Deep Learning

Training Strategies

Next Two Lectures…

- Why study (deep) neural networks at all?
- How do we train deep neural networks?

Universality

- Why study neural networks in general?
  - Neural network can approximate any continuous function, even with a single hidden layer!
  - http://neuralnetworksanddeeplearning.com/chap4.html

Why Study Deep Networks?

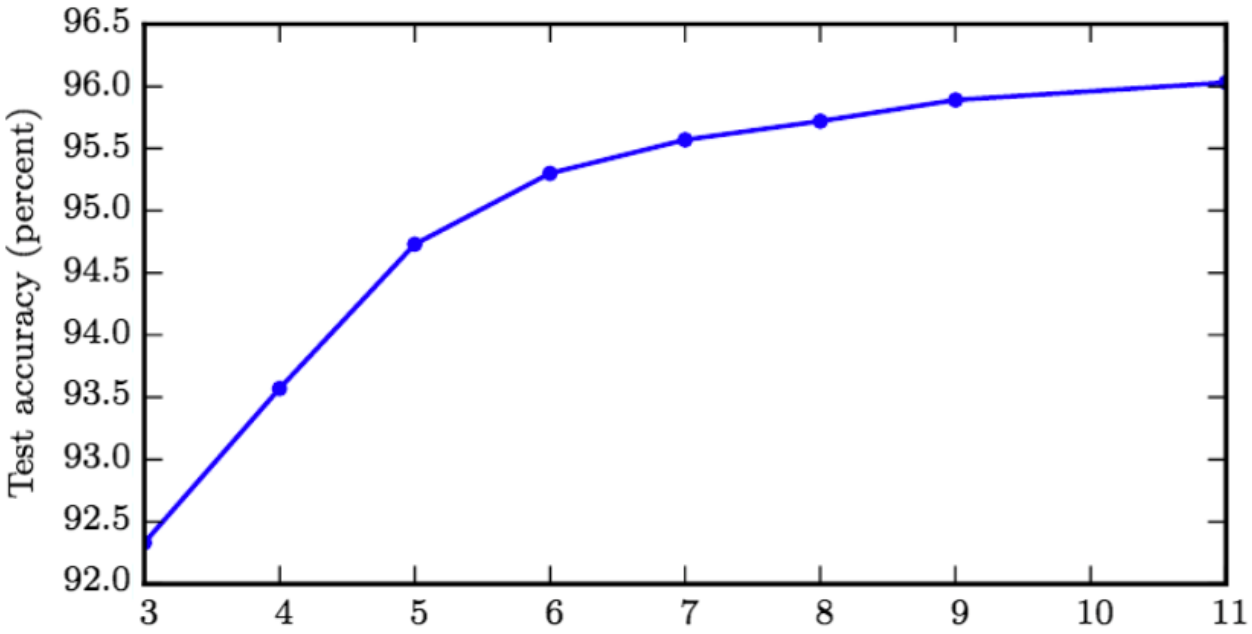
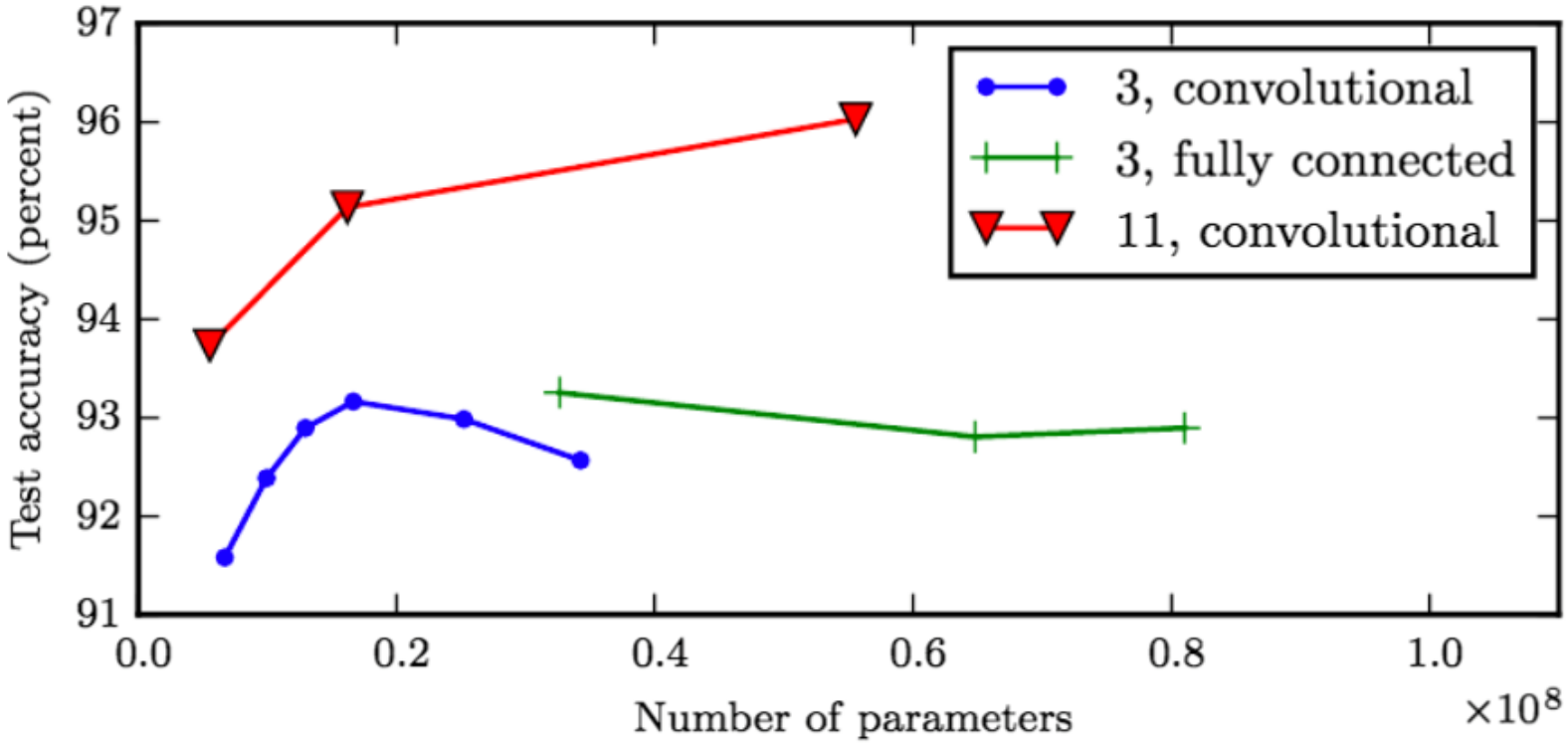
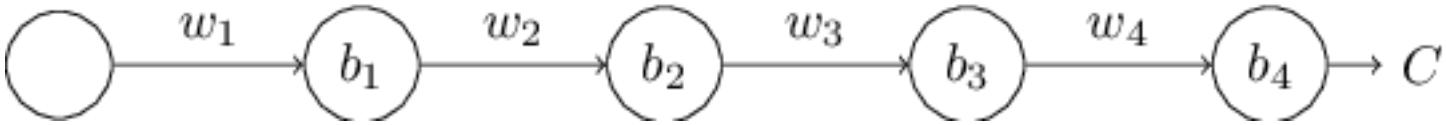


