Lecture 2

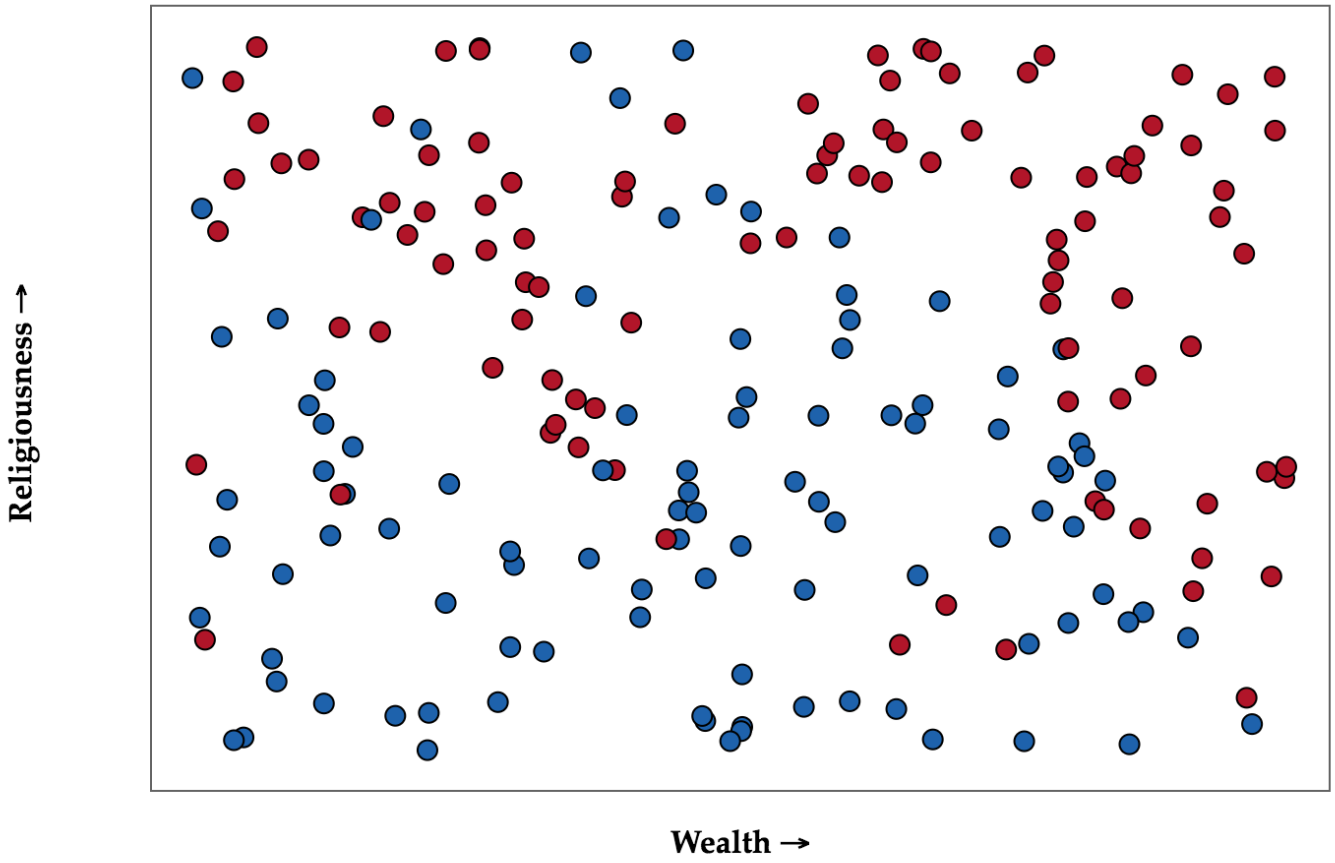
Neural Networks

David Gifford and Konstantin Krismer

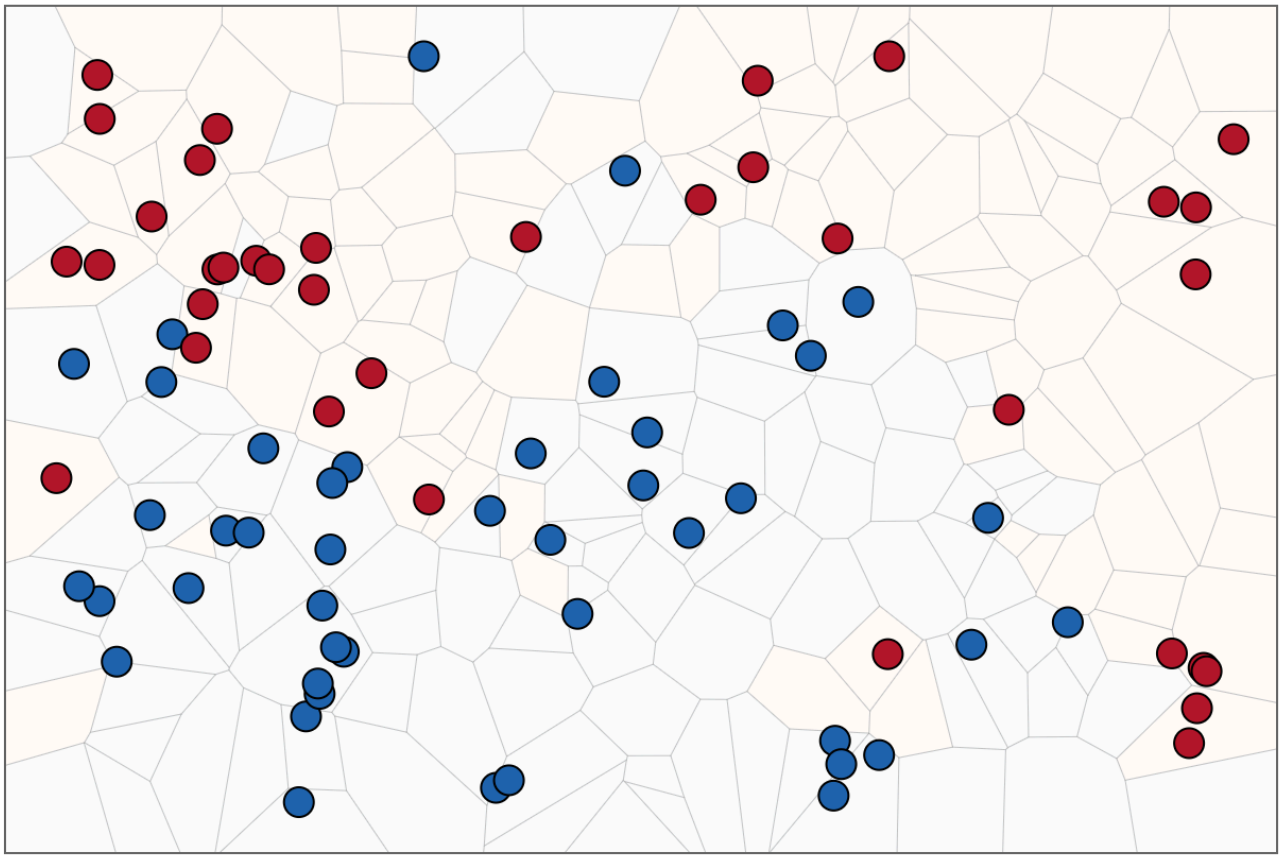
MIT - 6.802 / 6.874 / 20.390 / 