Unidad 3: Técnicas de Regularización

Curso: Redes Neuronales Profundas

Regularización de parámetros

Teorema de Bayes

$$p(h|D) = \frac{p(D|h)p(h)}{p(D)}$$

Probabilidad de haber visto los datos, suponiendo la hipótesis
Likelihood. Si suponemos i.i.d., es un producto.

$$\sum_{d_i \in D} log(p(d_i|h))$$

Log likelihood

Teorema de Bayes

$$p(h|D) = \frac{p(D|h)p(h)}{p(D)}$$

Probabilidad de haber visto los datos, suponiendo la hipótesis

Likelihood. Si suponemos i.i.d., es un producto.

Primera aproximación:

Maximizar log-likelihood

Teorema de Bayes

$$p(h|D) = \frac{p(D|h)p(h)}{p(D)}$$

Likelihood multiplicado por la probabilidad a priori de la hipótesis

Segunda aproximación:
Maximum A Posteriori (MAP)

Teorema de Bayes

Podemos tratar de maximizar algo similar:

$$p(D|h)\frac{p(h)^{\alpha}}{Z}$$

$$\sum \log p(d_i|h) + \alpha \log p(h) - \log Z$$

Regularización de Parámetros

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

http://www.deeplearningbook.org/contents/regularization.html

Regularización L2 (Weight decay)

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

weight decay, ridge regression, Tikhonov regularization

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon \left(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})\right)$$

http://www.deeplearningbook.org/contents/regularization.html

Regularización L2 (Weight decay)

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon \left(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \right)$$

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

http://www.deeplearningbook.org/contents/regularization.html

Aproximación cuadrática del costo

Sea: $$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

Matriz hessiana

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

http://www.deeplearningbook.org/contents/regularization.html

Aproximación cuadrática del costo

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

$$\alpha \tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0$$

$$(\boldsymbol{H} + \alpha \boldsymbol{I})\tilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^*$$

$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha \boldsymbol{I})^{-1} \boldsymbol{H}\boldsymbol{w}^*$$

http://www.deeplearningbook.org/contents/regularization.html

Aproximación cuadrática del costo

$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha \boldsymbol{I})^{-1} \boldsymbol{H} \boldsymbol{w}^*$$

$$\boldsymbol{H} = \boldsymbol{Q} \boldsymbol{\Lambda} \boldsymbol{Q}^\top$$

matriz diagonal

matriz ortonormal

$$\tilde{\boldsymbol{w}} = (\boldsymbol{Q} \boldsymbol{\Lambda} \boldsymbol{Q}^\top + \alpha \boldsymbol{I})^{-1} \boldsymbol{Q} \boldsymbol{\Lambda} \boldsymbol{Q}^\top \boldsymbol{w}^*$$

http://www.deeplearningbook.org/contents/regularization.html

Aproximación cuadrática del costo

$$\tilde{\boldsymbol{w}} = (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top + \alpha \boldsymbol{I})^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \boldsymbol{w}^*$$

$$= \left[\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})\boldsymbol{Q}^\top\right]^{-1} \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \boldsymbol{w}^*$$

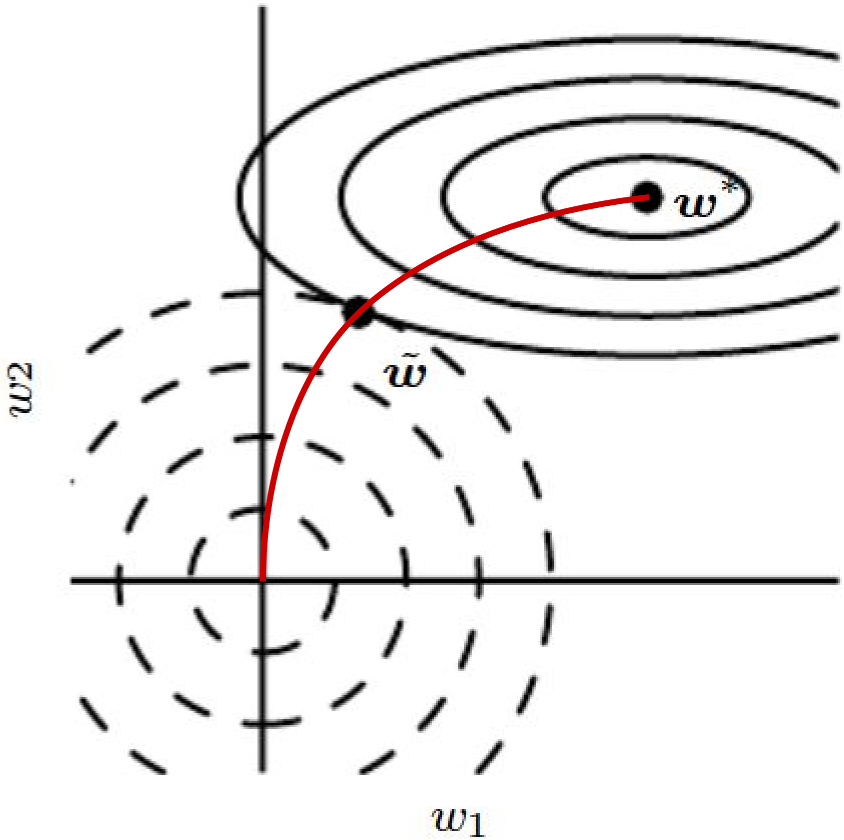
$$= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \boldsymbol{\Lambda}\boldsymbol{Q}^\top \boldsymbol{w}^*$$

scaling factor: $\dfrac{\lambda_i}{\lambda_i + \alpha}$

http://www.deeplearningbook.org/contents/regularization.html

Interpretación gráfica

$$\tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top\boldsymbol{w} + J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y})$$



$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$

$$\begin{aligned}\tilde{\boldsymbol{w}} &= (\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^* \\ &= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^*\end{aligned}$$

$$\boxed{\alpha\tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0}$$

http://www.deeplearningbook.org/contents/regularization.html

Ejemplo: regresión lineal

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^\top (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) + \frac{1}{2}\alpha \boldsymbol{w}^\top \boldsymbol{w}$$

$$\boldsymbol{w} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

$$\boldsymbol{w} = (\boldsymbol{X}^\top \boldsymbol{X} + \alpha \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

http://www.deeplearningbook.org/contents/regularization.html

Regularización L1

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|$$

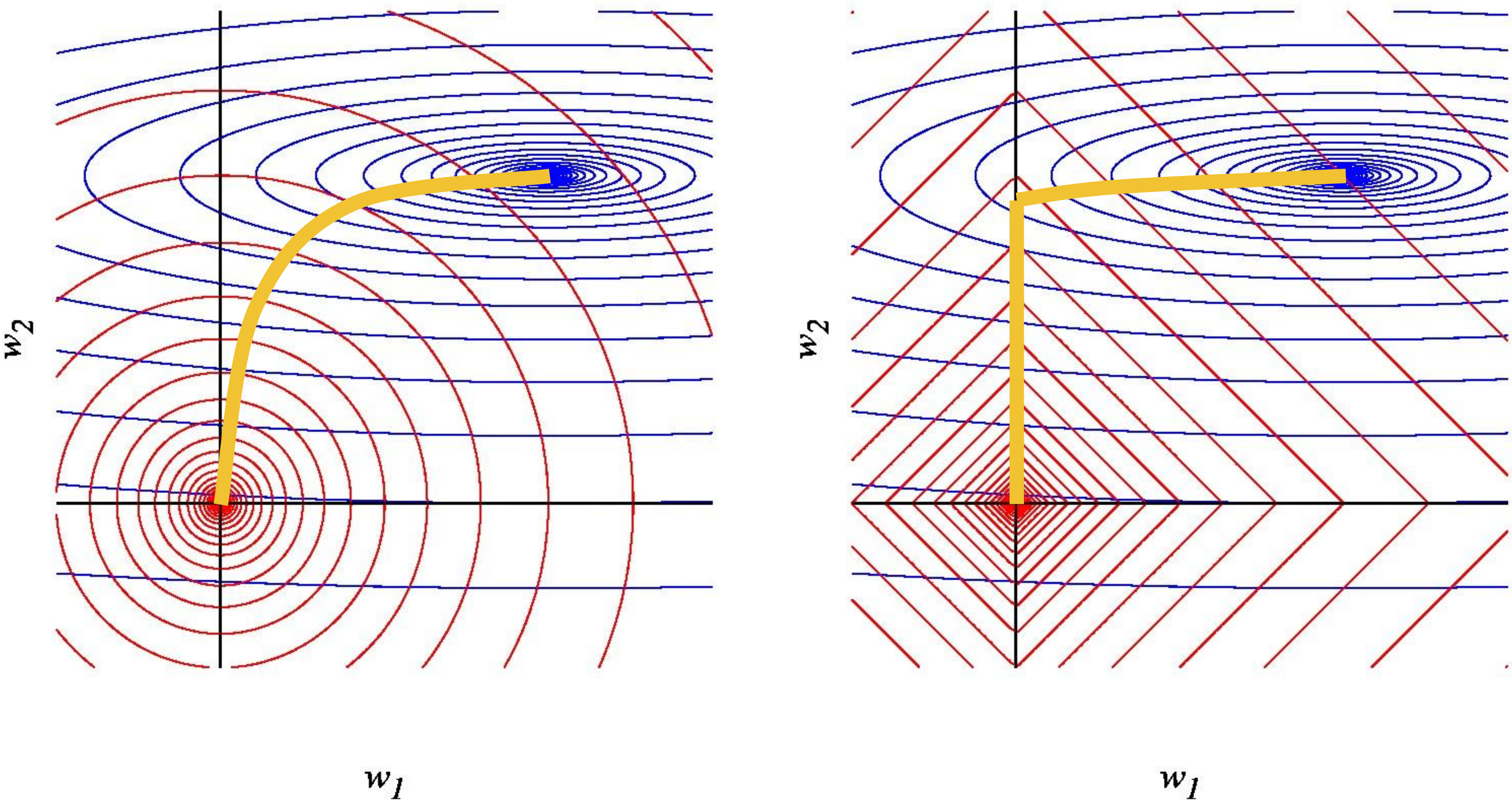
$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha ||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \mathrm{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w})$$

http://www.deeplearningbook.org/contents/regularization.html

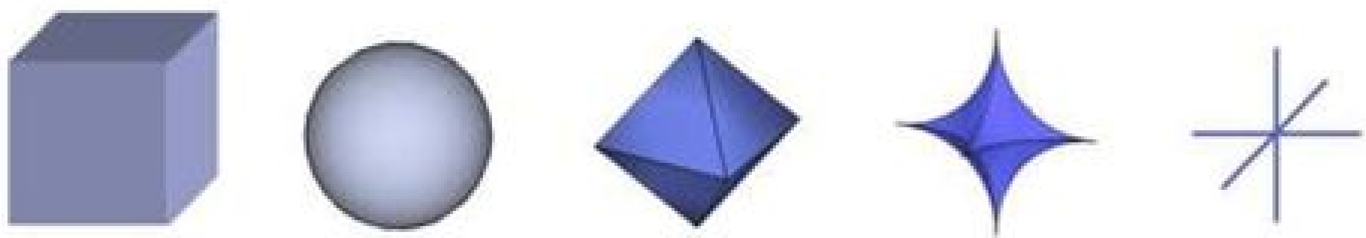
L2 vs L1

L2 L1



Generalización de la norma

$$p\text{-norm of } \theta \qquad \|\theta\|_p = \left(\sum_{j=0} |\theta_j|^p \right)^{\frac{1}{p}}$$

