The Gory Details of Neural Network Training: A Gentle Overview

Outline

- Optimization
  - Mini-batch SGD
  - Learning rate decay
  - Adaptive methods
- Massaging the numbers
  - Data augmentation
  - Data preprocessing
  - Weight initialization
  - Batch normalization
- Regularization
  - Classic regularization: L2 and L1
  - Dropout
  - Label smoothing
- Test time: ensembles, averaging predictions

A not-so-deep dive into optimization

D. Hockney, Pool with two figures, 1972

Mini-batch SGD

- Iterate over epochs
  - Group data into mini-batches of size b
    - Compute gradient of the loss for the mini-batch $(x_1, y_1), \ldots, (x_b, y_b)$:

$$\nabla \hat{L} = \frac{1}{b} \sum_{i=1}^{b} \nabla l(w, x_i, y_i)$$

    - Update parameters:

$$w \leftarrow w - \eta \nabla \hat{L}$$

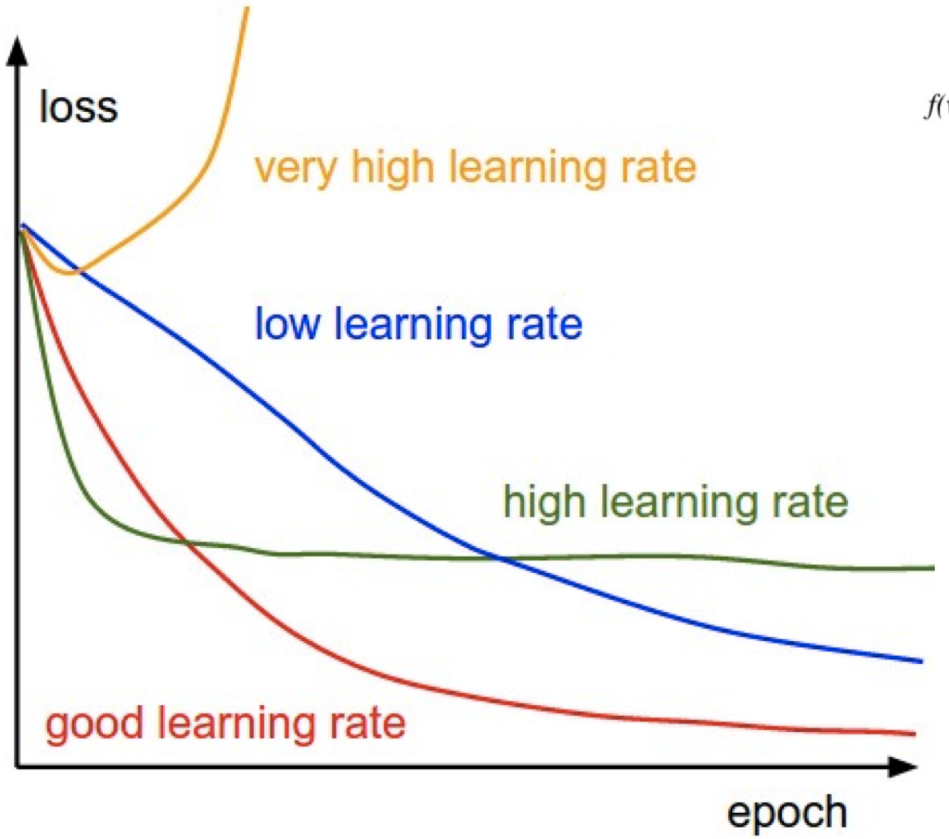
  - Check for convergence, decide whether to decay learning rate

- What are the hyperparameters?
  - Mini-batch size, learning rate decay schedule, deciding when to stop

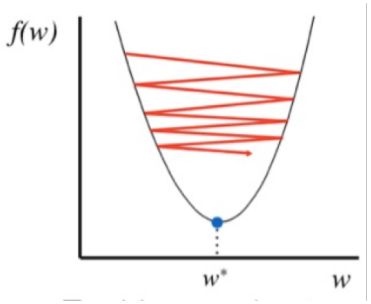
Setting the mini-batch size

- Larger mini-batches: more expensive and less frequent updates, lower gradient variance, more parallelizable
- SGD with larger batches may generalize more poorly (e.g., Keskar et al., 2017)
- But can be made to work well by carefully controlling learning rate and addressing other optimization issues (Goyal et al., 2018)

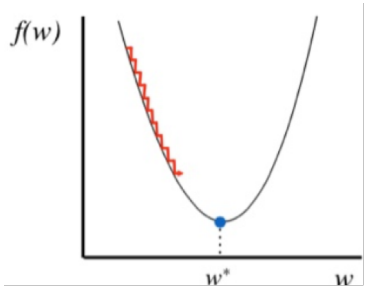
Setting the learning rate



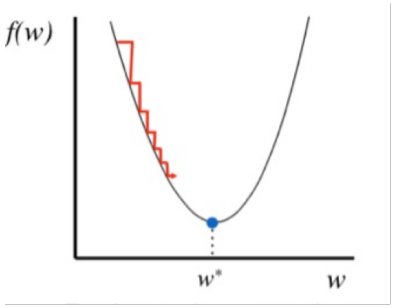
Too high



Too low



Want: good *decay schedule*



Source: Stanford CS231n

Figure source

Learning rate decay

- Decay formulas
  - Exponential: $\eta_t = \eta_0 e^{-kt}$, where η_0 and k are hyperparameters, t is the iteration or epoch number
  - Inverse: $\eta_t = \eta_0/(1+kt)$
  - Inverse sqrt: $\eta_t = \eta_0/\sqrt{t}$
  - Linear: $\eta_t = \eta_0(1 + t/T)$, where *T* is the total number of epochs
  - Cosine: $\eta_t = \frac{1}{2}\eta_0(1 + \cos(t\pi/T))$

