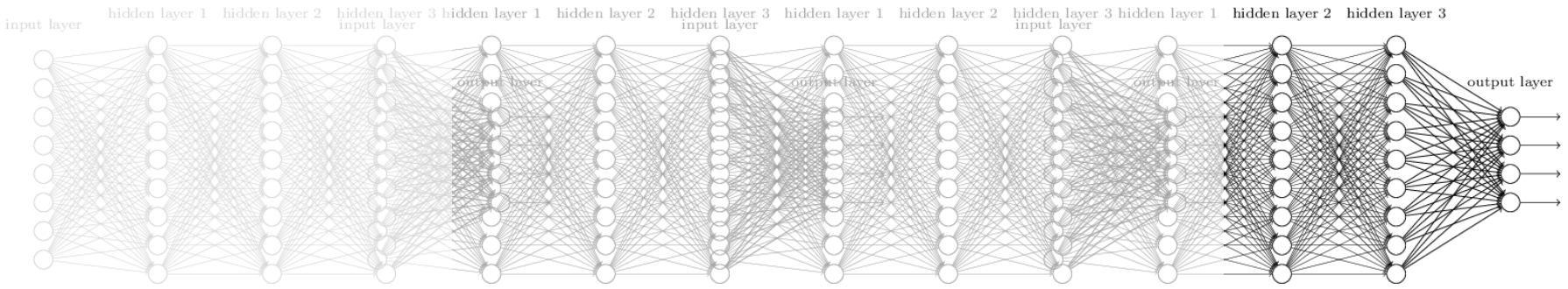
심층 신경망

Deep Neural Network (DNN)

Problem

① Neural Network(NN) 을 이용한 XOR 문제 해결
② Deep Neural Network (DNN) 을 통한 어려운 문제 도전
③ DNN에서 Gradient Vanishing 문제 발생



Geoffrey Hinton's summary of findings up to today

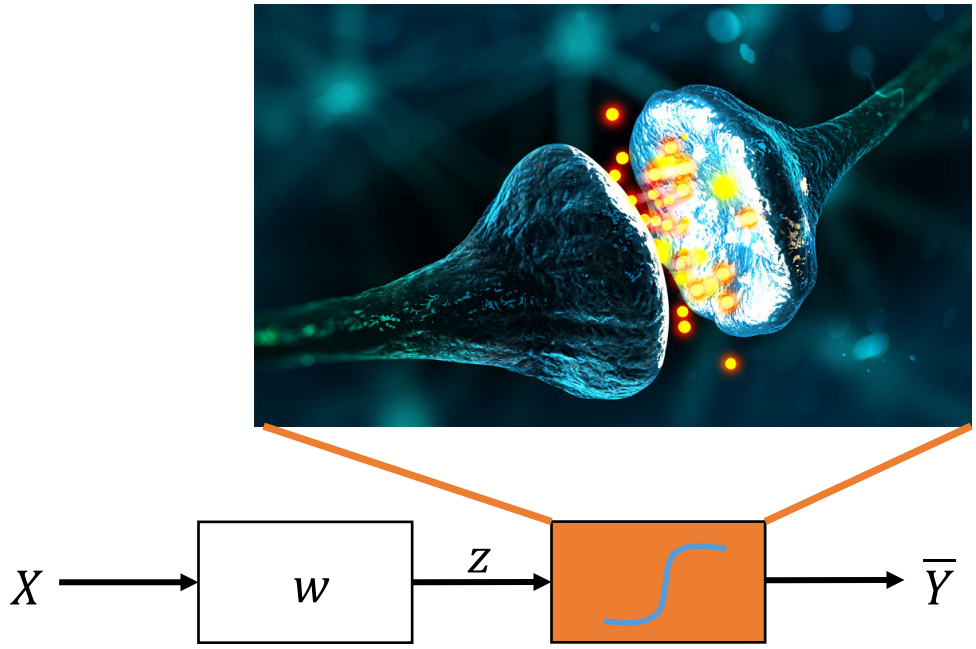
➔ DNN에서 Gradient Vanishing 문제 해결법

- Our labeled datasets were thousands of times too small
- Our computers were millions of times too slow
- We initialized the weights in a stupid way
- We used the wrong type of non-linearity

Activation Function

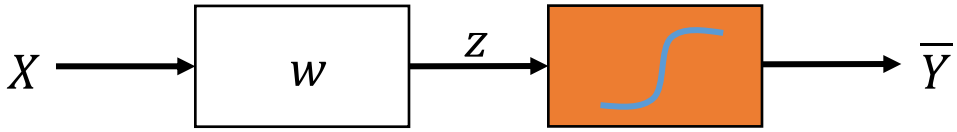
Activation Function

- 활성화 함수(Activation Function)는 신경학적으로 볼 때 뉴런 발사 (Firing of a Neuron)의 과정에 해당함
- 최종 출력 신호를 다음 뉴런으로 보내줄지 말지 결정하는 역할을 함



