Using coordinate descent to train 1 hidden-layer neural nets

Volodymyr Lyubinets

10/11/2017

Coordinate Descent

procedure $\mathrm{CD}(f, x_0, N_{iter})$

  while $0 \leq k < N_{iter}$ **do**

    Pick a coordinate $i \in [n]$.

    Update $x_{k+1} = x_k - s_k e_i$ for some step-size s_k.

- We can also update a block of weights.

Coordinate Descent Choice Rules

Strategies for choosing the coordinate:

- Cyclic
- Random with uniform distribution
- Random according to some crafted distribution
- Greedy according to some criterion (e.g. steepest direction)

Existing Coordinate Descent Results

For convex functions (+ Lipschitz gradient) we can compute bounds on expected improvement of one step and provide convergence rates (in Nesterov, ‘10). With even stronger assumption (strong convexity) it is shown that RCDM has linear rate of convergence.

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{1}{2}\sigma(f)\|y - x\|^2$$

Strong convexity assumption

Existing Coordinate Descent Results

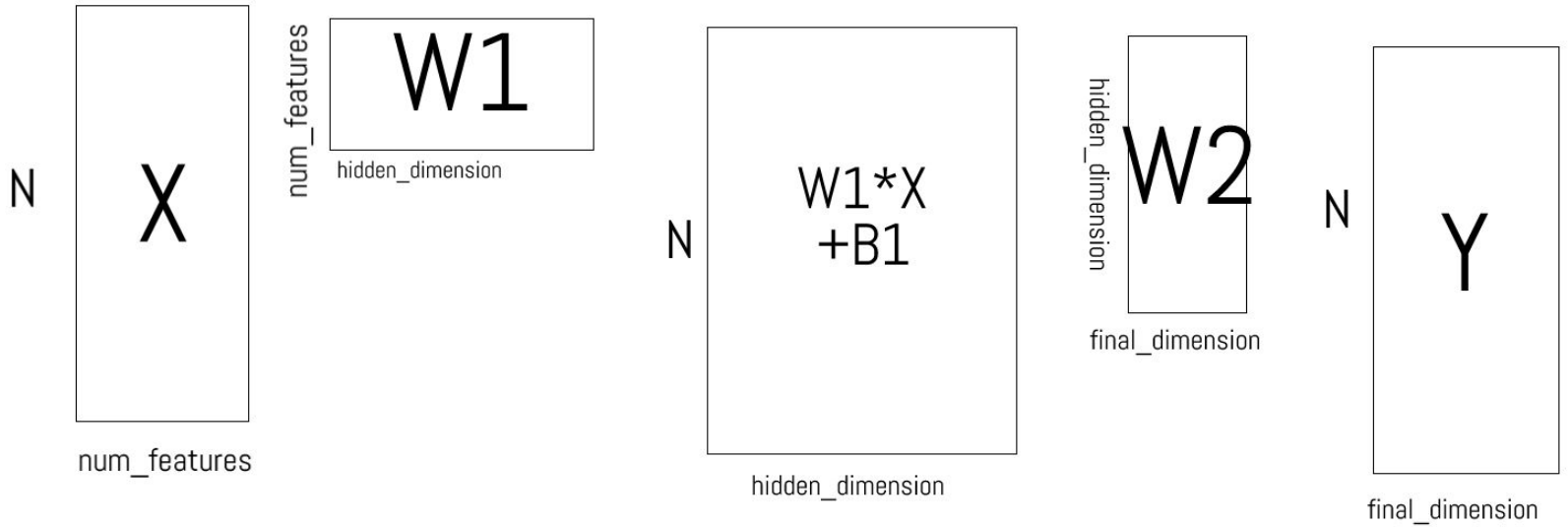
Csiba and Richtarik (‘17) analyze a wider range of functions, including “general nonconvex”.

7 General Nonconvex Functions

In this section we establish a generic convergence result applicable to general nonconvex functions. This is done at the expense of losing global optimality: we will show that either $\lambda(\mathbf{x}^k)$ gets small, or that $F(\mathbf{x}^k)$ is close to the global minimum $F(\mathbf{x}^*)$. Recall that in the smooth case ($g = 0$) we have $\lambda(\mathbf{x}^k) = \frac{1}{2}\|\nabla f(\mathbf{x}^k)\|^2$.