Learning rate decay

- Decay formulas
- Most common in practice:
  - **Step decay:** reduce rate by a constant factor every few epochs, e.g., by 0.5 every 5 epochs, 0.1 every 20 epochs
  - **Manual:** watch validation error and reduce learning rate whenever it stops improving
    - "Patience" hyperparameter: number of epochs without improvement before reducing learning rate
- **Warmup:** train with a low learning rate for a first few epochs, or linearly increase learning rate before transitioning to normal decay schedule (Goyal et al., 2018)

A typical phenomenon

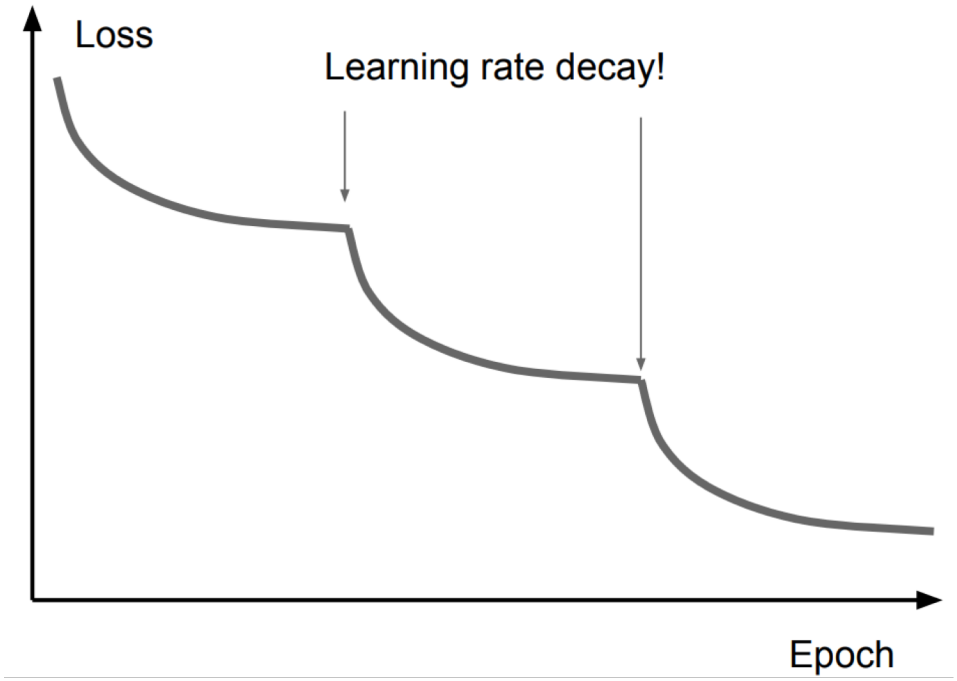
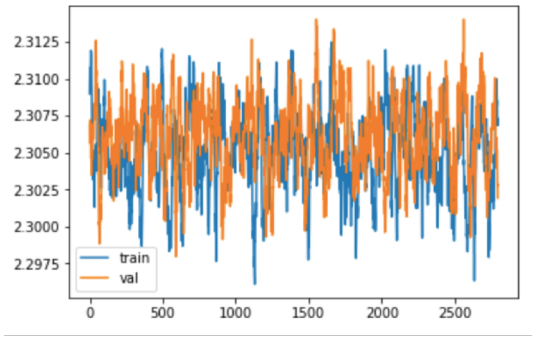


Image source: Stanford CS231n

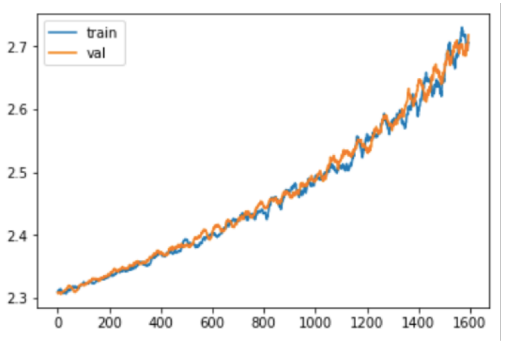
Possible explanation

Image source

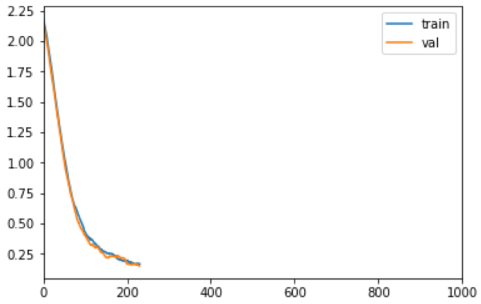
Diagnosing learning curves: Obvious problems



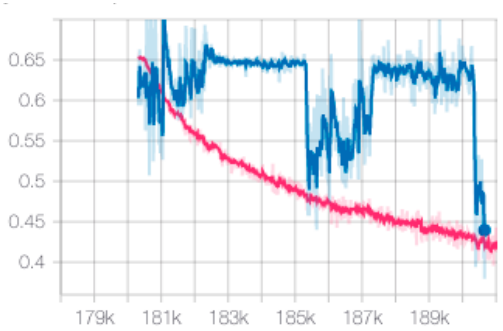
Not training
Bug in update calculation?



Error increasing
Bug in update calculation?



Get NaNs in the loss after a number of iterations:
Numerical instability



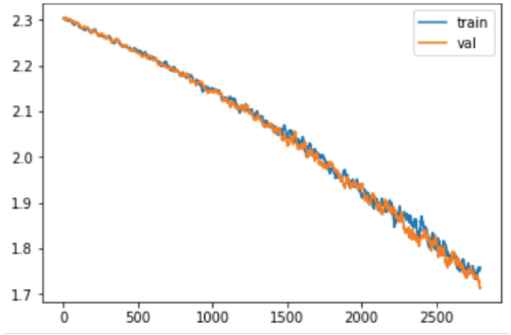
Shuffling off

Shuffling on

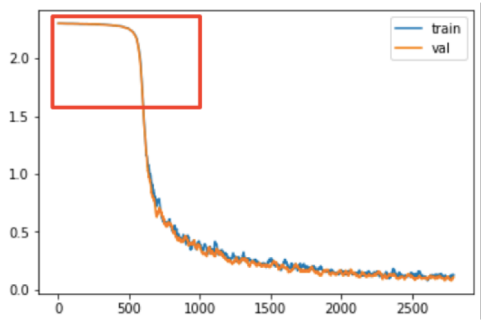
Weird cyclical patterns in loss:
Data not shuffled

