Evaluating models fairly
MLP Tips

Yingming Li
yingming@zju.edu.cn

Data Science & Engineering Research Center, ZJU

13th March 2018

Adapted from slides provided by Prof. Michael Mandel.

MLP design parameters

- Several parameters to choose when designing an MLP (best to evaluate empirically)
- Number of hidden layers
- Number of units in each hidden layer
- Activation function
- Error function

Optimization tricks

- For a given network, local minima of the cost function are possible
- Many tricks exist to try to find better local minima
  - Momentum: mix in gradient from step
  - Weight initialization: small random values
  - Stopping criterion: early stopping
  - Learning rate annealing: start with large, slowly shrink
  - Second order methods: use a separate for each parameter or pair of parameters based on local curvature
  - Randomization of training example order
  - Regularization, i.e., terms in E(w) that only depend on w

Learning rate control: momentum

- To ease oscillating weights due to large η, some inertia (momentum) of weight update is added

$$\Delta w_{ji}(n) = \eta \delta_j y_i + \alpha \Delta w_{ji}(n-1), \quad 0 < \alpha < 1$$

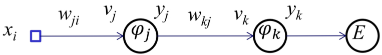
  - In the downhill situation, $\Delta w_{ji}(n) \approx \frac{\eta}{1-\alpha} \delta_j y_i$
    - thus accelerating learning by a factor of $1/(1-\alpha)$
  - In the oscillating situation, it smooths weight change, thus stabilizing oscillations

Input pre-processing

- Remove mean
  - Avoids extra update steps to learn it
- Divide by standard deviation
  - Or whiten by multiplying by the square root of the covariance matrix
  - Make dimensions commensurate
  - Scales curvature of error surface to be less canyon-like

Weight initialization

- Consider a network with one hidden layer and a single output neuron
- What happens if we initialize all weights to 0?



$$y_k = \varphi\left(\sum\nolimits_j w_{kj}\varphi\left(\sum\nolimits_i w_{ji}x_i\right)_j\right)_k$$

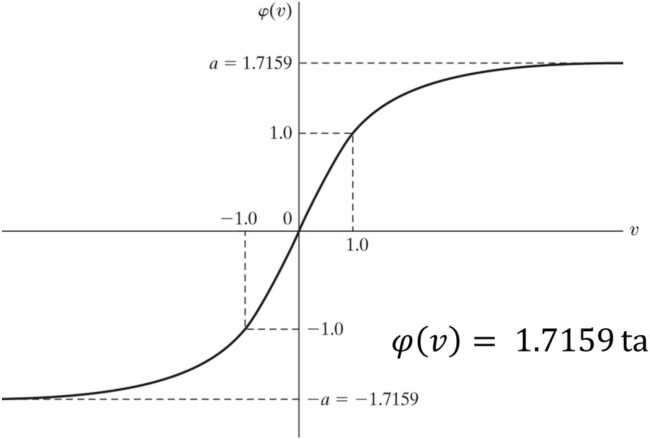
$$\frac{\partial}{\partial w_{ji}}E(\mathbf{w}) = -e_k\varphi'(v_k)y_j$$

$$\frac{\partial}{\partial w_{ji}}E(\mathbf{w}) = -e_j\varphi'(v_j)x_i$$

Weight initialization

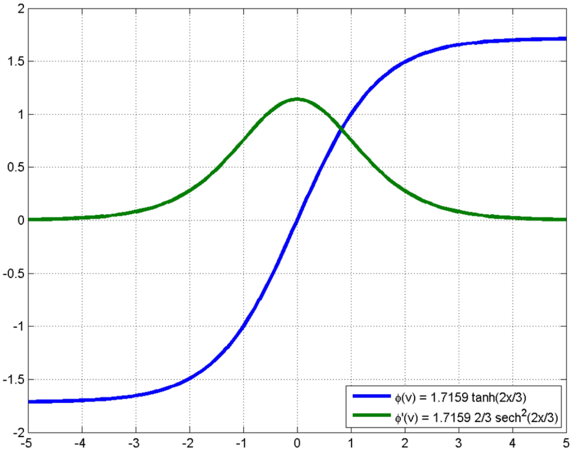
- Break symmetry by initializing with random values
- If inputs are normalized, they are uncorrelated, with zero-mean, and unit-variance
- Would like output to be approximately the same
- So inputs to sigmoid nonlinearity must be too

Hyperbolic tangent function



$$\varphi(v) = 1.7159 \tanh\left(\frac{2}{3} v\right)$$

Hyperbolic tangent function



Weight initialization

$$\begin{aligned}
\sigma^2_{y_i} &= E_x\{y_i^2\} = E_x\left\{\varphi\left(\sum_j w_{ij}x_j\right)\right\} \\
&\approx E_x\left\{\left(\sum_j w_{ij}x_j\right)^2\right\} \approx \sum_j w_{ij}^2 E_x\{x_j^2\} \\
&= \sum_{j=1}^{m} w_{ij}^2
\end{aligned}$$

- So in order to make $\sigma^2_{y_i} = 1$
  - Initialize w_{ij} randomly with $\sigma^2_w = \frac{1}{m}$

