Learning and Adaptivity

Neural Network II - Lecture XII

Videos for This Lecture

- Repetition (Part 0)
- Computational Graphs (Part 1)
- Gradient Descent (Part 2)
- Optimisation Methods (Part 3)

Course Outline

Basic Concepts

- Parametric Method,
- Bayesian Learning and Nonparametrics Methods
- Clustering and Mixture of Gaussians

Classification Approaches

- Linear Discriminants
- Ensemble Methods and Boosting
- Randomized Trees, Forest

Reinforcement Learning

- Classical Reinforcement Learning
- Deep Reinforcement Learning

Deep Learning

- Foundations
- Optimization

Today's topics

Learning Multi-layer Networks

- Recap: Backpropagation
- Computational graphs
- Automatic differentiation

Gradient Descent

- Stochastic Gradient Descent
- Choosing Learning Rates
- Momentum
- Update learning rate
- RMS Prop
- Other Optimizers

Recap: Learning with Hidden Units

How can we train multi-layer networks efficiently?

- Need an efficient way of adapting all weights, not just the last layer.

Idea: Gradient Descent

- Set up an error function

$$E(\mathrm{W}) = \sum_n L(t_n, y(\mathrm{x}_n; \mathrm{W})) + \lambda \Omega(\mathrm{W})$$

- with a loss L(.) and a regularise $\Omega(.)$
- E.g., $L(t, y(\mathrm{x}; \mathrm{W})) = \sum_n (y(\mathrm{x}_n; \mathrm{W}) - t_n)^2$ L2-loss

$$\Omega(\mathrm{W}) = ||\mathrm{W}||_F^2$$ L2-regulariser

=> Update each weight $W_{ij}^{(k)}$ in the direction of the gradient $\frac{\partial E(\mathrm{W})}{\partial W_{ij}^{(k)}}$

Gradient Descent

Two main steps

1. Computing the gradient for each weight
2. Adjusting the weights in the direction of the gradient

Recap: Backpropagation Algorithm

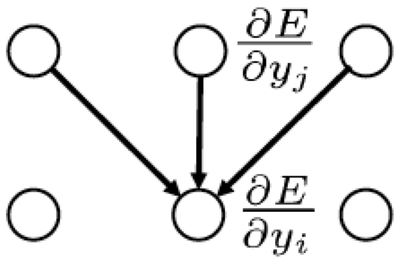
Core steps

- Convert the discrepancy between each output and its target value into an error derivate

$$E = \frac{1}{2} \sum_{j \in output} (t_j - y_j)^2$$

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$

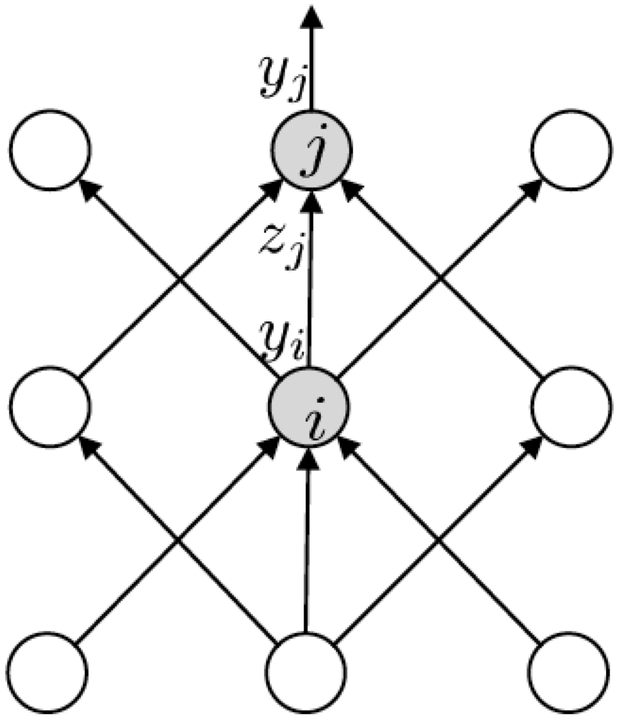
- Compute error derivatives in each hidden layer from error derivatives in the layer above



- Use error derivatives w.r.t. activities to get error derivatives w.r.t. the incoming weights

$$\frac{\partial E}{\partial y_j} \rightarrow \frac{\partial E}{\partial w_{ik}}$$

Recap: Backpropagation Algorithm



$$\frac{\partial E}{\partial z_j} = \frac{\partial y_j}{\partial z_j}\frac{\partial E}{\partial y_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial z_j}{\partial y_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$

Efficient propagation scheme

- y_j is already known from forward pass!
- =>Propagate back the gradient from layer j and multiply with y_j

Recap: MLP Backpropagation

Forward Pass

$$\mathbf{y}^{(0)} = \mathbf{x}$$

for $k = 1, ..., l$ do

$$\mathbf{z}^{(k)} = \mathbf{W}^{(k)}\mathbf{y}^{(k-1)}$$

$$\mathbf{y}^{(k)} = g_k(\mathbf{z}^{(k)})$$

endfor

$$\mathbf{y} = \mathbf{y}^{(l)}$$

$$E = L(\mathbf{t}, \mathbf{y}) + \lambda\Omega(\mathbf{W})$$

Backward Pass

$$\mathbf{h} \leftarrow \frac{\partial E}{\partial \mathbf{y}} = \frac{\partial}{\partial \mathbf{y}} L(\mathbf{t}, \mathbf{y}) + \lambda \frac{\partial}{\partial \mathbf{y}} \Omega$$

for $k = l, l\text{-}1, ..., 1$ do

$$\mathbf{h} \leftarrow \frac{\partial E}{\partial \mathbf{z}^{(k)}} = \mathbf{h} \odot g'(\mathbf{y}^{(k)})$$

$$\frac{\partial E}{\partial \mathbf{W}^{(k)}} = \mathbf{h}\mathbf{y}^{(k-1)\top} + \lambda \frac{\partial \Omega}{\partial \mathbf{W}^{(k)}}$$

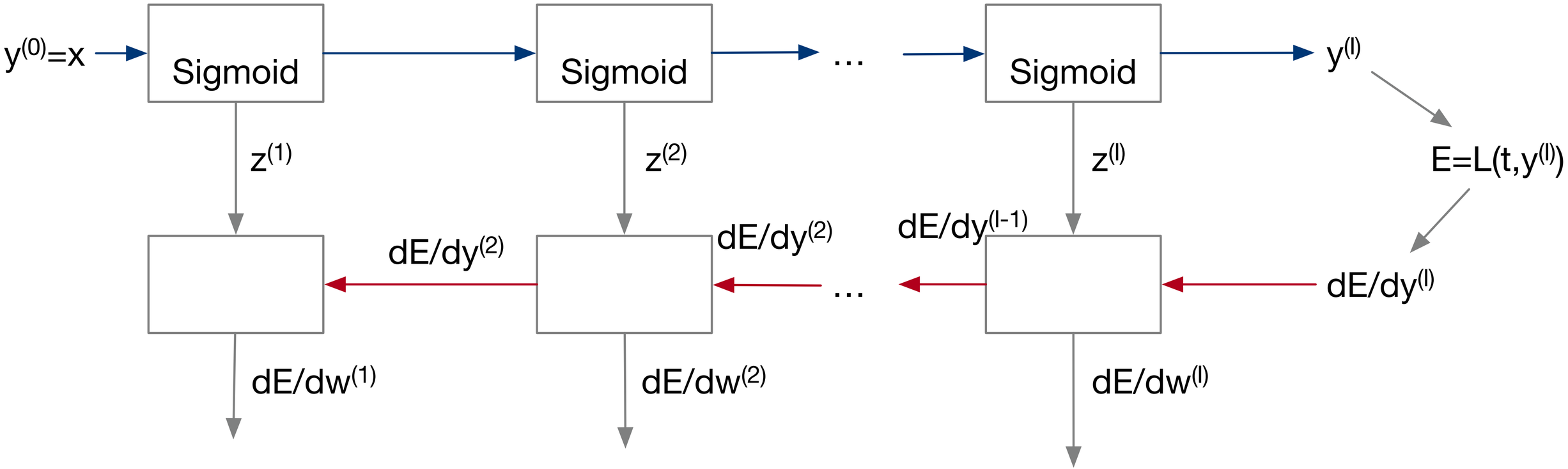
$$\mathbf{h} \leftarrow \frac{\partial E}{\partial \mathbf{y}^{(k-1)}} = \mathbf{W}^{(k)\top}\mathbf{h}$$

endfor

Notes

- For efficiency, an entire batch of data X is processed at once.
- $\odot$ denotes the element-wise product

