CSE 559A: Computer Vision

Fall 2018: T-R: 11:30-1pm @ Lopata 101

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Course Staff: Zhihao Xia, Charlie Wu, Han Liu

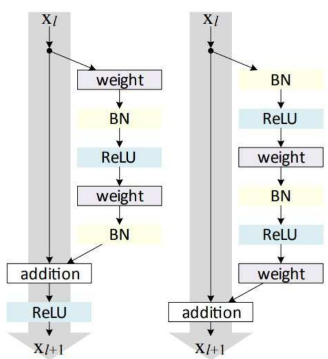
http://www.cse.wustl.edu/~ayan/courses/cse559a/

November 20, 2018

GENERAL

- Problem Set 5: Deadline Extended to Dec 4th.
- Recitation on Nov 30th (Friday after Thanksgiving)

BATCH NORMALIZATION



(a) original (b) proposed

He et al., "Identity Mappings in Deep Residual Networks". 2016.

REGULARIZATION

- Given a limited amount of training data, deep architectures will begin to overfit.
- **Important**: Keep track of training and dev-set errors
  Training errors will keep going down, but dev will saturate. Make sure you don't train to a point when dev errors start going up.
- So how do we prevent, or delay, overfitting so that our dev performance increases ?

Solution 1: Get more data.

REGULARIZATION

Data Augmentation

- Think of transforms to the images that you have that would still keep them in the distribution of real images.
- Typical Transforms
  - Scaling the image
  - Taking random crops
  - Applying Color-transformations (change brightness, hue, saturation randomly)
  - Horizontal Flips (but not vertical)
  - Rotations upto +- 5 degrees.
- Are a good way of getting more training data for 'free'.
- Teaches your network to be invariant to these transformations ....
- Unless your output isn't. If your output is a bounding box, segmentation map, or other quantities that would change with these augmentation operations, you need to apply them to the outputs too.

REGULARIZATION

Weight Decay

- Add a squared or absolute value penalty on all weight values (for example, on each element of every convolutional kernel, matmul matrix) except biases. $\sum_i w_i^2$ or $\sum_i |w_i|$
- So now your effective loss is $L' = L + \lambda \sum_i w_i^2$
- How would you train for this ?
  - Let's say you use backprop to compute $\nabla_{w_i} L$.
  - What gradient would you apply to your weights ? What is $\nabla_{w_i} L'$?

$$\nabla L' = \nabla L + 2\lambda w_i$$

- So in addition to the standard update, you will also be subtracting a scaled version of the weight itself.
- What about for $L' = L + \lambda \sum_i |w_i|$?

$$\nabla L' = \nabla L + \lambda Sign(w_i)$$

REGULARIZATION

Regularization: Dropout

- Key Idea: Prevent a network from "depending" too much on the presence of a specific activation.
- So, randomly drop these values during training.

g =Dropout(f,p): f and g will have the same shape.

Different behavior during training and testing.

- Training
- For each element f_i of f,
  - Set $g_i = 0$ with probability p, and $\frac{f_i}{(1-p)}$ with probability $(1-p)$
- Testing: $g_i = f_i$
- Why does this make sense ? Because in *expectation*, our value during training and test will be the same.
- Dropout is a layer. You will backpropagate through it ! How ?

REGULARIZATION

Regularization: Dropout

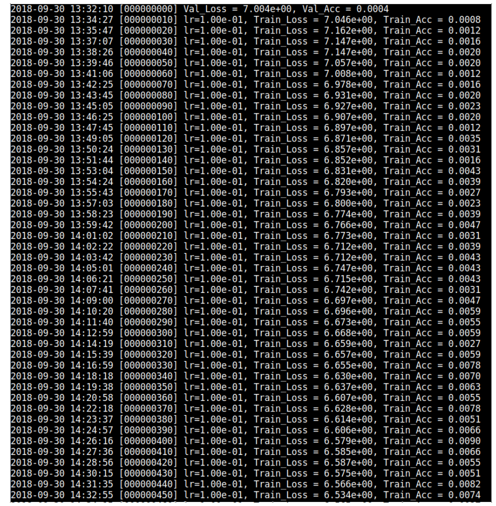
- Write the function as $g = f \cdot \epsilon$
  - Here ϵ is a random array same size as f, with values 0 and $1/(1-p)$ with probability p and $(1-p)$.
  - $\cdot$ denotes element-wise multiplication.
- $\nabla_f = \nabla_g \cdot \epsilon$
  - Even though ϵ is random, you must use the same ϵ in the backward pass that you generated for the forward pass.
  - Don't backpropagate to ϵ because it is not a function of the input.
- Like RELU, but kills gradients based on an external random source---whether you dropped that activation or not in the forward pass. If you didn't, remember to multiply by the $1/(1-p)$.