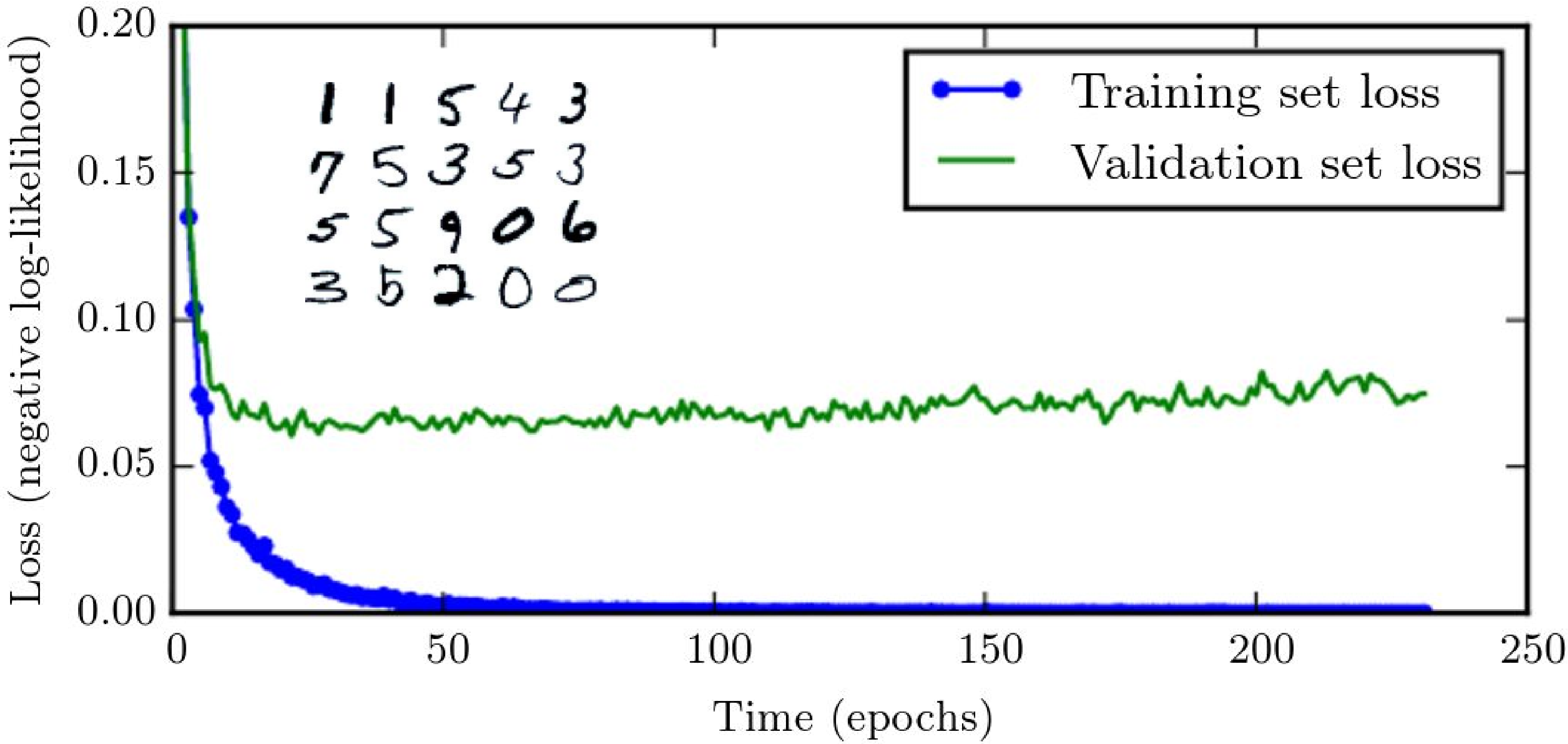


$p = \infty$ $\quad$ $p = 2$ $\quad$ $p = 1$ $\quad$ $0 < p < 1$ $\quad$ $p = 0$

Detención temprana

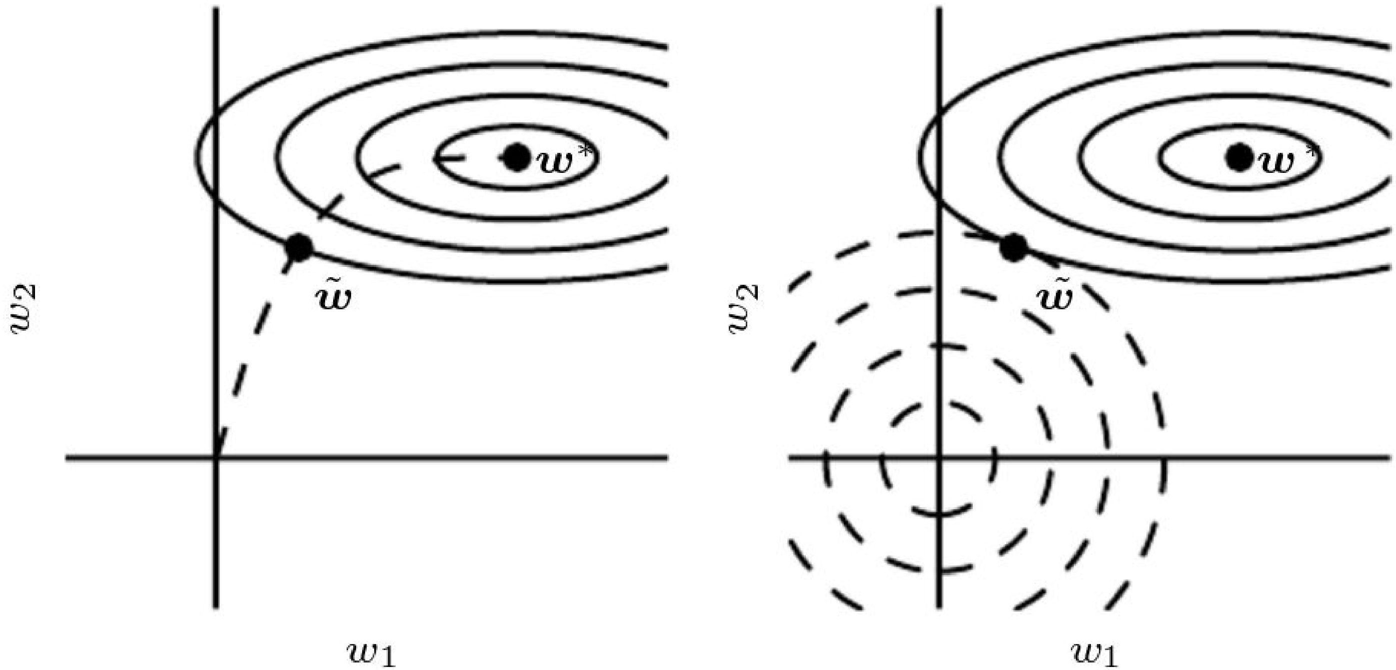
Detención temprana

maxout network on MNIST



http://www.deeplearningbook.org/contents/regularization.html

Detención temprana

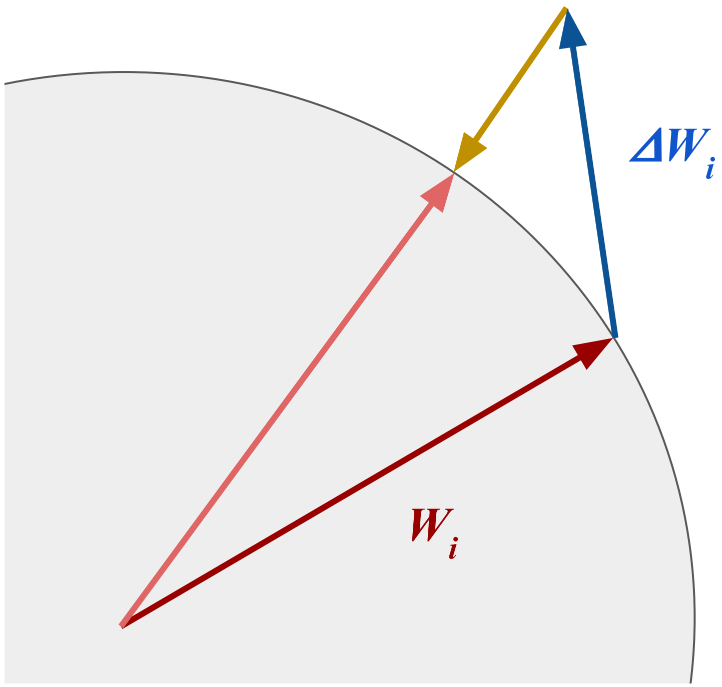


http://www.deeplearningbook.org/contents/regularization.html

Restricciones a los parámetros

Restricciones

- Norma de la matriz W
- Signo de los bias
- Ortonormalización de W



En optimización con restricciones clásica se utilizan multiplicadores de Lagrange pero en la literatura de Deep Learning es común ir proyectando los parámetros después de cada actualización

