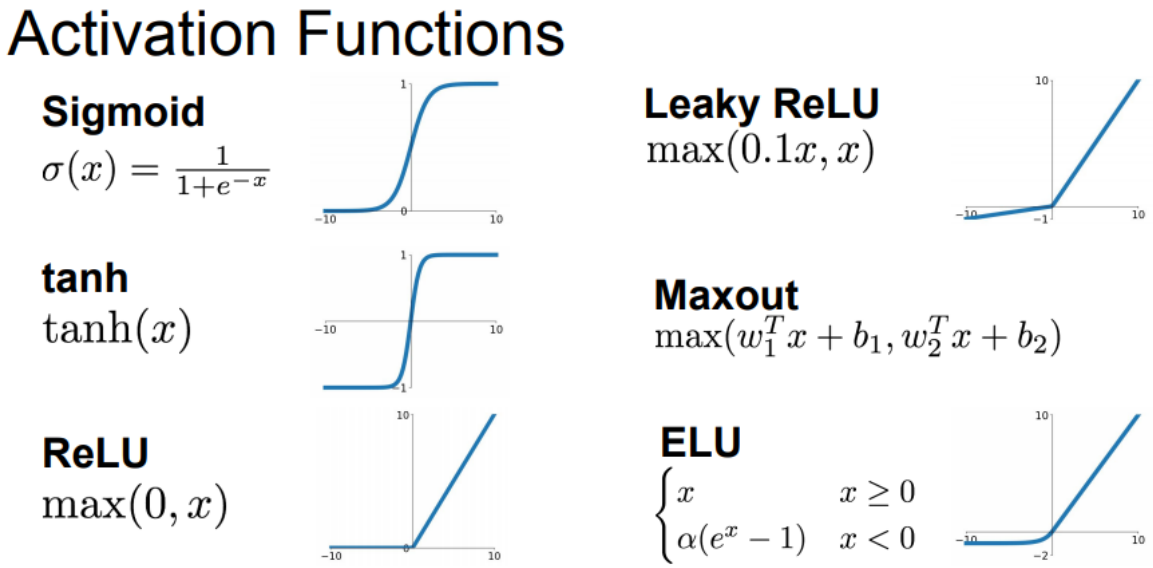
Activation Functions



Sigmoid: 0 과 1 사이의 output 을 내놓음. 문제점은 아래와 같음.

- saturation 되는 것이 gradient 를 kill 할 수 있음.
- zero-centered 되지 않는 output
- exp() 함수가 비용이 많이 듦. (이 문제는 그리 크지 않음)

-> input 이 모두 같은 부호가 되면, W 가 모두 같은 방향으로만 움직이게 됨. 그래서 파라미터 업데이트 시에 다같이 증가하거나 다같이 감소하여 사분면 중 두개의 영역만 사용하게 됨. 따라서 zero-mean data 인 것이 좋음

tanh: sigmoid 의 zero-centered 문제를 해결한 모습으로, -1 과 1 사이의 output 을 내놓음.

- 여전히 saturation 문제 있음

ReLU: 0 보다 작은 input 에서는 0 을 output 으로, 그 외의 구간에서는 x 를 output 으로 내놓음.

- 일부 saturation 문제 해결
- 계산 효율 좋음
- zero-centered data 가 아님

- 양수값에서는 saturation 발생하지 않으나 음수값에서는 saturation 발생

-> dead ReLU: data 의 절반만 activate 될 수 있음.

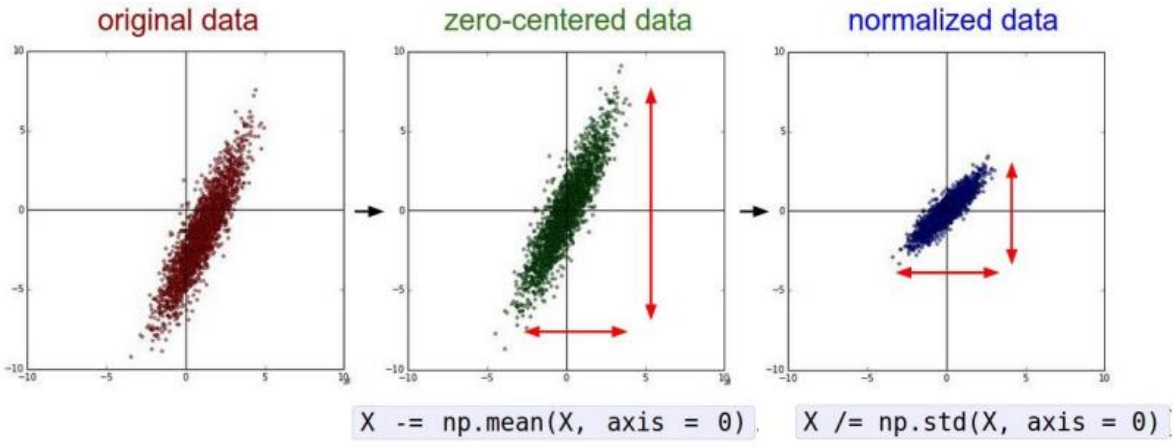
- 초기화를 잘못한 경우
- learning rate 가 지나치게 높은 경우