MLP Backpropagation Summary



Part 1, Video NeuralNetworkII_p1

- Computational Graphs

Today's topics

Learning Multi-layer Networks

- Recap: Backpropagation
- Computational graphs
- Automatic differentiation

Gradient Descent

- Stochastic Gradient Descent
- Choosing Learning Rates
- Momentum
- Update learning rate
- RMS Prop
- Other Optimizers

Computational Graphs

We can think of mathematical expressions as graphs

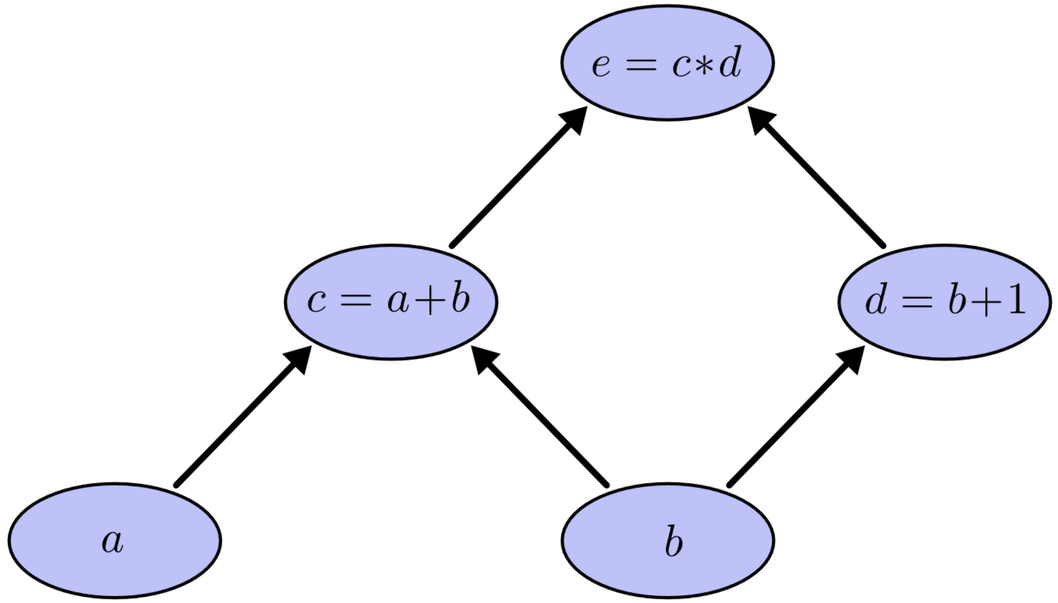
E.g., consider the expression

$$e = (a + b) * (b + 1)$$

We can decompose this into the operations

$$c = a + b$$
$$d = b + 1$$
$$e = c * d$$

and visualize this as a computational graph



Evaluating partial derivatives $\frac{\partial Y}{\partial X}$ in such a graph

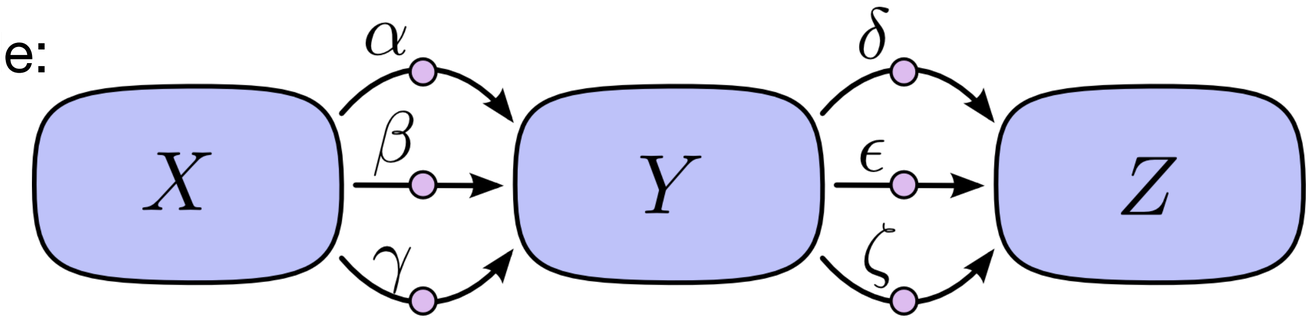
- General rule: sum over all possible paths from Y to X and multiply the derivatives on each edge of the path together.

Image source: Christopher Olah, colah.github.io

Factoring Paths

Problem: Combination explosion

Example:



There are 3 paths from X to Y and 3 more from Y to Z.

If we want to compute $\frac{\partial Z}{\partial X}$, we need to sum over 3x3 paths:

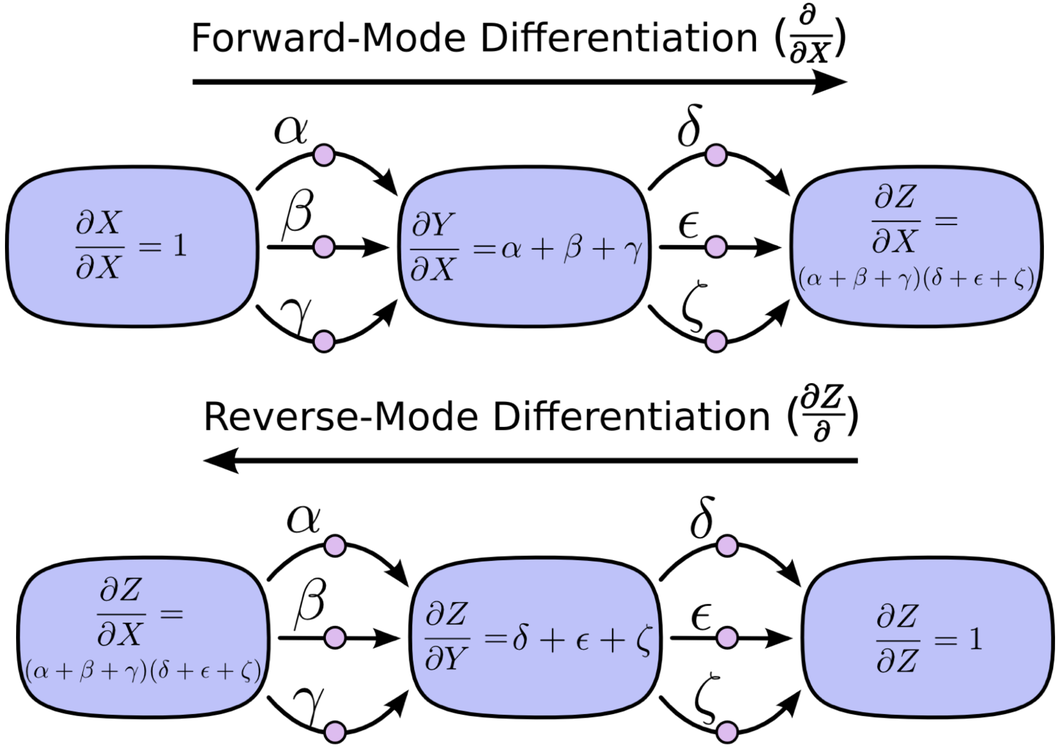
$$\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\epsilon + \alpha\zeta + \beta\delta + \beta\epsilon + \beta\zeta + \gamma\delta + \gamma\epsilon + \gamma\zeta$$

Instead of naively summing over paths, it's better to factor them

$$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma) * (\delta + \epsilon + \zeta)$$

Image source: Christopher Olah, colah.github.io

Efficient Factoring Algorithms



Apply operator $\frac{\partial}{\partial X}$ to every node.

Apply operator $\frac{\partial Z}{\partial}$ to every node.

Efficient algorithms for computing the sum

- Instead of summing over all of the paths explicitly, compute the sum more efficiently by merging paths back together at every node.

Image source: Christopher Olah, colah.github.io

Why Do We Care?

Let's consider the example again

- Using forward-mode differentiation from b up …
- Runtime: O(#edges)
- Result: derivative of every node with respect to b.