Max Norm Constraint

```python
class MaxNorm(Constraint):

    def __init__(self, m=2.0):
        self.m = m

    def __call__(self, p):
        norms = T.sqrt(T.sum(T.square(p), axis=0, keepdims=True))
        desired = T.clip(norms, 0, self.m)
        p = p * (desired / (1e-7 + norms))
        return p
```

SGD con MaxNorm

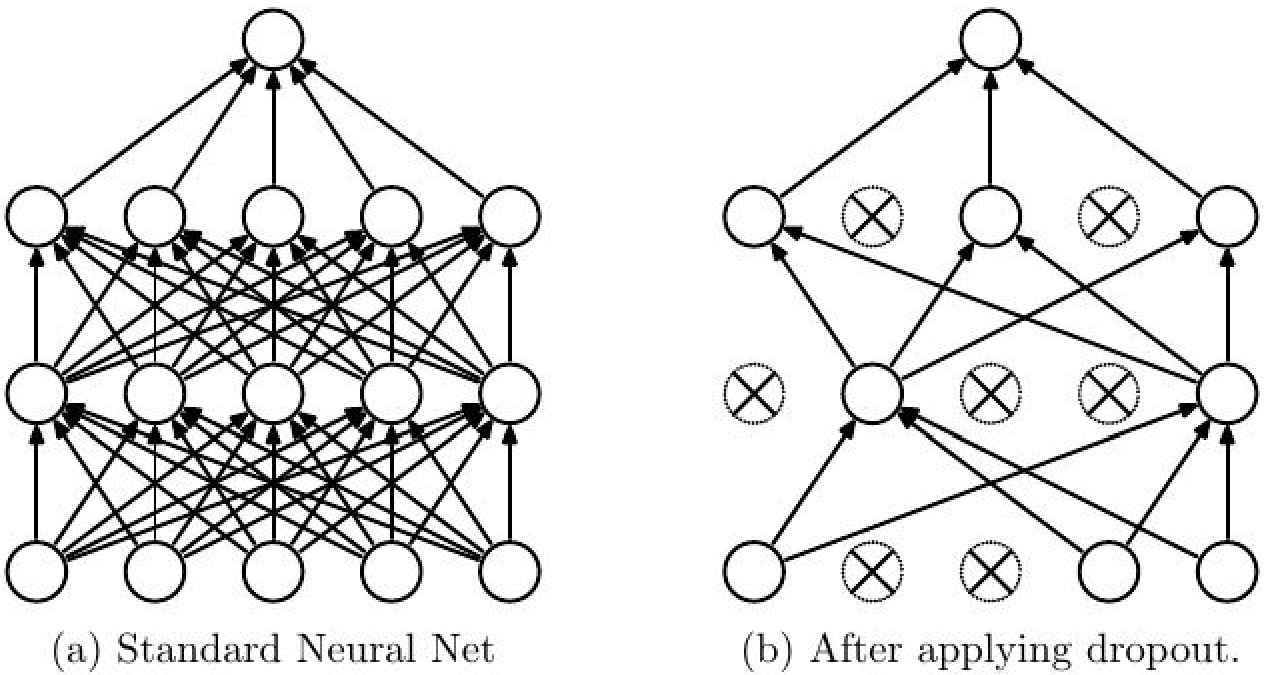
```
class SGD:

    def __init__(self, lr=0.01):
        self.lr = lr

    def __call__(self, params, cost):
        updates = []
        grads = T.grad(cost, params)
        for p,g in zip(params,grads):
            updated_p = p - self.lr * g
            updated_p = max_norm(updated_p)
            updates.append((p, updated_p))
        return updates
```

Dropout:

A Simple Way to Prevent Neural Networks from Overfitting

Srivastava, Nitish, et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research* 15 (2014): 1929-1958.

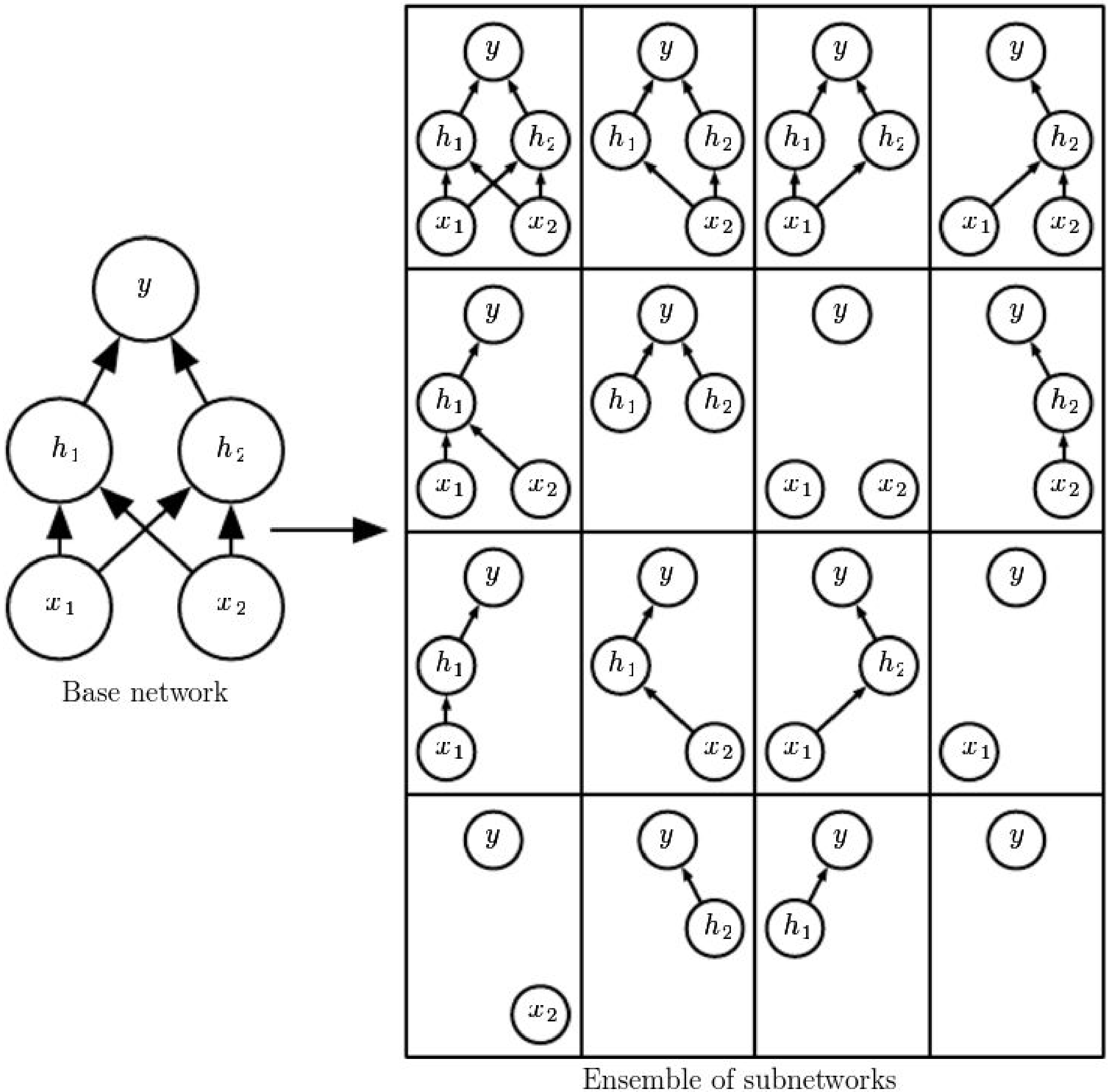
Dropout Neural Net Model



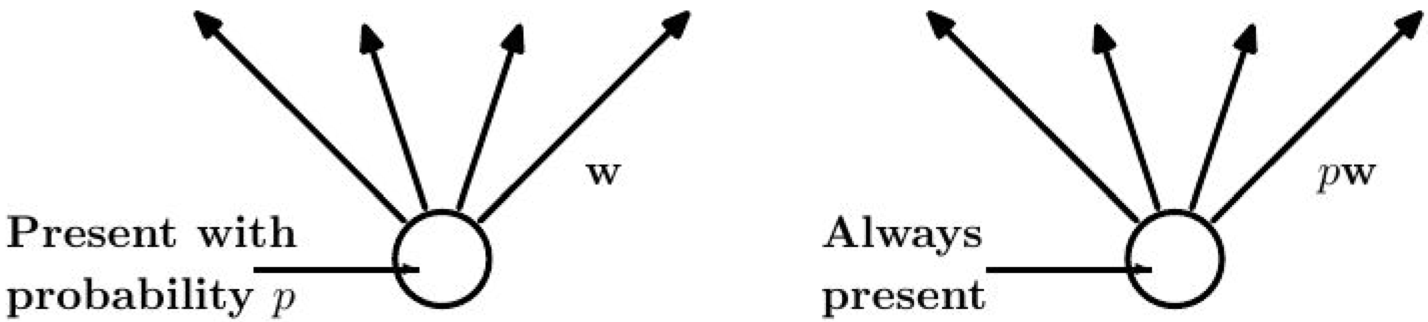
Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Srivastava, Nitish, et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research* 15 (2014): 1929-1958.

Dropout Neural Net Model



Dropout units



(a) At training time (b) At test time

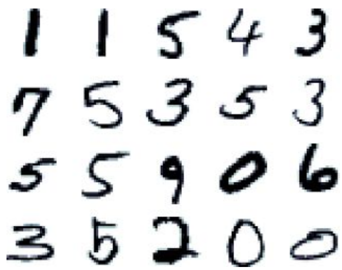
Left: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Srivastava, Nitish, et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research* 15 (2014): 1929-1958.

