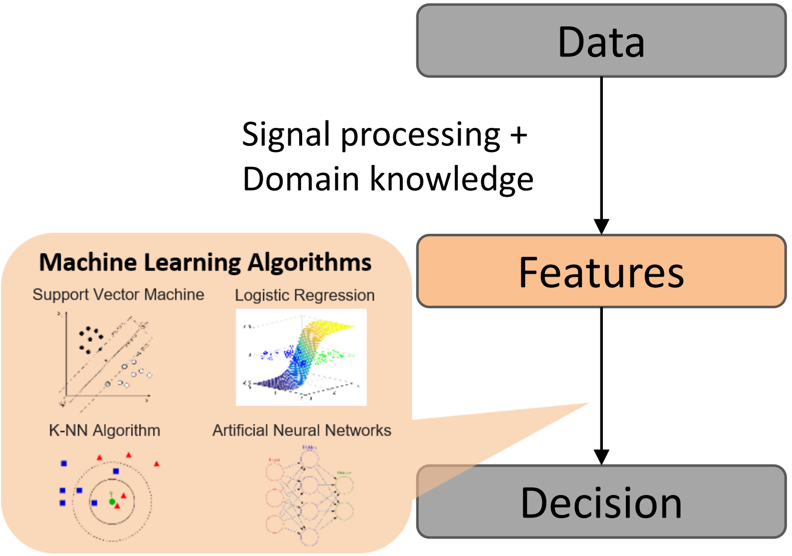
Deep Neural Networks: Deep Learning

Industrial AI Lab.

Prof. Seungchul Lee

Yunseob Hwang, Illjeok Kim

Machine Learning

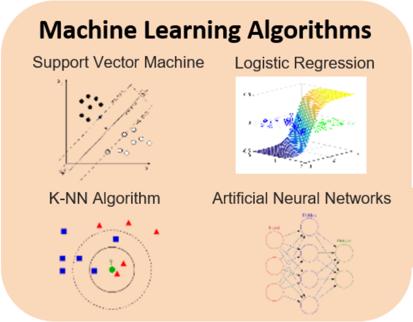


Machine Learning

Data

Signal processing +
Domain knowledge

Features



Decision

Deep Learning

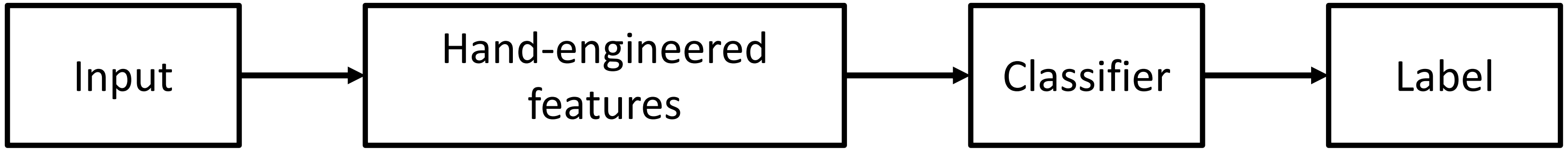
Data

Less depends on
domain knowledge

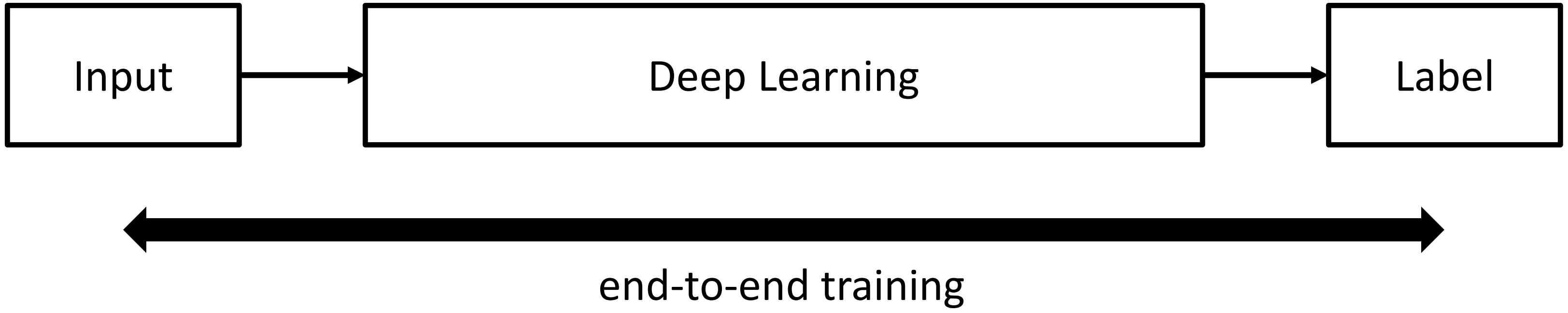
Decision

Machine Learning and Deep Learning

- Machine Learning



- Deep supervised learning

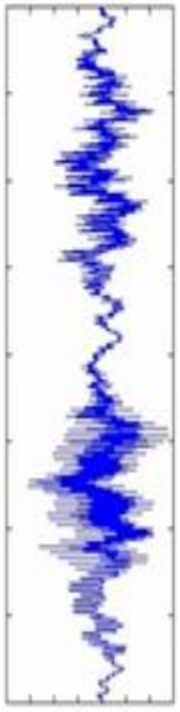


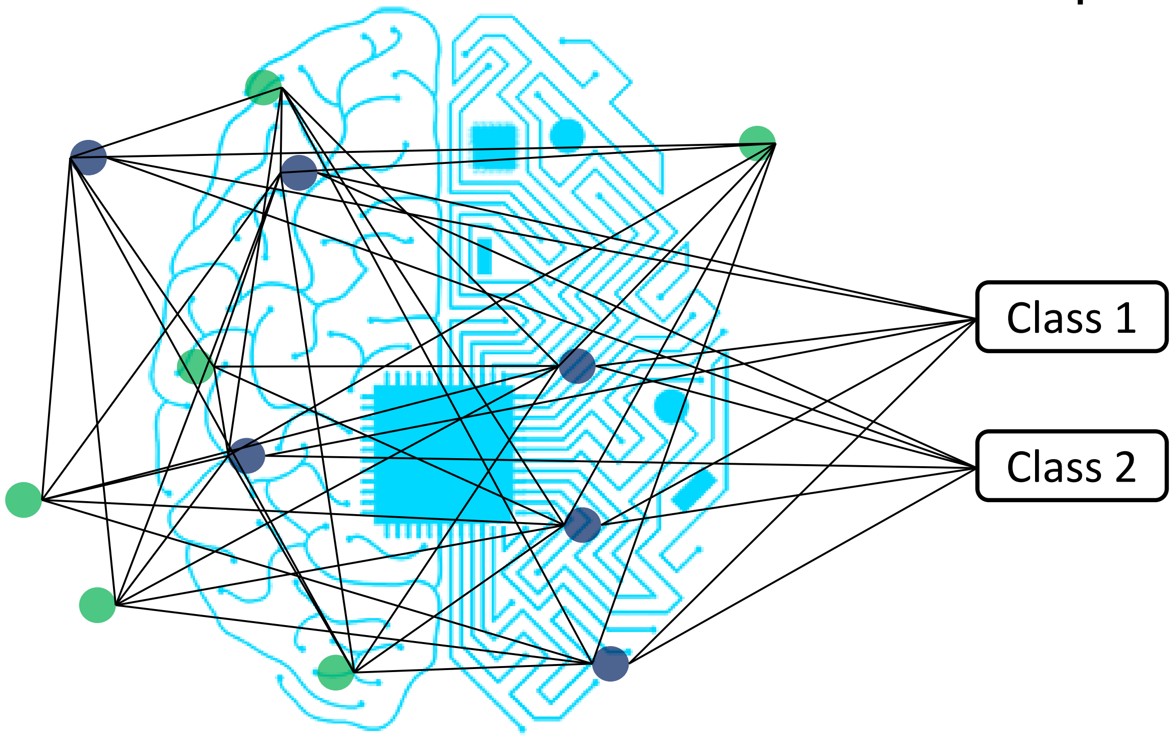
Deep Artificial Neural Networks

- Complex/Nonlinear universal function approximator
  - Linearly connected networks
  - Simple nonlinear neurons