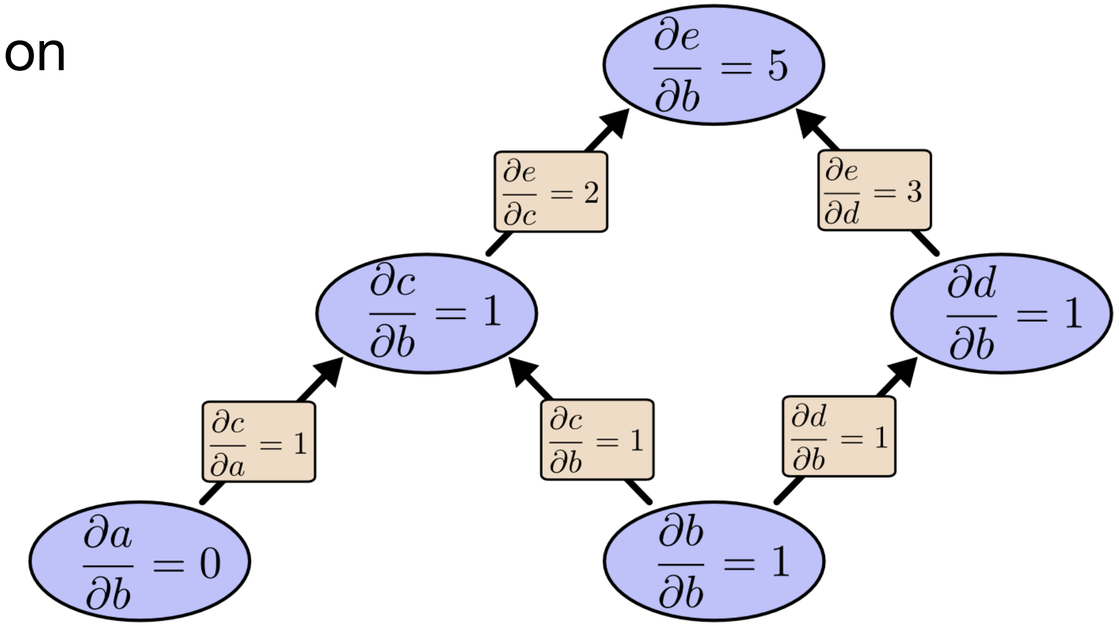
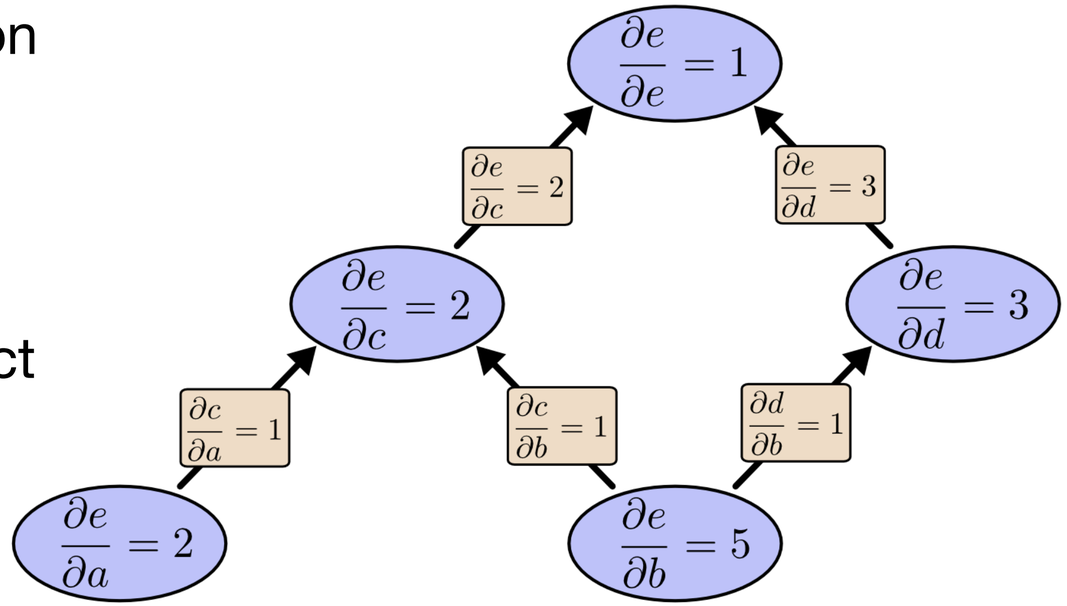


Image source: Christopher Olah, colah.github.io

Why Do We Care?

Let's consider the example again

- Using reverse-mode differentiation from e down …
- Runtime: O(#edges)
- Result: derivative of e with respect to every node..



➡*This is what we want to compute in Backpropagation!*

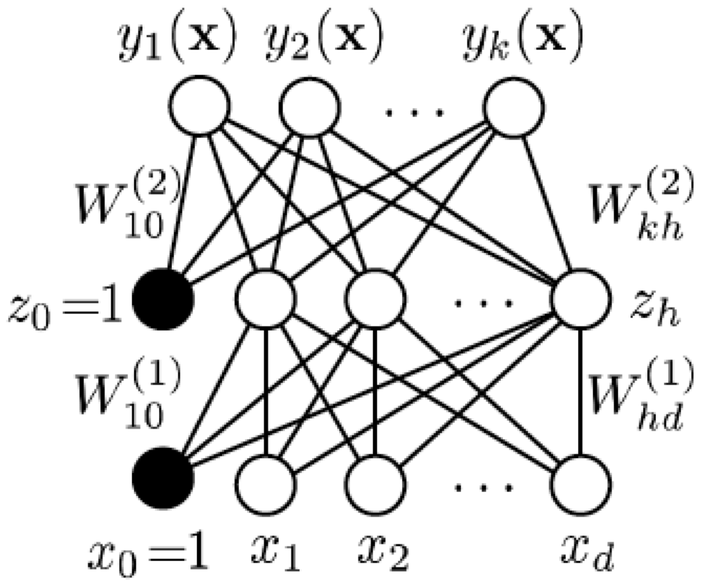
- Forward differentiation needs one pass per node. With backward differentiation can compute all derivatives in one single pass.

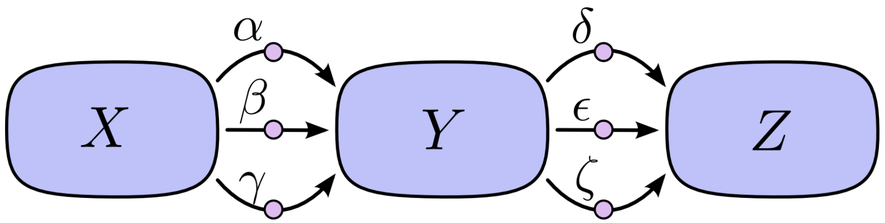
➡Speed-up in O(#inputs) compared to forward differentiation!

Image source: Christopher Olah, colah.github.io

Obtaining the Gradients

Approach 4: Automatic Differentiation





- Convert the network into a computational graph.
- Each new layer/module just needs to specify how it affects the forward and backward passes.
- Apply reverse-mode differentiation.

=> Very general algorithm, used in today's Deep Learning packages

Image source: Christopher Olah, colah.github.io

Implementing Softmax Correctly

Softmax output

- De-facto standard for multi-class outputs.

$$E(\mathrm{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{\mathbb{I}(t_n = k)\ln\frac{\exp(\mathrm{w}_k^T\mathrm{x})}{\sum_{j=1}^{K}\exp(\mathrm{w}_j^T\mathrm{x})}\}$$

Practical issue

- Exponentials get very big and can have vastly different magnitudes.
- **Trick 1**: Do not compute first softmax, then log, but instead directly evaluate log-exp in the denominator.
- **Trick 2**: Softmax has the property that for a fixed vector b

$$\text{softmax}(a+b)=\text{softmax}(a)$$

=> Subtract the largest weight vector w_j from the others.

Part 2, Video NeuralNetworkII_p2

- Gradient Descent

Today's topics

Learning Multi-layer Networks

- Recap: Backpropagation
- Computational graphs
- Automatic differentiation

Gradient Descent

- Stochastic Gradient Descent
- Choosing Learning Rates
- Momentum
- Update learning rate
- RMS Prop
- Other Optimizers

Gradient Descent

Two main steps

1. Computing the gradient for each weight
2. Adjusting the weights in the direction of the gradient

Recall: Basic update equation

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

Main questions

- On what data do we want to apply this?
- How should we choose the step size ´ (the learning rate)?

Stochastic vs. Batch Learning

Batch learning

- Process the full dataset at once to compute the gradient.

