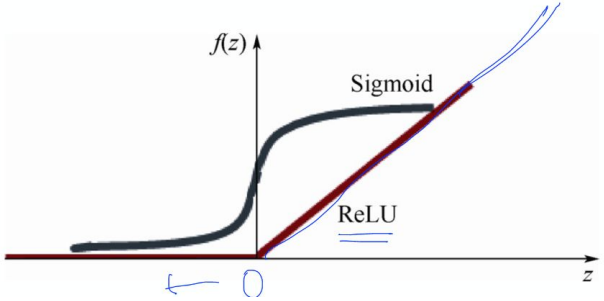
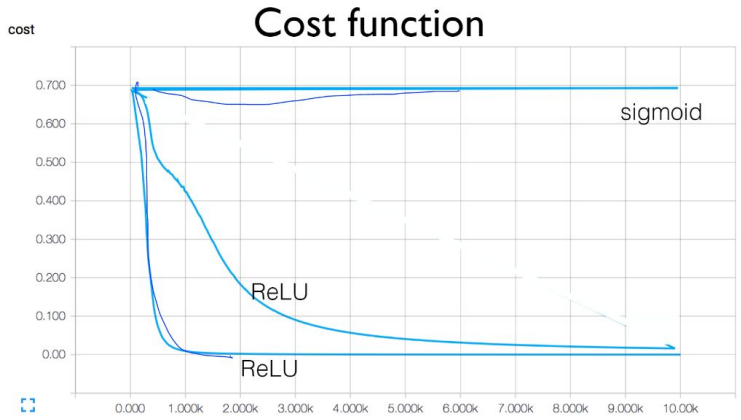
머신러닝 스터디 6th week 보조 자료

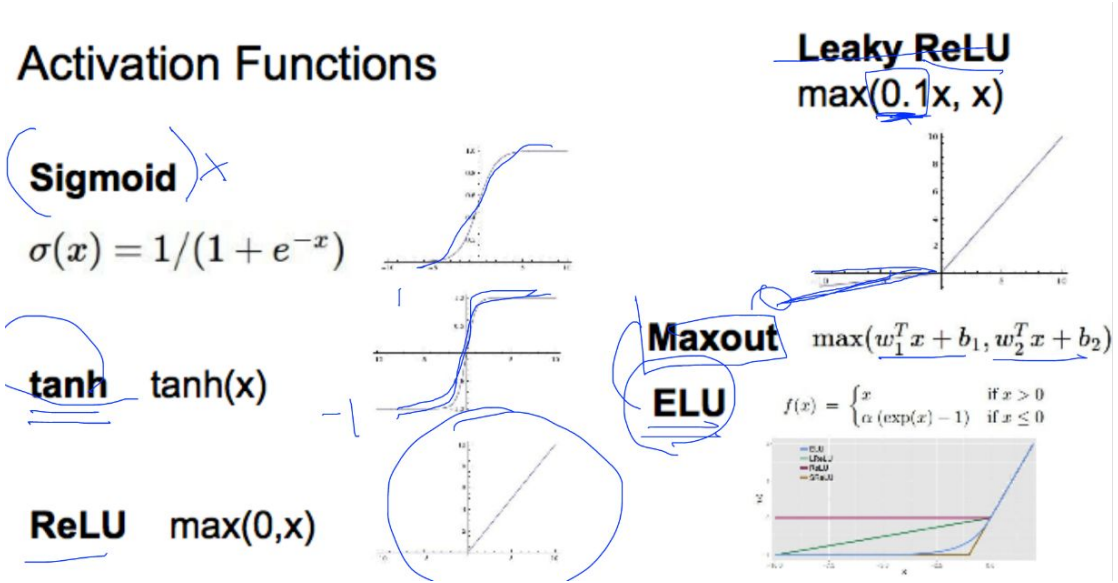
20180223 김성헌

summary



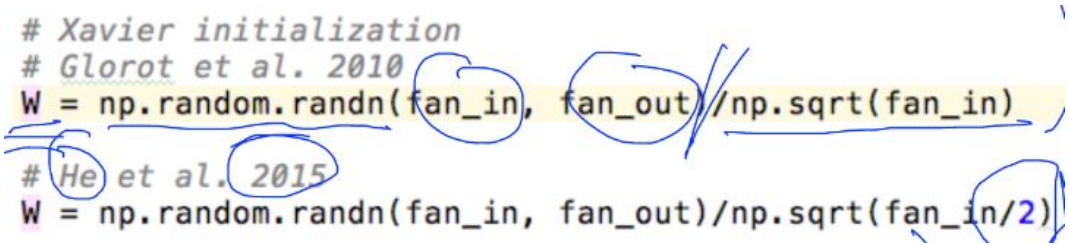
- ch.10-1 ReLU: Better non-linearity
  - 더 효과적인 Activation function 을 사용하자
  - ReLU(Rectified Linear Unit)
    - deep network에서는 Sigmoid 보다 더 좋은 성능을 보임





summary

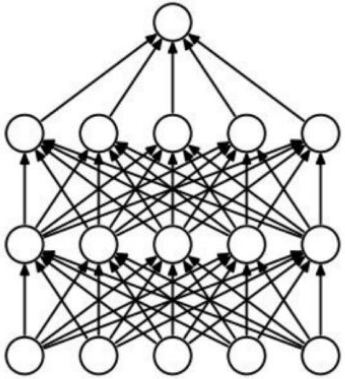
- ch.10-2 Initialize weights in a smart way
  - weight의 초기값을 잘 설정해야 학습이 잘 된다.
