Neural Network training

- 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

Slides from L. Lazebnik

Mini-batch SGD

- Iterate over epochs
  - Iterate over dataset mini-batches $(x_1, y_1), \ldots, (x_b, y_b)$
    - Compute gradient of the mini-batch loss:

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

    - 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

SGD and mini-batch size

- Larger mini-batches: more expensive and less frequent updates, lower gradient variance, more parallelizable
- In the literature, SGD with larger batches is generally reported to generalize more poorly (e.g., Keskar et al., 2016)
  - But can be made to work by using larger learning rates with larger mini-batches (Goyal et al., 2017)

Learning rate decay

- **Exponential decay**: $\eta = \eta_0 e^{-kt}$, where η_0 and k are hyperparameters, t is the iteration or epoch number
- **$1/t$ decay**: $\eta = \eta_0/(1 + kt)$
- **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

Diagnosing learning rates

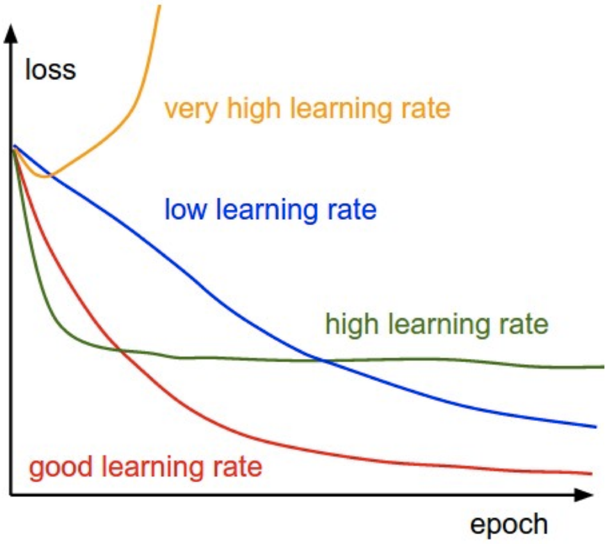
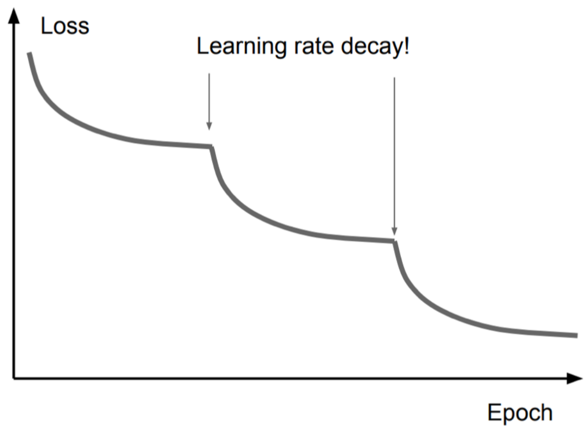


Image source: [Stanford CS231n]

A typical phenomenon

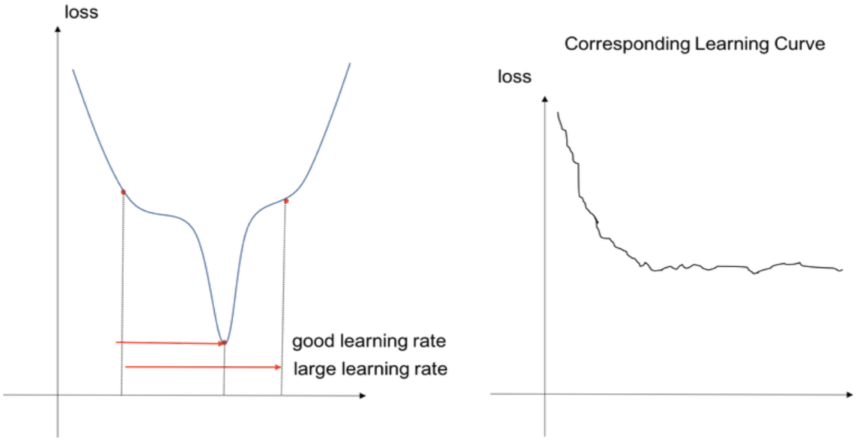


- Why does the learning curve look like this?

Image source: [Stanford CS231n]

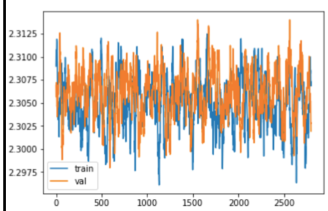
A typical phenomenon

Possible explanation

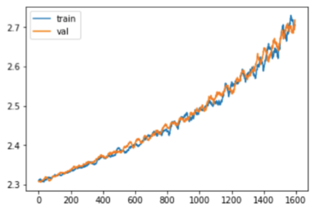


[Image source](#)

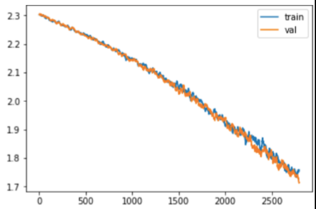
Debugging learning curves



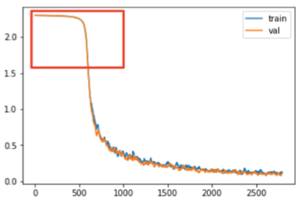
Not training
Bug in update calculation?



