Supervised Learning Performance Issues

[Presented by Anna Rakitianskaia]

17 August 2015
Activation functions
Why are some better than others?

Bounded $f(\text{net})$
- How would you scale the inputs?
- How would you scale the outputs?
- Are some functions “better” than others?
- Which works best for you?
Yann LeCun’s activation function
Reference paper: “Efficient Backprop”

LeCun TanH
- Modified hyperbolic tangent
- \( f(\text{net}) = 1.7159 \tanh \left( \frac{2}{3} \text{net} \right) \)
- Softer slope
- Wider activation range

Engelbrecht’s adaptive activation function
Learn the function slope together with the weights

Lambda-Gamma Learning
- Learn the steepness and range of $f(net)$
- $f(net, \lambda, \gamma) = \frac{\gamma}{1 + e^{-\lambda net}}$
- $\lambda$ determines the slope steepness
- $\gamma$ determines the range
- Slope/range is learned $\Rightarrow$ no need for scaling
- Is there a catch?
Engelbrecht’s adaptive activation function
Learn the function slope together with the weights

Lambda-Gamma Learning

- Update the training algorithm:
  - $o_k = f(net, \lambda_{o_k}, \gamma_{o_k})$
  - $\delta_{o_k} = -\frac{\lambda_{o_k}}{\gamma_{o_k}}(t_k - o_k) o_k (\gamma_{o_k} - o_k)$
  - $\lambda_{o_k} = \lambda_{o_k} + \eta_2 \delta_{o_k} \frac{net_{o_k}}{\lambda_{o_k}}$
  - $\gamma_{o_k} = \gamma_{o_k} + \eta_3 (t_k - o_k) \frac{1}{\gamma_{o_k}} o_k$
- Have to choose values for $\eta_2$ and $\eta_3$ in addition to $\eta_1$
Learning Rate and Momentum

- Backpropagation algorithm:
  - \( w(t) = w(t) + \Delta w(t) + \alpha \Delta w(t-1) \)
  - \( \Delta w(t) = \eta \left( - \frac{\delta E}{\delta w(t)} \right) \)

- \( \alpha \) - momentum; controls the influence of past weight changes on the current weight change

- \( \eta \) - learning rate; controls the magnitude of the step size

- How do we choose values for \( \eta \) and \( \alpha \)?
Effect of Learning Rate on Training

Learning Rate

- Backpropagation algorithm:
  \[ w(t) = w(t) + \Delta w(t) + \alpha \Delta w(t - 1) \]
  \[ \Delta w(t) = \eta \left( -\frac{\delta E}{\delta w(t)} \right) \]

- If \( \eta \) is small, step size will be small
  - Search path will closely resemble the gradient path
  - Learning will be slow

- If \( \eta \) is large, step size will be large
  - Might skip over good regions
  - Learning will be fast
Effect of Learning Rate on Training

Learning Rate

Large learning rate: Overshooting.

Small learning rate: Many iterations until convergence and trapping in local minima.
Choosing the Learning Rate

Choosing $\eta$

- Cross-validation: try a selection of values, choose the best-performing one
Choosing the Learning Rate

Choosing $\eta$

- Cross-validation: try a selection of values, choose the best-performing one.
- Start with a small value (0.1), increase if convergence is slow, decrease if oscillation/stagnation is observed.

Plaut et al: $\eta \approx \frac{1}{\text{fanin}}$

Every weight $w_i$ can have its own $\eta_i$.

If direction of change (i.e. sign of $\Delta w_i$) has not changed since previous weight change, increase $\eta_i$ (go faster).

Else, decrease $\eta_i$ (go slower).

Start with large $\eta$ (go fast), decrease $\eta$ over time (go slower as you approach the optimum).

