Neural Networks

Lecture 9
Ways of speeding up the learning and preventing overfitting

Five ways to speed up learning

- Use an adaptive global learning rate
  - Increase the rate slowly if its not diverging
  - Decrease the rate quickly if it starts diverging
- Use separate adaptive learning rate on each connection
  - Adjust using consistency of gradient on that weight axis
- Use momentum
  - Instead of using the gradient to change the position of the weight "particle", use it to change the velocity.
- Use a stochastic estimate of the gradient from a few cases
  - This works very well on large, redundant datasets.
- Don't go in the direction of steepest descent.
  - The gradient does not point at the minimum.
    - Can we preprocess the data or do something to the gradient so that we move directly towards the minimum?

The momentum method

Imagine a ball on the error surface with velocity v.

- It starts off by following the gradient, but once it has velocity, it no longer does steepest descent.
  - It damps oscillations by combining gradients with opposite signs.
  - It builds up speed in directions with a gentle but consistent gradient.
  - On an inclined plane it reaches a terminal velocity.

$$v(t) = \alpha\, v(t-1) - \varepsilon \frac{\partial E}{\partial w}(t)$$

$$\Delta w(t) = v(t)$$

$$= \alpha\, v(t-1) - \varepsilon \frac{\partial E}{\partial w}(t)$$

$$= \alpha\, \Delta w(t-1) - \varepsilon \frac{\partial E}{\partial w}(t)$$

$$v(\infty) = \frac{1}{1-\alpha}\left(-\varepsilon \frac{\partial E}{\partial w}\right)$$

Adaptive learning rates on each connection

- Use a global learning rate multiplied by a local gain on each connection.
- Increase the local gains if the gradient does not change sign.
- Use additive increases and multiplicative decreases.
  - This ensures that big learning rates decay rapidly when oscillations start.

$$\Delta w_{ij} = -\varepsilon \, g_{ij} \frac{\partial E}{\partial w_{ij}}$$

$$if \quad \left(\frac{\partial E}{\partial w_{ij}}(t) \frac{\partial E}{\partial w_{ij}}(t-1) \right) > 0$$

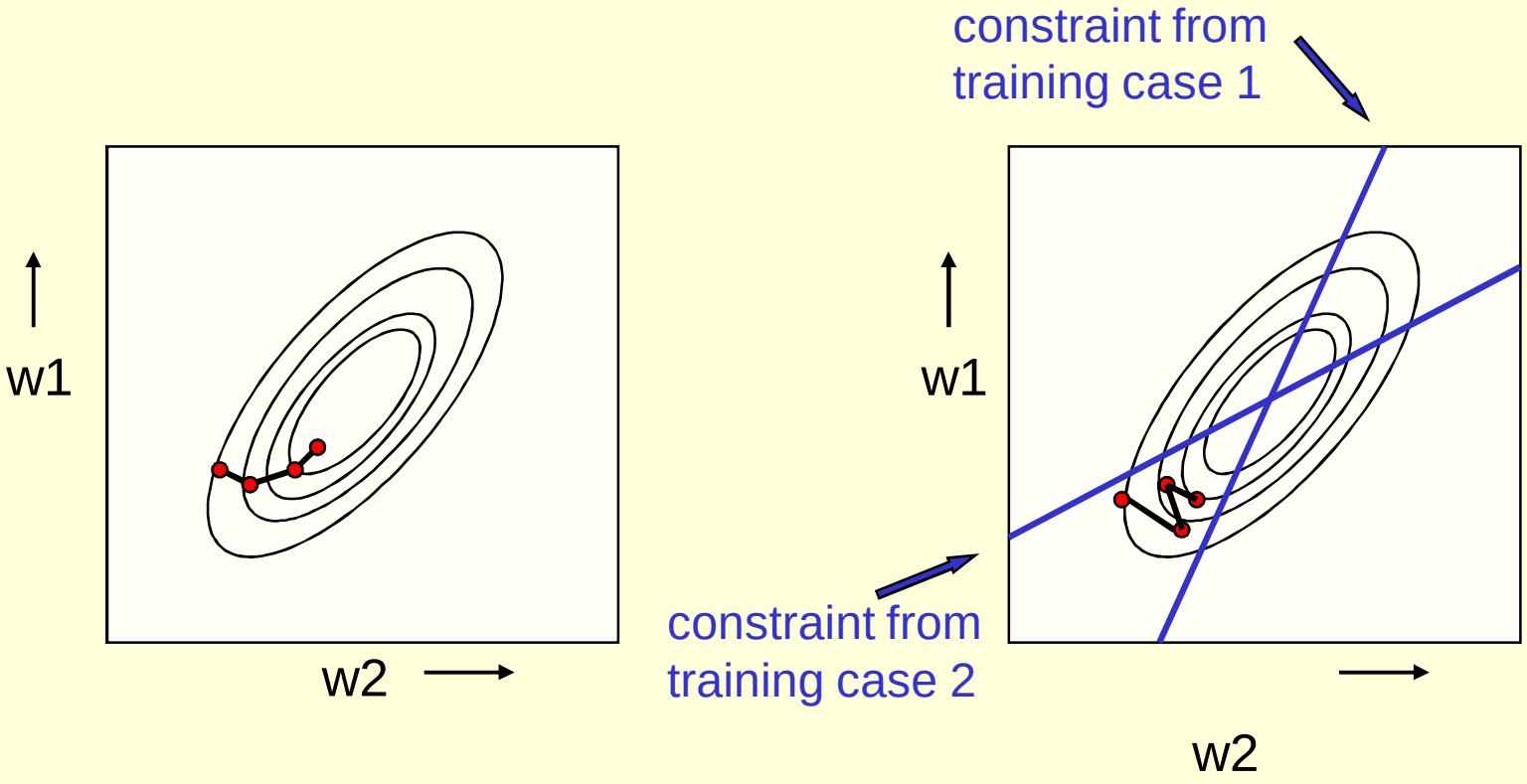
$$then \quad g_{ij}(t) = g_{ij}(t-1) + .05$$