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

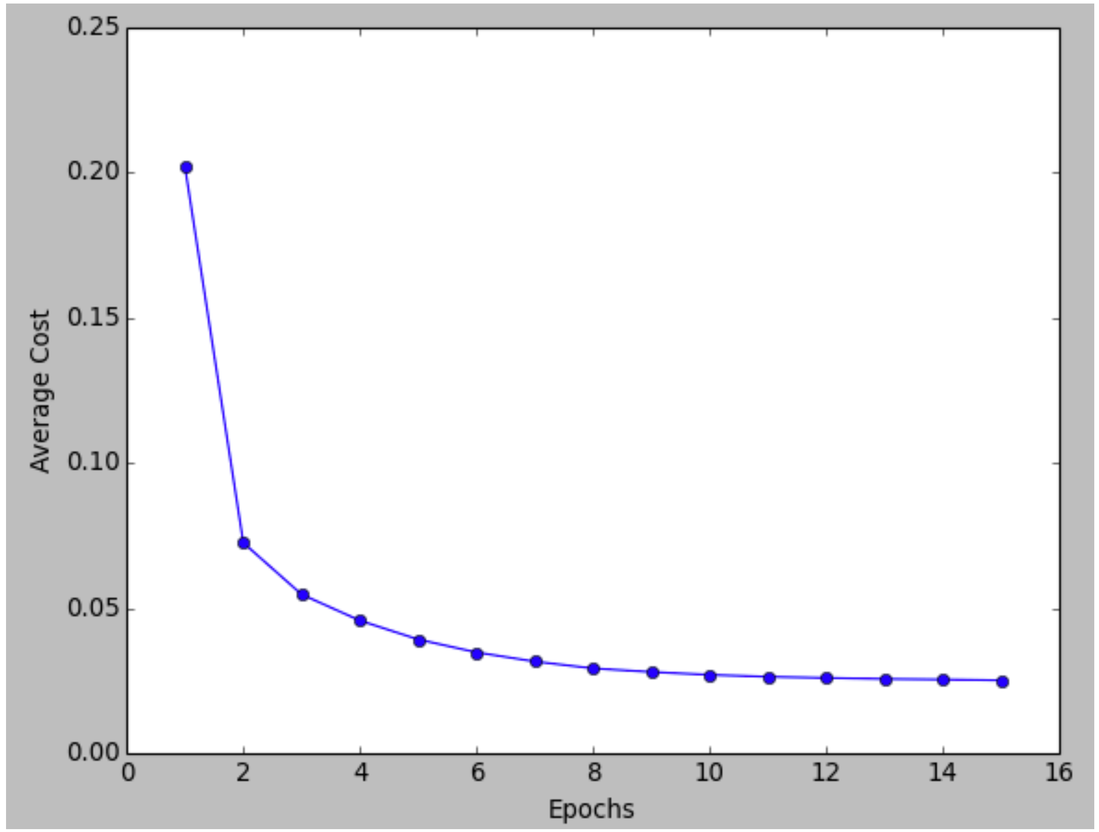
Stochastic learning

- Choose a single example from the training set.
- Compute the gradient only based on this example
- This estimate will generally be noisy, which has some advantages.

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

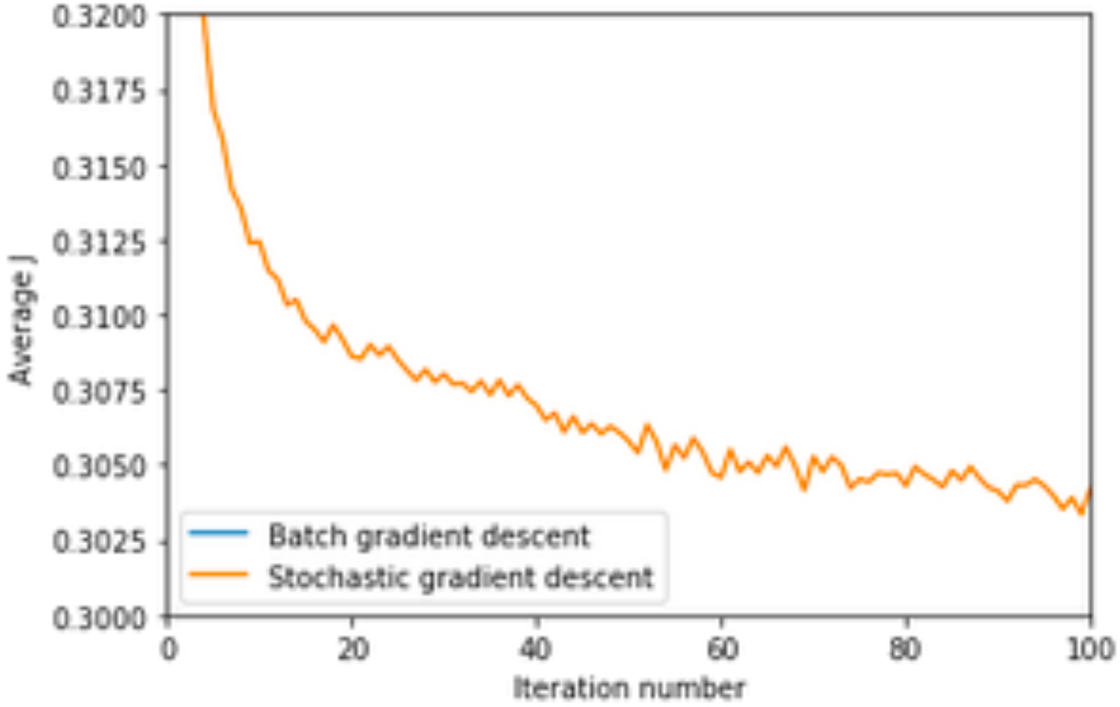
Stochastic vs. Batch Learning

Batch learning



Cost vs Epochs (Source: https://www.bogotobogo.com/python/scikit-learn/scikit-learn_batch-gradient-descent-versus-stochastic-gradient-descent.php)

Stochastic learning



Cost vs Epochs in SGD (Source: https://adventuresinmachinelearning.com/stochastic-gradient-descent/)

Stochastic vs. Batch Learning

Batch learning advantages

- Conditions of convergence are well understood.
- Many acceleration techniques (e.g., conjugate gradients) only operate in batch learning.
- Theoretical analysis of the weight dynamics and convergence rates are simpler.

Stochastic learning advantages

- Usually much faster than batch learning.
- Often results in better solutions.
- Can be used for tracking changes.

Middle ground: Minibatches

Minibatches

Idea

- Process only a small batch of training examples together
- Start with a small batch size & increase it as training proceeds.

Advantages

- Gradients will more stable than for stochastic gradient descent, but still faster to compute than with batch learning.
- Take advantage of redundancies in the training set.
- Matrix operations are more efficient than vector operations.

Caveat

- Error function should be normalized by the minibatch size, s.t. we can keep the same learning rate between minibatches

$$E(\mathrm{W}) = -\frac{1}{N}\sum_{n} L(t_n, y(\mathrm{x}_n; \mathrm{W})) + \frac{\lambda}{N}\Omega(\mathrm{W})$$

Today's topics

Learning Multi-layer Networks

- Recap: Backpropagation
- Computational graphs
- Automatic differentiation

Gradient Descent

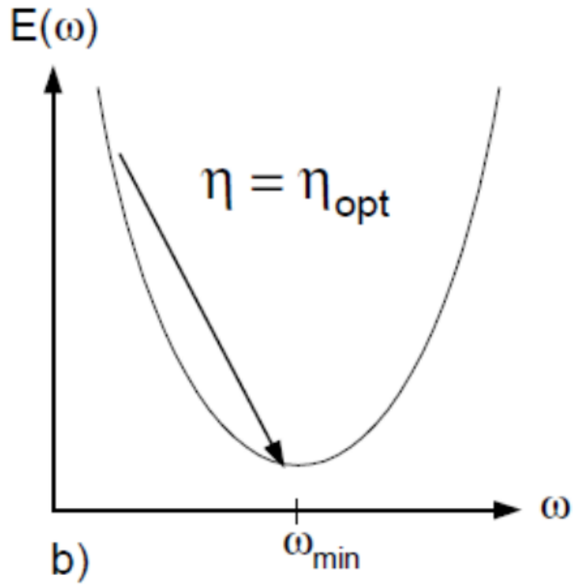
- Stochastic Gradient Descent
- Choosing Learning Rates
- Momentum
- Update learning rate
- RMS Prop
- Other Optimizers

Choosing the Right Learning Rate

Analyzing the convergence of Gradient Descent

- Consider a simple 1D example first

$$W^{(\tau-1)} = W^{(\tau)} - \eta \frac{\mathrm{d}E(W)}{\mathrm{d}W}$$



- What is the optimal learning rate η_{opt}?
- If E is quadratic, the optimal learning rate is given by the inverse of the Hessian

$$\eta_{opt} = \left(\frac{\mathrm{d}^2 E(W^{(\tau)})}{\mathrm{d}W^2}\right)^{-1}$$