  - 0으로 설정하지 마라
  - RBM(Restricted Boltzmann Machine)(2006)
    - 효과적인 초기값 설정 알고리즘
  - RBM과 같은 효과를 갖는 간단한 알고리즘
    - Xavier initialization(2010)
    - He's initialization(2015)



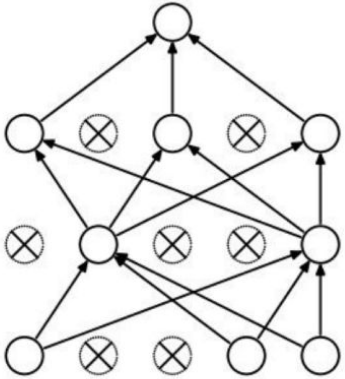
```
# Xavier initialization
# Glorot et al. 2010
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)

# He et al. 2015
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

summary

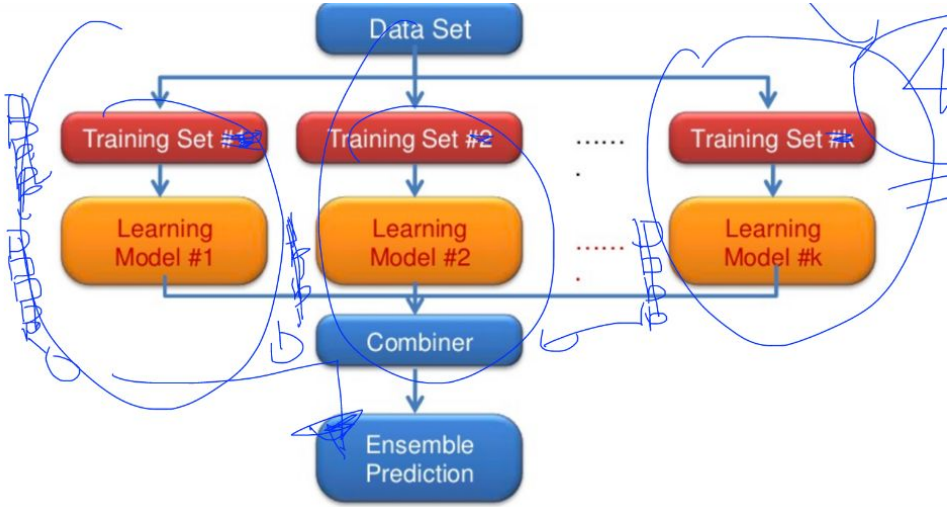
- ch.10-3 NN dropout and model ensemble
  - dropout
    - 신경망이 복잡해 지면 가중치감소 방법으로는 overfitting을 방지하기 어렵다.
    - 복잡한 신경망에서는 overfitting을 방지하기위해 dropout을 사용하자
  - ensemble



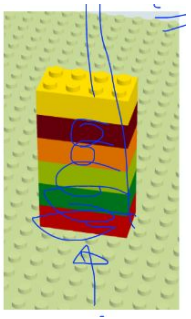
(a) Standard Neural Net



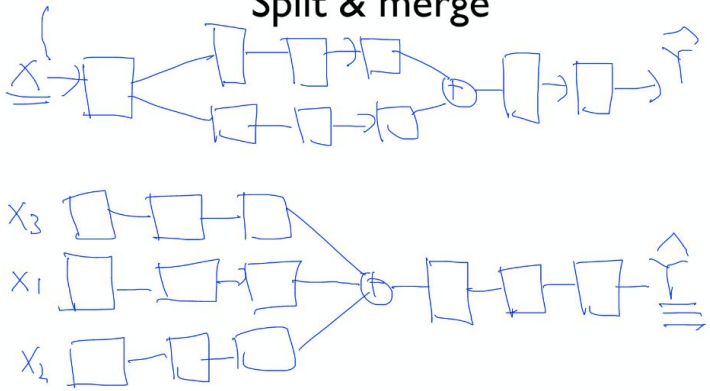
(b) After applying dropout.



