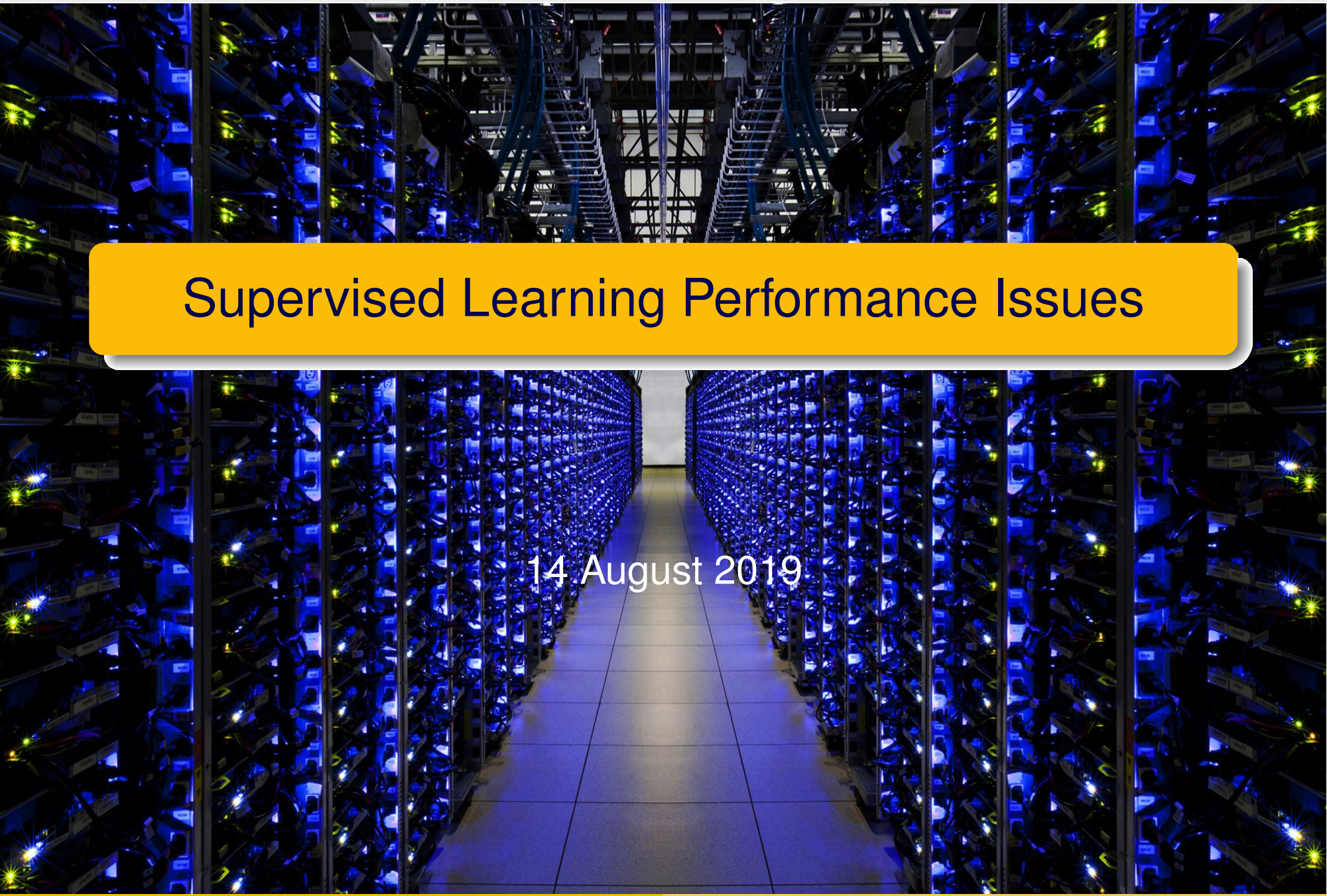
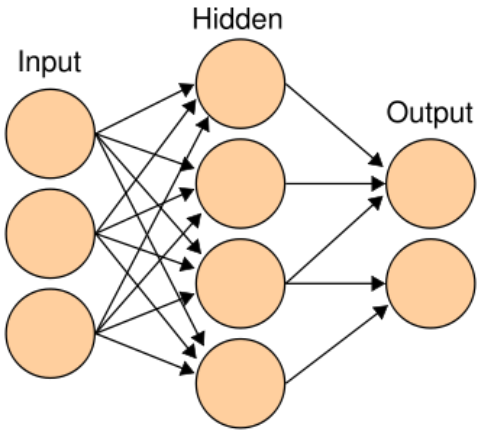


