Neural Networks and Deep Learning

CS5242

Wei WANG

National University of Singapore

cs5242@comp.nus.edu.sg

Recap

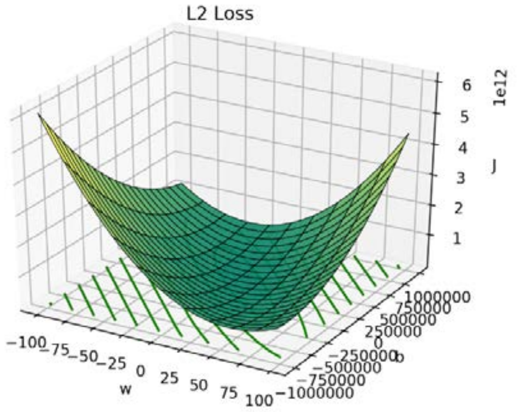
Machine learning basics

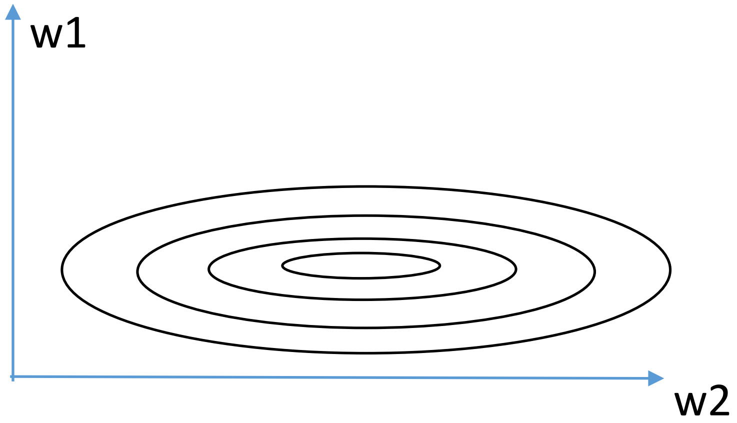
- Linear regression for house price prediction
- Notations/concepts, e.g. feature, loss, sample, parameter
- Backpropagation
    - Single sample, single feature
    - Single sample, multiple features
    - Multiple samples and multiple features
- Gradient descent and challenges →
    - Data normalization
    - SGD, mini-batch SGD, RMSprop and Adam
- Fundamentals of training
    - Underfitting, overfitting, bias and variance
    - Capacity/complexity

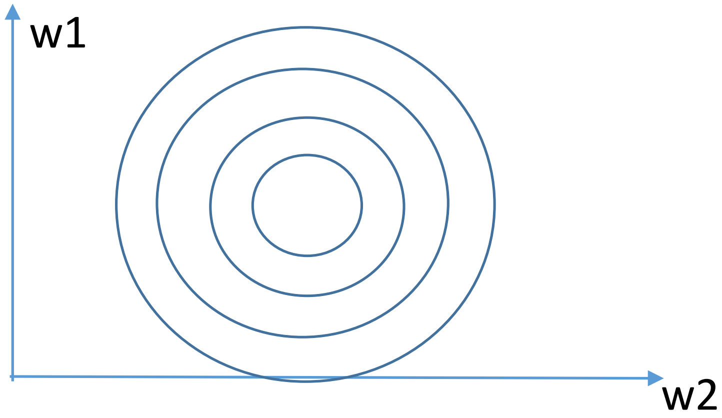
Challenges of GD

1. Features of different scale
2. Deep non-linear transformation

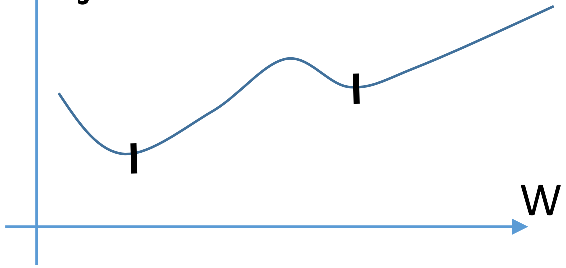
→ Steepest direction is not optimal; different learning rates are necessary for different parameters



