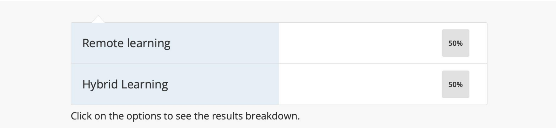
Applied Machine Learning for Business Analytics

Lecture 4: Deep Learning Practices

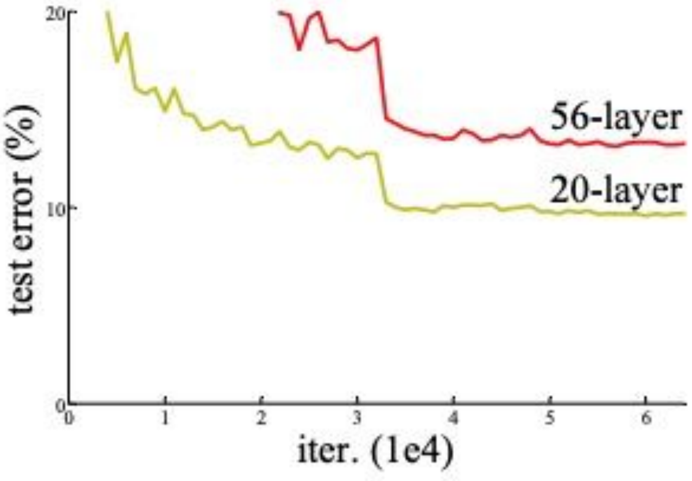
Lecturer: Zhao Rui

Logistics



- Received 68 responses for learning options
  - Maybe we will do the hybrid learning for the last few lectures.
- Assignment II is out. The due is next Friday 11:59 pm (SGT)
- Sanjay will conduct a training session on Keras tonight
- Appreciate if you keeps video on!

Overfitting?

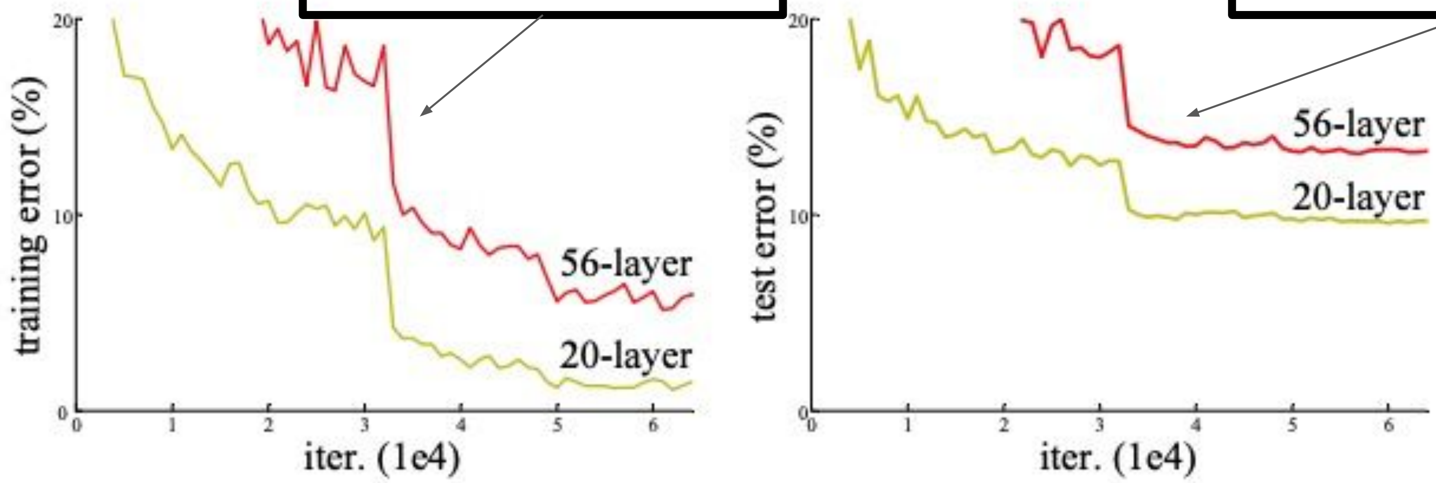


https://arxiv.org/abs/1512.03385

Training a deep model is challenging

The deeper model is not converged in training.

May not be due to overfitting



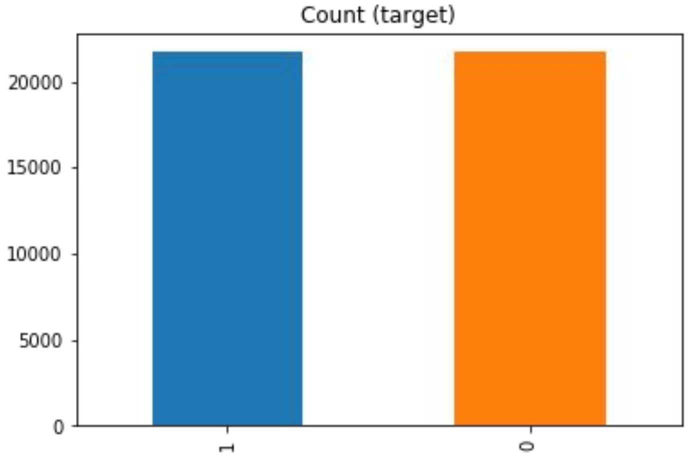
Source: https://arxiv.org/abs/1512.03385

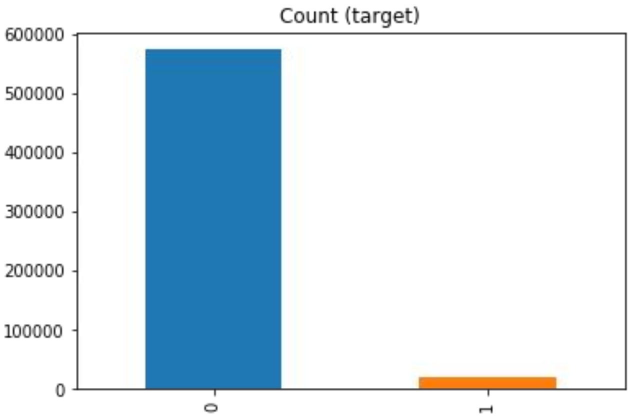
Agenda

1. Class Imbalance
2. Data Augmentation
3. Batch Normalization
4. Network Configuration
5. Parameters Initialization
6. Optimizers
7. Regularization Techniques

1. Class Imbalance

Small data in some categories





Class imbalance is the norm

- Bridge Structural Fault Detection
- Fraud Detection
- Disease Diagnosis
- Spam Detection