Adaptive $\eta$ - multiple variants proposed.
Choosing the Learning Rate

Choosing $\eta$

- Cross-validation: try a selection of values, choose the best-performing one
- Start with a small value (0.1), increase if convergence is slow, decrease if oscillation/stagnation is observed
- Plaut et al: $\eta \approx \frac{1}{\text{fanin}}$
Choosing the Learning Rate

Choosing $\eta$

- Cross-validation: try a selection of values, choose the best-performing one
- Start with a small value (0.1), increase if convergence is slow, decrease if oscillation/stagnation is observed
- Plaut et al: $\eta \approx \frac{1}{\text{fanin}}$
- Every $w_i$ can have its own $\eta_i$
  - If direction of change (i.e. sign of $\Delta w_i$) has not changed since previous weight change, increase $\eta_i$ (go faster)
  - Else, decrease $\eta_i$ (go slower)
Choosing the Learning Rate

Choosing $\eta$

- Cross-validation: try a selection of values, choose the best-performing one
- Start with a small value (0.1), increase if convergence is slow, decrease if oscillation/stagnation is observed
- Plaut et al: $\eta \approx \frac{1}{fanin}$
- Every $w_i$ can have its own $\eta_i$
  - If direction of change (i.e. sign of $\Delta w_i$) has not changed since previous weight change, increase $\eta_i$ (go faster)
  - Else, decrease $\eta_i$ (go slower)
- Start with large $\eta$ (go fast), decrease $\eta$ over time (go slower as you approach the optimum)
Choosing the Learning Rate

Choosing $\eta$

- Cross-validation: try a selection of values, choose the best-performing one
- Start with a small value (0.1), increase if convergence is slow, decrease if oscillation/stagnation is observed
- Plaut et al: $\eta \approx \frac{1}{\text{fanin}}$
- Every $w_i$ can have its own $\eta_i$
  - If direction of change (i.e. sign of $\Delta w_i$) has not changed since previous weight change, increase $\eta_i$ (go faster)
  - Else, decrease $\eta_i$ (go slower)
- Start with large $\eta$ (go fast), decrease $\eta$ over time (go slower as you approach the optimum)
- Adaptive $\eta$ - multiple variants proposed
Effect of Momentum on Training

Momentum term

- Backpropagation algorithm:
  \[ w(t) = w(t) + \Delta w(t) + \alpha \Delta w(t - 1) \]
  \[ \Delta w(t) = \eta \left( -\frac{\delta E}{\delta \hat{w}(t)} \right) \]
- **Stochastic learning**: adjust weights after each pattern
- **Result**: sign of the error derivatives fluctuates, making the NN unlearn what it has learned in the previous steps
- **Solution**: Batch learning
- **Alternatively**: add momentum to the equation - average the weight changes on the fly, maintain direction
- Larger \( \alpha \) => direction of \( \Delta w(t) \) must be preserved for longer to affect the direction of weight changes
Choosing \( \alpha \)

- Use a static value of 0.9
Choosing the Momentum

Choosing $\alpha$

- Use a static value of 0.9
- Cross-validation: try a selection of values, choose the best-performing one

Quickprop (Fahlman): $\alpha_i(t) = \frac{\delta E \delta w_i(t)}{\delta E \delta w_i(t) - \frac{1}{2} \delta E \delta w_i(t-1)}$

Follow a quadratic approximation of the previous gradient step and the current gradient.

Becker and LeCun calculated $\alpha$ as a function of second order derivatives:

$$\alpha = \frac{\delta^2 E \delta^2 w_i(t)}{2} - 1$$

Numerous adaptive $\alpha$ mechanisms proposed.
Choosing $\alpha$

- Use a static value of 0.9
- Cross-validation: try a selection of values, choose the best-performing one
- Every $w_i$ can have its own $\alpha_i$
  - Quickprop (Fahlman):
    $$\alpha_i(t) = \frac{\delta E}{\delta w_i(t)} \left( \frac{\delta E}{\delta w_i(t-1)} - \frac{\delta E}{\delta w_i(t)} \right)$$
  - Follow a quadratic approximation of the previous gradient step and the current gradient
Choosing the Momentum