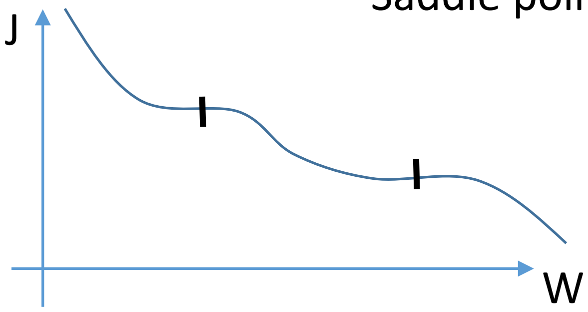


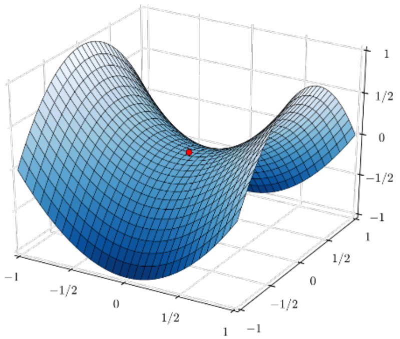


Local optimum



Saddle points





from: https://en.wikipedia.org/wiki/Saddle_point

$w \in R^m$ is local optimum $\rightarrow \forall i, w_i$ is local optimum when m is large $\rightarrow$ unlikely

Saddle points are more common than local optimum

Efficiency of memory and speed

BP or GD?

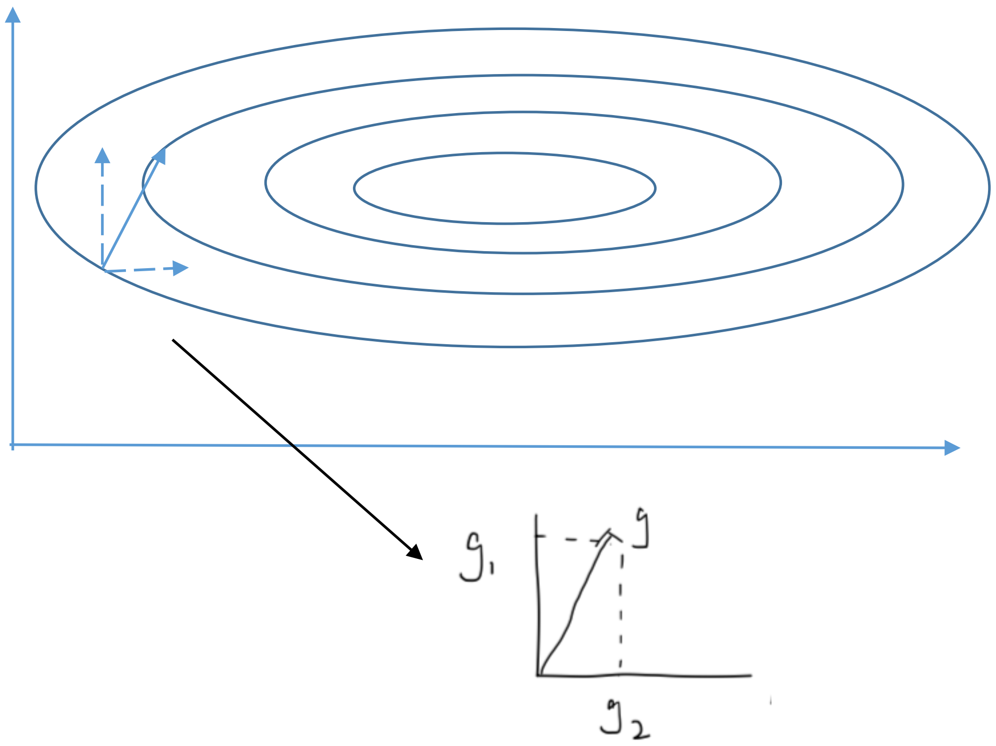
Improvements against GD

- Local optimum/saddle points
  - Stochastic gradient descent (SGD)
    - $\frac{\partial J^{(i)}}{\partial w}$ may not be zero although $\frac{\partial J}{\partial w}$=0; $\frac{\partial J^{(i)}}{\partial w}$ may not be zero although $\frac{\partial J^{(i-1)}}{\partial w}$ =0
      - Able to jump out of local optimum/saddle points (in the next iterations)
    - Faster per iteration and consumes less memory
  - However, is not stable as the gradient is computed over only a single sample
    - Noisy data
    - Moving in wrong direction → more iterations

- Mini-batch SGD improves over SGD
  - b samples per iteration (b is smaller than the dataset size, e.g. 32, 64)
  - More stable than SGD by averaging the gradients from b samples
  - Converge faster than SGD (fewer iterations)
- **Momentum**
  - Accumulate the historical gradients by exponential average
  - Accelerate the update if the current gradient is consistent with the momentum
  - Avoid big changes (errors) if the current (noisy) gradient is very different to the momentum
- RMSprop improves of SGD
  - Different (adaptive) learning rates for different parameters, e.g. for **ellipse loss contour**
- **Adam**
  - Combine momentum and RMSprop