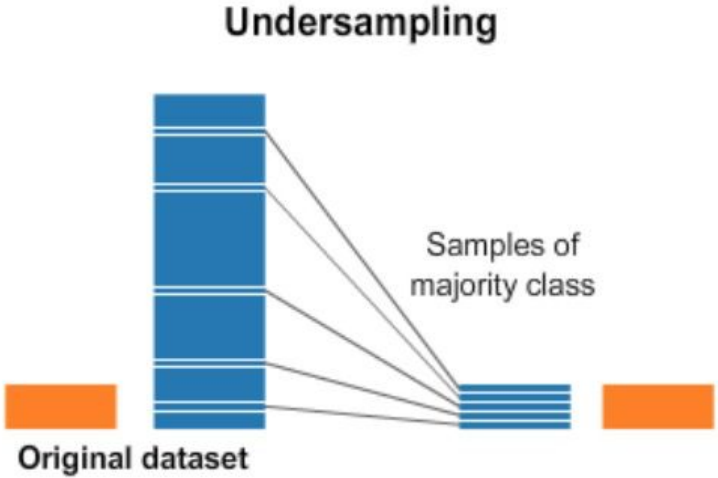
Class Imbalance is Challenging

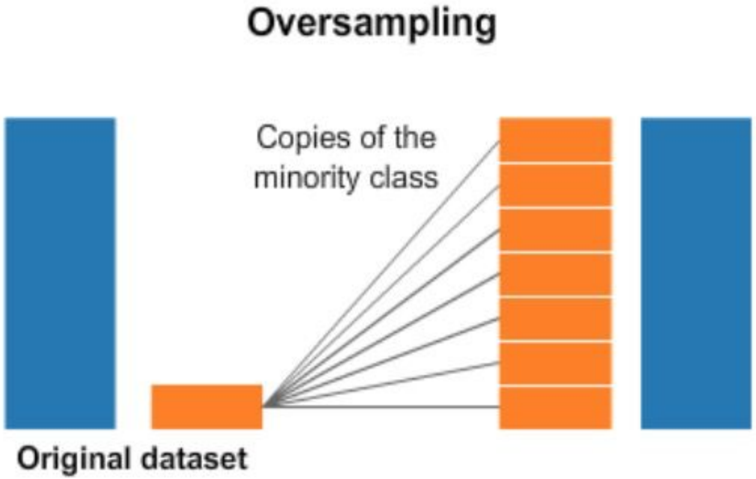
- Not enough knowledge to learn about rare classes
- Imbalanced problem: the number of fraud cases are much less than the one of normal cases.
- Rare classes are usually with high cost of wrong predictions.

How to deal with class imbalance

- Resampling
  - Add more minority samples
  - Remove majority samples
- Weights Balancing
  - Tweak the loss function
- Choose robust algorithms to class imbalance

Resampling

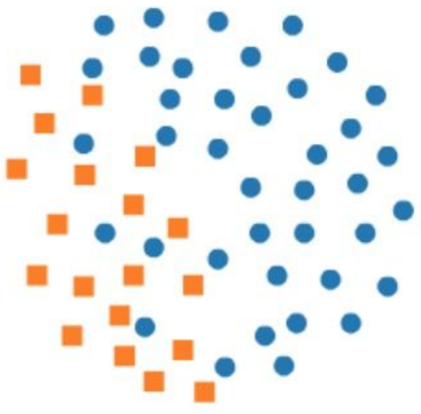




https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets#t1

Undersampling: Tomek Links

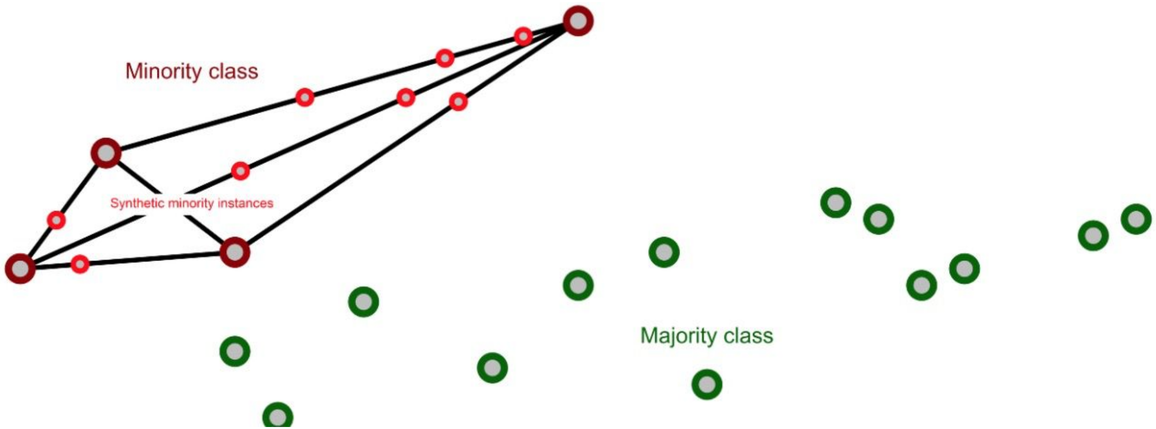
- Find pairs of close samples of opposite classes
- Remove the sample of majority class in each pair



https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets#t1

Oversampling: SMOTE

- Synthesize samples of minority class are convex(~linear) combinations of existing points and their nearest neighbors of same class.



https://rikunert.com/SMOTE_explained

Weight Balancing

- Normal Loss

$$L_\theta = \sum_i L_\theta(x_i)$$

- Weighted Loss

$$L_\theta = \sum_i w_{y_i} L_\theta(x_i)$$

$$w_c = \frac{N}{N_{y=c}}$$

the number of training samples

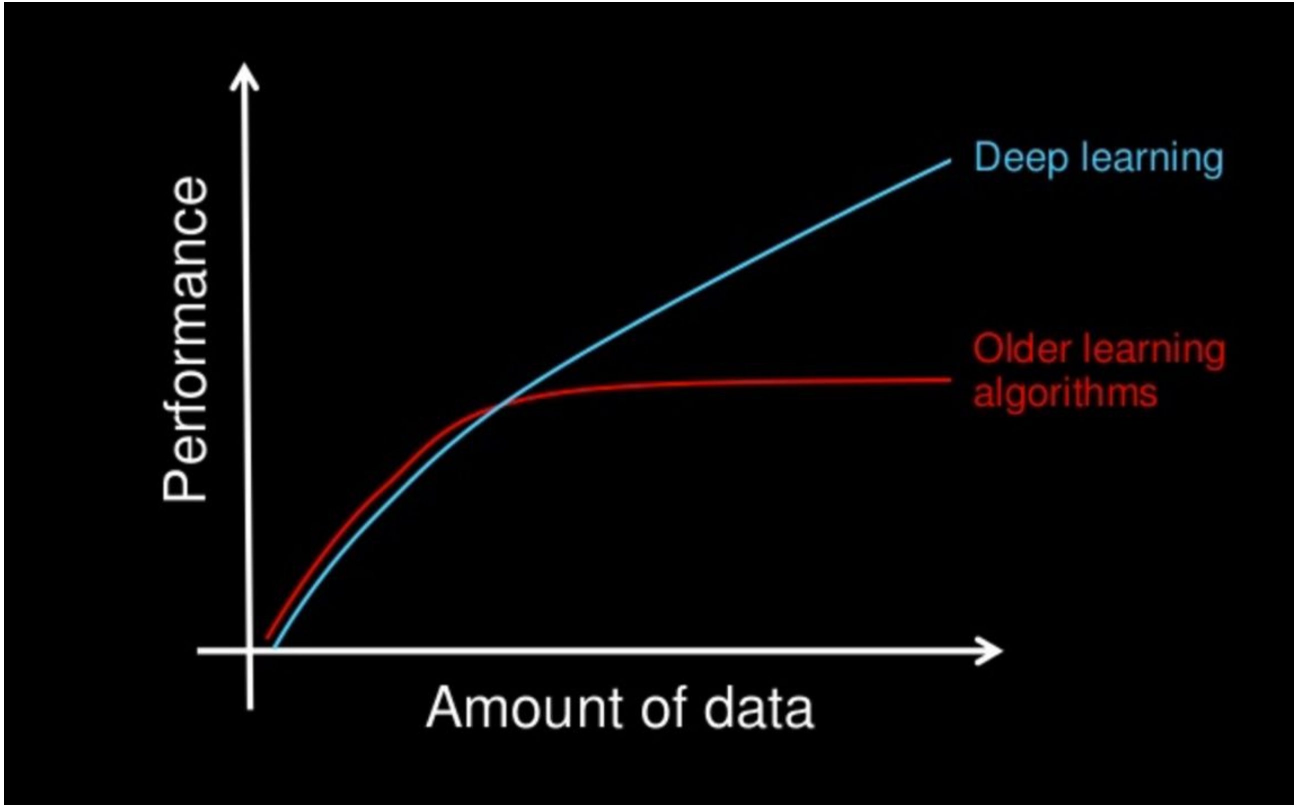
the number of training samples in the category: c

fit method

```
Model.fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose=1,
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
)
```

From Keras

2. Data Augmentation



From Andrew Ng

Data Augmentation

- Deep learning models usually have billions of parameters and then require massive labeled training data
- To improve the generalization capability

Data Augmentation: create artificially labeled training datasets

Image Augmentation

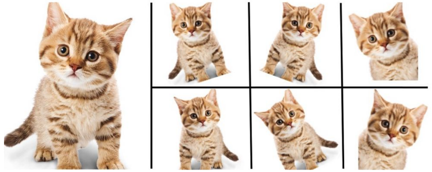
Original	Flip	Rotation	Random crop
			
• Image without any modification	• Flipped with respect to an axis for which the meaning of the image is preserved	• Rotation with a slight angle • Simulates incorrect horizon calibration	• Random focus on one part of the image • Several random crops can be done in a row

Color shift	Noise addition	Information loss	Contrast change
			
• Nuances of RGB is slightly changed • Captures noise that can occur with light exposure	• Addition of noise • More tolerance to quality variation of inputs	• Parts of image ignored • Mimics potential loss of parts of image	• Luminosity changes • Controls difference in exposition due to time of day

source : https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks

How about Text Data

- In computer version, data augmentation is quite common.