Figure 6.6: Empirical results showing that deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses. Data from Goodfellow *et al.* (2014d). The test set accuracy consistently increases with increasing depth. See figure 6.7 for a control experiment demonstrating that other increases to the model size do not yield the same effect.


Test accuracy (percent)
97
96
95
94
93
92
91
0.0
0.2
0.4
0.6
0.8
1.0
Number of parameters
×10^8
3, convolutional
3, fully connected
11, convolutional

But… Watch Out for Vanishing Gradients

- Consider a simple network, and perform backpropagation



  - For simplicity, just a single neuron
  - Sigmoid at every layer, $z_j = \sigma(w_j a_{j-1} + b_j), a_j = \sigma(z_j)$
  - Cost function C

- Gradient $\partial C / \partial b_1$ is a product of terms:

$$\partial C/\partial b_1 = \sigma'(z_1) w_2 \sigma'(z_2) w_3 \sigma'(z_3) w_4 \sigma'(z_4) (\partial C/\partial a_4)$$

Vanishing Gradients

- Gradient of sigmoid is in (0,1/4)
- Weights are also typically initialized in (0,1)
- Products of small numbers → small gradients
- Backprop does not change weights in earlier layers very much!
  - This is an issue with backprop, not with the model itself

Rectified Linear Units

- Alternative non-linearity:

$$g(x) = \max(0, x)$$

- Gradient of this function?
  - Note: need subgradient descent here.
- https://cs224d.stanford.edu/notebooks/vanishing_grad_example.html
- Increasing the number of layers can result in requiring exponentially fewer hidden units per layer (see "Understanding Deep Neural Networks with Rectified Linear Units")
- Biological considerations
  - On some inputs, biological neurons have no activation
  - On some inputs, neurons have activation proportional to input

Other Activation Functions

- Leaky ReLU: $g(x) = \max(0, x) + \alpha \min(0, x) \quad (\alpha \approx .01)$
- Tanh: $g(x) = 2\sigma(2x) - 1$
- Radial Basis Functions: $g(x) = \exp(-(w - x)^2/\sigma^2)$
- Softplus: $g(x) = \log(1 + e^x)$
- Hard Tanh: $g(x) = \max(-1, \min(1, x))$
- Maxout: $g(x) = \max_{j \in \mathbb{G}} x_j$
- ....

Architecture Design and Training Issues

- How many layers? How many hidden units per layer? How to connect layers together? How to optimize?
  - Cost functions
  - L2/L1 regularization
  - Data Set Augmentation
  - Early Stopping
  - Dropout
  - Minibatch Training
  - Momentum
  - Initialization
  - Batch Normalization

Cost Functions

- For regression problems, quadratic error is typical
- For classification, quadratic loss is not as effective
  - Instead one typically uses softmax outputs with cross-entropy error function
  - Discussed earlier, won't review in depth

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$
$$= -\frac{1}{m} [\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

