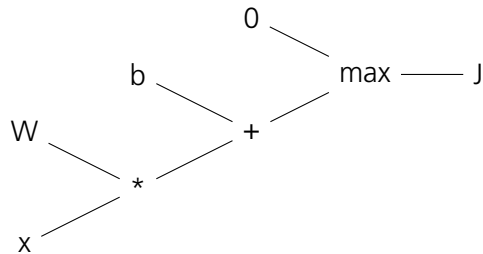
Practical overview of optimization of Deep Networks

Carl Åkerlindh

Backpropagation

Write your expression as an expression tree. Compute your objective during a forward pass. Use chain rule to compute gradient during a backward pass. For $J = \max(Wx + b)$ we get



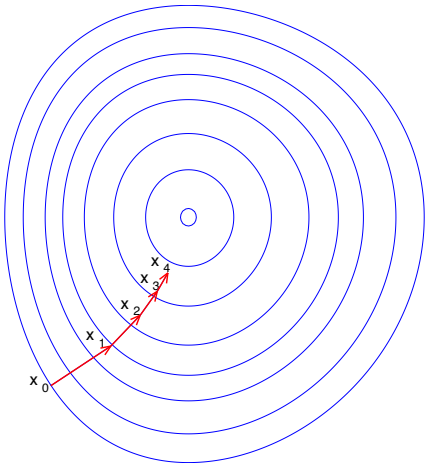
Useful resource: `http://www.psi.toronto.edu/~andrew/papers/matrix_calculus_for_learning.pdf`

Gradient descent

Vanilla gradient descent, also known as batch gradient descent.

$$\theta_{k+1} = \theta_k - \eta \nabla J(\theta_k)$$

Updates the parameters with gradient based on the entire data set.



Gradient descent

```
# gradient descent
for i in 1:num_epochs
  grad = eval_grad(loss, data, params)
  params = params - learning_rate * grad
end
```

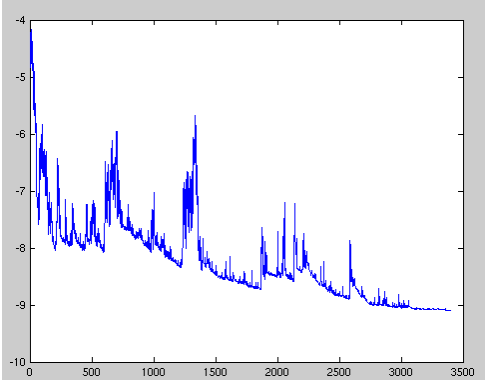
- Can be very slow
- Intractable if dataset does not fit in memory
- Not possible to perform online

Stochastic gradient descent

Online version of gradient descent.

$$\theta_{k+1} = \theta_k - \eta \nabla J(\theta_k; x_j, y_j)$$

Updates the parameters with approximated gradient based on single data point.



Stochastic gradient descent

```
# stochastic gradient descent
for i in 1:num_epochs
  shuffle!(data)
  for j in 1:lenght(data)
    grad = eval_grad(loss, data[j], params)
    params = params - learning_rate * grad
  end
end
```

- Usually faster convergence
- High variance

Mini-batch gradient descent

Best of both worlds.

$$\theta_{k+1} = \theta_k - \eta \nabla J(\theta_k; x_{j:j+n}, y_{j:j+n})$$

Updates the parameters with approximated gradient based on single data point.

Mini-batch gradient descent

```
# mini-batch gradient descent
for i in 1:num_epochs
  shuffle!(data)
  for j in 1:num_batches
    batch = get_batch(data, j, batch_size)
    grad = eval_grad(loss, batch, params)
    params = params - learning_rate * grad
  end
end
```

- Faster than batch gradient descent
- Reduced variance compared to (online) gradient descent

Guaranteed convergence $\neq$ fast convergence

- Choosing learning rate is very important, but non-trivial
- Pre-defined learning rate schemes can be used, but might not fit all data sets
- Make bigger updates for rarely occurring features, and smaller updates for more common ones
- Local minima and saddle points

Momentum

Prevents oscillations by accumulating the momentum of the gradient updates in the dimensions that does not change direction and vice versa.

$$v_{k+1} = \gamma v_k + \eta \nabla J(\theta_k)$$
$$\theta_{k+1} = \theta_k - v_{k+1}$$

Nestrov accelerated gradient

Similar to the momentum method, but evaluates the gradient in an approximation of the next parameter vector.

$$v_{k+1} = \gamma v_k + \eta \nabla J(\theta_k - \gamma v_k)$$
$$\theta_{k+1} = \theta_k - v_{k+1}$$

Adagrad

Adapts the update of individual parameters based on their importance. Let $g_{k+1,i} = \nabla J(\theta_i)$, the Adagrad update per parameter is then

$$\theta_{k+1} = \theta_k - \frac{\eta}{\sqrt{G_{k,ii} - \epsilon}} \nabla J(\theta_k)$$

where G_k is a diagonal matrix where each diagonal element $G_{k,ii}$ is the sum of squares of all previous gradients with respect to θ_i.

Pros and cons

- Eliminates need to tune learning rate manually
- Every term added to G is positive, so eventually the learning rate will become too small

Other variants of SGD

- Adadelta
- RMSprop
- Adam

Additional tricks

Curriculum learning Shuffling the dataset in the beginning of every optimization epoch.

Batch normalization Every mini-batch is normalized individually during training. A post-training step is then applied, where mean and variance is computed for the whole dataset.

Early stopping Free lunch according to Hinton. During training, compute error on a validation dataset, if it increases, stop.

Additional tricks

- Layer normalization
- Randomly select hyperparameters
- Dropout
- Gradient noise

Homework

Implement a training algorithm from scratch for any model introduced in this course, e.g. Autoencoder, Restricted Boltzmann Machine or Convolutional Neural Network.

Further reading I

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer Normalization". In: *arXiv.org* (July 2016). arXiv: 1607.06450v1 [stat.ML].

Yoshua Bengio. "Practical Recommendations for Gradient-Based Training of Deep Architectures". In: *Neural Networks: Tricks of the Trade*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 437–478.

Geoffrey E Hinton. "A Practical Guide to Training Restricted Boltzmann Machines". In: *Neural Networks: Tricks of the Trade*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 599–619.

Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv.org* (Feb. 2015). arXiv: 1502.03167v3 [cs.LG].

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in neural information …* (2012), pp. 2951–2959.

Further reading II

Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research (JMLR)* 15 (2014), pp. 1929–1958.

I Sutskever et al. "On the importance of initialization and momentum in deep learning." In: *ICML (3)* (2013).

Tijmen Tieleman. "Training restricted Boltzmann machines using approximations to the likelihood gradient". In: *the 25th international conference*. New York, New York, USA: ACM Press, 2008, pp. 1064–1071.

LUND
UNIVERSITY