Outline

- Optimization
- CNN basics
- Examples of CNN Architectures
- Applications of CNN

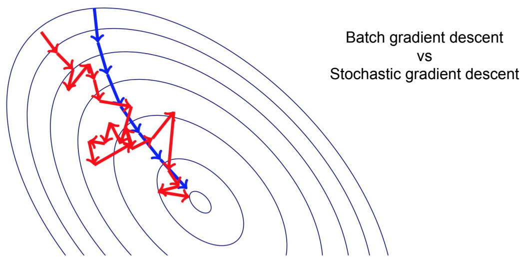
Optimization

- Stochastic Gradient Descent
- Momentum Method
- Adaptive Learning Methods
- (AdaGrad, RMSProp, Adam)

Broadly applicable for many ML methods

- Batch Normalization ← specific to neural nets

Stochastic Gradient Descent



Slide credit: Shubhendu Trivedi, Risi Kondor

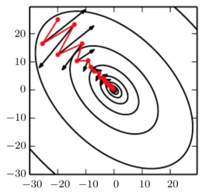
Limitations of GD and SGD

- GD and SGD suffer in the following scenarios:
  - Error surface has high curvature
  - We get small but consistent gradients
  - The stochastic gradients are very noisy (for SGD)

Slide credit: Shubhendu Trivedi, Risi Kondor

Momentum

- The Momentum method is a method to accelerate learning using GD or SGD

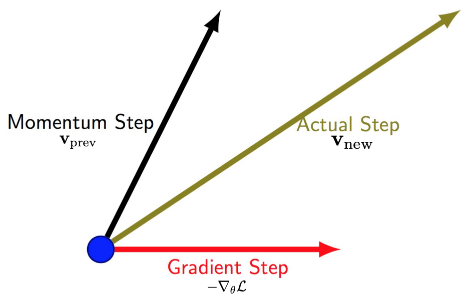


- Illustration of how momentum traverses such an error surface better compared to gradient descent
- Visualization of GD with momentum

Slide credit: Shubhendu Trivedi, Risi Kondor

Momentum

$$\mathbf{v}_{\text{new}} = \alpha \mathbf{v}_{\text{prev}} - \epsilon \nabla_\theta \left(\mathcal{L}\left(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)} \right) \right)$$



Visualization of GD with momentum

Slide credit: Shubhendu Trivedi, Risi Kondor

Adaptive Learning Methods: Motivation

- Until now we assigned the same learning rate to all features
  - If the features vary in importance and frequency, is this a good idea?
  - Probably not!
- Popular methods for adaptive learning rates:
  - AdaGrad, RMSProp, Adam, etc.
- High-level idea:
  - Discount the learning rate for each parameter by dividing with the "amplitude" (running average) of the gradient for that parameter.
  - Training is more stable and robust to noisy gradients and the choice of initial learning rates (e.g., even a reasonably large initial learning rate works)

Optimization with SGD and its variants

SGD: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

Momentum: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \mathbf{v}$

AdaGrad: $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ then $\Delta\theta - \leftarrow \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \Delta\theta$

RMSProp: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1-\rho) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ then $\Delta\theta - \leftarrow \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \Delta\theta$

ADAM: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1-\rho_1) \hat{\mathbf{g}}$

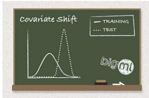
$\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1-\rho_2) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$

$\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\rho_2^t}$ then $\Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$ then $\theta \leftarrow \theta + \Delta\theta$

- Visualization:
  - Visualization of optimizers
  - Distill: why momentum really works

Slide credit: Shubhendu Trivedi, Risi Kondor

Batch Normalization



- **BN is specific to neural networks**
- We have a recipe to compute gradients (backpropagation) and update all parameters (SGD, adaptive learning rate methods, etc.)
- **Challenge: Internal Covariate Shift**
  - Implicit assumption during backpropagation: layer inputs remain unchanged.
  - Reality: Simultaneous updates across layers alter distributions.
  - Consequences:
    - Shifts in activation distributions across layers.
    - Training instability: unstable training, longer convergence times, suboptimal performance.
- **Insights: Addressing internal covariate shift can significantly enhance training efficiency.**

Batch normalization standardizes inputs to each layer…

- …which helps to stabilize training

- Consider standardizing the input to the input layer, i.e. our data
  - For minibatch mean μ_β and standard deviation σ_β^2, we can normalize each input x_i to

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

(ϵ added for numerical stability)

which shifts our input to a distribution w/ mean = 0 and std dev = 1

  - Output of batch normalization (with γ, β being learnable parameters for the BN layer):

$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

- Batch normalization extends this idea to the input of *every* layer, not just the input layer
  - But after going through previous layers and activations, the input does not necessarily reflect the original input distribution
  - To reflect the true distribution of the data, keep track of scale γ and shift β for each weight

Batch Normalization: forward prop

- Normalize distribution of each input feature in each layer across each minibatch to $N(0, 1)$
- Learn the scale and shift

- After training, at test time: Use running averages of μ and σ collected during training, use these for evaluating new input x

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Batch Normalization: backpropagation

- Differentiable via chain rule

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum\nolimits_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}} = \sum\nolimits_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum\nolimits_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum\nolimits_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.