Supervised Learning: Performance Issues

[Part II]

26 February 2020

Training the NN
Accuracy Estimates

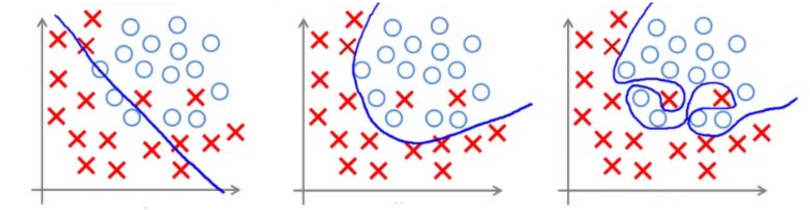
Estimating model accuracy

- NN's error is an estimate of model accuracy: the lower the error, the more accurate the model
- We minimize the error produced by the training set (E_T) to train the NN
- What we actually want to minimize is the generalisation error (E_G)

Overfitting

- Will E_G always go down as E_T decreases?
- No. It is possible to learn the training data too well, preventing meaningful extrapolation/interpolation
- This phenomenon is known as overfitting

Overfitting



Under-fitting

(too simple to explain the variance)

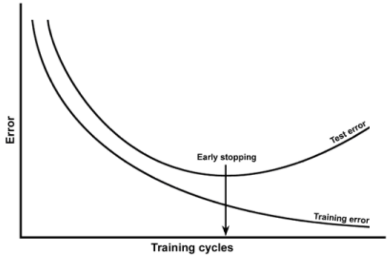
Appropriate-fitting

Over-fitting

(forcefitting -- too good to be true)

Overfitting
Causes and remedies

- If $E_G >> E_T$, your model is probably overfit



- Causes of overfitting:
  - Too many parameters (excessively large NN) [?]
  - Too few data patterns
  - Poor quality data (noise etc.)
  - Training for too long
- Detect overfitting:
- $E_G > \bar{E}_G + \sigma_{E_G}$
- $\rho = \frac{E_G}{E_T}$ (generalisation factor)
- If $\rho > 1$, there might be overfitting

Measuring NN accuracy

How do we calculate E_T, E_G, and E_V?

Mean Squared Error

$$E_T = \frac{\sum_{p=1}^{P_T} \sum_{j=1}^{J} (t_{jp} - y_{jp})^2}{P_T J}$$

$P_T =$ number of patterns in the training set,
$J =$ number of output units

How good is this error metric?

Measuring NN accuracy
How good is Squared Error?

Squared Error for a single Sigmoid unit

$$E_T = \frac{1}{2}(t_j - y_j)^2$$

Generalized Delta Rule applied to Sigmoid output unit

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} = (y_j - t_j)y_j(1 - y_j)y_i$$

Measuring NN accuracy

Single weight update

$\Delta w_{ij} = -\eta(y_j - t_j)y_j(1 - y_j)y_i$

- What will happen if $y_j \approx 0$?
- What will happen if $y_j \approx 1$?
- Weight updates will be very small
- OK if $y_j \approx t_j$
- Not OK if $y_j \approx 0$ and $t_j = 1$, or vice versa
- Weights will learn the fastest when they are somewhat off
- Weights will learn very slowly if they are way off
- This, however, makes no sense!

Measuring NN accuracy
An alternative

Cross-Entropy Error

$$E_T = -\frac{1}{P_T J}\sum_{p=1}^{P_T}\sum_{j=1}^{J}\left(t_{jp}\ln y_{jp} + (1-t_{jp})\ln(1-y_{jp})\right)$$

Output to hidden gradient for Sigmoid for single unit

$$\frac{\partial E}{\partial w_{ij}} = (t_j - y_j)y_i$$

- Dependency on the sigmoid gradient is gone!
- How did we get this function and what does it mean?