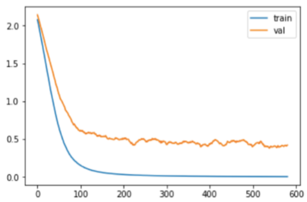
Error increasing
Bug in update calculation?



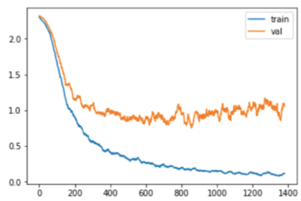
Error decreasing
Not converged yet



Slow start
Suboptimal initialization?



Possible overfitting



Definite overfitting

Image source: [Stanford CS231n](#)

Early stopping

- Idea: do not train a network to achieve too low training error
- Monitor validation error to decide when to stop

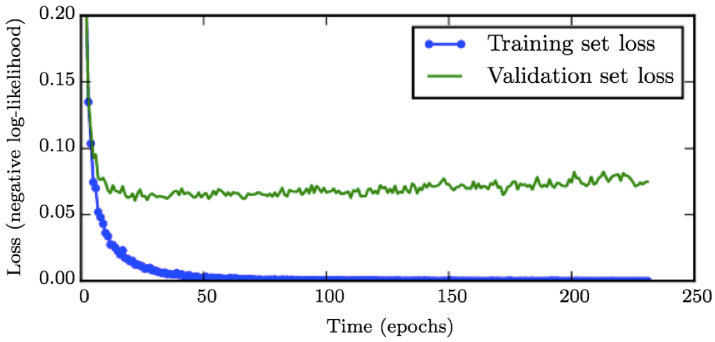


Figure from [Deep Learning Book]

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

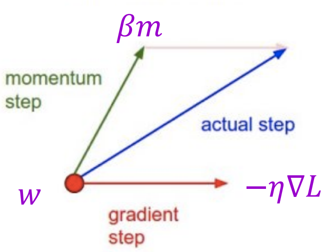


Image source

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

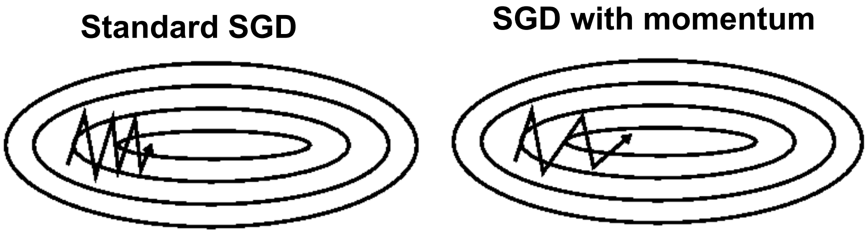


Image source

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 $w + \beta m$

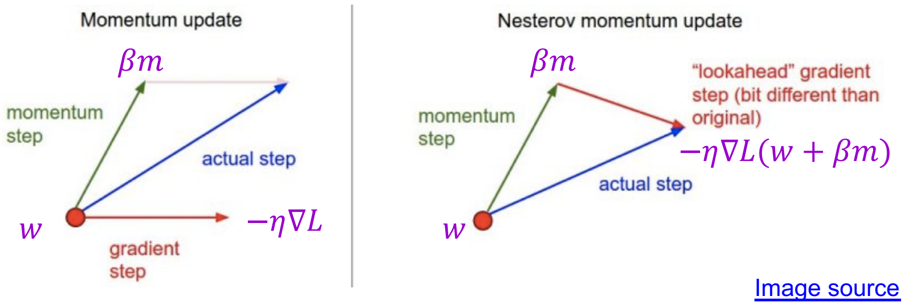


Image source

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
  - 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:

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

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

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

  - Default parameters from paper: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e - 8$
  - Full algorithm includes *bias correction* term 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)

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
  - Adam with default parameters is a popular choice, SGD+momentum may work better but requires more tuning
- However, adaptive methods may quickly plateau on the validation set or generalize more poorly
  - Use Adam first, then switch to SGD?
  - Or just stick with plain old SGD? (Wilson et al., 2017)
- All methods require careful tuning and learning rate control