- *What happens if we exceed this learning rate?*

Image source: Yann LeCun et al., Efficient BackProp (1998)

Choosing the Right Learning Rate

Behavior for different learning rates

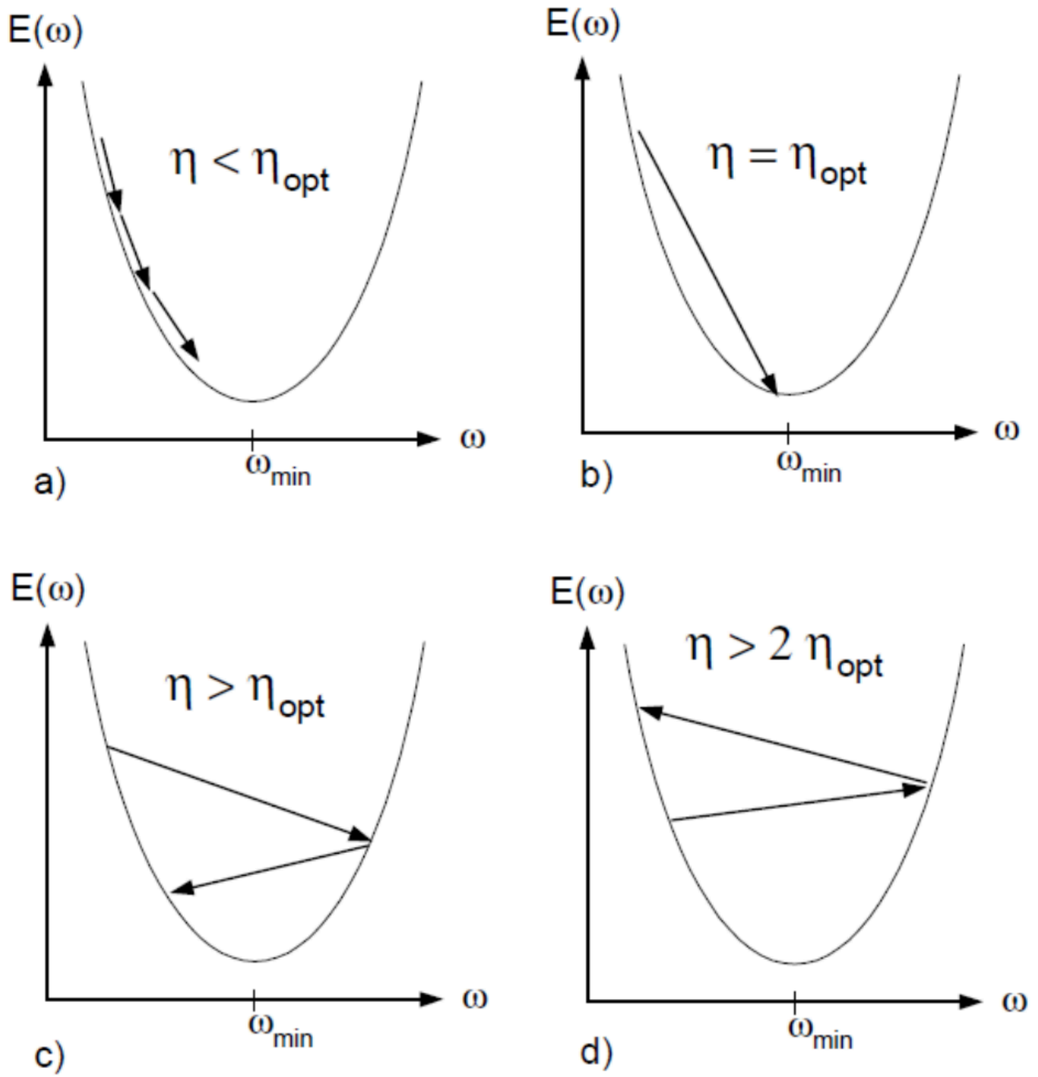
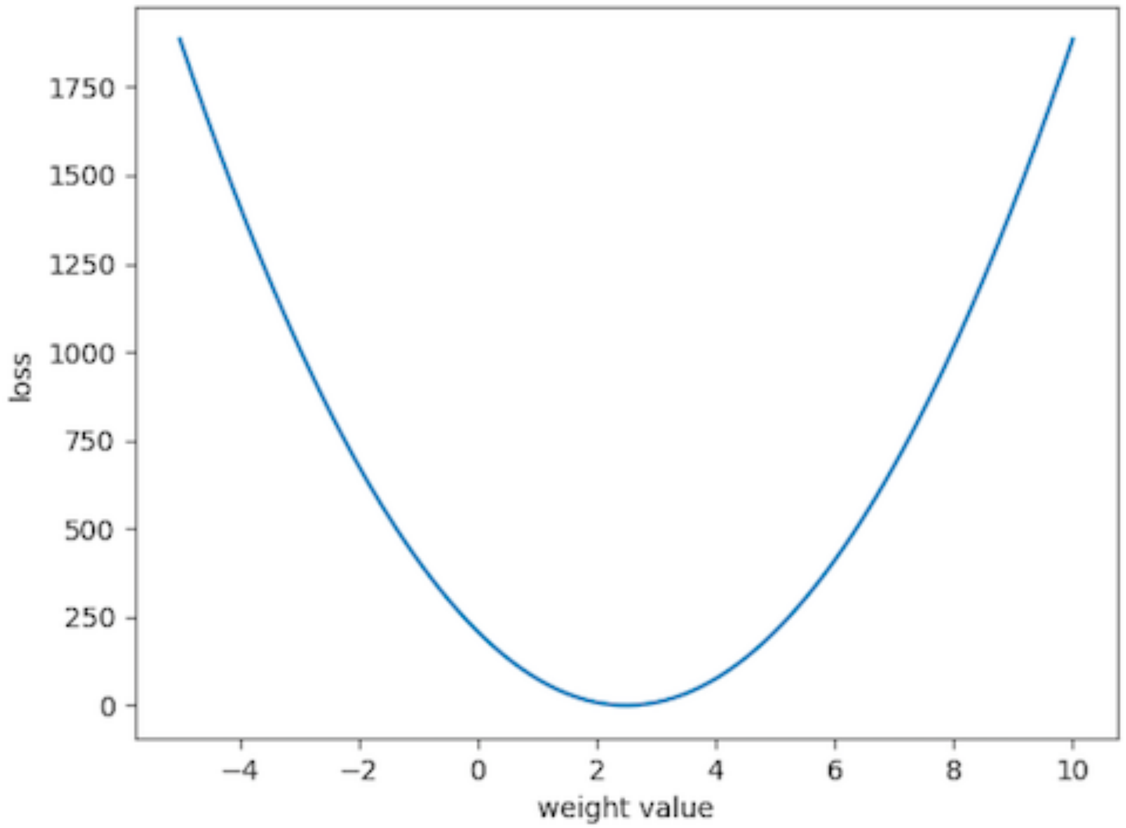


Image source: Yann LeCun et al., Efficient BackProp (1998)