Enlarge your Dataset

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

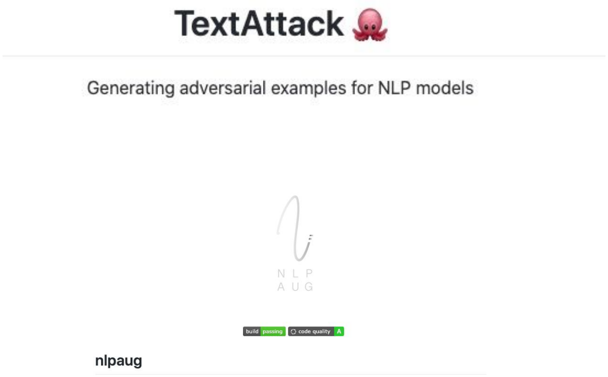
Rotating an image a few degrees does not change its semantics

- In NLP or text mining, data augmentation is challenging.

This is simple → (shuffle) Is this simple **Semantics changed**

Text Augmentation

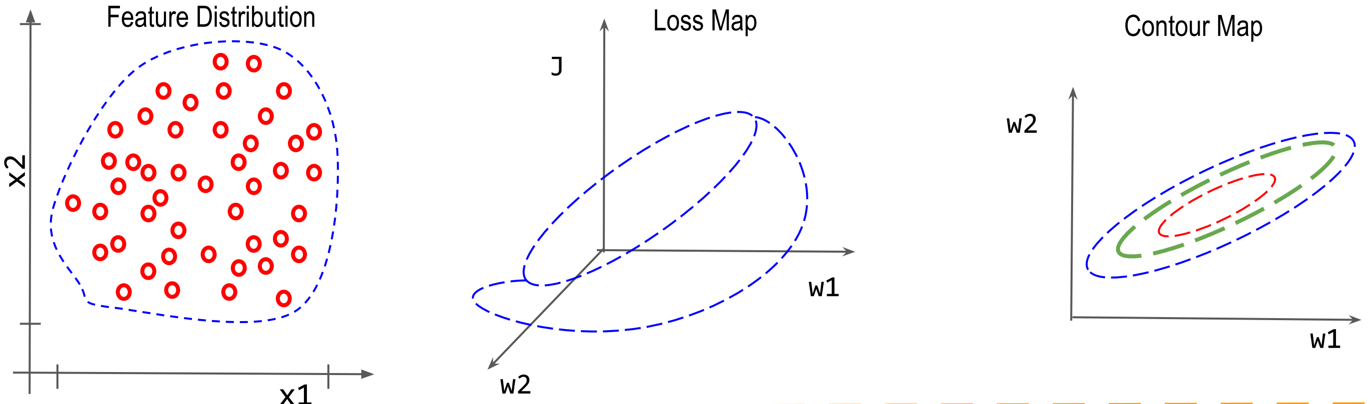
- Most of methods are very task-specific
  - Lexical Replacement
  - Back Translation
  - Text Surface Transformation
  - Random Noise Injection
  - Instance Crossover Augmentation
  - Generative Methods



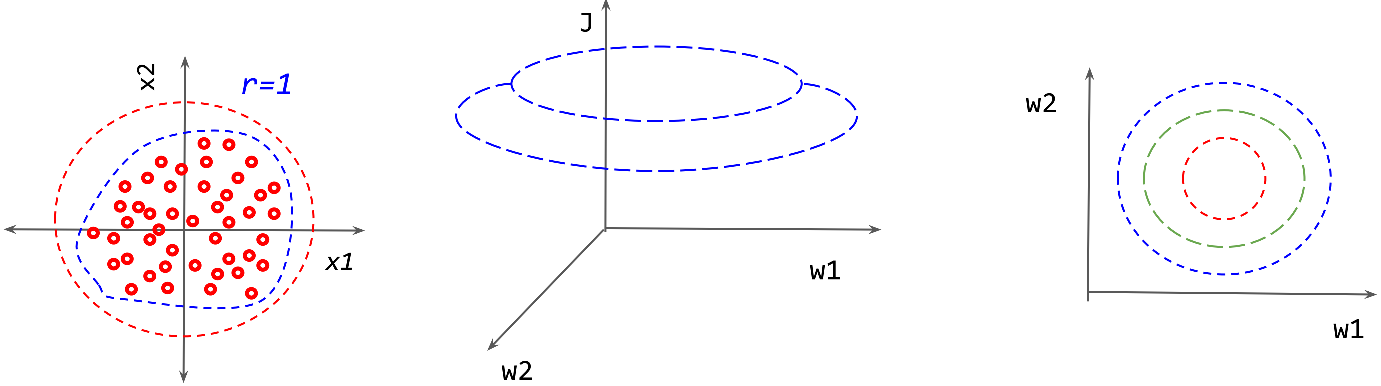
3. Batch Normalization

Normalization for Neural Network

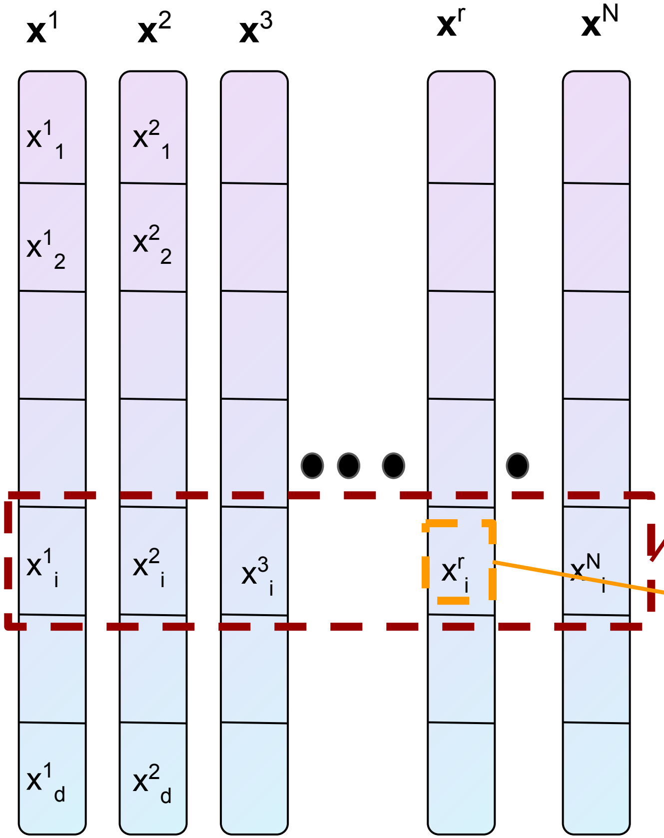
Before normalization



After normalization



Feature Normalization



1. For each dimension i, compute its mean: m_i and its standard deviation: σ_i
2. Apply the following transformation:

$$x_i^r = \frac{x_i^r - m_i}{\sigma_i}$$

It will speed up the gradient descent process.