Source: Stanford CS231n

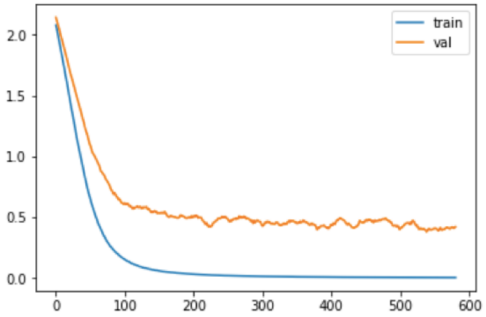
Diagnosing learning curves: Subtler behaviors



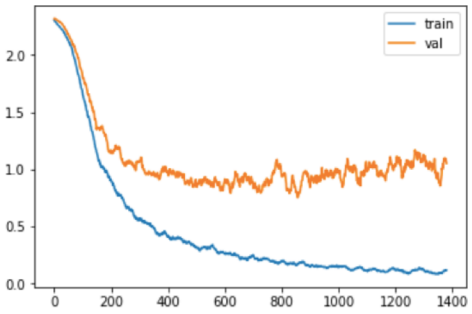
Not converged yet
Keep training, possibly increase learning rate



Slow start
Bad initialization?



Possible overfitting



Definite overfitting

Source: [Stanford CS231n]

When to stop training?

- Monitor validation error to decide when to stop
  - "Patience" hyperparameter: number of epochs without improvement before stopping
  - *Early stopping* can be viewed as a kind of regularization

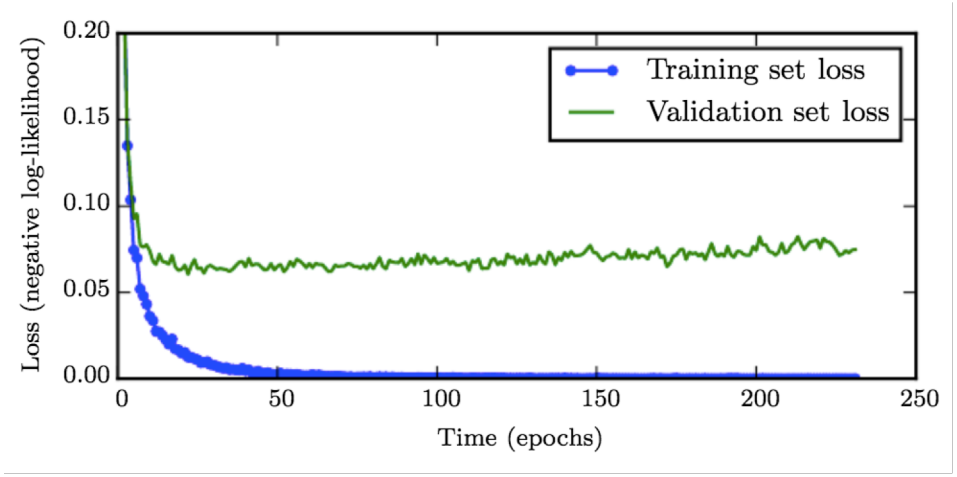


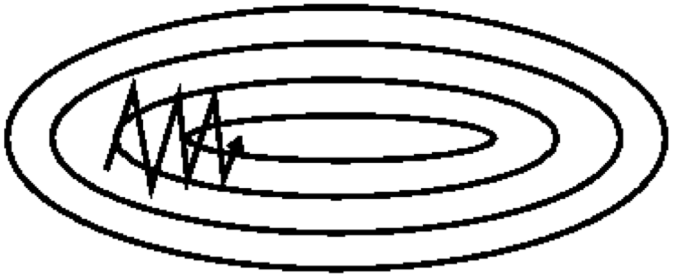
Figure from Deep Learning Book

Advanced optimizers

- SGD with momentum
- RSMProp
- Adam

SGD with momentum

What will SGD do?



[Image source]

SGD with momentum

- Introduce a "momentum" variable m and associated "friction" coefficient β:

$$m \leftarrow \beta m - \eta \nabla L$$
$$w \leftarrow w + m$$

  - Typically start with $\beta = 0.5$, gradually increase over time

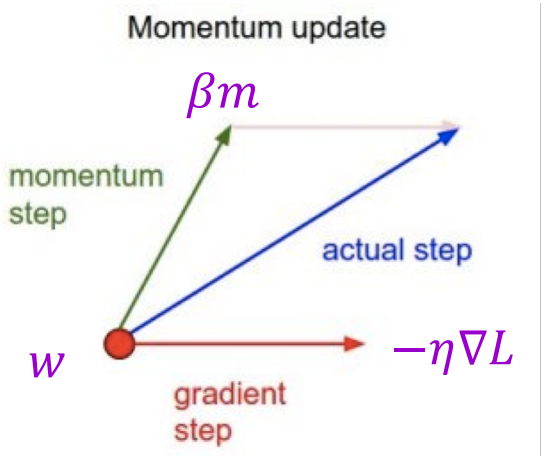


Image source

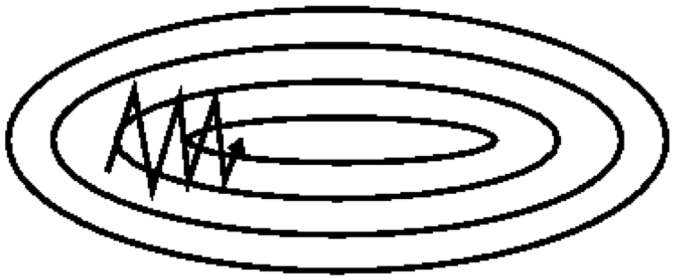
SGD with momentum

- Introduce a "momentum" variable m and associated "friction" coefficient β:

$$m \leftarrow \beta m - \eta \nabla L$$
$$w \leftarrow w + m$$

  - Move faster in directions with consistent gradient
  - Avoid oscillating in directions with large but inconsistent gradients