Supervised Learning Performance Issues

14 August 2019

Training Neural Nets

Decisions, decisions...



- How do we...
  - Prepare the data?
  - Initialise the weights?
  - Decide on the architecture?
  - Choose activation functions?
  - Choose a training algorithm?
  - Choose training algorithm parameters?
  - Measure model accuracy?
  - Measure model complexity?
- Trial and error
- "Whatever I am familiar with!"

Data Preparation

- Correctly pre-processing your data takes you half-way to constructing a good model

Representation

- Decide on the inputs and the outputs
- Remove obviously irrelevant inputs (names, unique IDs...)

Missing Values

Solutions?

- Remove the entire pattern if some data is missing
- Replace the missing value with an average for that value over all patterns (continuous) or the most frequently occuring one (discrete)
- Add an extra input unit to indicate if a parameter is missing

Data Preparation

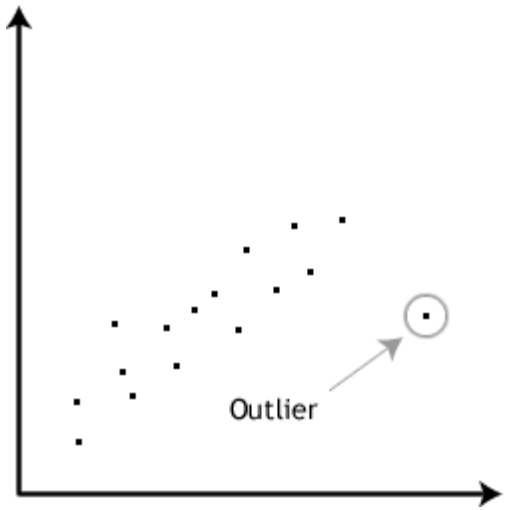
Coding of the Inputs/Outputs

- All inputs/outputs must be numeric
- In case of discrete values/labels, use binary one-hot encoding:
  - A nominal input that takes n different values can be coded as n binary input parameters
  - For each pattern, input corresponding to a specific nominal value is set to 1, the rest are set to 0
- Alternatively, use a single continuous value where every nominal value corresponds to a specific continuous value
  - Discrete characteristic is lost
  - Distance between categories: what does it represent?
  - The representation is more dense (less sparse)
- Which one is better: **dense or sparse**?

Data Preparation

Outliers

Outlier patterns produce large errors and divert the search



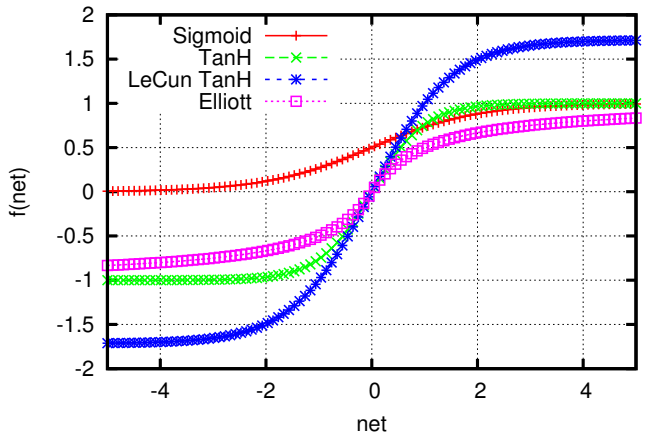
Solutions:

- Remove outliers using statistical techniques
- Use a robust objective function that is not influenced by outliers
  - if $|E| > \epsilon$, set E to a constant value
  - Problems with this approach?
  - if $|E| > \epsilon$, minimise $|E|$ instead of E^2
  - This approach is called “Huber loss” in statistics

Data Preparation
Scaling and Normalization

Scaling Inputs

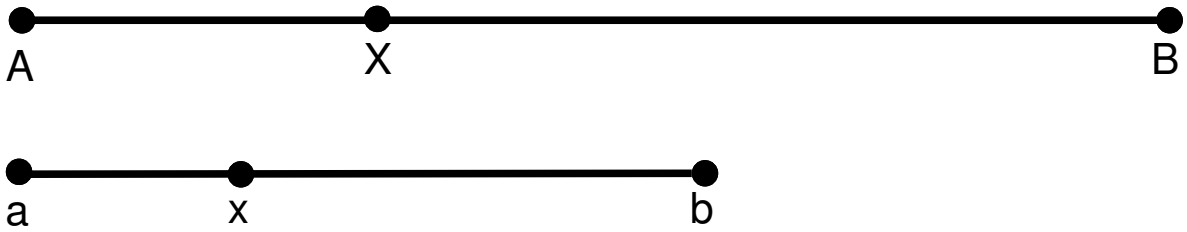
- Inputs outside the active domain of the chosen activation function may cause saturation.
- What is saturation and why is it bad?



Saturation

- Derivatives near assymptotes are close to 0 => slow learning
- A saturated output unit does not indicate the “confidence” level of the NN: all patterns, even the ones not fitted very well by the NN, will be classified with the same “strength”

Data Preparation
Linear (Min-Max) Scaling



- Scale inputs/outputs to the necessary range linearly
- $\frac{X-A}{B-A} = \frac{x-a}{b-a}$
- $x = \frac{X-A}{B-A}(b-a) + a$
- x - scaled, X - unscaled, A, B - unscaled min and max, a, b - scaled min and max
- Obvious disadvantage: you need to know the current range of values before you scale them to the desired range
- What if there is an outlier?

Data Preparation
Scaling and Normalization

Mean Centering: Mean of 0

- Convert the existing distribution to a "gaussian" one
- Average value of variable Z_i for all P: $\bar{Z}_i = \sum_{p=1}^{P} Z_{i,p}/P$
- Scale:
  - $Z_{i,p}^M = Z_{i,p} - \bar{Z}_i$