Entropy
Some notes

Entropy of a set

- $H(S) = -\sum_{i=1}^{N} p_i \log p_i$
- S is a set of N independent events, each occuring with probability p_i
- $H(S)$ is the entropy of set S (equal probabilities result in highest entropy)

Cross-entropy: Entropy between two distributions

- $H(p, q) = -\sum_{i=1}^{N} p_i \log q_i$
- p is the true distribution of S events, q is the observed distribution
- $H(p, q)$ is the cross entropy between the distributions: how "surprising" is q given p

Measuring NN accuracy
Cross-Entropy Error

Cross-Entropy Error

$$E_T = -\sum_{j=1}^{J} \left(t_j \ln y_j + (1 - t_j) \ln(1 - y_j) \right)$$

Understanding Cross-Entropy

- Targets $t \in \{0, 1\}$ are the “true probabilities”
- Actual outputs $y \in (0, 1)$ (for Sigmoid) are “observed probabilities”
- $t_j \ln y_j$: if $t = 1$, how much does y surprise us?
- $(1 - t_j) \ln(1 - y_j)$: if $t = 0$, how much does y surprise us?
- We use cross-entropy to get a measure of similarity between the two probability distributions

Measuring NN accuracy
Deriving weight update

Cross-Entropy Error

$$E = -t_j \ln y_j - (1 - t_j) \ln(1 - y_j)$$

Generalized Delta Rule applied to Sigmoid output unit

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} y_j(1 - y_j) y_i$$

$$\frac{\partial E}{\partial y_j} = \frac{-t_j}{y_j} + \frac{1 - t_j}{1 - y_j} = \frac{-t_j(1 - y_j)}{y_j(1 - y_j)} + \frac{y_j(1 - t_j)}{y_j(1 - y_j)} = \frac{-t_j + t_j y_j + y_j - y_j t_j}{y_j(1 - y_j)}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} y_j(1 - y_j) y_i = \frac{(y_j - t_j) y_j (1 - y_j) y_i}{y_j(1 - y_j)} = (y_j - t_j) y_i$$

Measuring NN accuracy
CE vs MSE

Cross-Entropy Error

$$E_T = -\frac{1}{P_T J} \sum_{p=1}^{P_T} \sum_{j=1}^{J} \left(t_{jp} \ln y_{jp} + (1 - t_{jp}) \ln(1 - y_{jp}) \right)$$

When should you use it?

- Does cross-entropy make sense for regression?
- No: you can only use it for classification
- Used extensively by the deep learning models
- Punishes “very bad” outputs more than MSE does
- $\therefore$ Tends to give much faster convergence
- It even requires a much smaller learning rate than MSE

More on probabilities

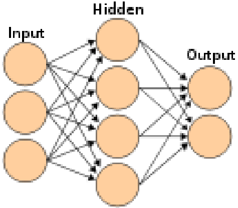
Interpreting the outputs

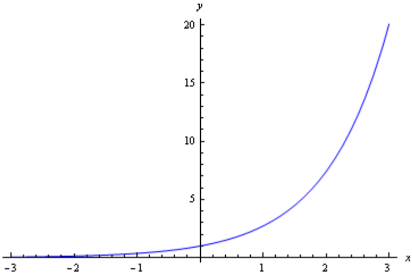
Training set for three classes: chair, table, bed

h	*w*	*chair*	*table*	*bed*
0.5	0.3	1	0	0
0.4	0.4	1	0	0
0.9	1.2	0	1	0
0.8	1.5	0	1	0
0.4	2.0	0	0	1
0.6	1.9	0	0	1

- Suppose your NN's output is $[0.9, 0.9, 0.2]$
- Is it a chair or a table?
- What if the actual output is $[0.9, 0.9, 0.9]$, and the desired output is $[0, 1, 0]$ - did the NN actually learn anything, or is it just saturated?
- Imagine the outputs represented a **probability distribution** instead
- I.e., $o_0 + o_1 + o_2 = 1$
- If at least one output is 0.9, other two must add up to 0.1!

Softmax function
Incorporating probabilities





Softmax activation

$$net_{y_j} = \sum_{i=1}^{I+1} w_{ij} y_i$$

$$y_j = f(net_{y_j}) = \frac{e^{net_{y_j}}}{\sum_k e^{net_{y_k}}}$$

$$\sum_k \frac{e^{net_{y_j}}}{\sum_k e^{net_{y_k}}} = \frac{\sum_k e^{net_{y_k}}}{\sum_k e^{net_{y_k}}} = 1$$

Output signals add up to 1 now, and represent probabilities

Softmax function
Incorporating probabilities

Log-Likelihood Objective Function

Combine Softmax with log-likelihood (multiclass cross-entropy) to get the best of both:

$$E_T = -\frac{1}{P_T J} \sum_{p=1}^{P_T} \sum_{j=1}^{J} (t_{jp} \ln y_{jp})$$

- All output units form a single distribution, thus a single cross-entropy term is used
- Calculating gradients for backpropagation: `https://www.ics.uci.edu/~pjsadows/notes.pdf`

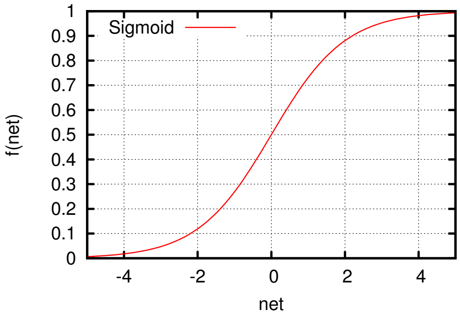
Classification Accuracy

What does the error tell you?

