Improving Neural Networks

Functional Programming and Intelligent Algorithms

Que Tran

Høgskolen i Ålesund

20th March 2017

Design the Network

- Training algorithms (Gradient descent backpropagation, Resilient backpropagation, Conjugate gradient backpropagation)
- Error function (e.g. Mean squared error, cross-entropy)
- Activation functions (sigmoid, tanh, soft-max, ReLU)
- Network architecture (number of hidden layers, nodes per layer)

Soft-max activation function

- It is most commonly used for classification problems where the 1-of-N output encoding is used.
- It can be written as:

$$y_k = g(h_k) = \frac{exp(h_k))}{\sum_j exp(h_j)}) \tag{1}$$

Soft-max activation function

- The soft-max function rescales the outputs so that the activations sum to 1 and lie between 0 and 1.
- The output from the soft-max layer can be interpreted of as a probability distribution.
- Error at the output layer:

$$\delta_{ok} = t_k - y_k \tag{2}$$

Overfitting

- *Overfitting*: model captures the noise of the data.
- Solutions:
  - Increasing the amount of training data
  - Reducing the network size
  - Early stopping
  - Regularization
  - Dropout

Regularization

- Adding complexity penalty to the cost function
- Some techniques:
  - L2 regularization
  - L1 regularization

L2 Regularization

— Adding an extra term, called *regularization term* to the cost function

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2 \tag{3}$$

where C_0 is the original, unregularized cost function, and $\lambda > 0$ is known as the regularization parameter, and n is the size of the training set

— The regularization term doesn't include the biases

L2 Regularization

— Taking the partial derivatives:

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w \tag{4}$$

$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b} \tag{5}$$

— The learning rules:

$$\begin{aligned} b &\rightarrow b - \eta \frac{\partial C_0}{\partial b} \\ w &\rightarrow w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta\lambda}{n} w \\ &= \left(1 - \frac{\eta\lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w} \end{aligned}$$

L1 Regularization

- Adding an extra term, called *regularization term* to the cost function

$$C = C_0 + \frac{\lambda}{n} \sum_w |w| \tag{6}$$

- The regularization term doesn't include the biases

L1 Regularization

- Taking the partial derivatives:

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} sgn(w) \tag{7}$$

  where $sgn(w)$ is the sign of w, that is, $+1$ if w is positive, and -1 if w is negative.

- The learning rules:

$$w \rightarrow w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta\lambda}{n} sgn(w)$$

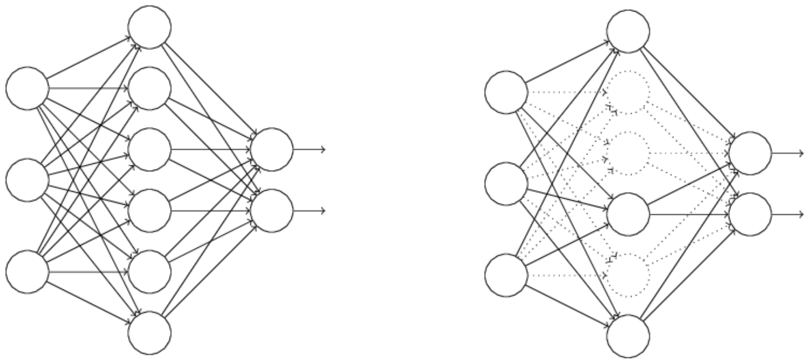
L1 Regularization

- Issue: the partial derivative $\frac{\partial C}{\partial w}$ is not defined when $w = 0$.
- Apply the unregularized rule when $w = 0$
- Intuitively the effect of regularization is to shrink weights, and when a weight is already 0, don't need to shrink.

Dropout

- Modifying the network itself.
- The modified process:
  - Randomly (and temporarily) deleting half the hidden neurons in the network, while leaving the input and output neurons untouched.
  - Forward-propagate the input x through the modified network, and then backpropagate the result, also through the modified network
  - Update the appropriate weights and biases.
  - Repeat the process
  - When we actually run the full network that means that twice as many hidden neurons will be active. To compensate for that, we halve the weights outgoing from the hidden neurons.

Dropout



Dropout

— Dropout as a form of model averaging

— Dropout as reducing complex co-adaptations of neurons

> *"This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons." (Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton (2012))*