11. Neural Network Regularization

CS 519 Deep Learning, Winter 2016

Fuxin Li

With materials from Andrej Karpathy, Zsolt Kira

Preventing overfitting

- Approach 1: Get more data!
  - Always best if possible!
  - If no natural ones, use data augmentation
- Approach 2: Use a model that has the right capacity:
  - enough to fit the true regularities.
  - not enough to also fit spurious regularities (if they are weaker).
  - Parameter tuning
- Approach 3: Average many different models.
  - Models with different forms.
  - Train on different subsets
- Approach 4: Use specific regularizing structures

Regularization: Preventing Overfitting

- To **prevent overfitting**, a large number of different methods have been developed.
  - Data Augmentation (talked about)
  - Weight-sharing structures (talked about, e.g. CNN, RNN)
  - Weight-decay (talked about)
  - Early stopping (talked about)
  - **Model averaging**
  - **Dropout**
  - **Batch normalization**
  - **Weight regularization structures**
  - Bayesian fitting of neural nets
  - Generative pre-training (will talk later)
  - Sparsity in hidden units (will talk later)

Making models differ by changing their training data

- Bagging: Train different models on different subsets of the data.
  - Sample data with replacement
    a,b,c,d,e → a c c d b
  - Random forests use lots of different decision trees trained using bagging. They work well.
- We could use bagging with neural nets.

- Boosting: Train a sequence of low capacity models. Weight the training cases differently for each model in the sequence.
  - Boosting up-weights cases that previous models got wrong.
  - An early use of boosting was with neural nets for MNIST.
  - It focused the computational resources on modeling the tricky cases.

Bagging in Deep Neural Networks

- Deep networks are inherent local optimization algorithms
- Different starting points give very different result networks!
- Directly averaging networks with different initializations
  - No bootstrapping!

Some model averaging results

Error %	Val Top-1	Val Top-5	Test Top-5
Krizhevsky *et al.* [67], 1 convnet	40.7	18.2	––
Krizhevsky *et al.* [67], 5 convnets	38.1	16.4	16.4
Krizhevsky *et al.* [67], 1 convnets*	39.0	16.6	––
Krizhevsky *et al.* [67], 7 convnets*	36.7	15.4	15.3
Our replication of Krizhevsky *et al.* [67], 1 convnet	40.5	18.1	––
1 convnet as per Figure 7.2	38.3	16.4	16.5
5 convnets as per Figure 7.2	36.6	15.3	15.3

Table 7.2: ImageNet 2012 classification error rates. The $*$ indicates models that were trained on both ImageNet 2011 and 2012 training sets with an additional convolution layer.

Multiple examples from one test data: Test time Cropping

- e.g. Resize the image into different sizes/aspect ratios, crop squares at different places of the image
  - Similar to object proposals, but squared
  - Reduce the error significantly with 144/150 crops (proposals)

Effect of Test-time Cropping/model Averaging

VGG single model:

ConvNet config. (Table 1)	Evaluation method		top-1 val. error (%)	top-5 val. error (%)
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

VGG multiple models:

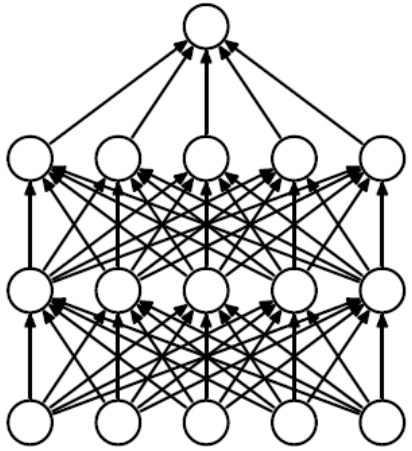
E	dense	24.8	7.5
	multi-crop	24.6	7.4
	multi-crop & dense	**24.4**	**7.1**

Inception (GoogLeNet):

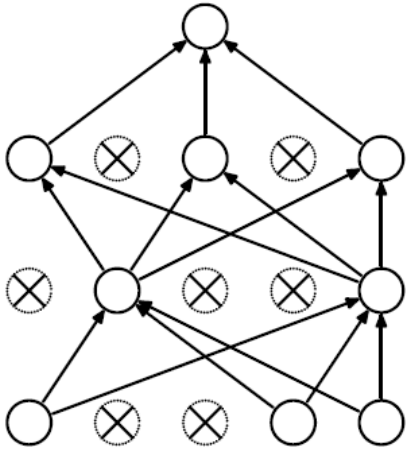
Number of models	**Number of Crops**	**Cost**	**Top-5 error**	**compared to base**
1	1	1	10.07%	base
1	10	10	9.15%	-0.92%
1	144	144	7.89%	-2.18%
7	1	7	8.09%	-1.98%
7	10	70	7.62%	-2.45%
7	144	1008	6.67%	-3.45%

