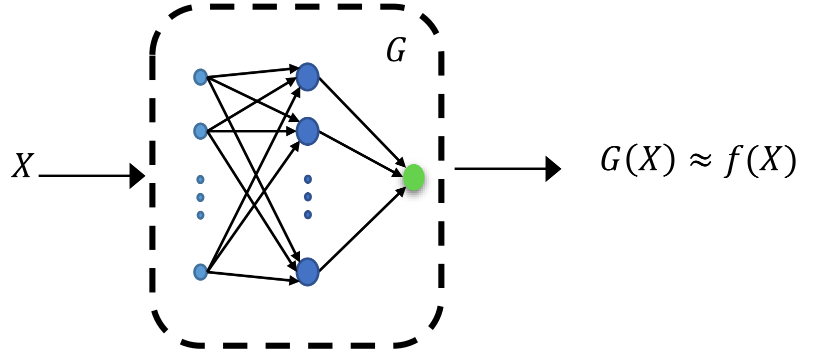
Deep Learning

What is Deep Learning

- Deep Learning is hierarchical feature learning
  - *The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. (*Ian Goodfellow and Aaron Courville)
  - *Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. (*Yoshua Bengio and Geoffrey Hinton)

- Deep Learning is large Neural nets
  - *When you hear the term deep learning, just think of a large deep neural net. (Jeff Dean)*

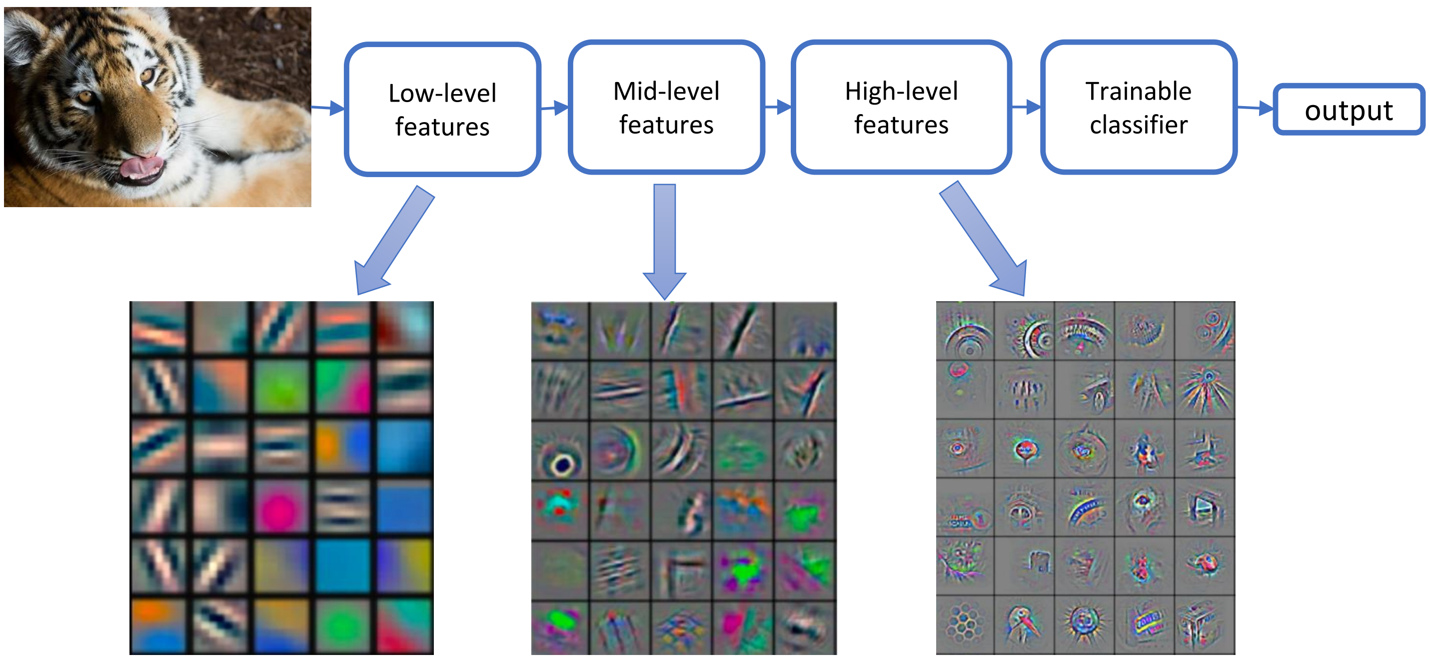
Universal approximation theorem



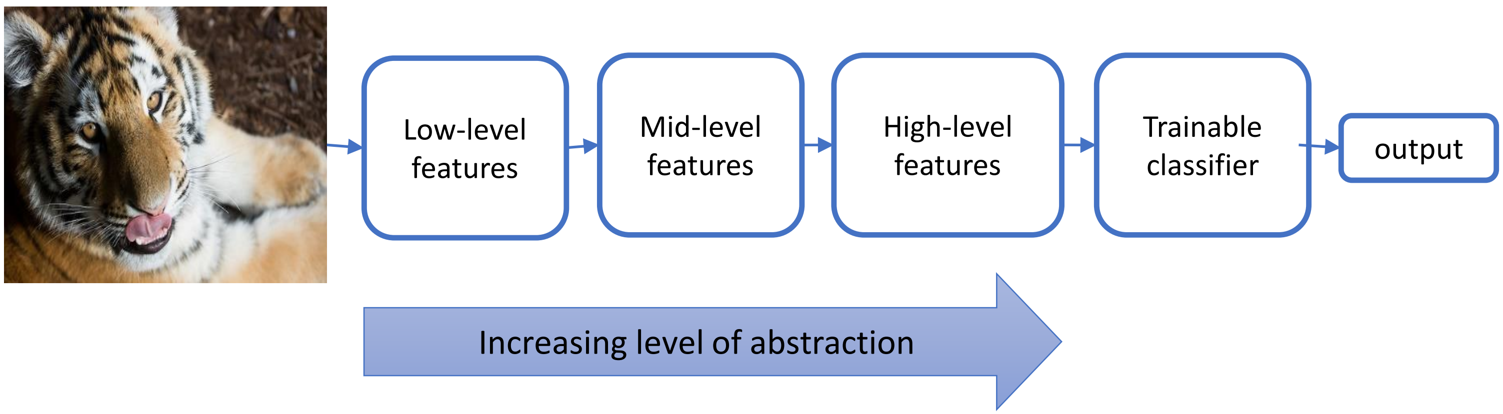
Theorem. A feedforward single hidden layer network with finite width can approximate continuous functions on compact subsets of $\mathbb{R}^{n}$ under mild assumptions on the activation function.

Deep Nets as Generalizer

- Deep learning (a.k.a. representation learning) seeks to learn rich hierarchical representations (i.e. features) automatically through multiple stage of feature learning process.