How about Hidden Outputs?

It is challenging to normalize hidden outputs: $\mathbf{h}^1$ and $\mathbf{h}^2$

During training, their distributions are changed.

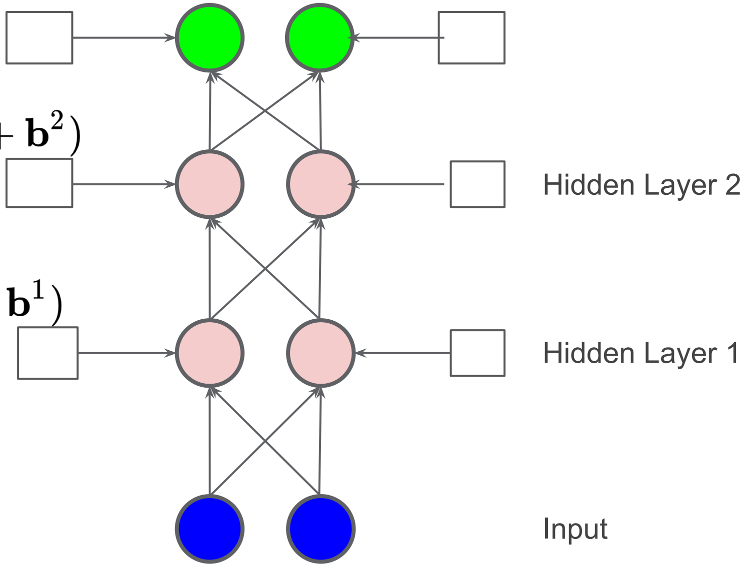
Batch Normalization

normalization is restrained to each mini-batch in the training process

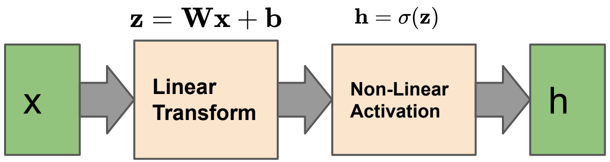
$$y = \mathbf{W}^3\mathbf{h}^2 + \mathbf{b}^3$$

$$\mathbf{h}^2 = \sigma(\mathbf{W}^2\mathbf{h}^1 + \mathbf{b}^2)$$

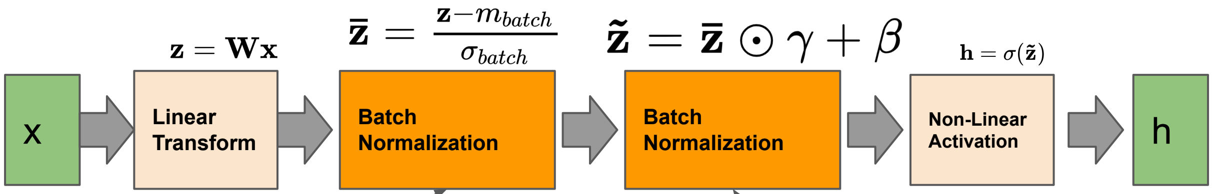
$$\mathbf{h}^1 = \sigma(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1)$$



Batch Normalization




```
model.add(layers.Dense(64, activation='relu'))
```




```
model.add(layers.Dense(64, use_bias=False))
model.add(layers.BatchNormalization())
model.add(Activation("relu"))
```

mean and std are computed from batch samples

Introduce network parameters to restore the representation power of the network

Why Batch Normalization?

- During testing: how to compute the mean and std
  - Ideal: computing mean and std using the whole training dataset.
  - In practice: compute the moving average of mean and std of the batches during training.

```
model.add(layers.Dense(64, use_bias=False))
model.add(layers.BatchNormalization())
model.add(Activation("relu"))
```

What is the number of parameters?

- Benefits behind BN
  - Reduce training times, make very deep structure trainable
  - Learning is more stable and less affected by initialization

4. Network Configuration

Last-Layer Configuration

Depends on the task type	Last-layer activation	Loss function
Binary Classification	sigmoid	binary_crossentropy
Multi-class Classification	softmax	categorical_crossentropy

Binary Classification:

```
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

Multi-class Classification:

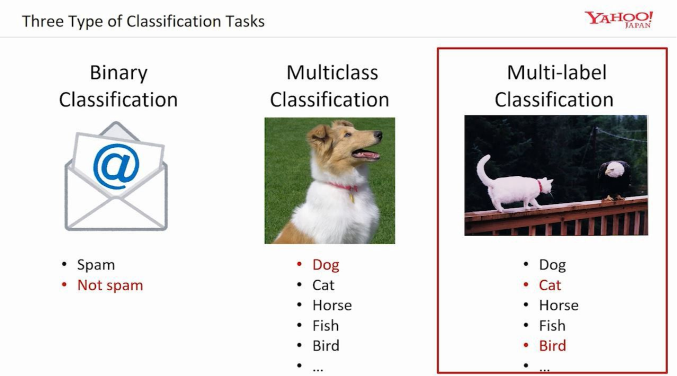
Number of unique labels in the task

```
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Last-Layer Configuration

Depends on the task type	Last-layer activation	Loss function
Multi-label Classification	Sigmoid	binary_crossentropy

```
model.add(layers.Dense(10, activation='sigmoid'))
model.compile(loss="binary_crossentropy", optimizer='rmsprop')
```



https://www.microsoft.com/en-us/research/uploads/prod/2017/12/40250.jpg

Last-Layer Configuration

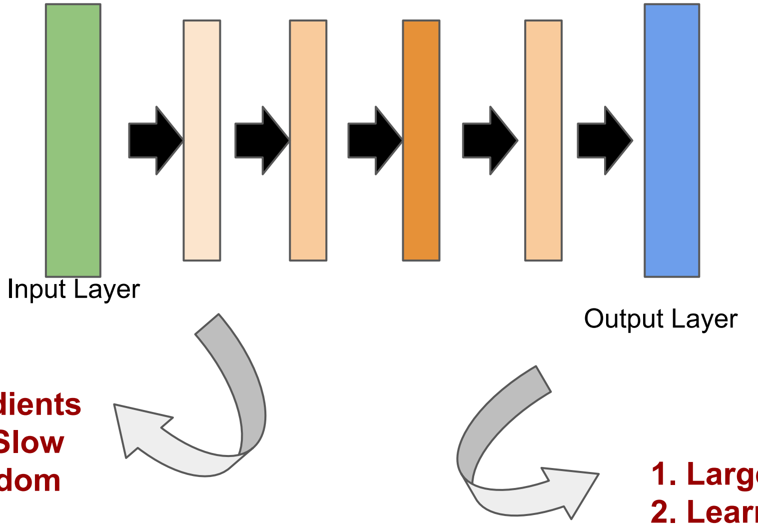
Depends on the task type	Last-layer activation	Loss function
Regression to arbitrary values	Linear	mse

```
model.add(layers.Dense(1))
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

Depends on the task type	Last-layer activation	Loss function
Regression to scaled values ranging from 0 to 1	sigmoid	mse

```
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

Vanishing Gradient Problem



1. Smaller Gradients
2. Learn Very Slow
3. Almost Random

1. Larger Gradients
2. Learn Very Fast
3. Converged Already

Backpropagation (From Last Lecture)

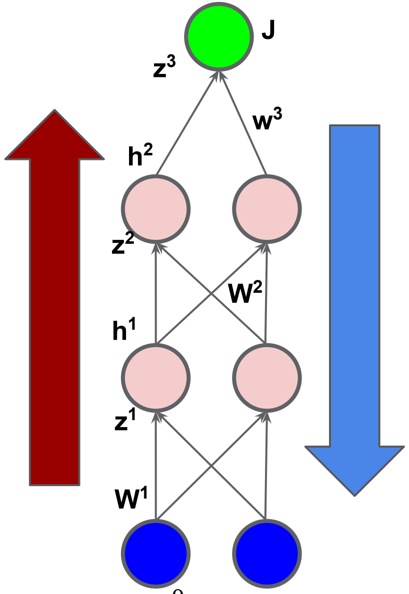
Definition（from wiki):