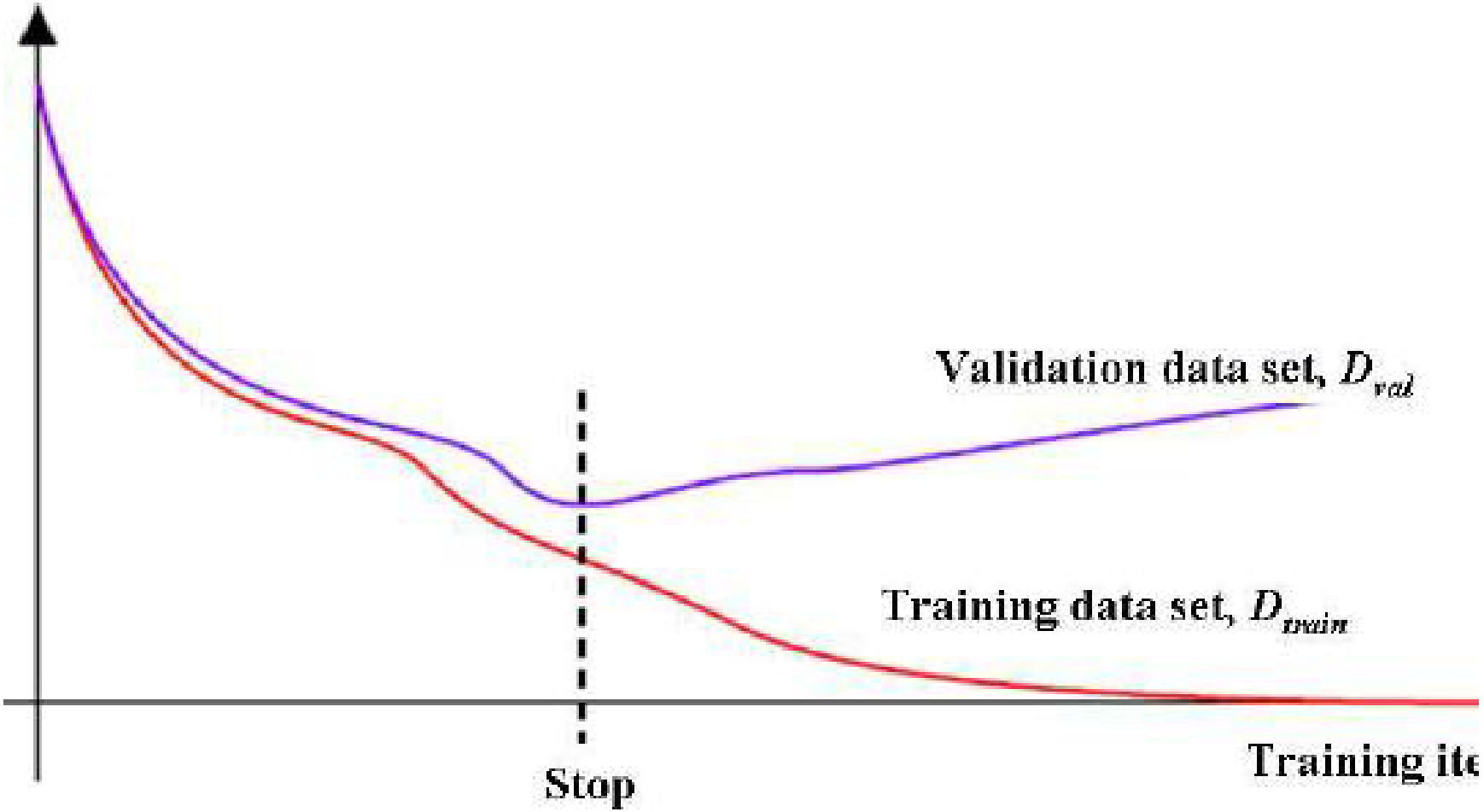
Regularization

- In machine learning, we care about generalization performance, not just training error
- With many parameters, models are prone to overfitting
- How to regularize?
  - Restrictions on parameter values or function classes
  - Adding terms to the objective function
  - Examples of the latter: L2 or L1 regularization

Data Set Augmentation

- The more data, the better for generalization (usually)
- Sometimes we can augment our existing data set
  - Example: for image classification, mirror-image all images to double the size of the training set
  - Injecting noise to training data is also a form of data augmentation

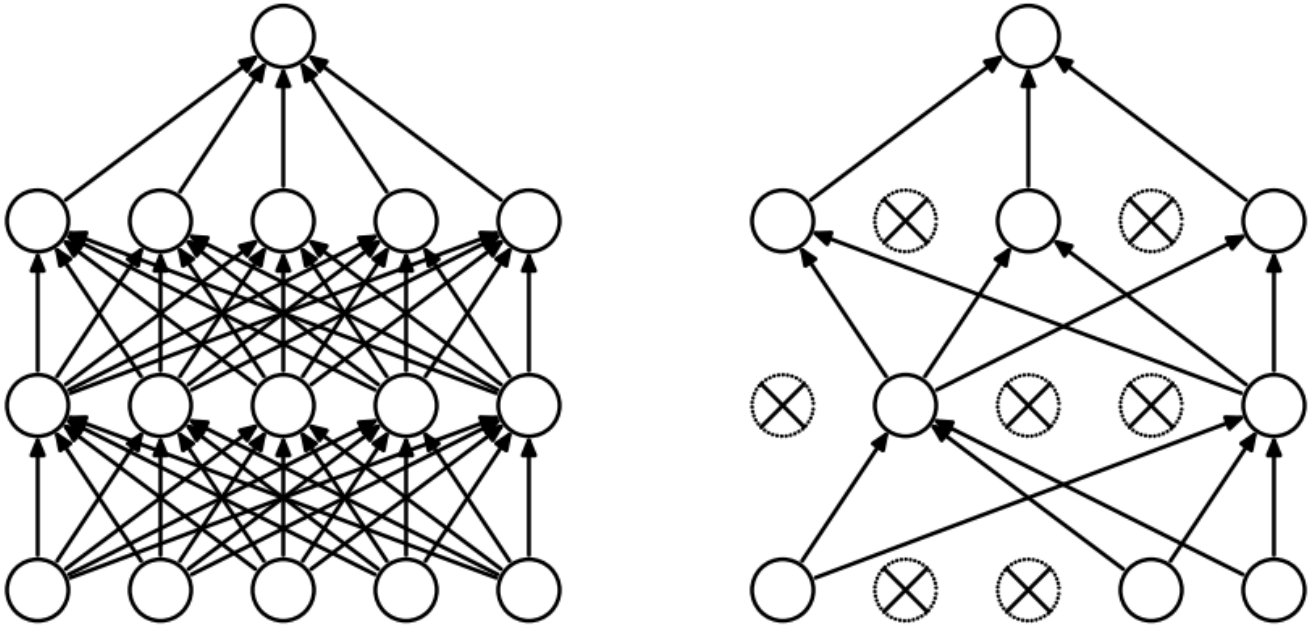
Early Stopping



Algorithm 7.1 The early stopping meta-algorithm for determining the best amount of time to train. This meta-algorithm is a general strategy that works well with a variety of training algorithms and ways of quantifying error on the validation set.