RMSprop

- $s = (1-\beta)s + \beta g^2$
- $w = w - \alpha g / \sqrt{s + \epsilon}$



Rescale g_1, g_2 to decrease g_1 and increase g_2

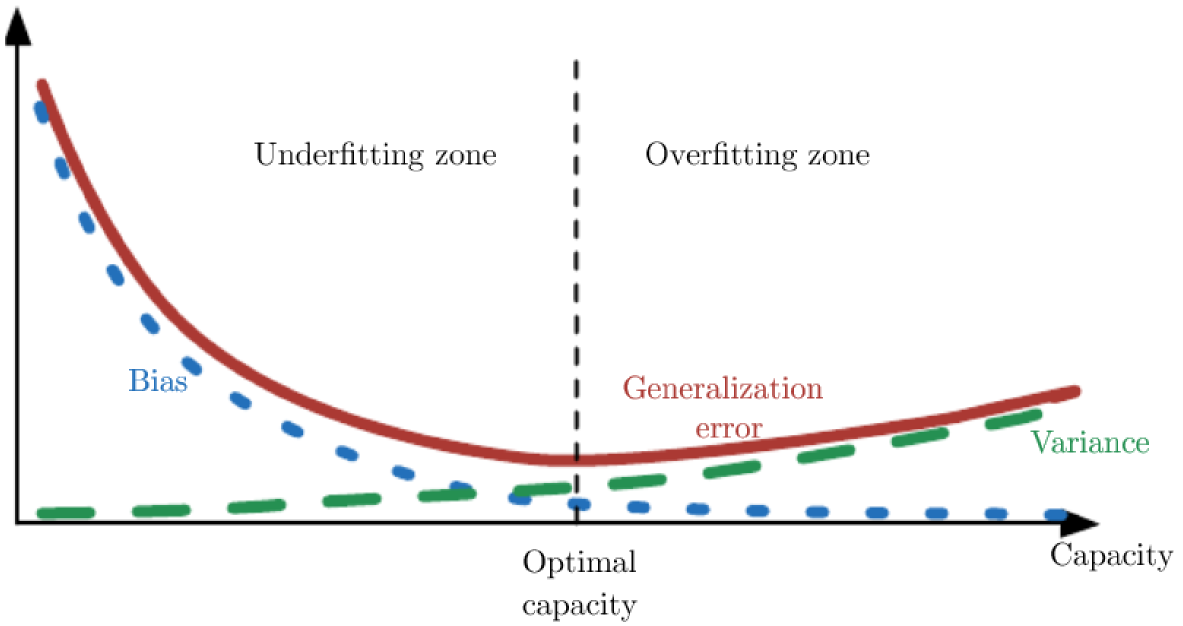
Adaptive learning rate for every parameters

Intuition 1: $|g_2| < |g_1|$
but moving along g_2 would decrease the loss faster $\rightarrow$ need to rescale g_1, g_2

Intuition 2: to prevent large noisy (wrong) derivatives along some directions.

Jupyter notebook for optimization

Fundamentals of machine learning



Source from [1]

- Underfitting
  - More complex model
- Overfitting
  - Regularization/constraint to the hypothesis space
  - Ensemble modeling
  - Early stopping
- L2 regularization
  - $min_{\theta} J(\theta) + \lambda|\theta|^2$
  - From Bayesian's perspective, L2 norm is equivalent to gaussian prior for the parameters
  - Partial derivative
    - $\frac{\partial J}{\partial \theta} + 2\lambda\theta$

What’s next?

- Deep neural networks
    - More complex models → less underfitting?
        - MLP, CNN, RNN
    - How to avoid **underfitting** due to poor optimization?
        - Saddle points, local optimum, ellipse loss contour, etc.
    - How to avoid **overfitting** when the models are very complex?

Multilayer Perceptron

Binary image classification

Source: https://goo.gl/SkMNZh

- Task
  - Differentiate images for cat and dog
- Dataset
  - a subset from Kaggle https://www.kaggle.com/c/dogs-vs-cats/data
  - Pre-processed by resize to 32x32
  - 5000 training images, 500 validation and 500 test images
    - Each of size 3x32x32
    - A numpy array of shape (5000, 3, 32, 32), (500, 3, 32, 32) and (500,3, 32, 32)

Binary image classification

- Feature (image representation in computer)
  - Traditional approaches
    - Colour histogram
    - Local interest point, e.g. SIFT
  - Deep learning approach
    - Pixel values flattened into a vector
    - Feature learning

Red	Green	Blue	Pixel Count
0	0	0	7414
0	0	1	230
0	0	2	0
0	0	3	0
0	1	0	8
0	1	1	372

https://en.wikipedia.org/wiki/Color_histogram

https://en.wikipedia.org/wiki/Scale-invariant_feature_transform

Binary image classification

- Label
  - 0 for cat, 1 for dog
- Performance
  - percentage of correctly labeled images, i.e. accuracy
  - VS loss?