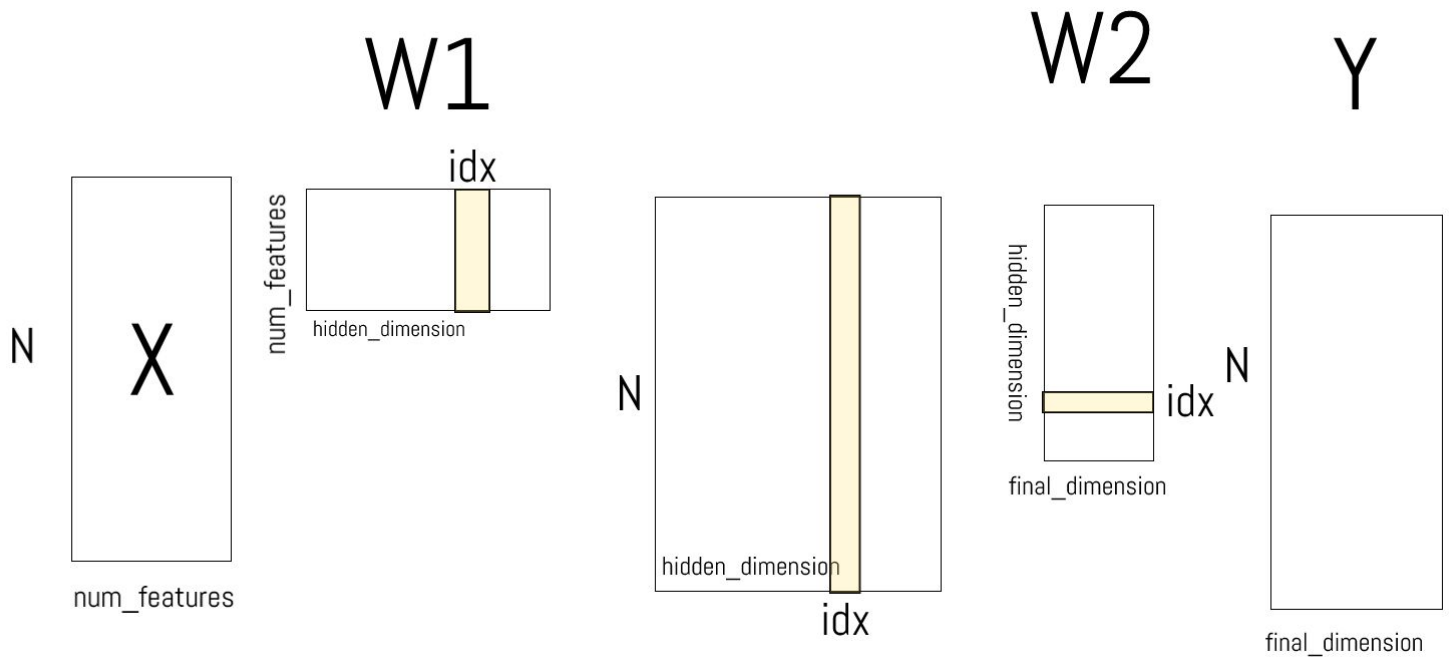
1 hidden-layer neural nets

f(X) = W2 * f(W1 * X + B1) + B2, where f is the activation function



Using CD for 1 hidden-layer neural nets

We only update weights that correspond to one “neuron” of hidden layer



Using CD for 1 hidden-layer neural nets

Update weights that correspond to M1[:, idx] - W2[idx, :] and W1[:, idx].

Use “regular” GD backpropagation, but only do it to the columns in question, e.g:

Regular: grads['W2'] = M1.T.dot(probs) + 2 * self.reg * W2

Ours: grad_W2 = M1[:, idx].T.dot(probs) + 2 * self.reg * self.params['W2'][idx, :]