Variance Scaling: Variance of 1

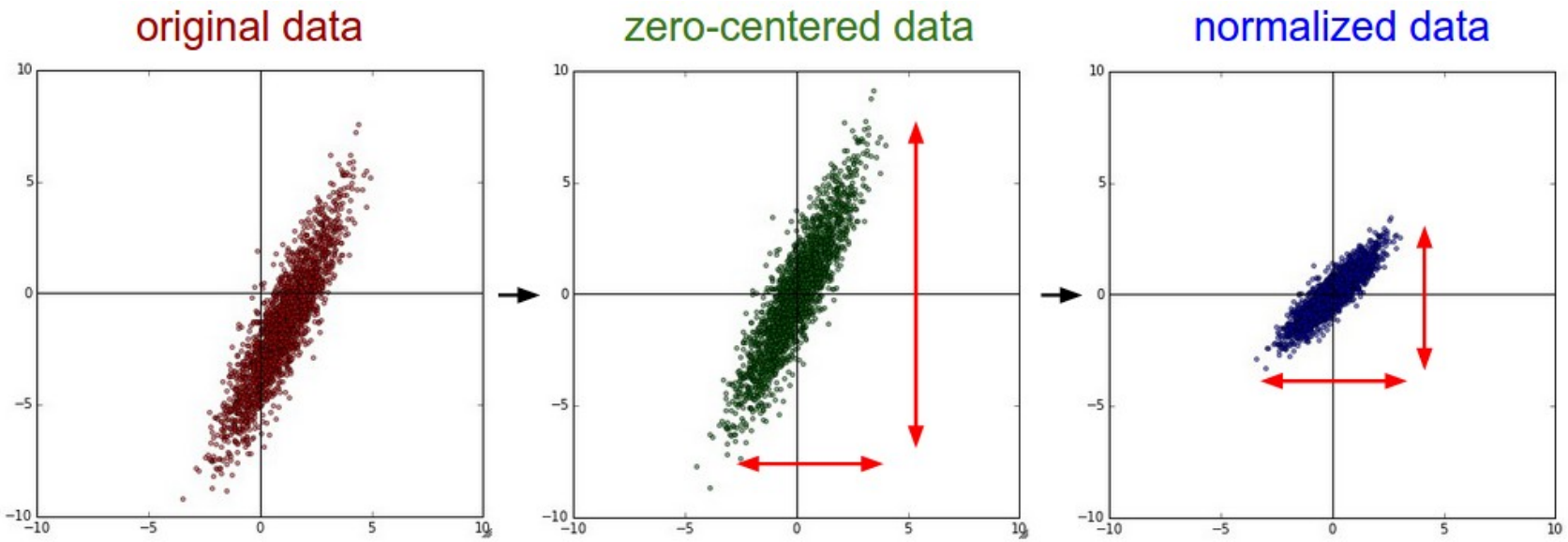
- Let σ_{z_i} be the standard deviations of $Z_{i,p}$. Then:
  - $Z_{i,p}^V = \frac{Z_{i,p}}{\sigma_{z_i}}$

Data Preparation

Combine mean centering and variance scaling

Z-score ("standard score") normalization

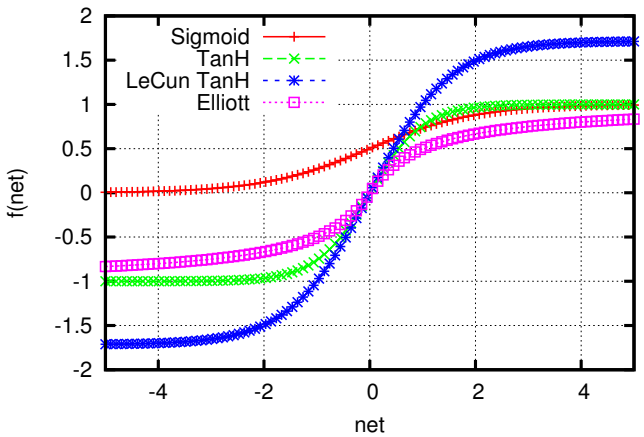
- Combine mean centering and variance scaling to normalize the data:
  - $Z_{i,p}^{MV} = \frac{Z_{i,p} - \bar{Z}_i}{\sigma_{z_i}}$



Data Preparation
Scaling and Normalization

Scaling Outputs

- Outputs outside of the activation function range will be unreachable
- NN will always produce a large error



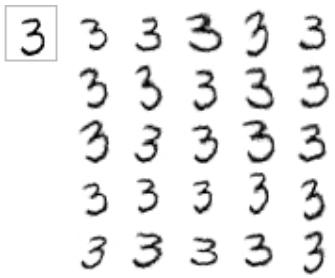
Outputs for bounded functions

- Make outputs reachable: scale to $[0, 1]$ for Sigmoid, $[-1, 1]$ for TanH, etc.
- Trick of the trade: scale to $[0.1, 0.9]$ (Sigmoid) and $[-0.9, 0.9]$ (TanH) instead (why?)

Data Preparation

Do you have enough data?