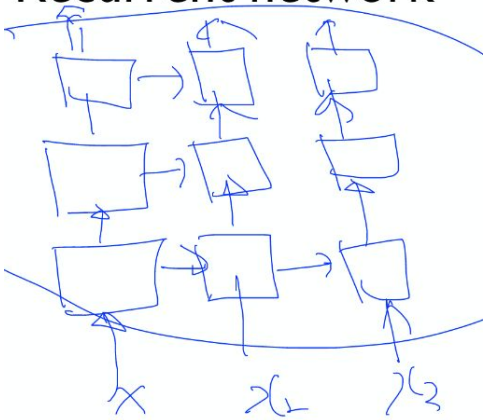
summary

- ch.10-4 NN LEGO Play
  - 

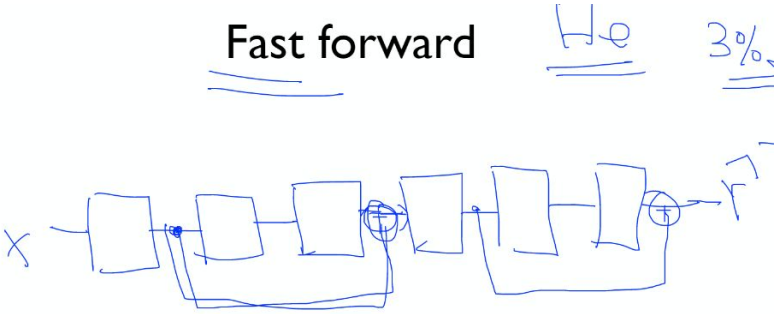
Split & merge



Recurrent network



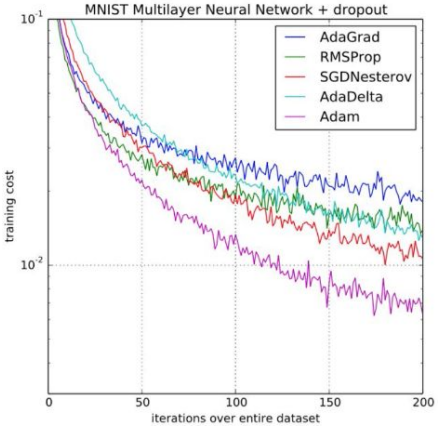
Fast forward



summary

- ch.10 Lab
  - Softmax VS Neural Nets for MNIST, 90% and 94.5%
  - Xavier initialization: 97.8%
  - Deep Neural Nets with Dropout: 98%
  - Adam and other optimizers
  - Exercise: Batch Normalization

[Kingma et al. 2015]



Activation function 보충

- http://nmhkahn.github.io/NN
- 결론
  - 가장 먼저 ReLU를 사용하자. 변형된 버전인 Leaky ReLU, ELU, Maxout들이 있지만 가장 많이 사용되는 activation 함수는 ReLU이다.
  - Leaky ReLU / Maxout / ELU도 시도해보자.
  - tanh도 사용할 순 있지만 큰 기대는 하지 않는게 좋다.
  - sigmoid는 절대 사용하지 말자 (RNN에서는 사용하긴 하지만 다른 이유가 있기 때문이다).