Input

Output

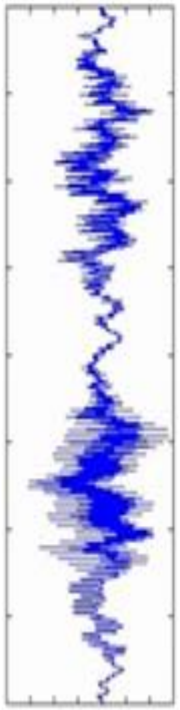




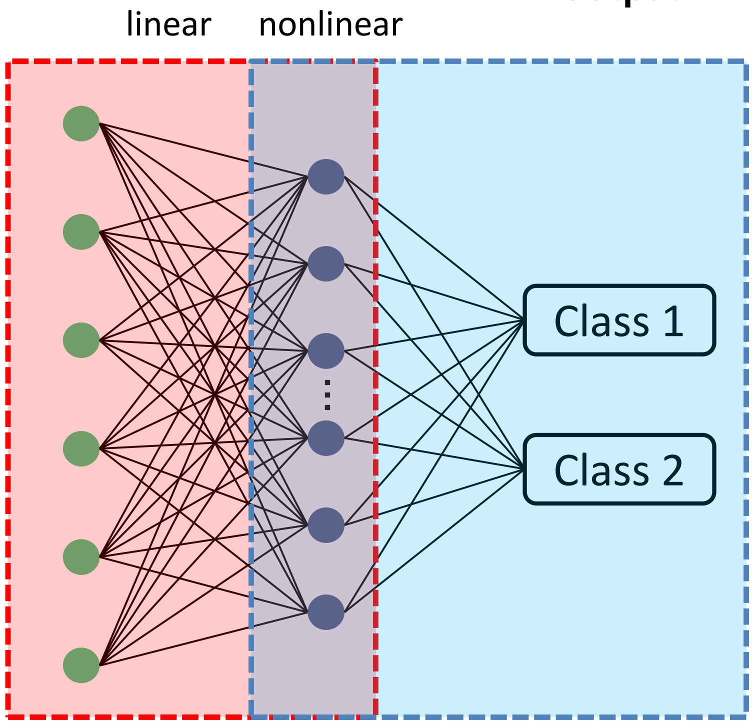
Deep Artificial Neural Networks

- Complex/Nonlinear universal function approximator
  - Linearly connected networks
  - Simple nonlinear neurons

Input

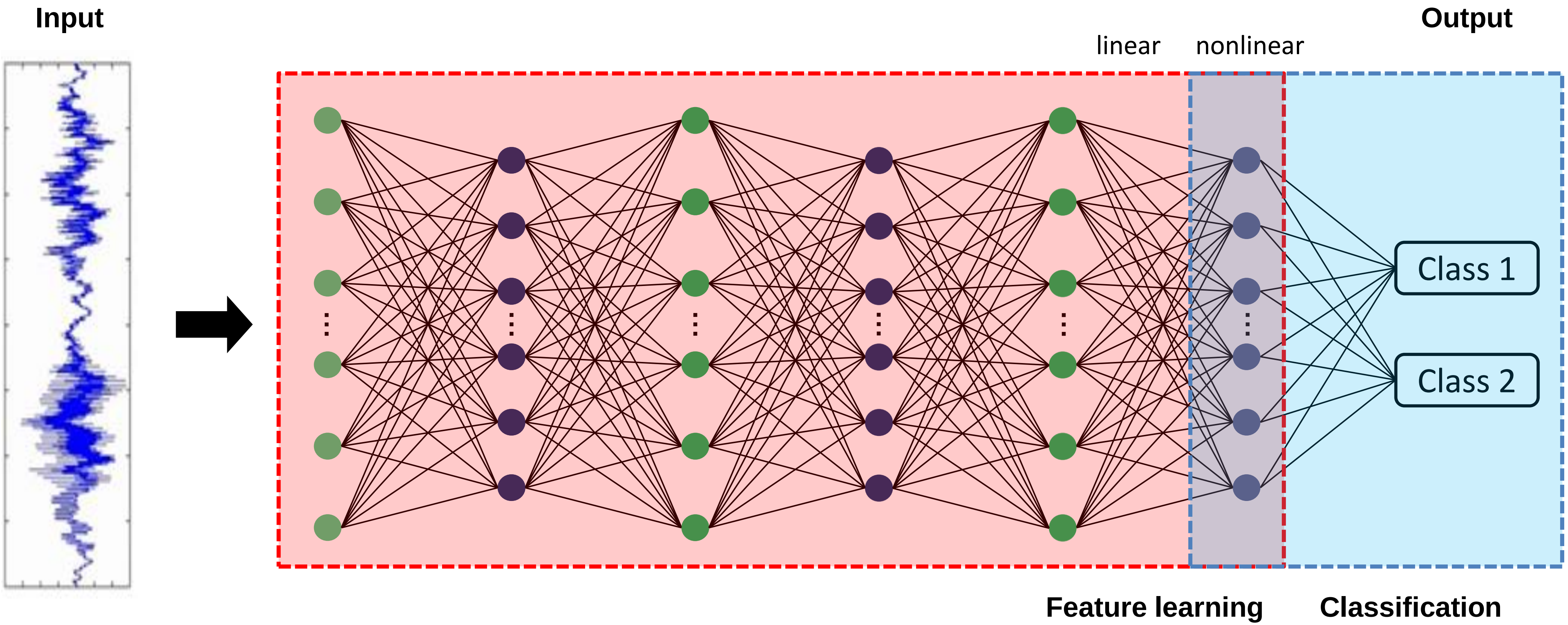


Output



Deep Artificial Neural Networks

- Complex/Nonlinear universal function approximator
  - Linearly connected networks
  - Simple nonlinear neurons



Training NN: Backpropagation

Training Neural Networks: Optimization

- Learning or estimating weights and biases of multi-layer perceptron from training data

- 3 key components
  - objective function $f(\cdot)$
  - decision variable or unknown ω
  - constraints $g(\cdot)$

- In mathematical expression

$$\min_{\omega} \quad f(\omega)$$

Training Neural Networks: Loss Function

- Measures error between target values and predictions

$$\min_{\omega} \sum_{i=1}^{m} \ell\left(h_\omega\left(x^{(i)}\right), y^{(i)}\right)$$

- Example
  - Squared loss (for regression):

$$\frac{1}{m} \sum_{i=1}^{m} \left(h_\omega\left(x^{(i)}\right) - y^{(i)}\right)^2$$

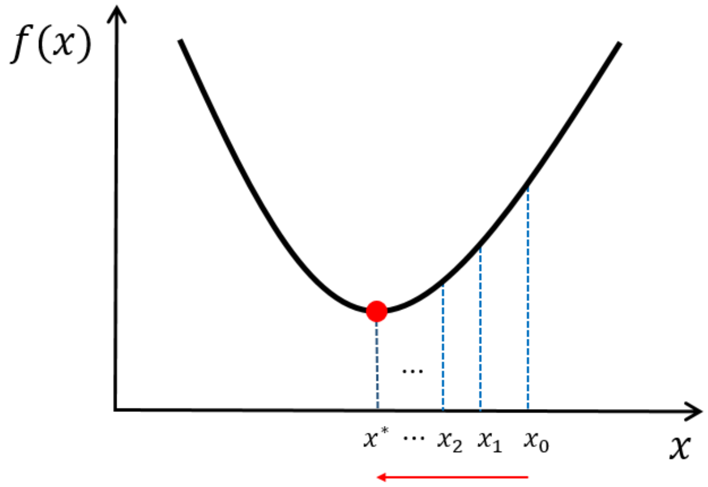
  - Cross entropy (for classification):

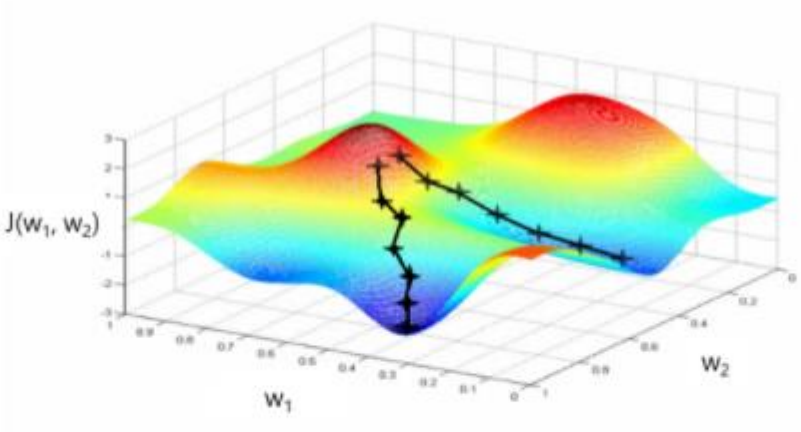
$$-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log\left(h_\omega\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\omega\left(x^{(i)}\right)\right)$$

Training Neural Networks: Gradient Descent

- Negative gradients points directly downhill of the cost function
- We can decrease the cost by moving in the direction of the negative gradient (α is a learning rate)

$$\omega \Leftarrow \omega - \alpha \nabla_{\omega} \ell \left(h_{\omega} \left(x^{(i)} \right), y^{(i)} \right)$$





