DSCI 572: Supervised Learning II

Muhammad Abdul-Mageed

muhammad.mageed@ubc.ca

Natural Language Processing Lab

The University of British Columbia

Bias & Variance

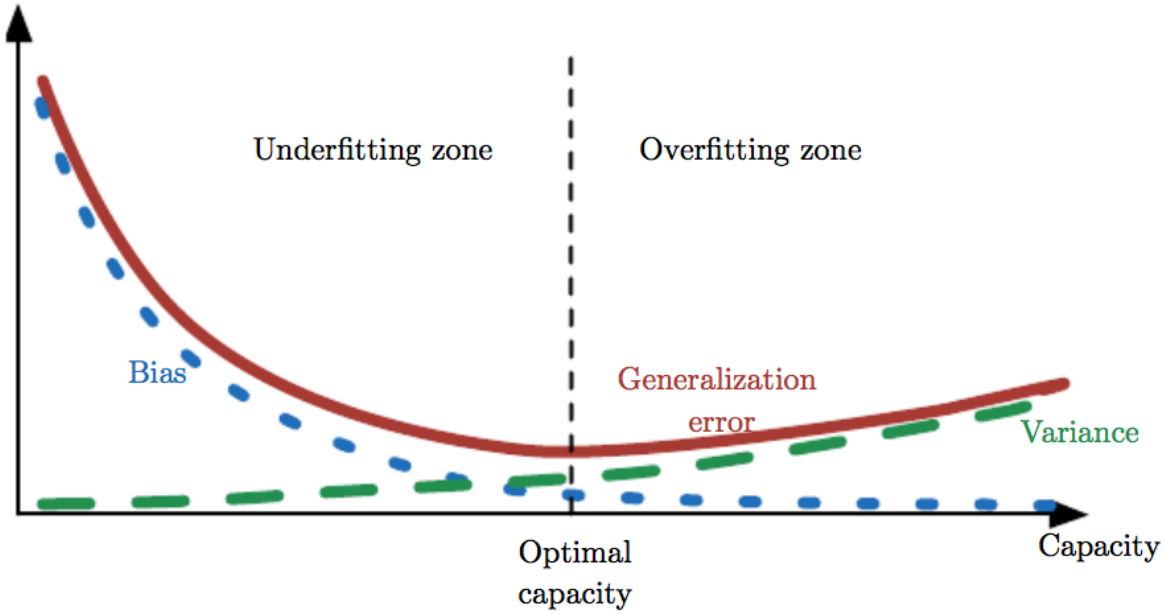


Figure 5.6: As capacity increases (x-axis), bias (dotted) tends to decrease and variance (dashed) tends to increase, yielding another U-shaped curve for generalization error (bold curve). If we vary capacity along one axis, there is an optimal capacity, with underfitting when the capacity is below this optimum and overfitting when it is above. This relationship is similar to the relationship between capacity, underfitting, and overfitting, discussed in section 5.2 and figure 5.3.

Figure: Goodfellow et al. (2016)

Regularization

Limiting Model Capacity

- Many regularization approaches based on limiting model capacity by **adding a parameter norm penalty** $\Omega(\theta)$ to the **objective function J**

$$\tilde{J}(\theta, X, y) = J(\theta, X, y) + \Omega(\theta) \tag{1}$$

- In DL, **we typically regularize only the weights (w), but not the biases**
- Each bias controls only a single variable and so **we don't induce too much variance by not regularizing bias**. Hence:

$$\tilde{J}(w, X, y) = J(w, X, y) + \Omega(w) \tag{2}$$

Parameter Regularization

Vector Norm

- Given a vector space V, a norm is a function $p : V \to \mathbb{R}$ (thus the function gives a real-valued length to the vector) s.t.
  - $\|x\| \geq 0$. ($\|x\| = 0$ only if $x = 0$)
  - $\|x + y\| \leq \|x\| + \|y\|$ (triangle inequality)
  - $\|\alpha x\| = |\alpha| \, \|x\|$. (That is, if we scale a vector by α, then what we get is the absolute value of α times the length of the vector.)