Optimization 보충

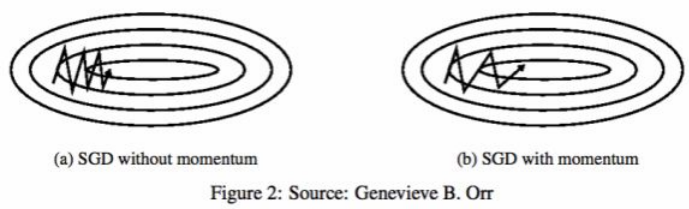


Figure 2: Source: Genevieve B. Orr

- SGD
  - 방향에 따라 기울기 차이가 심할 때 비효율적인 움직임을 보임
  - $$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$
- Momentum
  - SGD의 단점을 개선하기 위해 “관성”을 넣어 줌

$$\mathbf{v} \leftarrow \alpha\mathbf{v} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

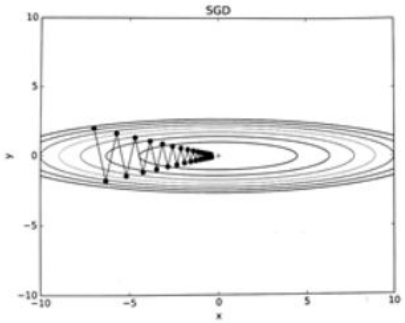
$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}$$

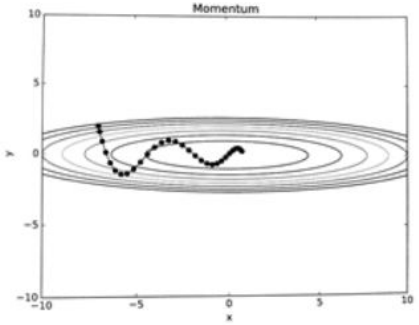
- AdaGrad
  - 학습률을 처음에는 크게 & 점점 작게

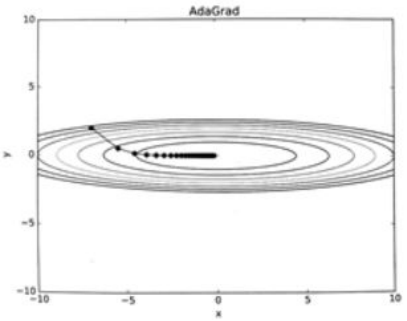
$$\mathbf{h} \leftarrow \mathbf{h} + \frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}}$$

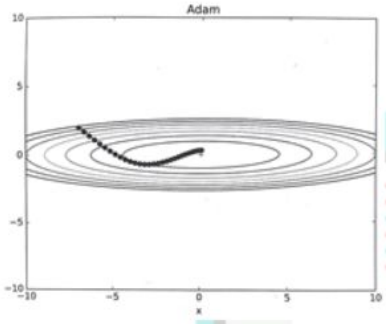
$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{1}{\sqrt{\mathbf{h}}} \frac{\partial L}{\partial \mathbf{W}}$$

- Adam
  - 개념: Momentum + AdaGrad



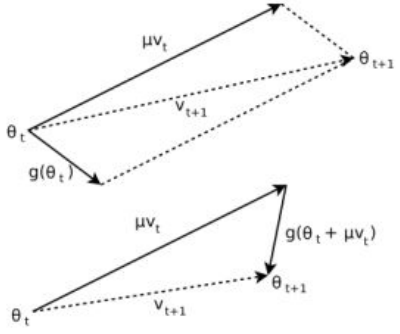






Optimization 보충

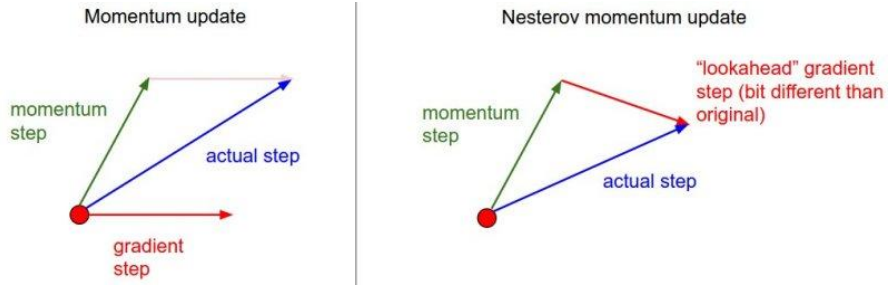
- RMSProp
  - AdaGrad는 무한히 반복되면 갱신량이 0이 되는 단점
  - 먼 과거의 기울기는 잊고 새로운 기울기 정보를 크게 반영
- Nesterov momentum
  - 참고: https://tensorflow.blog/2017/03/22/momentum-nesterov-momentum/#comments
- 참고 사이트
  - http://aikorea.org/cs231n/neural-networks-3/#sgd

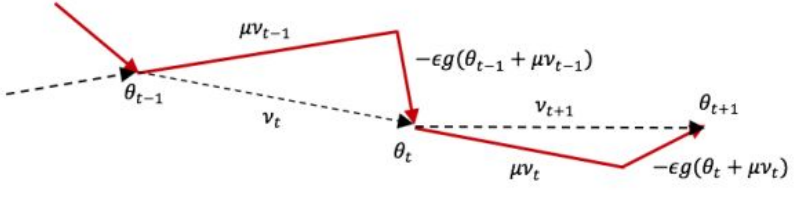


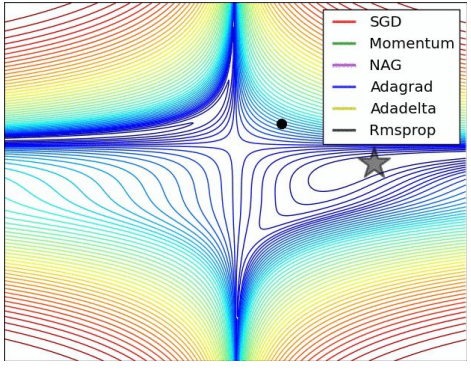