Linear classifier

- $\tilde{y} = w^T x + b, w \in R^m, b \in R$ (m=?)
- $L(x, y|w, b) = |\tilde{y} - y|^2$

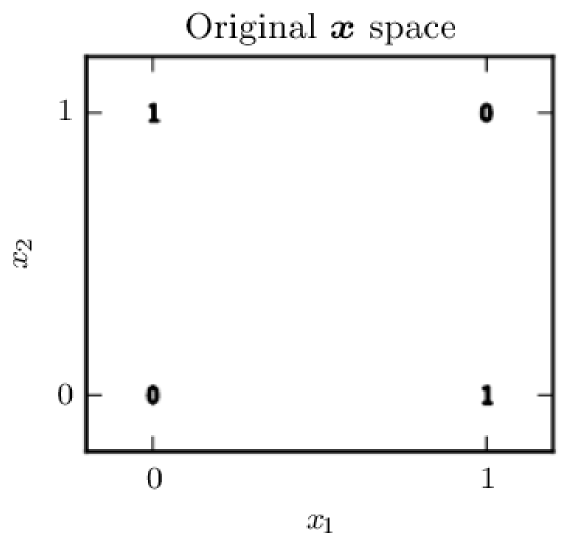
- Problems
  - Linear models are too simple to process images with complex structures

Linear classifier

training data

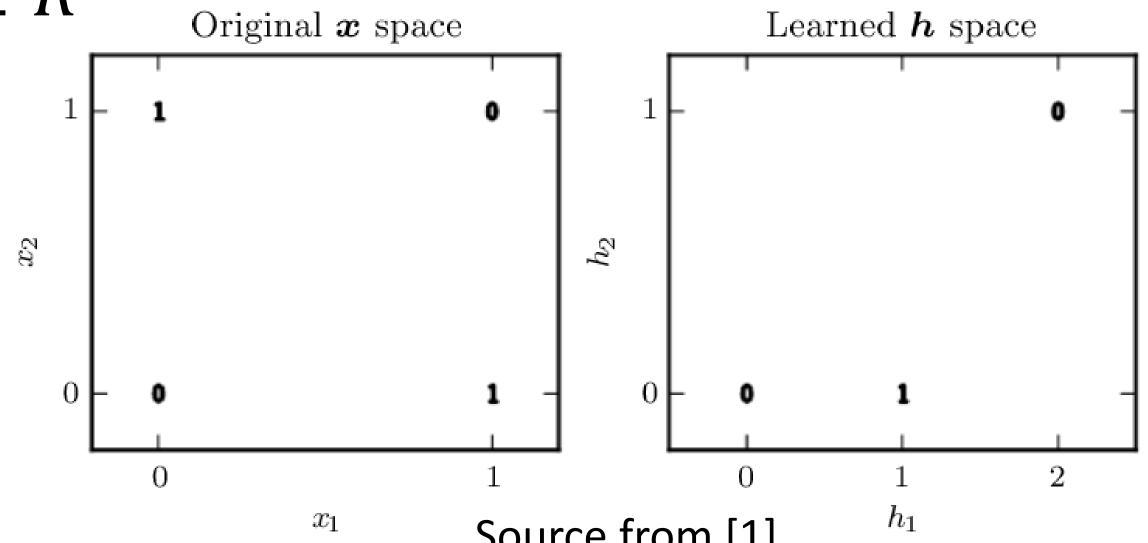
x1	x2	Y
0	0	0
0	1	1
1	0	1
1	1	0

- (Single layer) Perceptron for XOR
- $\tilde{y} = w^T x + b, w \in R^2, b \in R$
  - Cannot fit the training data no what the values of w and b are. Why?
  - Linear boundary



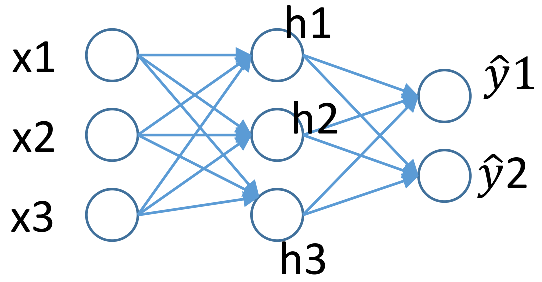
Feature transformation

- $h = \max(0, W^T x + c), W \in R^{2\times 2}, c \in R^2$
- $\tilde{y} = w^T x + b, w \in R^2, b \in R$