Leaky ReLU: 음의 구간에 어떤 파라미터 alpha 를 곱해 음의 구간에서의 saturation 문제 해결
ELU: exp() 이용한 것. 하지만 saturation 될 수 있음
Maxout Neuron: 기존 파라미터의 두배의 파라미터 수를 가지게 됨. ReLU 와 Leaky ReLU 를 좀 더 일반화 시킨 것.

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

결론적으로, ReLU 를 주로 사용하고 Leaky ReLU, Maxout, ELU 를 시도해볼 수 있음. tanh 도 가능은 하나 권장하지 않음. 그러나 sigmoid 는 사용하지 않는 것이 좋음

Data Preprocessing



주로 zero-centered data 로 만들어주고, normalization 을 거침. normalization 은 모든 차원이 동일한 범위 내에 존재하게 하여 동등한 기여를 할 수 있게 함.

(주로, 이미지의 경우는 zero-centered data 까지만 맞춰줌. 이미지는 대부분 이미 스케일이 어느정도 맞춰져있어서 그 외의 것들은 굳이 필요 없음.)

Weight initialization

모든 W=0 이라면, 모든 뉴런이 전부 똑같은 연산을 수행하게 될 것. 모두 같은 연산을 하면, output 이 모두 같아지고, gradient 도 같아짐. 모든 가중치가 똑같은 값으로 업데이트 되며, 모든 뉴런의 모양이 같아질 것.(=> Symmetric breaking")
따라서, weight 는 같은 값으로 초기화해서는 안됨.
해결)

- weight 를 임의의 작은 값으로 초기화: 작은 network 에서는 괜찮지만 좀 더 깊은 network 에서는 update 가 제대로 이루어지지 않을 것.
- weight 를 그보다 큰 값으로 초기화: saturation 발생으로 출력은 항상 +1/-1 의 값만 가지며, 업데이트가 제대로 이루어지지 않을 것.

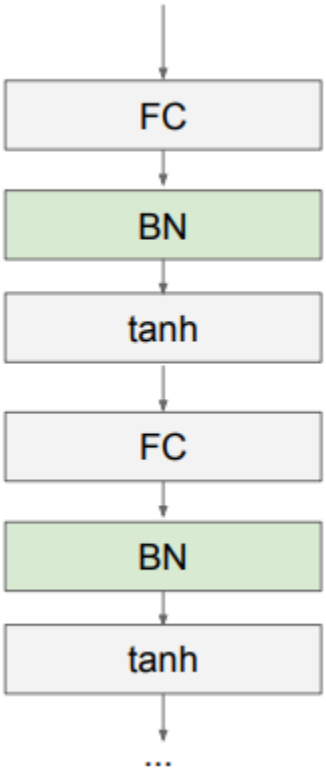
-> *Xavier initialization*: 입출력의 분산을 막아줌. 그러나 ReLU 를 사용하게 되면 잘 작동하지 않음 =>ReLU 는 데이터의 절반을 0 으로 만들어버리므로, 분포가 줄어들게 됨. 이를 해결하기 위해 뉴런의 절반이 사라졌다는 것을 반영하여 2 로 나눠주면 됨.

Batch Normalization

Gradient Vanishing 이 나오지 않도록 하는 아이디어. activation function 의 변화가 아닌 training 과정 자체를 안정화하는 역할을 함.
-> training 하는 동안 모든 layer 의 input 이 unit gaussian 이 됐으면 함 -> forward pass 동안 그렇게 되도록 명시적으로 만들어 주면 됨. -> batch 단위로 한 레이어에 input 으로 들어오는 모든 값들을 이용하여 평균과 분산을 구해 normalization 해주면 됨. (미분 가능한 함수에서 가능함; 평균과 분산을 상수로 가지고만 있으면 미분 가능)

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

BN 은 input 의 scale 만 살짝 변경하는 역할을 하며, FC 나 Conv 뒤에 사용함. bad scaling effect 가 발생하나, 해당 bad effect 는 충분히 상쇄될 수 있음.