Using CD for 1 hidden-layer neural nets: code

```
# Subtract reg. contribution of weights to be updated
self.reg_loss -= self.reg * (self.params['W2'][hlidx, :] * self.params['W2'][hlidx, :]).sum()
self.reg_loss -= self.reg * (self.params['W1'][:, hlidx] * self.params['W1'][:, hlidx]).sum()

M1 = self.params['M1']

# Subtract old contribution from scores
scores -= np.outer(M1[:, hlidx], self.params['W2'][hlidx, :])

grad_W2 = M1[:, hlidx].T.dot(probs) + 2 * self.reg * self.params['W2'][hlidx, :]

hid_grad = probs.dot(self.params['W2'][hlidx, :].T)
hid_grad[M1[:, hlidx] <= 0] = 0

grad_W1 = X.T.dot(hid_grad) + 2 * self.reg * self.params['W1'][:, hlidx]

# Update weights
self.params['W2'][hlidx, :] -= self.learning_rate * grad_W2
self.params['W1'][:, hlidx] -= self.learning_rate * grad_W1

# Recompute M1
M1[:, hlidx] = X.dot(self.params['W1'][:, hlidx])
M1[:, hlidx] = np.maximum(M1[:, hlidx], np.zeros(M1[:, hlidx].shape)) # Apply ReLU
self.params['M1'] = M1 # Save the hidden state

# Update scores
scores += np.outer(M1[:, hlidx], self.params['W2'][hlidx, :])
```

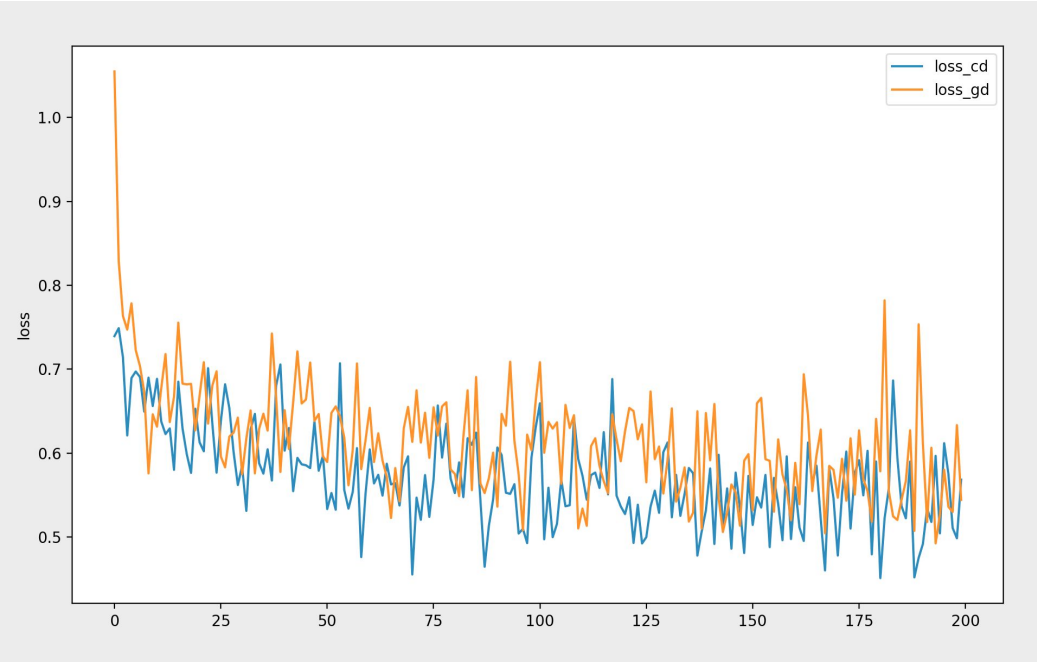
Using CD for 1 hidden-layer neural nets: properties

- A backward pass of GD updates all W1 and W2 weights - that's what hidden_layer_size partial updates will do.
- We need to do forward pass only once for each hidden_layer_size partial updates.
- Has potential to be parallelized.