By computing the gradient of the loss function with respect to each weight by the **chain rule**, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule

Predictions flows forward

$$h^l = \sigma(\mathbf{z}^l)$$

$$\mathbf{z}^l = \mathbf{W}^l h^{l-1} + \mathbf{b}^l$$



$$\mathbf{x} = \mathbf{h}^0$$

Error flows backward

$$\frac{\partial J}{\partial \mathbf{w}^3} = \frac{\partial J}{\partial \mathbf{z}^3} \frac{\partial \mathbf{z}^3}{\partial \mathbf{w}^3}$$

$$\frac{\partial J}{\partial \mathbf{W}^2} = \frac{\partial J}{\partial \mathbf{z}^3} \frac{\partial \mathbf{z}^3}{\partial \mathbf{h}^2} \frac{\partial \mathbf{h}^2}{\partial \mathbf{z}^2} \frac{\partial \mathbf{z}^2}{\partial \mathbf{W}^2}$$

From $\mathbf{w}^3$

$$\frac{\partial J}{\partial \mathbf{W}^1} = \frac{\partial J}{\partial \mathbf{z}^3} \frac{\partial \mathbf{z}^3}{\partial \mathbf{h}^2} \frac{\partial \mathbf{h}^2}{\partial \mathbf{z}^2} \frac{\partial \mathbf{z}^2}{\partial \mathbf{h}^1} \frac{\partial \mathbf{h}^1}{\partial \mathbf{z}^1} \frac{\partial \mathbf{z}^1}{\partial \mathbf{W}^1}$$

From $\mathbf{w}^3$ From $\mathbf{W}^2$

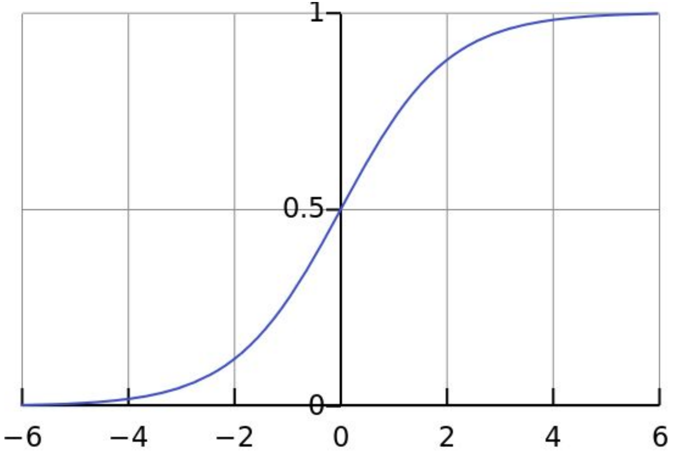
More details:
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Sigmoid Function

Equation:

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

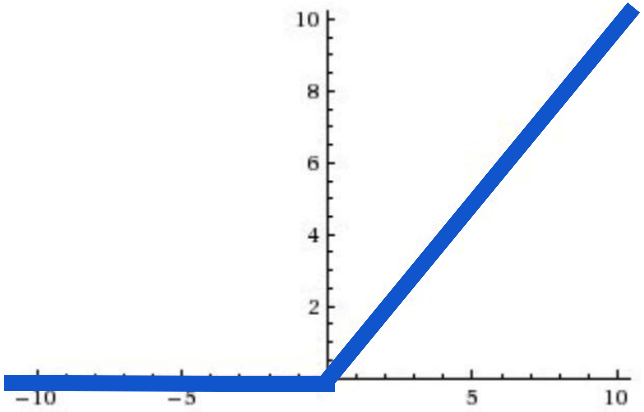
- Vanishing Gradient Problem



How about gradient curve？

ReLU Function

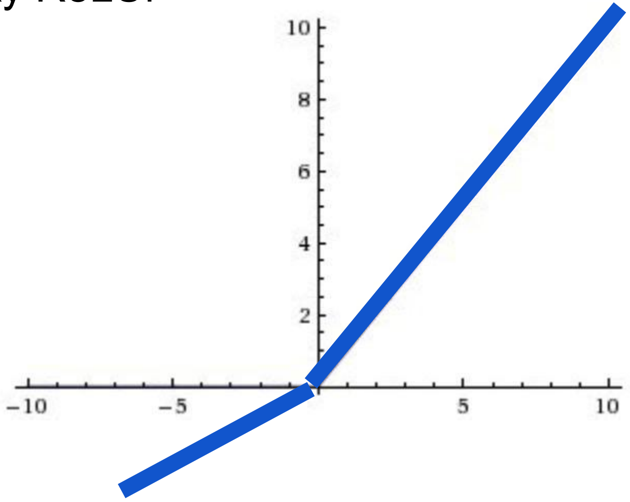
- Fast Compute
- Still have vanishing gradient problem



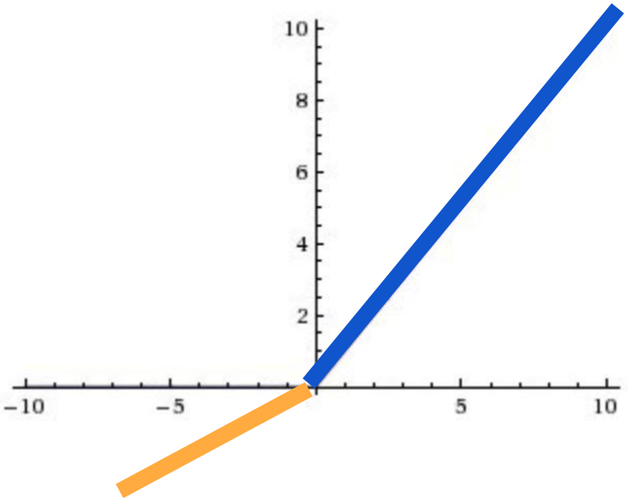
How about gradient curve？

ReLU Variants

Leaky ReLU:



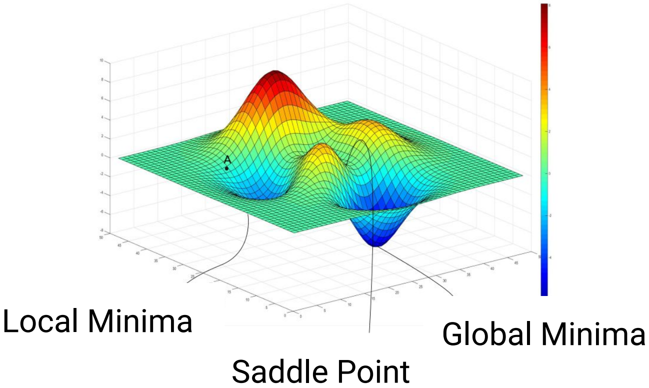
Parametric ReLU::



5. Parameters Initializations

Initialization

- Optimization for neural network in nature is a iterative method, which requires initialization



- Some general rules for initialization of model parameters:
  - Can not initialize all weights to the same value
  - Randomness should be incorporated

Normal Distribution

- Initialize weights randomly, following standard normal distribution
  - The normal distribution should take into account characteristics that are unique to the architecture

For Layers with ReLu

$$\sqrt{\frac{2}{size^{[l-1]}}}$$

$$W^{[l]} = np.random.randn(size\_l, size\_l\text{-}1) * np.sqrt(2/size\_l\text{-}1)$$

https://keras.io/api/layers/initializers/

For Layers with Tanh/Sigmoid

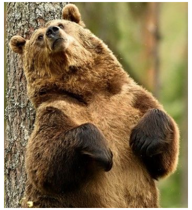
$$\sqrt{\frac{1}{size^{[l-1]}}}$$

$$W^{[l]} = np.random.randn(size\_l, size\_l\text{-}1) * np.sqrt(1/size\_l\text{-}1)$$

https://datascience.stackexchange.com/questions/17987/how-should-the-bias-be-initialized-and-regularized