Choosing $\alpha$

- Use a static value of 0.9
- Cross-validation: try a selection of values, choose the best-performing one
- Every $w_i$ can have its own $\alpha_i$
  - Quickprop (Fahlman):
    \[ \alpha_i(t) = \frac{\delta E}{\delta w_i(t)} \left( \frac{\delta E}{\delta w_i(t-1)} - \frac{\delta E}{\delta w_i(t)} \right) \]
    - Follow a quadratic approximation of the previous gradient step and the current gradient
- Becker and LeCun calculated $\alpha$ as a function of second order derivatives:
  \[ \alpha = \left( \frac{\delta^2 E}{\delta w_i^2(t)} \right)^{-1} \]
Choosing the Momentum

Choosing $\alpha$

- Use a static value of 0.9
- Cross-validation: try a selection of values, choose the best-performing one
- Every $w_i$ can have its own $\alpha_i$
  - Quickprop (Fahlman):
    \[
    \alpha_i(t) = \frac{\frac{\delta E}{\delta w_i(t)}}{\frac{\delta E}{\delta w_i(t-1)} - \frac{\delta E}{\delta w_i(t)}}
    \]
  - Follow a quadratic approximation of the previous gradient step and the current gradient
- Becker and LeCun calculated $\alpha$ as a function of second order derivatives:
  \[
  \alpha = \left( \frac{\delta^2 E}{\delta w_i^2(t)} \right)^{-1}
  \]
- Numerous adaptive $\alpha$ mechanisms proposed
Training Algorithms

Gradient-based
- Use derivative information
- Error function/activation functions have to be differentiable
- Sensitive to starting point
- Gets stuck in local optima (saddle points)

Population-based (PSO, EA, Simulated Annealing, etc)
- No derivative information used - less informed
- Error function/activation functions can be discontinuous
- Less sensitive to starting point
- Exploses when not optimized: goes out into far ends of the search space and stagnates there
- Adaptive: PSO performed better than BP on dynamic NN problems
Architecture Selection
How many layers? How many neurons? How many weights?

- Occam’s razor: the simplest network is always the best
- Too few neurons: insufficient complexity, poor (underfit) model
- Too many neurons: excessive complexity, poor (overfit) model
- Layers: more can be better (only for complex problems), but is harder to train
- Weights: if the training algorithm is good, it should be capable of setting irrelevant weights to zero
Architecture Selection

What are the options?

- Exhaustive search: try them all
  - Infeasible: For $W$ number of weights, the number of possible architectures is $2^W$
  - Pick a few and do cross-validation
  - Randomly try architectures until a good one is discovered

- Regularisation
  - Add a penalty term to the objective function to minimize complexity
  - Requires a measure of complexity, changes the search space

- Construction (growing) - start with a small architecture, add neurons as necessary

- Pruning - start with an oversized architecture, remove unnecessary neurons
Penalizing complexity

- Add a penalty term to the objective function:
  - \[ E = E_T + \lambda E_C \]
- Now we are minimizing both the error and the complexity
- How do you measure complexity?
Regularisation

Penalizing complexity

- Add a penalty term to the objective function:
  \[ E = E_T + \lambda E_C \]
- Now we are minimizing both the error and the complexity
- How do you measure complexity?
- Weight decay:
  \[ E_C = \sum_{i=1}^{W} w_i^2 \]
  - Minimize weight vector magnitude; only constantly reinforced weights will survive
Regularisation

Penalizing complexity

- Add a penalty term to the objective function:
  \[ E = E_T + \lambda E_C \]
  - Now we are minimizing both the **error** and the **complexity**

- How do you measure complexity?
  - **Weight decay:**
    - \[ E_C = \sum_{i=1}^{W} w_i^2 \]
    - Minimize weight vector magnitude; only constantly reinforced weights will survive

