High Level Computer Vision

Exercise 3 | SS 2018

28/05/2018   - Rakshith Shetty
Tentative exercise and project schedule

- Ex4 → Friday 1st June. Due on Fri. 15th June.
- Project Intro - Monday 11th June.
- Student project proposal presentation → Monday 18th June.
  - From 1pm - 4pm.
- Student project mid-presentation → Monday July 2nd. (again 1-4pm)
- Final presentation → Monday July 23rd + maybe Tuesday.
- Final report due → Friday July 27th.
Exercise 3 -- Implement and train neural networks

● Implement a feed-forward neural network to perform digit classification

● You will train this network using backpropagation.

● Derive (optional) and implement the algorithm.

● Verify if the gradient computed is correct using numerical gradients.

● Finally, train the network using
  ○ Gradient descent.
  ○ Stochastic gradient descent.
Neural networks are function approximators

- Universal function approximators
  - Networks with at least one hidden layer can approximate any function*

- Previously - Feature extract + Classifier

- Now → Let the neural network learn this from scratch.
Function fitting - convex optimization

- Need a loss function to measure the task.
- Smooth convex loss-functions are great!
- We will use gradient descent to optimize our function approximation.
- Compute gradients w.r.t. to the loss and change the parameters in the direction of steepest descent.

Gif from https://hackernoon.com/life-is-gradient-descent-880c60ac1be8
Loss function - Cross Entropy loss

- Cross entropy loss

\[ J(u) = \sum_{k=1}^{K} (-y_k \log u_k - (1 - y_k) \log(1 - u_k)) \]

- Measures the conditional entropy between predicted label and the true label.

- Lower loss implies predicted and true labels are close to each other.
Side-note → Differentiability

If $f$ is smooth and $g$ is smooth, then $g \circ f$ is also smooth.

“Smooth”:
• differentiable, twice differentiable, ..., infinitely differentiable ($C^\infty$).
• continuously differentiable ($C^1$), twice continuously differentiable ($C^2$), ..., infinitely differentiable ($C^\infty$).

Our neural network is $C^\infty$. 

Slide credit - Seong Joon Oh
Backpropagation

- How do you change the weights to optimize the loss?
  - Since we use gradient descent, we compute the gradient of the loss function w.r.t each weight.

- Simply apply chain rule to compute the gradients.

\[
\frac{\partial (f \circ g)_i(x)}{\partial x_k} \bigg|_{x=u} = \sum_{j=1}^{M} \frac{\partial f_i(y)}{\partial y_j} \bigg|_{y=g(u)} \frac{\partial g_j(x)}{\partial x_k} \bigg|_{x=u}
\]
Example

\[ f(y) = \sum_{p=1}^{3} y_p^2 \]

\[ g_p(x) = \sum_{q=1}^{2} w_{pq} x_q^2 \]

\[ f : \mathbb{R}^3 \rightarrow \mathbb{R} \]
\[ g : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \]
Example, continued

\[
\frac{\partial f(y)}{\partial y_j} = \frac{\partial}{\partial y_j} \sum_{p=1}^{3} y_p^2 = \sum_{p=1}^{3} \frac{\partial}{\partial y_j} y_p^2 = \sum_{p=1}^{3} 2y_p \delta_{jp} = 2y_j
\]
Example, continued

\[
\frac{\partial g_j(x)}{\partial x_k} = \frac{\partial}{\partial x_k} \sum_{q=1}^{2} w_{jq} x_q^2
\]

\[
= \sum_{q=1}^{2} \frac{\partial}{\partial x_k} (w_{jq} x_q^2)
\]

\[
= \sum_{q=1}^{2} (2w_{jq} x_q \delta_{kq})
\]

\[
= 2w_{jk} x_k
\]
\[
\left. \frac{\partial (f \circ g) (x)}{\partial x_k} \right|_{x=u} = \sum_{j=1}^{3} \left. \frac{\partial f(y)}{\partial y_j} \right|_{y=g(u)} \left. \frac{\partial g_j(x)}{\partial x_k} \right|_{x=u}
\]

\[
= \sum_{j=1}^{3} 2g_j(u) (2w_{jk}u_k)
\]

\[
= 4u_k \sum_{j=1}^{3} w_{jk}g_j(u)
\]

\[
= 4u_k \sum_{j=1}^{3} w_{jk} \sum_{q=1}^{2} w_{jq}u_q^2
\]
Numerical Gradients

- Wiggle the parameters and compute gradients numerically
  \[
  \frac{\partial \tilde{J}}{\partial \theta_p}(\theta) \approx \frac{\tilde{J}(\theta + \epsilon e_p) - \tilde{J}(\theta - \epsilon e_p)}{2\epsilon}
  \]

- Can do this for all parameters in the network.

- Too slow for practical use in training but great for verifying backpropagation equations.
Batch gradient descent

- Once you have the gradients you can update the parameters.

\[ \nu \leftarrow -\alpha \nabla_{\theta} \tilde{J}(\theta^{(t-1)}) \; ; \]
\[ \theta^{(t)} \leftarrow \theta^{(t-1)} + \nu ; \]

-ve sign to decrease the loss  Averaged over all training samples

- Guaranteed to converge to local minima.
- Very slow since parameters are updated once for each pass on the data.
- Large memory consumption on large datasets.
Momentum

- Adding momentum can speed up the training
  \[ v \leftarrow \beta v - \alpha \nabla_{\theta} \tilde{J}(\theta^{(t-1)}); \]
  \[ \theta^{(t)} \leftarrow \theta^{(t-1)} + v; \]

- It also helps overcome local minima to an extent.

- Parameter should be carefully chosen.

Figure Credit: AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors by Alex J. Champandard
**Stochastic Gradient descent**

- Compute the gradients for every sample and update instantly.
- Fast and low memory consumption.
- Can be noisy.
- But noise is good! Can again avoid getting stuck in local minima.
- Better generalization properties*

Visualization from [https://wikidocs.net/3413](https://wikidocs.net/3413)
Submission

- Next week, Sunday midnight (03/06/2018 23:59)
- Send to rshetty@mpi-inf.mpg.de
- One zip file per team
- Solutions next tutorial

Questions?