Transfer Learning

Task: Build a bear/cat classifier



bear

cat

Available Data: not **directly** related

dog

panda

bear

cat

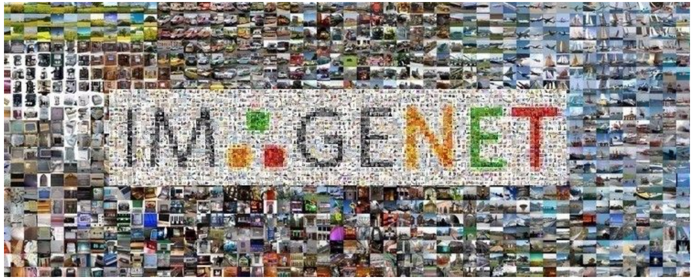
similar domain, different tasks

Different domains, similar task

Applications

- Sentiment Analysis
  - Available data: IMDB reviews

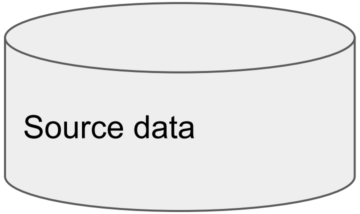
  - Target taks: Teaching feedback analysis
- Image Classification:
  - Available data: Imagenet Dataset



  - Target Task: Cancer Diagnostic (Medial Image)

How to transfer knowledge

- Task Definition:



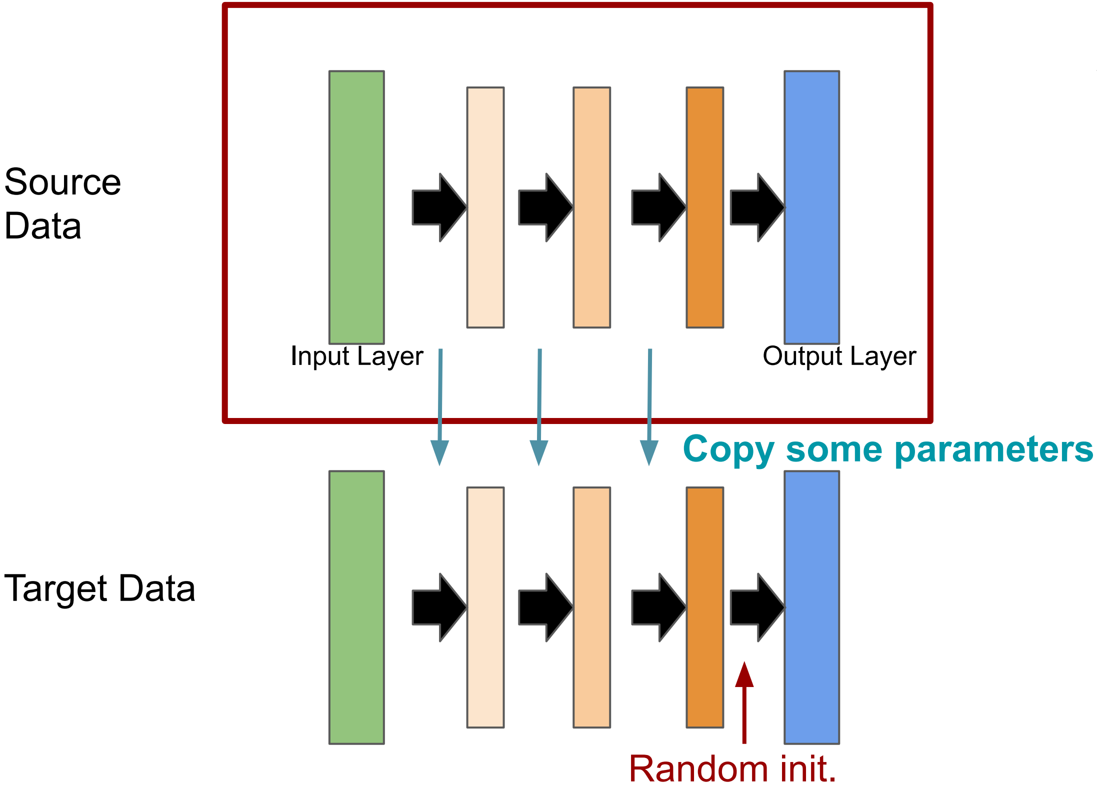
(x^s, y^s) Large-size

(x^t, y^t) Small-size

- Steps:
  - Train a model using the source data
  - Transfer layer from the model trained in source domain to the model in target domain
  - Fine-tune the model using the target data

Any concerns?

Layer Transfer



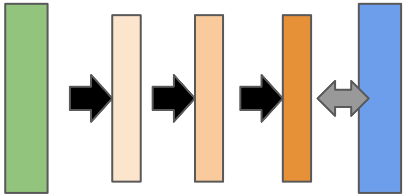
Neural Network: Layer-wise self-contained

1. **Same Task:**
   Copy all layers' parameters
2. **Different Tasks:**
   Random initialize the softmax/last layer and copy the rest layers' parameters

Fine-tune

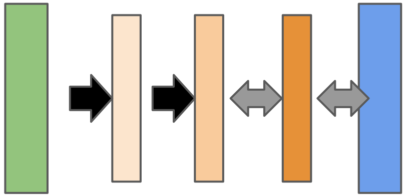
Target Data Size

Small



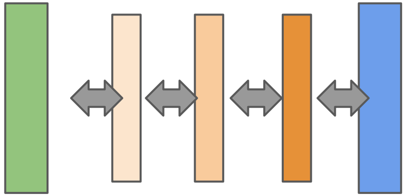
Freeze all layers, train weights on softmax/regression layer

Medium



Freeze most layers, train weights on last layers and softmax/regression layer

Large



Fine-tune all layers

6. Optimizers for Neural Network

SGD

Gradient for loss function f over parameters, which computed by BP algorithm

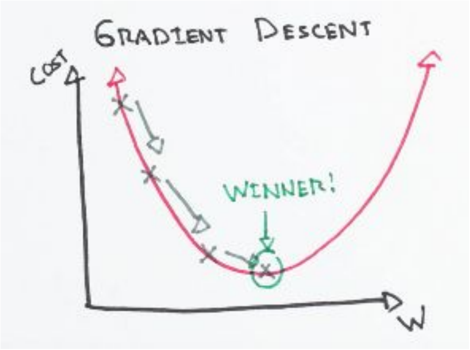
$$\theta_{t+1} = \theta_t - \alpha \boxed{\nabla J(\theta_t)}$$

