Neural Networks

BP - Practical considerations

Dr. Petr Musilek

Department of Electrical and Computer Engineering
University of Alberta

Fall 2019

Practical Considerations

Preparation of Training Data

- In principle, *raw data* can be used to train networks
- In practice, data *preparation* methods are often used to help appropriate learning
  - Data selection - training data should be representative
  - Rescaling and normalizing data
  - Shuffling data if on–line training is used

More on data preparation later (next section)

Weight Initialization

- Gradient descent treats all weights the same way – if they were all set to 0, no learning would occur
- Generally, weights are initialized at the beginning of learning process to small random values (usually using uniform distribution from interval $[-r, +r]$)
- Value of r should be as large as possible, but such that sigmoid functions are not saturated (i.e. 0.5)
- To make sure that network performance is independent of the initialization, several sets of initial weights should be generated and the best one selected based on observing network performance during learning

Learning Rate η

- If η is too small, learning takes too long (too many incremental weight changes are needed to get close to the minimum of error function)
- If η is too large, the wight updates can over–shoot the minimum and oscillate (or even diverge)

The optimal value of η is problem and network dependent; a range of different values can be used (e.g. $\eta \in \{.001, 0.01, 0.1\}$) and the most suitable value chosen by observing the process.
The value can be changed during the learning process so that it is large at the beginning and keeps getting smaller as learning proceeds (i.e. as the weights are getting close to the minimum)

$$\eta(t) = \frac{\eta(0)}{t} \quad \text{or} \quad \eta(t) = \frac{\eta(0)}{1 + t/\tau}$$

Batch vs. on-line Training

- *Batch training* approximates true gradient descent (weight updates are performed only after all patterns have been presented)
- *On–line training* (or sequential training) performs weight updates immediately after processing each training pattern.

Because there are $n = \texttt{number of patterns}$ updates per epoch in on–line training (as opposed to 1 update per epoch in batch training), it is usually much quicker.
At the same time, the weight changes are rather erratic - this may be advantageous to escape from the vicinity of a local minimum.

Activation (transfer) Function

- Differentiability
- Computational efficiency
    - Sigmoid (logistic) function

$$f^{'}(x) = f(x)\,(1 - f(x))$$

    - Hyperbolic tangent

$$f^{'}(x) = 1 - f(x)^2$$

    - Linear function $f^{'}(x) = 1$
- Hardware implementation (avoid complicated functions)
    - Eliott function

$$f(x) = \frac{x}{1 + |x|}$$

Flat spots in the error function

- Flat spots are caused by saturated portions of sigmoids
- It takes long time to pass through them, and get anywhere close to a minimum
- Sigmoidal activation function has derivative $\rightarrow 0$ as it saturates i.e. if the outputs are totally wrong (0 instead of 1, or vice versa), weight updates are very small
- Solutions:
    - Target off–sets: 0.1 and 0.9 instead of 0 and 1 (no saturation; learning does not get stuck)
    - Sigmoid prime off–set – add a small value (0.1) to the sigmoid derivative so it is no longer 0 even if saturated

Note:

By offsetting the targets the weights will not grow too large in attempt to achieve high precision (exactly 0 or 1)

Local minima in the error function

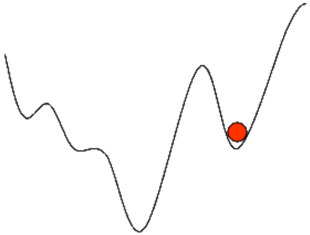
- Error function can easily have more than one minimum
- If learning starts in vicinity of a local minimum, it will likely end in this minimum

Solutions:

- Starting with a range of different initial weight sets increases chance to find the global minimum
- Variation from the true gradient descent (e.g. online learning) also increases chances of getting to the g.

m.

- Other methods (simulated annealing, etc.)



