Hyperparameter selection

Yvenn Amara-Ouali

2024

Hyperparameters

Meta-learning

Activation functions

Examples

- Heaviside
- Linear
- Sigmoid
- Tanh
- ReLU
- eLU
- Leaky ReLU
- Parameterised ReLU
- Swish
- Softmax

Output and loss functions

(Multivariate) Regression

- Linear Output: $\hat{\boldsymbol{y}} = \boldsymbol{W}^T \boldsymbol{a} + \boldsymbol{b}$
- Quadratic loss: $\ell(y, y) = \|y - \hat{y}\|_2^2$

Binary Classification

- Binary Output: $\hat{y} = \sigma(\boldsymbol{W}^T \boldsymbol{a} + b)$ or $\hat{y} = \tanh(\boldsymbol{W}^T \boldsymbol{a} + b)$
- Loss: $\ell(y, y) = \log\left(1 + e^{-y\hat{y}}\right)$ for $y \in \{-1,1\}$

K-class Classification

- Softmax Output: $\hat{y} = softmax(\boldsymbol{W}^T \boldsymbol{a} + b)$, with $softmax(z) = \left\{\frac{e^{z_1}}{\sum_{k=1}^K e^{z_k}}, \dots, \frac{e^{z_K}}{\sum_{k=1}^K e^{z_k}}\right\}$
- Loss: $\ell(y, y) = -\sum_{k=1}^K \mathbf{1}(k = y) \log(\hat{y}_k)$, for $y \in \{1, \dots, K\}$

More loss functions

(Multivariate) Regression

- **Mean Squared Error (MSE)**
- **Mean Absolute Error (MAE)**
- **Mean Absolute Percentage Error (MAPE)**
- Huber loss
- Log-cosh loss
- Quantile loss
- Mean Squared Logarithmic Error (MSLE)
- Cosine Similarity loss
- Poisson loss
- Kullback-Leibler divergence loss (KLD)

(K-class) Classification

- **Binary Cross-Entropy Loss**
- **Categorical Cross-Entropy Loss**
- Sparse Categorical Cross-Entropy Loss
- Kullback-Leibler divergence Loss (KLD)
- Hinge Loss
- Squared Hinge Loss
- Multi-class Hinge Loss
- Logistic Loss
- Focal Loss
- Softmax Loss

Weights initialisation

The importance of initial conditions

- Convex problems: convergence is guaranteed regardless of initialisation
- Non-convex problems: initial conditions are paramount

Symmetry

- One important thing is to break symmetry in the initialisation procedure
- For instance, it is usually not advised to set all weights and biases to the same value
- Similarly, small and large random numbers usually lead to dead or saturated neurons

Solutions preserving variance across layers

- Gaussian Initialisation – draw initial parameters from the normal distribution $N(\mu, \sigma^2)$
- Xavier Glorot[1] Initialisation – draw initial parameters from $N(0, \sigma^2)$ where $\sigma = g\sqrt{\frac{2}{n_{in}+n_{out}}}$ with g a hyperparameter to define, n_{in} the number if input units and n_{out} the number of output units. It can also be rewritten with a uniform distribution.
- LeCun's uniform initialisation is very similar

[1] Understanding the difficulty of training deep feedforward neural networks, Glorot X. et al. (2010)

Weights initialisation

General Principle

"We almost always initialize all the weights in the model to values drawn randomly from a Gaussian or uniform distribution. The choice of Gaussian or uniform distribution does not seem to matter very much but has not been exhaustively studied. The **scale** of the **initial distribution**, however, does **have a large effect** on both the **outcome** of the optimization procedure and on the ability of the network to **generalize**."

Goodfellow et al. "Deep Learning„ (2015): Section 8.4.