New Parameters Guess

Current Parameters Guess

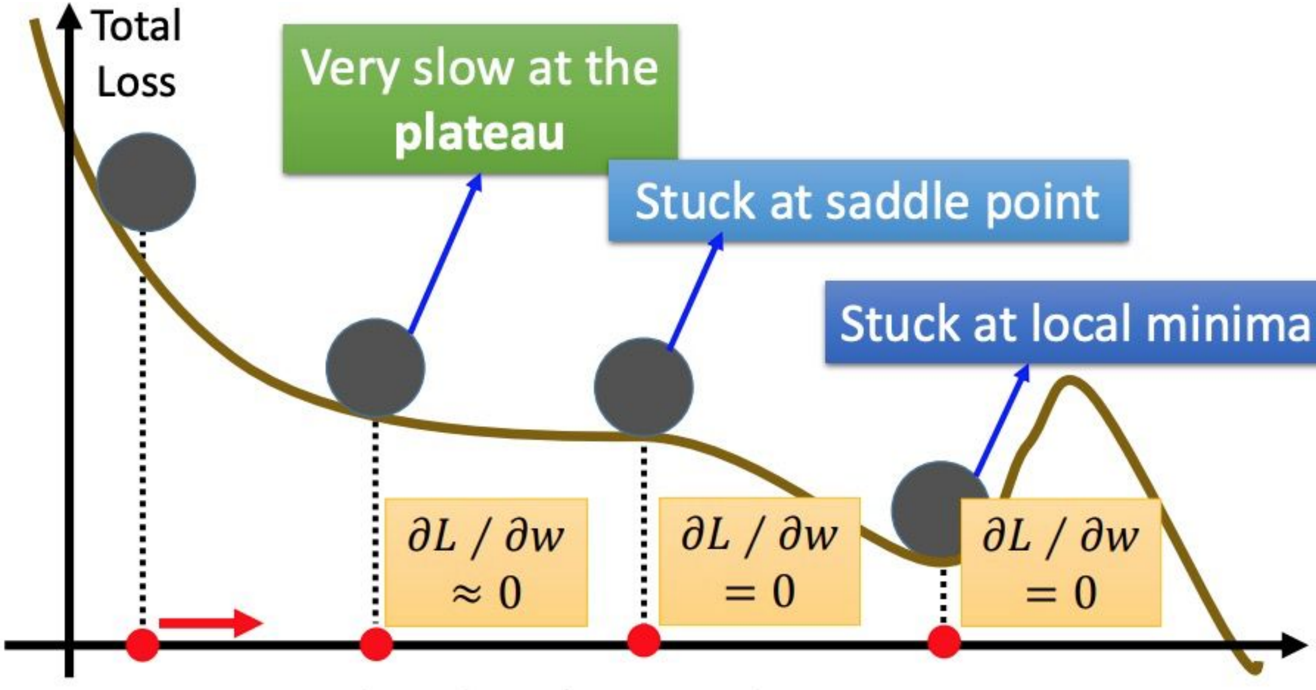
Learning Rate

Like hiking down a mountain



Credit:https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

Hard to find optimal network parameters



Source: https://speech.ee.ntu.edu.tw/~tlkagk/

Momentum

- Core idea: the current gradient computation will keep the direction as the previous gradient computation

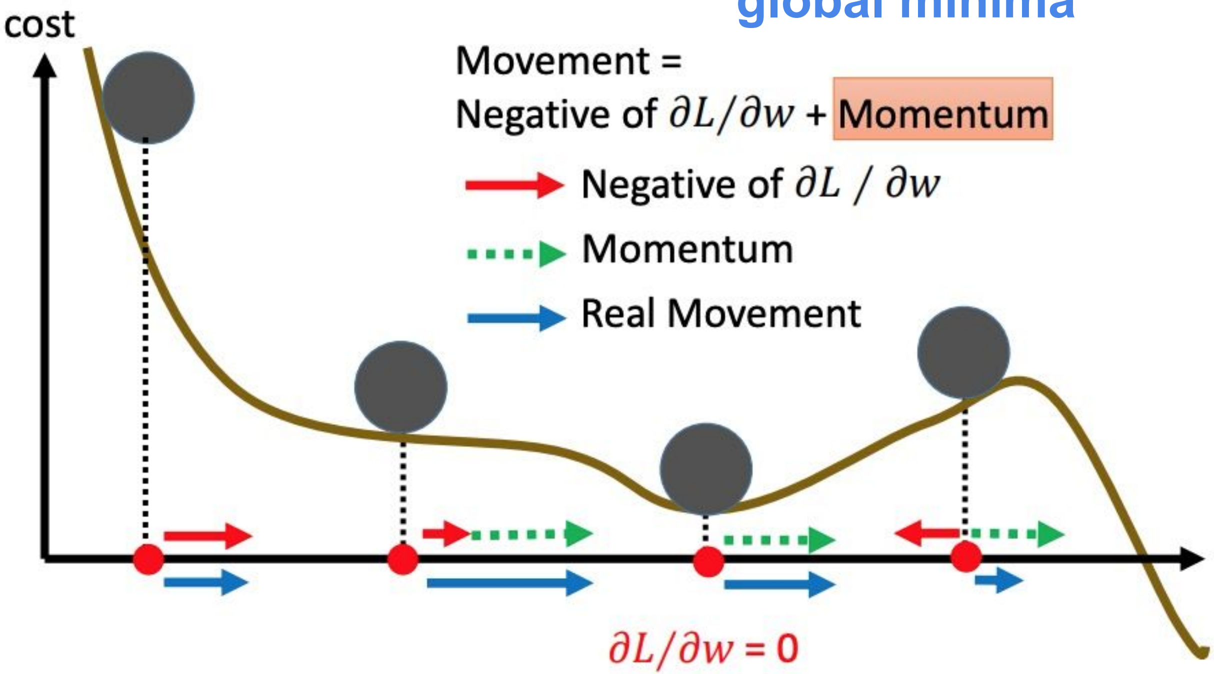
Updated vector of the previous time step

$$v_t = \beta v_{t-1} + \alpha \bigtriangledown J(\theta_t)$$
$$\theta_{t+1} = \theta_t - v_t$$

- Accelerate SGD
- Dampens Oscillations
- Two Parameters to tune

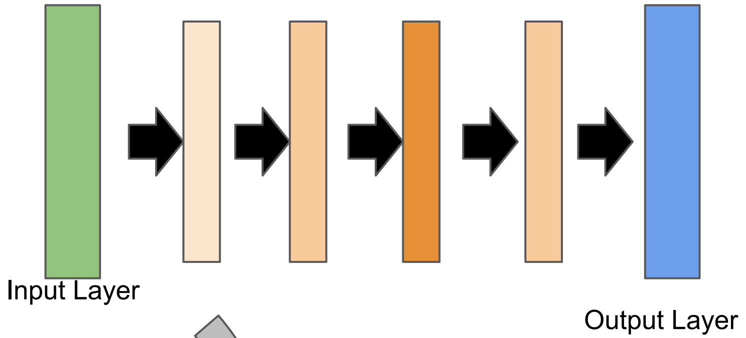
Momentum

Improve the chance to find the global minima



Source: https://speech.ee.ntu.edu.tw/~tlkagk/

Separated Adaptive Learning Rate



Parameters Update = Gradients * Learning Rate

1.Smaller Gradients
2. Learn Very Slow
3. Almost Random

Large Learning Rate

1. Larger Gradients
2. Learn Very Fast
3. Converged Already

Small Learning Rate

Keep a moving average of the squared gradient for each parameter to change the learning rate.

How to select the optimizer

- Except SGD, Momentum, RMSprop and Adam, other popular methods include Adadelta and Adagrad.
- It is hard to find a general answer
- Adam is the most commonly used technique
- If you want to train a deep or complex neural networks with fast converge, do not just use SGD.

7. Regularization Techniques

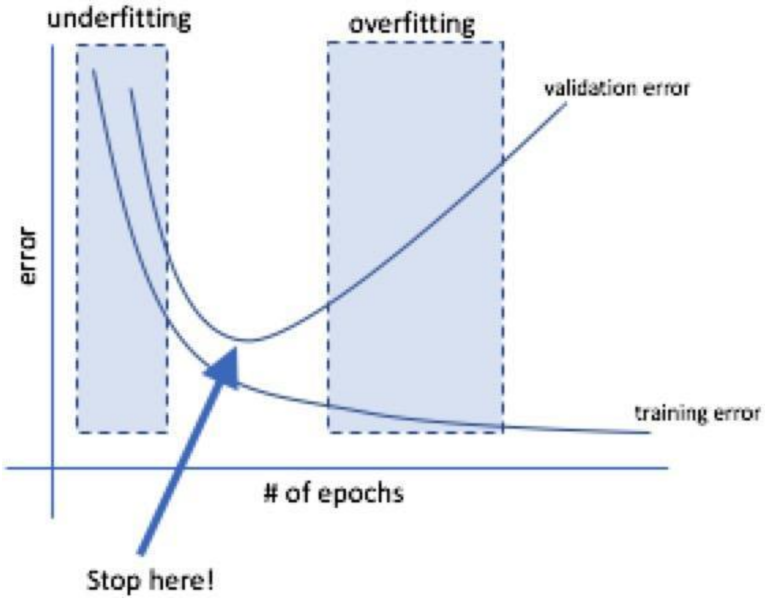
Overfitting for NN

- Neural Network with a deep structure easily get overfitted
  - Early stopping
  - Parameters Regularization
  - Dropout
  - Most effective: Train with more data.