Massaging the numbers

Data augmentation

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

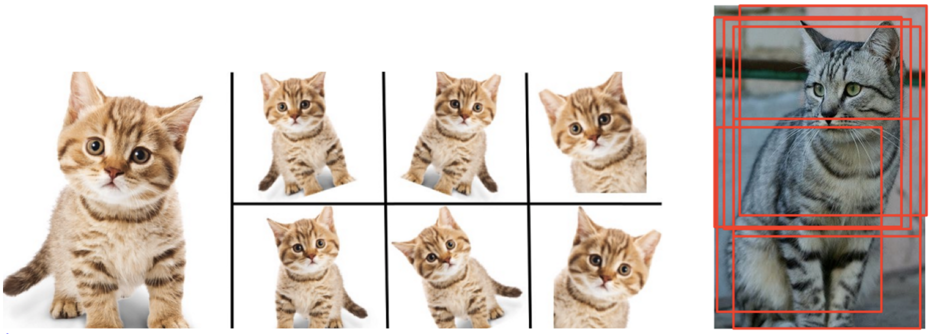


Image source

Image source

Data augmentation

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

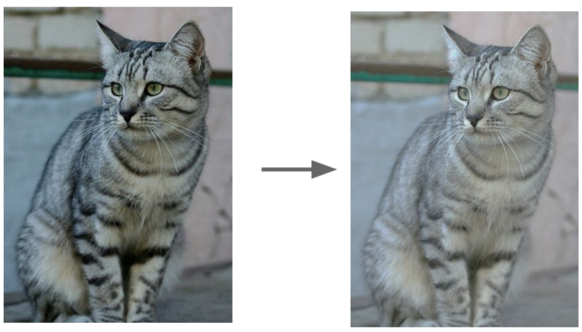


Image source

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 obvious artifacts

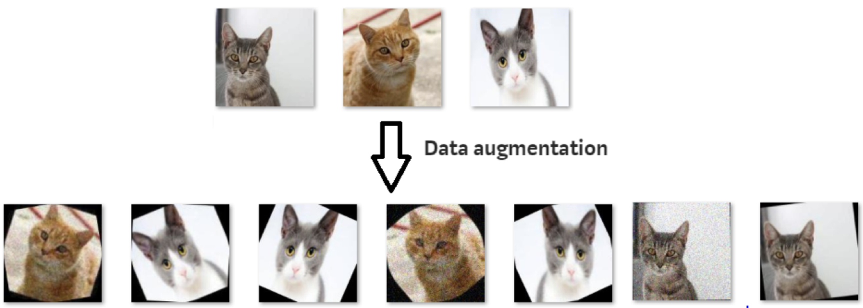


Image source

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:
  - $\sigma = 1/\sqrt{n_{\text{in}}}$, where n_{in} is the number of inputs to a layer
  - $\sigma = 2/\sqrt{n_{\text{in}} + n_{\text{out}}}$ (Glorot and Bengio, 2010)
  - $\sigma = \sqrt{2/n_{\text{in}}}$ for ReLU (He et al., 2015)
- Initializing biases: just set them to 0

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

Review: 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:

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

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

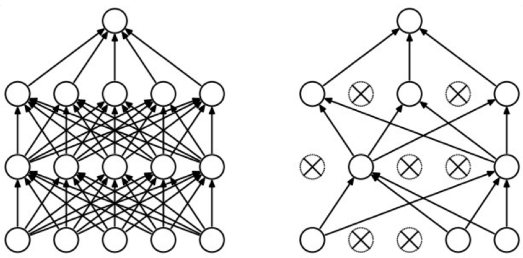
- Gradient: $\nabla \hat{L}(w) = \lambda \operatorname{sgn}(w) + \sum_{i=1}^{n} \nabla l(w, x_i, y_i)$
- SGD update:

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

- Interpretation: encouraging sparsity

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

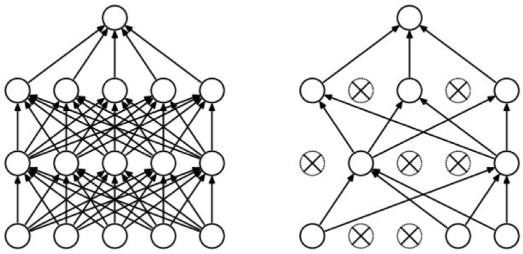


(a) Standard Neural Net (b) After applying dropout.

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*



(a) Standard Neural Net (b) After applying dropout.

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

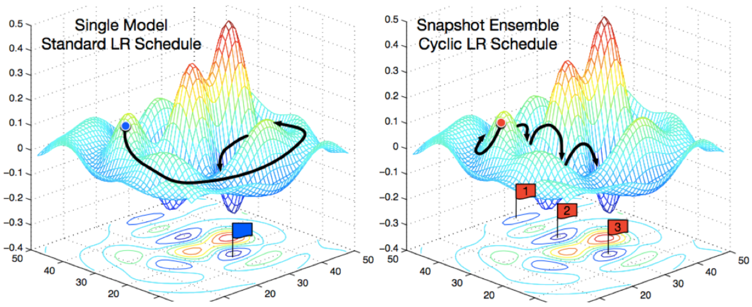
- Against
  - Slows down convergence
  - Made redundant by batch normalization or possibly even clashes with it
  - Unnecessary for larger datasets or with sufficient data augmentation
- In favor
  - Can still help in certain scenarios: e.g., used in Wide Residual Networks

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

Test time

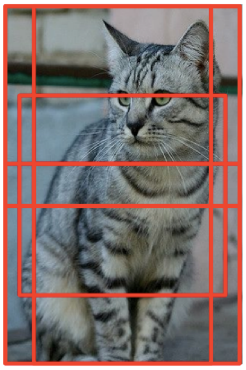
- **Ensembles:** train multiple independent models, then average their predicted label distributions
  - Gives 1-2% improvement in most cases
  - Can take multiple snapshots of models obtained during training, especially if you cycle the learning rate



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