Standard SGD



SGD with momentum

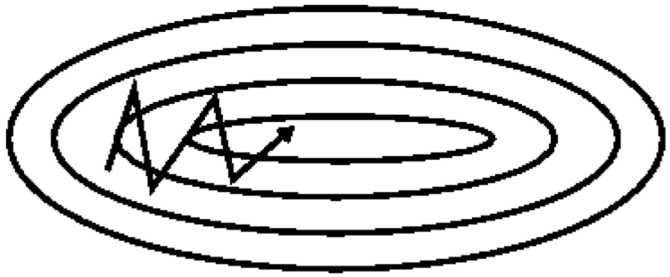


Image source

SGD with momentum

- Introduce a "momentum" variable m and associated "friction" coefficient β:

$$m \leftarrow \beta m - \eta \nabla L$$
$$w \leftarrow w + m$$

- Nesterov momentum: evaluate gradient at "lookahead" position $w + \beta m$

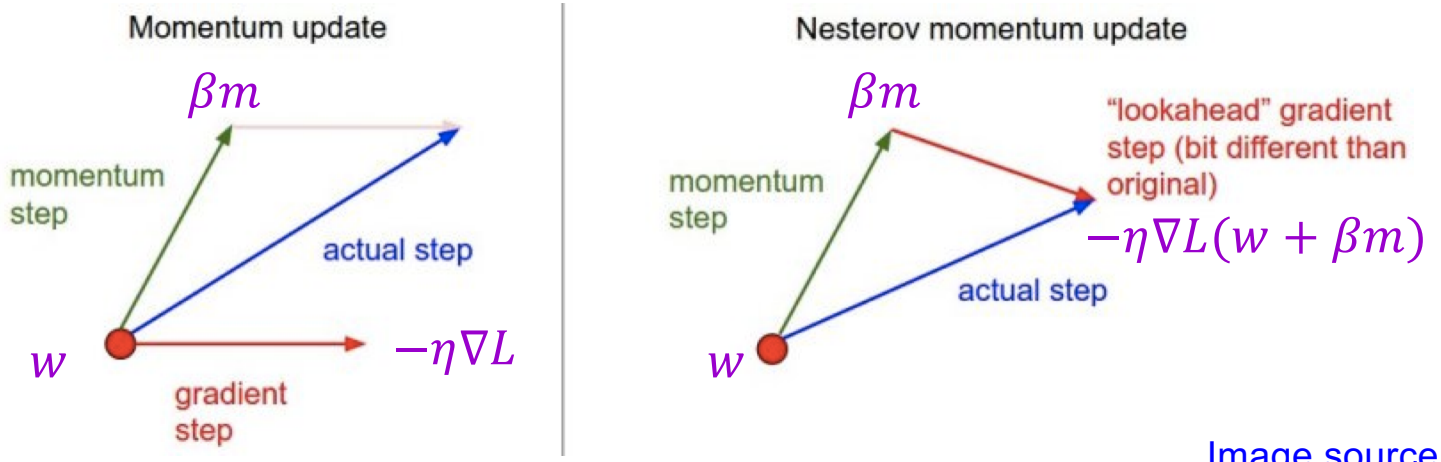


Image source

Adaptive per-parameter learning rates

- Gradients of different layers have different magnitudes
- Want an automatic way to set different learning rates for different parameters

Adagrad

- Keep track of history of gradient magnitudes, scale the learning rate for each parameter based on this history:

$$v_k \leftarrow v_k + \left\| \frac{\partial L}{\partial w_k} \right\|^2$$

$$w_k \leftarrow w_k - \frac{\eta}{\sqrt{v_k} + \epsilon} \frac{\partial L}{\partial w_k}$$

  - Parameters with small gradients get large updates and vice versa
  - Problem: long-ago gradient magnitudes are not “forgotten” so learning rate decays too quickly

J. Duchi, Adaptive subgradient methods for online learning and stochastic optimization, JMLR 2011

RMSProp

- Introduce decay factor β (typically ≥ 0.9) to downweight past history exponentially:

$$v_k \leftarrow \beta v_k + (1-\beta)\left\|\frac{\partial L}{\partial w_k}\right\|^2$$

$$w_k \leftarrow w_k - \frac{\eta}{\sqrt{v_k} + \epsilon}\frac{\partial L}{\partial w_k}$$

http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Adam

- Combine RMSProp with momentum:

$$m \leftarrow \beta_1 m + (1 - \beta_1)\nabla L$$

$$v_k \leftarrow \beta_2 v_k + (1 - \beta_2)\left\|\frac{\partial L}{\partial w_k}\right\|^2$$

$$w_k \leftarrow w_k - \frac{\eta}{\sqrt{v_k} + \epsilon} m_k$$

  - Full algorithm includes *bias correction* to account for m and v starting at 0:

$$\hat{m} = \frac{m}{1-\beta_1^t},\ \hat{v} = \frac{v}{1-\beta_2^t}$$ (t is the timestep)

  - Default parameters from paper: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\eta = 1e-3$, $\epsilon = 1e-8$
    - Reputed to be good starting point for many models

D. Kingma and J. Ba, Adam: A method for stochastic optimization, ICLR 2015

Which optimizer to use in practice?

- Adaptive methods tend to reduce initial training error faster than SGD and are “safer”
  - Andrej Karpathy: *“In the early stages of setting baselines I like to use Adam with a learning rate of 3e-4. In my experience Adam is much more forgiving to hyperparameters, including a bad learning rate. For ConvNets a well-tuned SGD will almost always slightly outperform Adam, but the optimal learning rate region is much more narrow and problem-specific.”*
  - Use Adam at first, then switch to SGD?
- However, some literature reports problems with adaptive methods, such as failing to converge or generalizing poorly (Wilson et al. 2017, Reddi et al. 2018)
  - YMMV!