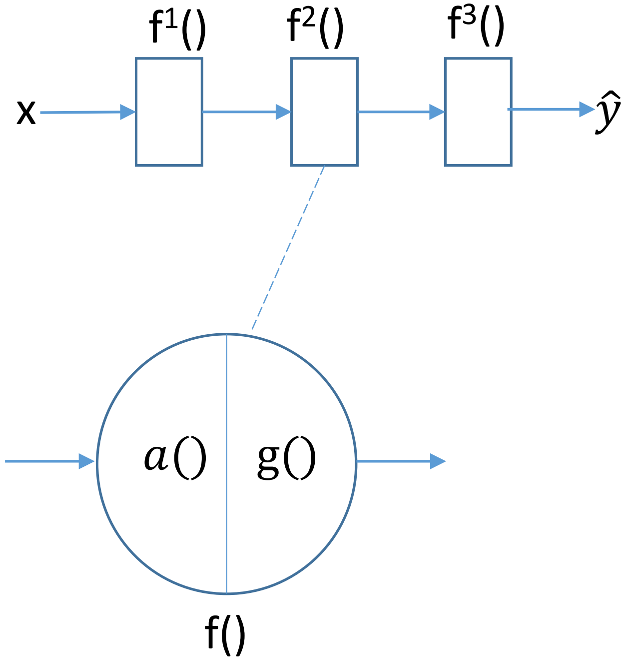
Source from [1]

Multilayer perceptron



- A net with multiple layers that transform input features into hidden features and then make predictions
    - At least one hidden layer
    - i-th layer consists of a linear transformation function

$$u^i = a^i\left(h^{i-1}\right) = W^{(i)T}h^{i-1} + b^i; W^{(i)T} \in R^{d_i \times d_{i-1}}, b^i \in R^{d_i}$$



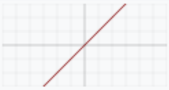
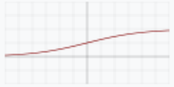
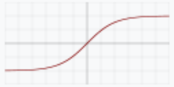
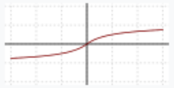
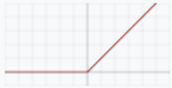
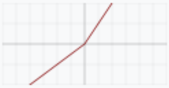
-
    - d_i is the number of hidden units (a hyper-parameter)
    - followed by a **non-linear activation** function

$$h^i = g^i(u^i), \in R^{d_i}$$

Activation functions

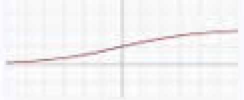
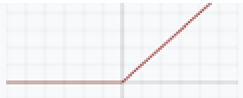
Source from: https://en.wikipedia.org/wiki/Activation_function

- Non-linear feature transformation
- Otherwise, MLP == single layer perceptron

Identity		$f(x)=x$	$f'(x)=1$	$(-\infty,\infty)$
Binary step		$f(x)=\begin{cases}0 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0 & \text{for } x\neq 0\\ ? & \text{for } x=0\end{cases}$	$\{0,1\}$
Logistic (a.k.a. Soft step)		$f(x)=\frac{1}{1+e^{-x}}$	$f'(x)=f(x)(1-f(x))$	$(0,1)$
TanH		$f(x)=\tanh(x)=\frac{2}{1+e^{-2x}}-1$	$f'(x)=1-f(x)^2$	$(-1,1)$
ArcTan		$f(x)=\tan^{-1}(x)$	$f'(x)=\frac{1}{x^2+1}$	$\left(-\frac{\pi}{2},\frac{\pi}{2}\right)$
Softsign [7][8]		$f(x)=\frac{x}{1+\|x\|}$	$f'(x)=\frac{1}{(1+\|x\|)^2}$	$(-1,1)$
Rectified linear unit (ReLU)[9]		$f(x)=\begin{cases}0 & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$[0,\infty)$
Leaky rectified linear unit (Leaky ReLU)[10]		$f(x)=\begin{cases}0.01x & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0.01 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$(-\infty,\infty)$
Parameteric rectified linear unit (PReLU)[11]		$f(\alpha,x)=\begin{cases}\alpha x & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(\alpha,x)=\begin{cases}\alpha & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$(-\infty,\infty)$
Randomized leaky rectified linear unit (RReLU)[12]		$f(\alpha,x)=\begin{cases}\alpha x & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$ [1]	$f'(\alpha,x)=\begin{cases}\alpha & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$(-\infty,\infty)$
Exponential linear unit (ELU)[13]		$f(\alpha,x)=\begin{cases}\alpha(e^x-1) & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(\alpha,x)=\begin{cases}f(\alpha,x)+\alpha & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$(-\alpha,\infty)$

Activation functions

- Logistic (Sigmoid) VS ReLU ?

Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$	$(0,1)$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0,\infty)$

Output layer

- Problems of linear output?
- Probabilities for the Dog and Cat respectively
- logistic function
  - $\hat{y} = \sigma(h) = \frac{1}{1+e^{-h}}, \in R$, probability for P(y=1), i.e. Dog
  - Probability for Cat is 1 - $\hat{y}$

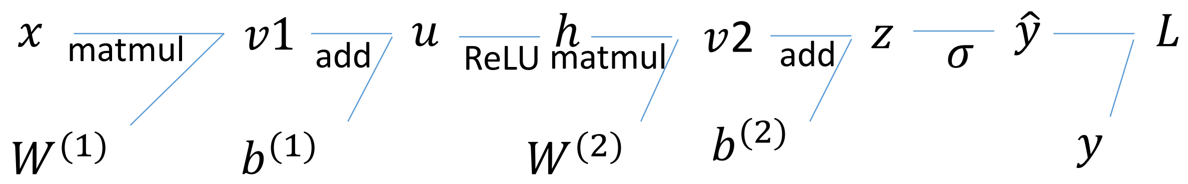
Cross-entropy loss

- Max Loglikelihood
  - $\mathrm{P}(correct|x) = P(predict = \mathrm{Cat}|\mathrm{y} = \mathrm{Cat}, \mathrm{x}) * \mathrm{P}(predict = Dog|\mathrm{y} = \mathrm{Dog}, \mathrm{x})$
  -
  - log$\mathrm{P}(correct|x) =$ log $P(predict = \mathrm{Cat}|\ \mathrm{x})$ = log $1 - \hat{y}$ if y=0;
  - log $\mathrm{P}(predict = Dog|\ \mathrm{x})$ = log $\hat{y}$ if y = 1;
  - = y log $\hat{y}$ + (1-y)log (1- $\hat{y}$) ?
- Min negative loglikelihood

Back-propagation

- MLP with one hidden layer for Dog-Cat classification

Back-propagation

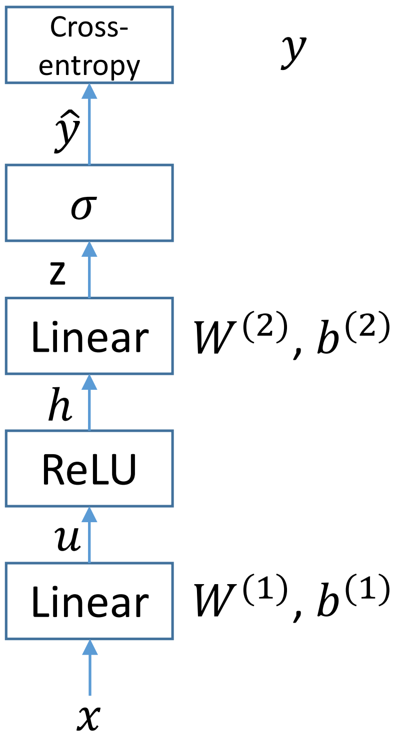


Back-propagation (Layer API)

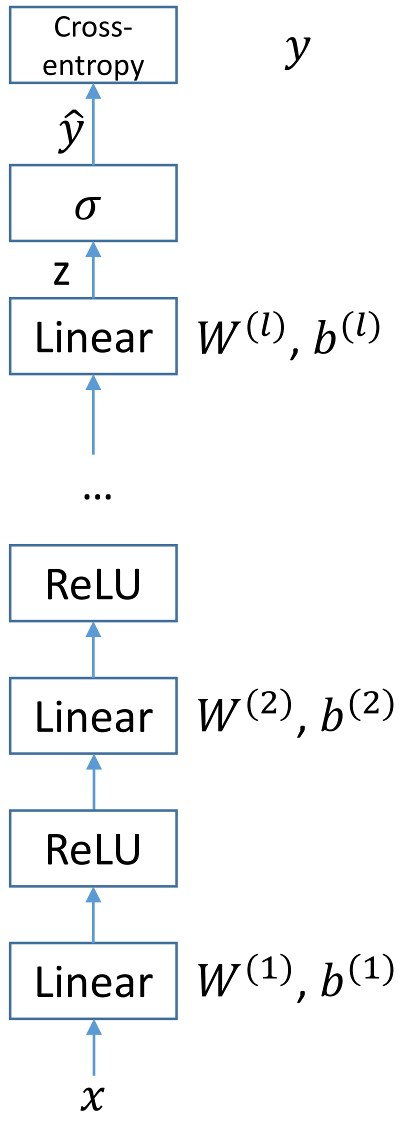
```python
class Layer(object):
    def __init__(self, name):
        self.name = name

    def forward(self, x, args=None):
        # return y, the output of this layer
        pass

    def backward(self, dy, args=None):
        # return gradients of the input x and parameters.
        pass
```