Learning Hierarchical Representations

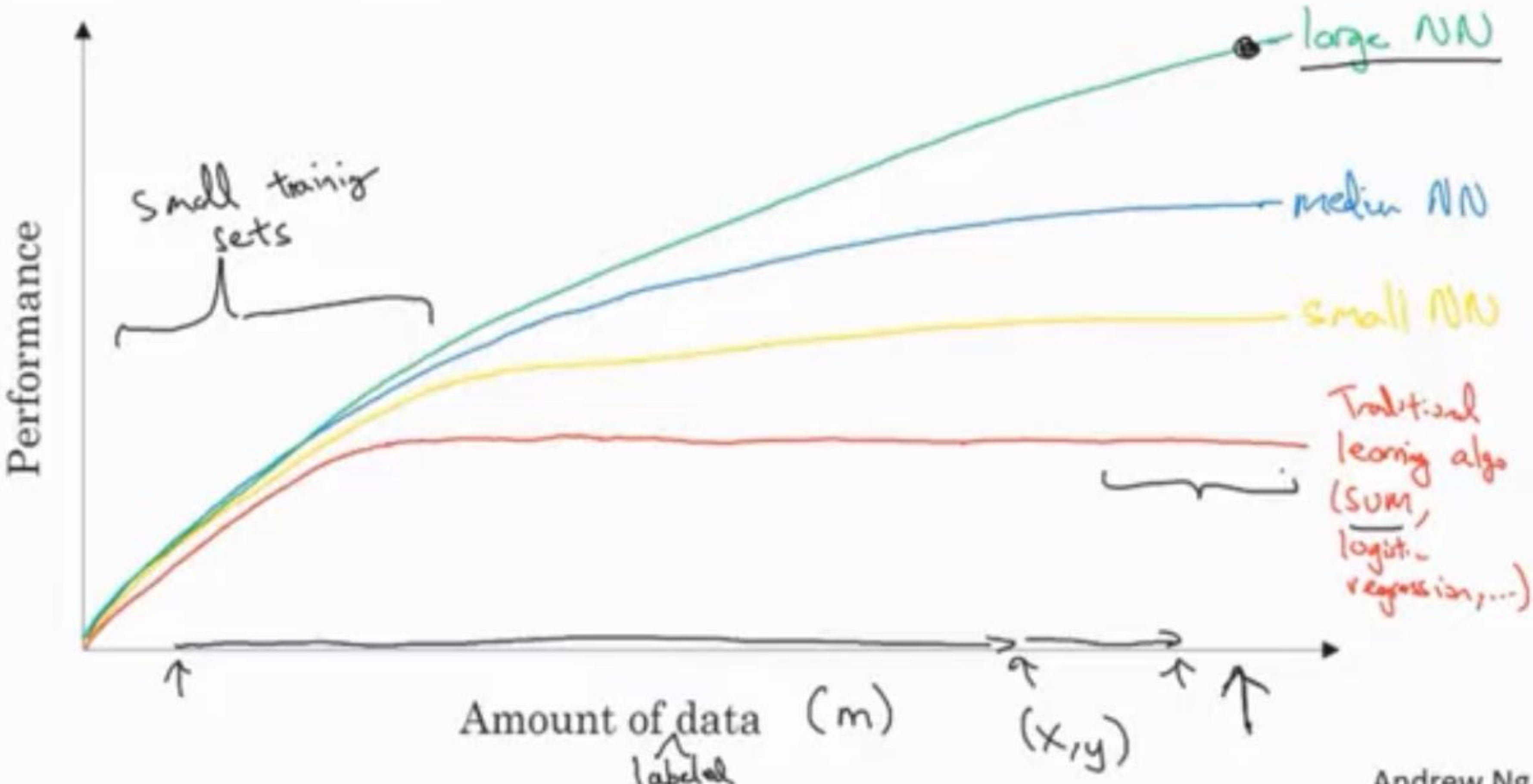


- Image recognition
  - Pixel → edge → texton → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story

Why there is a sudden surge in interest

- Research on deep neural networks almost abandoned from 2000-2006
  - Overfitting, extremely slow training, local minima, gradient vanishing/exploding
- In 2006, Hinton, et. al. proposed RBMs to pretrain a deep neural network
- In 2009, Raina, et. al. proposed to use GPUs to train deep neural network
- In 2010, Dahl, et. al. trained a deep neural network using GPUs to beat the state-of-the-art in speech recognition
- In 2012, Krizhevsky, et. al. won the ImageNet challenge with NN
- In 2012, Mikolov, et. al. trained a recurrent neural network to achieve state-of-the-art in language modelling

Scale drives deep learning progress



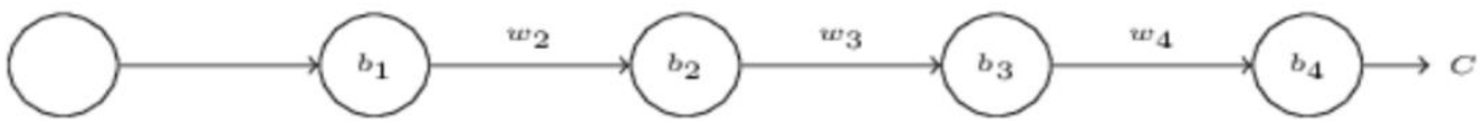
Andrew Ng

Challenges of Deep Learning

- Deep Neural nets are data hungry
- Vanishing/exploding gradients
- Overfitting
- Hyperparameter Optimization
- Requires high-performance hardware

Vanishing/Exploding Gradient

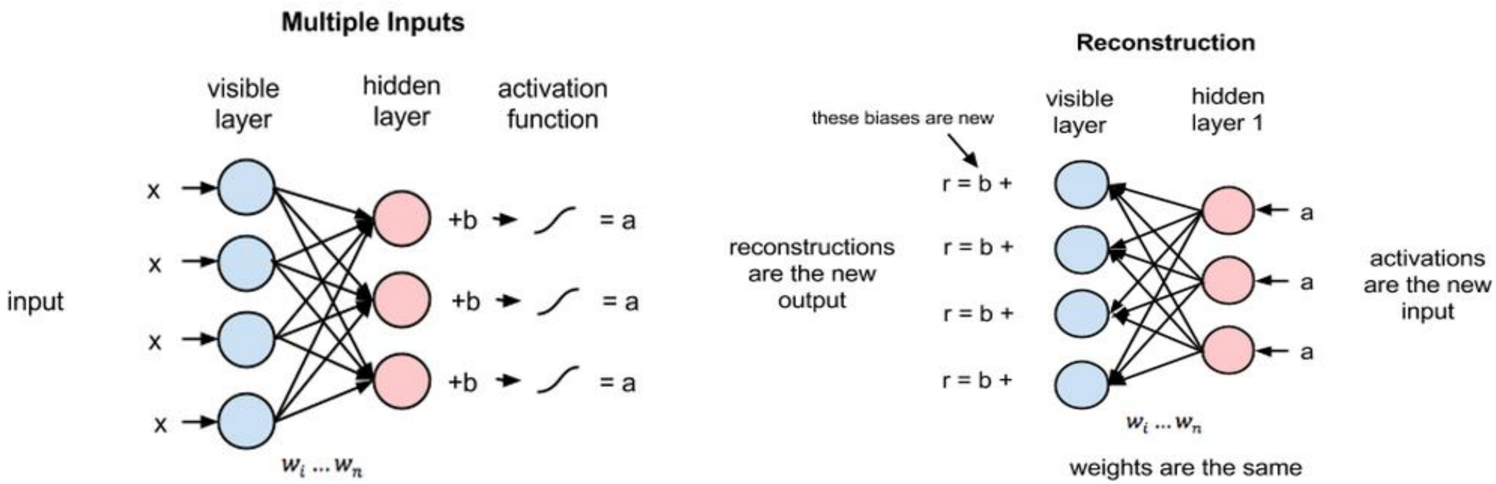
$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