REGULARIZATION

Regularization: Early Stopping

- Keep track of dev set error. Stop optimization when it starts going up.
- This is a legitimate regularization technique !
- Essentially, you are restricting your hypothesis space to functions that are reachable within N iterations of a random initialization.

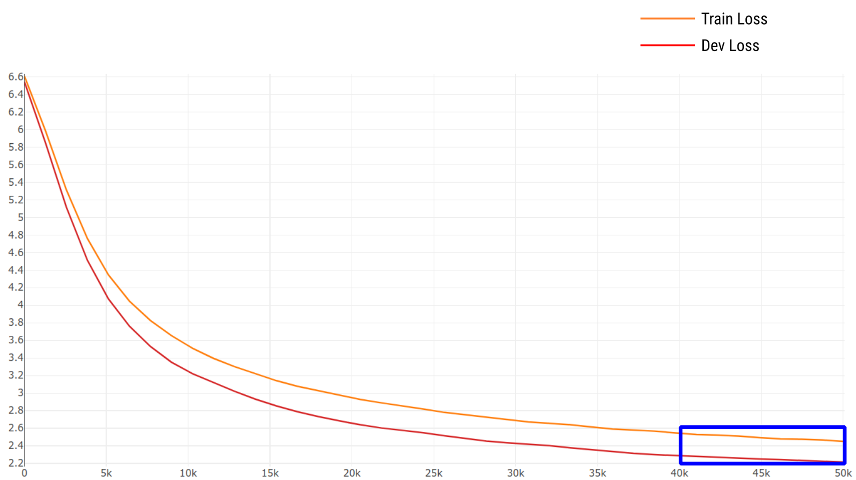
TRAINING IN PRACTICE



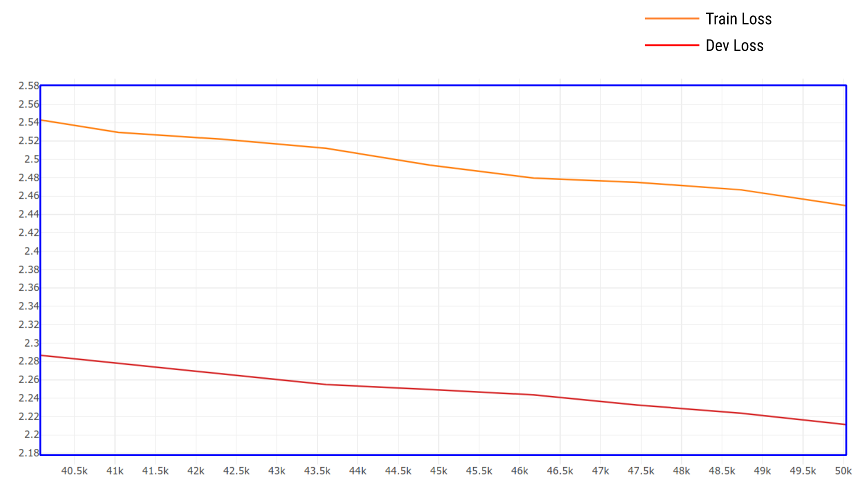
```
2018-09-30 13:32:10 [000000000] Val_Loss = 7.004e+00, Val_Acc = 0.0004
2018-09-30 13:34:27 [000000010] lr=1.00e-01, Train_Loss = 7.046e+00, Train_Acc = 0.0008
2018-09-30 13:35:47 [000000020] lr=1.00e-01, Train_Loss = 7.162e+00, Train_Acc = 0.0012
2018-09-30 13:37:07 [000000030] lr=1.00e-01, Train_Loss = 7.147e+00, Train_Acc = 0.0016
2018-09-30 13:38:26 [000000040] lr=1.00e-01, Train_Loss = 7.147e+00, Train_Acc = 0.0020
2018-09-30 13:39:46 [000000050] lr=1.00e-01, Train_Loss = 7.057e+00, Train_Acc = 0.0020
2018-09-30 13:41:06 [000000060] lr=1.00e-01, Train_Loss = 7.008e+00, Train_Acc = 0.0012
2018-09-30 13:42:25 [000000070] lr=1.00e-01, Train_Loss = 6.978e+00, Train_Acc = 0.0016
2018-09-30 13:43:45 [000000080] lr=1.00e-01, Train_Loss = 6.931e+00, Train_Acc = 0.0020
2018-09-30 13:45:05 [000000090] lr=1.00e-01, Train_Loss = 6.927e+00, Train_Acc = 0.0023
2018-09-30 13:46:25 [000000100] lr=1.00e-01, Train_Loss = 6.907e+00, Train_Acc = 0.0020
2018-09-30 13:47:45 [000000110] lr=1.00e-01, Train_Loss = 6.897e+00, Train_Acc = 0.0012
2018-09-30 13:49:05 [000000120] lr=1.00e-01, Train_Loss = 6.871e+00, Train_Acc = 0.0035
2018-09-30 13:50:24 [000000130] lr=1.00e-01, Train_Loss = 6.857e+00, Train_Acc = 0.0031
2018-09-30 13:51:44 [000000140] lr=1.00e-01, Train_Loss = 6.852e+00, Train_Acc = 0.0016
2018-09-30 13:53:04 [000000150] lr=1.00e-01, Train_Loss = 6.831e+00, Train_Acc = 0.0043
2018-09-30 13:54:24 [000000160] lr=1.00e-01, Train_Loss = 6.820e+00, Train_Acc = 0.0039
2018-09-30 13:55:43 [000000170] lr=1.00e-01, Train_Loss = 6.793e+00, Train_Acc = 0.0027
2018-09-30 13:57:03 [000000180] lr=1.00e-01, Train_Loss = 6.800e+00, Train_Acc = 0.0023
2018-09-30 13:58:23 [000000190] lr=1.00e-01, Train_Loss = 6.774e+00, Train_Acc = 0.0039
2018-09-30 13:59:42 [000000200] lr=1.00e-01, Train_Loss = 6.766e+00, Train_Acc = 0.0047