- **Weight elimination:**
  - \[ E_C = \sum_{i=1}^{W} \frac{w_i^2}{1 + \frac{w_i^2}{w_0^2}} \]
  - \( w_0 \) determines the “significance” of weights
  - \( |w_i| >> w_0 \Rightarrow \) high complexity, penalize more
  - \( |w_i| << w_0 \Rightarrow \) low complexity, penalize less
Regularisation

Penalizing complexity

- Laplace: L1 regularisation
  - $E_C = \sum_{i=1}^{W} |w_i|$
  - Contribution of each $w$ to the penalty term increases linearly with the increase of the weight

- Weight decay minimizes entropy $\Rightarrow$ introduce an entropy penalty term (Kamimura and Nakanishi, 1993)

- Multiple other penalty functions were proposed
Regularisation

**Penalizing complexity**

- **Laplace: L1 regularisation**
  - $E_C = \sum_{i=1}^{W} |w_i|$
  - Contribution of each $w$ to the penalty term increases linearly with the increase of the weight

- Weight decay minimizes entropy $\Rightarrow$ introduce an entropy penalty term (Kamimura and Nakanishi, 1993)

- Multiple other penalty functions were proposed

- Consider the objective function:
  - $E = E_T + \lambda E_C$

- How do we choose $\lambda$?
  - Cross-validation
  - Make it adaptive (how?)
Neural Network Construction

Minimalistic approach

- Start with just a few neurons, add more when stagnation occurs
  - The NN contains a working model when a new neuron is added => integrating the new neuron may slow down training
  - How do we decide when to add a neuron, and when to stop growing?

Alternatively: optimise both the weights and the architecture using a genetic algorithm

Evolutionary algorithms were successfully used to "evolve" NN architectures

If you can represent it, you can evolve it

Probably the best "growing" approach

One problem: evolving NNs is a slow process
Neural Network Construction

Minimalistic approach

- Start with just a few neurons, add more when stagnation occurs
  - The NN contains a working model when a new neuron is added => integrating the new neuron may slow down training
  - How do we decide when to add a neuron, and when to stop growing?
- Alternatively: optimise both the weights and the architecture using a genetic algorithm
  - Evolutionary algorithms were successfully used to “evolve” NN architectures
  - If you can represent it, you can evolve it
  - Probably the best “growing” approach
  - One problem: evolving NNs is a slow process
Neural Network Construction
NEAT: NeuroEvolution of Augmenting Topologies
Neural Network Pruning

Applying Occam’s Razor

- Start with just an oversized architecture, remove unnecessary parameters
  - Weights
  - Hidden units
  - Input units
  - Need a way of quantifying relevance of each parameter
- Large architectures have large functional flexibility => a lot of potential for a good fit
- And a lot of potential for an over-fit?
  - Yes, if you do not prune enough
Intuitive pruning

- Determine the “active” neurons, remove inactive ones
  - “An important unit is the one that fires frequently and has strong connections to other units”
  - $G_i = \sum \sum (w_{ij}o_i)^2$ - Goodness factor
  - $E_i = \sum \sum (w_{ij}o_i o_i^{l+1})$ - Consuming energy
- Units that output 0 more often than 1 are considered irrelevant - is it fair?
- Weight magnitude pruning: remove small weights

Evolutionary pruning

- Make different architectures compete for survival
- Assign fitness points to smaller architectures
- Effective, but slow
Neural Network Pruning

Information Matrix pruning

- Fisher information: a way of measuring the amount of information that an observable random variable $X$ carries about an unknown parameter $\theta$ upon which the probability of $X$ depends.
  
  $$ I = \frac{1}{P} \sum_{p=1}^{P} \frac{\delta f_{NN}}{\delta w} \left( \frac{\delta f_{NN}}{\delta w} \right)^T $$

  - information carried by a weight
  - May be time-consuming to compute
  - Makes assumptions: linear $f(net)$ in output units, error is normally distributed

- Singular value decomposition: analyse hidden unit covariance matrix

- Principal Component Analysis: prune parameters (weights) that do not account for data variance