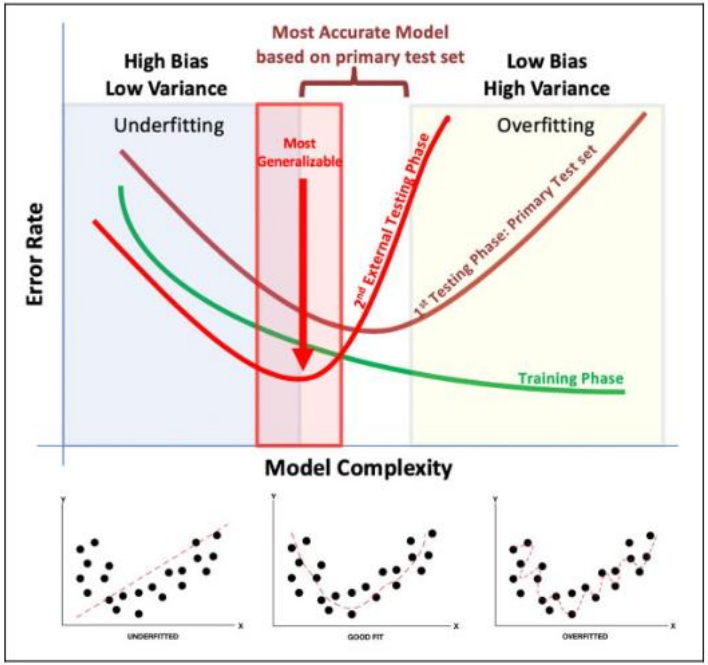
Regularisation

Penalisation

- Replacing $C_n(\theta)$ with $C_n(\theta) + pen(\theta)$

Dropout

- Killing random neurons at each learning iteration

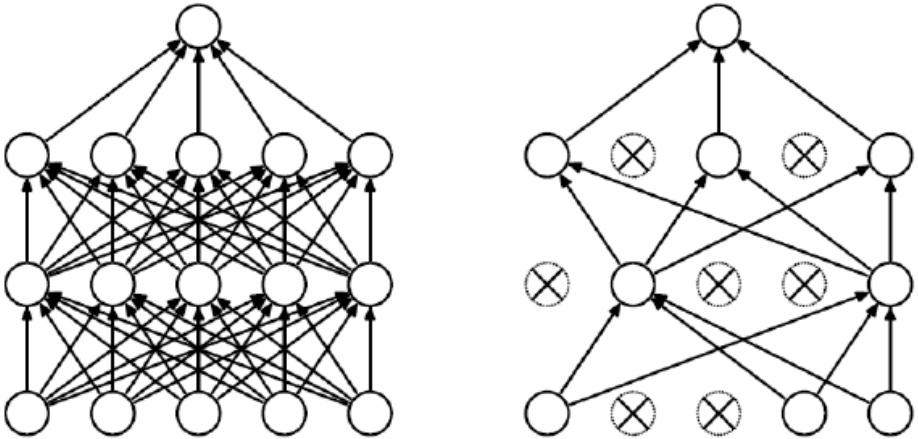


"Artificial Intelligence and Machine Learning in Pathology: The Present Landscape of Supervised Methods", Rashidi et al. (2019)

Dropout

In practice

- Each unit is indepedently retained with a certain probability
- The dropout layer is usually set after the activation layer



(a) Standard Neural Net

(b) After applying dropout.

