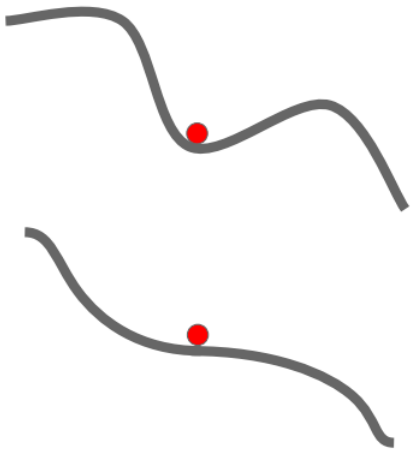
# CS231N 7강
# Training Neural Networks part2

# Optimization

1) **Stochastic Gradient Descent(SGD)** : 손실함수의 값이 최소가 되는 방향으로 가중치 업데이트

   *단점- local minima와 saddle point 문제



Zero gradient, gradient descent gets stuck

Saddle points much more common in high dimension

# Optimization

Local minima, saddle 문제 해결->**SGD+ Momentum**
- 가중치 업데이트 시 속도 vx 추가

## SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

## SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

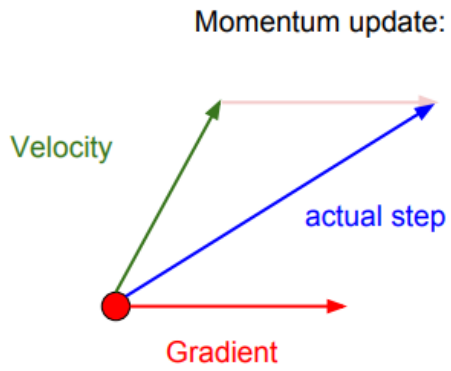
$$x_{t+1} = x_t - \alpha v_{t+1}$$

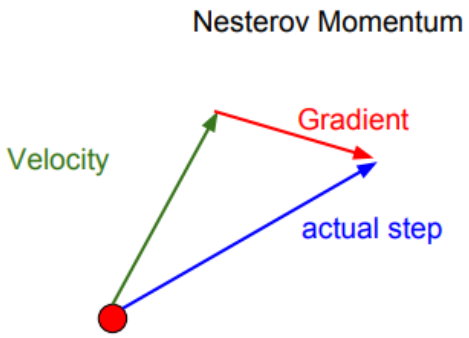
```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x += learning_rate * vx
```

# Optimization

**Nestero Momentum**

- 직전 지점의 velocity 벡터의 종착점 예상해서 그 지점에서의 gradient 벡터 합하여 다음 지점 구함




$$
\begin{aligned}
v_{t+1} &= \rho v_t - \alpha \nabla f(\tilde{x}_t) \\
\tilde{x}_{t+1} &= \tilde{x}_t - \rho v_t + (1+\rho) v_{t+1} \\
&= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)
\end{aligned}
$$

# Optimization

**AdaGrad**

- dx의 제곱에 루트를 씌운 값을 x값 업데이트시 나눠줌

*단점: 가중치가 0이 되어 학습이 종료될 수 있음

-> **RMSProp**

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

# Optimization

**Adam**

- RMSProp +Momentum 방식
- 대부분 많이 쓰임
- Bias correction term 추가

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

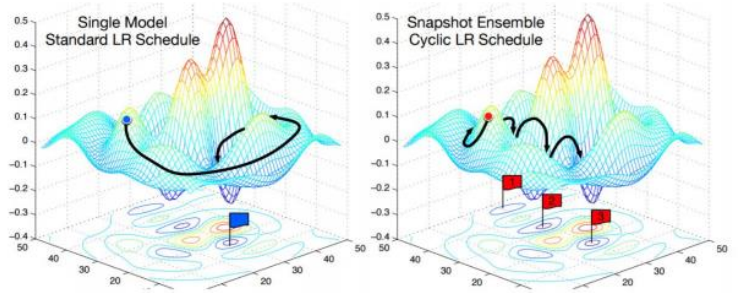
Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

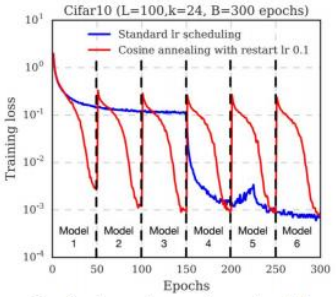
# Optimization

**Model Ensembles**

- Train/test error 격차 줄이는 방법으로 n개의 모델 독립적으로 학습 후 n개의 모델 결과의 평균을 사용함
- 학습 도중 중간 모델을 snapshot하여 앙상블로 사용, test에서 snapshot의 예측값을 평균내서 사용하는 방법 있음
- Poyak averaging

Instead of training independent models, use multiple snapshots of a single model during training!



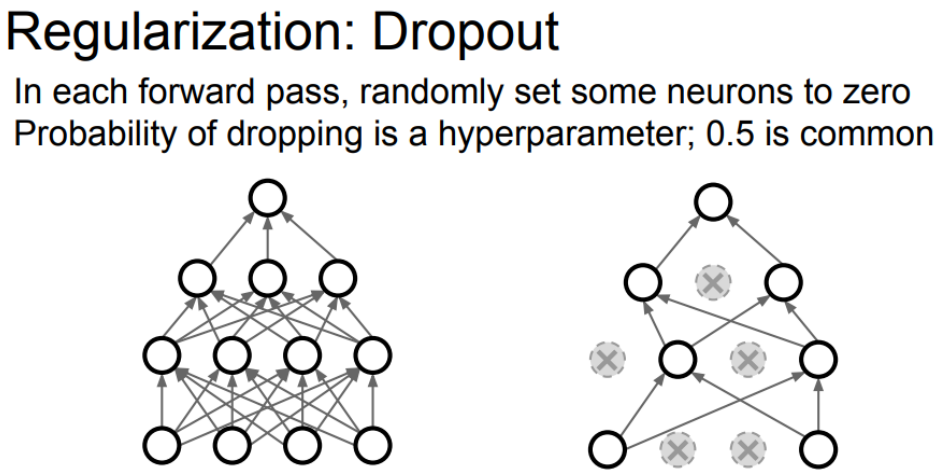




Loshchilov and Hutter, "SGDR: Stochastic gradient descent with restarts", arXiv 2016
Huang et al, "Snapshot ensembles: train 1, get M for free", ICLR 2017
Figures copyright Yixuan Li and Geoff Pleiss, 2017. Reproduced with permission.