Outline

- Optimization
  - Mini-batch SGD
  - Learning rate decay
  - Adaptive methods
- Massaging the numbers
  - Data augmentation
  - Data preprocessing
  - Weight initialization
  - Batch normalization
- Regularization
  - Classic regularization: L2 and L1
  - Dropout
  - Label smoothing
- Test time: ensembles, averaging predictions

Data augmentation

- Introduce transformations not adequately sampled in the training data
  - Geometric: flipping, rotation, shearing, multiple crops

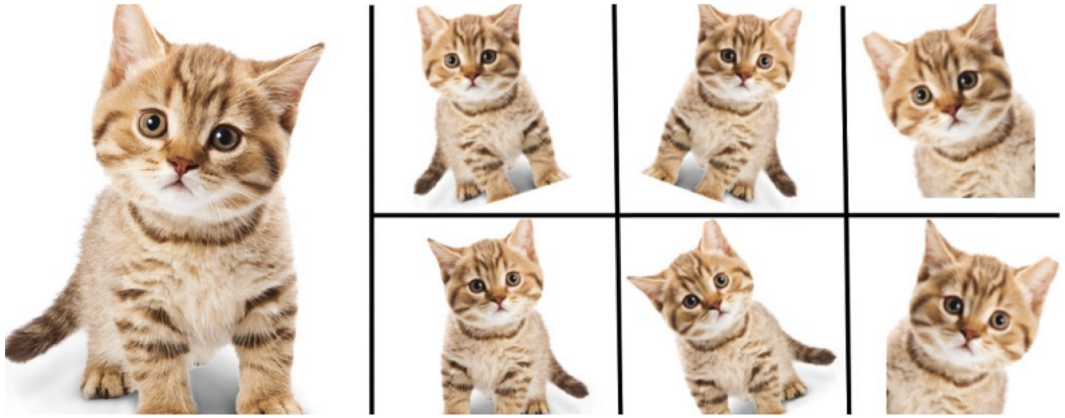


Image source

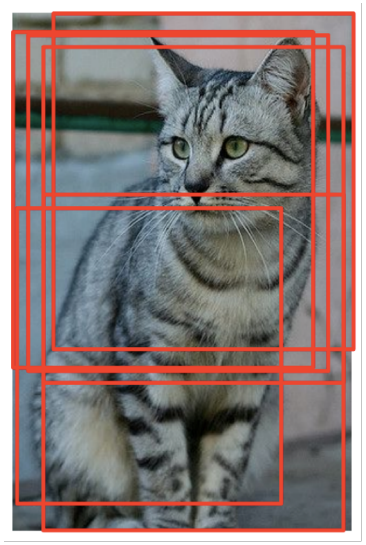


Image source

Data augmentation

- Introduce transformations not adequately sampled in the training data
  - Geometric: flipping, rotation, shearing, multiple crops
  - Photometric: color transformations

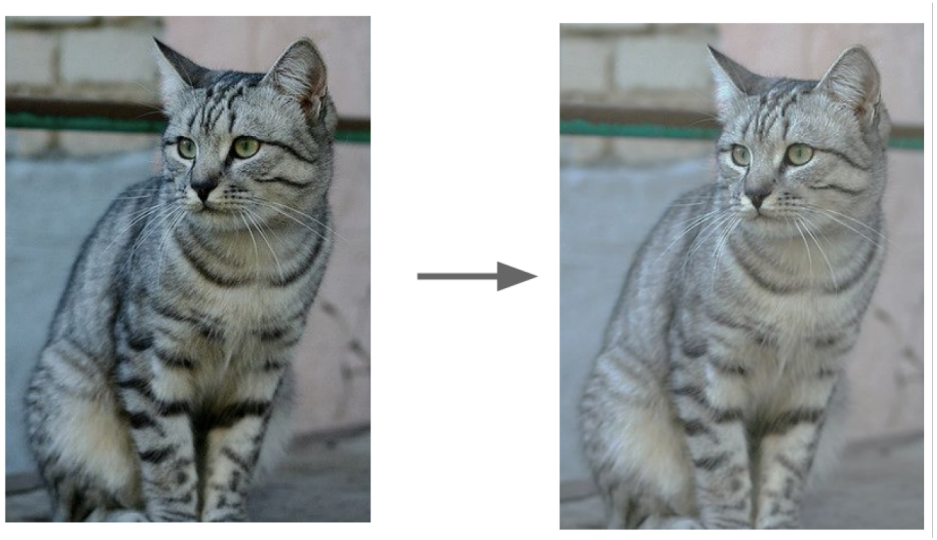


Image source

Data augmentation

- Introduce transformations not adequately sampled in the training data
  - Geometric: flipping, rotation, shearing, multiple crops
  - Photometric: color transformations
  - Other: add noise, compression artifacts, lens distortions, etc.

Image source

Data augmentation

- Introduce transformations not adequately sampled in the training data
- Limited only by your imagination and time/memory constraints!
- Avoid introducing artifacts