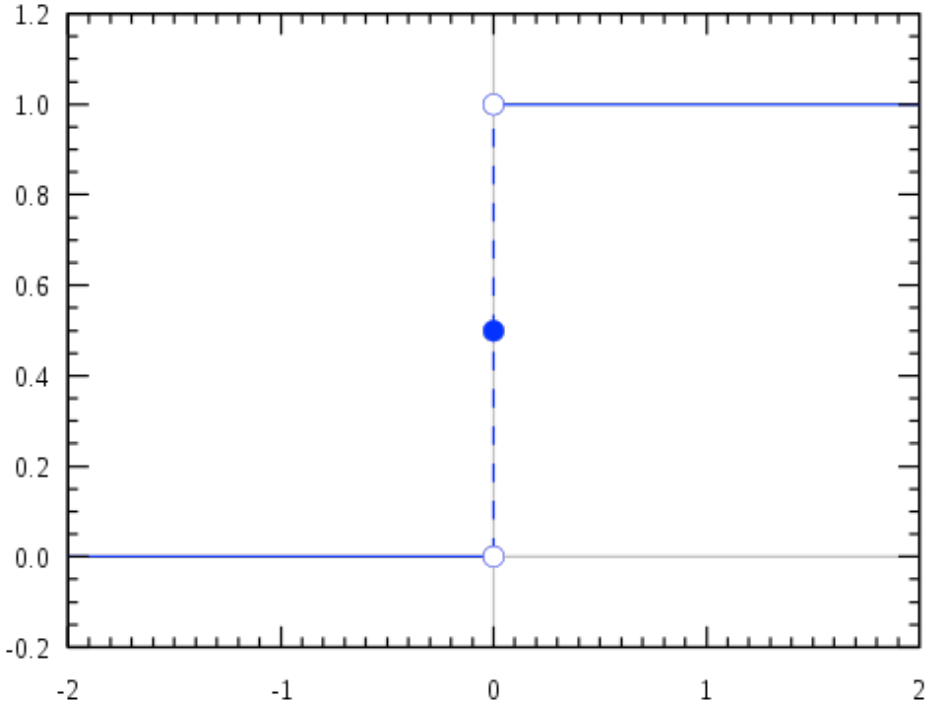
Activation Function

- 뉴런이 다음 뉴런으로 신호를 보낼 때 입력신호가 일정 기준 이상이면 보내고 기준에 달하지 못하면 보내지 않을 수도 있음. 즉, 활성 함수란 그 신호를 결정해주는 것
- 많은 종류의 활성화 함수가 있고, Activation function의 결정이 결과에 크게 영향을 미침



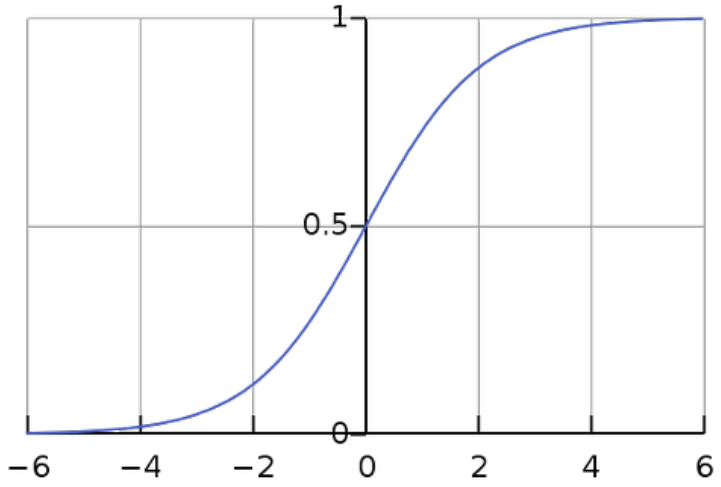
Activation Function: Step

- 입력이 양수 일 때는 1, 음수 일 때는 0 의 신호를 보내주는 이진 함수
- 미분 불가능한 함수로 모델 Optimization과정에 사용이 어려워 신경망의 활성 함수로 사용하지 않음



Activation Function: Sigmoid

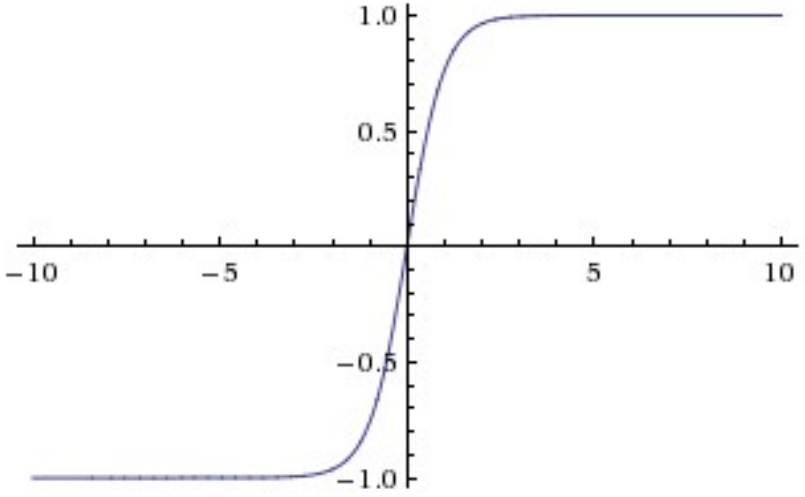
- 단일 퍼셉트론(perceptron)에서 사용했던 활성 함수
- 입력을 (0,1) 사이로 정규화(normalization) 함
- Backpropagation 단계에서 NN layer 를 거칠 때마다 작은 미분 값이 곱해져, Gradient Vanishing 을 야기함. 여러 개의 Layer를 쌓으면 신경망 학습이 잘 되지 않는 원인
- Deep Layer (3개 이상)에서 활성 함수로 사용을 권하지 않음



$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Activation Function: tanh

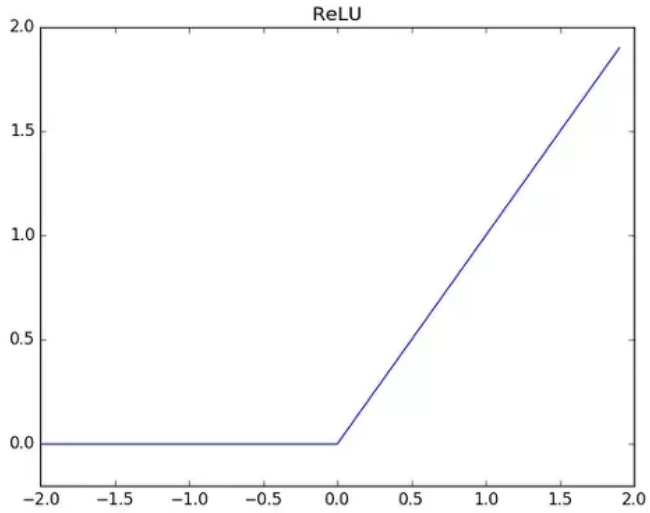
- Sigmoid를 보완하고자 제안된 활성 함수
- 입력을 (-1, 1)사이의 값으로 정규화(normalization) 함
- Sigmoid 보다 tanh 함수가 전반적으로 성능이 좋음
- 여전히 Gradient Vanishing 문제는 발생함 (Sigmoid 보다는 덜 발생함)



$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Activation Function: ReLU (Rectified Linear Unit)