p-Norms

For a vector x

- $\|x\|_p = \left(\sum_{i=1}^{m} .|x_i|^p\right)^{\frac{1}{p}}$

Important Norms

- **(p=1)** $\|x\|_1 = \sum_{i=1}^{m} .|x_i|$ (taxicab/Manhattan norm. All elements weighted equally)
- **(p=2)** $\|x\|_2 = \left(\sum_{i=1}^{m} .|x_i|^2\right)^{\frac{1}{2}}$ (Euclidean length. larger elements have more weight)
- **(p=∞)** $\|x\|_\infty = \max_{1\leq i\leq m} |x_i|$ (magnitude of largest element. Only biggest element has any effect)
- L1 introduces sparsity, hence less computational cost & better storage.
- L2 is known as weight decay.

Norm in Pytorch

```
c = torch.tensor([[ 1, 2, 3],[-1, 1, 4]] , dtype= torch.float)
```

```
torch.norm(c, dim=0) #torch.norm(input, p='fro', dim=None, keepdim=False, out=None, dtype=None)
# default norm is `fro' --> "Frobenius norm"
```

```
tensor([1.4142, 2.2361, 5.0000])
```

```
torch.norm(c, dim=0, p=2)
```

```
tensor([1.4142, 2.2361, 5.0000])
```

```
torch.norm(c, dim=0, p=1)
```

```
tensor([2., 3., 7.])
```

Figure: Pytorch code. See `https://pytorch.org/docs/stable/torch.html`.

L2 Penalty / Weight Decay

CLASS `torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)` [SOURCE]

Implements Adam algorithm.

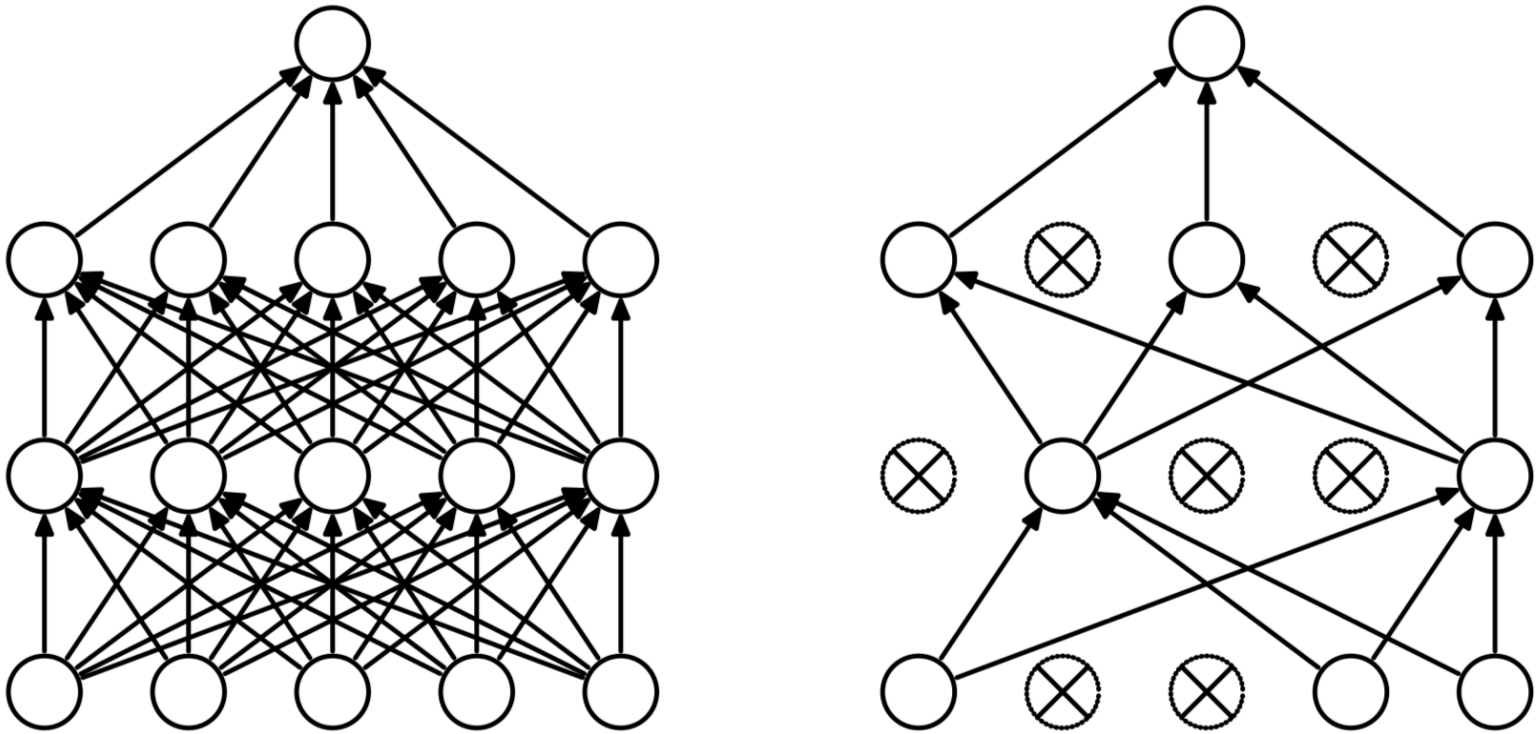
It has been proposed in Adam: A Method for Stochastic Optimization.