Learning Rate vs. Training Error



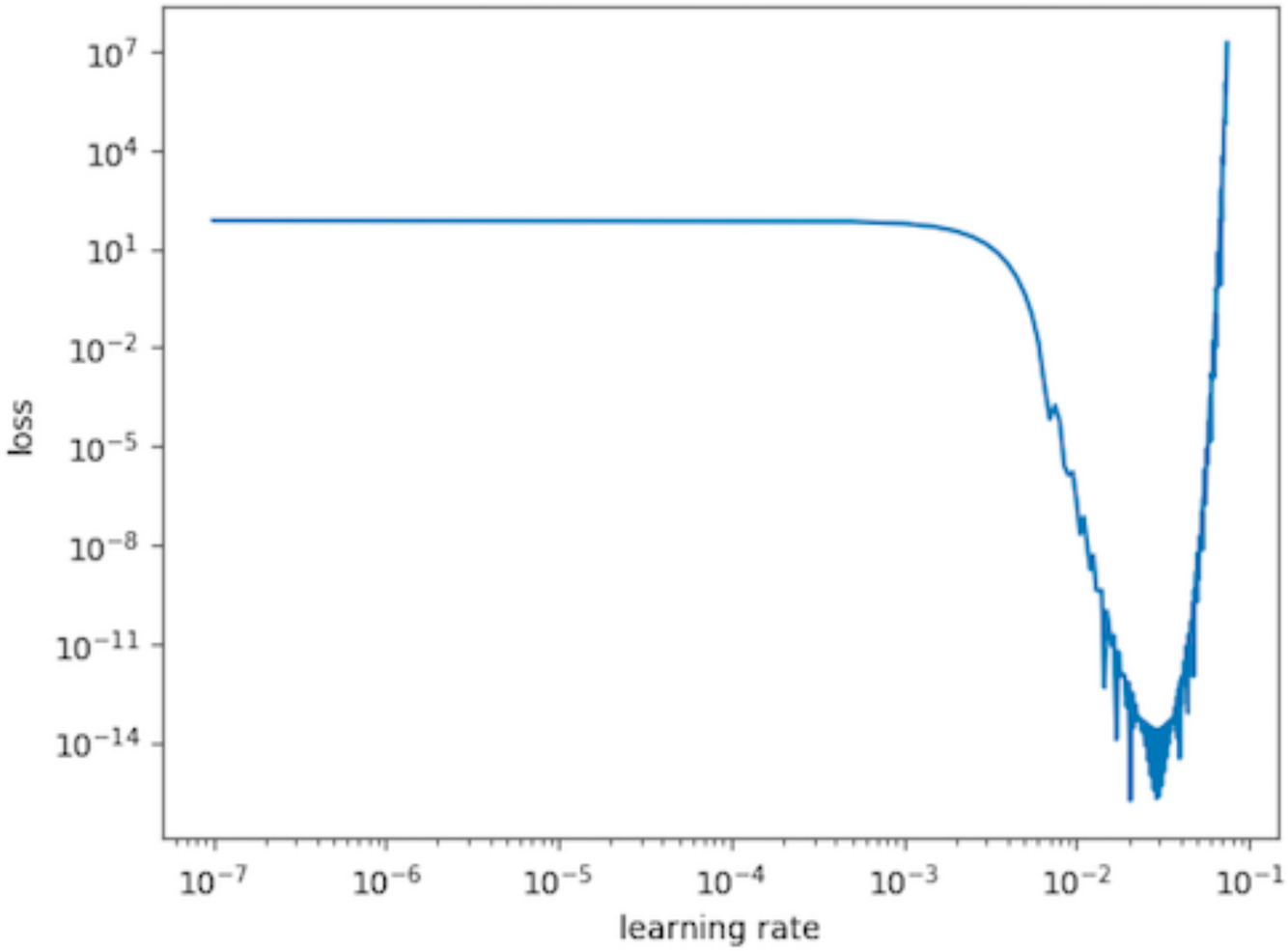


Image source:http://www.bdhammel.com/learning-rates

Part 3, Video NeuralNetworkII_p3

- Optimisation Methods

Today's topics

Learning Multi-layer Networks

- Recap: Backpropagation
- Computational graphs
- Automatic differentiation

Gradient Descent

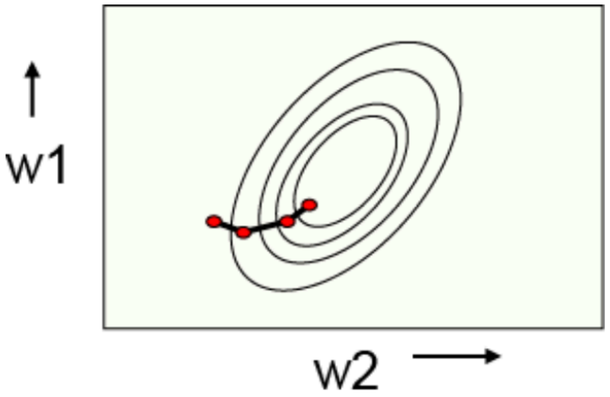
- Stochastic Gradient Descent
- Choosing Learning Rates
- Momentum
- Update learning rate
- RMS Prop
- Other Optimizers

Batch vs. Stochastic Learning

Batch Learning

- Simplest case: steepest decent on the error surface.

=> Updates perpendicular to contour lines



Stochastic Learning

- Simplest case: zig-zag around the direction of steepest descent.

=> Updates perpendicular to constraints from training examples.

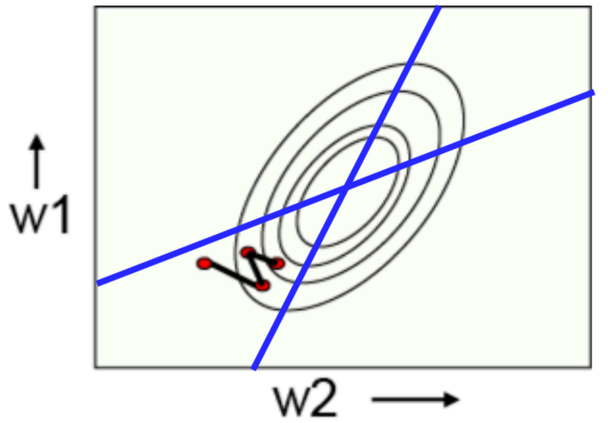
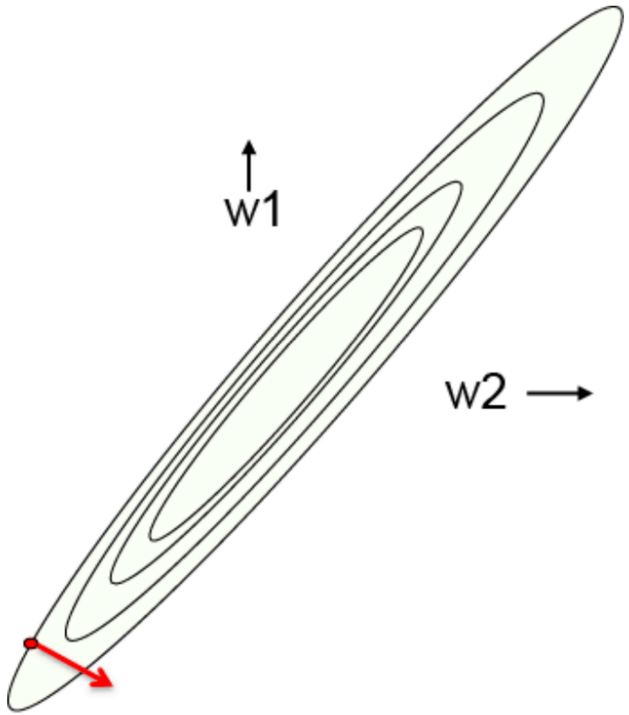


Image source:Geoff Hinton

Why Learning Can Be Slow

If the inputs are correlated

- The ellipse will be very elongated.
- The direction of steepest descent is almost perpendicular to the direction towards the minimum!



This is just the opposite of what we want!

Image source:Geoff Hinton

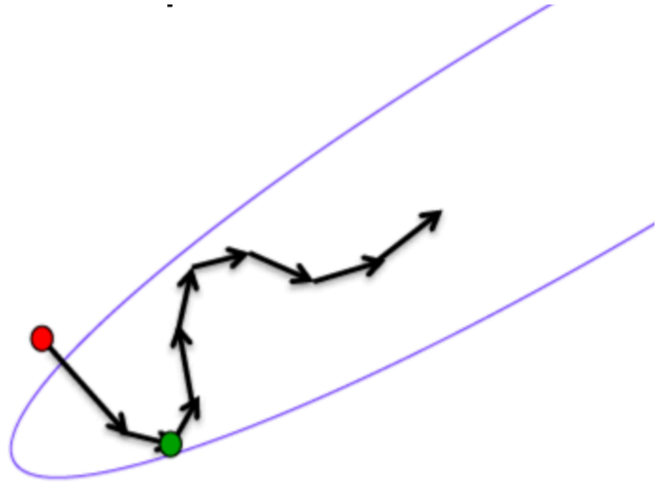
The Momentum Method

Idea

- Instead of using the gradient to change the position of the weight “particle”, use it to change the velocity.

Intuition

- Example: Ball rolling on the error surface
- It starts off by following the error surface, but once it has accumulated momentum, it no longer does steepest decent



Effect

- Dampen oscillations in directions of high curvature by combining gradients with opposite signs.
- Build up speed in directions with a gentle but consistent gradient.

Image source:Geoff Hinton

The Momentum Method: Implementation

Change in the update equations

- Effect of the gradient: increment the previous velocity, subject to a decay by $\alpha < 1$

$$v(t) = \alpha v(t-1) - \eta \frac{dE}{dw}(t)$$

- Set the weight change to the current velocity

$$\begin{aligned} \Delta w &= v(t) \\ &= \alpha v(t-1) - \eta \frac{dE}{dw}(t) \\ &= \alpha \Delta w(t-1) - \eta \frac{dE}{dw}(t) \end{aligned}$$

The Momentum Method: Behaviour

Behavior

- If the error surface is a tilted plane, the ball reaches a terminal velocity

$$v(\infty) = \frac{1}{1-\alpha}\left(-\eta \frac{dE}{dw}\right)$$

  - If the momentum α is close to 1, this is much faster than simple gradient descent.