Dropout en Theano

```
from theano.sandbox.rng_mrg import MRG_RandomStreams as RandomStreams
t_rn = RandomStreams(123)

def dropout(X, p=0.):
    """
    dropout using activation scaling to avoid test time
    weight rescaling
    """
    if p > 0:
        retain_prob = 1 - p
        X *= t_rng.binomial(X.shape,
                            p=retain_prob,
                            dtype=theano.config.floatX)
        X /= retain_prob
    return X
```

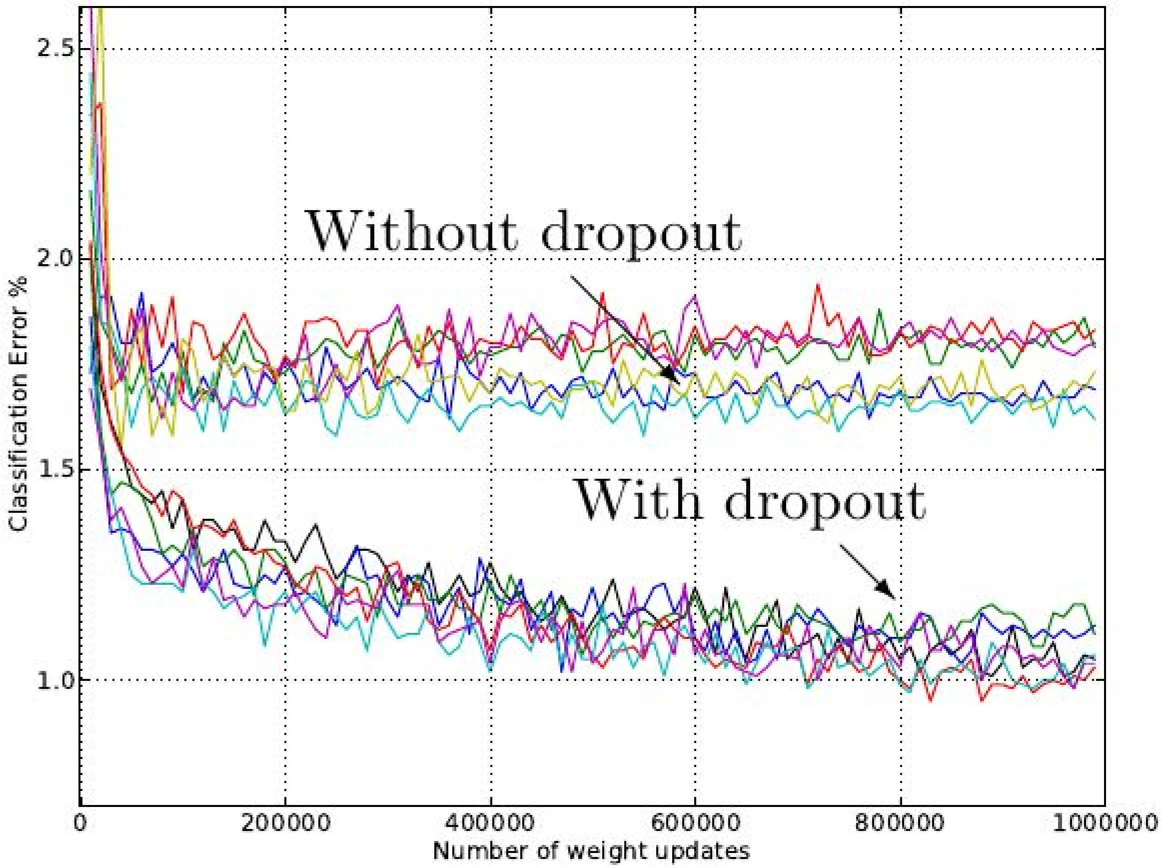
MNIST results



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**

Srivastava, Nitish, et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research* 15 (2014): 1929-1958.

Dropout: Robustness



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

Srivastava, Nitish, et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research* 15 (2014): 1929-1958.

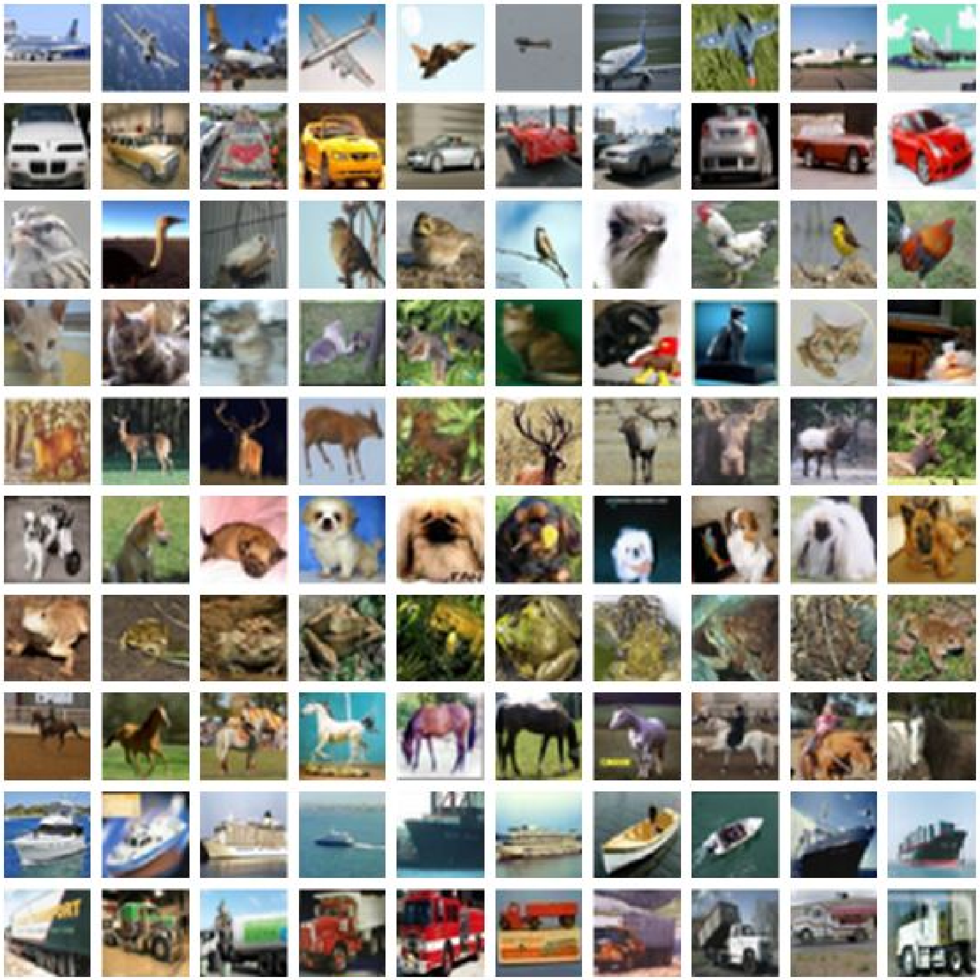
Dropout: Street View House Numbers

Method	Error %
Binary Features (WDCH) (Netzer et al., 2011)	36.7
HOG (Netzer et al., 2011)	15.0
Stacked Sparse Autoencoders (Netzer et al., 2011)	10.3
KMeans (Netzer et al., 2011)	9.4
Multi-stage Conv Net with average pooling (Sermanet et al., 2012)	9.06
Multi-stage Conv Net + L2 pooling (Sermanet et al., 2012)	5.36
Multi-stage Conv Net + L4 pooling + padding (Sermanet et al., 2012)	4.90
Conv Net + max-pooling	3.95
Conv Net + max pooling + dropout in fully connected layers	3.02
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	2.80
Conv Net + max pooling + dropout in all layers	2.55
Conv Net + maxout (Goodfellow et al., 2013)	**2.47**
Human Performance	2.0

Table 3: Results on the Street View House Numbers data set.

Srivastava, Nitish, et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research* 15 (2014): 1929-1958.

CIFAR-10 and CIFAR-100 datasets

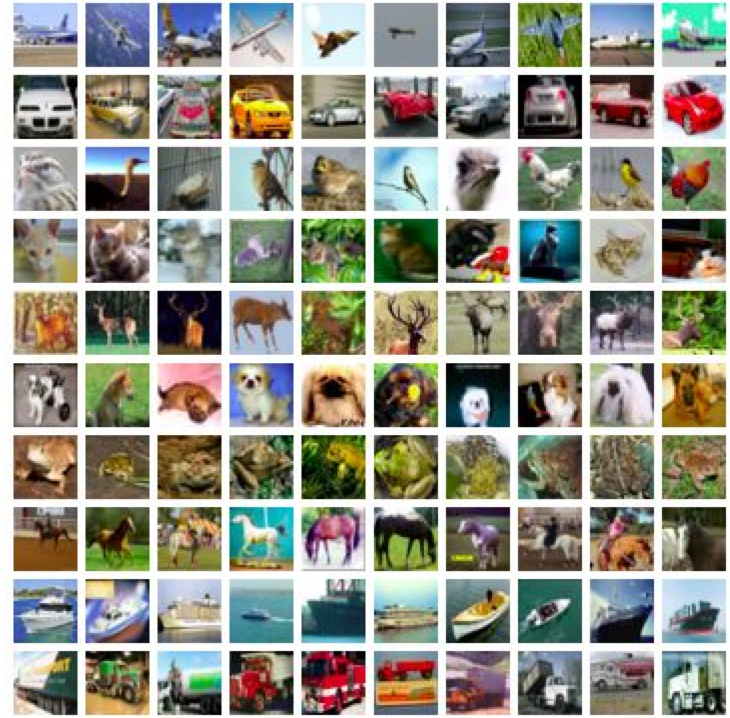


The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

Srivastava, Nitish, et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research* 15 (2014): 1929-1958.

Dropout: CIFAR

Error rates on CIFAR-10 and CIFAR-100.



Method	CIFAR-10	CIFAR-100
Conv Net + max pooling (hand tuned)	15.60	43.48
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	15.13	42.51
Conv Net + max pooling (Snoek et al., 2012)	14.98	-
Conv Net + max pooling + dropout fully connected layers	14.32	41.26
Conv Net + max pooling + dropout in all layers	12.61	**37.20**
Conv Net + maxout (Goodfellow et al., 2013)	**11.68**	38.57

Srivastava, Nitish, et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research* 15 (2014): 1929-1958.