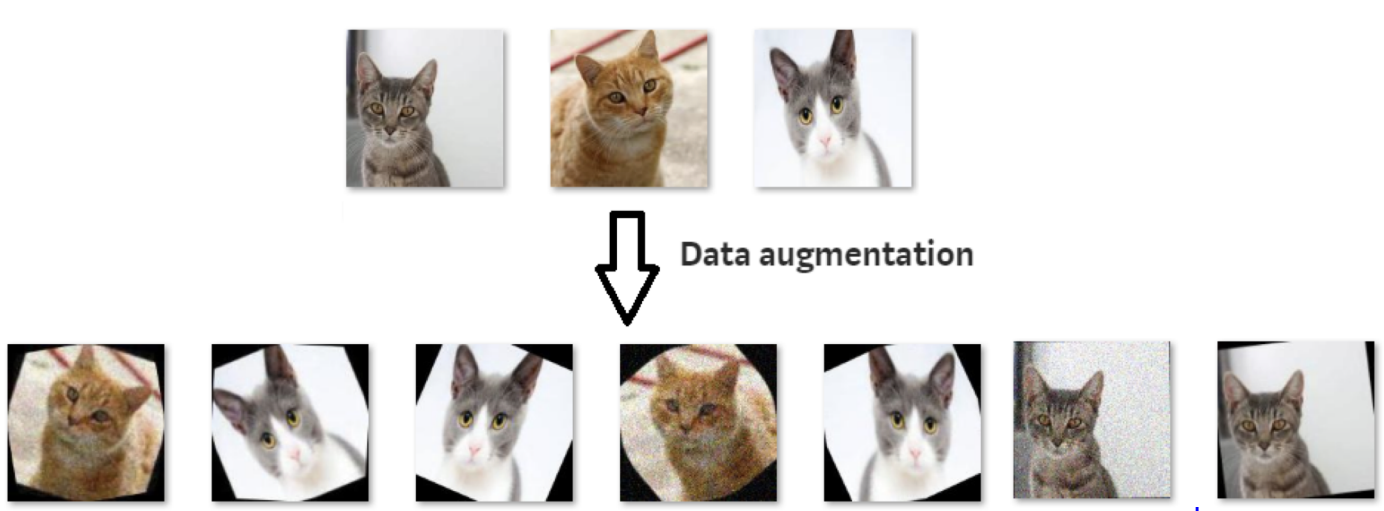


Image source

Data augmentation

- Introduce transformations not adequately sampled in the training data
- Limited only by your imagination and time/memory constraints!
- Avoid introducing artifacts
- Automatic augmentation strategies: AutoAugment, RandAugment

Data preprocessing

- Zero centering
  - Subtract *mean image* – all input images need to have the same resolution
  - Subtract *per-channel means* – images don't need to have the same resolution
- Optional: rescaling – divide each value by (per-pixel or per-channel) standard deviation

- Be sure to apply the same transformation at training and test time!
  - Save training set statistics and apply to test data

Weight initialization

- What's wrong with initializing all weights to the same number (e.g., zero)?

Weight initialization

- Typically: initialize to random values sampled from zero-mean Gaussian: $w \sim \mathcal{N}(0, \sigma^2)$
  - Standard deviation matters!
  - Key idea: avoid reducing or amplifying the variance of layer responses, which would lead to vanishing or exploding gradients
- Common heuristics:
  - Xavier initialization: $\sigma^2 = 1/n_{\text{in}}$ or $\sigma^2 = 2/(n_{\text{in}} + n_{\text{out}})$, where n_{in} and n_{out} are the numbers of inputs and outputs to a layer (Glorot and Bengio, 2010)
  - For ReLU: $\sigma^2 = 2/n_{\text{in}}$ (He et al., 2015)
- Initializing biases: just set them to 0

More details: http://cs231n.github.io/neural-networks-2/#init

Batch normalization

- The authors' intuition

Image source, via Prajit Ramachandran

S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015

Batch normalization

- **Key idea:** shifting and rescaling are differentiable operations, so the network can *learn* how best to normalize the data
- Statistics of activations (outputs) from a given layer across the dataset can be approximated by statistics from a mini-batch

S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015

Batch normalization

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

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

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

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

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

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

Why?

S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015

Batch normalization

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

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

At test time (usually):

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

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

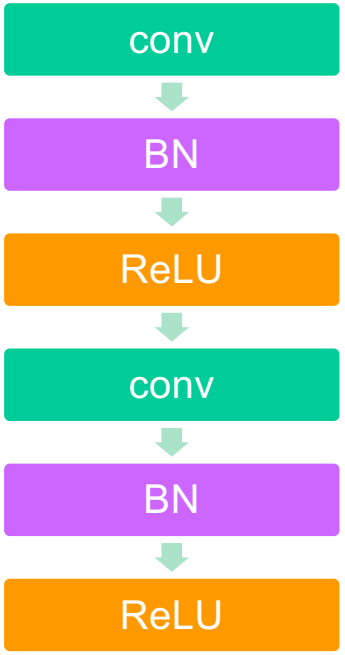
$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$ // normalize

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

S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015

Batch normalization

- Common configuration: insert BN layers right after conv or FC layers, before ReLU nonlinearity (but this is purely empirical)



S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015

Batch normalization

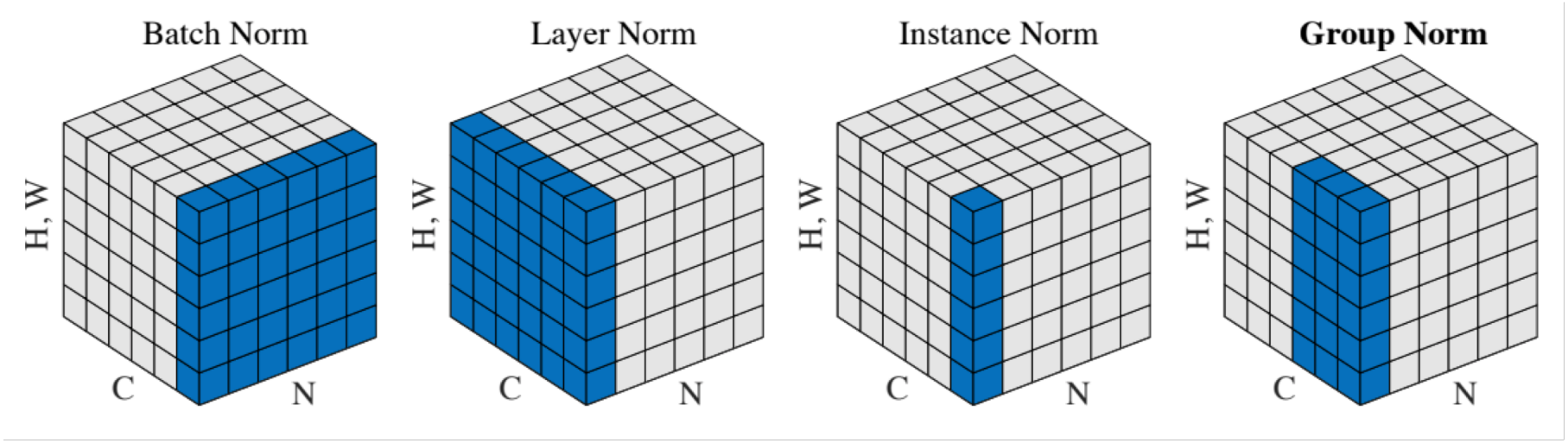
- Benefits
  - Prevents exploding and vanishing gradients
  - Keeps most activations away from saturation regions of non-linearities
  - Accelerates convergence of training
  - Makes training more robust w.r.t. hyperparameter choice, initialization
- Pitfalls
  - Behavior depends on composition of mini-batches, can lead to hard-to-catch bugs if there is a mismatch between training and test regime (example)
  - Doesn’t work well for small mini-batch sizes
  - Cannot be used in recurrent models