Neural Network

- The sigmoid’(z1),sigmoid’(z2).. etc are less than ¼
- The weight matrices w1,w2,w3,w4 are initialized using gaussian method to have a mean of 0 and standard deviation of 1. Hence ||w(i)|| is less than 1
- If we initialize our weight matrices with very large values then gradient explodes and model becomes unusable

Restricted Boltzmann Machine



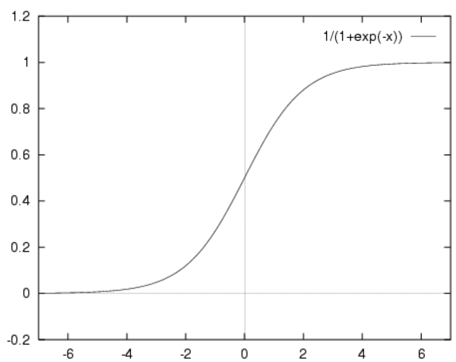
- Two-layered artificial neural network with generative capabilities
- Learns by minimizing reconstruction error

Pretraining DNN with RBMs

- The first layer RBM is trained using the data vectors v, resulting in parameters θ1
- . In a subsequent layer k, the activation probabilities Q of the hidden units in the k – 1'th layer are calculated by propagating the data vectors v through the layers already learnt, and are used as the training vectors for the k'th layer resulting in parameters θk
- This is greedy, layerwise and unsupervised pre-training.

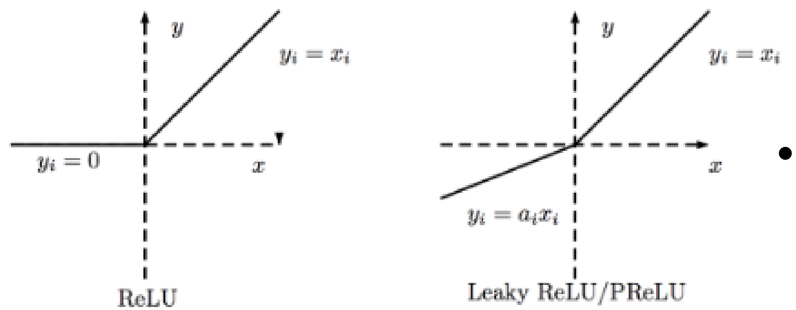
Activation Functions

- Sigmoid – $(1+e^{-x})^{-1}$



- Vanishing gradient
- Slow Convergence
- Saturate and kill gradients

- RELU –*max(0,x)*



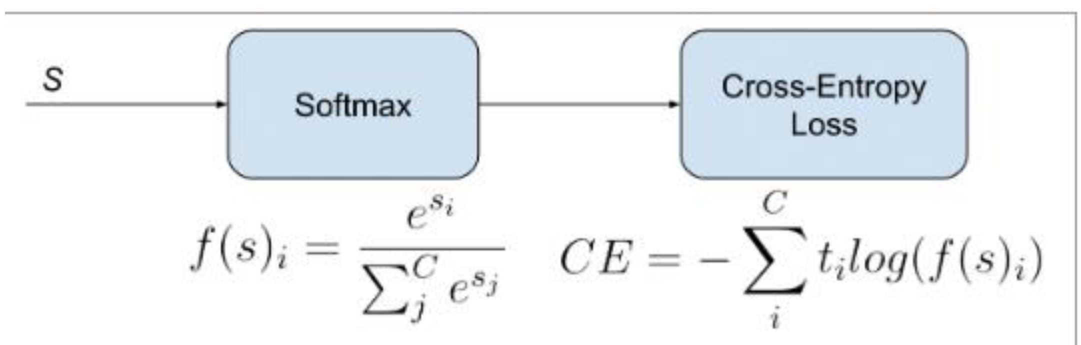
- Rectifies Vanishing gradient
- Leaky Relu resolves dead Neurons

- Softmax – $e^{z_j} / \sum_{k=1}^{K} e^{z_k}$

$$\begin{bmatrix}1.2\\0.9\\0.4\end{bmatrix} \xrightarrow{\text{Softmax}} \begin{bmatrix}0.46\\0.34\\0.20\end{bmatrix}$$

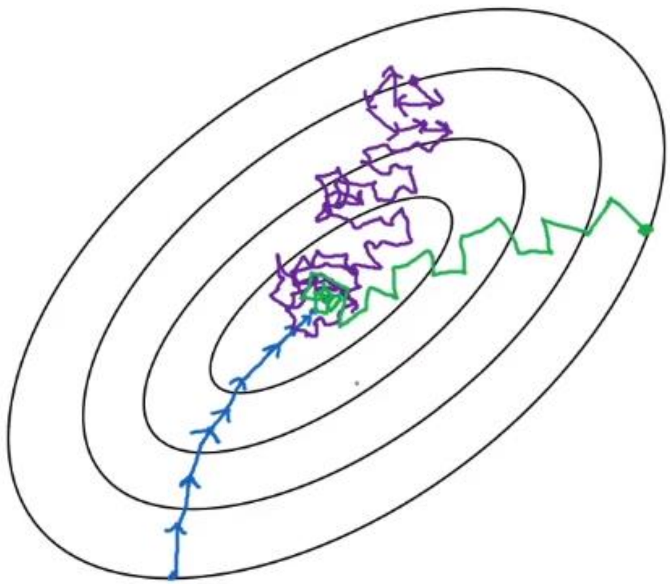
Loss and Cost Function

- MSE Loss = ½ (predicted value – actual value)^2
- Cost function (J) = 1/m (Sum of Loss error for ‘m’ examples)
- Binary cross entropy = – (Y log ($Y_{predicted}$) + (1-Y) log (1-$Y_{predicted}$)
- Categorical cross entropy



Mini Batch Gradient Descent

- Divide n training dataset into m batches with batch size n/m. Typical batch size ranges from 64 – 512.
- Update weight after processing each batch
- If batch size = 1 → Stochastic gradient descent
- Batch size = n → Batch gradient descent