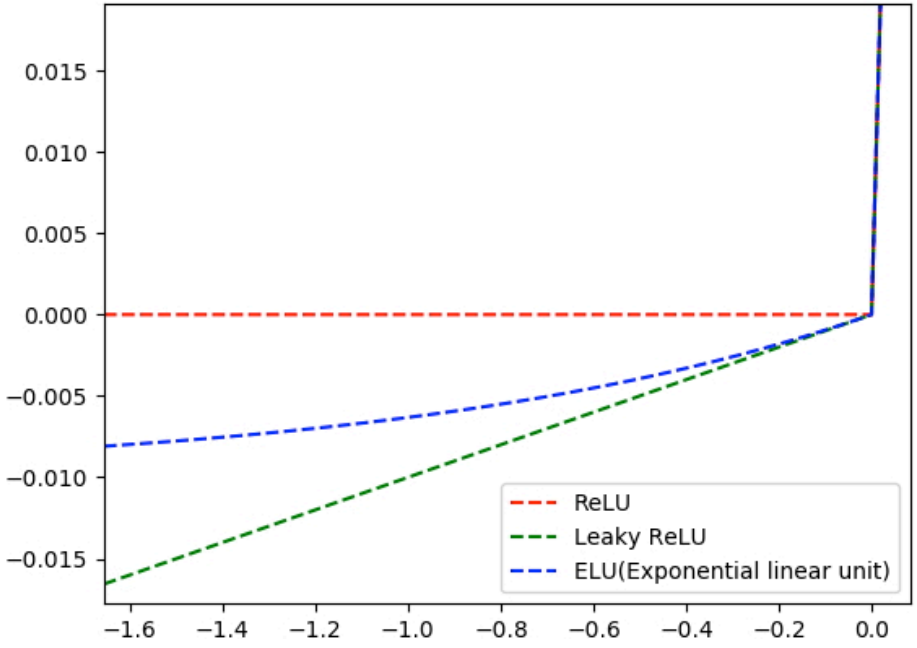
- 현재 가장 인기있는 활성화 함수
- 양수에서 Linear Function과 같으며 음수는 0을 출력하는 함수
- 미분 값을 0 또는 1의 값을 가지기 때문에 Gradient Vanishing 문제가 발생하지 않음
- Linear Function과 같은 문제는 발생하지 않으며, 엄연히 Non-Linear함수 이므로 Layer를 deep하게 쌓을 수 있음.
- exp() 함수를 실행하지 않아 sigmoid함수나 tanh함수보다 6배 정도 빠르게 학습이 진행됨



$$Relu(x)=max(0,x)$$

Activation Function: Leaky ReLU

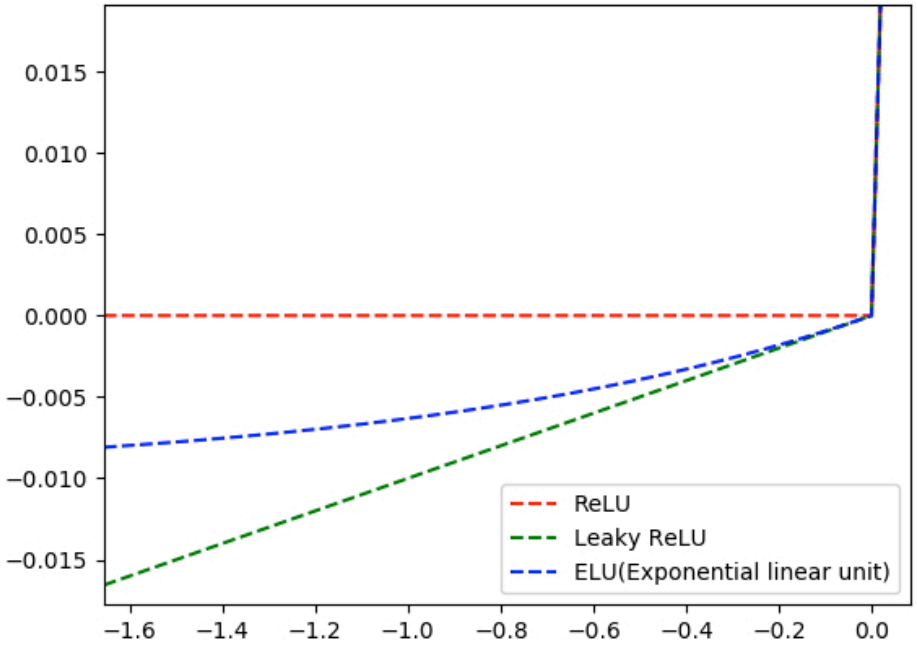
- Leaky ReLU는 “dying ReLU” 현상을 해결하기 위해 제시된 함수
- ReLU는 x<0인 경우 함수 값이 0이지만, Leaky ReLU는 작은 기울기를 부여함
- 보통 작은 기울기는 0.01을 사용함
- Leaky ReLU로 성능향상이 발생했다는 보고가 있으나 항상 그렇지는 않음



*Leaky Relu(x)=max(0.01*x,x)*

Activation Function: ELU (Exponential Linear Units)

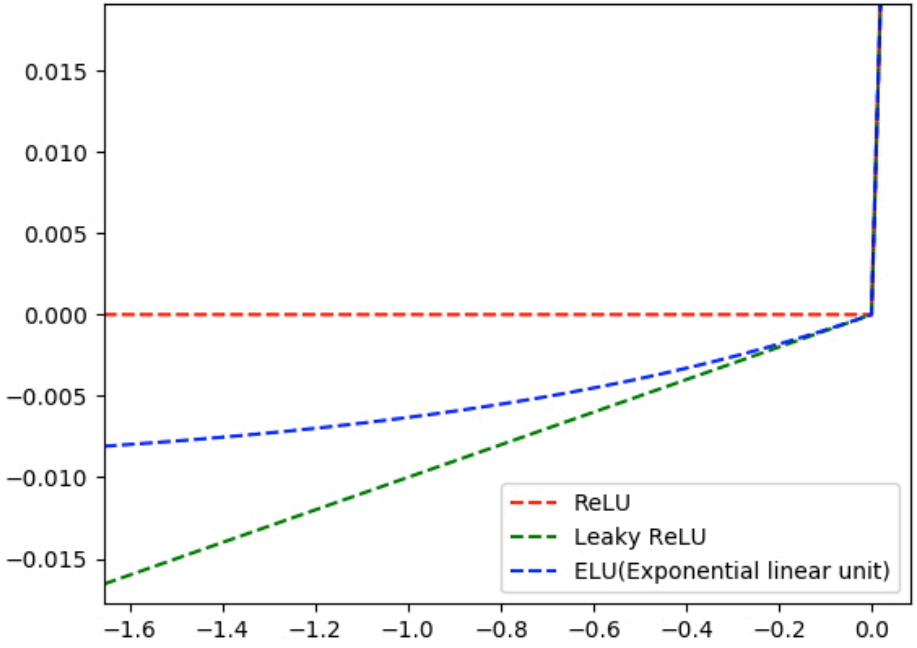
- ReLU의 threshold를 -1로 낮춘 함수를 exp^x를 이용하여 근사한 것
- dying ReLU 문제를 해결함
- 출력 값이 거의 zero-centered에 가까움
- 하지만 ReLU, Leaky ReLU와 달리 exp()를 계산해야하는 비용이 비듬