Why does BatchNorm *really* work?

- It may have to do not with internal covariate shift (ICS), but with making the optimization problem much smoother (Santurkar et al., 2018)
- *Is ICS even a thing?* (Lipton and Steinhardt, 2018)

Other types of normalization

- Layer normalization (Ba et al., 2016)
- Instance normalization (Ulyanov et al., 2017)
- Group normalization (Wu and He, 2018)
- Weight normalization (Salimans et al., 2016)



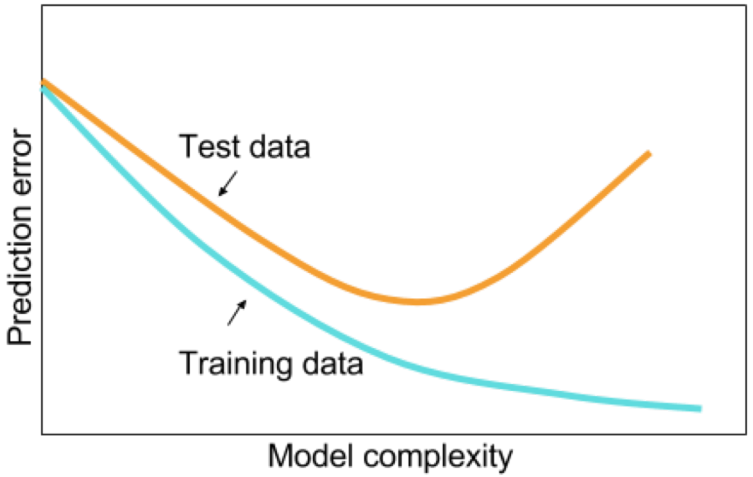
Y. Wu and K. He, Group Normalization, ECCV 2018

Outline

- Optimization
  - Mini-batch SGD
  - Learning rate decay
  - Adaptive methods
- Massaging the numbers
  - Data augmentation
  - Data preprocessing
  - Weight initialization
  - Batch normalization
- Regularization
  - Classic regularization: L2 and L1
  - Dropout
  - Label smoothing
- Test time: ensembles, averaging predictions

Recall: Regularization

- Techniques for controlling the capacity of a neural network to prevent overfitting



Recall: L2 regularization

- Regularized objective:

$$\hat{L}(w) = \frac{\lambda}{2}\|w\|_2^2 + \sum_{i=1}^{n} l(w, x_i, y_i)$$

- Gradient of objective:

$$\nabla\hat{L}(w) = \lambda w + \sum_{i=1}^{n} \nabla l(w, x_i, y_i)$$

- SGD update:

$$w \leftarrow w - \eta\big(\lambda w + \nabla l(w, x_i, y_i)\big)$$
$$w \leftarrow (1 - \eta\lambda)w - \eta\nabla l(w, x_i, y_i)$$

- Interpretation: weight decay

L1 regularization

- Regularized objective:

$$\hat{L}(w) = \lambda \|w\|_1 + \sum_{i=1}^{n} l(w, x_i, y_i)$$

$$= \lambda \sum_{d} |w_d| + \sum_{i=1}^{n} l(w, x_i, y_i)$$

- Gradient: $\nabla \hat{L}(w) = \lambda \operatorname{sgn}(w) + \sum_{i=1}^{n} \nabla l(w, x_i, y_i)$
- SGD update:

$$w \leftarrow w - \eta\lambda \operatorname{sgn}(w) - \eta \nabla l(w, x_i, y_i)$$

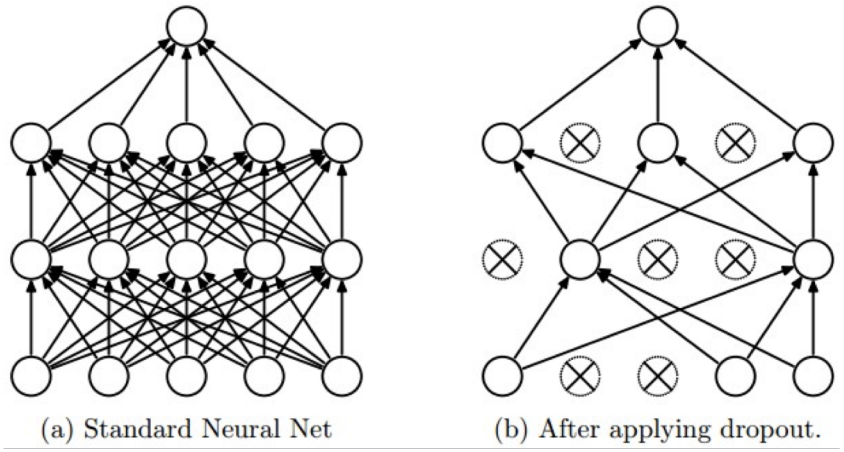
- Interpretation: encouraging sparsity

Other types of regularization

- Adding noise to the inputs
  - Recall motivation of max margin criterion
  - In simple scenario (linear model, quadratic loss, Gaussian noise), this is equivalent to weight decay
  - Data augmentation is a more general form of this
- Adding noise to the weights