Cyclic learning rate schedules can make this work even better!

# Regularization

- 단일 모델로 성능 향상시키는 방법
- NN에서 자주 사용하는것은 dropout(Fc,conv layer)
- Dropout: forward pass에서 임의로 일부 뉴런 0으로 함
- Test time에 dropout probability 네트워크 출력에 곱해줌
- Inverted dropout

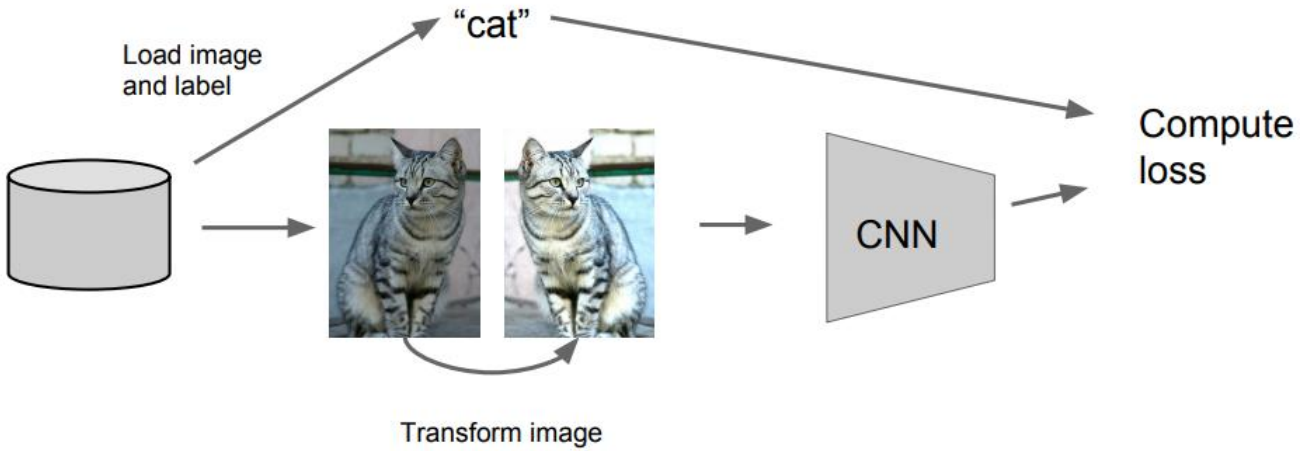


# Regularization

**Data Augmentation**

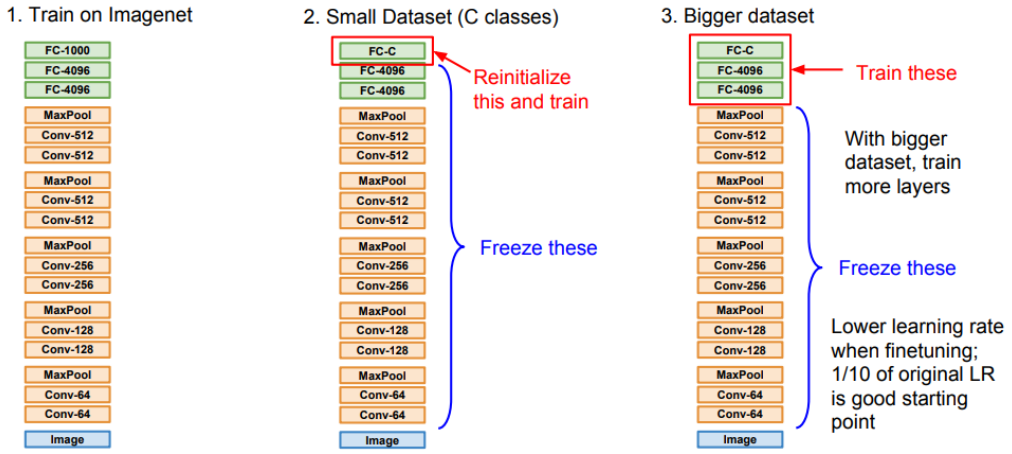
-Transform

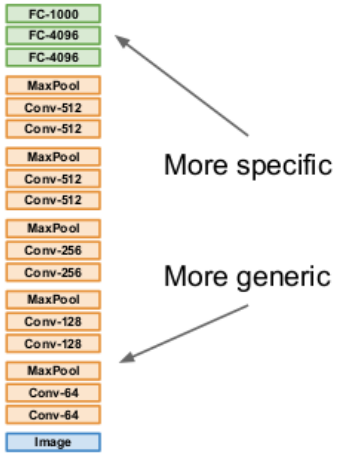
- Horizontal Flips
- Crops and Scales
- Color Jitter
- Rotation
- Stretching
- Shearing
- Lens distortions…



# Transfer Learning

- 충분한 데이터가 없을 때 적은 데이터로도 모델 빠르게 학습
- 많은 양의 데이터로 학습시킨 알고리즘 가져오고 마지막 FC layer만 초기화 시켜 작은 데이터 학습시킴




| | 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 |