- Many diverse examples are better than a few similar examples
- The existing data set can always be artificially expanded
  - Add noise sampled from a normal distribution with a small variance and zero mean to the existing patterns
  - Image data: rotate, distort, zoom in and out, etc.
- Adding random noise at every epoch helps prevent overfitting - it becomes harder to memorise the exact patterns, so the NN learns the bigger picture instead
- Deep learning: most record-breaking MNIST-recognizing NNs modified the data set by artificial expansion (rotations, distortions, etc.)
- `http://yann.lecun.com/exdb/mnist/`



Weight Initialisation

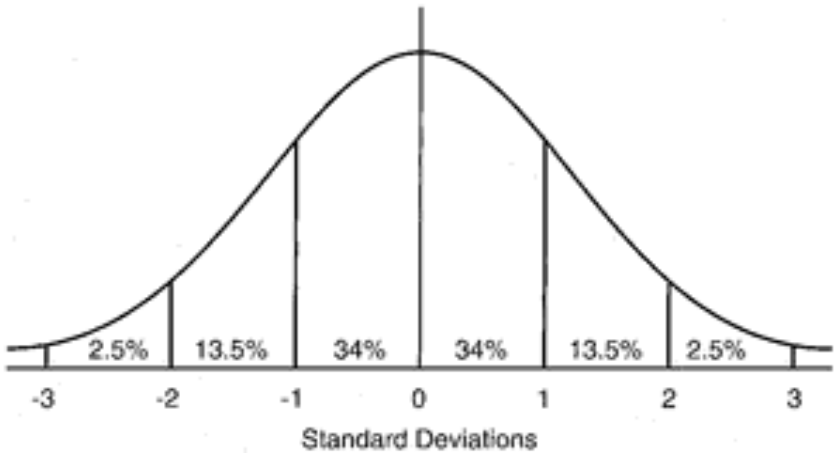
Choosing the starting point

- Gradient descent starts on a single point: choosing a “bad” starting point can be fatal
- Population-based algorithms such as PSO and GA use a population of points: not distributing them adequately can cause stagnation/premature convergence
- Can you set all weights to zero?
- Can you set weights to random numbers?
- Random weights in a small range around 0:
  - Small net input signals
  - Midrange values produced by activation functions
  - I.e. no instant saturation => better learning potential

Weight Initialisation

Choosing the starting point

- Gaussian distribution can be used instead of uniform to sample the weights:



- Wessels and Barnard: choose random weights in range $\left[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}\right]$, where *fanin* is the number of incoming connections for the given unit.
  - The larger the architecture, the smaller interval will be used
- Are smaller weights always better?

Data set
Training, Generalisation, Validation

- Supervised training: data patterns with known target values are presented to the NN
- Data set has to be subdivided into three parts:

Training set

Data that the training algorithm will use to iteratively adjust the weights

Generalisation set

Data that will be used to calculate the generalisation error during hyperparameter optimization

Validation set

Data that will be used to calculate the generalisation error of the optimized model

Data set
Training, Generalisation, Validation

Separating the data

Using training/optimization data for testing is not fair: generalisation error will not be an objective measure

Data normalisation

How would you scale the data in training, testing, validation subsets?

Data Scaling

Which one is correct?

1. Scale the whole data set before splitting:

$$T_i = \frac{T_i - \bar{T}_{all}}{\sigma_{all}}$$

2. Scale training and testing sets separately:

$$T_{i,train} = \frac{T_{i,train} - \bar{T}_{train}}{\sigma_{train}}, \; T_{i,test} = \frac{T_{i,test} - \bar{T}_{test}}{\sigma_{test}}$$

3. Apply training data transformation across the board:

$$T_{i,train} = \frac{T_{i,train} - \bar{T}_{train}}{\sigma_{train}}, \; T_{i,test} = \frac{T_{i,test} - \bar{T}_{train}}{\sigma_{train}}$$

Option (3) is the correct approach.

Data set
Training, Generalisation, Validation

- Training set + generalisation set make up the training set for the optimized model
- How do we subdivide the data set?

Training set

Over 50%, up to 90% of the entire data set

Generalisation set

About 10% to 30% of the training set

Validation set

About 10% to 30% of the data set. Should never be used for training.

Training the NN

Stochastic, batch, mini-batch

Stochastic (on-line) training

- Update weights after each pattern
- Backprop each gradient
- Bound to fluctuate: is a single pattern representative?