When to stop training

- Sigmoid takes on its extreme values 0 and 1 for $tot = \pm\infty$
- To achieve the binary targets, at least some weights would have to reach ∞, which is impossible in finite time due finite step size
- This is true even for off–set targets 0.1 and 0.9 as the gradient is increasingly getting close to a (flat) minimum
- Solutions:
    - Stop when the approximation is good enough (e.g. ±0.1) from targets; this depends on the problem to be solved
    - Stop when the training error (SSE or MSE) becomes less than a particular small value (again, problem dependent)
    - Use validation data to ensure generalization

Number of hidden units

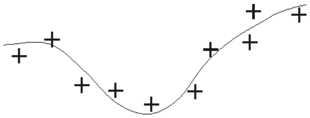
- This depends on many factors
  - Number of training patterns
  - Number of input and output units
  - Amount of noise in training data
  - Complexity of function or classification to be learned
  - Type of hidden unit activation function
  - Training algorithm
- FF NN with a *single layer* can approximate any continuous function - the number of neurons in that layer, however, depends on the degree of nonlinearity
- The smallest network that can perform the desired mapping provides the best generalization

How many hidden units?

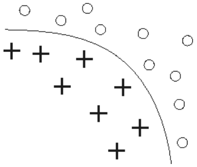
- Too few hidden units – high training and generalization errors (under–fitting)
- Too many hidden units – low training errors but slow learning and poor generalization (over–fitting)
- Solutions (rules of thumb usually do not work)
  - Trying a range of numbers of hidden units and observing the training process to find which works best
  - Network construction methods
    - Pruning: large $\rightarrow$ small
    - Growing: small $\rightarrow$ large
  - Network regularization including weight term $\sum_j w_{ij}$ in the error function causes the network to have smaller weights and this forces network response to be smoother and less likely to overfit

Generalization

Ability to predict function output or class for inputs that the network has not encounter before (during training)



Regression



Classification

Overtraining and specificity

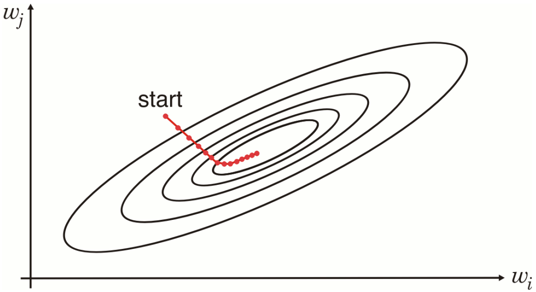
- Breaks generalization: network first learns rough structure of data; as it continues to learn, it will memorize the details (pick up noise)
- Indication that *memorization* occurs $E_v > \overline{E}_v + \sigma_{E_v}$
- Validation error E_v is often calculated on a set of *validation data* - similar to normal training set, but not used in training.
- To reduce chances of memorization - have many more training data points than parameters in the network

Learning rates variation

- Network as a whole learns most efficiently if all neurons learn at roughly the same speed – perhaps different parts of the network should have different learning rates:
    - Later layers (close to outputs) tend to have larger local gradients than layers closer to inputs
    - Activation of units with many connections (both incoming and outgoing) tend to change faster
    - Activations for linear units are different than those for sigmoids
- Approaches:
    - In practice it is usually easier/faster to have the same rates for all units;
    - Evolutionary approaches are powerful for tuning η (as opposed to manual design based on the factors)

Learning visualization

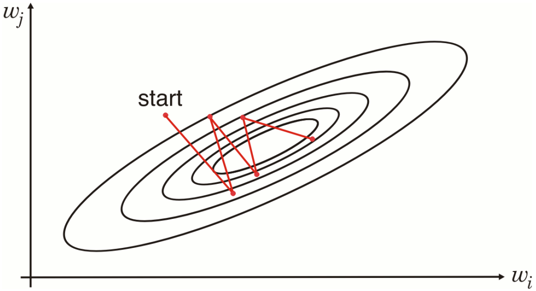
Visualizing learning in 2D to get the idea what is happening



Gradient descent produces a smooth curve perpendicular to the contours; small step η provides good approximation to true gradient descent

Step size too large

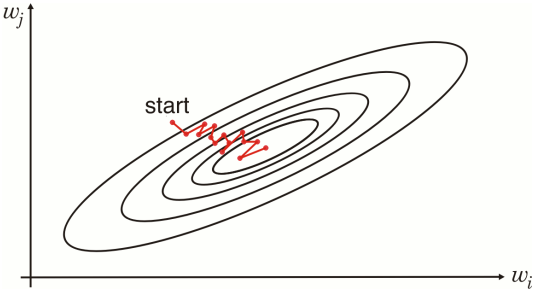
Approximation to true gradient descent is poor and the process ends up with overshoots or even divergence



This is also visible from error graph

Online learning

- Weights are updated after each pattern – no longer to true gradient descent; weight changes are not perpendicular