Dropout: An efficient way to average many large neural nets

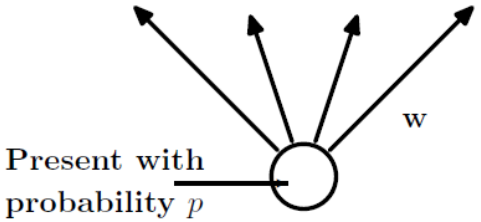
- Consider a neural net with one hidden layer.
- Each time we present a training example, we randomly omit each hidden unit with probability 0.5.
- So we are randomly sampling from 2^H different architectures.
  - All architectures share weights.



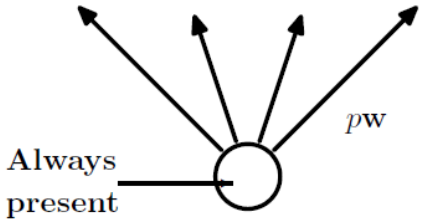
(a) Standard Neural Net



(b) After applying dropout.



(a) At training time



(b) At test time

Dropout as preventing co-adaptation

- If a hidden unit knows which other hidden units are present, it can co-adapt to them on the training data.
  - But complex co-adaptations are likely to go wrong on new test data.
  - Big, complex conspiracies are not robust.

- Dropout as orthogonalization

Dropout as a form of model averaging

- We sample from 2^H models. So only a few of the models ever get trained, and they only get one training example.

- The sharing of the weights means that every model is very strongly regularized.
  - It's a much better regularizer than L2 or L1 penalties that pull the weights towards zero.

But what do we do at test time?

- We could sample many different architectures and take the geometric mean of their output distributions.
- It better to use all of the hidden units, but to halve their outgoing weights.
  - This exactly computes the geometric mean of the predictions of all 2^H models.

What if we have more hidden layers?

- Use dropout of 0.5 in every layer.
- At test time, use the “mean net” that has all the outgoing weights halved.
  - This is not exactly the same as averaging all the separate dropped out models, but it’s a pretty good approximation, and its fast.
- Alternatively, run the stochastic model several times on the same input.
  - This gives us an idea of the uncertainty in the answer.

What about the input layer?

- It may help to use dropout there too, but with a higher probability of keeping an input unit.
  - Averaging out the noise in the input if it's noisy (don't use it if it's not noisy)
  - This trick is already used by the "denoising autoencoders" developed by Pascal Vincent, Hugo Larochelle and Yoshua Bengio.

Some dropout tips

- Dropout lowers your **capacity**
  - Increase network size by n/p where n is # hidden units in original, p is probability of dropout
- Dropout adds **noise** to gradients
  - Increase learning rate by 10-100
  - Or increase momentum (e.g. from 0.9 to 0.99)
  - These can cause large weight growths, use weight regularization

How well does dropout work?

- The record breaking object recognition net developed by Alex Krizhevsky (see lecture 5) uses dropout and it helps a lot.
- VGG network also uses dropout heavily (to the note of 90% dropout)
- The ResNet (state-of-the-art in 2015) doesn't use dropout
- If your deep neural net is significantly overfitting, dropout will usually reduce the number of errors by a lot.
  - Any net that uses "early stopping" can do better by using dropout (at the cost of taking quite a lot longer to train).
- If your deep neural net is not overfitting you should be using a bigger one!

Batch normalization (Ioffe and Szegedy 2015)

- Idea: Deep layers can have increased bias
- Suppose: $y = xw_1w_2w_3 \dots w_l$
- Update: $\boldsymbol{w} = \boldsymbol{w} - \epsilon \boldsymbol{g}$
- $y = x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2)(w_3 - \epsilon g_3) \dots (w_l - \epsilon g_l)$
- AdaGrad etc. sets $\epsilon = \epsilon / ||\boldsymbol{g}||$
- When deep, many terms with various levels of epsilon values!
  - E.g. a term $\epsilon^2 g_1 g_2 \prod_{i=3}^{l} w_i$ and a term $\epsilon g_5 \prod_{i=1, i\neq 5}^{l} w_i$
  - What if, e.g. $\prod_{i=3}^{l} w_i$ is very big?
  - Especially, during first few iterations?