- On a given data set, the final $E_G = 0.13$
- How good is the model?
- MSE or CE error is good for training, but is not easily interpretable
- The goal of classification is to classify each pattern correctly
- Thus, report E_C: classification error
  - Proportion of patterns incorrectly classified
  - A pattern is correctly classified if:

$$\forall j \in J, \begin{cases} y_j \geq 0.5 + \theta & \text{if } t_j = 1 \\ y_j < 0.5 - \theta & \text{if } t_j = 0 \end{cases}$$

Activation functions
Are some better than others?



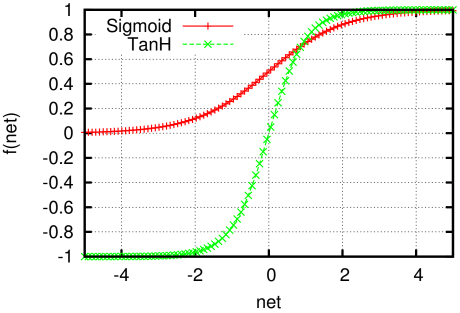
Sigmoid

$$f(net) = \frac{1}{1 + e^{-net}}$$

- Output can be interpreted as "binary"
- Closest to the original step function
- Range $(0, 1)$: the output will always be positive
- Mean output will not be zero
- What happens to the next layer?..

Activation functions

Are some better than others?

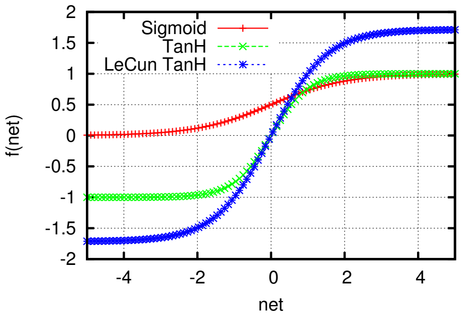


Hyperbolic tangent (tanh)

$$f(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}}$$

- Range $(-1, 1)$
- More likely to have the mean output of zero
- Now the outputs are “centred”
- Less chance of subsequent saturation
- Empirically shown to converge faster
- What about the variance?

Activation functions
"Efficient Backprop", Y. LeCun et al., 1998



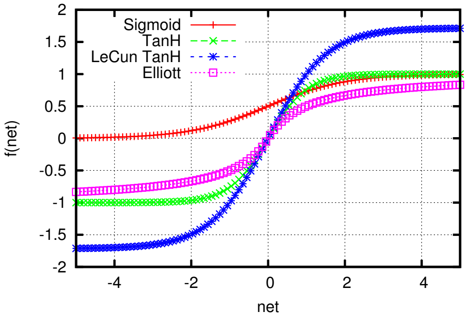
Yann LeCun tanh

$$f(net) = 1.7159 \tanh\left(\frac{2}{3} net\right)$$

- Range $(-1.7159, 1.7159)$
- ???
- The constants were derived by LeCun to ensure that the variance of outputs is 1
- Now the outputs are centred around zero with a variance of one
- I.e., they are standardized!

Activation functions

“A Better Activation Function for Artificial Neural Networks”, D.L. Elliott, 1993



Elliott

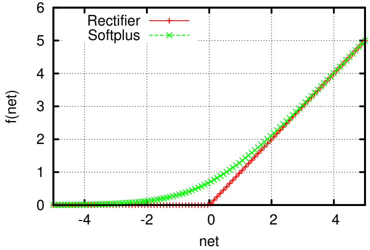
$$f(net) = \frac{net}{1+|net|}$$

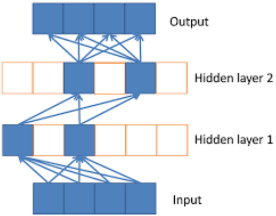
- Range $(-1, 1)$
- Easier to compute than Sigmoid
- Can be scaled to achieve variance of 1
- Softer slope than TanH
  - Slower learning
  - Less saturation

Deep Learning Activation: Rectified Linear Unit

ReLU: Biologically plausible?