- At the beginning of learning, there may be very large gradients.
  - Use a small momentum initially (e.g., $\alpha = 0.5$).
  - Once the large gradients have disappeared and the weights are stuck in a ravine, the momentum can be smoothly raised to its final value (e.g., $\alpha = 0.90$ or even $\alpha = 0.99$).

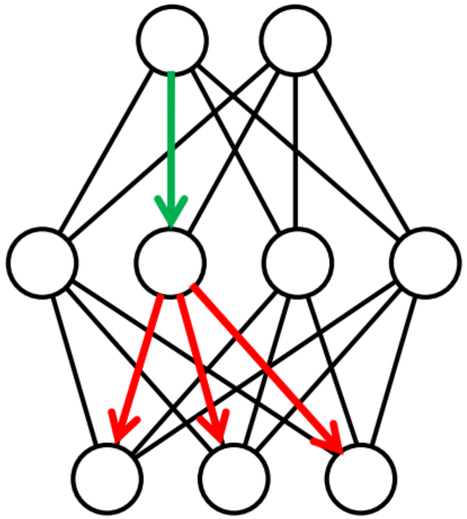
=> This allows us to learn at a rate that would cause divergent oscillations without the momentum.

Separate, Adaptive Learning Rates

Problem

- In multilayer nets, the appropriate learning rates can vary widely between weights.
- The magnitudes of the gradients are often very different for the different layers, especially if the initial weights are small.

  ➡Gradients can get very small in the early layers of deep nets.



$$\Delta w_{ij} = -\eta g_{ij} \frac{dE}{dw_{ij}}$$

$$if \ (\frac{dE}{dw_{ij}}(t)\frac{dE}{dw_{ij}}(t-1)) > 0$$

$$then \ g_{ij}(t) = g_{ij}(t-1) + 0.05$$

$$else \ g_{ij}(t) = g_{ij}(t-1) * 0.95$$

Better Adaptation: RMSProp

Motivation

- The magnitude of the gradient can be very different for different weights and can change during learning.
- This makes it hard to choose a single global learning rate.
- For batch learning, we can deal with this by only using the sign of the gradient, but we need to generalize this for minibatches.

Idea of RMSProp

- Divide the gradient by a running average of its recent magnitude

$$MeanSq(w_{ij}, t) = 0.9MeanSq(w_{ij}, t-1) + 0.1(\frac{\partial E}{\partial w_{ij}}(t))^2$$

- Divide the gradient by $sqrt(MeanSq(w_{ij}, t))$

Other Optimizers

AdaGrad [Duchi ’10]

AdaDelta [Zeiler ’12]

Adam [Ba & Kingma ’14]

Notes

- All of those methods have the goal to make the optimization less sensitive to parameter settings.
- Adam is currently becoming the quasi-standard