Whitening

- It makes sense to normalize the output of each layer
  - 0 Mean, 1 standard deviation
  - Empirically observed as improving convergence
- Latter layers can be considered “using previous layer’s output to perform machine learning”
- How to do this during stochastic mini-batch optimization?
  - “Approximate mean and standard deviation” using mini-batch

Imperfect approximation

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

- Use mini-batch to approximate E and Var
- What if it's wrong?
  - Make sure the network can "correct" this change

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}.$$

  - 2 learnable parameters for each x

Batch Normalization Layer

- This is done for each hidden dimension separately

- How many parameters?
- Gradient w.r.t. parameters?
- Gradient w.r.t. input?

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

All the gradients

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

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

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

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

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

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

Other stuff

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}.$$

- If $\gamma = 0$, equiv. to dropout
- No additional bias term needed in the conventional network (BN provides the bias term)

Result

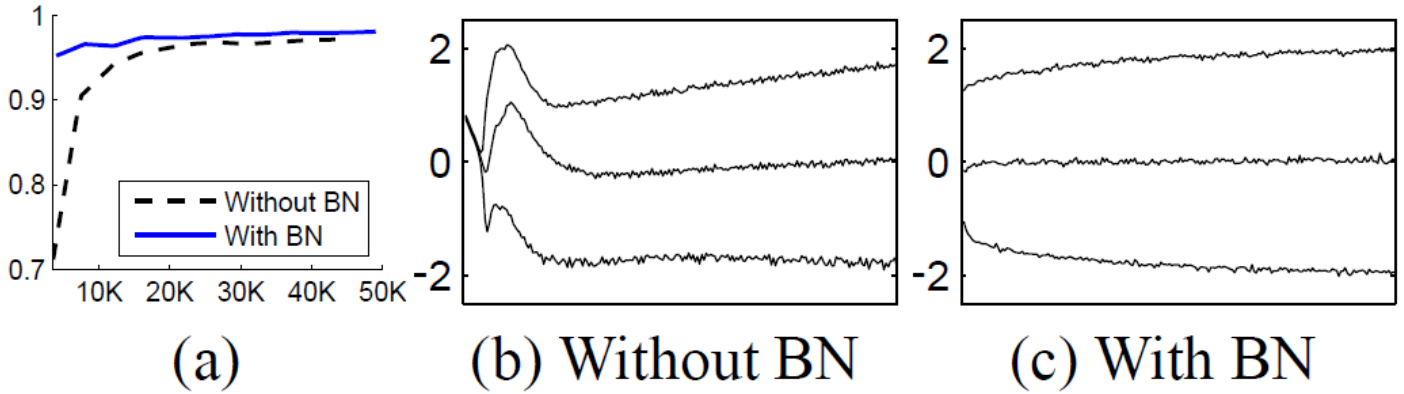


Figure 1: (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of training, shown as* $\{15, 50, 85\}$*th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.*

Result (faster learning rate!)