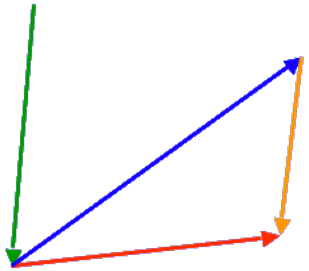


- If η is small enough, the erratic behavior is on a small scale and not much of a problem – increased number of weight changes will lead to the minimum faster than true gradient descent (batch learning)

Learning with momentum

- A compromise to smooth the erratic behaviour of the on-line updates, without significant slow–down
- Weights are updated with the moving average of the individual weight for single training patterns

$$\Delta w_{ij}(t) = -\eta\Delta(w)E(w) + \gamma\Delta w_{ij}(t-1)$$



Data preparation

- This discussion applies to all supervised learning methods
- Preprocessing data assists in convergence and developing an appropriately generalized networks
- Obviously if at all possible irrelevant inputs should be eliminated (training data should be *representative* and *consistent*)
- Preparing a set of training data may include
    1. Handling missing values
    2. Coding of input values
    3. Handling outliers
    4. Scaling and normalization
    5. Noise injection
    6. Manipulation of training set

1. Handling missing values

- eliminate the pattern;
- fill in using statistical methods; or
- add extra network input to indicate “data missing”

2. Coding of input values

- all variables must be numeric;
- enumerated inputs can be converted to unweighted binary inputs

$$\text{red} \in [0,1], \text{green} \in [0,1], \text{blue} \in [0,1]$$

- conversion to a continuous numerical value is possible but the discrete nature of the data is lost

$$\text{red} = 0.0, \text{green} = 0.5, \text{blue} = 1.0$$

3. Handling outliers

- data that deviates from the norm affects error surfaces, and therefore weight updates, drastically
- To handle:
  - remove the outliers before-hand using statistical methods;
  - use an error calculation that minimizes the effects of outliers (robust objective function); and/or
  - remove the outliers during training using, e.g. the frequency distribution of the error

4. Scaling and normalization

Data must be within limits imposed by the activation functions; if close to the limits, weight updates are small and convergence slow (e.g. for sigmoid, use $(0.1, 0.9)$ rather than $(0, 1)$)

- Amplitude (min-max) works but the smaller range (smaller error) leads to longer training times

$$v_s = \frac{v_u - v_u^{\min}}{v_u^{\max} - v_u^{\min}}(v_s^{\max} - v_s^{\min}) + v_s^{\min}$$

- Mean centering shifts the input and output pairs in the training data by their mean value across all patterns. Each individual input and output is scaled $v^{\text{new}} = v^{\text{old}} - \overline{v}$
- Variance scaling is also performed on each input and output individually $v^{\text{new}} = \frac{v^{\text{old}}}{\sigma_v}$ (σ_v is the standard deviation of the input or output across all training patterns)

4. Scaling and normalization (continued)

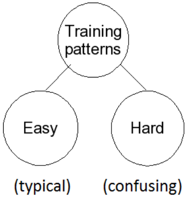
- Normalizing to unit length: some NNs require that the input vectors be of unit length
  - $x^{\text{new}} = \frac{x^{\text{old}}}{\sqrt{\sum x_i^2}}$
  - Problem: vectors in the same “direction” will normalize to the same input vector (magnitude is lost)
- Z-axis normalization preserves magnitude
  - Input values are scaled to $[-1, 1]$ and then normalized by $x_i^{\text{new}} = \frac{x_i^{\text{old}}}{\sqrt{m}}$
  - finally, a synthetic parameter is fed to the network through an extra input unit $x_0 = \sqrt{1 - \frac{D^2}{m}}$, where m is number of inputs and D is the Euclidean length of input vector

5. Injecting noise

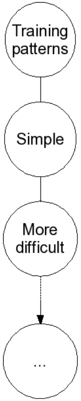
It is possible to generate new tarining patterns by injecting noise into the inputs. The error must be random on a normal distribution with relatively small variance.

6. Manipulating training set

There are several approaches outlining how the training data can be presented to the NN in a bid to speed up learning and increase accuracy.



- Selective presentation (fig L)
- Increased complexity training (fig R)
- Delta subset training (ordered by intra-pattern distance)
- Reduction of the training set (for enormous data sets)
- Randomization


Training patterns
Simple
More difficult
...