Dropout

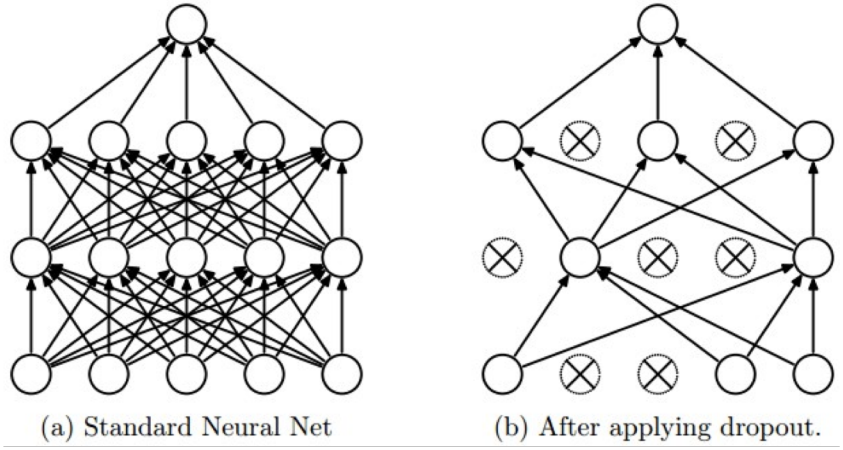
- At training time, in each forward pass, turn off some neurons with probability *p*
- At test time, to have deterministic behavior, multiply output of neuron by *p*



N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov.
Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR 2014

Dropout

- Intuitions
  - Prevent “co-adaptation” of units, increase robustness to noise
  - Train *implicit ensemble*



N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov.
Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR 2014

Current status of dropout

- Against
  - Slows down convergence
  - Made redundant by batch normalization or possibly even clashes with it
  - Unnecessary for larger datasets or with sufficient data augmentation
- In favor
  - Can still help in certain situations: e.g., used in Wide Residual Networks

Label smoothing

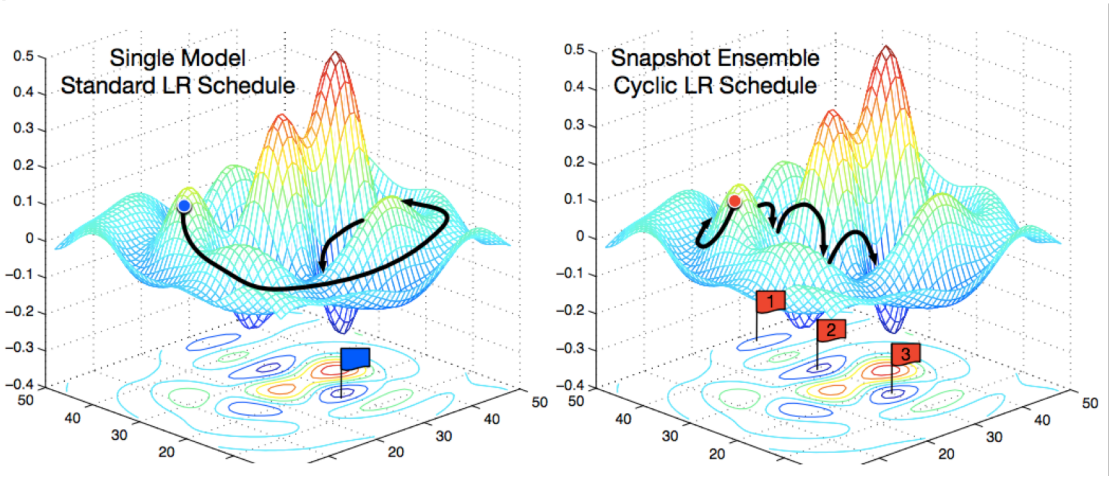
- **Idea:** avoid overly confident predictions, account for label noise
- When using softmax loss, replace hard 1 and 0 prediction targets with "soft" targets of
  $1 - \epsilon$ and $\frac{\epsilon}{C-1}$
- Used in Inception-v2 architecture

Outline

- Optimization
  - Mini-batch SGD
  - Learning rate decay
  - Adaptive methods
- Massaging the numbers
  - Data augmentation
  - Data preprocessing
  - Weight initialization
  - Batch normalization
- Regularization
  - Classic regularization: L2 and L1
  - Dropout
  - Label smoothing
- Test time: ensembles, averaging predictions

Test time

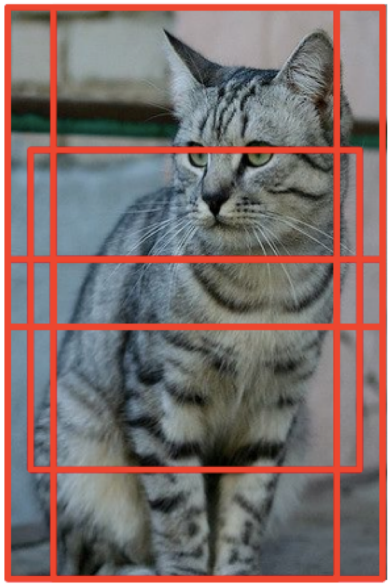
- **Ensembles:** train multiple independent models, then average their predicted label distributions
  - Gives 1-2% improvement in most cases
  - Can take multiple snapshots of models obtained during training, especially if you *cycle* the learning rate (increase to jump out of local minima)



G. Huang et al., Snapshot ensembles: Train 1, get M for free, ICLR 2017

Test time

- Average predictions across multiple crops of test image
  - There is a more elegant way to do this with *fully convolutional networks* (FCNs)



Attempt at a conclusion

- Training neural networks is still a black art
- Process requires close “babysitting”
- For many techniques, the reasons why, when, and whether they work are in active dispute – read everything but don’t trust anything
- It all comes down to (principled) trial and error
- Further reading: A. Karpathy, A recipe for training neural networks