Improving gradient descent

Jony Sugianto
jony@evolvemachinelearners.com
0812-13086659
github.com/jonysugianto

Improving Backprop Performance

- Avoiding local minima
- Keep derivatives from going to zero
- For classifiers, use reachable targets
- Compensate for error attenuation in deep layers
- Compensate for fan-in effects
- Use momentum to speed learning
- Reduce learning rate when weights oscillate
- Use small initial random weights and small initial learning rate to avoid “herd effect”
- Cross-entropy error measure

Avoiding Local Minima

One problem with backprop is that the error surface is no longer bowl-shaped.

Gradient descent can get trapped in local minima.

In practice, this does not usually prevent learning.

"Noise" can get us out of local minima:

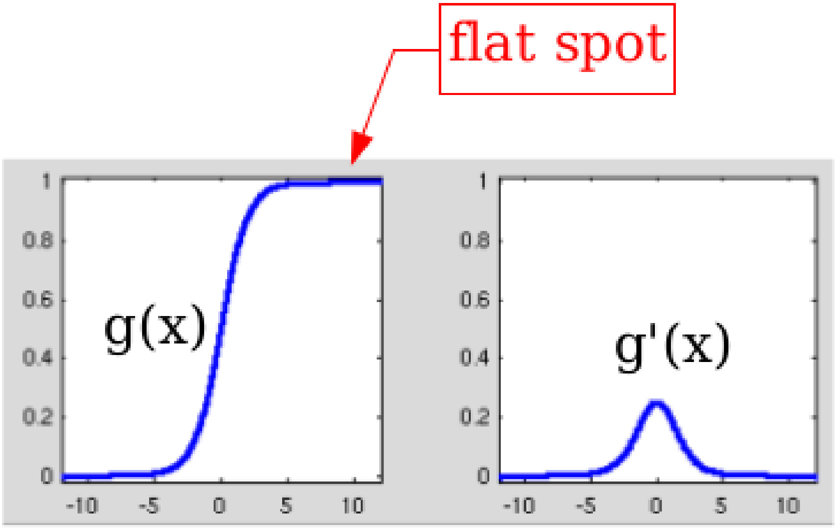
Stochastic update (one pattern at a time).

Add noise to training data, weights, or activations.

Large learning rates can be a source of noise due to overshooting.

Flat Spots

If weights become large, net_j becomes large, derivative of g() goes to zero.



Fahlman's trick: add a small constant to g'(x) to keep the derivative from going to zero. Typical value is 0.1.

Reachable Targets for Classifiers

Targets of 0 and 1 are unreachable by the logistic or tanh functions.

Weights get large as the algorithm tries to force each output unit to reach its asymptotic value.

Trying to get a “correct” output from 0.95 up to 1.0 wastes time and resources that should be concentrated elsewhere.

Solution: use “reachable targets” of 0.1 and 0.9 instead of 0/1. And don't penalize the network for overshooting these targets.

Error Signal Attenuation

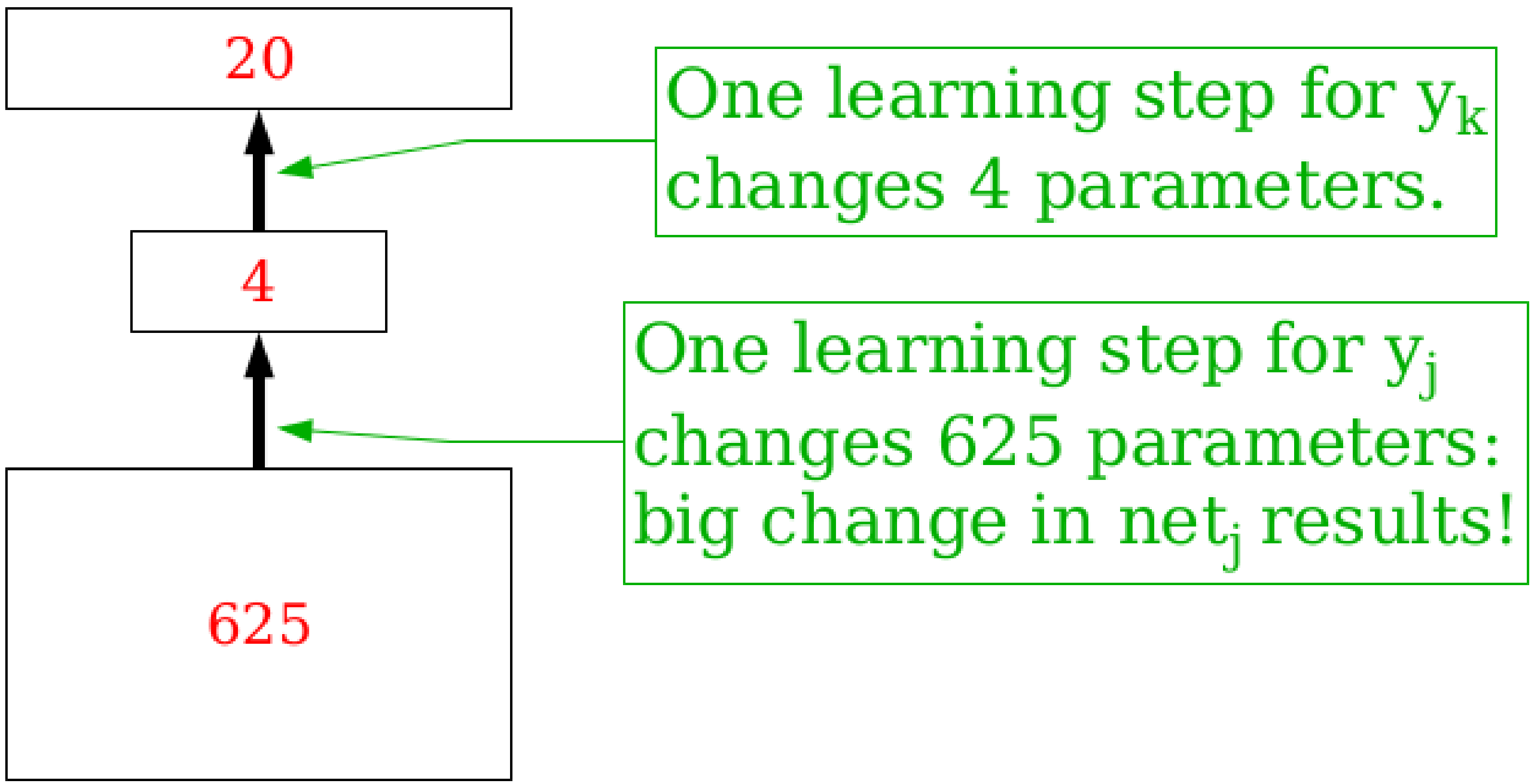
The error signal δ gets attenuated as it moves backward through multiple layers.