Dropout

Training

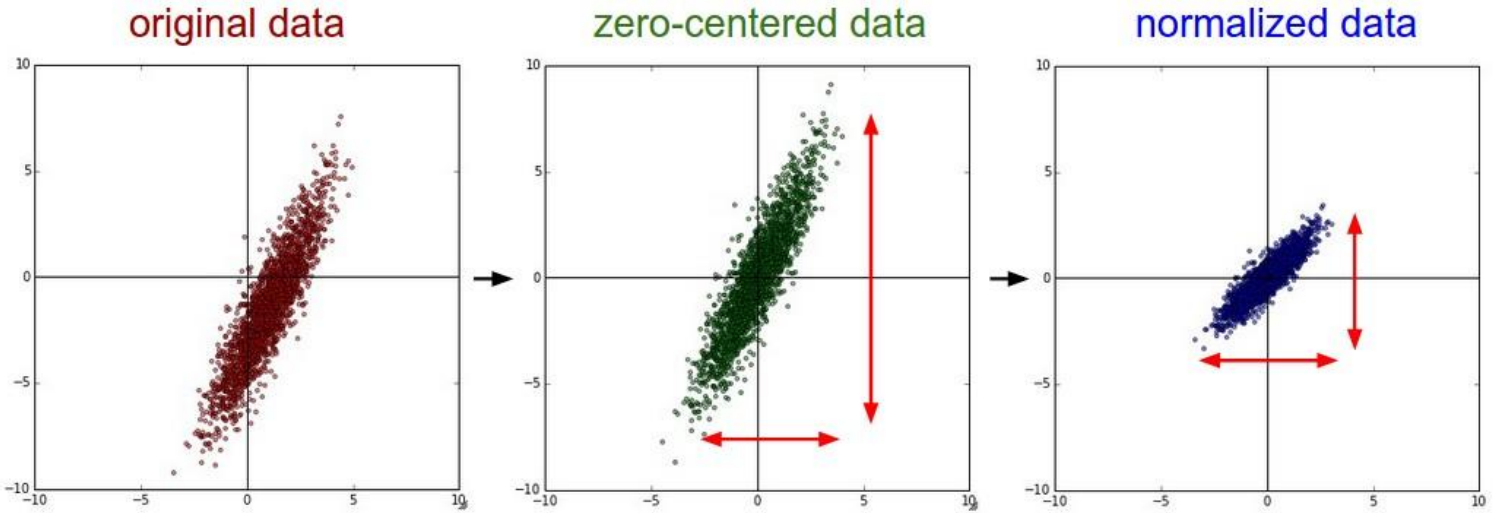
- Inside one epoch, for a mini-batch of size m
- Sample masks of i.i.d. Bernoulli random variables with probability p per node of the network (inner and input nodes but not output nodes).
- For each one of the m samples in the mini-batch:
    1. Do a forward pass on the masked network
    2. Compute the gradients on the masked network
    3. Update weights on the masked network

Prediction

- Use all neurons in the network with the weights learned during training multiplied by the dropout rate used during training $y = x(1 - p)$ with p the dropout rate