Behavior in a Long Valley

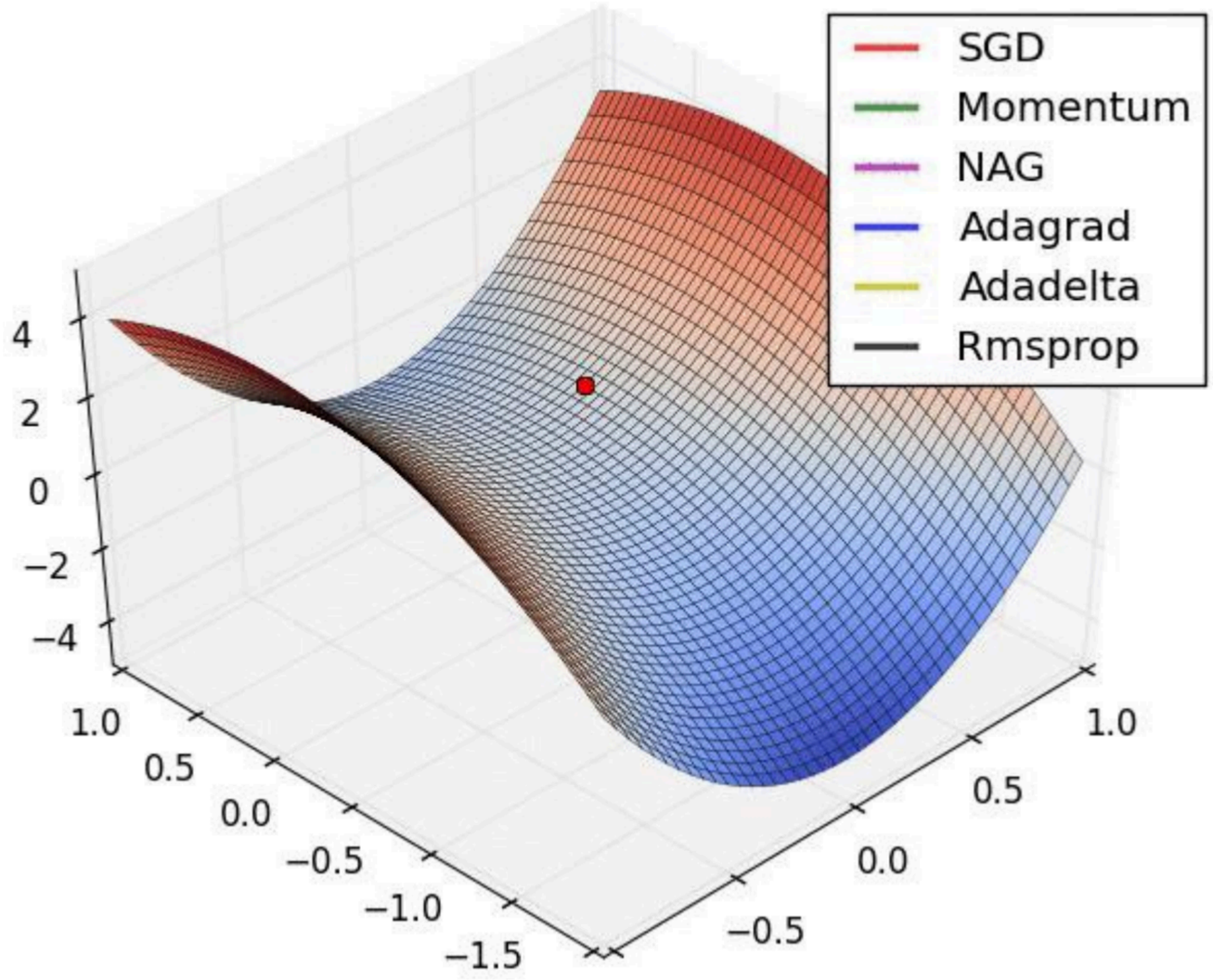


Image source: Aelc Radford, http://imgur.com/a/Hqolp

Behavior around a Saddle Point

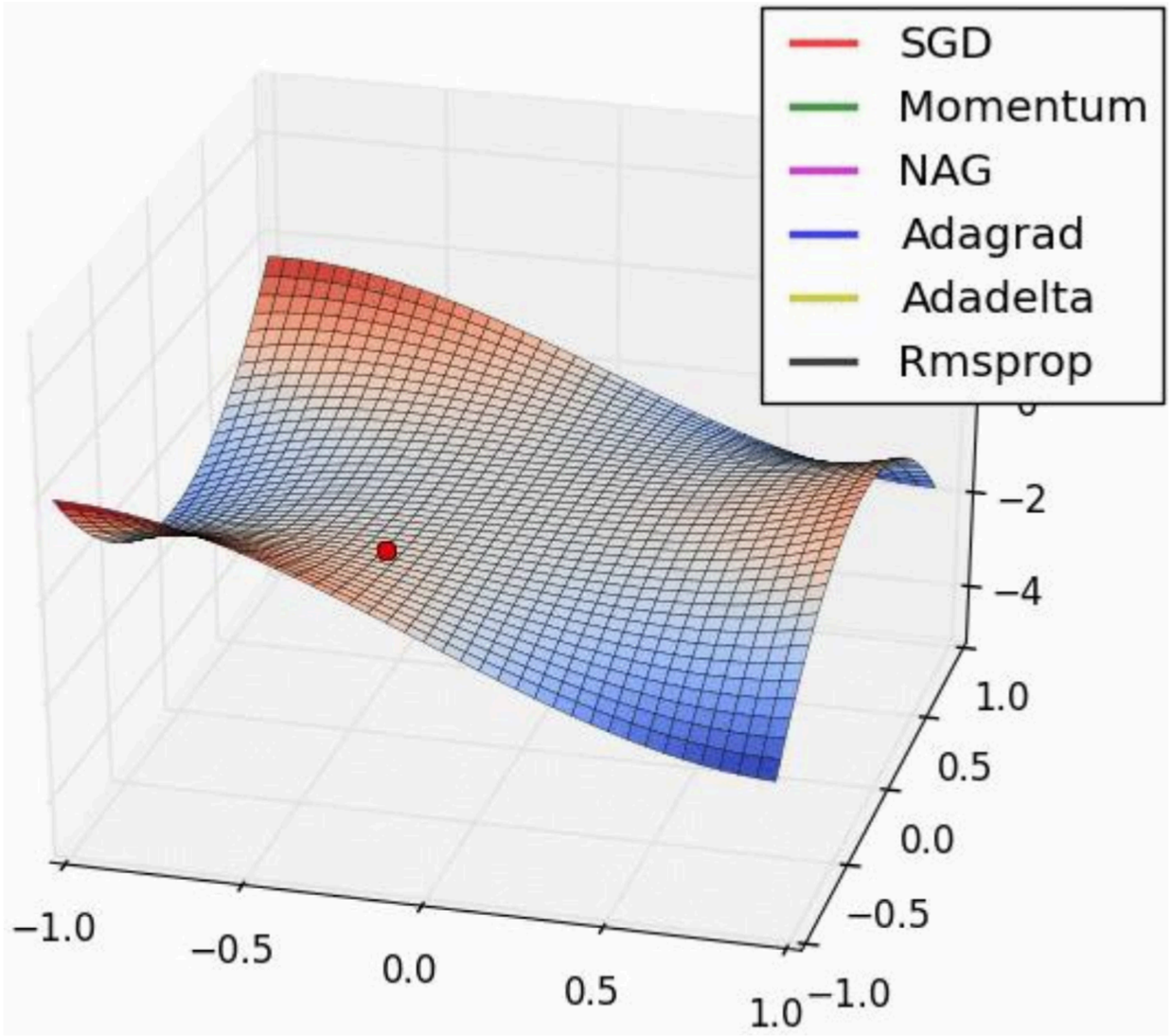


Image source: Aelc Radford, http://imgur.com/a/Hqolp

Visualization of Convergence Behavior

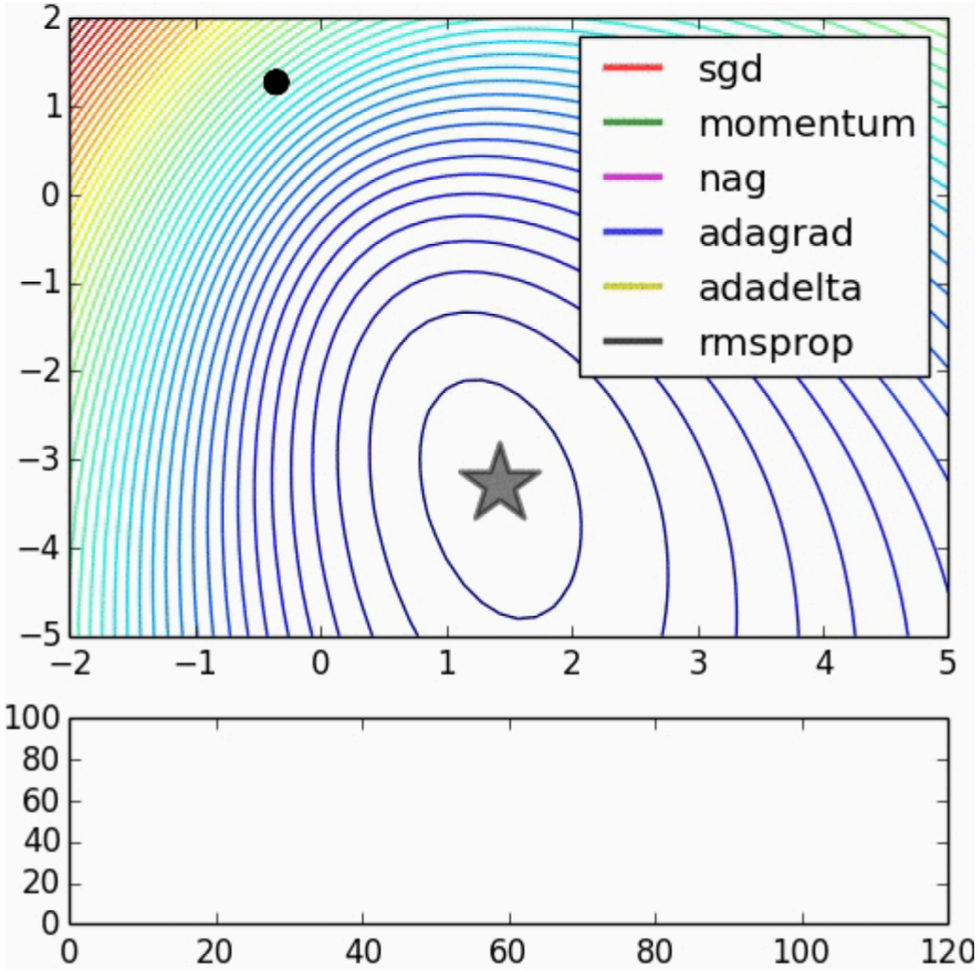
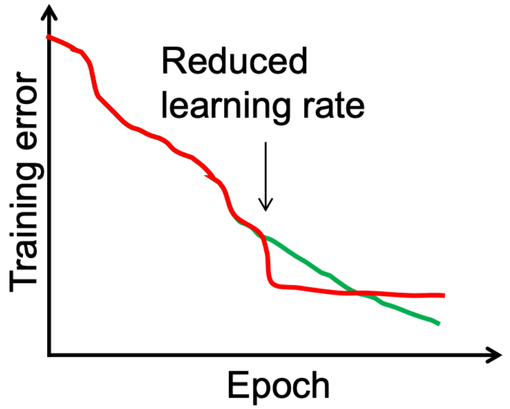


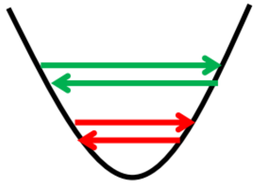
Image source: Aelc Radford, http://imgur.com/SmDARzn

Reducing the Learning Rate

Final improvement step after convergence is reached

- Reduce learning rate by a factor of 10.
- Continue training for a few epochs.
- Do this 1-3 times, then stop training.





Effect

- Turning down the learning rate will reduce the random fluctuations in the error due to different gradients on different mini-batches.

Be careful: Do not turn down the learning rate too soon!

- Further progress will be much slower/impossible after that.

Summary

Deep multi-layer networks are very powerful.

But training them is hard!

- Complex, non-convex learning problem
- Local optimization with stochastic gradient descent

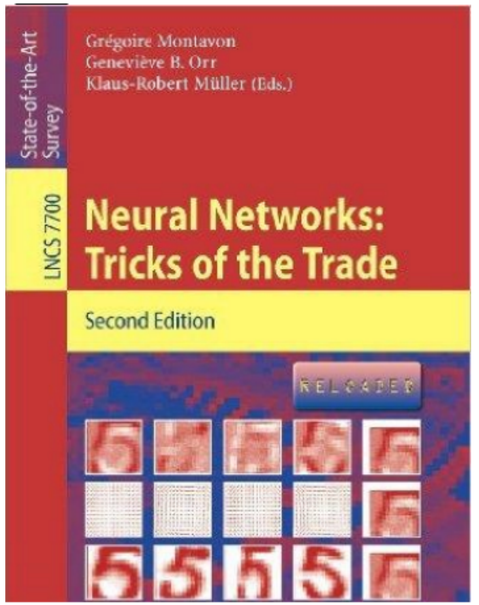
Main issue: getting good gradient updates for the lower layers of the network

- Many seemingly small details matter!
- Weight initialization, normalization, data augmentation, choice of nonlinearities, choice of learning rate, choice of optimizer,…
- In the following, we will take a look at the most important factors

References and Future Reading

More information on Neural Networks can be found in Chapter 1 and 7 of the book

G. Montavon, G. B. Orr, K-R Mueller (Eds.) Neural Networks: Tricks of the Trade Springer, 1998, 2012



Yann LeCun, Leon Bottou, Genevieve B. Orr, Klaus-Robert Mueller Efficient BackProp, Ch.1 of the above book., 1998