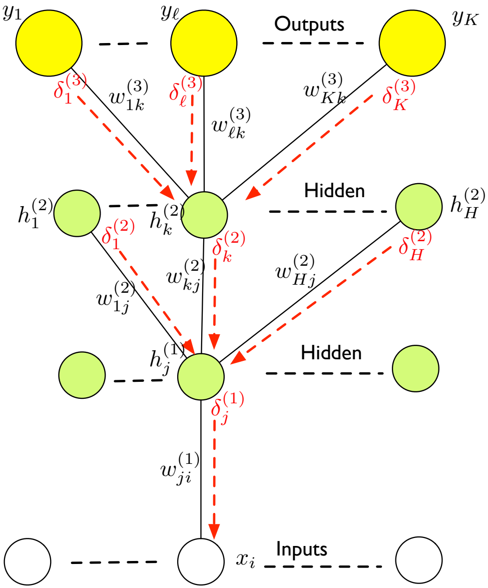
First Coursework & Generalisation

Steve Renals

Machine Learning Practical — MLP Lecture 4
14 October 2015

Recap: Training multi-layer networks



Coursework 1 – Training multi-layer networks to classify MNIST digits

Building on the lab example in which single layer networks are trained on MNIST:

- Task 1 Implement a `Sigmoid` layer (by extending the `Linear` layer class)
- Task 2 Implement a `Softmax` layer (by extending the `Linear` layer class)
- Task 3 Train a one-hidden-layer network and reporting classification results, exploring the effect of learning rates, and plotting Hinton Diagrams for the hidden units and output units.
- Task 4 Experiment with different numbers of hidden layers.

Coursework 1 – Training multi-layer networks to classify MNIST digits

Building on the lab example in which single layer networks are trained on MNIST:

- Task 1 Implement a `Sigmoid` layer (by extending the `Linear` layer class)
- Task 2 Implement a `Softmax` layer (by extending the `Linear` layer class)
- Task 3 Train a one-hidden-layer network and reporting classification results, exploring the effect of learning rates, and plotting Hinton Diagrams for the hidden units and output units.
- Task 4 Experiment with different numbers of hidden layers.

Any Questions?

Generalising to New Data

Generalization

- How many hidden units (or, how many weights) do we need?

Generalization

- How many hidden units (or, how many weights) do we need?
- How many hidden layers do we need?

Generalization

- How many hidden units (or, how many weights) do we need?
- How many hidden layers do we need?
- Generalization: what is the expected error on a test set?

Generalization

- How many hidden units (or, how many weights) do we need?
- How many hidden layers do we need?
- Generalization: what is the expected error on a test set?
- Causes of error
  - Network too "flexible": Too many weights compared with number of training examples
  - Network not flexible enough: Not enough weights (hidden units) to represent the desired mapping

When comparing models, it can be helpful to compare systems with the same number of *trainable parameters* (i.e. the number of trainable weights in a neural network)

Generalization

- How many hidden units (or, how many weights) do we need?
- How many hidden layers do we need?
- Generalization: what is the expected error on a test set?
- Causes of error
  - Network too "flexible": Too many weights compared with number of training examples
  - Network not flexible enough: Not enough weights (hidden units) to represent the desired mapping

  When comparing models, it can be helpful to compare systems with the same number of *trainable parameters* (i.e. the number of trainable weights in a neural network)
- Optimizing training set performance does not necessarily optimize test set performance....

Training / Test / Validation Data

- Partitioning the data...
  - **Training** data – used in as labelled data when training the network
  - **Validation** data – frequently used to measure the error of a network on "unseen" data (e.g. after each epoch)
  - **Test** data – less frequently used "unseen" data, ideally only used once
- Frequent use of the same test data can indirectly "tune" the network to that data (e.g. by influencing choice of *hyperparameters* such as learning rate, number of hidden units, number of layers, ...)

Measuring generalisation

- Generalization Error – The predicted error on unseen data. How can the generalization error be estimated?
  - Training error?

$$E_{\text{train}} = - \sum_{\text{training set}} \sum_{k=1}^{K} t_k^n \ln y_k^n$$

  - Validation error?

$$E_{\text{val}} = - \sum_{\text{validation set}} \sum_{k=1}^{K} t_k^n \ln y_k^n$$

Cross-validation

- Optimize network performance given a fixed training set

Cross-validation

- Optimize network performance given a fixed training set
- *Hold out* a set of data (validation set) and predict generalization performance on this set
  1. Train network in usual way on training data
  2. Estimate performance of network on validation set

Cross-validation

- Optimize network performance given a fixed training set
- *Hold out* a set of data (validation set) and predict generalization performance on this set
  1. Train network in usual way on training data
  2. Estimate performance of network on validation set
- If several networks trained on the same data, choose the one that performs best on the validation set (**not** the training set)

Cross-validation

- Optimize network performance given a fixed training set
- *Hold out* a set of data (validation set) and predict generalization performance on this set
  1. Train network in usual way on training data
  2. Estimate performance of network on validation set
- If several networks trained on the same data, choose the one that performs best on the validation set (**not** the training set)
- *n-fold* Cross-validation: divide the data into n partitions; select each partition in turn to be the validation set, and train on the remaining $(n-1)$ partitions. Estimate generalization error by averaging over all validation sets.