2018-09-30 14:01:02 [000000210] lr=1.00e-01, Train_Loss = 6.773e+00, Train_Acc = 0.0031
2018-09-30 14:02:22 [000000220] lr=1.00e-01, Train_Loss = 6.712e+00, Train_Acc = 0.0039
2018-09-30 14:03:42 [000000230] lr=1.00e-01, Train_Loss = 6.712e+00, Train_Acc = 0.0043
2018-09-30 14:05:01 [000000240] lr=1.00e-01, Train_Loss = 6.747e+00, Train_Acc = 0.0043
2018-09-30 14:06:21 [000000250] lr=1.00e-01, Train_Loss = 6.715e+00, Train_Acc = 0.0043
2018-09-30 14:07:41 [000000260] lr=1.00e-01, Train_Loss = 6.742e+00, Train_Acc = 0.0031
2018-09-30 14:09:00 [000000270] lr=1.00e-01, Train_Loss = 6.697e+00, Train_Acc = 0.0047
2018-09-30 14:10:20 [000000280] lr=1.00e-01, Train_Loss = 6.696e+00, Train_Acc = 0.0059
2018-09-30 14:11:40 [000000290] lr=1.00e-01, Train_Loss = 6.673e+00, Train_Acc = 0.0055
2018-09-30 14:12:59 [000000300] lr=1.00e-01, Train_Loss = 6.668e+00, Train_Acc = 0.0059
2018-09-30 14:14:19 [000000310] lr=1.00e-01, Train_Loss = 6.659e+00, Train_Acc = 0.0027
2018-09-30 14:15:39 [000000320] lr=1.00e-01, Train_Loss = 6.657e+00, Train_Acc = 0.0059
2018-09-30 14:16:59 [000000330] lr=1.00e-01, Train_Loss = 6.655e+00, Train_Acc = 0.0078
2018-09-30 14:18:18 [000000340] lr=1.00e-01, Train_Loss = 6.630e+00, Train_Acc = 0.0070
2018-09-30 14:19:38 [000000350] lr=1.00e-01, Train_Loss = 6.637e+00, Train_Acc = 0.0063
2018-09-30 14:20:58 [000000360] lr=1.00e-01, Train_Loss = 6.607e+00, Train_Acc = 0.0055
2018-09-30 14:22:18 [000000370] lr=1.00e-01, Train_Loss = 6.628e+00, Train_Acc = 0.0078
2018-09-30 14:23:37 [000000380] lr=1.00e-01, Train_Loss = 6.614e+00, Train_Acc = 0.0051
2018-09-30 14:24:57 [000000390] lr=1.00e-01, Train_Loss = 6.606e+00, Train_Acc = 0.0066
2018-09-30 14:26:16 [000000400] lr=1.00e-01, Train_Loss = 6.579e+00, Train_Acc = 0.0090
2018-09-30 14:27:36 [000000410] lr=1.00e-01, Train_Loss = 6.585e+00, Train_Acc = 0.0066
2018-09-30 14:28:56 [000000420] lr=1.00e-01, Train_Loss = 6.587e+00, Train_Acc = 0.0055
2018-09-30 14:30:15 [000000430] lr=1.00e-01, Train_Loss = 6.575e+00, Train_Acc = 0.0051
2018-09-30 14:31:35 [000000440] lr=1.00e-01, Train_Loss = 6.566e+00, Train_Acc = 0.0082
2018-09-30 14:32:55 [000000450] lr=1.00e-01, Train_Loss = 6.534e+00, Train_Acc = 0.0074
```