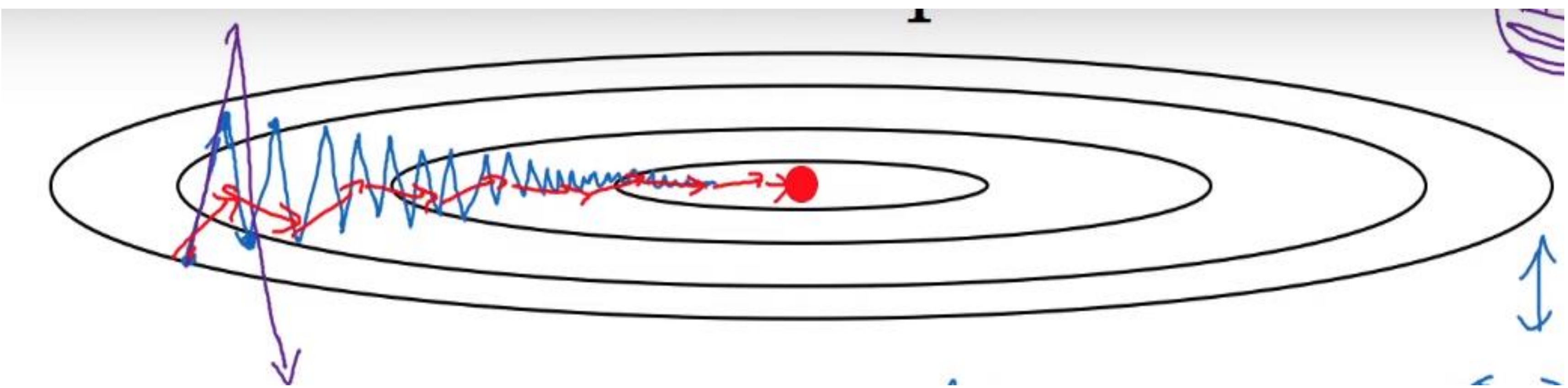


- Batch gradient → smooth but too long per iteration
- Stochastic gradient → can avoid local optima but too noisy
- Batch gradient → Fastest, avoid local optima, use vectorization less memory expensive

Gradient Descent with momentum

- Update weights with moving average of gradients
- V is momentum, gradient is acceleration and beta is friction

$$V_t = \beta V_{t-1} + (1-\beta)\nabla_w L(W, X, y)$$
$$W = W - \alpha V_t$$



RMSProp and Adam

- Adapt learning rate by root mean square of moving average of gradient

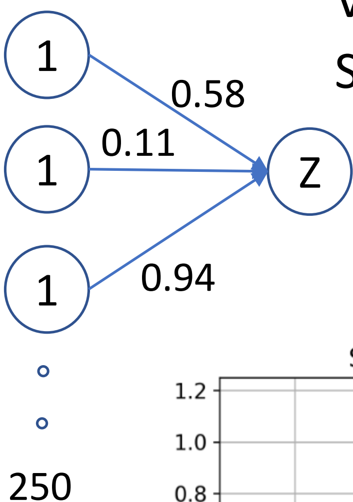
$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)\left(\frac{\delta C}{\delta w}\right)^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}} \frac{\delta C}{\delta w}$$

- Adam is adaptive moment estimation and is combination of momentum gradient descent and RMSprop

Weight Initialization

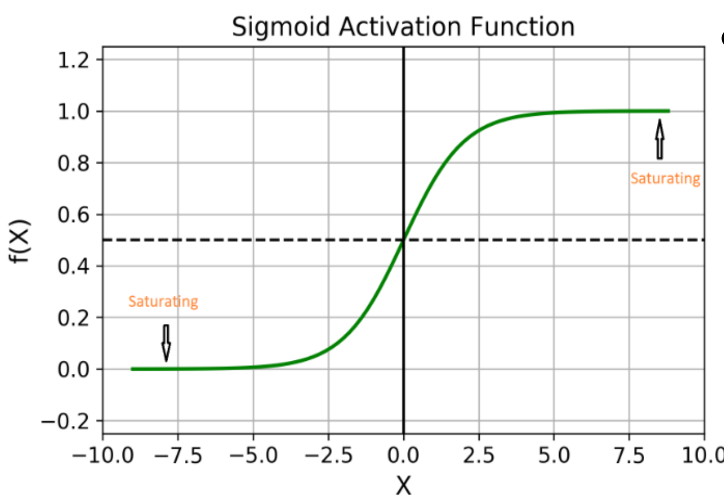
- Random Initialization



Var(Z) = 250

Std(z) = $\sqrt{250}$

= 15.811



- Xavier Initialization
  - $var(weights) = \frac{1}{N}$
  - $weight * \sqrt{\frac{1}{N}}$

- He Initialization
  - $weight * \sqrt{\frac{2}{N}}$

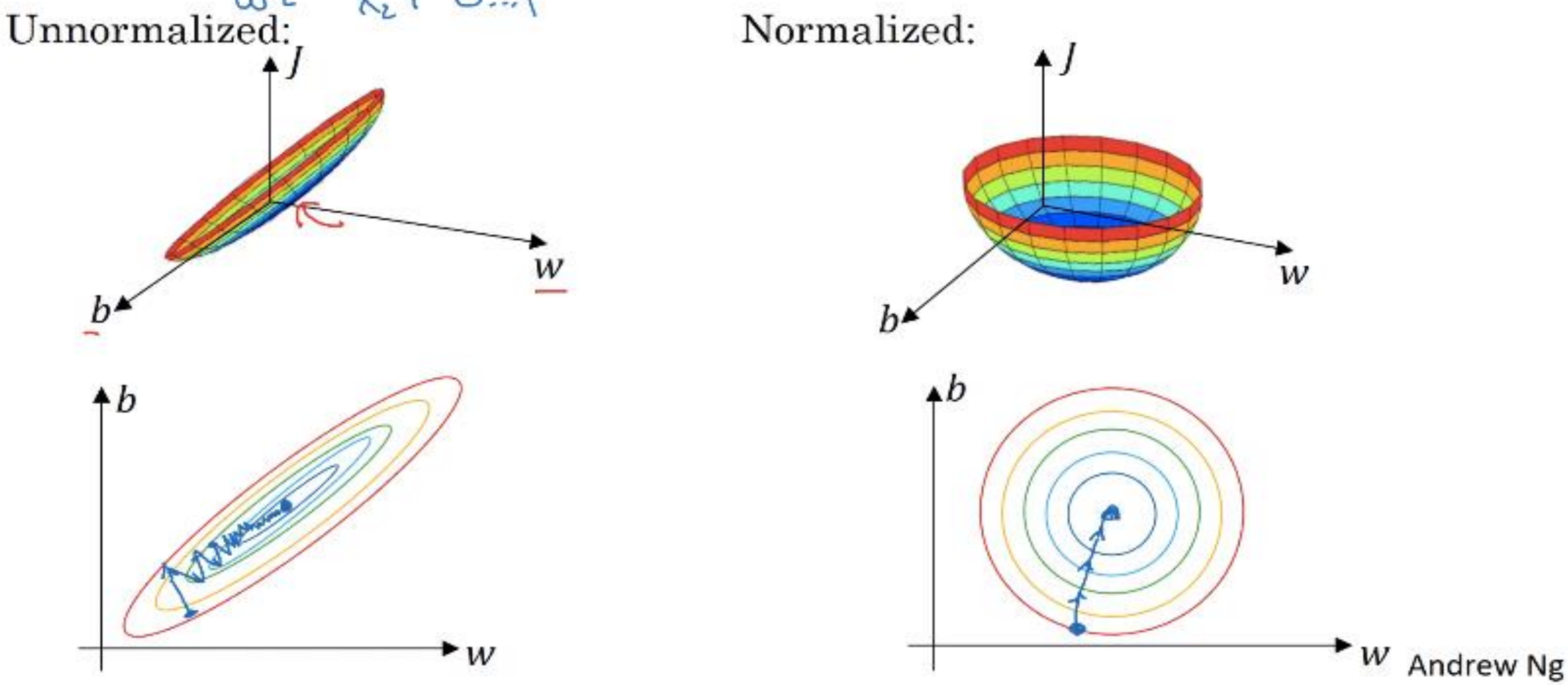
Data Normalization

Why normalize inputs?

