SVM hyperparameter tuning

Hyperparameters range	init W	batch size	best lr	best reg	num iters	val_acc	test_acc	visualized weights (0:random,10:very good)	time
lr, reg = 10**Uniform(-8, -5, 20)	`W = 1.0 * 0.0001 * np.random.randn(dim, num_classes)`	200	6.5388e-07	1.6053e-08	100	40.0%	35.2%	1	2m
			1.1874e-07	7.1516e-08	200	39.7%	35.5%	1	4m
			2.2479e-07	7.5640e-08	500	40.8%	37.0%	1.5	8m
			1.6587e-07	5.6261e-06	2000	42.2%	39.5%	2	25m
		600	3.7813e-07	1.4699e-06	500	41.7%	38.4%	1.5	22m
	`W = 1.5 * 0.0001 * np.random.randn(dim, num_classes)`	200	2.1245e-07	1.1011e-07	100	38.9%	33.9%	0	2m
			2.9340e-07	4.2148e-06	200	40.0%	37.0%	0	2m
			1.4184e-07	7.6640e-08	400	40.2%	36.7%	0.5	6m
			1.6944e-07	7.9099e-08	500	40.2%	38.5%	2	8m
			2.3325e-07	3.6891e-06	600	40.9%	35.9%	1.5	10m
			2.9044e-07	1.8757e-08	2000	41.5%	39.0%	2	28m
	`W = 0.001 * np.random.randn(dim, num_classes)`		9.2751e-06	2.2229e-07	50	33.6%	31.2%	0	2m
			2.2009e-06	1.3759e-07	200	36.6%	32.7%	0	2m 30s
			2.2926e-06	1.3355e-07	500	38.8%	35.2%	0.5	8m
lr = 10**Uniform(-8, -3, 50) reg = 10**Uniform(-8, -3, 20)	`W = 0.001 * np.random.randn(dim, num_classes)`	200	9.2930e-06	9.4511e-06	50	33.4%	29.8%	0.5	4m
			4.6023e-06	1.4493e-04	100	35.9%	34.2%	0	5m 30s
			7.0632e-05	1.8214e-06	200	35.1%	31.4%	2	9m
			7.3988e-07	2.4475e-08	500	35.3%	33.2%	0	19m 30s
lr = 10**Uniform(-8, 1, 50) reg = 10**Uniform(-8, 1, 8)	`W = 0.001 * np.random.randn(dim, num_classes)`	200	2.3653e-05	8.7578e-07	50	33.8%	30.7%	4.5	2m
lr = 10**Uniform(-8, 1, 50) reg = 10**Uniform(-8, 1, 50)	`W = 0.001 * np.random.randn(dim, num_classes)`	200	6.7213e-06	3.4157e-05	1500	37.9%	33.5%	1.5	2h

lr = 10** Linspace(-8, 1, 50) reg = 10** Linspace(-8, 1, 8)	`W = 0.001 * np.random.randn(dim, num_classes)`	200	5.1794e-01	1.9306e-07	50	33.5%	30.5%	4.5	2m
			3.7275e-06	1.3894e-03	200	37.1%	32.2%	0	3m
lr = 10** Linspace(-8, -5, 40) reg = 10** Linspace(0, 4, 20)	`W = 0.001 * np.random.randn(dim, num_classes)`	200	3.4551e-06	6.1584e+03	50	33.4%	31.6%	1.5	3m
lr = 10** Linspace(-8, -5, 80) reg = 10** Linspace(0, 4, 40)	`W = 0.001 * np.random.randn(dim, num_classes)`	200	2.4683e-06	1.1937e+03	50	34.0%	31.6%	0	12m
lr = 10** Linspace(-8, -5, 20) reg = 10** Linspace(0, 4, 20)	`W = 0.001 * np.random.randn(dim, num_classes)`	2000	4.8329e-06	2.9763e+01	50	32.9%	29.0%	0	7m
			5.4555e-07	3.7929e+03	500	39.6%	39.0%	0.5	1h 7m
		4000	1e-05	2.6366e+00	50	33.1%	29.3%	0.5	14m
			7.8476e-07	3.7926e+03	500	40.8%	38.7%	3	2h 7m
lr = 10** Linspace(-7.5, -6.5, 25) reg = 10** Linspace(3.8, 4.3, 25)	`W = 0.001 * np.random.randn(dim, num_classes)`	2000	2.6101e-07	6.9449e+03	2500	40.7%	38.3%	5	7h 36m
lr = 10** Linspace(-7.5, -6.5, 25) reg = 10** Linspace(3.8, 4.3, 25)	`W = 0.001 * np.random.randn(dim, num_classes)`	2000	1.2589e-07	1.4677+04	1500	40.2%	37.9%	6	49m

Hyperparameters: learning rate (lr), regularization term (reg);
Train data shape: (49 000, 32, 32, 3);
Validation data shape: (1 000, 32, 32, 3);
Test data shape: (1 000, 32, 32, 3);

The experiments were performed on an Asus laptop with the following characteristics:
Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80 GHz;
RAM: 16.0 GB.