"Deep Sparse Rectifier Neural Networks", Glorot et al, 2011





Rectifier

$$f(net) = \max(0, net)$$

Softplus

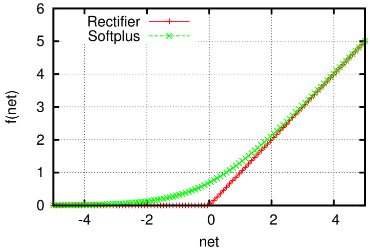
$$f(net) = \log(1 + e^{net})$$

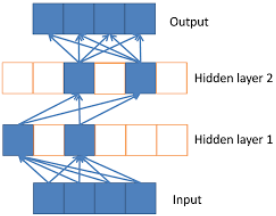
- No gradient when $f(net) = 0$
- Gradient is 1 when $f(net) > 0$
- No saturation for positive inputs!
- Sparse activations ($\approx 50\%$)
- Non-linearity: activation of different "paths"

Deep Learning Activation: Rectified Linear

Biologically plausible?

"Deep Sparse Rectifier Neural Networks", Glorot et al, 2011





Rectifier

$$f(net) = \max(0, net)$$

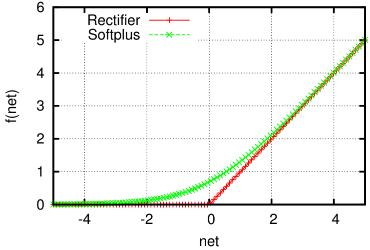
Softplus

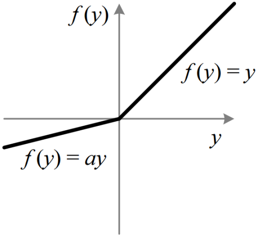
$$f(net) = \log(1 + e^{net})$$

- Problems?
- Non-zero centered
- Unbounded
- Saturated ReLUs "die"

Rectifier

"Delving Deep into Rectifiers: Surpassing Human-Level Performance on Image Net Classification", He et al, 2015





Rectifier

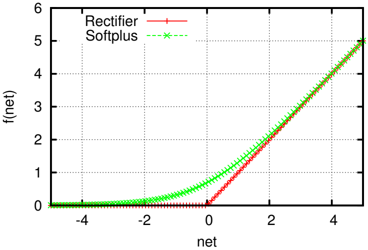
$$f(net) = \max(0, net)$$

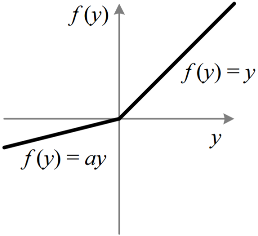
Leaky/Parametrised Rectifier

$$f(net) = \begin{cases} net & \text{if } net > 0 \\ a * net & \text{otherwise} \end{cases}$$

- Original paper: $a = 0.01$
- Parametrised: learn the value of a
- $\Delta a_{i+1} = \alpha \Delta a_i + \eta \frac{\partial E}{\partial a_i}$

Batch Normalisation

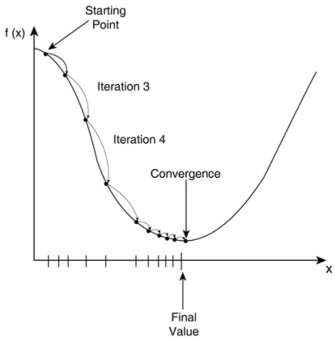




- How do we remedy skew activations?
- "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", Ioffe & Szegedy, 2015
- Cheat the system: standardise activations
- $\hat{f}(net_i) = \frac{f(net_i) - \bar{f}(net)}{\sigma f(net)}$
- $y_i = \gamma \hat{f}(net_i) + \beta$
- Values of γ and β are learned per layer

Backpropagation Parameters

Tuning the most popular NN training algorithm



Learning Rate and Momentum

- Stochastic Backprop:
- $w_t := w_t + \Delta w_t + \alpha \Delta w_{t-1}$
- $\Delta w_t = \eta(-\frac{\partial E}{\partial w_t})$
- α - momentum; controls the influence of past weight changes on the current weight change
- η - learning rate; controls the magnitude of the step size
- How do we choose values for η and α?