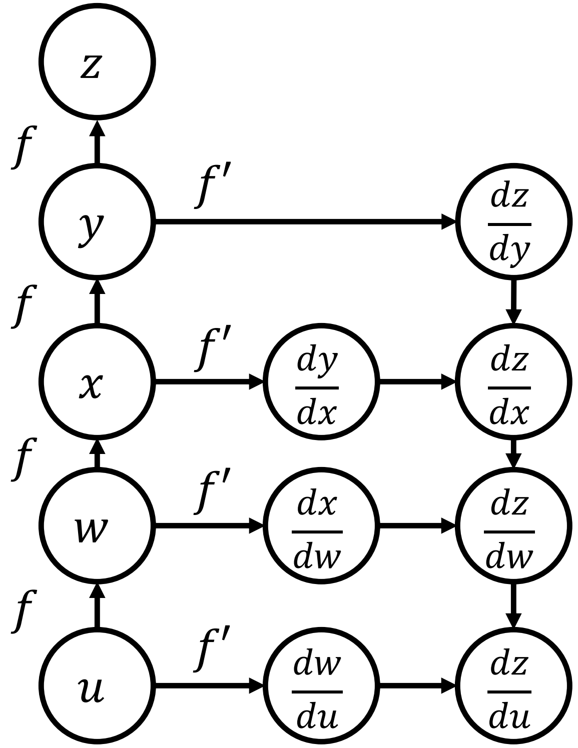
Training Neural Networks: Backpropagation Learning

- Forward propagation
  - the initial information propagates up to the hidden units at each layer and finally produces output

- Backpropagation
  - allows the information from the cost to flow backwards through the network in order to compute the gradients

Backpropagation

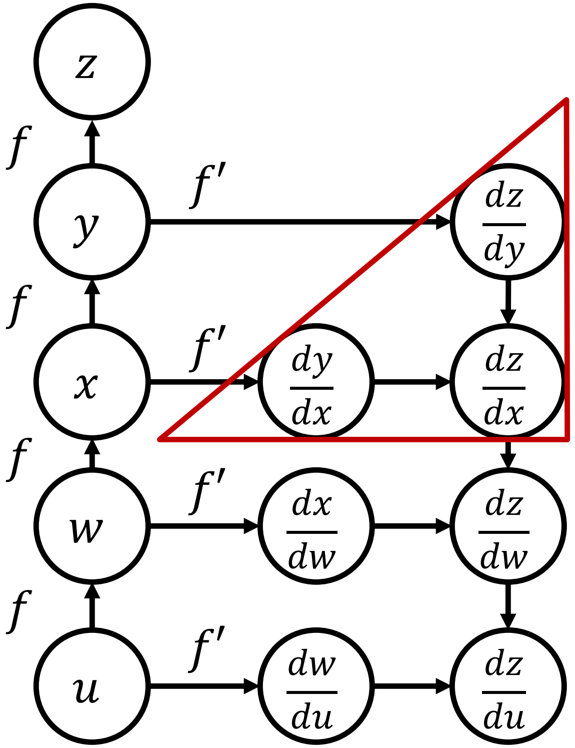
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$



- Backpropagation
  - Update weights recursively

Backpropagation

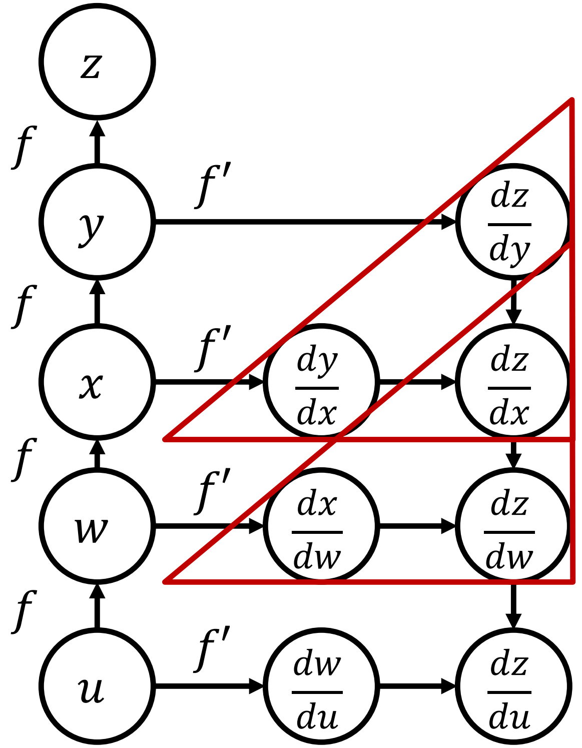
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\left(g(x)\right)' = f'\left(g(x)\right)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$



- Backpropagation
  - Update weights recursively

Backpropagation

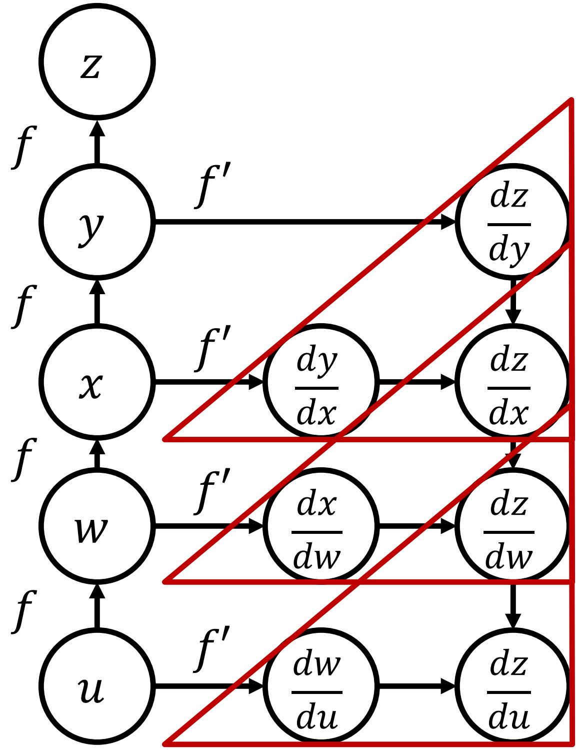
- Chain Rule
  - Computing the derivative of the composition of functions
    - $f\big(g(x)\big)' = f'\big(g(x)\big)g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$



- Backpropagation
  - Update weights recursively

Backpropagation

- Chain Rule
  - Computing the derivative of the composition of functions
    - $f(g(x))' = f'(g(x))g'(x)$
    - $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
    - $\frac{dz}{dw} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx}\right) \cdot \frac{dx}{dw}$
    - $\frac{dz}{du} = \left(\frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw}\right) \cdot \frac{dw}{du}$

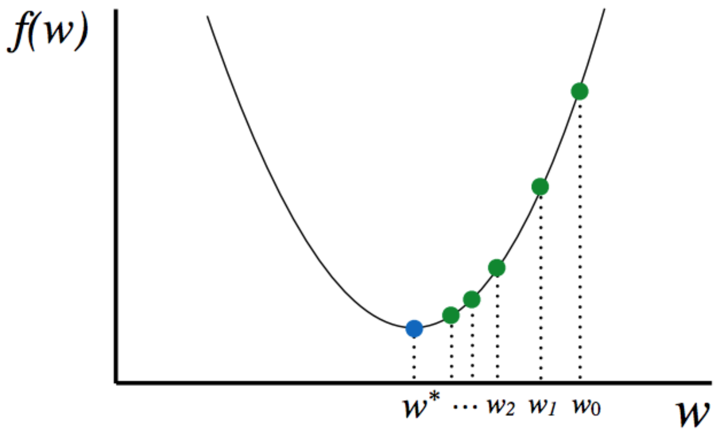


- Backpropagation
  - Update weights recursively with memory

Training Neural Networks with TensorFlow

- Optimization procedure

Start at a random point
Repeat
    Determine a descent direction
    Choose a step size
    Update
Until stopping criterion is satisfied