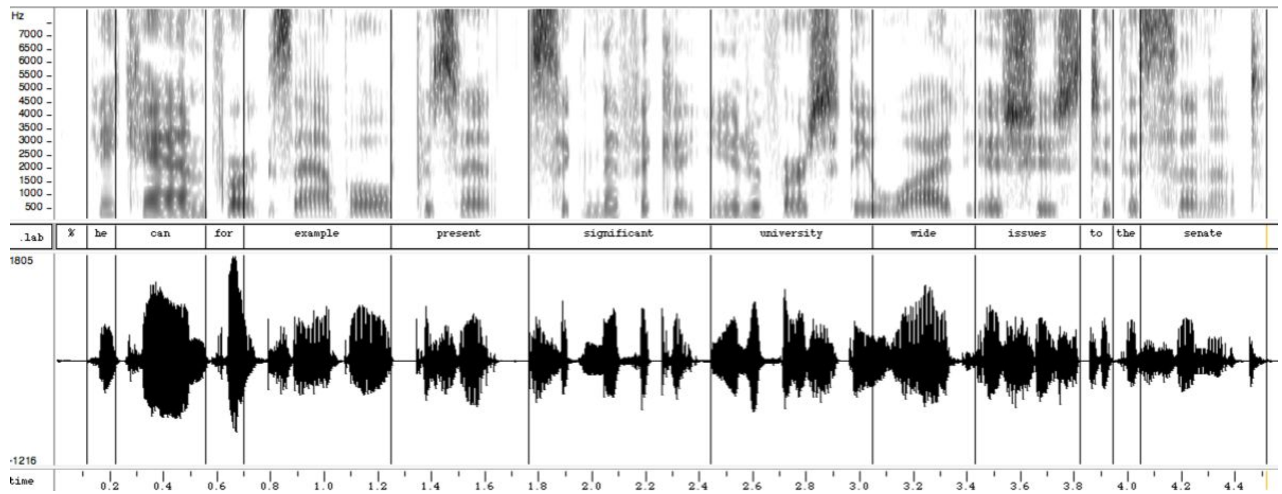
Dropout: ImageNet

Model	Top-1	Top-5
Sparse Coding (Lin et al., 2010)	47.1	28.2
SIFT + Fisher Vectors (Sanchez and Perronnin, 2011)	45.7	25.7
Conv Net + dropout (Krizhevsky et al., 2012)	37.5	17.0

Results on the ILSVRC-2010 test set

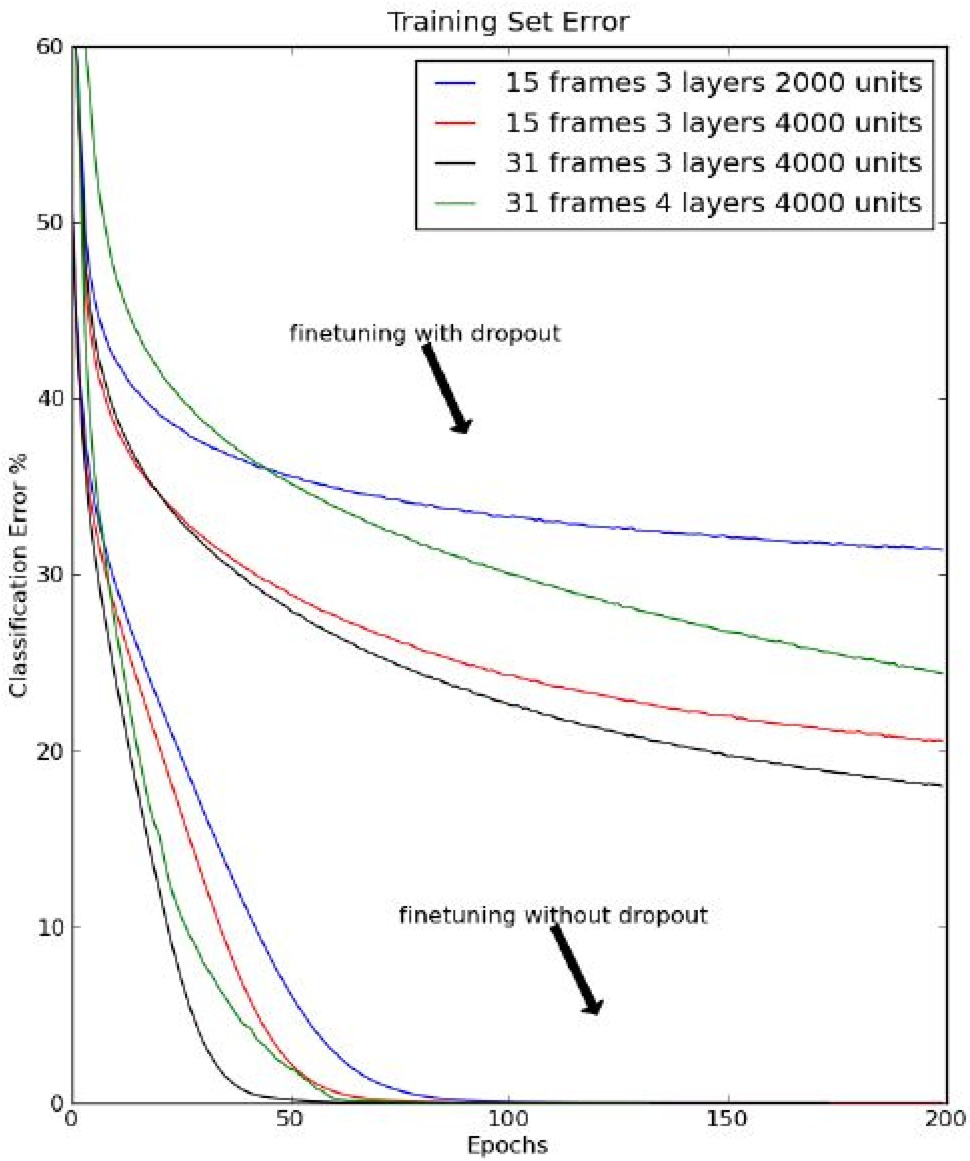
Dropout: TIMIT

A standard speech benchmark for clean speech recognition.



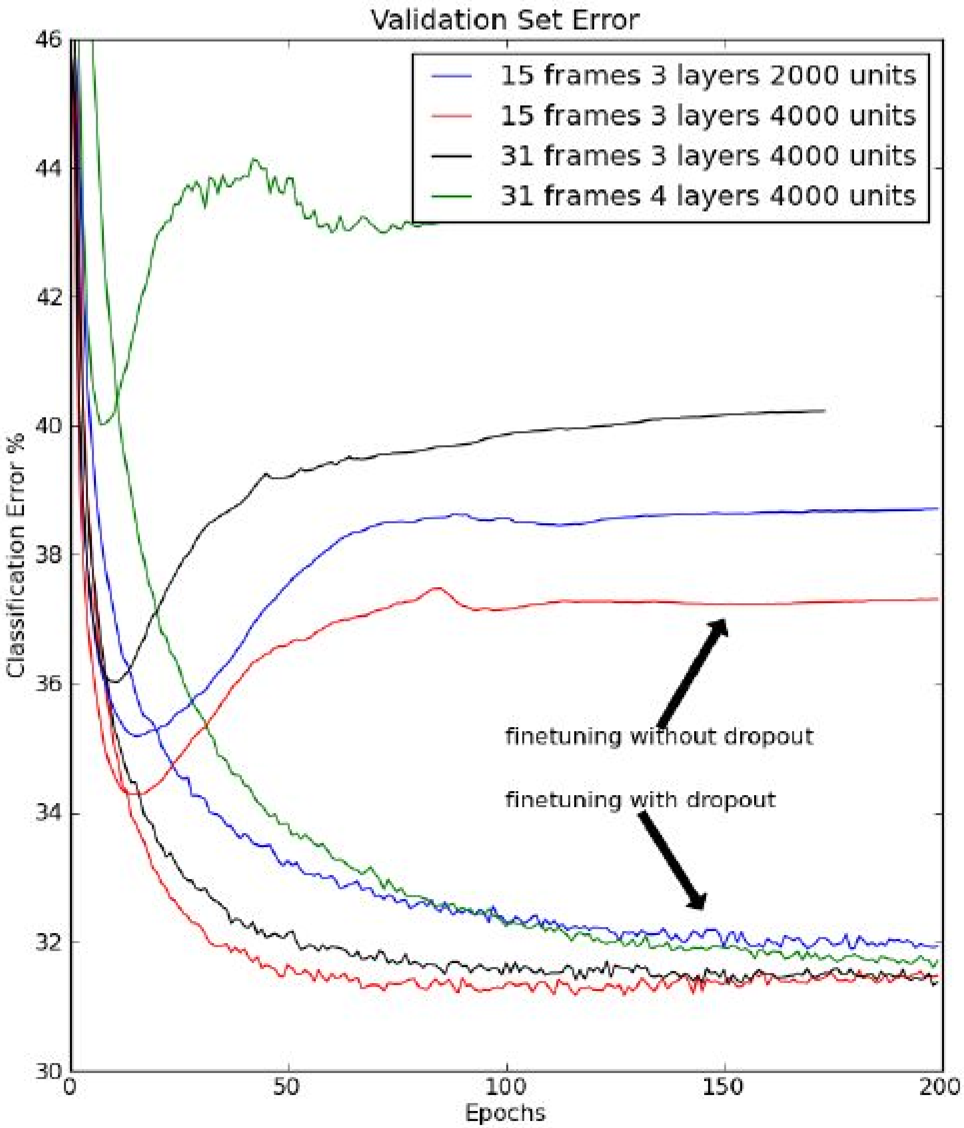
Method	Phone Error Rate%
NN (6 layers) (Mohamed et al., 2010)	23.4
Dropout NN (6 layers)	21.8
DBN-pretrained NN (4 layers)	22.7
DBN-pretrained NN (6 layers) (Mohamed et al., 2010)	22.4
DBN-pretrained NN (8 layers) (Mohamed et al., 2010)	20.7
mcRBM-DBN-pretrained NN (5 layers) (Dahl et al., 2010)	20.5
DBN-pretrained NN (4 layers) + dropout	**19.7**
DBN-pretrained NN (8 layers) + dropout	**19.7**