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

Activation Function: Maxout

- 이 함수는 ReLU와 Leaky ReLU를 일반화 한 것. ReLU와 Leaky ReLU는 이 함수의 특수한 경우
- Maxout은 ReLU가 갖고 있는 장점을 모두 가지며, dying ReLU 문제도 해결
- ReLU 함수와 달리 한 뉴런에 대해 파라미터가 두배이기 때문에 전체 파라미터가 증가한다는 단점이 있음



$$f(x) = max(w_1^T x + b_1 + w_1^T x + b_1)$$

Activation Function: Conclusion

❑가장 먼저 ReLU를 시도

- ❑ 다양한 ReLU인 Leaky ReLU, ELU, Maxout등이 있지만, 현재까지 가장 많이 사용되는 activation은 ReLU임

❑다음으로 Leaky ReLU, Maxout, ELU를 시도

- ❑ 성능이 좋아 질 수 있는 가능성이 있음
- ❑ 그러나 반드시 좋아지는 것은 아님

❑tanh를 사용해도 되지만 성능이 개선될 확률이 적음

❑앞으로 Deep NN에서는 Sigmoid는 피한다

Maxout > ELU, Leaky ReLU >= ReLU > tanh >= sigmoid

Activation Function: Conclusion

The compatibility of activation functions and initialization.
Dataset: CIFAR-10

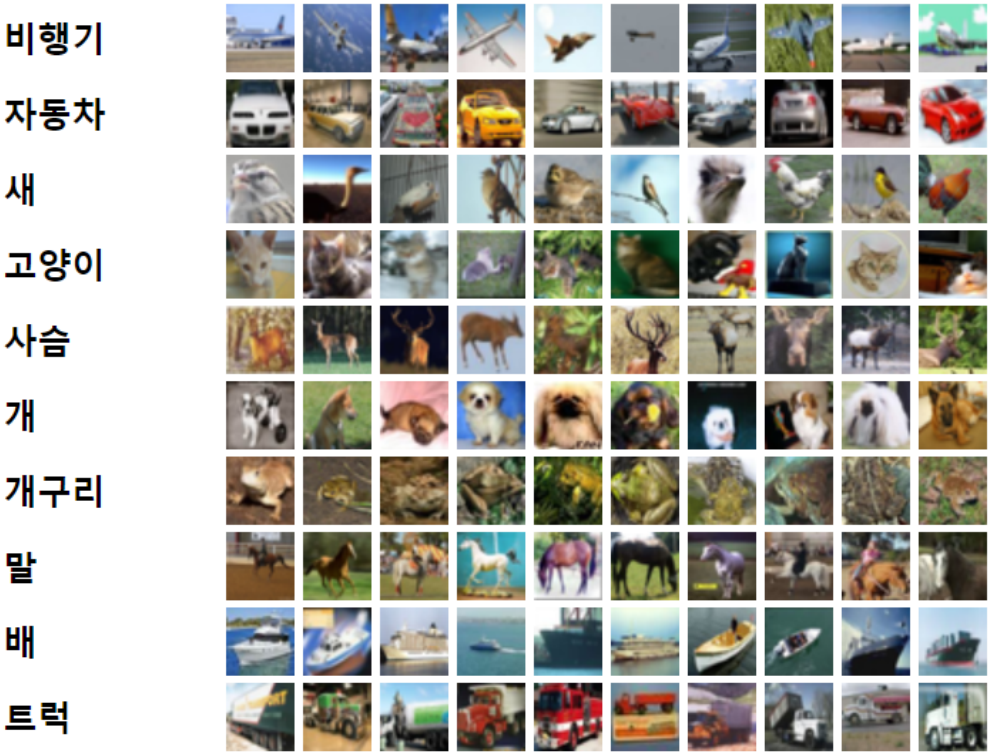
Init method	maxout	ReLU	tanh	Sigmoid
LSUV	**93.94**	**92.11**	89.28	n/c
OrthoNorm	93.78	91.74	89.48	n/c
OrthoNorm-MSRA scaled	–	91.93	–	n/c
Xavier	91.75	90.63	**89.82**	n/c
MSRA	n/c†	90.91	89.54	n/c

n/c symbol stands for "failed to converge, Architecture FitNets-17

Maxout > ELU, Leaky ReLU >= ReLU > tanh > sigmoid

"ALL YOU NEED IS A GOOD INIT", ICLR2016.