$$else \quad g_{ij}(t) = g_{ij}(t-1) * .95$$

Online versus batch learning

- Batch learning does steepest descent on the error surface

Online learning updates the weights after each training case. It zig-zags around the direction of steepest descent.



Stochastic gradient descent

- If the dataset is highly redundant, the gradient on the first half is almost identical to the gradient on the second half.
  - So instead of computing the full gradient, update the weights using the gradient on the first half and then get a gradient for the new weights on the second half.
  - The extreme version is to update the weights after each example, but balanced mini-batches are just as good and faster in matlab.

Extra problems that occur in multilayer non-linear networks

- If we start with a big learning rate, the bias and all of the weights for one of the output units may become very negative.
  - The output unit is then very firmly off and it will never produce a significant error derivative.
  - So it will never recover (unless we have weight-decay).
- In classification networks that use a squared error or a cross-entropy error, the best guessing strategy is to make each output unit produce an output equal to the proportion of time it should be a 1.
  - The network finds this strategy quickly and takes a long time to improve on it. So it looks like a local minimum.

Overfitting

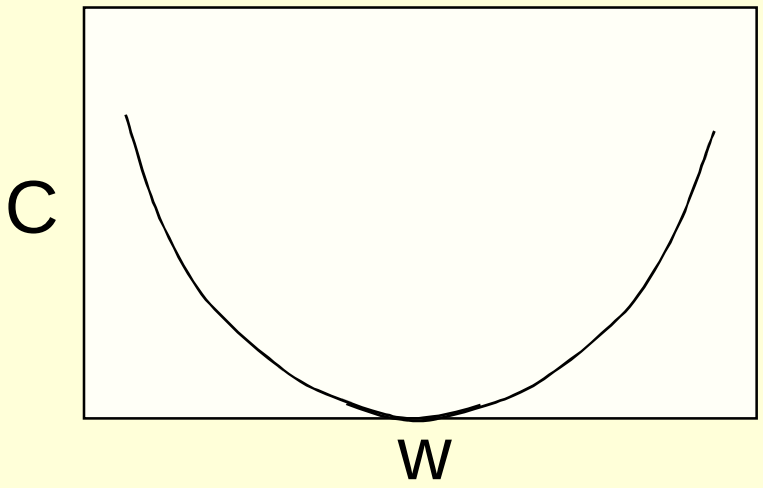
- The training data contains information about the regularities in the mapping from input to output. But it also contains noise
  - The target values may be unreliable.
  - There is sampling error. There will be accidental regularities just because of the particular training cases that were chosen.
- 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.