$$v_{t+1} = \mu v_t - \epsilon g(\theta_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

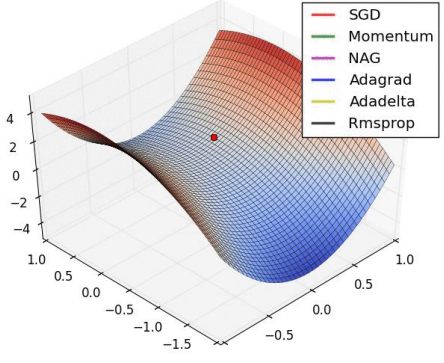
$$v_{t+1} = \mu v_t - \epsilon g(\theta_t + \mu v_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

Figure 1. **(Top)** Classical Momentum **(Bottom)** Nesterov Accelerated Gradient



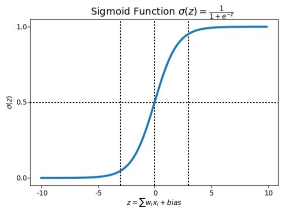


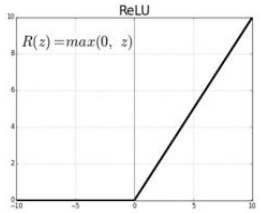




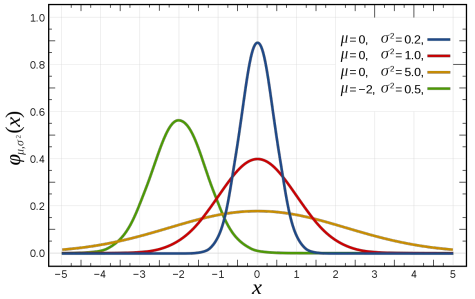
가중치 초기값 보충

$$a = sigmoid(l) = \frac{1}{1+e^{-l}}, \quad \frac{\partial a}{\partial l} = a(1-a)$$





$$Z = \frac{X-\mu}{\sigma}$$



- 실험해 보자
  - 5 layer, 100뉴런 network 사용
  - sigmoid , ReLU 사용
  - 가중치 초기화
    - 표준편차 1인 정규분포
    - 표준편차 0.01인 정규분포
    - Xavier (표준편차 sqrt(1/n) 인 정규분포, n은 앞계층노드 개수)
    - He (표준편차 sqrt(2/n) 인 정규분포)
  - 활성화값 분포를 분석

그림 6-10 가중치를 표준편차가 1인 정규분포로 초기화할 때의 각 층의 활성화값 분포

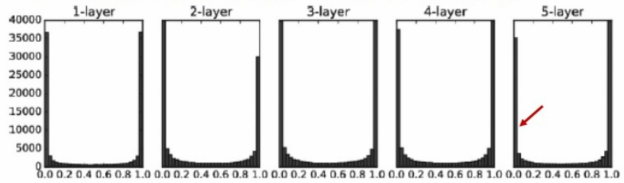


그림 6-14 활성화 함수로 ReLU를 사용한 경우의 가중치 초깃값에 따른 활성화값 분포 변화

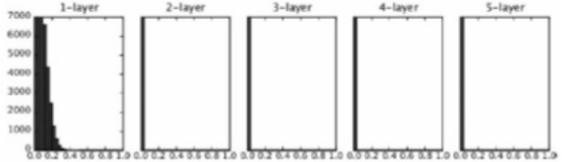
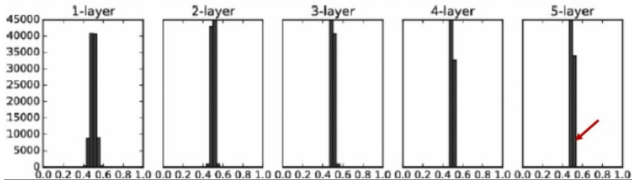


그림 6-11 가중치를 표준편차가 0.01인 정규분포로 초기화할 때의 각 층의 활성화값 분포



표준편차가 0.01인 정규분포를 가중치 초깃값으로 사용한 경우

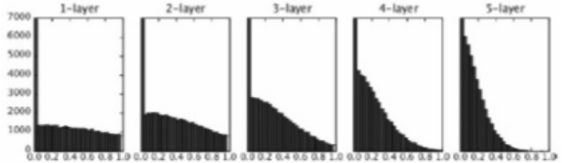
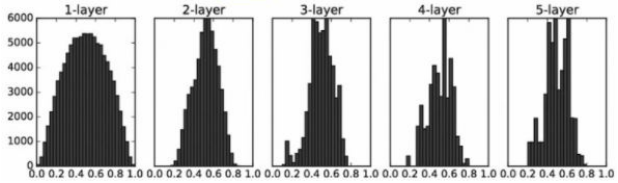
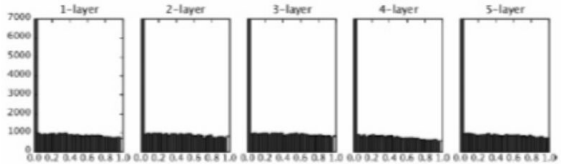


그림 6-13 가중치의 초깃값으로 'Xavier 초깃값'을 이용할 때의 각 층의 활성화값 분포

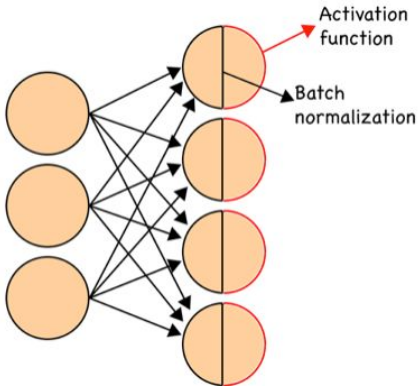