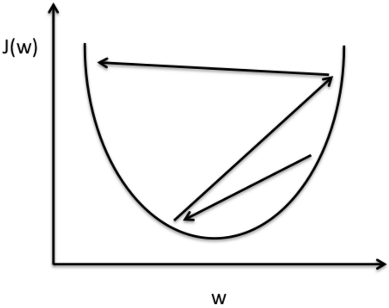
Effect of Learning Rate on Training

Learning Rate

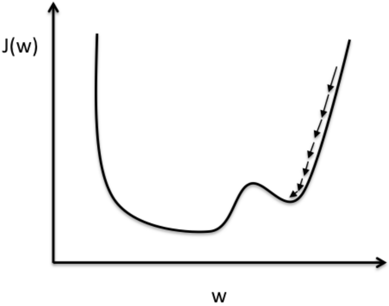
- Stochastic Backpropagation algorithm:
- $w_t = w_t + \Delta w_t + \alpha \Delta w_{t-1}$
- $\Delta w_t = \eta(-\frac{\partial E}{\partial w_t})$
- If η is small, step size will be small
    - Search path will closely resemble the gradient path
    - Learning will be slow
- If η is large, step size 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 η

- 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: $\eta \sim \frac{1}{fanin}$ (more weights => smaller steps)
- Every w_i can have its own η_i
  - If direction of change (i.e. sign of Δw_i) has not changed since previous weight change, increase η_i (go faster)
  - Else, decrease η_i (go slower)
- Decaying η: First explore, then exploit
  - Half-life: divide by 2 every 5 epochs
  - ...Or shrink in some other way every n iterations
  - Larger -> smaller

Effect of Momentum on Training

Momentum term

- Stochastic Backpropagation algorithm:
- $w_t = w_t + \Delta w_t + \alpha \Delta w_{t-1}$
- $\Delta w_t = \eta(-\frac{\partial E}{\partial w_t})$
- Stochastic learning: adjust weights after each pattern
- Result: sign of the error derivative 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 as you go, maintain direction
- Larger α => direction of Δw_t must be preserved for longer to affect the direction of weight changes
- Would this be necessary with batch, mini-batch learning?

Choosing the Momentum

Choosing α

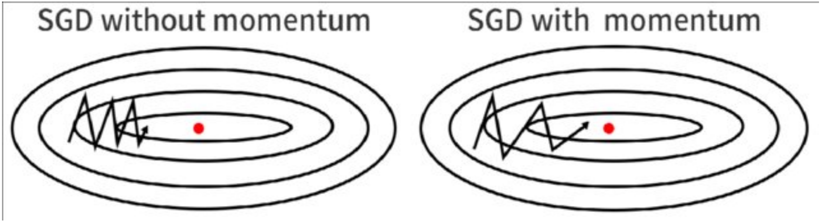
- Use a static value of 0.9 *[note: this is a very bad idea]*
- Cross-validation: try a selection of values, choose the best-performing one
- Adaptive α: start with a smaller value, increase over time
- Every w_i can have its own α_i
  - Why not follow a quadratic approximation of the previous gradient step and the current gradient?
  - Quickprop (Fahlman):
  - $\alpha_i(t) = \frac{\frac{\partial E}{\partial w_i(t)}}{\frac{\partial E}{\partial w_i(t-1)} - \frac{\partial E}{\partial w_i(t)}}$
  - Becker & LeCun (scale each weight by curvature):
  - $\alpha = \left(\frac{\partial^2 E}{\partial w_i^2(t)}\right)^{-1}$ (Becker & LeCun)

Modern Adaptive Methods

http://ruder.io/optimizing-gradient-descent/index.html

Momentum and learning rate are not independent

- Larger momentum allows larger step sizes
- Strategy:
  - Set momentum to as high a value as possible (0.999?)
  - Choose the largest convergent learning rate
- https://distill.pub/2017/momentum/