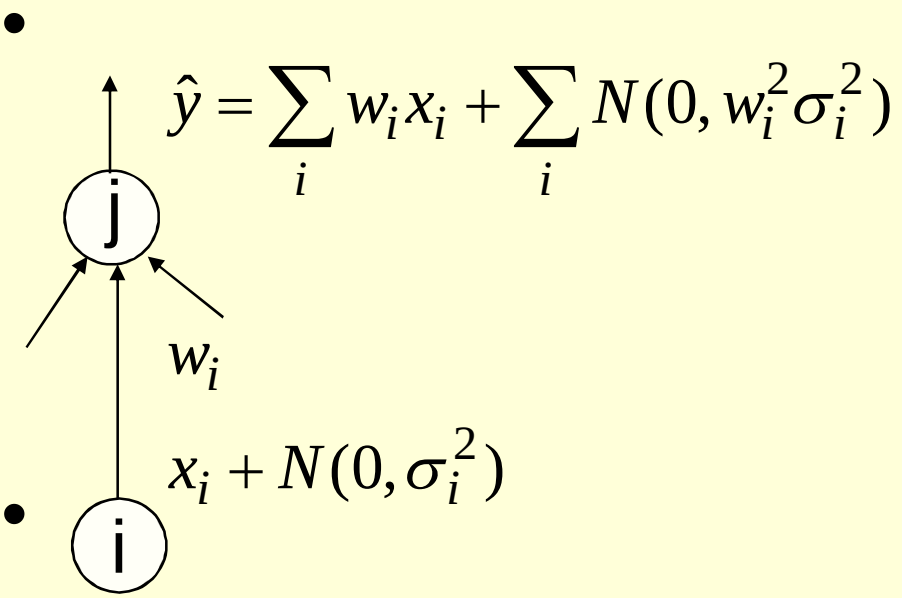
- $C = E + \frac{\lambda}{2} \sum_i w_i^2$

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

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

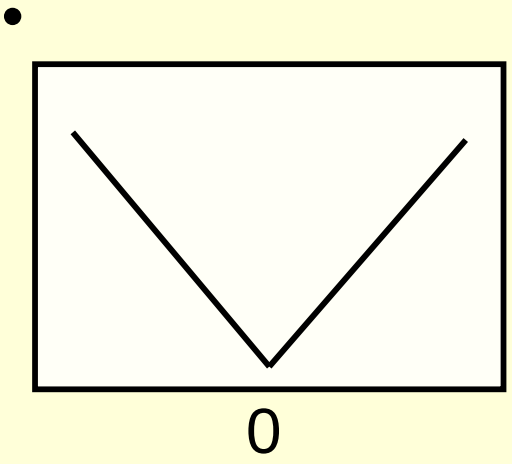
Weight-decay via noisy inputs

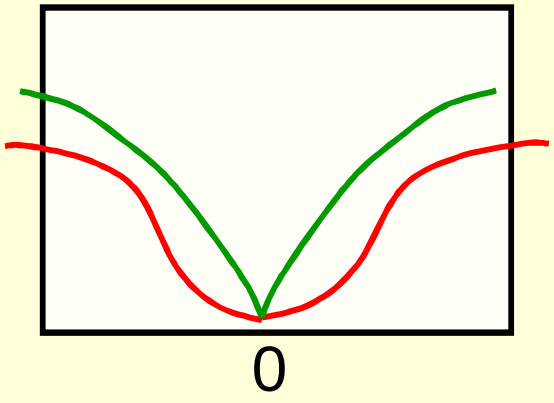
- Weight-decay reduces the effect of noise in the inputs.
  - The noise variance is amplified by the squared weight
- The amplified noise makes an additive contribution to the squared error.
  - So minimizing the squared error tends to minimize the squared weights when the inputs are noisy.
- It gets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



Other kinds of weight penalty

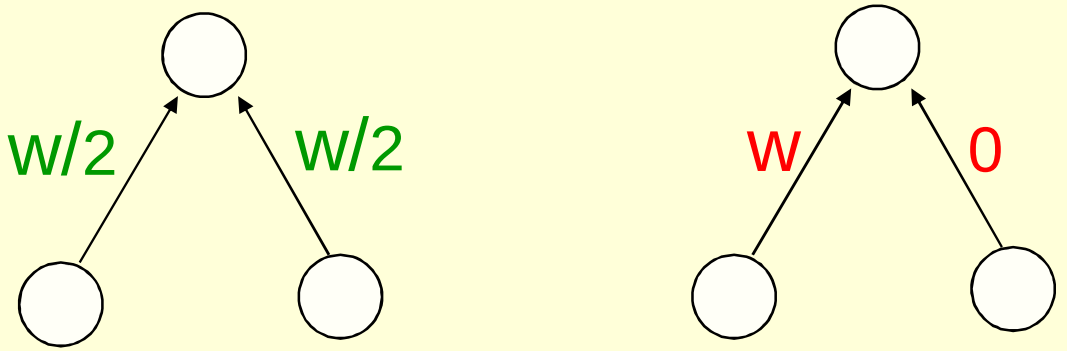
- Sometimes it works better to penalize the absolute values of the weights.
  - This makes some weights equal to zero which helps interpretation.
- Sometimes it works better to use a weight penalty that has negligible effect on large weights.





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.