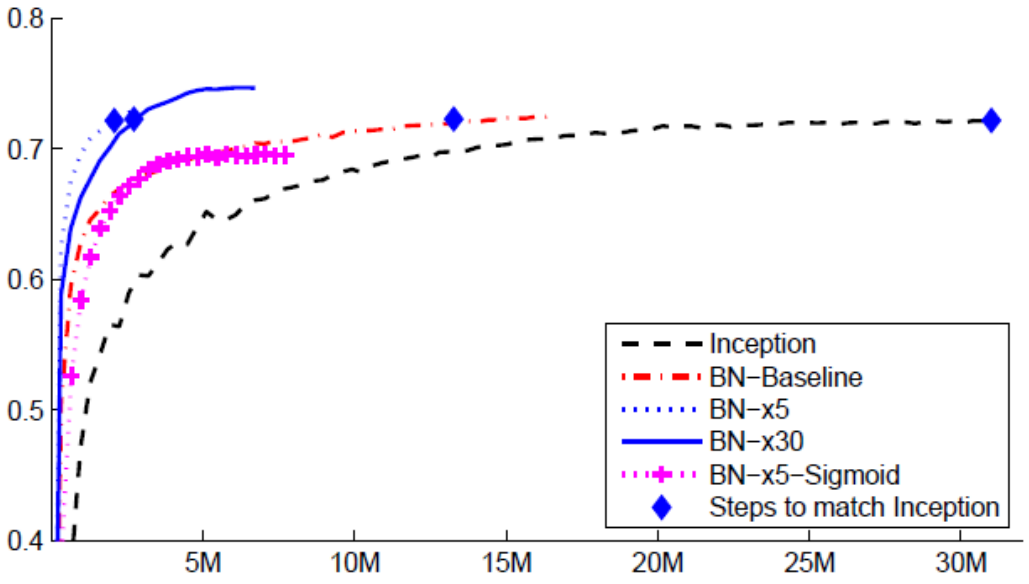


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

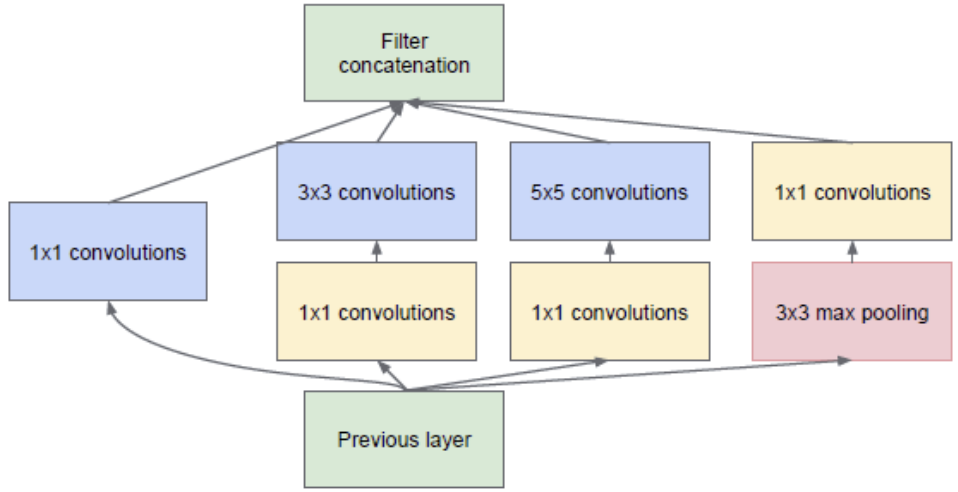
Result on ImageNet

Model	Resolution	Crops	Models	Top-1 error	Top-5 error
GoogLeNet ensemble	224	144	7	-	6.67%
Deep Image low-res	256	-	1	-	7.96%
Deep Image high-res	512	-	1	24.88	7.42%
Deep Image ensemble	variable	-	-	-	5.98%
BN-Inception single crop	224	1	1	25.2%	7.82%
BN-Inception multicrop	224	144	1	21.99%	5.82%
BN-Inception ensemble	224	144	6	20.1%	**4.9%***

The Inception Network

- Basic idea:
  - Wider networks have more representation power
    - But they are too slow
  - Dimensionality reduction to allow for wider network
    - Just drop some filters

Reduce number
of filters
+
Add nonlinearity with
another RELU



(b) Inception module with dimension reductions

GoogLeNet

- Observation:
  - Inception is mainly high-level
  - Low-level network is not as descriptive as VGG
  - On high-level, possibly wider network is useful

type	patch size/ stride	output size
convolution	7×7/2	112×112×64
max pool	3×3/2	56×56×64
convolution	3×3/1	56×56×192
max pool	3×3/2	28×28×192
inception (3a)		28×28×256
inception (3b)		28×28×480
max pool	3×3/2	14×14×480
inception (4a)		14×14×512
inception (4b)		14×14×512
inception (4c)		14×14×512
inception (4d)		14×14×528
inception (4e)		14×14×832
max pool	3×3/2	7×7×832
inception (5a)		7×7×832
inception (5b)		7×7×1024
avg pool	7×7/1	1×1×1024
dropout (40%)		1×1×1024
linear		1×1×1000
softmax		1×1×1000

The Crazy Pic

- The whole network looks like this:
- Mainly, beyond inception, the big deal is the additional yellow nodes
  - These are output nodes for prediction at middle layers
  - Improves gradient conditioning
    - Ameliorate vanishing gradient
    - Similar methods are used to train VGG