- 모든 mini-batch 마다 각각 평균과 분산을 계산해주고, normalization 이후 scaling, shifting factor 를 사용함
- gradient 의 흐름을 보다 원활하게 해주며, 학습이 더 잘되게 해주는 역할
- learning rate 를 높일 수 있으며, 다양한 initialization 을 수행할 수 있음
- regularization 의 역할을 수행하기도 함
- 선형 변환으로, 공간적 구조가 잘 유지됨
- test phase 에서 새로운 계산을 굳이 요구하지 않음

Babysitting the Learning Process

1. Preprocess the data: zero-center, normalization (cv 에서는 굳이 normalization x)
2. Choose the architecture: hidden layer 를 어떻게 구성할 것인지 대략적으로 선택

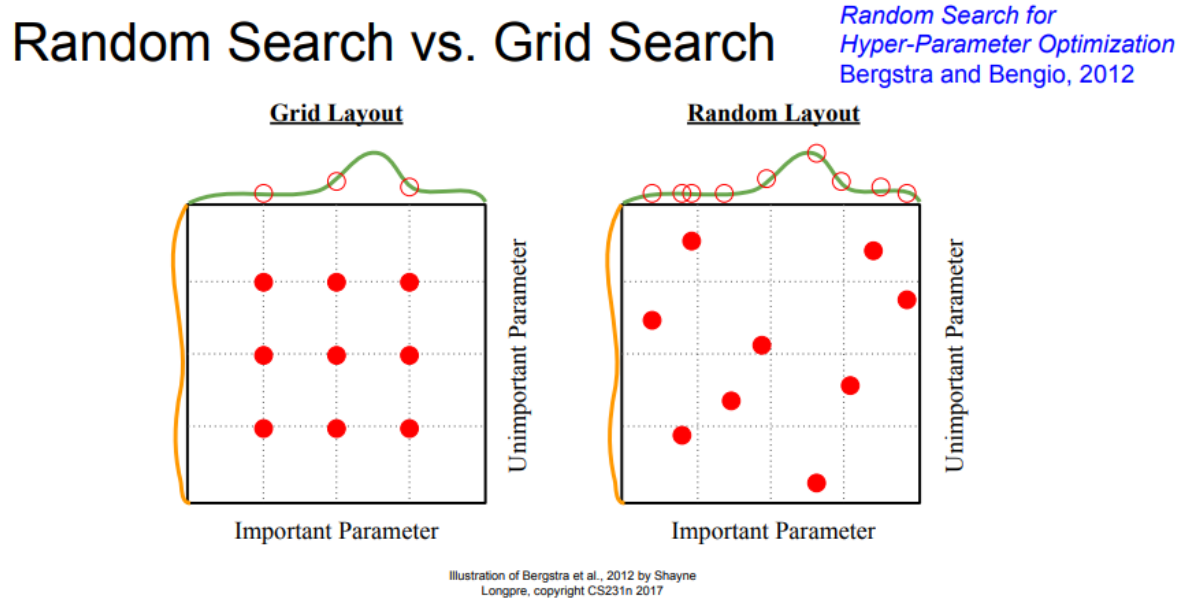
3. Check the loss is reasonable: sanity check 로 layer 가 동작하는걸 확인함
4. Train: 작은 dataset 을 먼저 넣어보고, 이때 overfitting 되어 train accuracy 가 100%가 나오면 모델이 동작하고 있음을 의미함
5. Training with regularization and learning rate: 각각에 대해 적절한 값을 찾아봄. 이때 cost 값과 training 값 이용

Hyperparameter Optimization

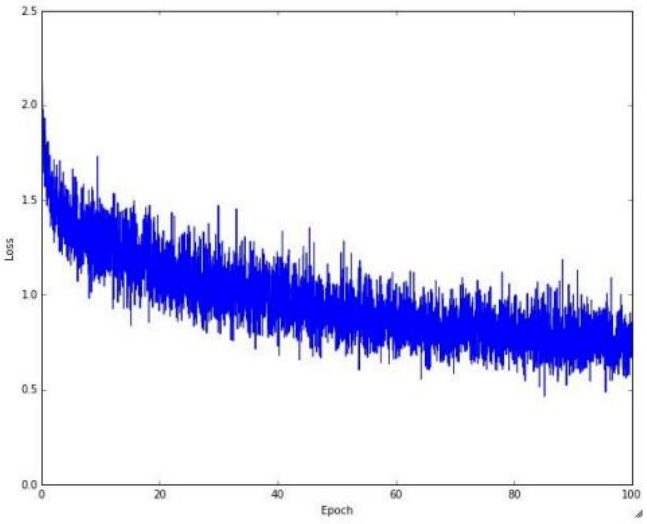
Cross-validation: training set 으로 학습시키고, validation set 으로 평가하는 방법. 값을 넓은 범위(coarse stage)에서 좁은 범위(fine stage)로 줄여나가는 것. learning rate 는 gradient 와 곱해지므로, 선택 범위는 log scale(-> 차수만 사용)을 사용하는 것이 좋음.