Appendix: CIFAR-10



- 32x32픽셀의 60000개 이미지
- 각 이미지는 10개의 클래스로 라벨링

Weight Initialization

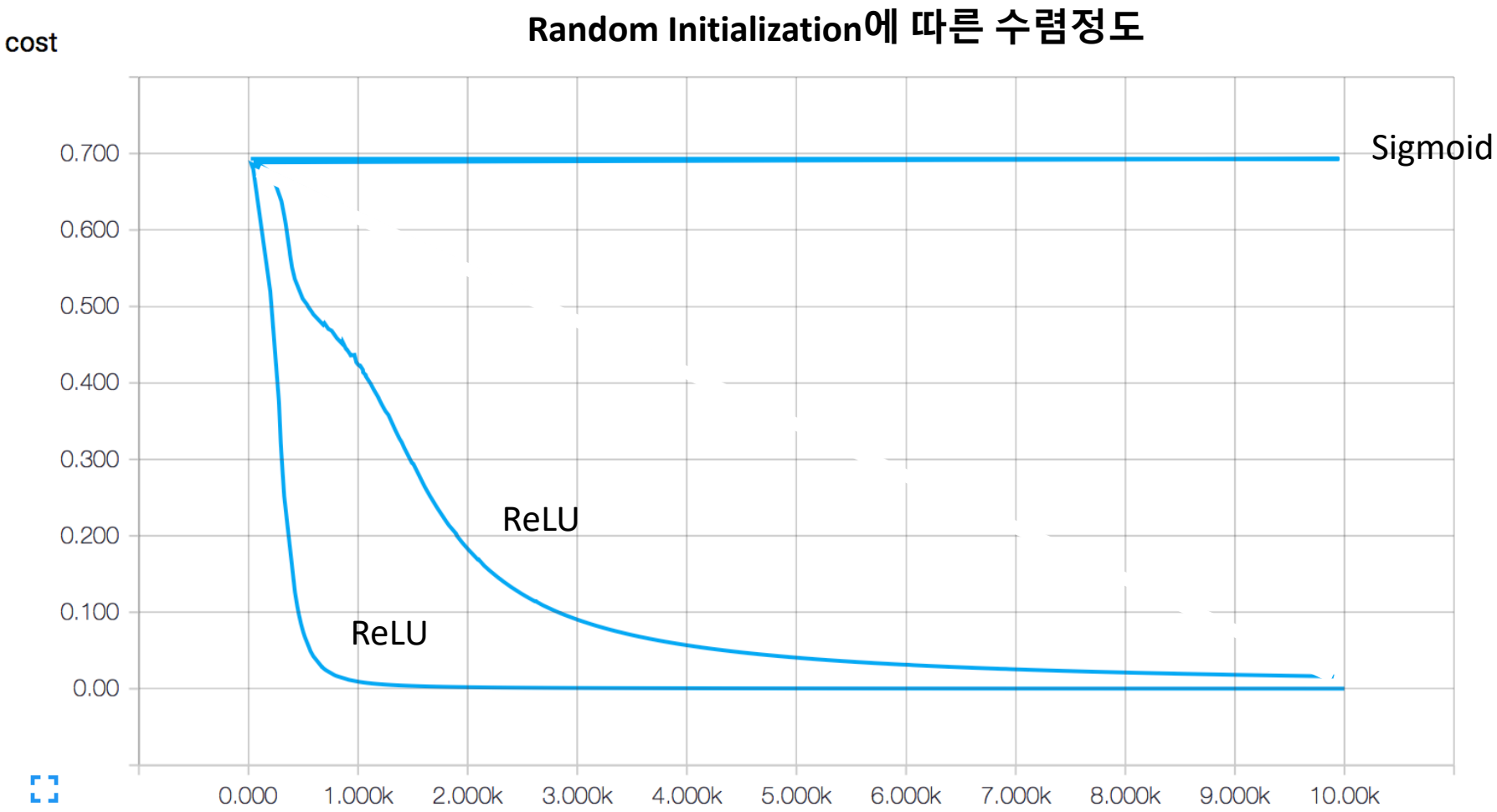
"Initialize weights in a smart way”

Geoffrey Hinton's summary of findings up to today

➔ DNN에서 Gradient Vanishing 문제 해결법

- Our labeled datasets were thousands of times too small
- Our computers were millions of times too slow
- We initialized the weights in a stupid way
- We used the wrong type of non-linearity

초기값에 따른 결과의 차이



Until Now

- 기본적인 선형 회귀나 Softmax 같은 알고리즘에서는 -1 ~ 1의 난 수를 Weight로 사용
- Neural Network에서는 weight 선정에 주의 요망
- W=0이면 Backpropagation 시 gradient 값이 0되어 Gradient Vanishing 현상이 발생

Need to set the initial weight values wisely

- 절대 모두 0으로 초기화 하지 말 것
- 가중치를 어떻게 초기화 할 것이냐는 무척 도전적인 이슈
- Hinton et al. (2006) "A Fast Learning Algorithm for Deep Belief Nets"
- Restricted Boatman Machine (RBM)을 이용한 초기화 제안

Good news

- No need to use complicated RBM for weight initializations

- **Simple methods are OK**
  - 노드의 입출력 수에 비례해서 초기값을 결정짓는 방법 제안

  - **Xavier initialization**
  - X. Glorot and Y. Bengio "Understanding the difficulty of training deep feedforward neural networks," in International conference on artificial intelligence and statistics, 2010

  - **He's initialization**
  - **K. He**, X. Zhang, S. Ren, and J. Sun "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," 2015

Xavier/He initialization

- Makes sure the weights are 'just right', not too small, not too big
- Using number of input (fan_in) and output (fan_out)

```
# 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)
```

Pytorch 초기화 방법 ➔ https://pytorch.org/docs/stable/nn.init.html

Xavier initialization

`torch.nn.init.xavier_normal_(`*`tensor`*`, `*`gain=1.0`*`)` [SOURCE]

Fills the input *Tensor* with values according to the method described in *Understanding the difficulty of training deep feedforward neural networks* - Glorot, X. & Bengio, Y. (2010), using a normal distribution. The resulting tensor will have values sampled from $\mathcal{N}(0, \text{std}^2)$ where

$$\text{std} = \text{gain} \times \sqrt{\frac{2}{\text{fan\_in} + \text{fan\_out}}}$$

Also known as Glorot initialization.

Parameters

- **tensor** – an n-dimensional *torch.Tensor*
- **gain** – an optional scaling factor

Examples

```
>>> w = torch.empty(3, 5)
>>> nn.init.xavier_normal_(w)
```

```
[docs]def xavier_normal_(tensor, gain=1.):
    # type: (Tensor, float) -> Tensor
    r"""Fills the input `Tensor` with values according to the method
    described in `Understanding the difficulty of training deep feedforward
    neural networks` - Glorot, X. & Bengio, Y. (2010), using a normal
    distribution. The resulting tensor will have values sampled from
    :math:`\mathcal{N}(0, \text{std}^2)` where

    .. math::
        \text{std} = \text{gain} \times \sqrt{\frac{2}{\text{fan\_in} + \text{fan\_out}}}

    Also known as Glorot initialization.

    Args:
        tensor: an n-dimensional `torch.Tensor`
        gain: an optional scaling factor

    Examples:
        >>> w = torch.empty(3, 5)
        >>> nn.init.xavier_normal_(w)
    """
    fan_in, fan_out = _calculate_fan_in_and_fan_out(tensor)
    std = gain * math.sqrt(2.0 / float(fan_in + fan_out))

    return _no_grad_normal_(tensor, 0., std)
```

He initialization

`torch.nn.init.kaiming_normal_(`*`tensor`*`, a=0, mode='fan_in', nonlinearity='leaky_relu')` [SOURCE]

Fills the input *Tensor* with values according to the method described in *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification* - He, K. et al. (2015), using a normal distribution. The resulting tensor will have values sampled from $\mathcal{N}(0, \text{std}^2)$ where

$$\text{std} = \frac{\text{gain}}{\sqrt{\text{fan\_mode}}}$$

Also known as He initialization.