Using CD for 1 hidden-layer neural nets: results

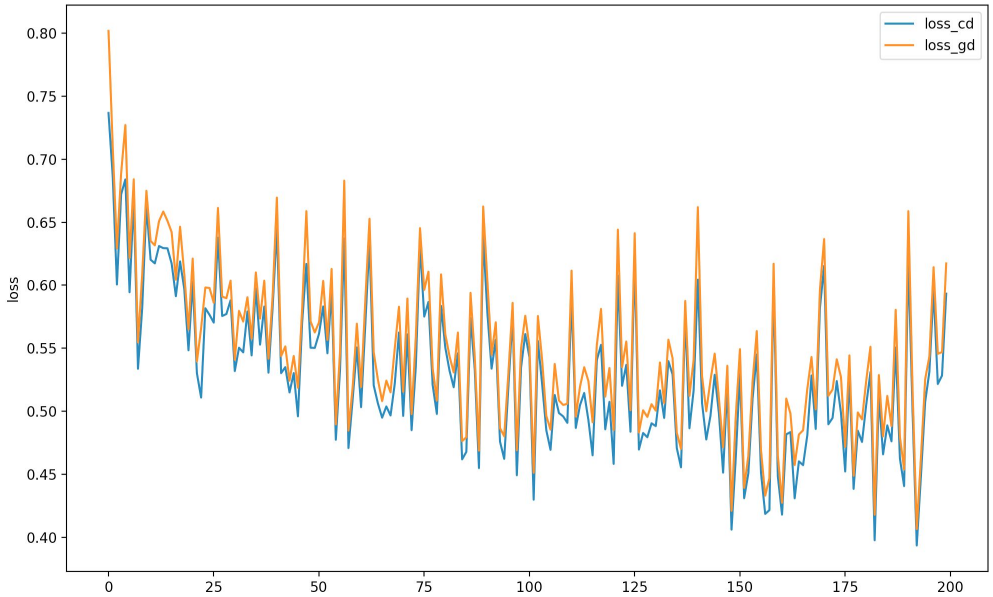
- Dataset: Higgs boson from kaggle, activation function is ReLU
- We treat num_hidden_layer steps of our algorithm as one step of GD

Per-batch loss, learning rate 0.1, random choice



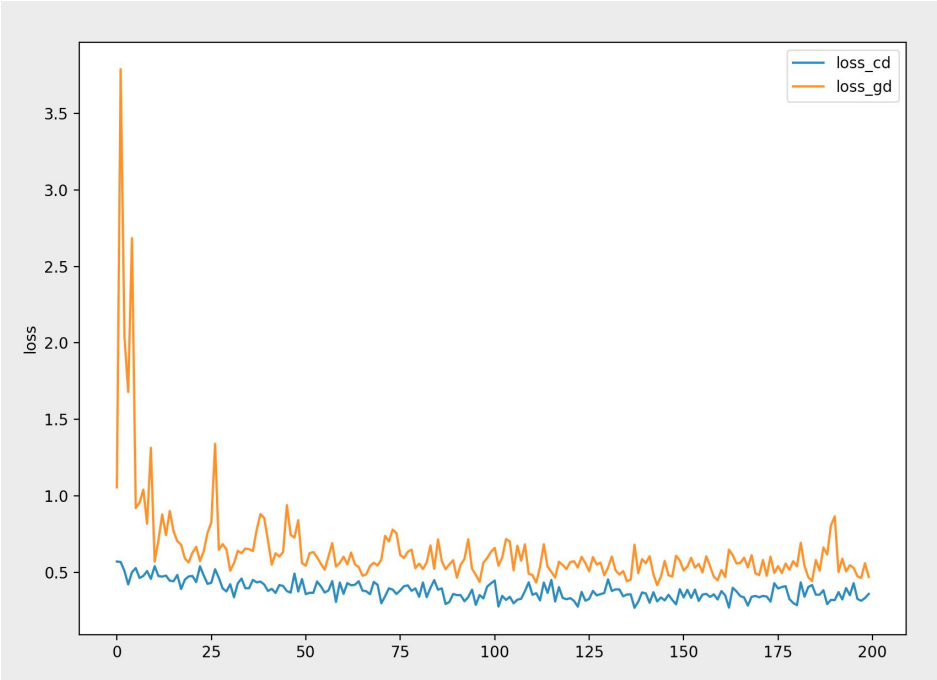
Validation scores end up being the same.