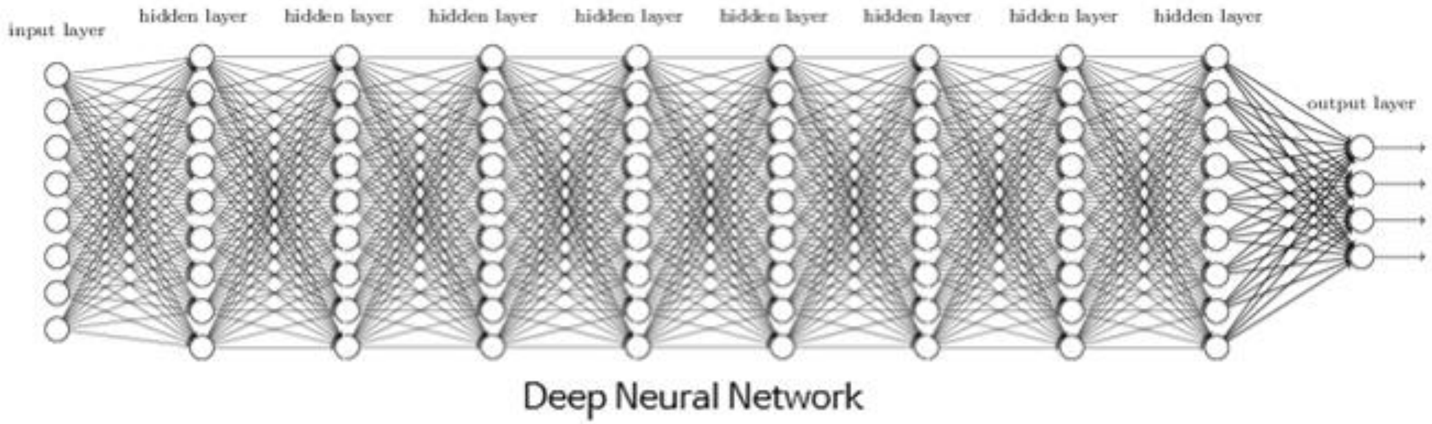


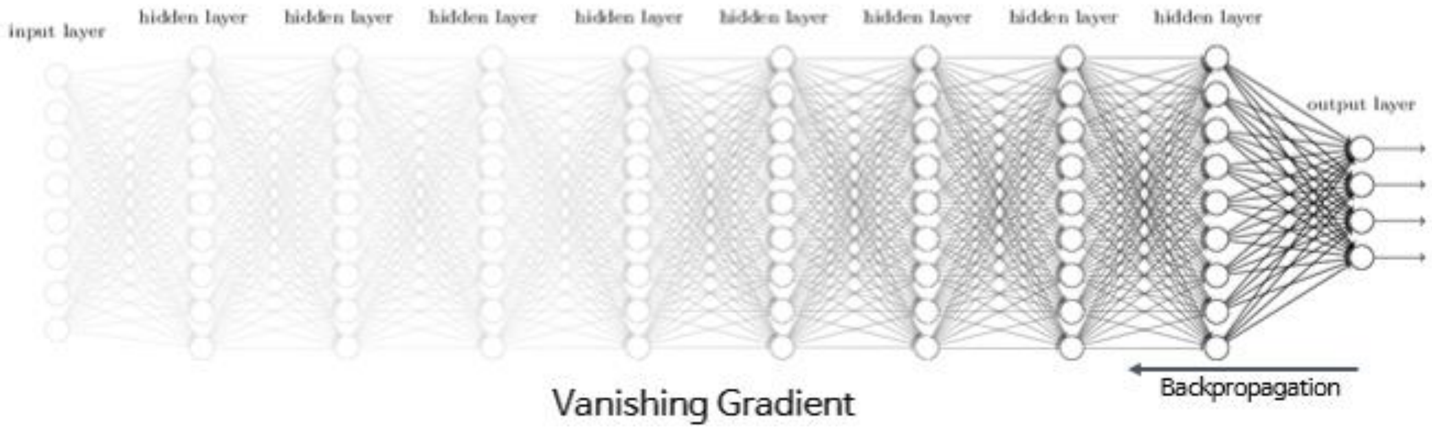
- It is not easy to numerically compute gradients in network in general.
  - The good news: people have already done all the "hard work" of developing numerical solvers (or libraries)
  - There are a wide range of tools → We will use the TensorFlow

Vanishing Gradient

The Vanishing Gradient Problem

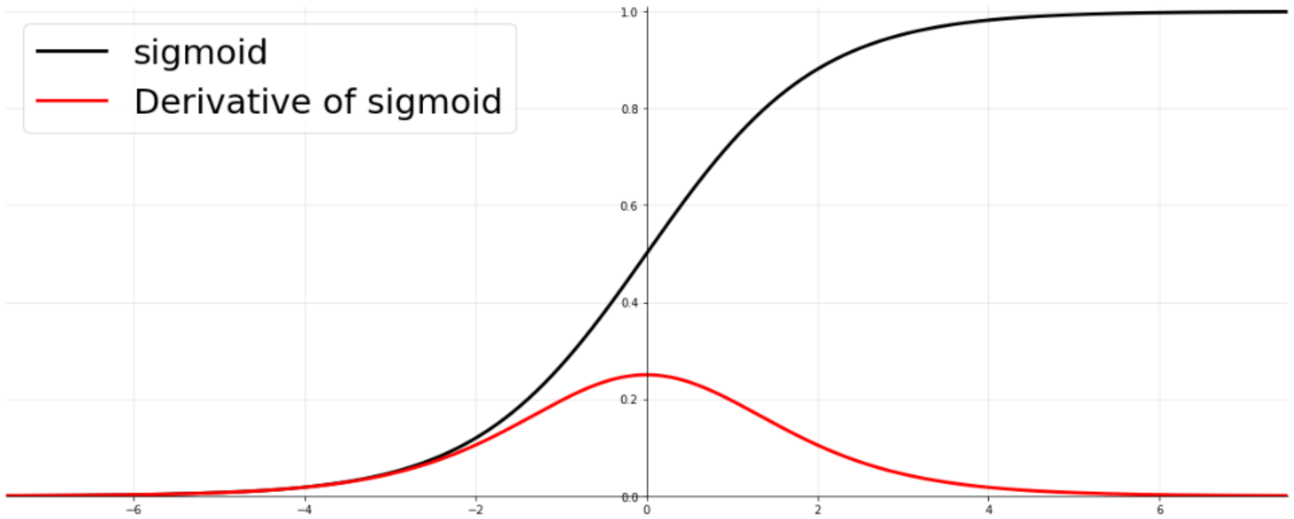
- As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train





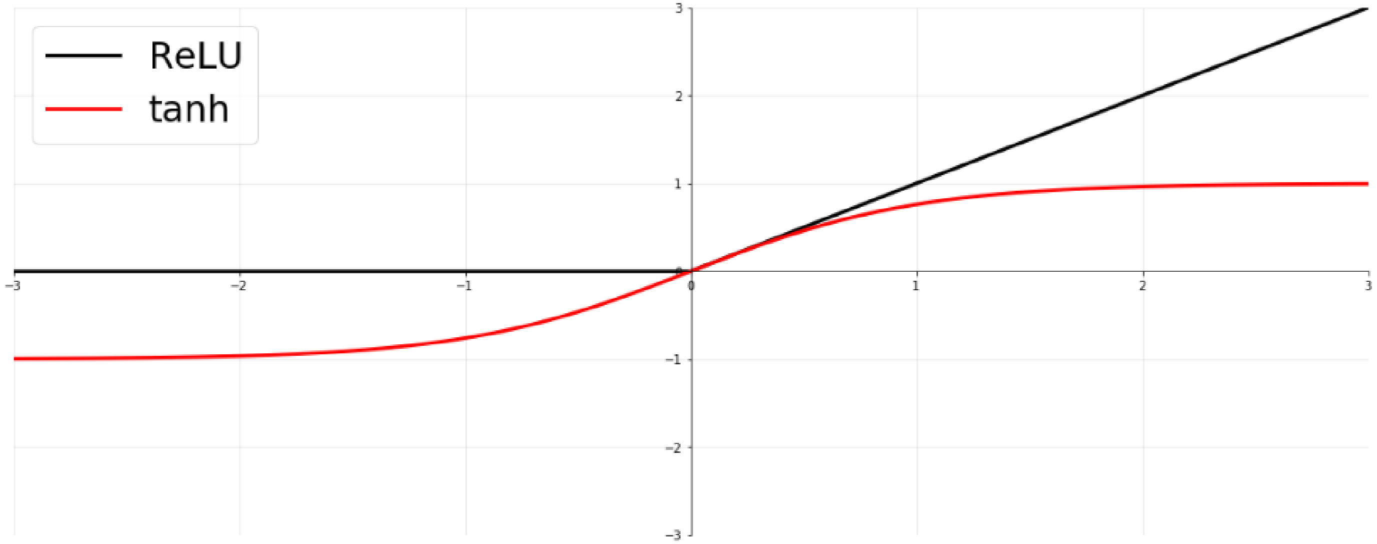
The Vanishing Gradient Problem

- As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train.
- For example,
  - $\frac{dz}{du} = \frac{dz}{dy} \cdot \frac{dy}{dx} \cdot \frac{dx}{dw} \cdot \frac{dw}{du}$