Overtraining

- Overtraining corresponds to a network function too closely fit to the training set (too much flexibility)
- Undertraining corresponds to a network function not well fit to the training set (too little flexibility)
- Solutions
  - If possible increasing both network complexity in line with the training set size
  - Use prior information to constrain the network function
  - Control the flexibility: **Structural Stabilization**
  - Control the *effective flexibility*: **early stopping** and **regularization**

Structural Stabilization

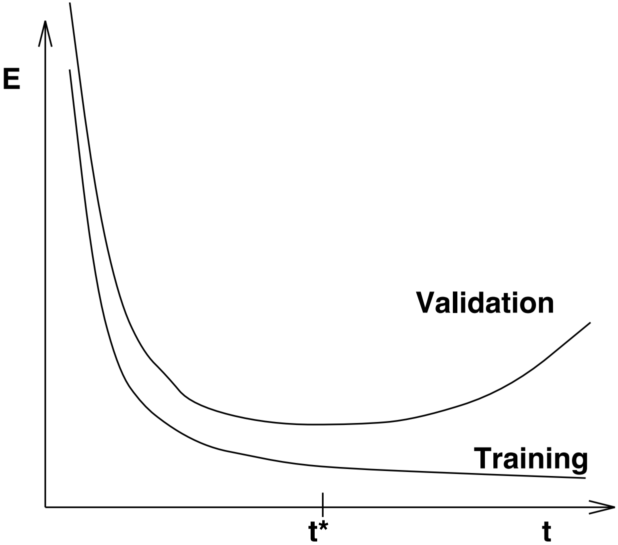
Directly control the number of weights:

- Compare models with different numbers of hidden units
- Start with a large network and reduce the number of weights by pruning individual weights or hidden units
- Weight sharing — use prior knowledge to constrain the weights on a set of connections to be equal.
  $\rightarrow$ Convolutional Neural Networks

Early Stopping

- Use validation set to decide when to stop training
- Training Set Error monotonically decreases as training progresses
- Validation Set Error will reach a minimum then start to increase

Early Stopping



Early Stopping

- Use validation set to decide when to stop training
- Training Set Error monotonically decreases as training progresses
- Validation Set Error will reach a minimum then start to increase
- Best generalization predicted to be at point of minimum validation set error

Early Stopping

- Use validation set to decide when to stop training
- Training Set Error monotonically decreases as training progresses
- Validation Set Error will reach a minimum then start to increase
- Best generalization predicted to be at point of minimum validation set error
- "Effective Flexibility" increases as training progresses
- Network has an increasing number of "effective degrees of freedom" as training progresses
- Network weights become more tuned to training data
- Very effective — used in many practical applications such as speech recognition and optical character recognition

Weight Decay

- Weight decay puts a "spring" on weights

Weight Decay

- Weight decay puts a "spring" on weights
- If training data puts a consistent force on a weight, it will outweigh weight decay

Weight Decay

- Weight decay puts a "spring" on weights
- If training data puts a consistent force on a weight, it will outweigh weight decay
- If training does not consistently push weight in a direction, then weight decay will dominate and weight will decay to 0

Weight Decay

- Weight decay puts a "spring" on weights
- If training data puts a consistent force on a weight, it will outweigh weight decay
- If training does not consistently push weight in a direction, then weight decay will dominate and weight will decay to 0
- Without weight decay, weight would walk randomly without being well determined by the data

Weight Decay

- Weight decay puts a "spring" on weights
- If training data puts a consistent force on a weight, it will outweigh weight decay
- If training does not consistently push weight in a direction, then weight decay will dominate and weight will decay to 0
- Without weight decay, weight would walk randomly without being well determined by the data
- Weight decay can allow the data to determine how to reduce the effective number of parameters

Penalizing Complexity

- Consider adding a *complexity term* E_w to the network error function, to encourage smoother mappings:

$$E = \underbrace{E_{\text{train}}}_{\text{data term}} + \underbrace{\beta E_W}_{\text{prior term}}$$

Penalizing Complexity

- Consider adding a *complexity term* E_w to the network error function, to encourage smoother mappings:

$$E = \underbrace{E_{\text{train}}}_{\text{data term}} + \underbrace{\beta E_W}_{\text{prior term}}$$

- E_{train} is the usual error function:

$$E^n_{\text{train}} = -\sum_{k=1}^{K} t^n_k \ln y^n_k$$

Penalizing Complexity

- Consider adding a *complexity term* E_w to the network error function, to encourage smoother mappings:

$$E = \underbrace{E_{\text{train}}}_{\text{data term}} + \underbrace{\beta E_W}_{\text{prior term}}$$

- E_{train} is the usual error function:

$$E^n_{\text{train}} = -\sum_{k=1}^{K} t^n_k \ln y^n_k$$

- If we choose the complexity term to be:

$$E_W = \frac{1}{2}\sum_i w_i^2$$

Then we have a simple partial derivative:

$$\frac{\partial E_W}{\partial w_i} = w_i$$

Backprop Training with Weight Decay

$$\frac{\partial E^n}{\partial w_i} = \frac{\partial(E^n_{\text{train}} + E_W)}{\partial w_i}$$
$$= \left(\frac{\partial E^n_{\text{train}}}{\partial w_i} + \beta \frac{\partial E_W}{\partial w_i}\right)$$
$$= \left(\frac{\partial E^n_{\text{train}}}{\partial w_i} + \beta w_i\right)$$
$$\Delta w_i = -\eta \left(\frac{\partial E^n_{\text{train}}}{\partial w_i} + \beta w_i\right)$$

- Weight decay corresponds to adding $E_w = 1/2 \sum_i w_i^2$ to the error function
- Addition of complexity terms is called *regularization*
- Weight decay is sometimes called L2 regularization
- E_W should be easily differentiable (for backprop) and should be some sort of flexibility measure

Summary

- The first coursework
- Generalisation
- Training / test / validation
- Early stopping and cross-validation
- Weight decay and regularization
- Reading:
  Michael Nielsen, chapters 2 & 3 of *Neural Networks and Deep Learning*
  http://neuralnetworksanddeeplearning.com/
  Chris Bishop, Chapters 6 & 9 of *Neural Networks for Pattern Recognition* (although a lot more detail than needed for now)