Batch training

- Update weights after all patterns were considered
- Backprop average gradient
- Stable, but computationally heavy

Mini-Batch training: Best of both

- Calculate average gradient over a subset of patterns
- Very popular in deep learning

Training the NN
Stochastic, batch, mini-batch

Shuffling the data

- Training set must be shuffled at every epoch (iteration)
- If patterns are presented in the same order, the NN may infer the order of patterns and learn it
- If subsets of the data set are used, the subsets may not be representative unless the data set is shuffled
- How does this apply to stochastic, batch, mini-batch training?

Training the NN
Accuracy Estimates

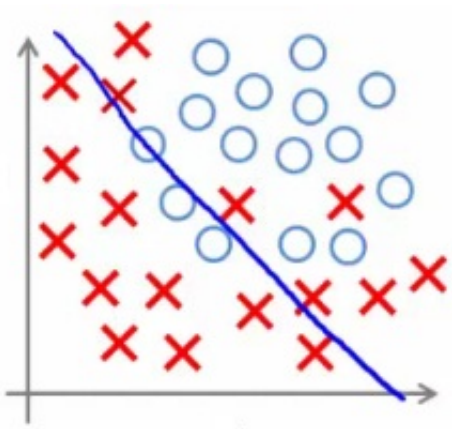
Estimating model accuracy

- 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

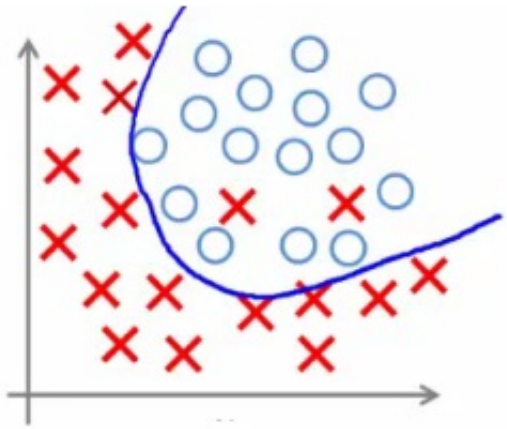
- Will E_G always go down as E_T decreases?
- No. It is possible to learn the training data too well, preventing meaningful extrapolation/interpolation
- This phenomenon is known as overfitting

Overfitting

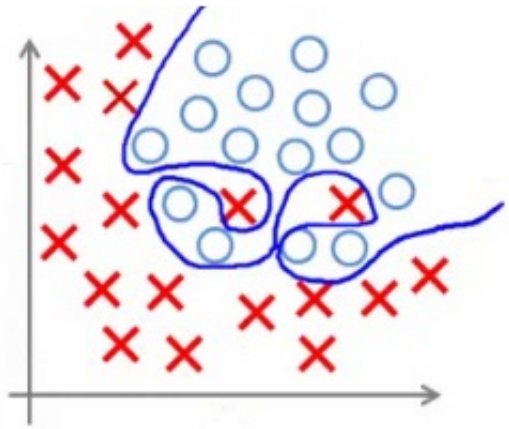


Under-fitting

(too simple to explain the variance)