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

$w_1 \quad x_1 : 1 \ldots 1000$

$w_2 \quad x_2 : 0 \ldots 1$



- Subtract mean and normalize variance
  - $x = x - \mu; \qquad x = \frac{x}{\sigma^2}$

Batch Normalization

- As the parameters of the preceding layers change, the distribution of inputs to the current layer such that the current layer needs to constantly readjust to new distributions
- Batch normalization accelerates training and allows using higher learning rate

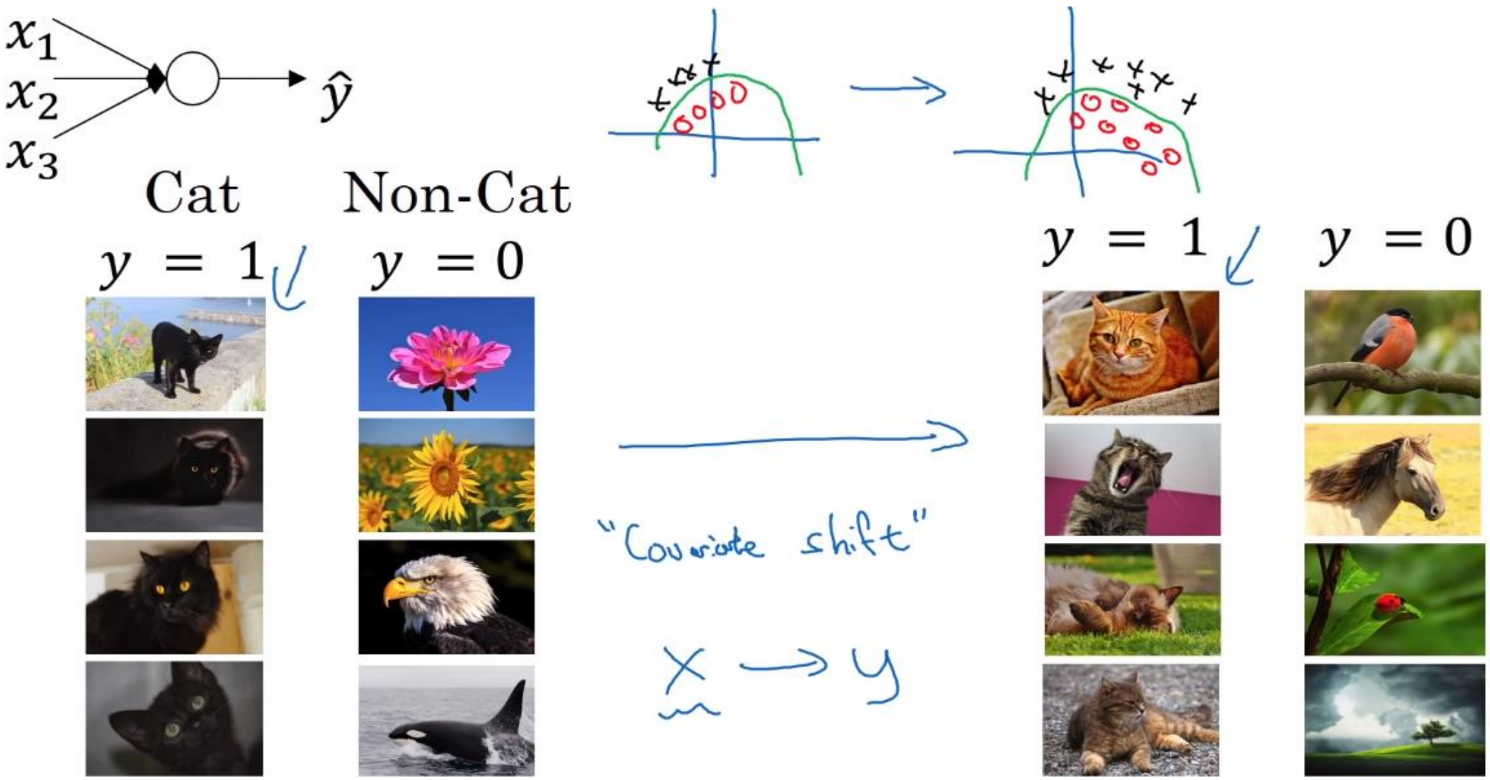
For a layer with d dimensional input, each dimension is normalized

$$\widehat{x_i}^{(k)} = \frac{x_i^{(k)} - \mu_B}{\sqrt{{\sigma_B^{(k)}}^2 + \epsilon}}$$

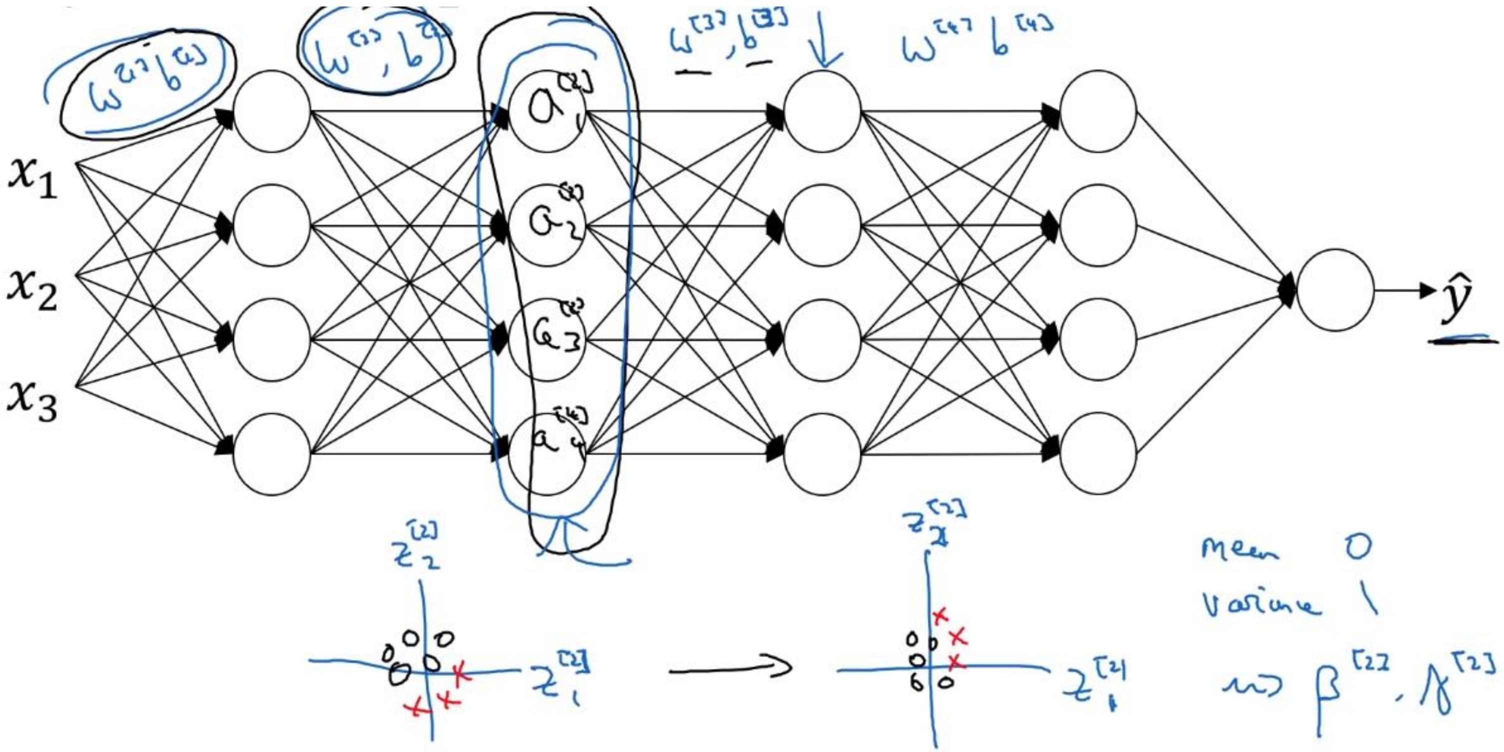
To restore the representation, the mean and variance is readjusted, where gamma and beta are learnable parameters