Parameters

- **tensor** – an n-dimensional *torch.Tensor*
- **a** – the negative slope of the rectifier used after this layer (only
- **with 'leaky_relu')** (*used*) –
- **mode** – either `'fan_in'` (default) or `'fan_out'`. Choosing `'fan_in'` preserves the magnitude of the variance of the weights in the forward pass. Choosing `'fan_out'` preserves the magnitudes in the backwards pass.
- **nonlinearity** – the non-linear function (*nn.functional* name), recommended to use only with `'relu'` or `'leaky_relu'` (default).

```
[docs]def kaiming_normal_(tensor, a=0, mode='fan_in', nonlinearity='leaky_relu'):
    r"""Fills the input `Tensor` with values according to the method
    described in `Delving deep into rectifiers: Surpassing human-level
    performance on ImageNet classification` - He, K. et al. (2015), using a
    normal distribution. The resulting tensor will have values sampled from
    :math:`\mathcal{N}(0, \text{std}^2)` where

    .. math::
        \text{std} = \frac{\text{gain}}{\sqrt{\text{fan\_mode}}}

    Also known as He initialization.

    Args:
        tensor: an n-dimensional `torch.Tensor`
        a: the negative slope of the rectifier used after this layer (only
        used with ``'leaky_relu'``)
        mode: either ``'fan_in'`` (default) or ``'fan_out'``. Choosing ``'fan_in'``
            preserves the magnitude of the variance of the weights in the
            forward pass. Choosing ``'fan_out'`` preserves the magnitudes in the
            backwards pass.
        nonlinearity: the non-linear function (`nn.functional` name),
            recommended to use only with ``'relu'`` or ``'leaky_relu'`` (default).

    Examples:
        >>> w = torch.empty(3, 5)
        >>> nn.init.kaiming_normal_(w, mode='fan_out', nonlinearity='relu')
    """
    fan = _calculate_correct_fan(tensor, mode)
    gain = calculate_gain(nonlinearity, a)
    std = gain / math.sqrt(fan)
    with torch.no_grad():
        return tensor.normal_(0, std)
```

Activation Function: Conclusion

The compatibility of activation functions and initialization.
Dataset: CIFAR-10

Init method	maxout	ReLU	VLReLU	tanh	Sigmoid
LSUV	**93.94**	**92.11**	92.97	89.28	n/c
OrthoNorm	93.78	91.74	92.40	89.48	n/c
OrthoNorm-MSRA scaled	–	91.93	**93.09**	–	n/c
Xavier	91.75	90.63	92.27	**89.82**	n/c
MSRA	n/c†	90.91	92.43	89.54	n/c

"ALL YOU NEED IS A GOOD INIT", ICLR2016.

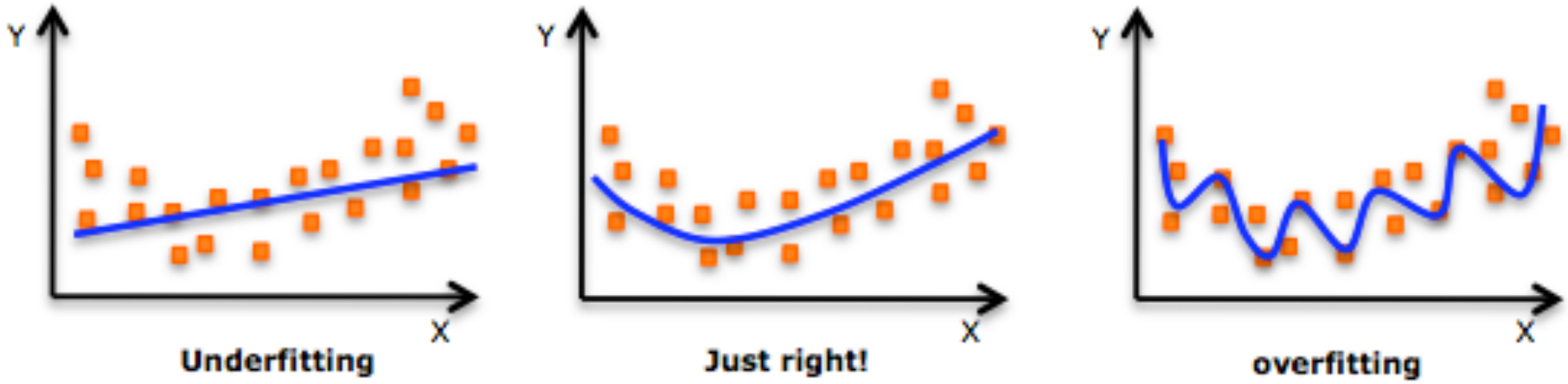
Still an active area of research

- **We don't know how to initialize perfect weight values, yet**
- Many new algorithms
  - Batch normalization
  - Layer sequential uniform variance
  - …