Examples of visualized SVM weights:

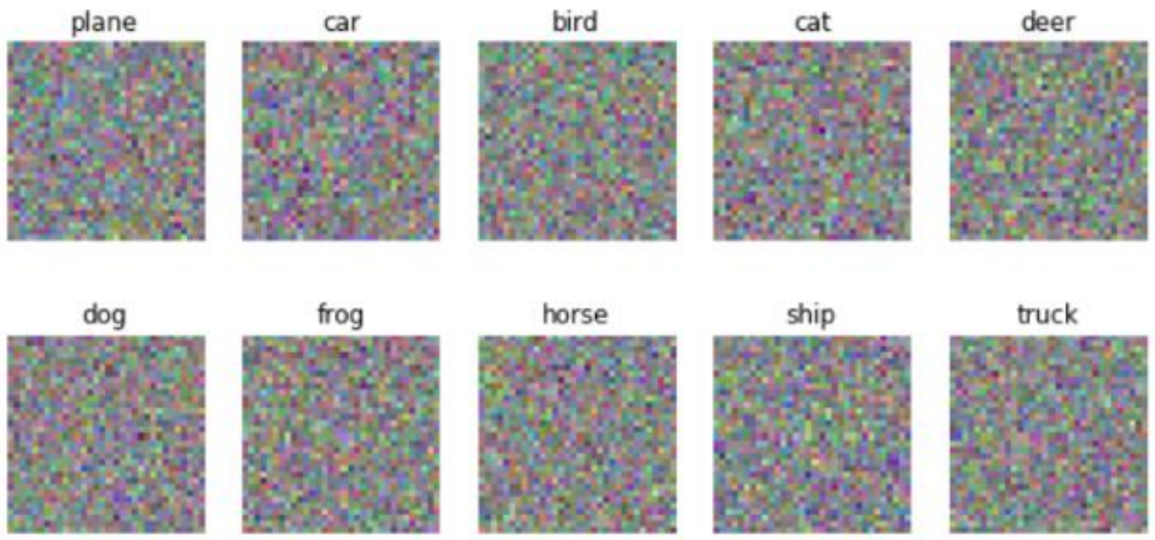

Figure 1. SVM weights (0: random)

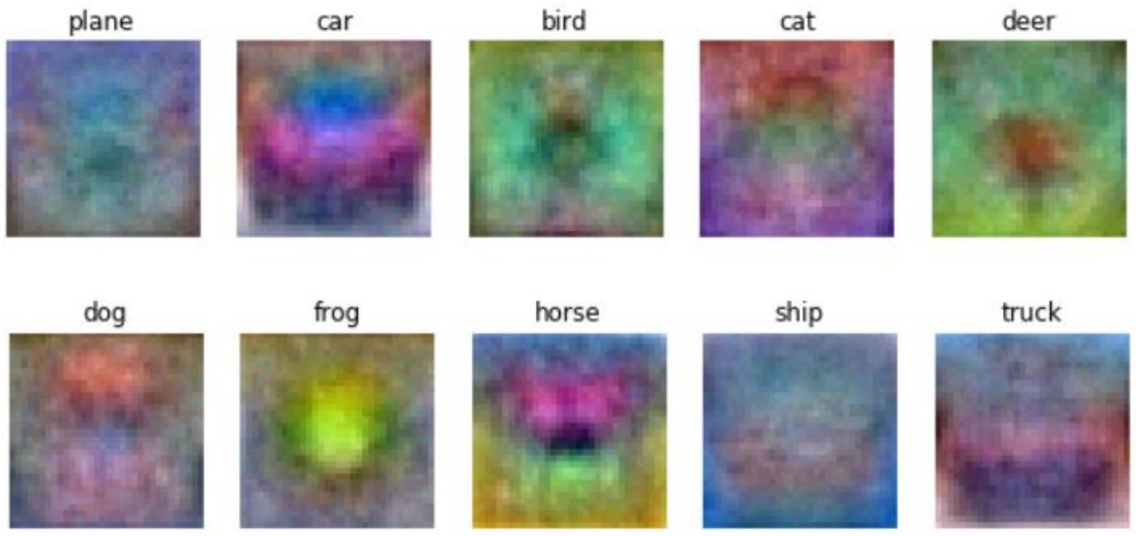

Figure 2. SVM weights (6: good)