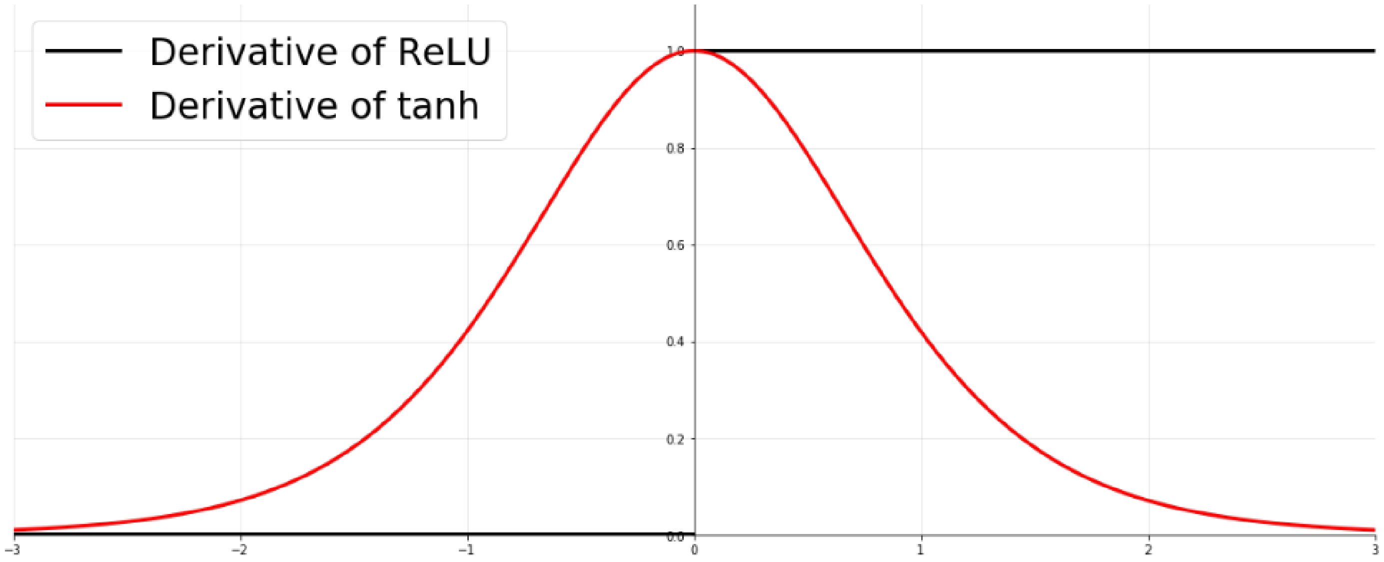
Rectifiers

- The use of the ReLU activation function was a great improvement compared to the historical tanh.



Rectifiers

- This can be explained by the derivative of ReLU itself not vanishing, and by the resulting coding being sparse (Glorot et al., 2011).



Gradient Descent in Deep Learning

Gradient Descent

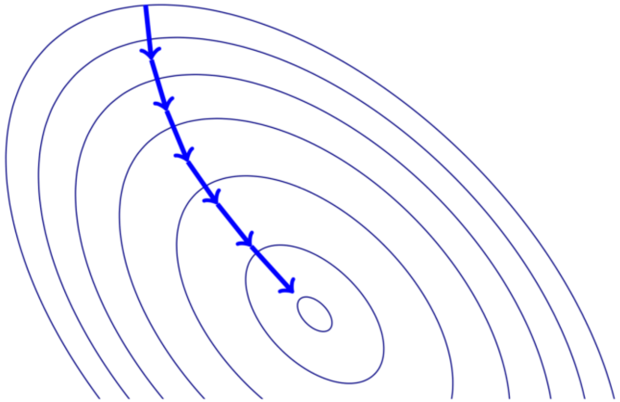
- We will cover gradient descent algorithm and its variants:
  - Batch Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent

- We will explore the concept of these three gradient descent algorithms with a logistic regression model in TensorFlow

- Limitation of the Gradient Descent
  - Adaptive learning rate

Batch Gradient Descent (= Gradient Descent)

Repeat: $\omega \leftarrow \omega - \alpha \nabla f(\omega)$ for some step size (or learning rate) $\alpha > 0$



Batch Gradient Descent

- Loss function ℓ has been the average loss over all of the training examples:

$$\mathcal{E}(\omega) = \frac{1}{m}\sum_{i=1}^{m} \ell(\hat{y}_i, y_i) = \frac{1}{m}\sum_{i=1}^{m} \ell(h_\omega(x_i), y_i)$$

- By linearity,

$$\nabla_\omega \mathcal{E} = \nabla_\omega \frac{1}{m}\sum_{i=1}^{m} \ell(h_\omega(x_i), y_i) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial \omega}\ell(h_\omega(x_i), y_i)$$

$$\omega \leftarrow \omega - \alpha \nabla_\omega \mathcal{E}$$

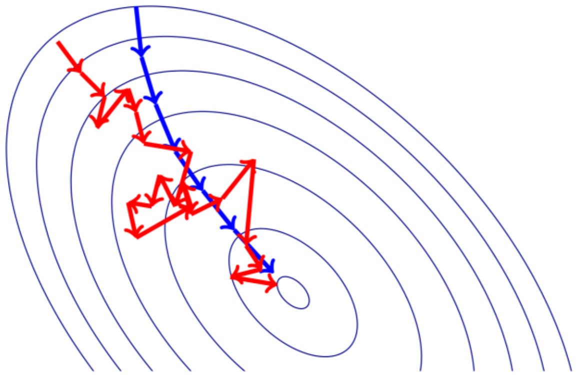
- Computing the gradient requires summing over all of the training examples.
- This is known as batch training.
- Batch training is impractical if you have a large dataset (e.g. millions of training examples) !

Stochastic Gradient Descent (SGD)

- Stochastic gradient descent (SGD): update the parameters based on the gradient for a randomly selected single training example:

$$\ell(\hat{y}_i, y_i) = \ell(h_\omega(x_i), y_i) = \ell^{(i)}$$

$$\omega \leftarrow \omega - \alpha \frac{\partial \ell^{(i)}}{\partial \omega}$$

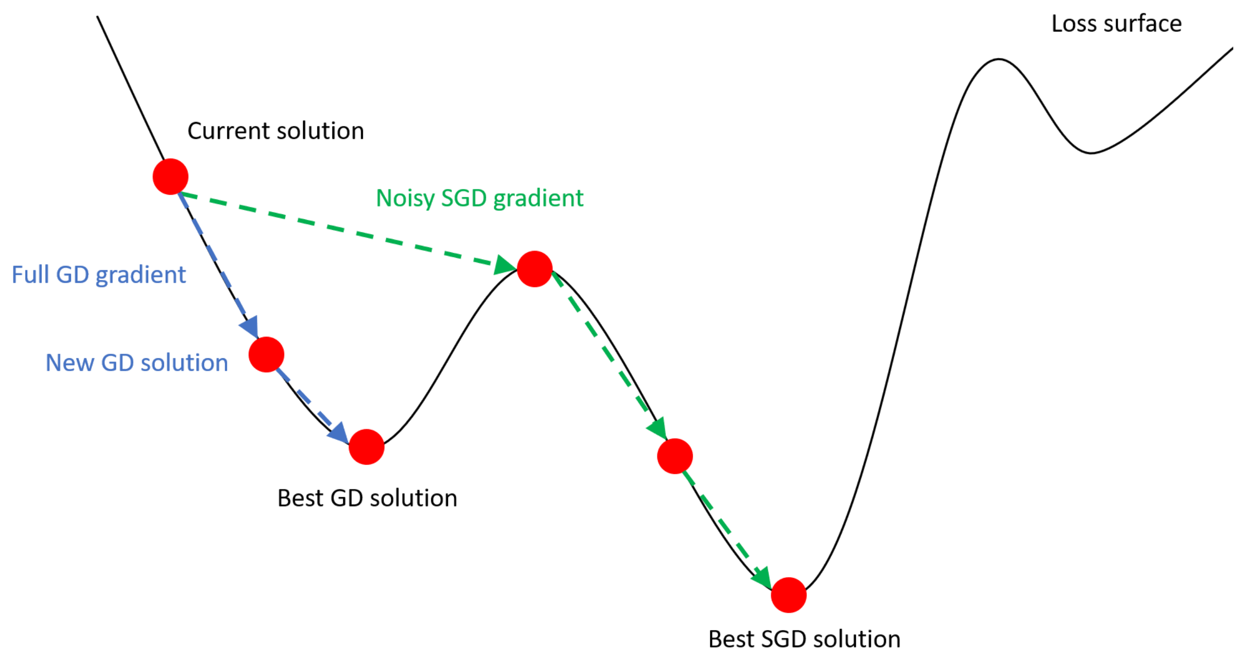


  – SGD takes steps in a noisy direction, but moves downhill on average.

- Mathematical justification: if you sample a training example at random, the stochastic gradient is an unbiased estimate of the batch gradient:

$$\mathbb{E}\left[\frac{\partial \ell^{(i)}}{\partial \omega}\right] = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial \ell^{(i)}}{\partial \omega} = \frac{\partial}{\partial \omega}\left[\frac{1}{m}\sum_{i=1}^{m}\ell^{(i)}\right] = \frac{\partial \mathcal{E}}{\partial \omega}$$

SGD is Sometimes Better



- No guarantee that this is what is going to always happen.
- But the noisy SGD gradients can help occasionally escaping local optima

Mini-batch Gradient Descent

- Potential problem of SGD: gradient estimates can be very noisy

- Compromise approach: compute the gradients on a medium-sized set of training examples s $(< m)$, called a mini-batch.