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*

Dropout



(a) Standard Neural Net

(b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Dropout

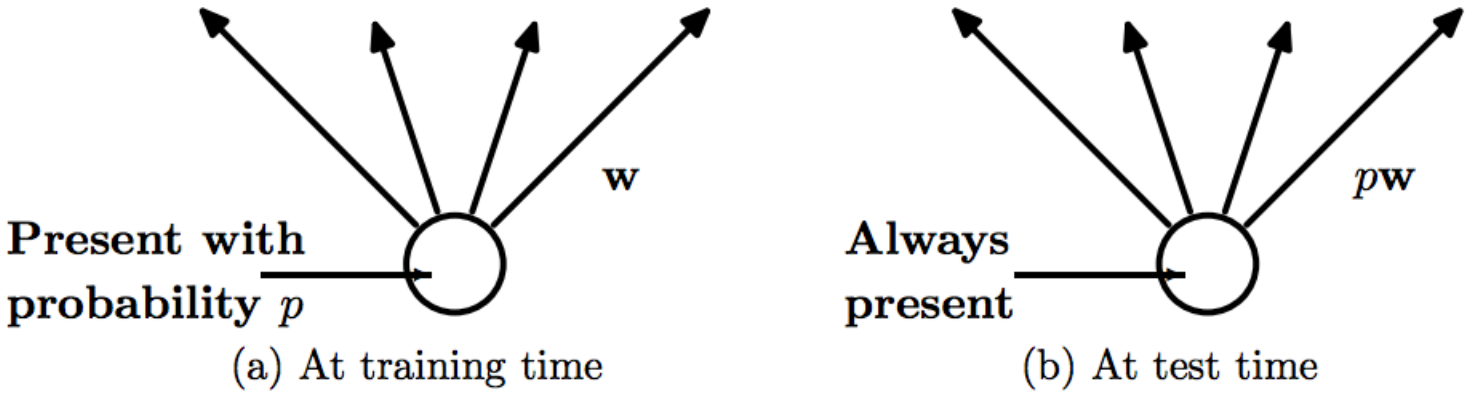


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Dropout

Method	Unit Type	Architecture	Error %
Standard Neural Net (Simard et al., 2003)	Logistic	2 layers, 800 units	1.60
SVM Gaussian kernel	NA	NA	1.40
Dropout NN	Logistic	3 layers, 1024 units	1.35
Dropout NN	ReLU	3 layers, 1024 units	1.25
Dropout NN + max-norm constraint	ReLU	3 layers, 1024 units	1.06
Dropout NN + max-norm constraint	ReLU	3 layers, 2048 units	1.04
Dropout NN + max-norm constraint	ReLU	2 layers, 4096 units	1.01
Dropout NN + max-norm constraint	ReLU	2 layers, 8192 units	0.95
Dropout NN + max-norm constraint (Goodfellow et al., 2013)	Maxout	2 layers, (5 × 240) units	0.94
DBN + finetuning (Hinton and Salakhutdinov, 2006)	Logistic	500-500-2000	1.18
DBM + finetuning (Salakhutdinov and Hinton, 2009)	Logistic	500-500-2000	0.96
DBN + dropout finetuning	Logistic	500-500-2000	0.92
DBM + dropout finetuning	Logistic	500-500-2000	**0.79**

Table 2: Comparison of different models on MNIST.