Xavier 초깃값을 사용한 경우



- 결론
  - sigmoid, tanh 등의 S자 모양 곡선에서는 Xavier를 사용하자
  - ReLU를 사용할 때는 He를 사용하자
- 추가로
  - 바이어스는 0으로 초기화 하자

Batch Normalization



$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

- 필요성
  - 뉴럴넷 학습에서는 필수화 되어가고 있음
  - 학습이 빨리 진행됨
  - 초깃값에 크게 의존하지 않음
  - 오버피팅을 억제함 (dropout등의 필요성 감소)
- Batch Normalization 설명 및 구현
  - https://shuuki4.wordpress.com/2016/01/13/batch-normalization-%EC%84%A4%EB%AA%85-%EB%B0%8F-%EA%B5%AC%ED%98%84/
- 기술을 사용하는 입장에서 정리된 사이트
  - http://eyeofneedle.tistory.com/25
  -
- 텐서플로우 가이드: 배치 노멀라이제이션
  - http://openresearch.ai/t/topic/80
  -

적절한 하이퍼 파라미터 값 찾기

- 하이퍼파라미터
  - 각층의 뉴런 수, 배치 크기, 학습률, 가중치 감소율
- 검증데이터를 사용해라
  - 훈련데이터 : 학습
  - 검증데이터 : 하이퍼파라미터 성능 평가
  - 시험데이터 : 신경망의 범용 성능 평가
- 방법
  - 0단계 : 하이퍼 파라미터 값의 범위 결정
  - 1단계 : 설정된 범위내에서 무작위 추출
  - 2단계 : 1단계에서 추출한 값으로 학습 후 검증 데이터로 평가 (에폭은 작게)
  - 3단계 : 1단계와 2단계 반복 (100회등). 정확도 결과를 보고 하이퍼파라미터 범위를 좁힌다.

```
weight _decay = 10 **
np.random.uniform(-8, -4)
lr = 10 ** np.random.uniform(-6, -2)
```

```
Best-1 (val acc:0.83) | lr:0.0092, weight decay:3.86e-07
Best-2 (val acc:0.78) | lr:0.00956, weight decay:6.04e-07
Best-3 (val acc:0.77) | lr:0.00571, weight decay:1.27e-06
Best-4 (val acc:0.74) | lr:0.00626, weight decay:1.43e-05
Best-5 (val acc:0.73) | lr:0.0052, weight decay:8.97e-06
```

그림 6-24 실선은 검증 데이터에 대한 정확도, 점선은 훈련 데이터에 대한 정확도