$$\mathcal{E}(\omega) = \frac{1}{s}\sum_{i=1}^{s} \ell(\hat{y}_i, y_i) = \frac{1}{s}\sum_{i=1}^{s} \ell(h_\omega(x_i), y_i) = \frac{1}{s}\sum_{i=1}^{s} \ell^{(i)}$$

$$\omega \leftarrow \omega - \alpha \nabla_\omega \mathcal{E}$$

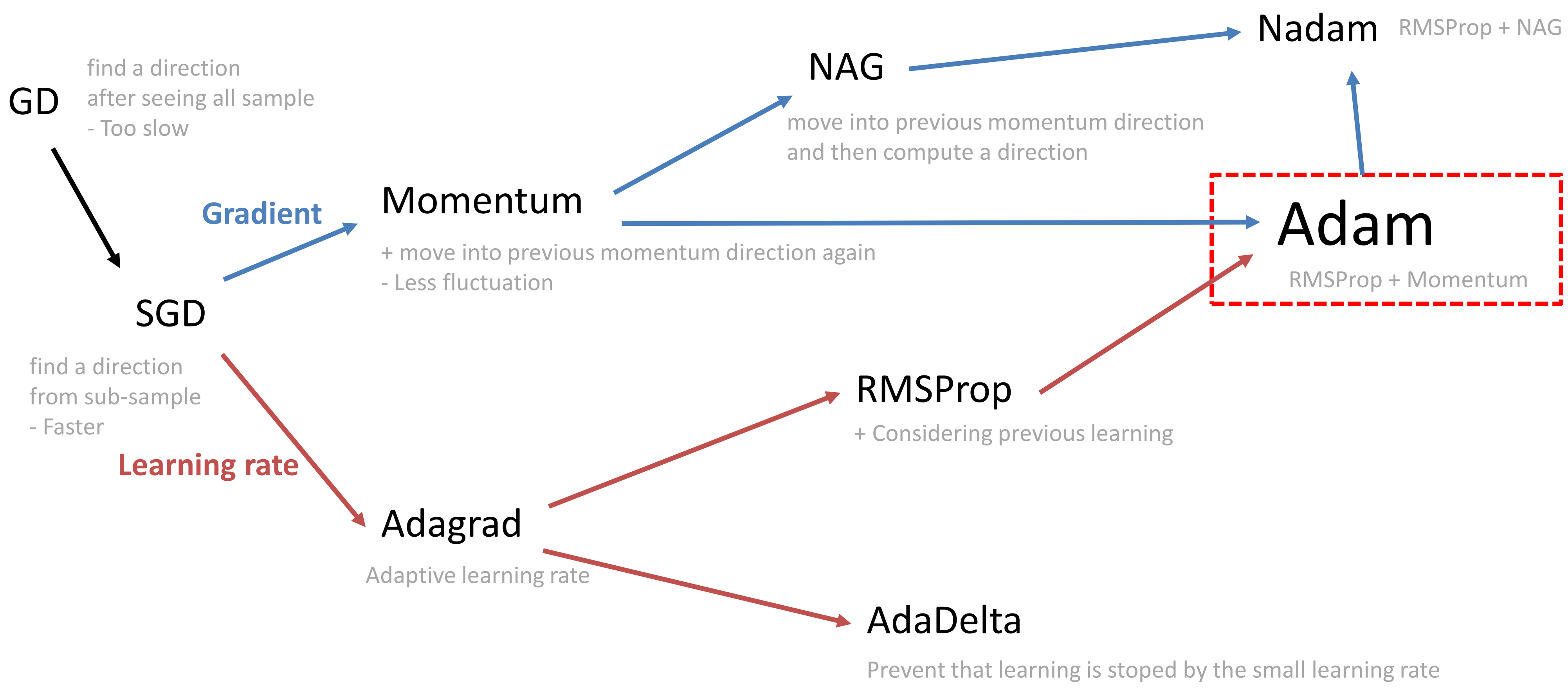
- Stochastic gradients computed on larger mini-batches have smaller variance:

$$\mathrm{var}\left[\frac{1}{s}\sum_{i=1}^{s} \frac{\partial \ell^{(i)}}{\partial \omega}\right] = \frac{1}{s^2}\mathrm{var}\left[\sum_{i=1}^{s} \frac{\partial \ell^{(i)}}{\partial \omega}\right] = \frac{1}{s}\mathrm{var}\left[\frac{\partial \ell^{(i)}}{\partial \omega}\right]$$

- The mini-batch size s is a hyper-parameter that needs to be set.

Advanced Optimizers from SGD

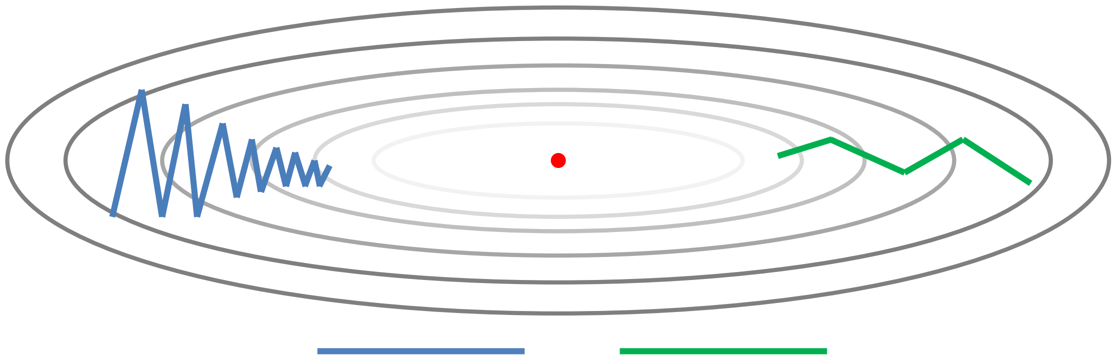
- History of Optimizers

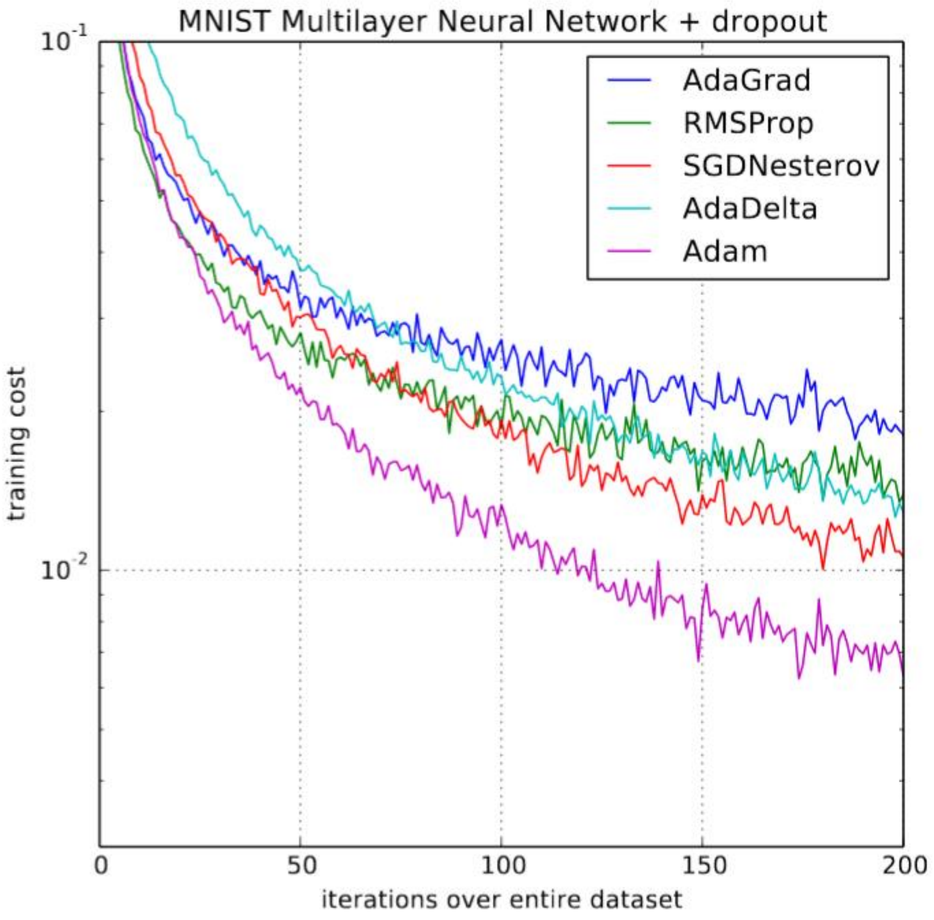


Reference: 자습해도 모르겠던 딥러닝, 머리속에 인스톨 시켜드립니다 - 하용호

Advanced Optimizers from SGD

- Smarter Gradient Descent
  - Toward gradient: Momentum, NAG
  - Toward learning rate: AdaGrad, AdaDelta, RMSProp
  - Combined: Adam (RMSProp + Momentum)





Source: Adam: A Method for Stochastic Optimization

Advanced Optimizers from SGD

- Momentum — `tf.train.MomentumOptimizer` — Qian et al. "On the momentum term in gradient descent learning algorithms." 1999.
- Adagrad — `tf.train.AdagradOptimizer` — Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.
- Adadelta — `tf.train.AdadeltaOptimizer` — Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.
- Adam — `tf.train.AdamOptimizer` — Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.
- RMSProp — `tf.train.RMSPropOptimizer`

- Additional detail: http://ruder.io/optimizing-gradient-descent/

Regularization and NN Techniques

Regularization (Shrinkage Methods)