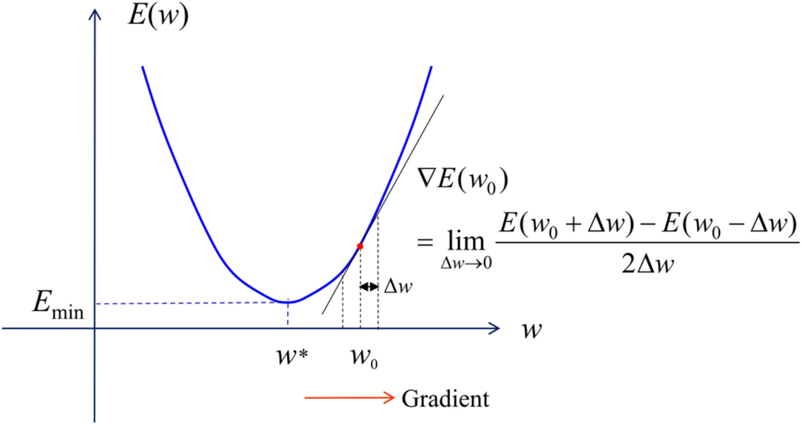
Debugging: Gradient checking

- Is your backpropagation code working properly?
  - I.e., is it computing the right gradient?
- Backpropagation computes

$$\nabla_{\mathbf{w}} E(\mathbf{x}_p; \mathbf{w}) = \left[\frac{\partial E}{\partial W_{111}}, \frac{\partial E}{\partial W_{121}}, \cdots, \frac{\partial E}{\partial W_{NML}} \right]$$

  - where $w_{i_1 i_2 l}$ is the weight in layer l connecting neurons i_1 and i_2
- Compute the gradient **numerically** and compare

Recall: Gradient illustration



Debugging: Gradient checking

- One-sided numerical gradient:

$$\frac{\partial E}{\partial w_{i_1 i_2 l}} \approx \frac{1}{\delta}\Big(E\left(\mathbf{x}_p; \mathbf{w} + \delta \mathbf{1}_{i_1 i_2 l}\right) - E(\mathbf{x}_p; \mathbf{w}) \Big)$$

  - where $\mathbf{1}_{i_1 i_2 l}$ is a vector that is 1 at entry $1_{i_1 i_2 l}$ and 0 everywhere else and δ is a "small" constant

- Two-sided numerical gradient:

$$\frac{1}{2\delta}\Big(E\left(\mathbf{x}_p; \mathbf{w} + \delta \mathbf{1}_{i_1 i_2 l}\right) - E(\mathbf{x}_p; \mathbf{w} - \delta \mathbf{1}_{i_1 i_2 l}) \Big)$$

  - More accurate approximation
  - But requires twice as many evaluations of $E(\mathbf{x}_p; \mathbf{w})$

Debugging: Gradient checking

- Complexity of backpropagation
  - 1 forward pass (O(1) multiply and add per weight)
  - 1 backward pass (O(1) multiply and add per weight)
- Complexity of numerical gradient
  - One-sided: 1 forward pass *per network weight*
    - So W + 1 forward passes total
  - Two-sided: 2 forward passes *per network weight*
- So numerical gradient is good for checking correctness of backpropagation
  - But very slow to use in training, especially for large W

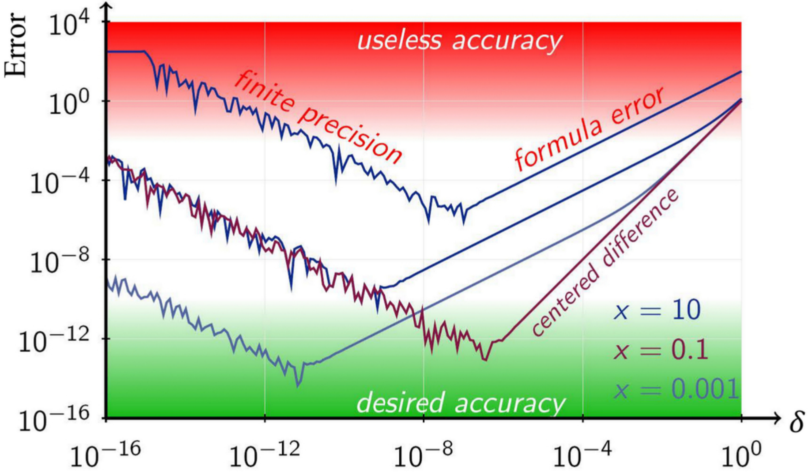
Gradient checking procedure

- Select an example data point, $\mathbf{x}_p$, initialize $\mathbf{w}$
- Compute the gradient of $E(\mathbf{x}_p; \mathbf{w})$ using backprop
  - Gives a vector of derivatives, one for each weight in the network
- Compute the gradient numerically
  - Evaluate $E(\mathbf{x}_p; \mathbf{w})$
  - Loop over each weight in the network
    - Evaluate $E(\mathbf{x}_p; \mathbf{w} + \sigma \mathbf{1}_{i_1 i_2 l})$, compute partial derivative
- If they are not the same, look for patterns as a function of $i_1 i_2 l$, etc

How to select δ?

- δ too big means derivative might be different at $E(\mathbf{x}_p; \mathbf{w} + \delta \mathbf{1}_{i_1 i_2 l})$ and $E(\mathbf{x}_p; \mathbf{w})$
  - Leading to a bad estimate using the above formulas
- δ too small runs into numerical issues
  - Need to be aware of limitations of floating point math
  - For δ too small, $1 + \delta = 1$
  - This might be around 1e-16, depending on the data type (e.g., float, double)
  - So $\delta = 1$e-8 might be reasonable

How to select δ?



"AbsoluteErrorNumericalDifferentiationExample" by Berland - Self-made using TikZ, Beamer and LaTeX. Licensed under Public domain via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:AbsoluteErrorNumericalDifferentiationExample.png #mediaviewer/File:AbsoluteErrorNumericalDifferentiationExample.png

Thank you!