The MNIST data set consists of 28 × 28 pixel handwritten digit images. The task is to classify the images into 10 digit classes. Table 2 compares the performance of dropout with other techniques. The best performing neural networks for the permutation invariant

Dropout

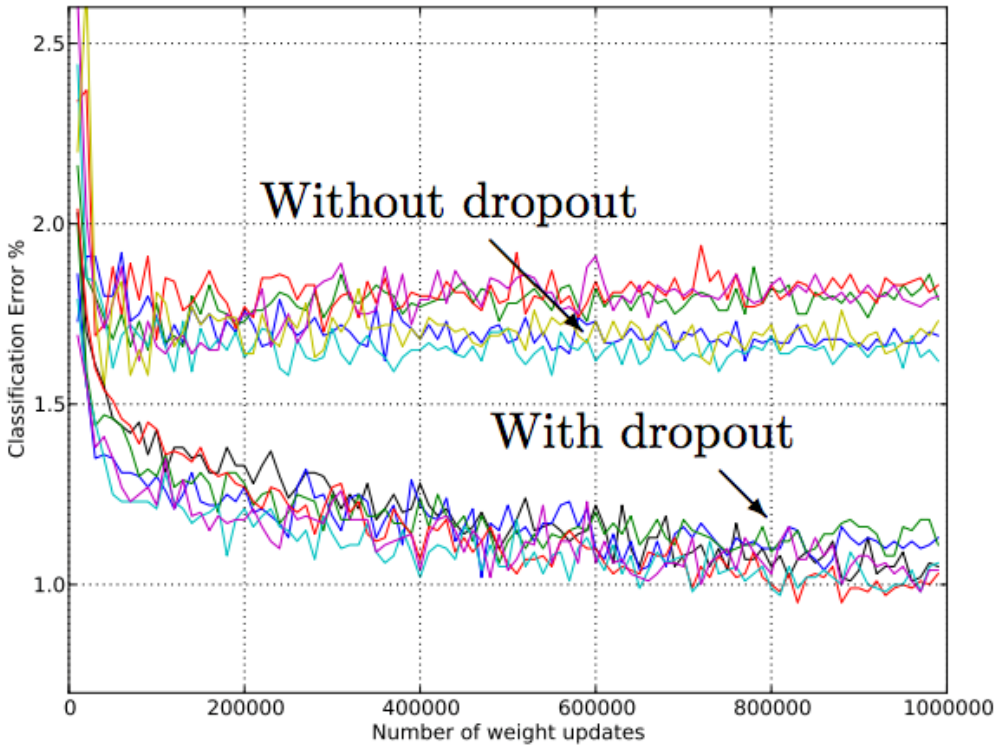


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Minibatch Training

- Gradient descent uses all training points, fully online (stochastic) methods update using a single point
- Many deep learning methods fall in between
- Example: computing the mean of a set of samples
  - Standard error based on a sample of n points is $\sigma / \sqrt{n}$
  - Consider using 100 versus 10,000 samples
  - Latter requires 100x more computation but reduces error by factor of 10

Minibatch Training

- Larger batches provide a more accurate estimate of the gradient, but with less than linear returns
- Multicore architectures are usually underutilized by extremely small batches, motivating a minimum batch size
- If all examples in the batch are processed in parallel, amount of memory scales with batch size
- When using GPUs it is common for power of 2 batch sizes to offer better runtime; typical sizes range from 32 to 256, with 16 being common for large models
- Small batches can offer a regularizing effect due to the noise added during the learning process
- Second-order methods typically require larger batch sizes
- Minibatches should be selected randomly!

Momentum

- Accumulates an exponentially decaying moving average of past gradients and continues to move in their direction

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Momentum

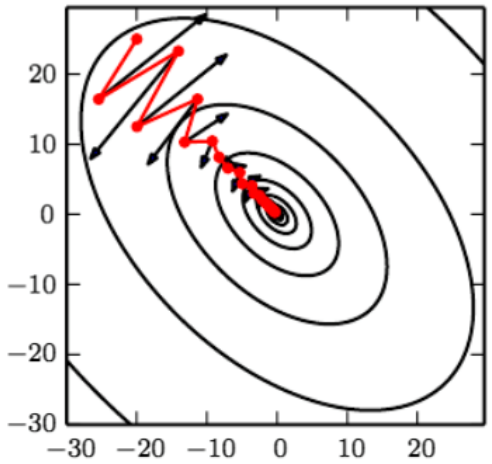


Figure 8.5: Momentum aims primarily to solve two problems: poor conditioning of the Hessian matrix and variance in the stochastic gradient. Here, we illustrate how momentum overcomes the first of these two problems. The contour lines depict a quadratic loss function with a poorly conditioned Hessian matrix. The red path cutting across the contours indicates the path followed by the momentum learning rule as it minimizes this function. At each step along the way, we draw an arrow indicating the step that gradient descent would take at that point. We can see that a poorly conditioned quadratic objective looks like a long, narrow valley or canyon with steep sides. Momentum correctly traverses the canyon lengthwise, while gradient steps waste time moving back and forth across the narrow axis of the canyon. Compare also figure 4.6, which shows the behavior of gradient descent without momentum.