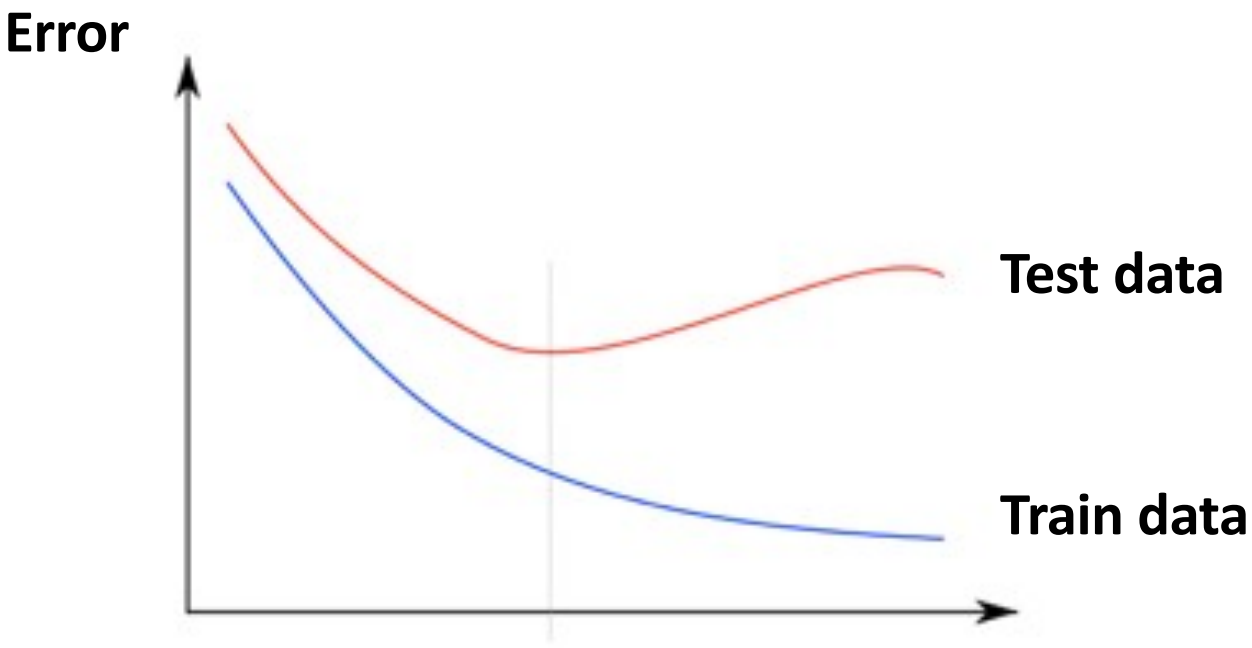
Dropout and model Ensemble

[복습] Overfitting

- ❑ 너무 과도하게 데이터에 대해 모델을 learning을 한 경우를 의미함
- ❑ 현 학습 데이터만 잘 표현하면, 새로운 데이터에 대한 대응력이 없어 모델 학습 의미 상실



[복습] Am I overfitting?



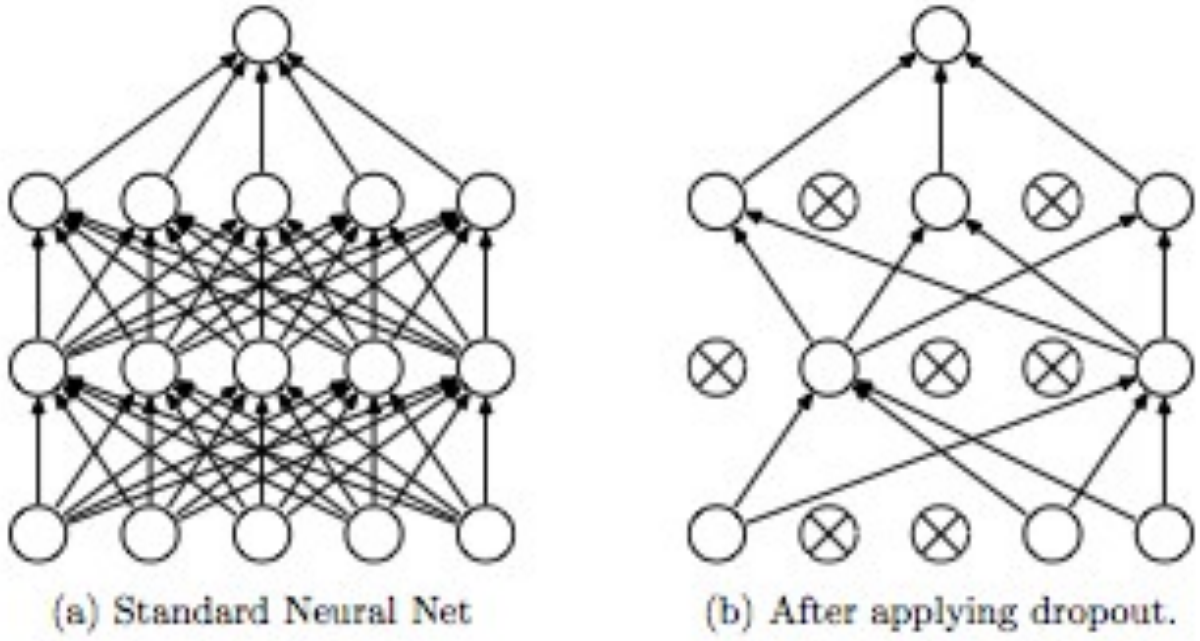
✓ Very high accuracy on the training dataset (0.99)
✓ Poor accuracy on the test data set (0.85)

[복습] Solution for overfitting

- More training data!
- Reduce the number of features
- **Regularization (−Dropout)**

Dropout for overfitting

- 훈련 데이터에 대한 복잡한 공동 적응을 방지하여 신경망의 과적 합을 줄이기 위한 Google이 제안한 정규화 기술 (regularization)
- "드롭 아웃"이라는 용어는 신경망에서 유닛을 제거하는 것
- 학습 시에만 적용하고 테스트 시에는 모든 유닛을 사용함



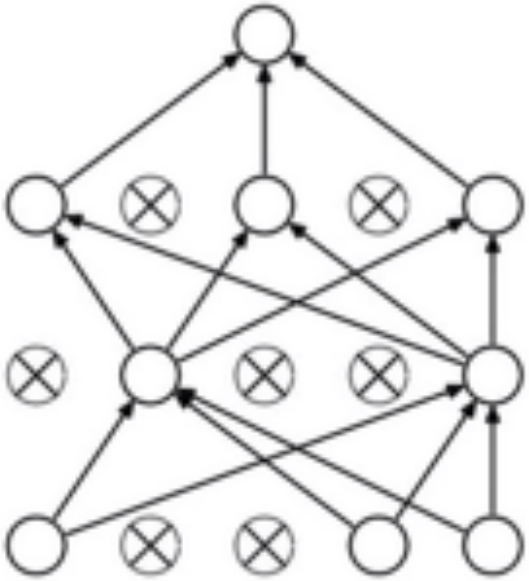
(a) Standard Neural Net

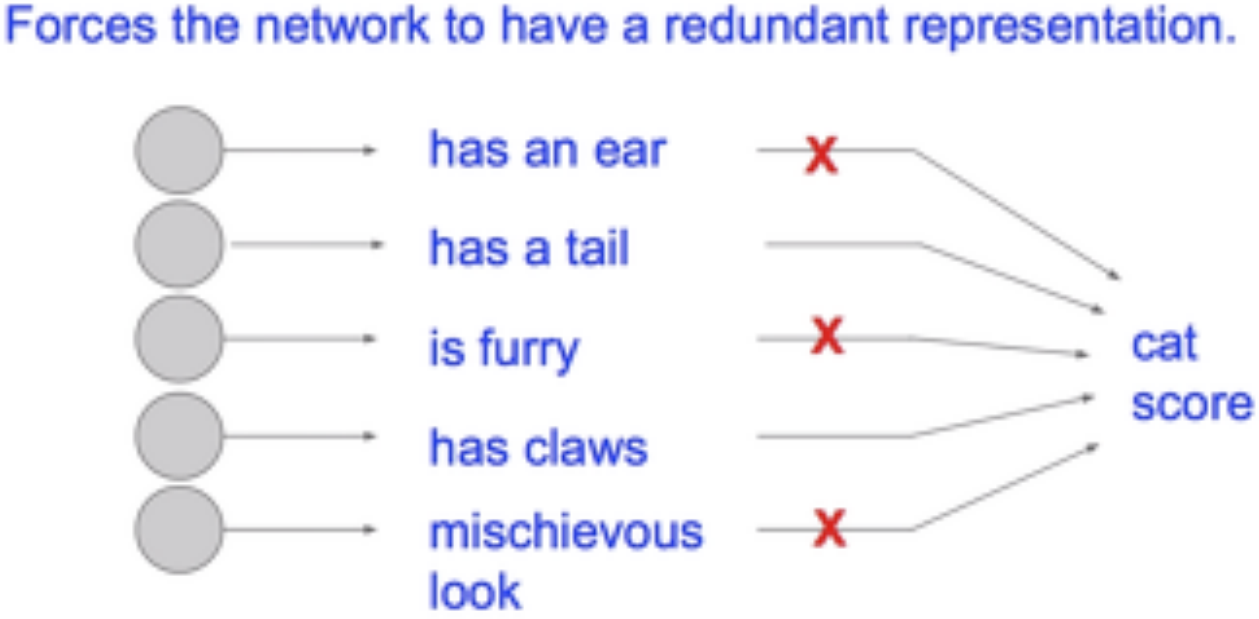
(b) After applying dropout.