- With many features, prediction function becomes very expressive (model complexity)
  - Choose less expressive function (e.g., lower degree polynomial, fewer RBF centers, larger RBF bandwidth)
  - Keep the magnitude of the parameter small
  - Regularization: penalize large parameters θ

$$\min \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2$$

  - λ: regularization parameter, trades off between low loss and small values of θ

Regularization (Shrinkage Methods)

- Often, overfitting associated with very large estimated parameters
- We want to balance
  - how well function fits data
  - magnitude of coefficients

$$\text{Total cost} = \underbrace{\text{measure of fit}}_{RSS(\theta)} + \lambda \cdot \underbrace{\text{measure of magnitude of coefficients}}_{\lambda \cdot ||\theta||_2^2}$$

$$\Longrightarrow \min \, ||\Phi\theta - y||_2^2 + \lambda||\theta||_2^2$$

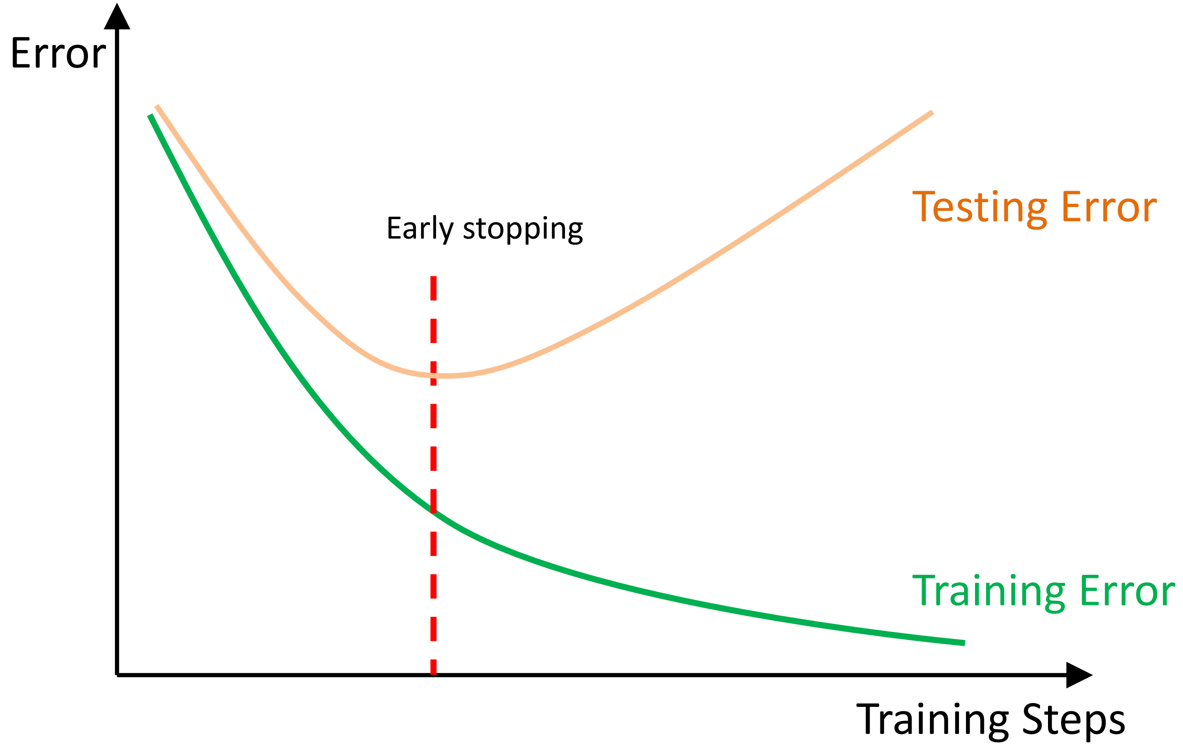
  - multi-objective optimization
  - λ is a tuning parameter

Different Regularization Techniques

- Big Data
- Data augmentation
  - The simplest way to reduce overfitting is to increase the size of the training data.

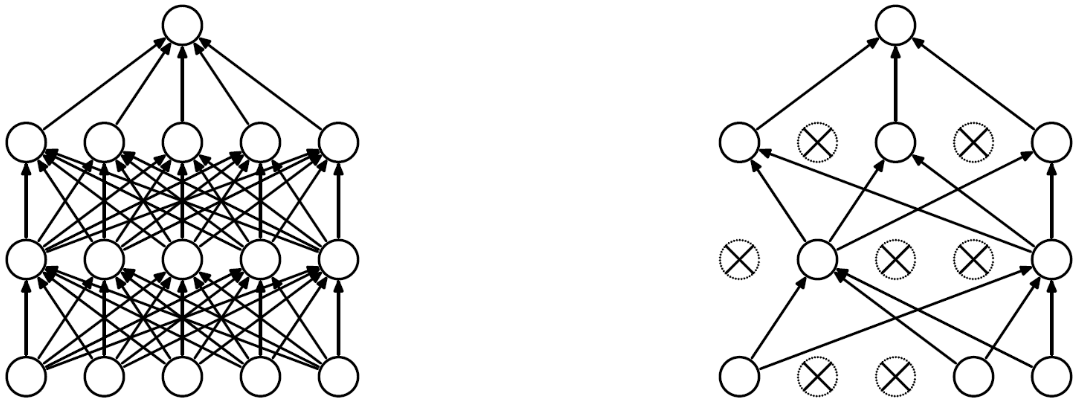
Different Regularization Techniques

- Early stopping
  - When we see that the performance on the validation set is getting worse, we immediately stop the training on the model.



Different Regularization Techniques in Deep Learning

- Dropout
  - This is the one of the most interesting types of regularization techniques.
  - It also produces very good results and is consequently the most frequently used regularization technique in the field of deep learning.
  - At every iteration, it randomly selects some nodes and removes them.
  - It can also be thought of as an ensemble technique in machine learning.



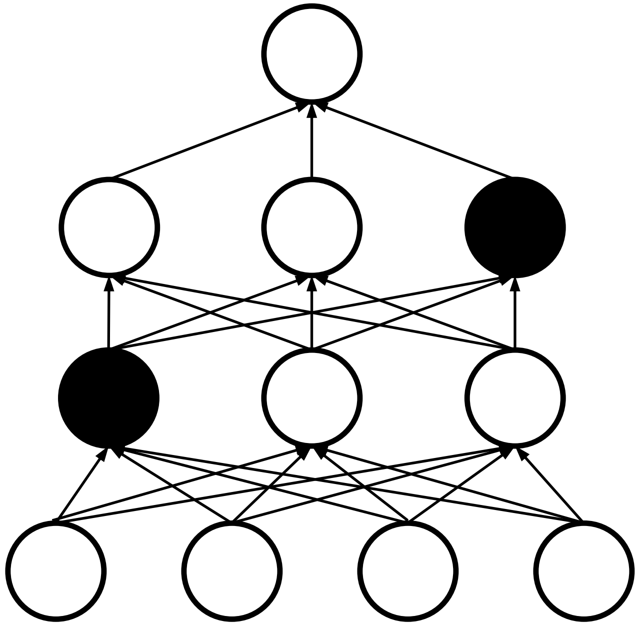
(a) Standard Neural Net (b) After applying dropout.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research (JMLR), 15:1929-1958, 2014.

Dropout Illustration

- Effectively, a different architecture at every training epoch
- It can also be thought of as an ensemble technique in machine learning.

Epoch 1



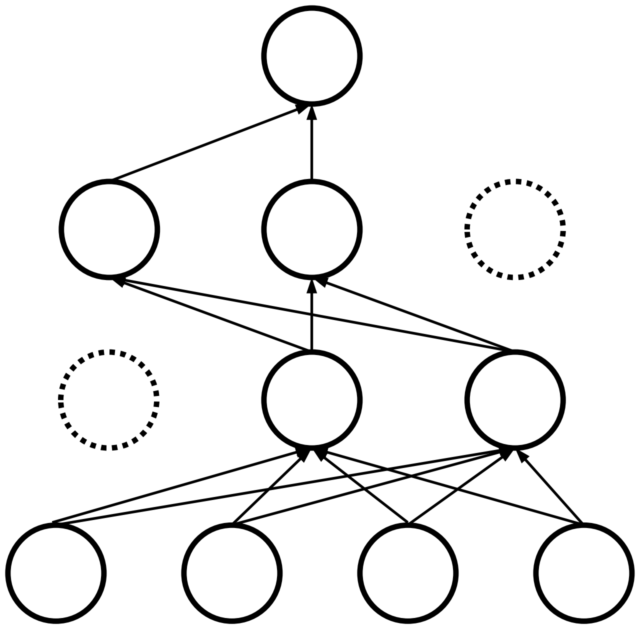
tf.nn.dropout(layer, rate = p)

rate: the probability that each element is dropped. For example, setting rate = 0.1 would drop 10% of input elements

Dropout Illustration

- Effectively, a different architecture at every training epoch
- It can also be thought of as an ensemble technique in machine learning.