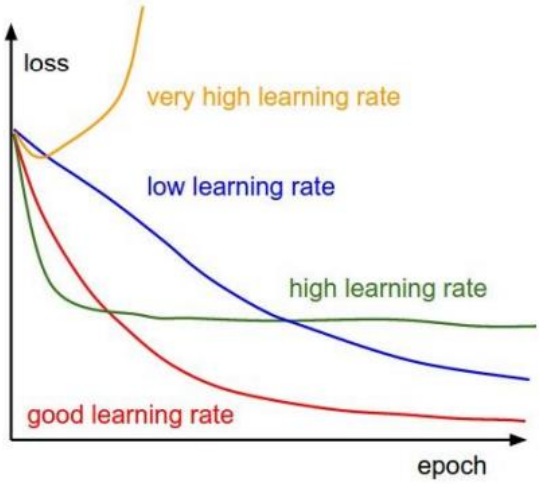
Grid search: 하이퍼파라미터를 고정된 값과 간격으로 샘플링하는 것. 정해진만큼의 sampling 만 가능하므로 good region 을 찾지 못할 수 있음

Random search: 랜덤값을 이용하는 것. grid search 보다 좋음. important variable 에서 더 다양한 값을 샘플링할 수 있음

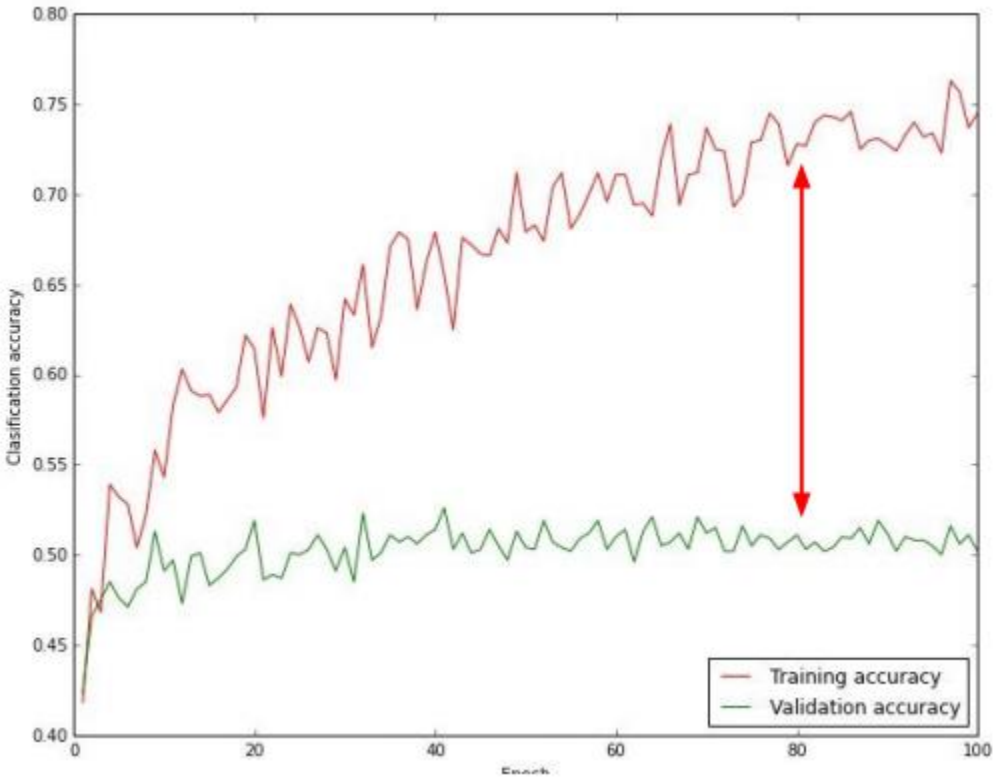


+) 하이퍼파라미터 종류: learning rate, decay schedule, update type, regularization, network architecture, hidden unit 과 depth 의 수 등





-> loss 가 발산하면 learning rate 이 높은 경우고, 평평하다면 learning rate 이 너무 낮은 경우임. loss 가 평평하다가 갑자기 가파르게 내려가는 경우는 주로 초기화에서 문제가 발생했다고 볼 수 있음(gradient 의 backprop 이 잘 되지 않다가 학습이 진행되면서 회복되는 것). 위의 좌측이 최적의 learning rate 의 모습이라 할 수 있음



- train_acc 와 va_cc 의 차이가 크다면, overfit 일 수 있음 -> regularization 의 강도 높이기

- gap 이 없는 경우, overfit 이 아직 일어나지 않은 경우로, capacity 를 높일 수 있는 여유가 된다는 의미임