Inputs initialisation

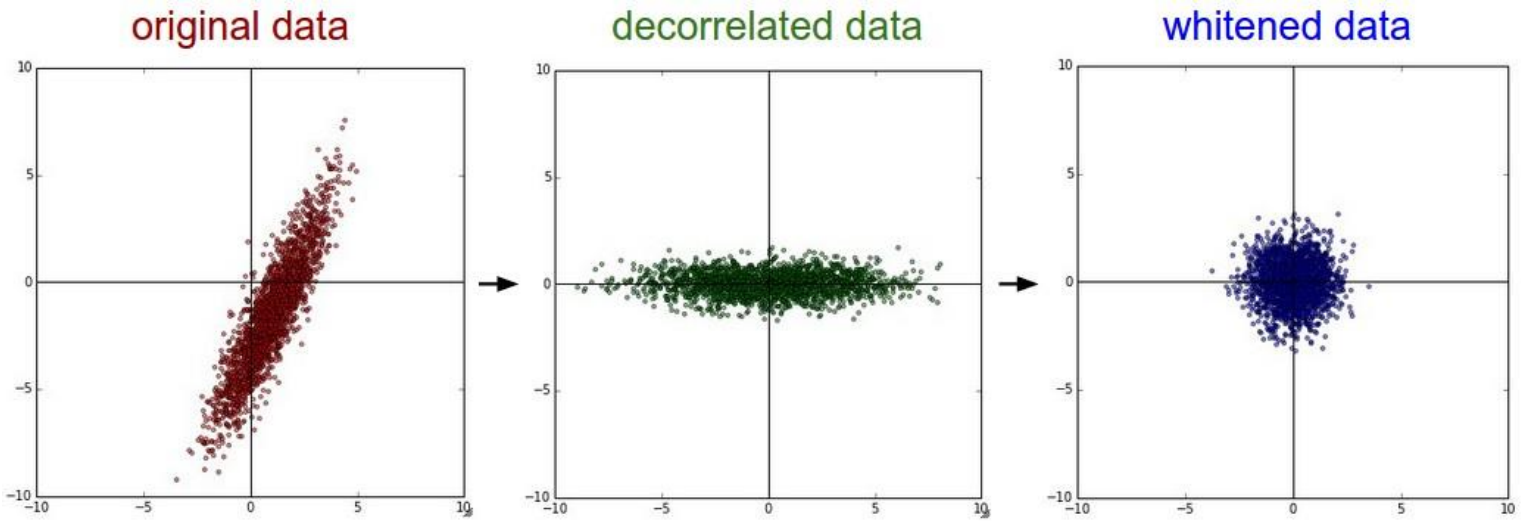
Normalisation



X $\quad X = X - mean(X)$ $\quad X = \frac{X - mean(X)}{std^2(X)}$

Inputs initialisation

PCA



- 2-dimensional input data

- Data centered at zero
- Rotated into the eigenbasis of the data covariance matrix
- Decorrelates the data; the covariance matrix becomes diagonal

- Each dimension scaled by the eigenvalues
- Transforms the data covariance matrix into the identity matrix

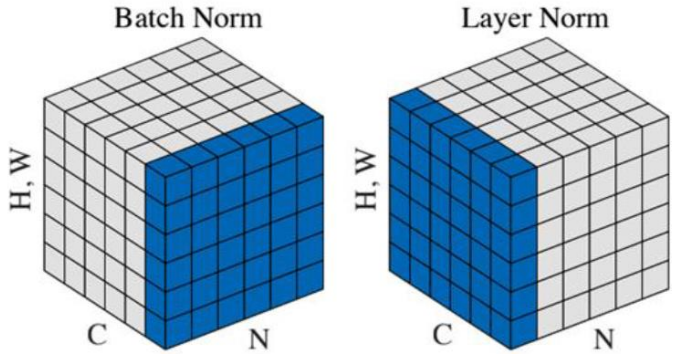
Normalisation

Batch-normalisation

- Networks converge faster if inputs are scaled
- It improves gradient flows and allows for higher learning rates
- Reduces the dependence on initial conditions
- Batch-normalisation scales inputs based on the mean and variance of the mini-batch

Layer-normalisation

- Same principles as batch-normalisation except that the standardisation occurs across a layer
- Particularly used for transformer architectures



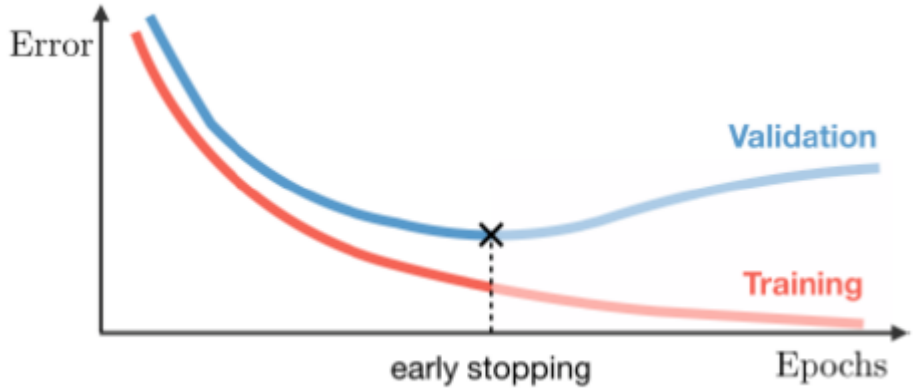
"Across Batch"
Statistics calculated over the whole training set but normalization applied to the current batch for each feature independently