Dropout



TIMIT Dataset

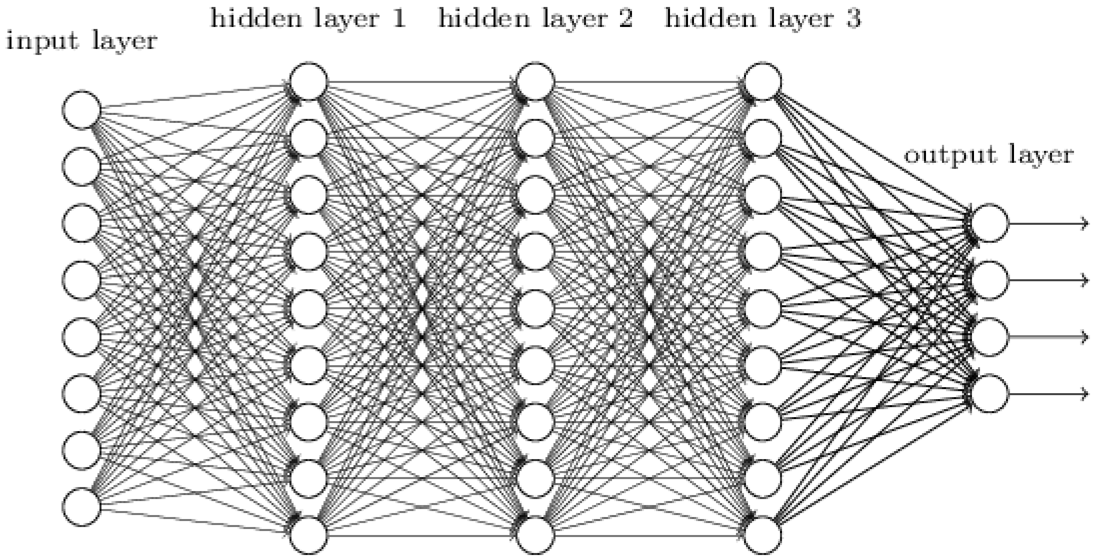
Dropout


Validation Set Error
15 frames 3 layers 2000 units
15 frames 3 layers 4000 units
31 frames 3 layers 4000 units
31 frames 4 layers 4000 units
finetuning without dropout
finetuning with dropout
Classification Error %
Epochs
46
44
42
40
38
36
34
32
30
0
50
100
150
200

Representaciones sparse

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{h})$$

$$\Omega(\boldsymbol{h}) = ||\boldsymbol{h}||_1 = \sum_i |h_i|$$



Ejemplo en Keras

```
keras.layers.core.Dense(output_dim,
                        init='glorot_uniform',
                        activation='linear',
                        weights=None,
                        W_regularizer=None,
                        b_regularizer=None,
                        activity_regularizer=None,
                        W_constraint=None,
                        b_constraint=None,
                        bias=True,
                        input_dim=None)
```

Inicializaciones (nota al margen)

```python
def glorot_normal(shape, name=None, dim_ordering='th'):
    ''' Reference: Glorot & Bengio, AISTATS 2010
    '''
    fan_in, fan_out = get_fans(shape, dim_ordering=dim_ordering)
    s = np.sqrt(2. / (fan_in + fan_out))
    return normal(shape, s, name=name)

def he_normal(shape, name=None, dim_ordering='th'):
    ''' Reference:  He et al., http://arxiv.org/abs/1502.01852
    '''
    fan_in, fan_out = get_fans(shape, dim_ordering=dim_ordering)
    s = np.sqrt(2. / fan_in)
    return normal(shape, s, name=name)
```

https://github.com/fchollet/keras/blob/master/keras/initializations.py

Ejemplo en Keras

```
keras.layers.core.Dense(output_dim,
                        init='glorot_uniform',
                        activation='linear',
                        weights=None,
                        W_regularizer=None,
                        b_regularizer=None,
                        activity_regularizer=None,
                        W_constraint=None,
                        b_constraint=None,
                        bias=True,
                        input_dim=None)
```

- `W_regularizer`: instance of `keras.regularizers.WeightRegularizer`
- `b_regularizer`: instance of `keras.regularizers.WeightRegularizer`
- `activity_regularizer`: instance of `keras.regularizers.ActivityRegularizer`

Ejemplo en Keras

```
from keras.regularizers import l2, activity_l2
model.add(Dense(64, input_dim=64, W_regularizer=l2(0.01),
activity_regularizer=activity_l2(0.01)))
```

Available penalties

```
keras.regularizers.WeightRegularizer(l1=0., l2=0.)
keras.regularizers.ActivityRegularizer(l1=0., l2=0.)
```

Shortcuts

- **l1**(l=0.01): L1 weight regularization penalty, also known as LASSO
- **l2**(l=0.01): L2 weight regularization penalty, also known as weight decay, or Ridge
- **l1l2**(l1=0.01, l2=0.01): L1-L2 weight regularization penalty, also known as ElasticNet
- **activity_l1**(l=0.01): L1 activity regularization
- **activity_l2**(l=0.01): L2 activity regularization
- **activity_l1l2**(l1=0.01, l2=0.01): L1+L2 activity regularization

Batch Normalization

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

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

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

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

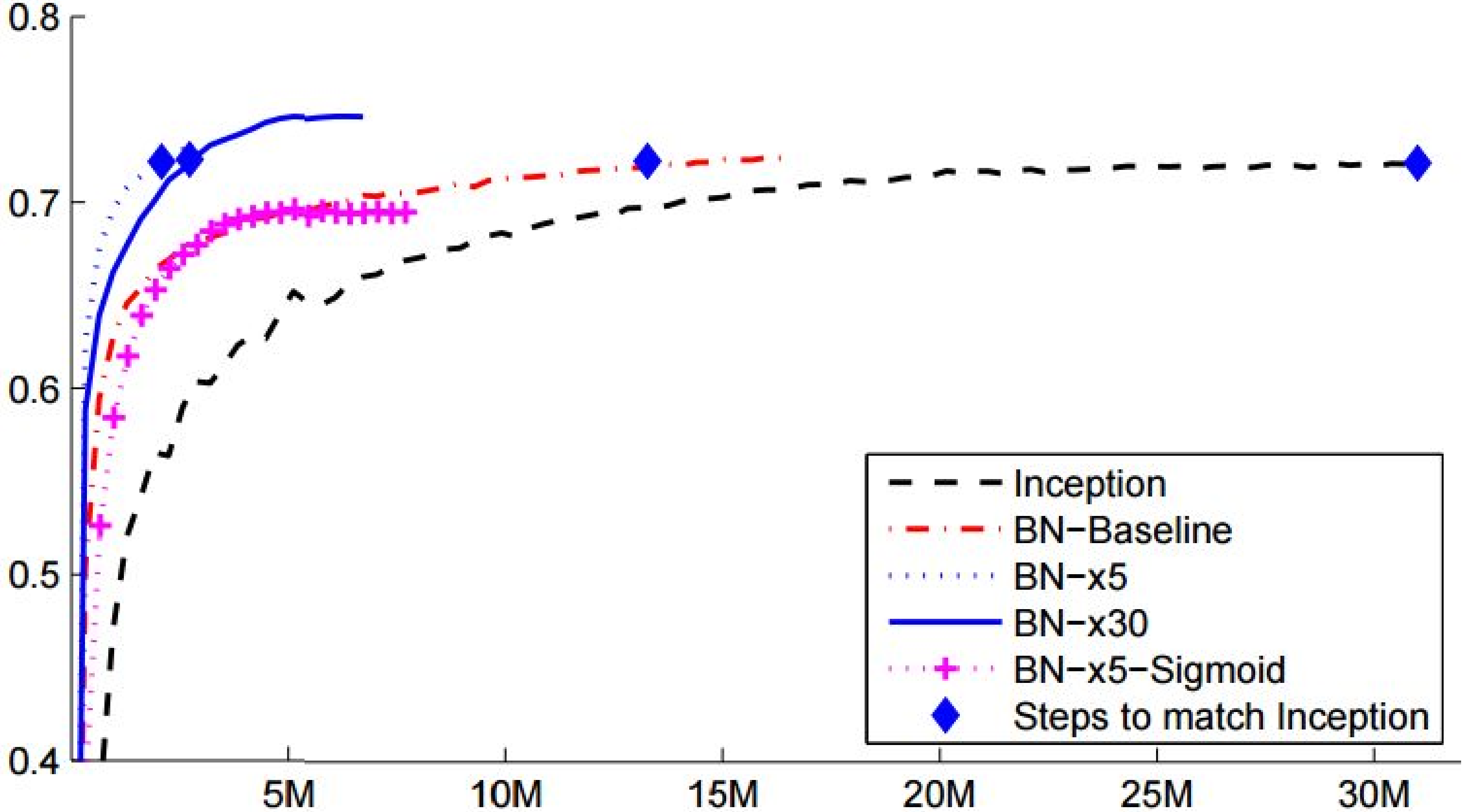
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

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

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)* (pp. 448-456).

