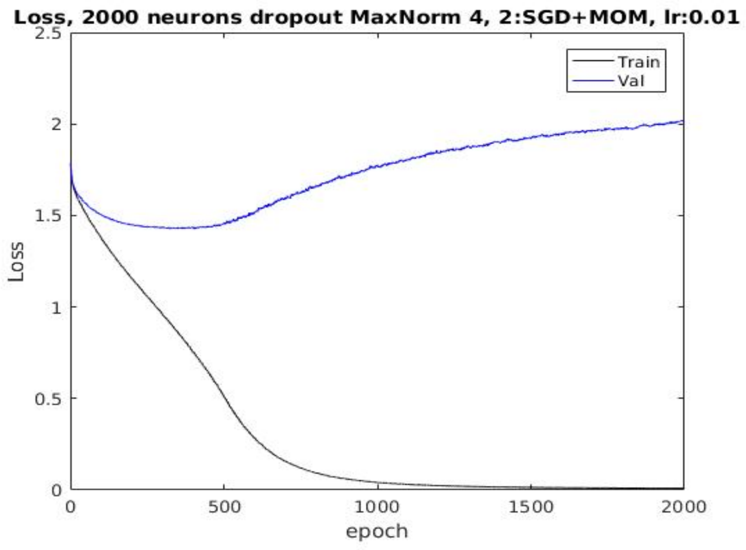
6. You can find many other approaches in the literature

Ways to avoid overfitting

- Early stopping (see slide 6)
- L1 Regularisation
- L2 Regularisation
- Dropout
- Data augmentation

Early Stopping





- Early stopping: should we use loss or Classification error?

- It’s common that classification error can go down while the loss goes up!

L2 Regularisation

- $E = E_0 + 0.5 * \lambda \sum_{all\ Weights} w^2$
- E_0 is the original error function, e.g., quadratic loss, negative log-likelihood
- It is NOT applied to the bias
- We wish to minimise the original error function (E_0)
- We also wish to penalise large weights, keep the weights small (second term)
- Small λ → we prefer to minimise E_0
- Large λ → we prefer small weights

L1 Regularisation

- $E = E_0 + \lambda \sum_{all\ Weights} |w|$
- E_0 is the original error function, e.g., quadratic loss, negative log-likelihood
- It is NOT applied to the bias
- We wish to minimise the original error function (E_0)
- We also wish to penalise large weights, keep the weights small (second term)
- Small λ → we prefer to minimise E_0
- Large λ → we prefer small weights

L1/L2 Regularisation

- So what's the difference between L1 and L2 regularisation?
- L2: $\frac{\partial E}{\partial w} = \frac{\partial E_0}{\partial w} + \lambda w \rightarrow \Delta w = -\eta \frac{\partial E_0}{\partial w} - \eta\lambda w$
- L1: $\frac{\partial E}{\partial w} = \frac{\partial E_0}{\partial w} + \lambda sign(w) \rightarrow \Delta w = -\eta \frac{\partial E_0}{\partial w} - \eta\lambda sign(w)$
- L1: The weights shrink by a constant amount towards 0
- L2: The weights shrink by an amount proportional to w
- L1 drives small weights to zero

L1/L2 Regularisation

- Why small weights prevent overfitting?
- When weights are 0 or close to zero this equivalent to removing the corresponding connection between the neurons
- Simpler architecture → avoids overfitting
- Network has the right capacity
- It is like we start with a high capacity (complex) network until we find a network with the right capacity for the problem