Per-batch loss, learning rate 0.1, cyclic choice



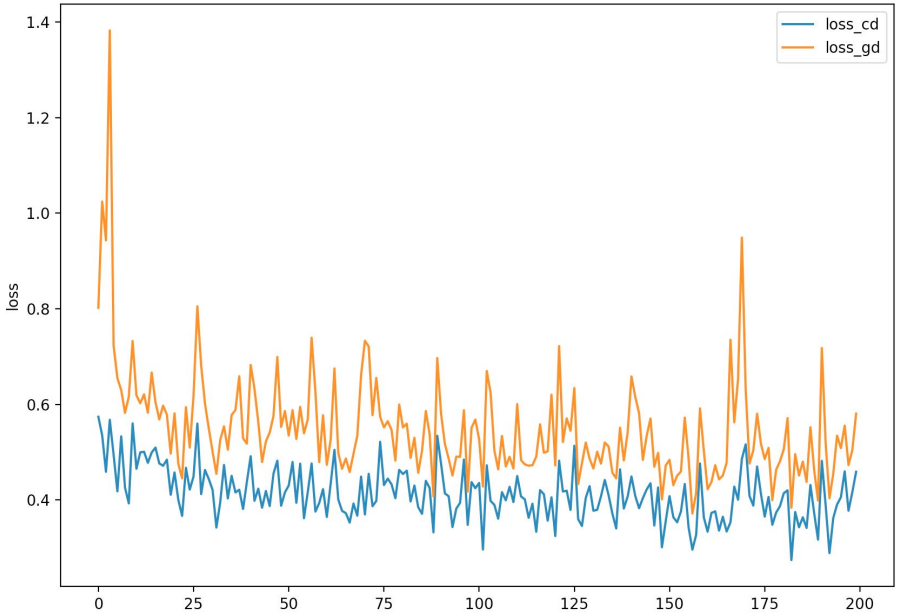
Validation scores end up being the same.

Per-batch loss, learning rate 1, random choice



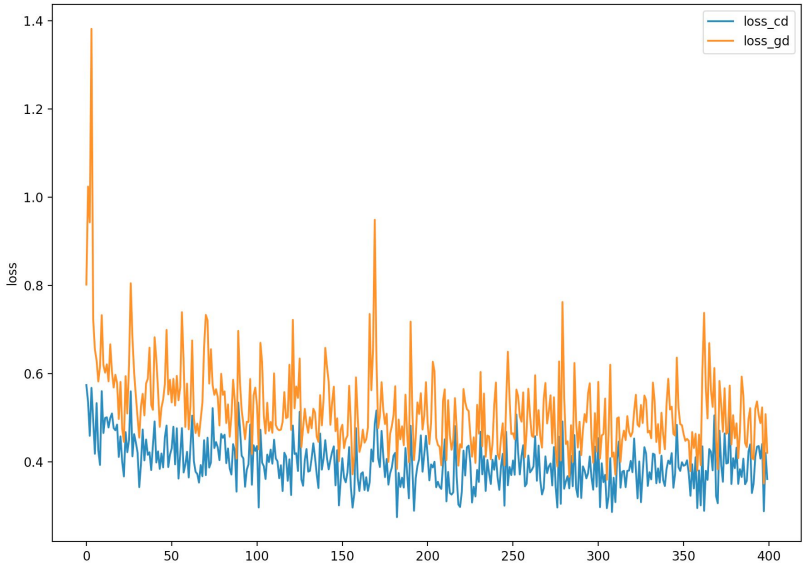
Validation score for a new method is noticeably lower.

Per-batch loss, learning rate 1, cyclic choice




```
Train results (new method) 160265 out of 200000 are correct (80.1325%).
Train results (GD) 159182 out of 200000 are correct (79.591%).
Validation results (new method) 40107 out of 50000 are correct (80.214%).
Validation results (GD) 39797 out of 50000 are correct (79.594%).
```

Per-batch loss, learning rate 1, cyclic choice




```
Train results (new method) 160844 out of 200000 are correct (80.422%).
Train results (GD) 159533 out of 200000 are correct (79.7665%).
Validation results (new method) 40284 out of 50000 are correct (80.568%).
Validation results (GD) 39894 out of 50000 are correct (79.788%).
```

Conclusions & Future work

1) Has potential to achieve faster convergence (per iteration).
2) Hard to prove precise guarantees because of non-convexity.
3) TODO: Come up with a way to compare performance of this update strategy vs vanilla GD with backprop.
4) TODO: Explore performance on a regression dataset.
5) TODO: Explore more complicated choice methods.
6) TODO: Extend to arbitrary depth.