Epoch 1

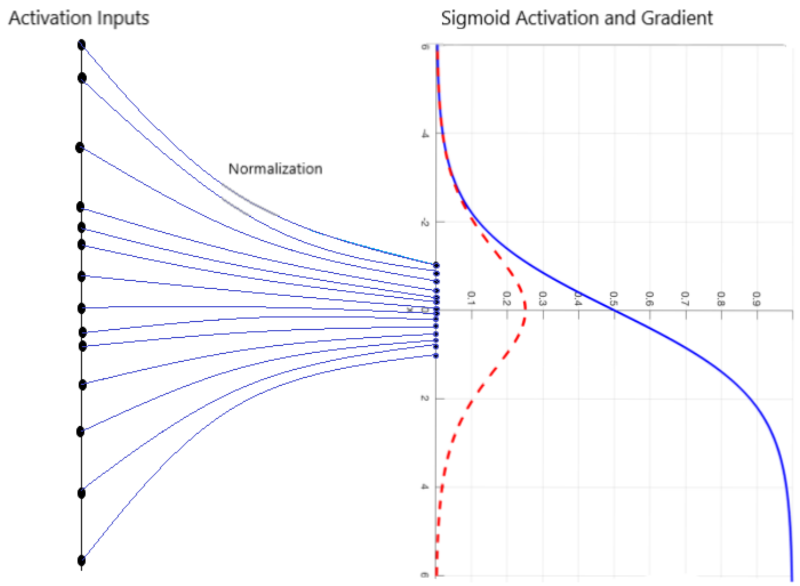


tf.nn.dropout(layer, rate = p)

rate: the probability that each element is dropped. For example, setting rate = 0.1 would drop 10% of input elements

Batch Normalization

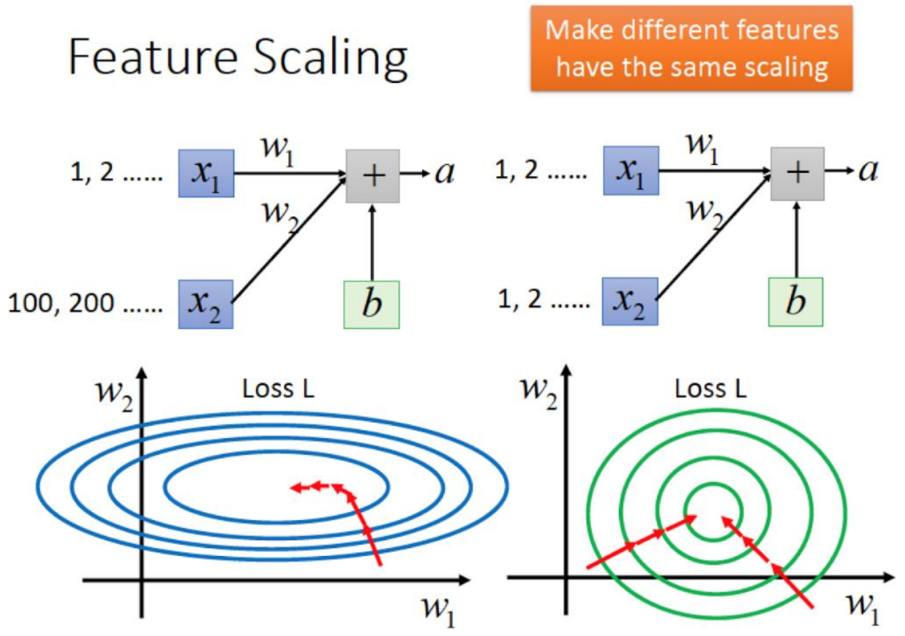
- Batch normalization is a technique for improving the performance and stability of artificial neural networks.
- It is used to normalize the input layer by adjusting and scaling the activations.



S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning (ICML), 2015.

Batch Normalization

- Batch normalization is a technique for improving the performance and stability of artificial neural networks.
- It is used to normalize the input layer by adjusting and scaling the activations.



S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning (ICML), 2015.

Batch Normalization

- During training batch normalization shifts and rescales according to the mean and variance estimated on the batch.

- During test, it simply shifts and rescales according to the empirical moments estimated during training.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning (ICML), 2015.

TensorFlow: DL Framework

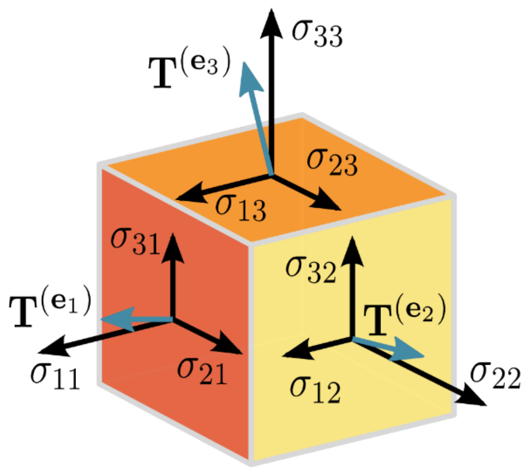
Training Neural Networks: Deep Learning Frameworks

- TensorFlow
  - Platform: Linux, Mac OS, Windows
  - Written in: C++, Python
  - Interface: Python, C/C++, Java, Go, R
- Keras
- PyTorch

TensorFlow

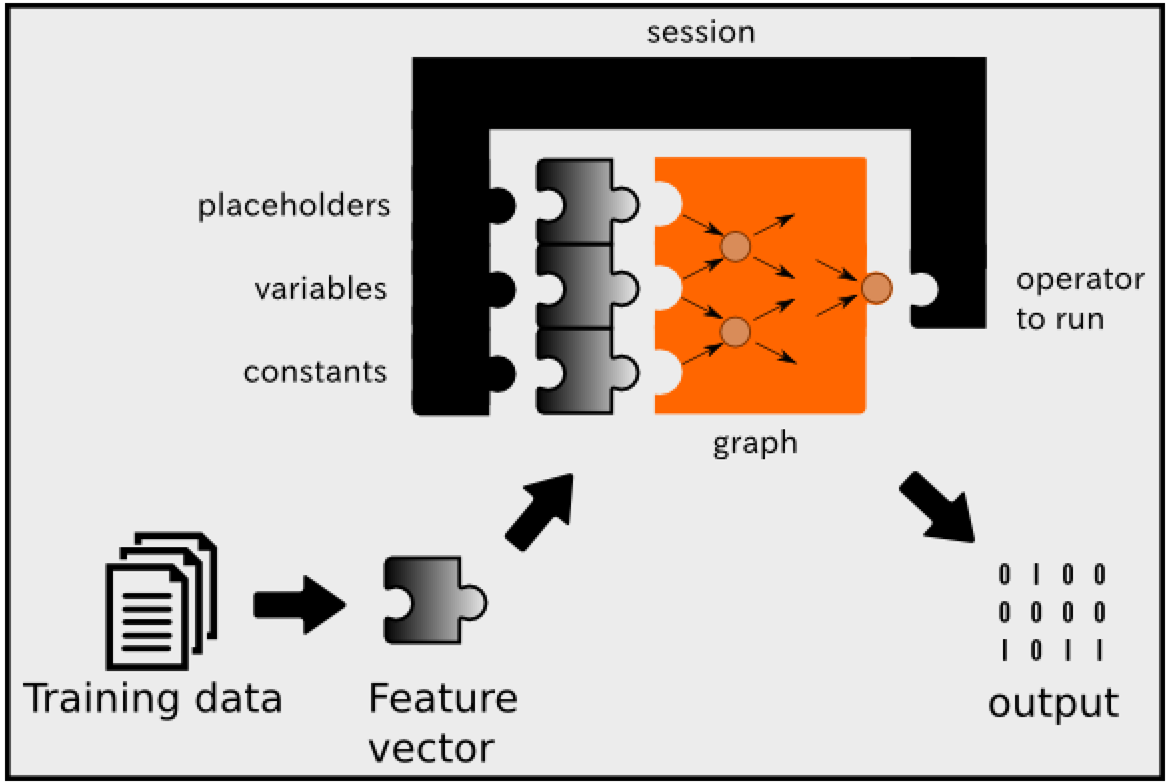
- Developed by Google and it is one of the most popular Machine Learning libraries on GitHub.
- It is a framework to perform computation very efficiently, and it can tap into the GPU in order to speed it up even further.
- TensorFlow is one of the widely used libraries for implementing machine learning and deep learning involving large number of mathematical operations.

- Tensor and Flow
  – TensorFlow gets its name from tensors, which are arrays of arbitrary dimensionality.
  – The "flow" part of the name refers to computation flowing through a graph.



TensorFlow: Session

- To run any of the three defined operations, we need to create a session for that graph. The session will also allocate memory to store the current value of the variable.




```
import tensorflow as tf

a = tf.constant([1,2,3])
b = tf.constant(4, shape=[1,3])

A = a + b
B = a*b

print(A)
```

```
Tensor("add_1:0", shape=(1, 3), dtype=int32)
```

```
sess = tf.Session()
sess.run(A)
```

```
array([[5, 6, 7]])
```