So different layers learn at different rates.

Input-to-hidden weights learn more slowly than hidden-to-output weights.

Solution: have different learning rates η for different layers.

Fan-In Affects Learning Rate



Solution: scale learning rate by fan-in.

Momentum

Learning is slow if the learning rate is set too low.

Gradient may be steep in some directions but shallow in others.

Solution: add a momentum term α.

$$\Delta w_{ij}(t) = -\eta \frac{\partial E}{\partial w_{ij}(t)} + \alpha \cdot \Delta w_{ij}(t-1)$$

Typical value for α is 0.5.

If the direction of the gradient remains constant, the algorithm will take increasingly large steps.

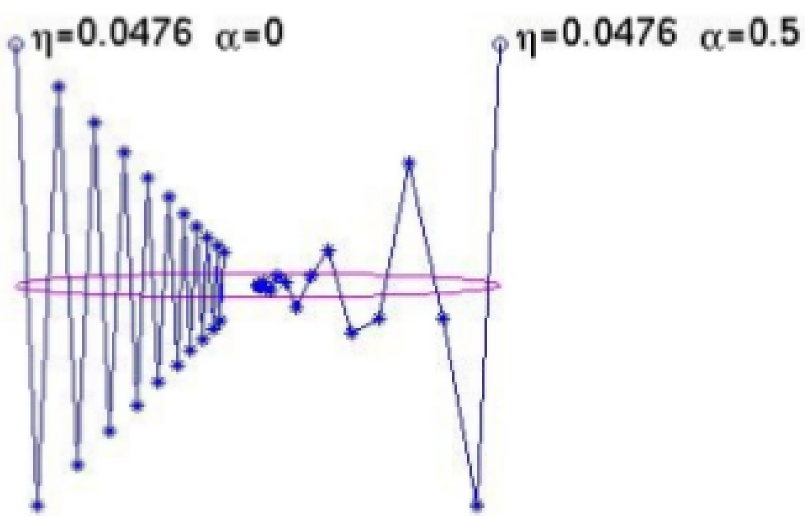
Momentum Demo

Hertz, Krogh & Palmer figs. 5.10 and 6.3: gradient descent on a quadratic error surface E (no neural net) involved:

$$E = x^2 + 20y^2$$

$$\frac{\partial E}{\partial x} = 2x, \quad \frac{\partial E}{\partial y} = 40y$$

Initial $[x, y] = [-1, 1]$ or $[1, 1]$



Weights Can Oscillate If Learning Rate Set Too High

Solution: calculate the cosine of the angle between successive weight vectors.

$$\cos\theta = \frac{\vec{\Delta} w(t) \cdot \vec{\Delta} w(t-1)}{\|\vec{\Delta} w(t)\| \cdot \|\vec{\Delta} w(t-1)\|}$$

If cosine close to 1, things are going well.

If cosine < 0.95, reduce the learning rate.

If cosine < 0, we're oscillating: cancel the momentum.

$$\Delta w(t) = -\eta \frac{\partial E}{\partial w} + \alpha \cdot \Delta w(t-1)$$

Cross-Entropy Error Measure

- Alternative to sum-squared error for binary outputs; diverges when the network gets an output completely wrong.

$$E = \sum_p \left[d^p \log \frac{d^p}{y^p} + (1-d^p) \log \frac{1-d^p}{1-y^p} \right]$$

- Can produce faster learning for some types of problems.
- Can learn some problems where sum-squared error gets stuck in a local minimum, because it heavily penalizes “very wrong” outputs.