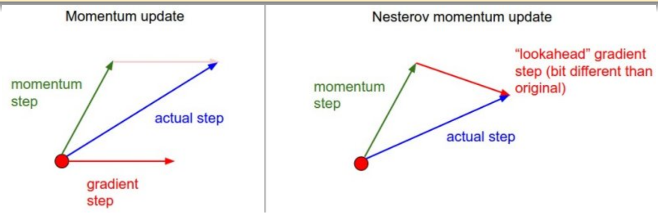


Modern Adaptive Methods

http://ruder.io/optimizing-gradient-descent/index.html

Nesterov accelerated gradient

- Stochastic Backpropagation algorithm:
  - $w_t = w_t - (\Delta w_t + \alpha \Delta w_{t-1})$, $\Delta w_t = \eta(\frac{\partial E}{\partial w_t})$
- Thus, we are moving towards $w_t - \alpha \Delta w_{t-1}$
- Since Δw_{t-1} has already been calculated, why not use it as a better gradient estimate?
- Nesterov accelerated gradient:
  - $w_t = w_t - (\Delta w_t + \alpha \Delta w_{t-1})$, $\Delta w_t = \eta(\frac{\partial E}{\partial (w_t - \Delta w_{t-1})})$



Modern Adaptive Methods

http://ruder.io/optimizing-gradient-descent/index.html

Weight-wise adaptation

- Will a single α and ν value be optimal for all weights?
- No, because the NN loss surfaces are ill-conditioned (high curvature in some dimensions, low curvature in others).
- Result: GD zig-zags instead of taking a straight path.

AdaGrad: adaptive learning rate per weight

- Idea: make rare (sparse) events count more
  - $s_{w_t} = s_{w_{t-1}} + (\nabla w_t)^2$
  - $\Delta w_t = -\frac{\eta}{\sqrt{s_{w_t} + \epsilon}} \nabla w_t$
- High gradients -> smaller update
- Smaller/infrequent gradients -> larger update
- Over the course of training, $\Delta w_t \to 0$

Modern Adaptive Methods

http://ruder.io/optimizing-gradient-descent/index.html

AdaDelta / RMSProp (Resilient mean squared propagation)

- AdaGrad variation: prevent Δw_t from decaying to zero:
  - Exponentially decaying avg: $s_{w_t} = d * s_{w_{t-1}} + (1-d)(\nabla w_t)^2$
  - $d = 0.9$, can be optimised
  - $\Delta w_t = -\frac{\eta}{\sqrt{s_{w_t} + \epsilon}} \nabla w_t$

Adam

- Combine RMSProp with momentum:
  - $m_{w_t} = d_1 * m_{w_{t-1}} + (1-d_2)\nabla w_t$
  - $s_{w_t} = d_2 * s_{w_{t-1}} + (1-d_2)(\nabla w_t)^2$
  - $d_1 = 0.9, d_2 = 0.999$
  - $\Delta w_t = -\frac{\eta}{\sqrt{s_{w_t} + \epsilon}} m_{w_t}$

Neural Network Training

Passive VS Active

- Training algorithm is important, but so is the data
- Data determines the information available to the NN

Passive learning

Neural network passively accepts the training data, and tries to fit the data 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 slow down the training
- If one class is over-represented, it may bias the NN
- Choosing most informative and relevant patterns:
  - Decrease training time
  - Improves generalisation
- Active learning approaches:
  - Selective learning
  - Incremental learning
  - Curriculum 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

Incremental learning

Information theory

- 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 expected value of the error
  - Expensive: need to calculate the information matrix inverse
- A problem: Fukumizu showed that the Fisher information matrix may be singular if redundant units are present

Simpler approaches

- Information gain can be maximized by simply choosing patterns that yield the largest error
- Use Robel's factor ($\frac{E_G}{E_T}$): when overfitting is observed, add patterns that yield the largest errors

Curriculum learning

- Teaching humans often requires a curriculum.
- It is important to start with simpler concepts, and build up to more complex concepts.
- Simple concepts can be explained quickly, while more complex concepts may require more time.
- Can the idea of curriculum learning be leveraged in the NN training context?

Curriculum learning

Defining the NN curriculum

- The "difficulty" (complexity) of the patterns has to be estimated.
  - Patterns that yield a higher error are harder
  - Or: use a non-NN function (prior knowledge) to evaluate hardness
  - Sort patterns from easiest to hardest
- Pacing: determine the sampling strategy
  - Start with easier examples;
  - Otimise the number of epochs spent before increasing difficulty
  - Classification: start with few classes, gradually add more
- Cognitive science supports the notion of curriculum learning, and experiments show that it helps with training and generalisation:
  `https://arxiv.org/abs/1904.03626`

The End

- Questions?
- Assignment 1 is available. Due date: 16 March 2020
- Next lecture: Architecture selection for NNs