- All of these techniques do not scale very well to large NNs
Neural Network Pruning

Hypothesis Testing

- Use statistical tests to calculate the significance of weights/hidden units
  - Null hypothesis: a subset of weights is equal to zero
  - If weights associated with a neuron are not statistically different from zero, prune the neuron
- Inject a noisy input
  - If the statistical significance of an original parameter is not higher than that of random noise, prune the parameter
- Weights are $\approx$ normally distributed
  - Remove the weights that are in the distribution tails
Neural Network Pruning

Sensitivity analysis pruning

- Saliency: the influence small perturbations to a parameter have on the approximated error/output function
- Prune parameters with low saliency
- Optimal Brain Damage (OBD), introduced by Yann LeCun:
  1. Choose a reasonable NN architecture
  2. Train until a reasonable solution is obtained
  3. Compute second order derivatives $h_{kk}$ for each parameter (diagonal of the Hessian matrix)
  4. Compute the saliencies for each parameter (based on $h_{kk}$)
  5. Sort parameters by saliency and delete low-saliency ones
  6. Go back to step 2
- Optimal Brain Surgeon (OBS) - adjust remaining weights
- Optimal Cell Damage (OCD) - prune inputs
- Hessians are a little expensive to calculate
Neural Network Training
Passive VS Active

- Architecture and training algorithm are important, but so is the data

**Passive learning**
Neural network passively accepts the training data as given to it, and tries to fit it as well as possible

**Active learning**
Neural network is presented with a candidate training set. Heuristics are then used to choose the patterns that are most informative
Active Learning

- Redundant data may be dangerous
- If one class is over-represented, it may bias the NN
- Choosing most informative and relevant patterns:
  - Decrease training time
  - Improve generalisation
- Two main active learning approaches:
  - Selective learning
  - Incremental learning
Selective learning

**Selecting patterns for training**

- Given a candidate set, a subset of informative patterns is chosen as the training set.
- The model is trained until convergence/stopping criteria.
- New cycle starts by selecting a new subset for training.
- Selective Updating:
  - Start training on the candidate set.
  - At each epoch, see which patterns had the most influence on the weights, and discard the patterns that had the least influence.
  - Training set may change from epoch to epoch.
- Discard the patterns that have been classified correctly: this knowledge has already been absorbed.
- Engelbrecht: choose patterns that are close to decision boundaries (sensitivity analysis).
Incremental learning

Training incrementally

- Given a candidate set, a subset of informative patterns is chosen as the training set.
- That subset of patterns is removed from the candidate set.
- The model is trained until convergence/stopping criteria.
- New cycle starts by adding more patterns from the candidate set to the training set.
- As training progresses, the candidate set decreases, and the training set grows.
- Incremental learning does not discard patterns. Rather, it attempts to get the “best” ones first, and uses “weaker” ones to tweak a working model later.
- Eventually, the entire candidate set may be used for training.
Most incremental learning approaches are based on information theory (Fisher information matrix).

Optimal Experiment Design:
- At each iteration, choose a pattern from the candidate set that minimizes the E(MSE) (expected value).
- Expensive: need to calculate the information matrix inverse.

A problem: Fukumizu showed that the Fisher information matrix may be singular:
- It does not have an inverse!
- Same paper: Fisher matrix is singular iff the are redundant units.
- Remove units => solve the problem.
- Very complex and computationally heavy.
Incremental learning

Simpler approaches

- Information gain can be maximized by simply choosing patterns that yield the largest MSE.
- Use Robel’s factor \( \frac{E_G}{E_T} \): when overfitting is observed, add patterns that yield the largest errors.
- Many more methods exist, but all suffer from the following:
  - Overhead of using a heuristic
  - If we use more time to pick patterns than we save on training, was it worth it?
  - The data set should be bad/hard enough to justify these techniques.
Questions?

Next lecture: Unsupervised nets, presented by Will van Heerden