Don't look at this !

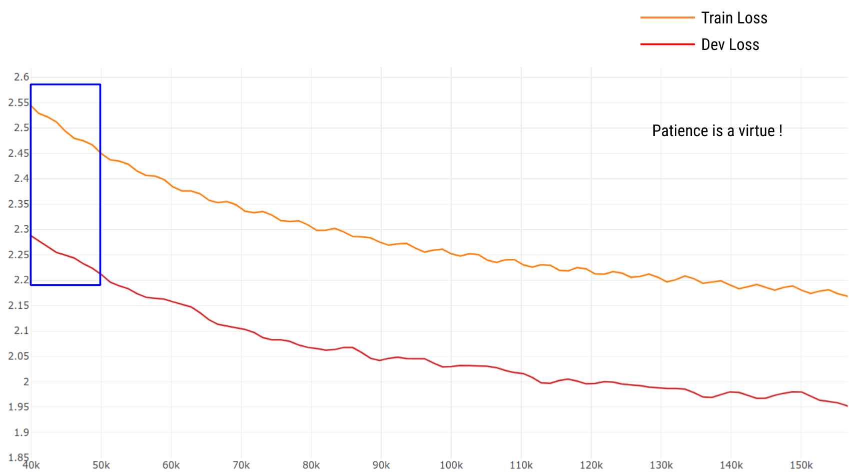
TRAINING IN PRACTICE



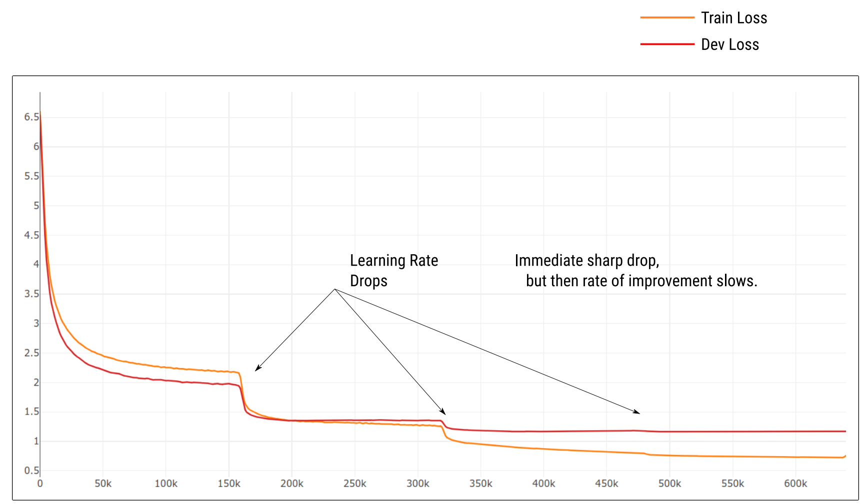
TRAINING IN PRACTICE



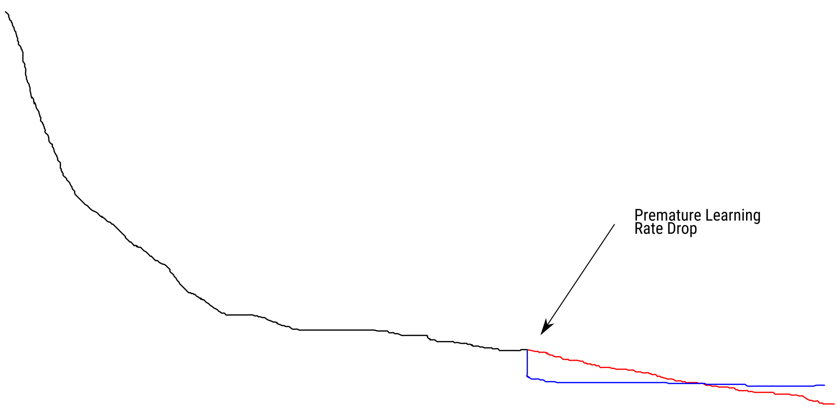
TRAINING IN PRACTICE



TRAINING IN PRACTICE



TRAINING IN PRACTICE



DIFFERENT OPTIMIZATION METHODS

- Standard SGD

$$w_i \leftarrow w_i - \lambda \nabla_{w_i}$$

- Momentum

$$g_i \leftarrow \nabla_{w_i} + \gamma g_i$$
$$w_i \leftarrow w_i - \lambda g_i$$

- But we are still applying the same learning rate for all parameters / weights.