$$y_i^{(k)} = \gamma^k \widehat{x_i}^{(k)} + \beta^{(k)}$$

Covariate Shift

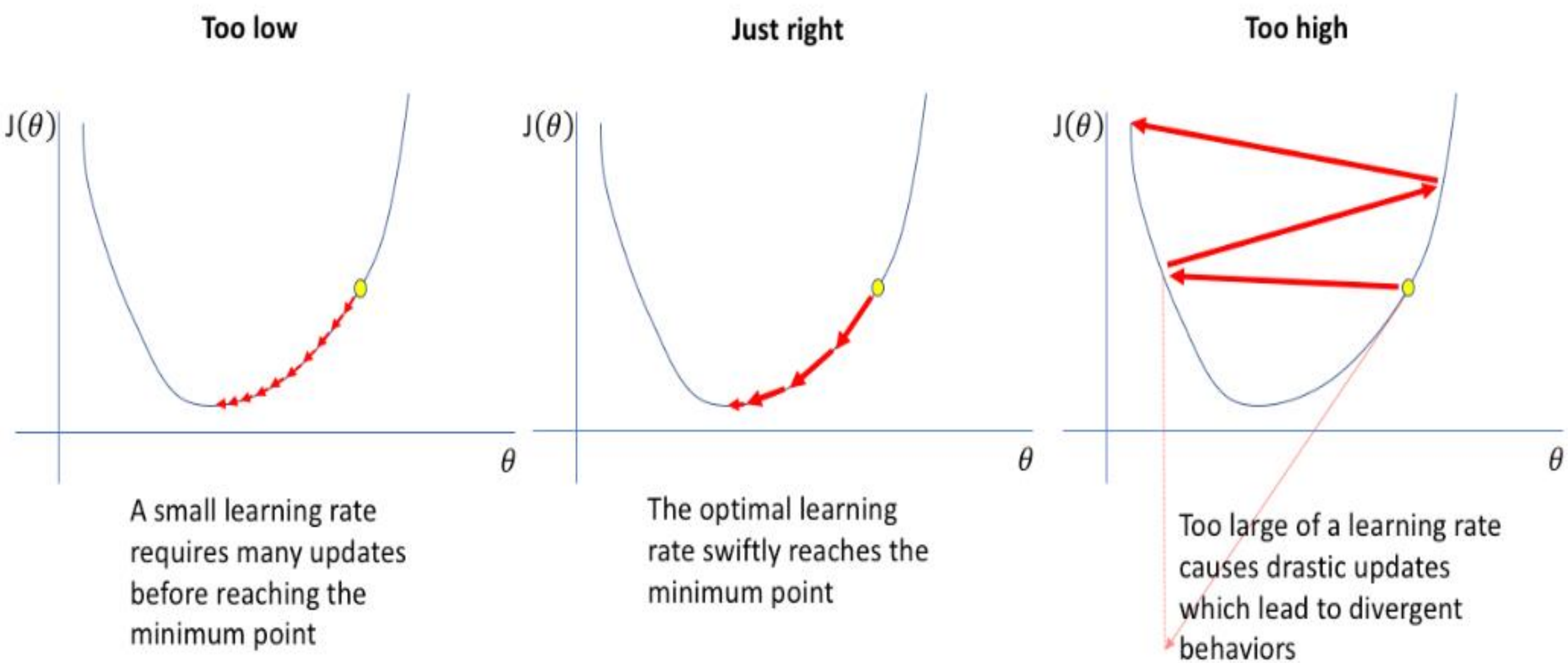


Why BatchNorm works



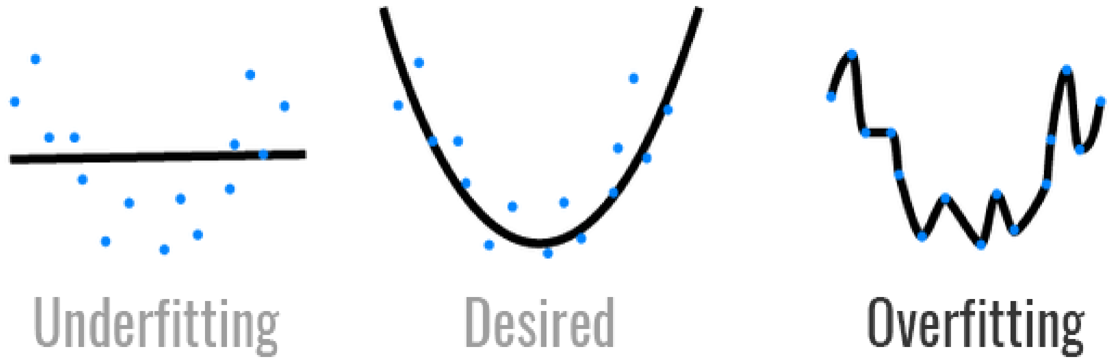
Andrew Ng

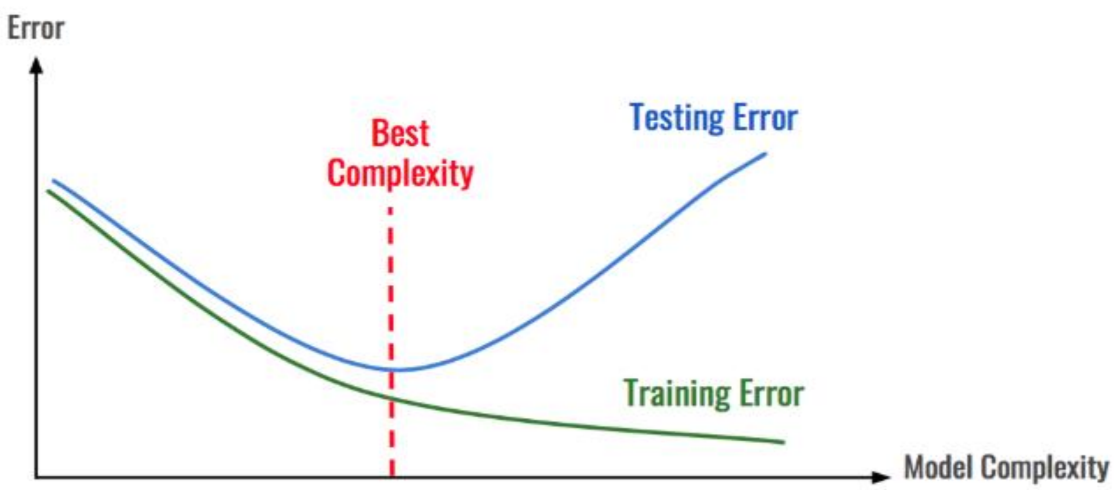
Learning rate decay



- Step decay with epochs
- Exponential decay with epochs

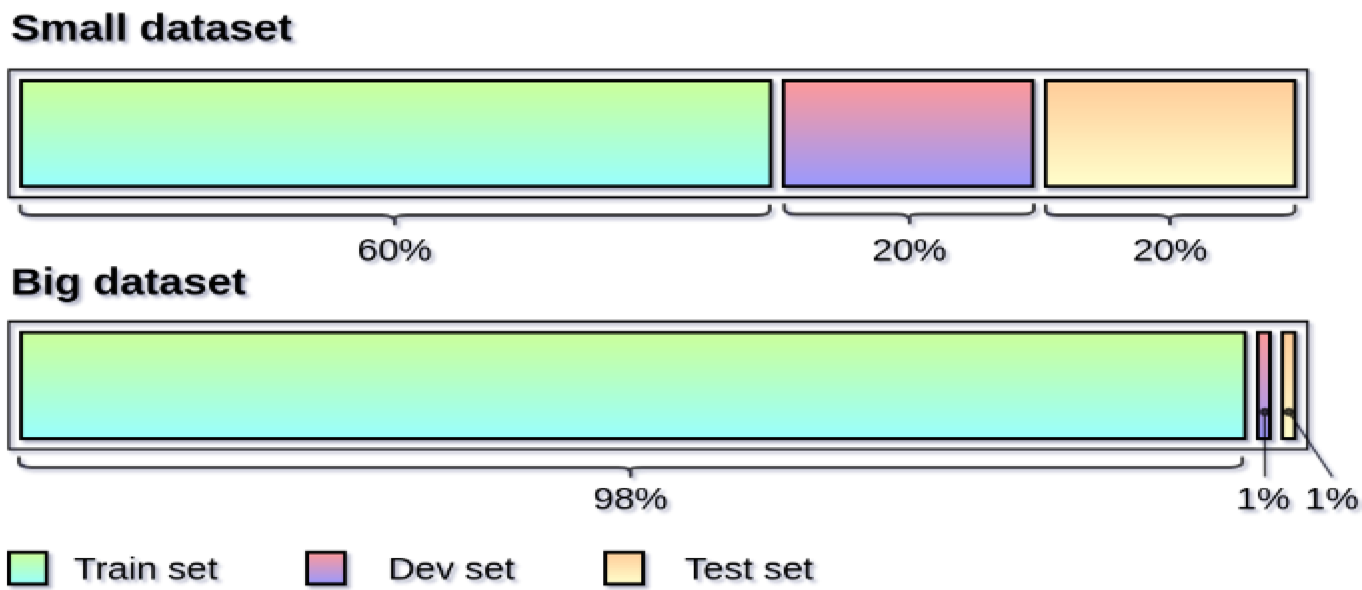
Overfitting in deep NN

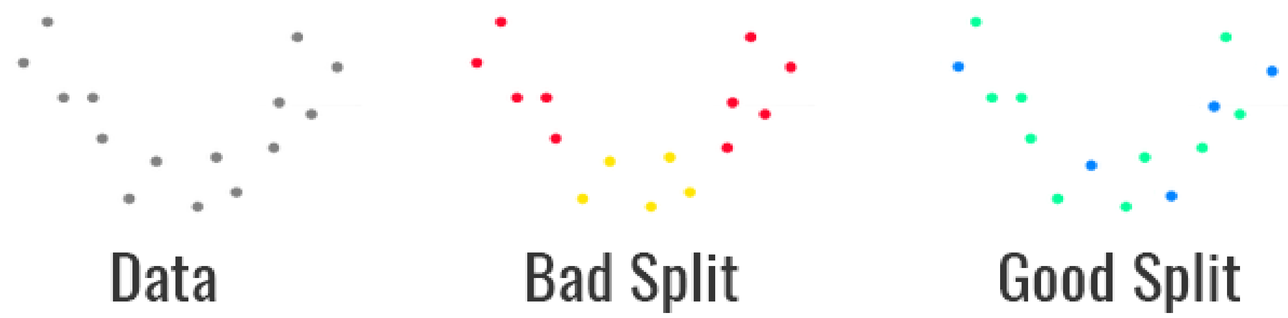




- Complex Deep NN can memorize complex features
- Memorization is not learning!

Data Split


Small dataset
60%
20%
20%
Big dataset
98%
1%
1%
Train set
Dev set
Test set


Data
Bad Split
Good Split

Detect Overfitting

	Low *Training* Error	High *Training* Error