Batch Normalization

ImageNet classification task



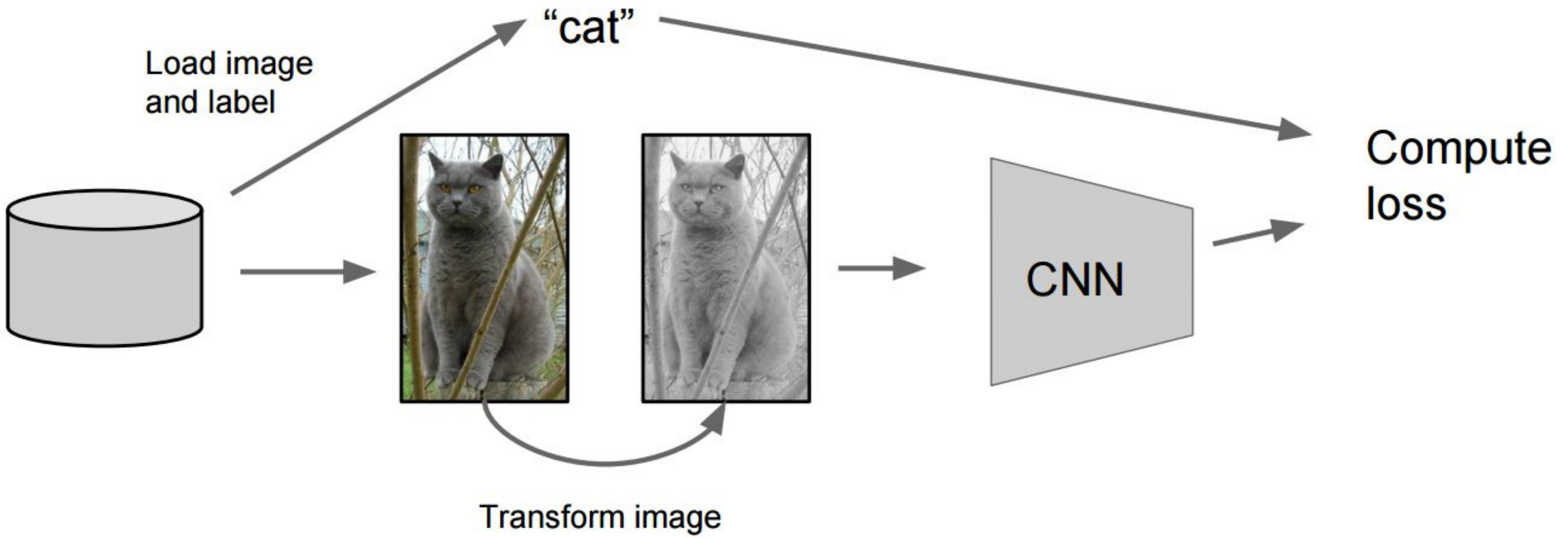
Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)* (pp. 448-456).

Batch Normalization

- Increase learning rate.
- Remove Dropout. Removing Dropout from BN-Inception allows the network to achieve higher validation accuracy.
- Shuffle training examples more thoroughly. This led to about 1% improvement in the validation accuracy.
- Reduce the L2 weight regularization by a factor of 5. This improves the accuracy on the held-out validation data.
- Accelerate the learning rate decay (6 times faster).
- Remove Local Response Normalization.
- Reduce the "photometric distortions" [Howard, Andrew G. "Some improvements on deep convolutional neural network based image classification." arXiv preprint arXiv:1312.5402 (2013).]

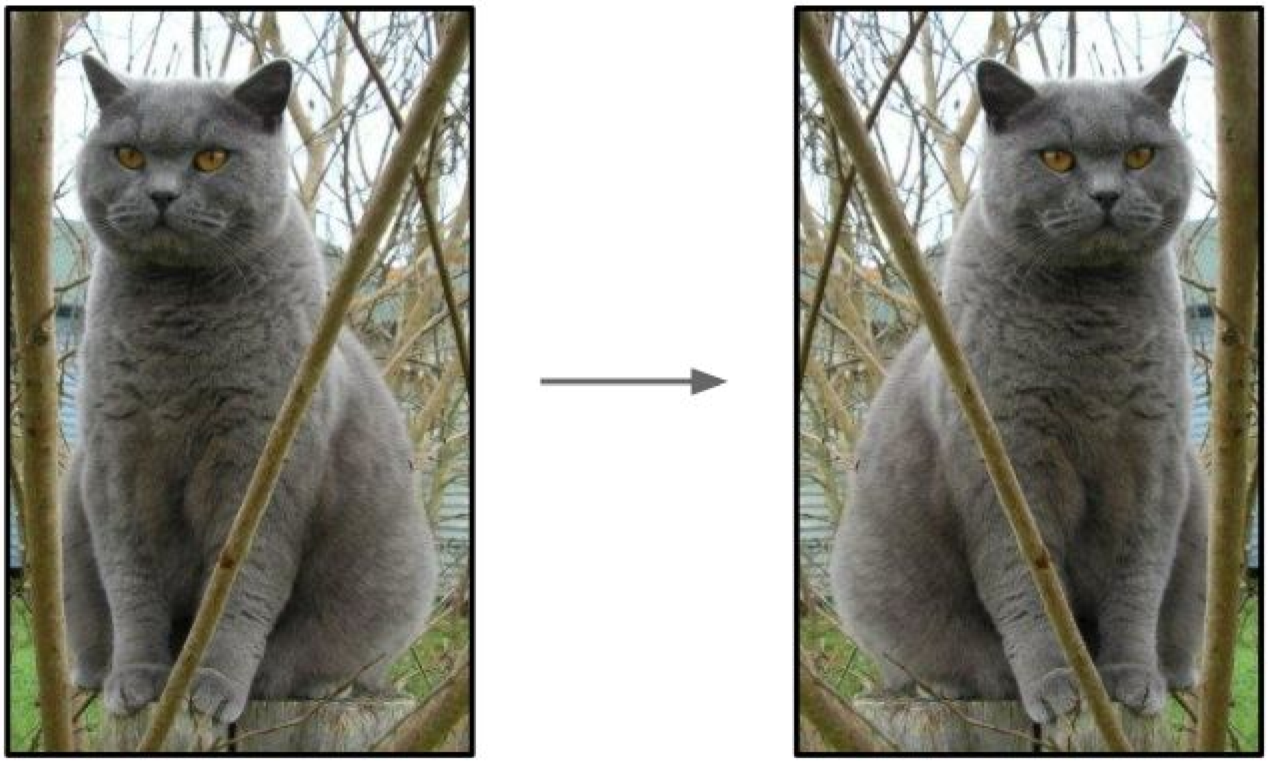
Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)* (pp. 448-456).

Data Augmentation



http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Data Augmentation: Horizontal flips

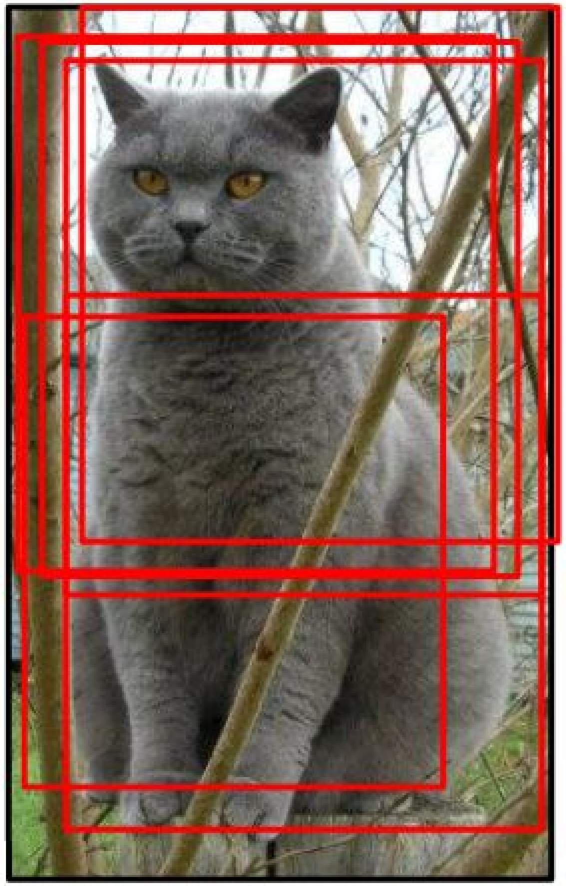


http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Data Augmentation: Random crops/scales

Training: sample random crops / scales
Por ejemplo [ResNet]:
1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops
Por ejemplo [ResNet]
1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips



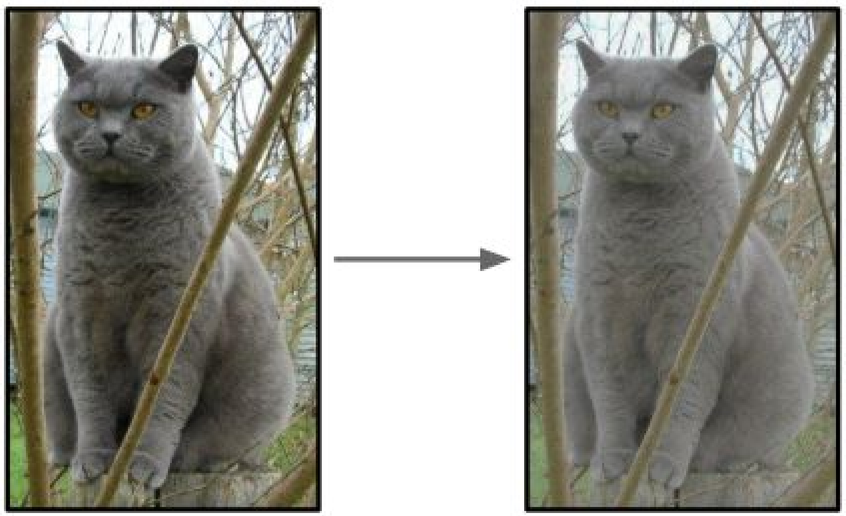
[ResNet] He, Kaiming, et al. "Deep residual learning for image recognition." arXiv preprint arXiv:1512.03385 (2015).

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Data Augmentation: Color jitter

Versión simple:

Alterar el contraste aleatoriamente



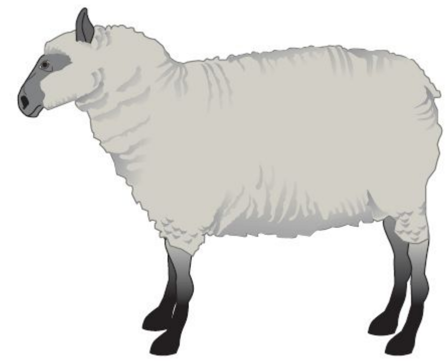
Versión compleja:

1. Aplicar PCA a todos los píxeles [R,G,B] del training set
2. Samplear un "desplazamiento de color" a lo largo de las direcciones principales
3. Aplicarle este desplazamiento a todos los píxeles de una imagen de entrenamiento.