Parameters

- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*python:float, optional*) – learning rate (default: 1e-3)
- **betas** (*Tuple[python:float, python:float], optional*) – coefficients used for computing running averages of gradient and its square (default: (0.9, 0.999))
- **eps** (*python:float, optional*) – term added to the denominator to improve numerical stability (default: 1e-8)
- **weight_decay** (*python:float, optional*) – weight decay (L2 penalty) (default: 0)
- **amsgrad** (*boolean, optional*) – whether to use the AMSGrad variant of this algorithm from the paper On the Convergence of Adam and Beyond (default: False)

Figure: Pytorch code. See `https://pytorch.org/docs/stable/optim.html`.

Dropout

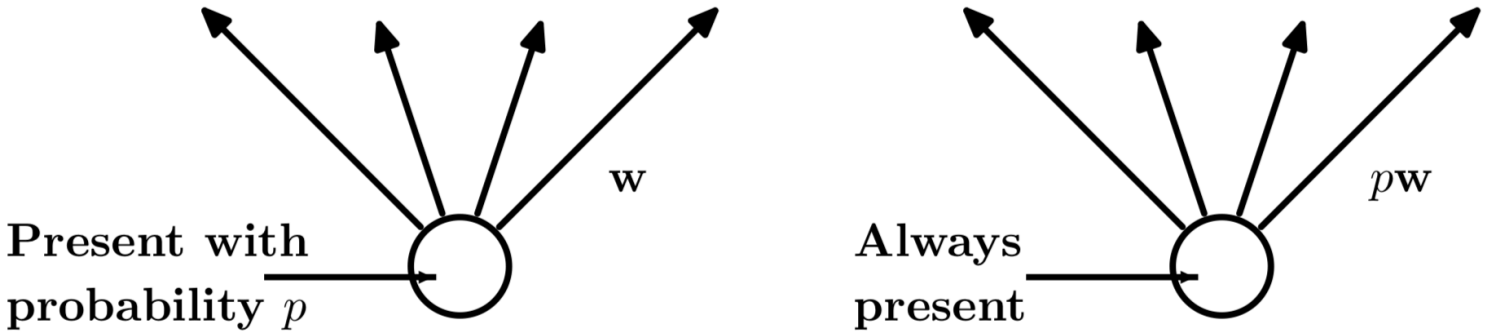


(a) Standard Neural Net

(b) After applying dropout.

Figure: Srivastava et al. (2014). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting".

Dropout



(a) At training time (b) At test time

Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights **w**. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Figure: Srivastava et al. (2014).

Data Augmentation

Augmentation

- How do we do that for image data, for example?
- How about text?
- Speech?

Semi-Supervised Learning

SSL

- Examples of SSL
- Self-training as an example
- Why does it help?

Multi-Task Learning

MTL

- Analogy to human learning
- Why does it help?
- How does it work in DL?