Dropout: A SimpleWay to Prevent Neural Networks from Overfitting

Dropout for overfitting

- 왜 성능향상을 가져오는가?
  - 드랍아웃을 통해 Ensemble model 학습과 같은 효과가 있기 때문이다.
  - Ensemble model이란 집단지성으로 이해할 수 있다.





Dropout

CLASS `torch.nn.Dropout(p=0.5, inplace=False)` [SOURCE]

During training, randomly zeroes some of the elements of the input tensor with probability `p` using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons as described in the paper Improving neural networks by preventing co-adaptation of feature detectors .

Furthermore, the outputs are scaled by a factor of $\frac{1}{1-p}$ during training. This means that during evaluation the module simply computes an identity function.

Parameters

- **p** – probability of an element to be zeroed. Default: 0.5
- **inplace** – If set to `True`, will do this operation in-place. Default: `False`

Shape:

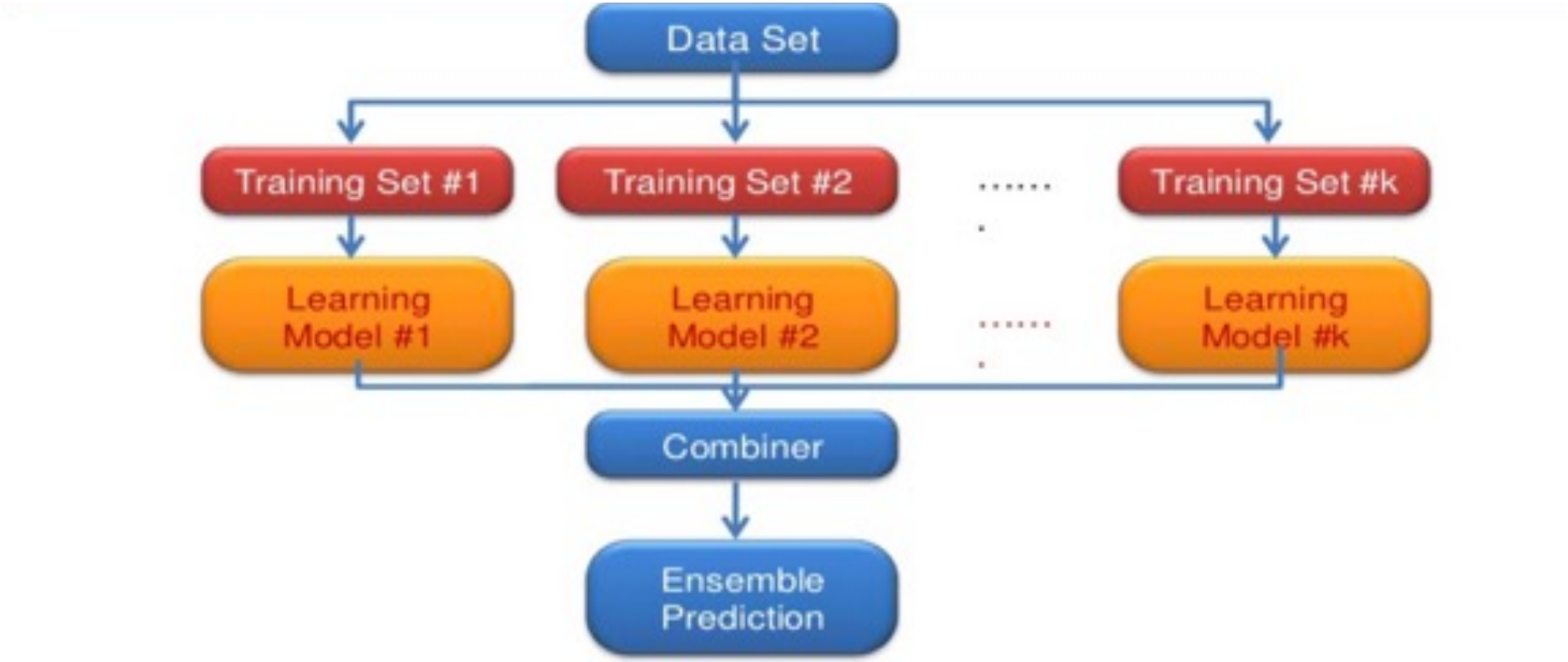
- Input: $(*)$. Input can be of any shape
- Output: $(*)$. Output is of the same shape as input

Examples:

```
>>> m = nn.Dropout(p=0.2)
>>> input = torch.randn(20, 16)
>>> output = m(input)
```

Ensemble

- (통계학과 기계 학습에서) 앙상블 학습법은 학습 알고리즘들을 따로 쓰는 경우에 비해 더 좋은 예측 성능을 얻기 위해 다수의 학습 알고리즘을 사용하는 방법을 말함



Summary

- **DNN 모델 학습을 위한 팁**
  - 활성 함수를 잘 선택한다
    - ReLU가 가장 널리 사용된다
  - 가중치 초기화 방법을 잘 선택한다
    - Xavier가 가장 널리 사용된다
  - 드랍 아웃을 잘 적용한다
    - “NN-ReLU-Dropout”을 하나의 블락으로 쌓는다
  - BN을 잘 적용한다
    - “NN-ReLU-BN”을 하나의 블락으로 쌓는다

END