"Across Features"
Statistics calculated for each observation of the current batch across the features

Early Stopping

Principle

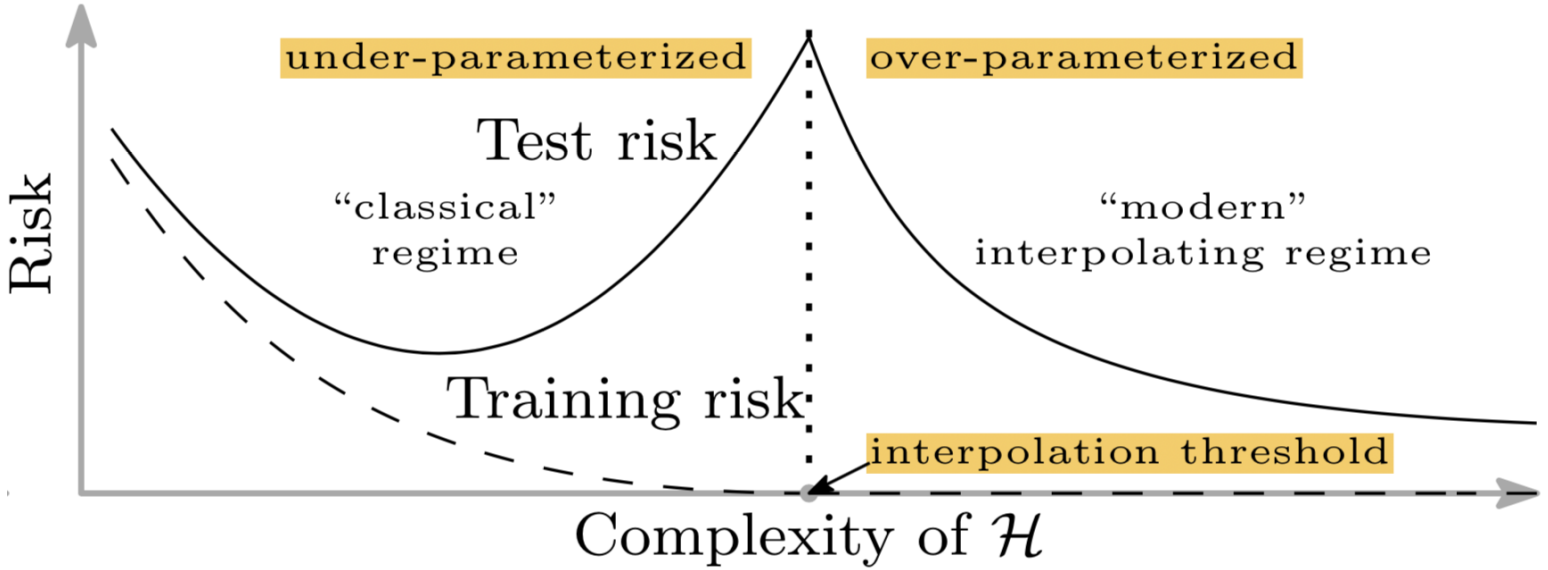
- Essentially, early stopping aims at ensuring the model generalization
- In practice, we stop training when the validation stops improving for a certain number of subsequent epochs



Avoiding overfitting

- **Penalization** methods such as Ridge and L1 are used to replace certain elements in the optimization process.
- **Dropout** is a method where some neurons are randomly shut off during optimization and predictions are made using the entire network.
- **Batch normalization** involves normalizing the layers within a smaller subset of the data, to prevent the network from overfitting on a particular batch.
- **Early stopping** is a method where the optimization process is halted when the validation error fails to decrease over a certain number of iterations.

Final comment



"Reconciling modern machine learning practice and the bias-variance trade-off", Belkin et al. (2018)