CSC321: Neural Networks

Lecture 7: Ways to prevent overfitting

Geoffrey Hinton

## Overfitting

- The training data contains information about the regularities in the mapping from input to output. But it also contains noise
  - The target values may be unreliable.
  - There is sampling error. There will be accidental regularities just because of the particular training cases that were chosen.
- When we fit the model, it cannot tell which regularities are real and which are caused by sampling error.
  - So it fits both kinds of regularity.
  - If the model is very flexible it can model the sampling error really well. This is a disaster.

## Preventing overfitting

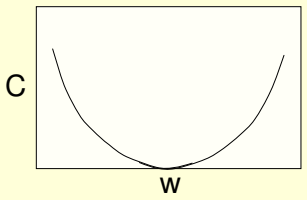
- Use a model that has the right capacity:
  - enough to model the true regularities
  - not enough to also model the spurious regularities (assuming they are weaker).
- Standard ways to limit the capacity of a neural net:
  - Limit the number of hidden units.
  - Limit the size of the weights.
  - Stop the learning before it has time to overfit.

## Limiting the size of the weights

- Weight-decay involves adding an extra term to the cost function that penalizes the squared weights.
  - Keeps weights small unless they have big error derivatives.

$$C = E + \frac{\lambda}{2}\sum_i w_i^2$$

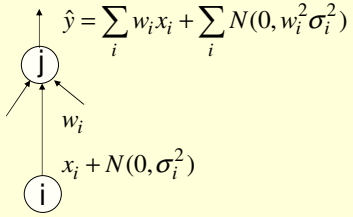
$$\frac{\partial C}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda w_i$$




$$when \ \frac{\partial C}{\partial w_i} = 0, \quad w_i = -\frac{1}{\lambda}\frac{\partial E}{\partial w_i}$$

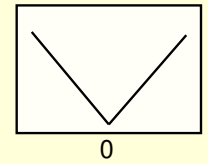
## Weight-decay via noisy inputs

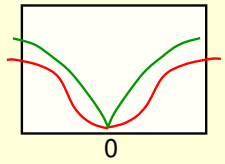
- Weight-decay reduces the effect of noise in the inputs.
  - The noise variance is amplified by the squared weight
- The amplified noise makes an additive contribution to the squared error.
  - So minimizing the squared error tends to minimize the squared weights when the inputs are noisy.
- It gets more complicated for non-linear networks.




## Other kinds of weight penalty

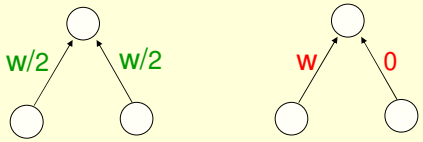
- Sometimes it works better to penalize the absolute values of the weights.
  - This makes some weights equal to zero which helps interpretation.
- Sometimes it works better to use a weight penalty that has negligible effect on large weights.




## The effect of weight-decay

- It prevents the network from using weights that it does not need.
  - This can often improve generalization a lot.
  - It helps to stop it from fitting the sampling error.
  - It makes a smoother model in which the output changes more slowly as the input changes. w
- If the network has two very similar inputs it prefers to put half the weight on each rather than all the weight on one.




## Deciding how much to restrict the capacity

- How do we decide which limit to use and how strong to make the limit?
  - If we use the test data we get an unfair prediction of the error rate we would get on new test data.
  - Suppose we compared a set of models that gave random results, the best one on a particular dataset would do better than chance. But it wont do better than chance on another test set.
- So use a separate validation set to do model selection.

## Using a validation set

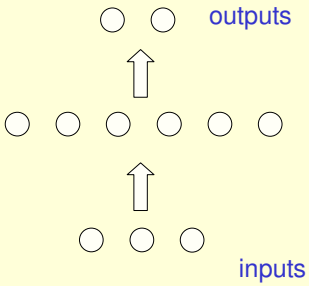
- Divide the total dataset into three subsets:
  - Training data is used for learning the parameters of the model.
  - Validation data is not used of learning but is used for deciding what type of model and what amount of regularization works best.
  - Test data is used to get a final, unbiased estimate of how well the network works. We expect this estimate to be worse than on the validation data.
- We could then re-divide the total dataset to get another unbiased estimate of the true error rate.

## Preventing overfitting by early stopping

- If we have lots of data and a big model, its very expensive to keep re-training it with different amounts of weight decay.
- It is much cheaper to start with very small weights and let them grow until the performance on the validation set starts getting worse (but don't get fooled by noise!)
- The capacity of the model is limited because the weights have not had time to grow big.

## Why early stopping works

- When the weights are very small, every hidden unit is in its linear range.
  - So a net with a large layer of hidden units is linear.
  - It has no more capacity than a linear net in which the inputs are directly connected to the outputs!
- As the weights grow, the hidden units start using their non-linear ranges so the capacity grows.




## Combining networks

- When the amount of training data is limited, we need to avoid overfitting.
  - Averaging the predictions of many different networks is a good way to do this.
  - It works best if the networks are as different as possible.
- If the data is really a mixture of several different "regimes" it is helpful to identify these regimes and use a separate, simple model for each regime.
  - We want to use the desired outputs to help cluster cases into regimes. Just clustering the inputs is not as efficient.

## Combining networks reduces variance

- We want to compare two expected squared errors
  - Method 1: Pick one of the predictors at random
  - Method 2: Use the average of the predictors, $\bar{y}$

$$\bar{y} = \langle y_i \rangle_i = \frac{1}{N}\sum_{i=1}^{N} y_i$$

$$\langle (d-y_i)^2 \rangle_i = \langle ((d-\bar{y})-(y_i-\bar{y}))^2 \rangle_i$$

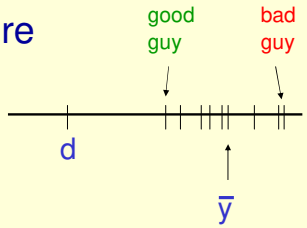
$$= \langle (d-\bar{y})^2 + (y_i-\bar{y})^2 - 2(d-\bar{y})(y_i-\bar{y}) \rangle_i$$

$$= \langle (d-\bar{y})^2 \rangle_i + \langle (y_i-\bar{y})^2 \rangle_i \ldots$$

$$-2(d-\bar{y})\langle (y_i-\bar{y}) \rangle_i$$

This term vanishes

## A picture




- The predictors that are further than average from d make bigger than average squared errors.
- The predictors that are nearer than average to d make smaller then average squared errors.
- The first effect dominates because squares work like that.
  - Don't try averaging if you want to synchronize a bunch of clocks !

$$(a+\varepsilon)^2 + (a-\varepsilon)^2 = 2a^2 + 2e^2$$

## How the combined predictor compares with the individual predictors

- On any one test case, some individual predictors will be better than the combined predictor.
  - But different individuals will be better on different cases.
- If the individual predictors disagree a lot, the combined predictor is typically better than all of the individual predictors when we average over test cases.
  - So how do we make the individual predictors disagree? (without making them much worse individually).

## Ways to make predictors differ

- Rely on the learning algorithm getting stuck in a different local optimum on each run.
  - A dubious hack unworthy of a true computer scientist (but definitely worth a try).
- Use lots of different kinds of models:
  - Different architectures
  - Different learning algorithms.
- Use different training data for each model:
  - Bagging: Resample (with replacement) from the training set: a,b,c,d,e -> a c c d d
  - Boosting: Fit models one at a time. Re-weight each training case by how badly it is predicted by the models already fitted.
    - This makes efficient use of computer time because it does not bother to "back-fit" models that were fitted earlier.