DIFFERENT OPTIMIZATION METHODS

Adaptive Learning Rate Methods

Key idea: Set the learning rate for each parameter based on the magnitude of its gradients.

- Adagrad

$$g_i^2 \leftarrow g_i^2 + (\nabla_{w_i})^2$$
$$w_i \leftarrow w_i - \lambda \frac{\nabla_{w_i}}{\sqrt{g_i^2 + \epsilon}}$$

Global learning rate divided by sum of magnitudes of past gradients.

Problem: Will always keep dropping the effective learning rate.

- RMSProp

$$g_i^2 \leftarrow \gamma g_i^2 + (1-\gamma)(\nabla_{w_i})^2$$
$$w_i \leftarrow w_i - \lambda \frac{\nabla_{w_i}}{\sqrt{g_i^2 + \epsilon}}$$

DIFFERENT OPTIMIZATION METHODS

Adaptive Learning Rate Methods

- Adam: RMSProp + Momentum

$$m_i \leftarrow \beta_1 m_i + (1-\beta_1)\nabla_{w_i}$$
$$v_i \leftarrow \beta_2 v_i + (1-\beta_2)(\nabla_{w_i})^2$$
$$w_i \leftarrow w_i - \frac{\lambda}{\sqrt{v_i} + \epsilon} m_i$$

- How do you initialize m_i and v_i ? Typically as 0 and 1.
- This won't matter once the values of m_i, v_i stabilize. But in initial iterations, they will be biased towards their initial values.

DIFFERENT OPTIMIZATION METHODS

Adaptive Learning Rate Methods

- Adam: RMSProp + Momentum + Bias Correction

$$m_i \leftarrow \beta_1 m_i + (1-\beta_1)\nabla_{w_i}$$
$$v_i \leftarrow \beta_2 v_i + (1-\beta_2)(\nabla_{w_i})^2$$
$$\hat{m}_i = \frac{m_i}{1-\beta_1^t}$$
$$\hat{v}_i = \frac{v_i}{1-\beta_2^t}$$
$$w_i \leftarrow w_i - \frac{\lambda}{\sqrt{\hat{v}_i} + \epsilon} \hat{m}_i$$

Here, t is the iteration number.

As $t \rightarrow \infty$, $1 - \beta^t = 1$.

DISTRIBUTED TRAINING

- Neural Network Training is Slow.
- But many operations are parallelizable. In particular, operations for different batches are independent.
- That's why GPUs are great for deep learning! But even so, you will begin to saturate the computation (or worse, memory) on a GPU.
- Solution: Break up computation across multiple GPUs.
- Two possibilities:
  - Model Parallelism
  - Data Parallelism

DISTRIBUTED TRAINING

Model Parallelism

- Less popular, doesn't help for many networks.
- Essentially, if you have two independent "paths" in your network, you can place them on different devices. And sync, when they join.

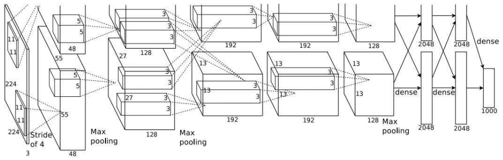


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Was used in the Sutskever et al., 2012 ImageNet paper.

DISTRIBUTED TRAINING

Data Parallelism

- Begin with all devices having the same model weights.
- One each device, load a separate batch of data.
- Do forward-backward to compute weight gradients on each GPU with its own batch.
- Have a single device (one of the GPUs, or a CPU) recover gradients from all devices.
- Average these gradients and apply the update to the weights.
- Distribute new weights to all devices.
- Works well in practice, especially for multiple GPUs in the same machine.
- Communication overhead of transferring gradients and weights back and forth. Can be large if distributing across multiple machines.
- Approximate Distributed Training
    - Let each worker keep updating its own weights independently for multiple iterations. Then, transmit back weights to single device, average weights, and sync to all devices.
    - Other options, quantize gradients when sending back and forth (while making sure all workers have the same models).