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

en Keras:

```
keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    shear_range=0.,
    zoom_range=0.,
    channel_shift_range=0.,
    fill_mode='nearest',
    cval=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    dim_ordering=K.image_dim_ordering())
```



sheep

sheared sheep

Algunos argumentos:

rotation_range: Degree range for random rotations.

width_shift_range: (fraction of total width). Range for random horizontal shifts.

height_shift_range: (fraction of total height). Range for random vertical shifts.

shear_range: Shear Intensity (Shear angle in counter-clockwise direction as radians)

zoom_range: Range for random zoom.

channel_shift_range: Range for random channel shifts.

fill_mode: One of {"constant", "nearest", "reflect" or "wrap"}.

horizontal_flip: Boolean. Randomly flip inputs horizontally.

vertical_flip: Boolean. Randomly flip inputs vertically.

rescale: rescaling factor. Defaults to None. If None or 0, no rescaling is applied, otherwise we multiply the data by the value provided (before applying any other transformation).

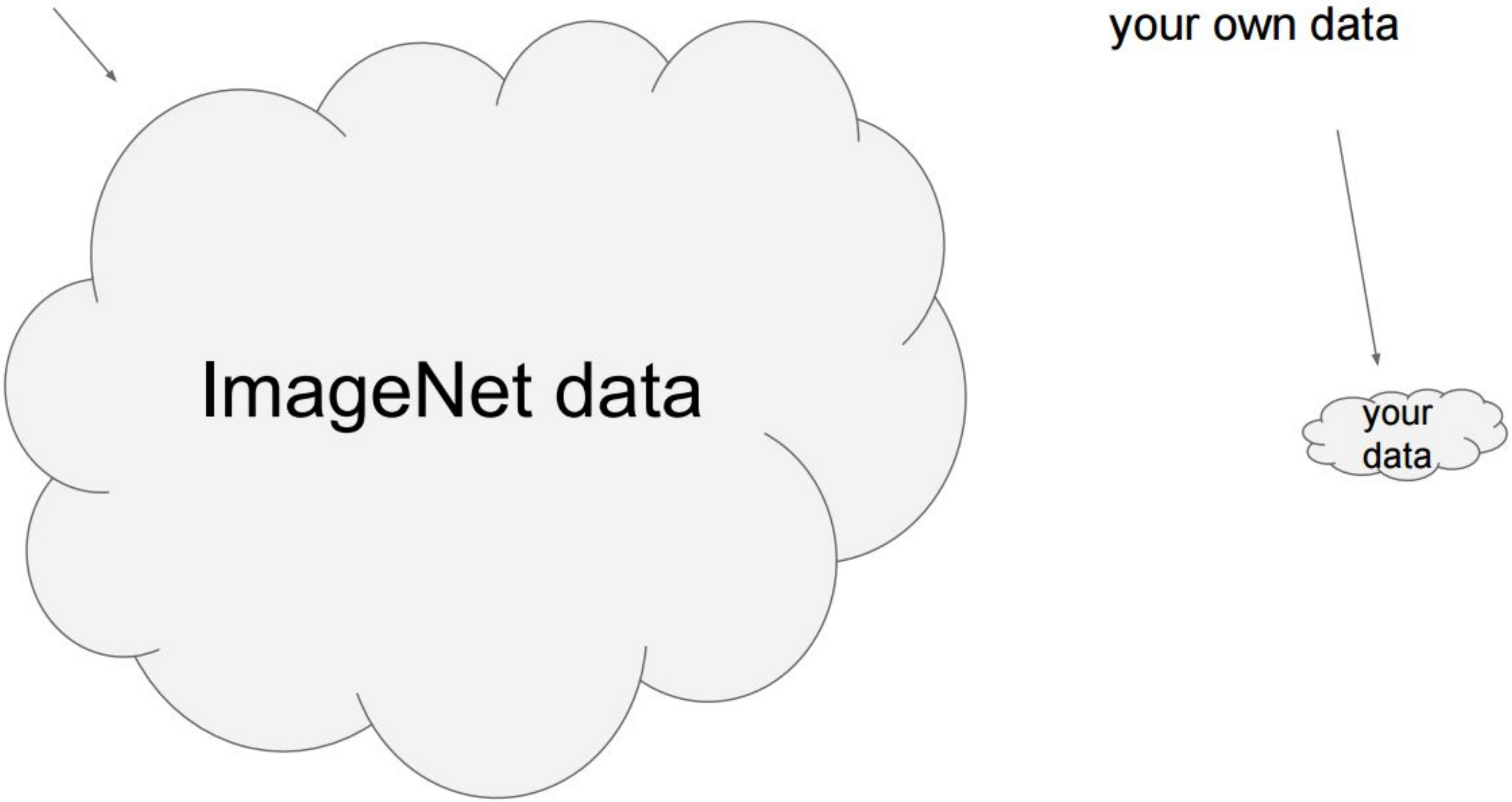
https://keras.io/preprocessing/image/

Transfer Learning

Pre entrenamiento sobre ImageNet

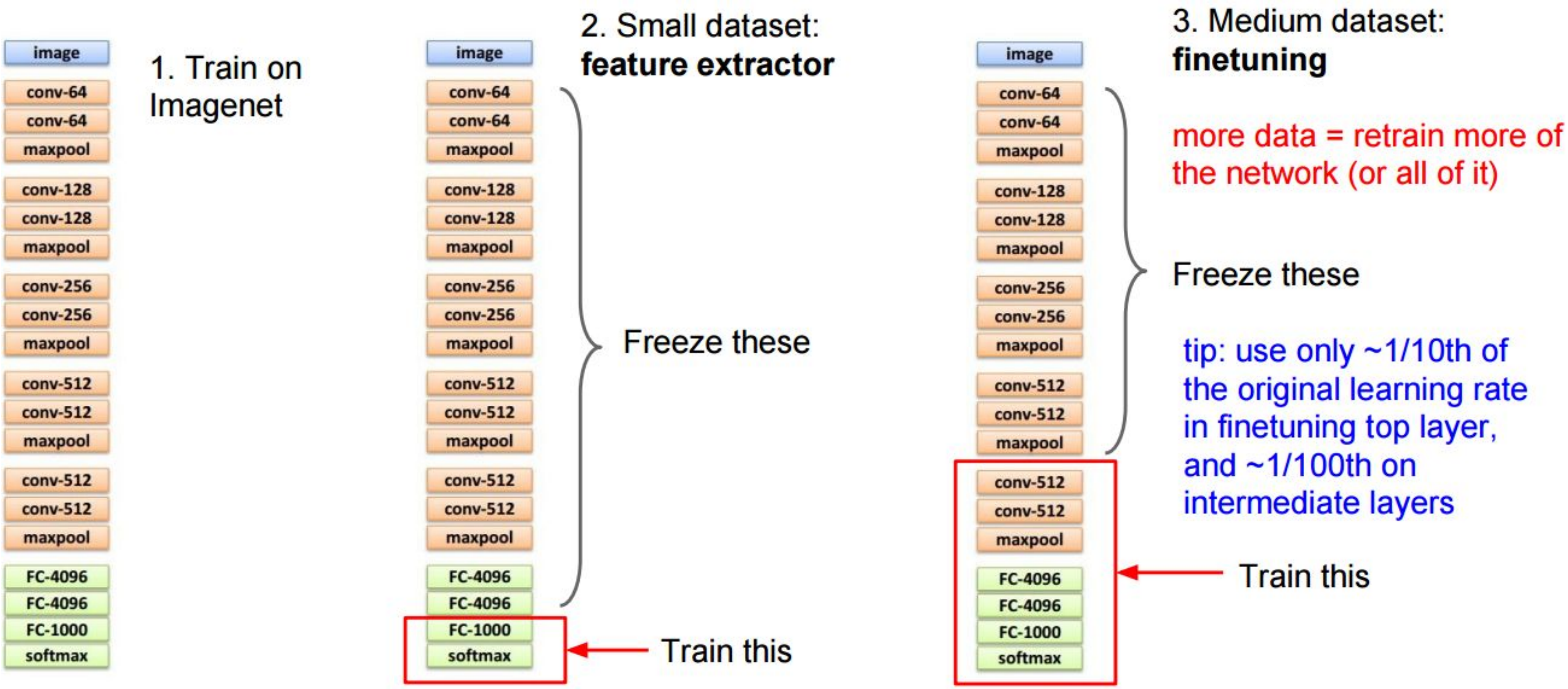
1. Train on ImageNet

2. Finetune network on your own data



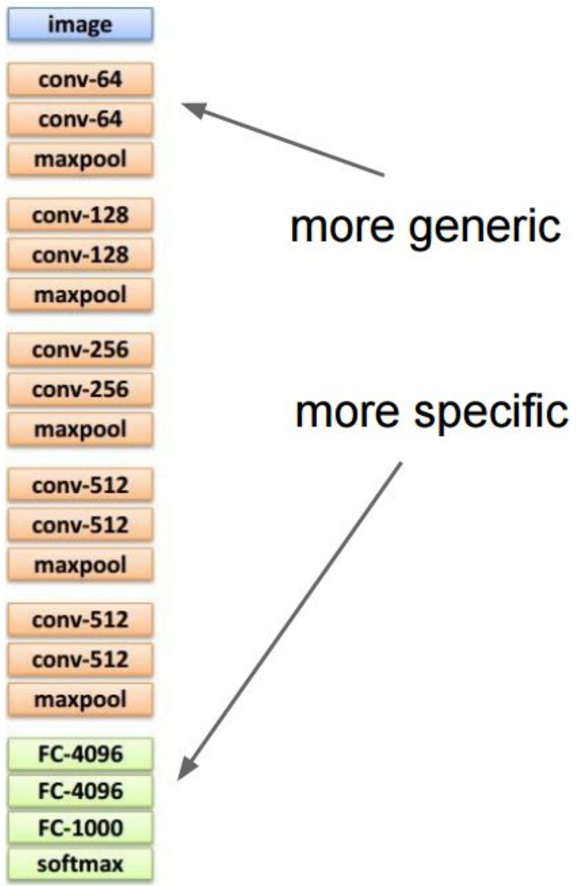
http://cs231n.stanford.edu/syllabus.html

Ejemplo con redes convolucionales



http://cs231n.stanford.edu/syllabus.html

Ejemplo con redes convolucionales



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

http://cs231n.stanford.edu/syllabus.html