From shallow to deep



MLP online playground

http://playground.tensorflow.org

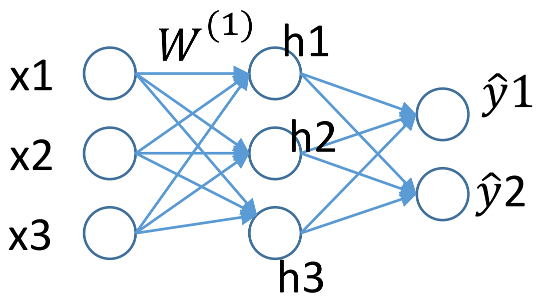
Universal approximate theorem quoted from [1]

- MLP---a feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function (such as the logistic sigmoid activation function)
- can approximate any Borel measurable function from one finite-dimensional space to another with any desired nonzero amount of error, provided that the network is given enough hidden units.
- Any continuous function on a closed and bounded subset of R^n is Borel measurable
- First, the optimization algorithm used for training may not be able to find the value of the parameters that corresponds to the desired function. Second, the training algorithm might choose the wrong function as a result of overfitting.

Training tricks for MLP

Asymmetric initialization

- All columns of W are the same
- → all hidden units are the same
- → derivatives of all hidden units are the same
- → derivatives of all columns of W are the same
- → W are updated in the same direction and length
- Repeat

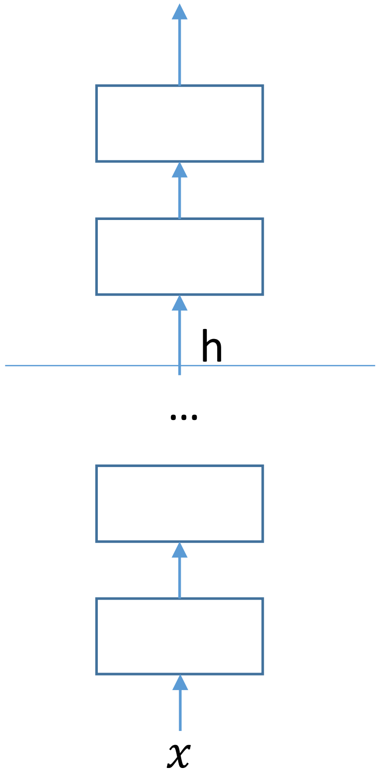


Asymmetric initialization

- Gaussian, N(0, 0.01)
- Uniform, U(-0.05, 0.05)
- Glorot/Xavier [20]
  - Uniform U(-sqrt(6/(fan_in + fan_out)), +sqrt(6/(fan_in + fan_out)))
  - **Gaussian N(0, sqrt(2/(fan_in + fan_out))**
- He/MSRA [21]
  - Uniform U(-sqrt(6/fan_in), +sqrt(6/fan_in))
  - **Gaussian N(sqrt(2/fan_in))**

Batch normalization

- Train a model M over dataset D
- If D's distribution changes (e.g. by adding new data samples)
- M should be updated



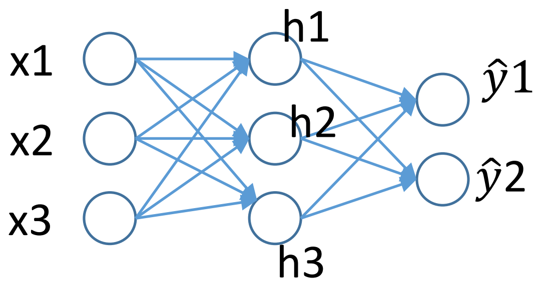
Batch normalization

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}} \qquad y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}.$$

- Normalize
  - Applied after convolution/fully connected before activation
  - After convolution layer. Compute mean and var per channel; one (γ, β) per channel.
- Running smooth to get global mean and var for inference
- Converge faster