Appropriate-fitting

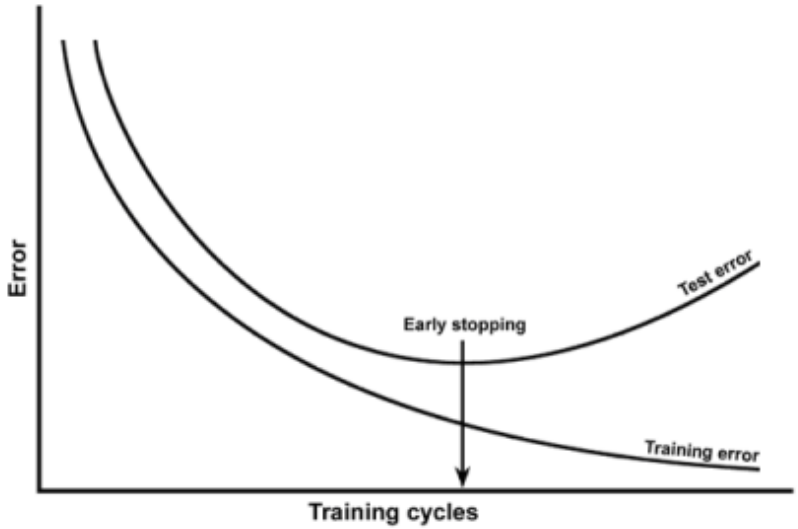


Over-fitting

(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

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

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

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

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

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

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

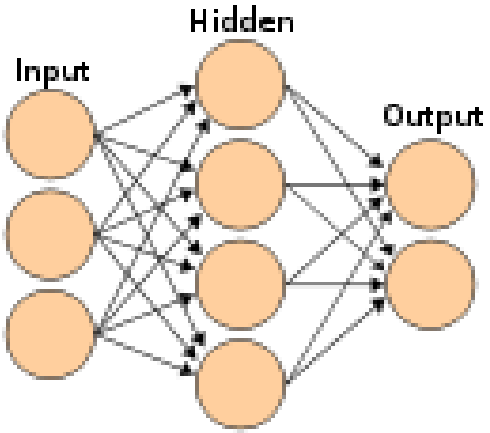
Training set for three classes: chair, table, bed

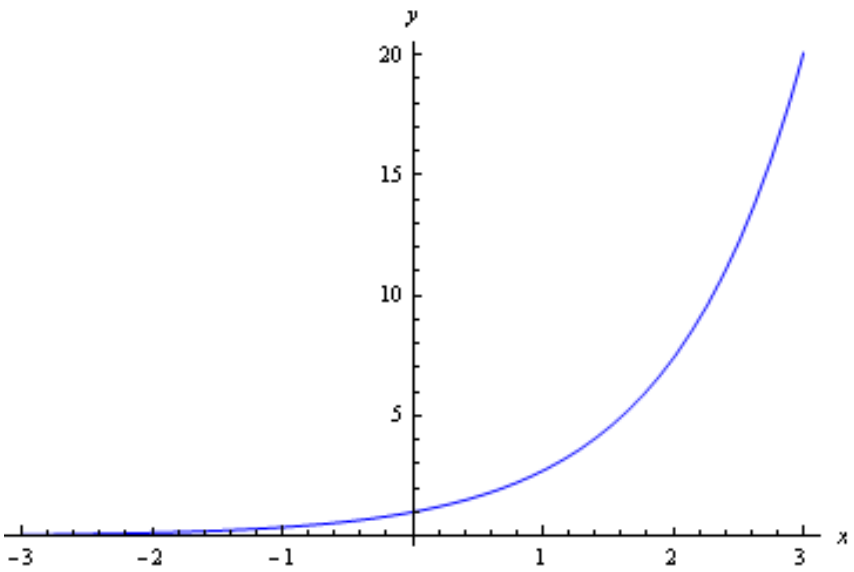
h	*w*	*chair*	*table*	*bed*
0.5	0.3	1	0	0
0.4	0.4	1	0	0
0.9	1.2	0	1	0
0.8	1.5	0	1	0
0.4	2.0	0	0	1
0.6	1.9	0	0	1

- 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 to get the best of both:

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

- 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:

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

How good is your NN?

- Accuracy is important, but there are other characteristics of a NN that should also be considered:
  - Complexity
    - How large is the model?
    - How expensive is it to train/execute?
  - Convergence
    - Can the NN reach a pre-determined error level?
    - Can the NN reach a pre-determined error level within a pre-determined number of epochs?
- How would you present your accuracy measures?
  - Run at least 30 simulations
  - Calculate average E_T, E_G, and any other measures you are using
  - Report as $\bar{E} \pm \sigma_E$

The End

- Questions?
- We will discuss activation functions and learning algorithm hyperparameters in the next lecture