Initialization

- Important: need to “break symmetry”
  - Choose weights randomly
  - (Biases typically chosen heuristically)
- Combined with early stopping, can think of initialization as as prior on the weights
- Usually use uniform or Gaussian weights with a zero-mean
- Examples with m inputs and n outputs:

$$W_{ij} \sim U\left(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\right) \qquad W_{ij} \sim U\left(-\frac{6}{\sqrt{m+n}}, \frac{6}{\sqrt{m+n}}\right)$$

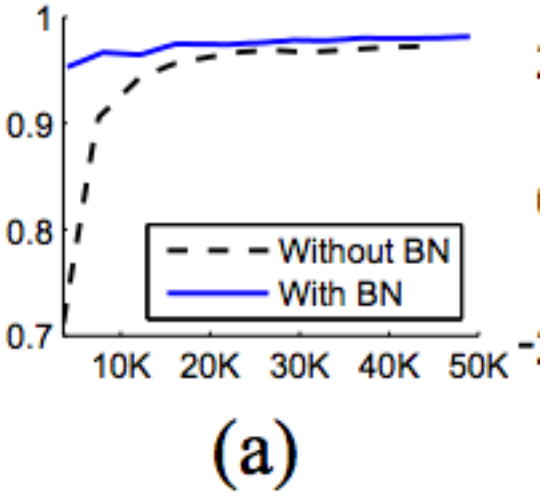
Initialization

- For biases:
  - For output units, may want to initialize to match the marginal distribution of the output
  - For other units, do not want to saturate too much at initialization (e.g., set the bias of a ReLU unit to 0.1 rather than 0 to avoid saturation)

Batch Normalization

- Technique for reducing internal covariate shift
- High-level idea: whitening the data at each layer makes training faster
- See paper or 8.7.1 in book

Batch Normalization



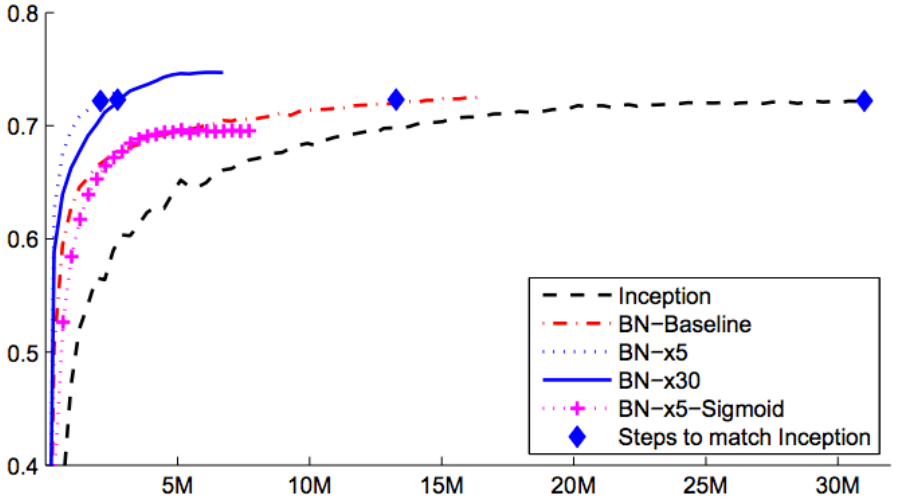


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

Left: MNIST. Right: ImageNet

Batch Normalization

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%***

Figure 4: *Batch-Normalized Inception comparison with previous state of the art on the provided validation set comprising 50000 images. *BN-Inception ensemble has reached 4.82% top-5 error on the 100000 images of the test set of the ImageNet as reported by the test server.*