Early Stopping

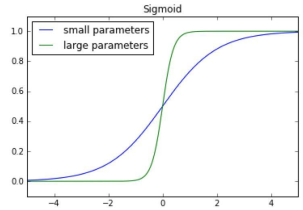
- Watch the validation curve
- Stop updating the weights once validation errors starts increasing

In Keras: https://keras.io/api/callbacks/early_stopping/



Parameter Regularization

- Why large model parameters should be penalized:
  - In NN, inputs are linearly combined with parameters. Therefore, large parameters can amplify small changes in the input.
  - Large parameters may **arbitrarily** increases the confidence in our predictions.



To make sure that parameters are not too large and then the model is not overfitting

Add regularization terms to the loss function

$$.....+\lambda g(\theta)$$

Control the degree to which we select to penalize large parameters

Regularization Terms

- L1 Regularization:

$$g(\theta) = ||\theta||_1$$

L1-norm is commonly used for feature selection as it tends to produce sparse parameter vectors where only the important features take on non-zero values

- L2 Regularization:

$$g(\theta) = ||\theta||_2^2$$

L2-Norm does not tend to push less important weights to zero and typically produces better results when training a model.

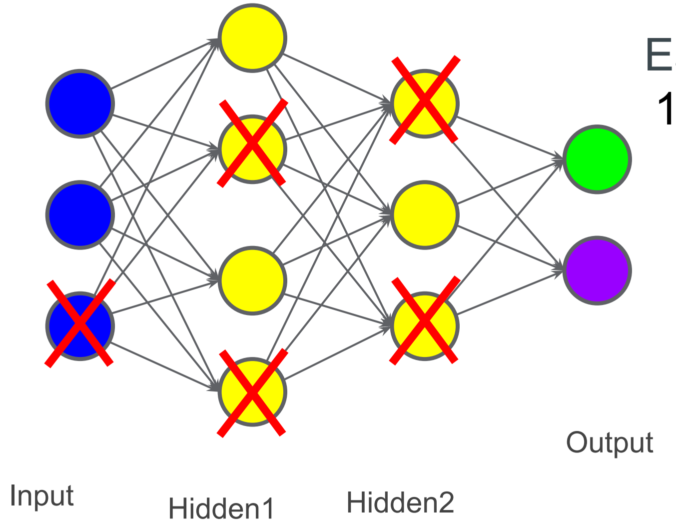
- Elastic Net:

$$g(\theta) = \alpha||\theta||_1^1 + (1-\alpha)||\theta||_2^2$$

Trade-off between L1 and L2 Regularization techniques

Dropout

Training:

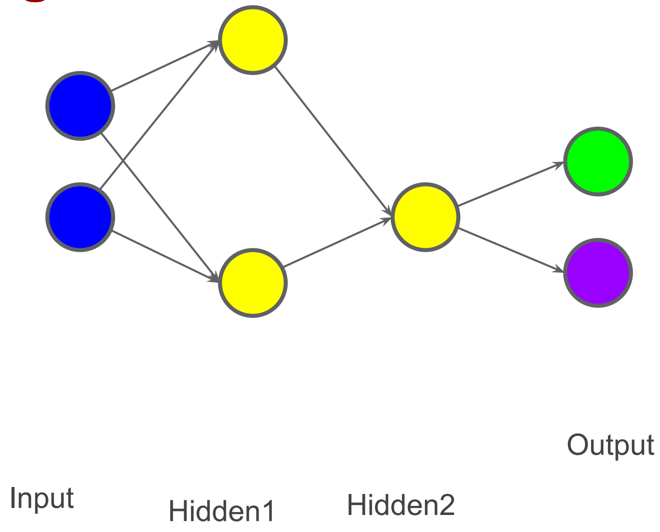


Each mini-batch before updating the parameters

1. Each neuron has **%p** to dropout(mask)

Dropout

Training:



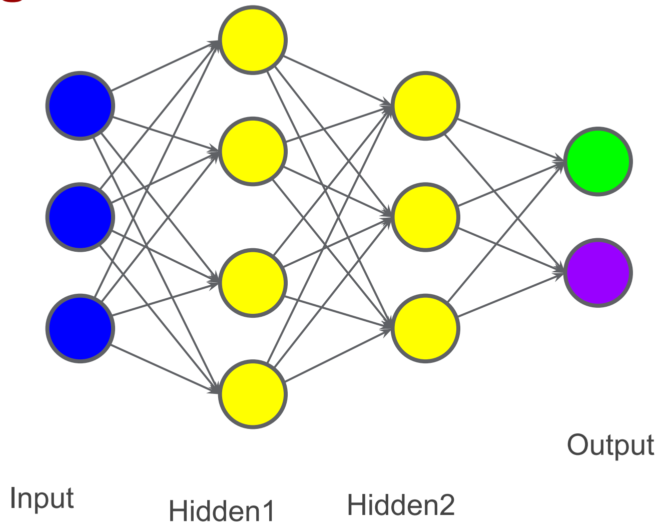
Each mini-batch before updating the parameters

1. Each neuron has **%p** to dropout(mask)
2. The network structure is changed (More Thinner!)
3. Using the updated network structure for training

For each mini-batch, we resample the dropout neurons.

Dropout

Testing:



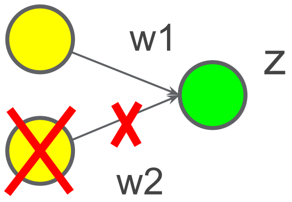
No dropout, but shrink weights following the rule:

If the dropout rate during training is p%, all the weights will time 1-p%.

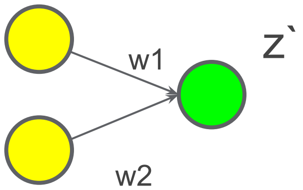
When many people work together, they usually rely on others to do more of the work and share the same results.

Dropout

Training: Assume dropout rate is 50%



Testing: No dropout



Directly Copy:

z` = 2z

Weight multiply 1-p%:

z` ~= z

Dropout Effects

Experimental Studies on MINIST dataset:

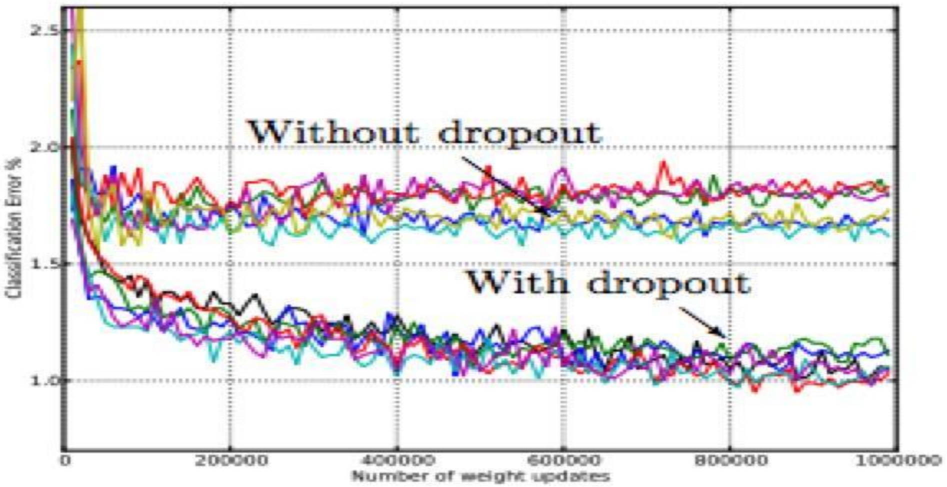


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.