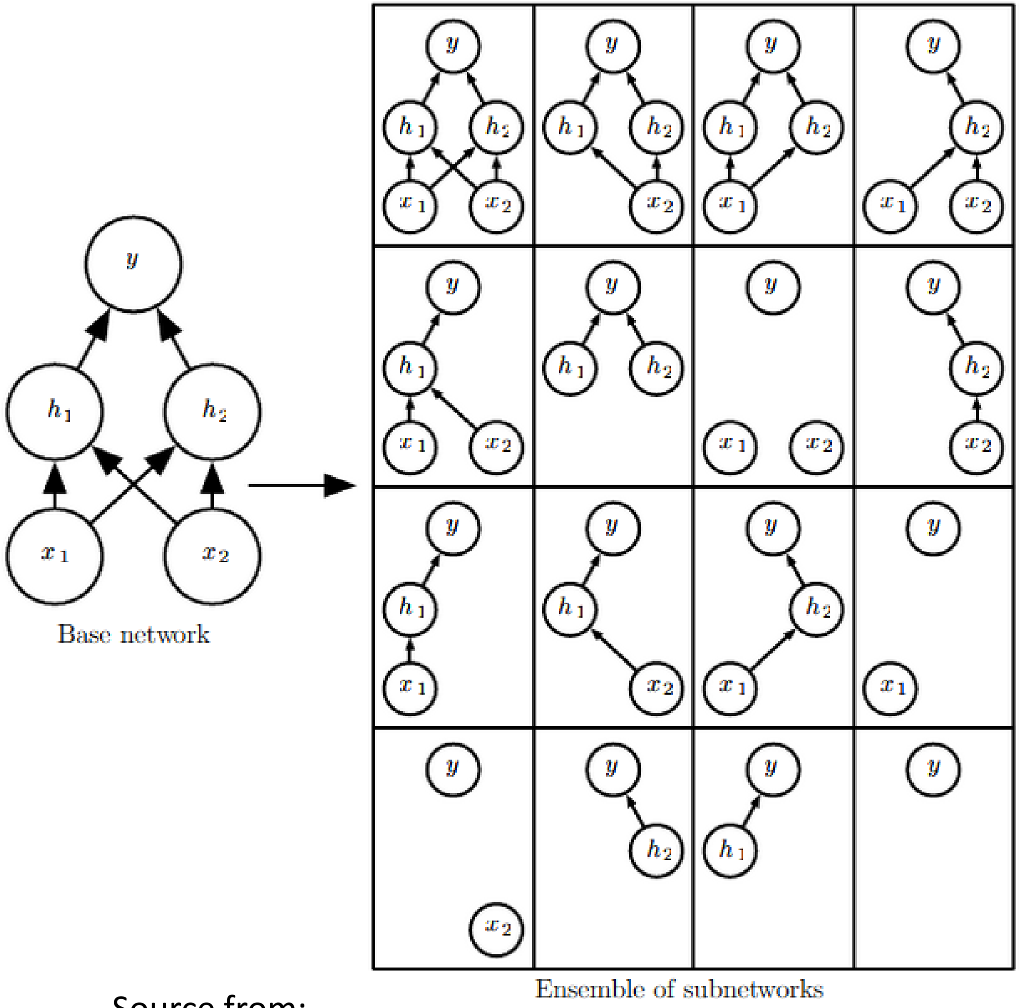
Dropout

- Training
  - randomly set some neurons to 0 with probability p (0.5)
  - (Caffe) Multiple the outputs (h) with scale 1/(1-p) ;
- During inference
  - Do nothing



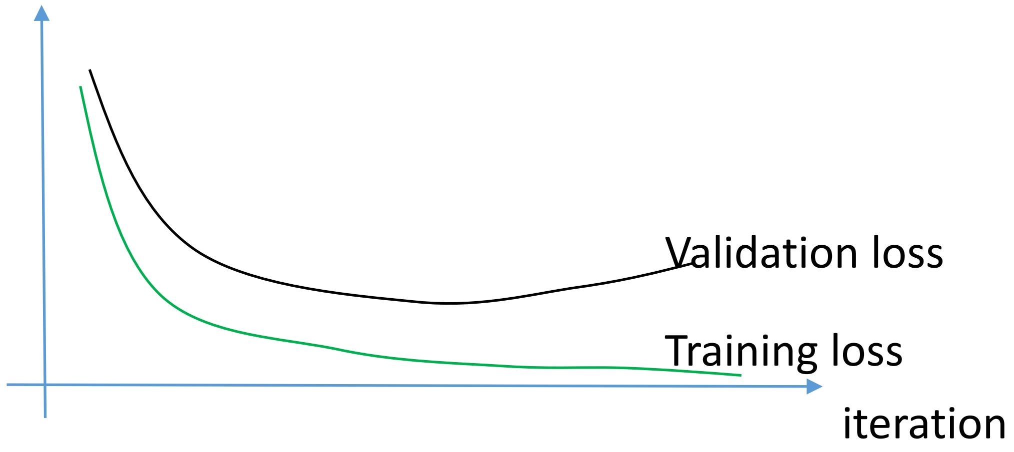
Dropout

- Ensemble modeling
  - $Pensemble(y|x) = \sum_z P(z)P(y|x,z)$



Source from:
http://www.deeplearningbook.org/contents/regularization.html

Early stopping



References

- MLP
  - https://www.youtube.com/watch?time_continue=181&v=aircAruvnKk
  - https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi
- BP
  - https://www.youtube.com/watch?v=tIeHLnjs5U8
  - https://www.youtube.com/watch?v=q555kfIFUCM

- [1] Goodfellow Ian, Bengio Yoshua, Courville Aaron. Deep learning. MIT Press. http://www.deeplearningbook.org.