Low *Testing* Error	The model is learning!	Probably some error in your code. Or you've created a *psychic* AI.
High *Testing* Error	OVERFITTING	The model is not learning.

Data Augmentation

- Gather more data
- Get diverse data
- Add noise

Each iteration sees as **different variation** of the original sample.

L1/L2 Regularization

- $Cost = Loss + \frac{\lambda}{2m}||w||^l \begin{cases} l = 2\ for\ L2 \\ l = 1\ for\ L1 \end{cases}$
- L2 regularization is also known as *weight decay* as it forces the weights to decay towards zero (but not exactly zero).
- Smaller weight leads to simpler model

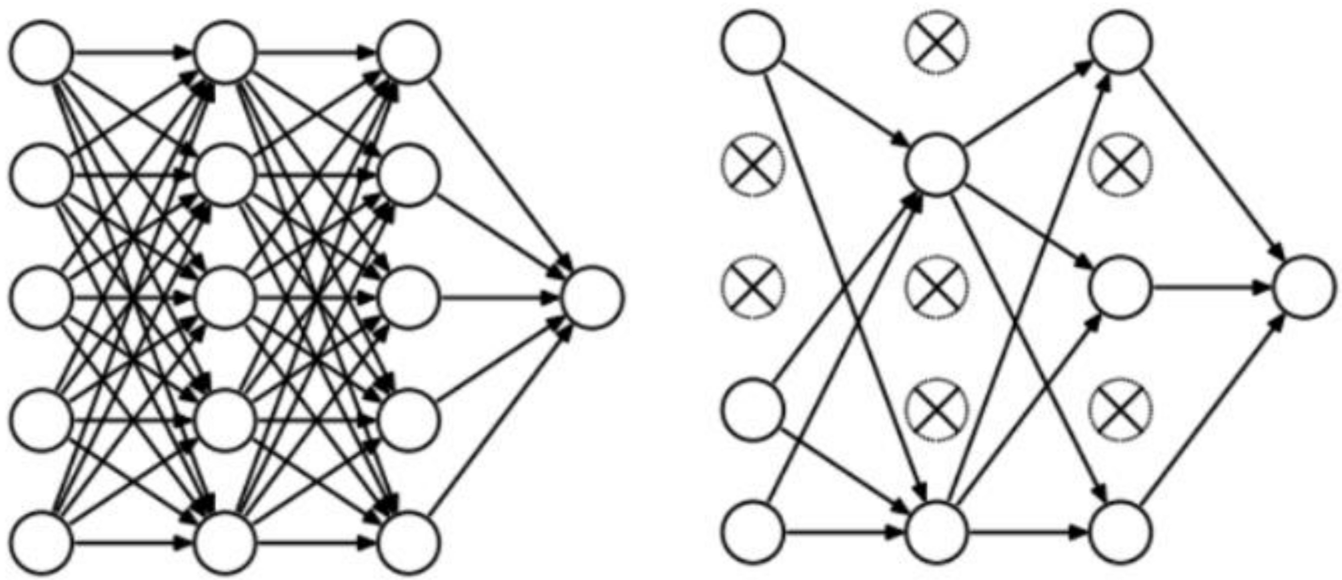
L1:

$$w_{\text{new}} = \begin{cases} (w - H) - \lambda, & w > 0 \quad \text{———} \quad (1.1) \\ (w - H) + \lambda, & w < 0 \quad \text{———} \quad (1.2) \end{cases}$$

L2:

$$w_{\text{new}} = (w - H) - 2\lambda w \quad \text{———} \quad (2)$$

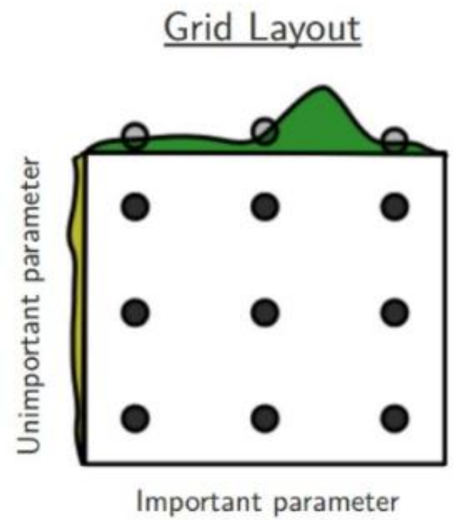
Dropout

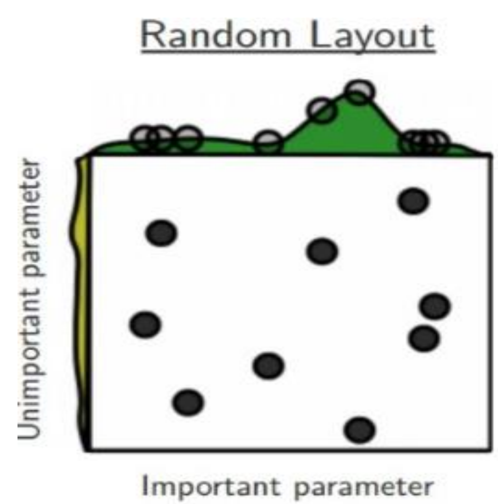


- At every iteration, it randomly selects some nodes and removes them along with all of their incoming and outgoing connections
- Probability of choosing how many nodes to be dropped is another hyperparameter

Hyperparameter Tuning

- Learning rate
- Beta of RMSProp
- Hidden layers config
- Learning rate decay
- Mini batch size
- Regularization parameters



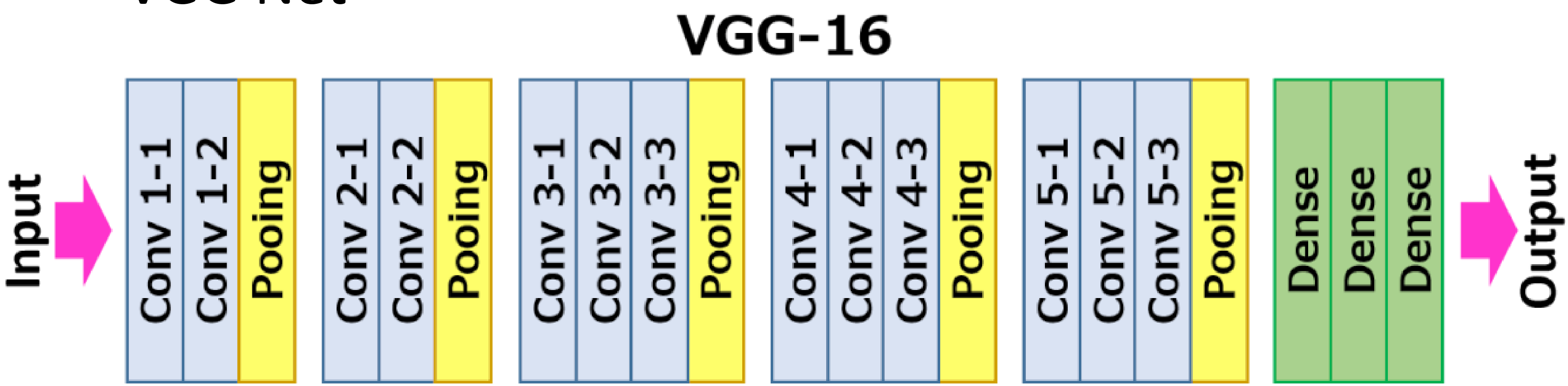


Training a Deep NN

- Data preprocess
  - Data augmentation and normalization
  - Create training, validation and test set
- Design Network
  - Layers selection and configuration
  - Activation, Regularization and BatchNormalization
  - Weight Initialization
- Training model
  - Select optimizer like RMSProp, Adam etc.
  - Set hyperparameters
  - Train and record validation accuracy
- Fine tune and repeat
  - Tune hyperparameters and design and retrain to improve accuracy
  - Stop when required accuracy level reached

State of Art Architectures

- LeNet – for OCR
  - Input ->Conv -> Relu -> Pool -> Conv -> Relu -> Pool -> FC ->Relu -> FC
- VGG Net



ResNet

- Motivation for skipping over layers is to avoid the problem of vanishing gradients, by reusing activations from a previous layer until the adjacent layer learns its weights
- Overcomes vanishing gradient for very deep nets
- Won Imagenet challenge on 2015

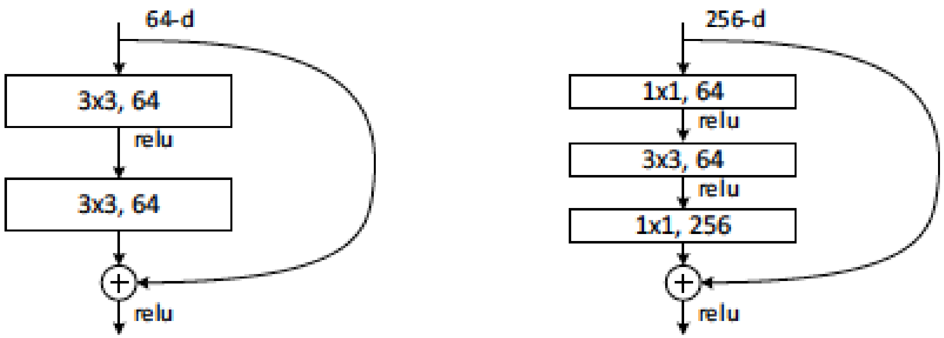


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

Thank You