20.490 / HST.506 - Spring 2020

2019-02-06

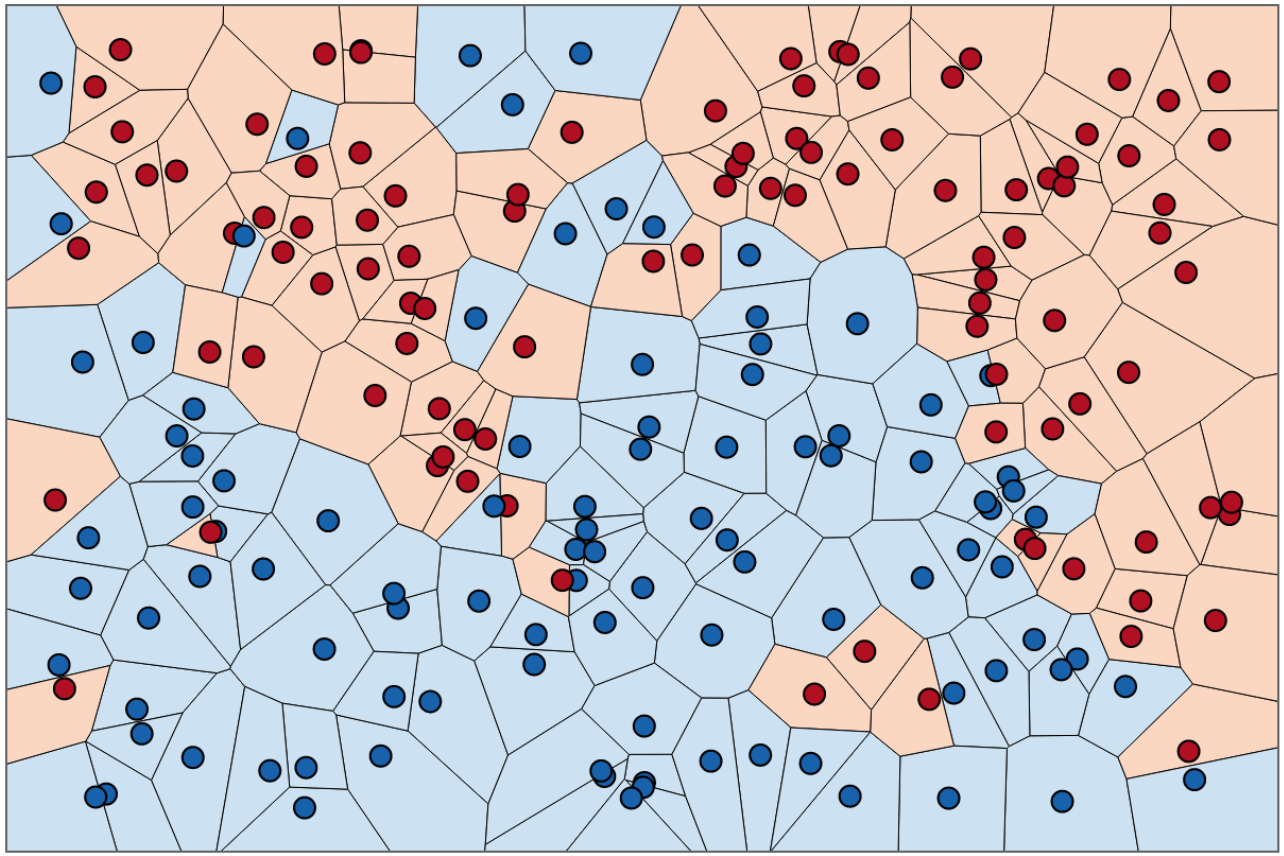
Training set: observations of wealth and religiousness features with #blue or #red labels



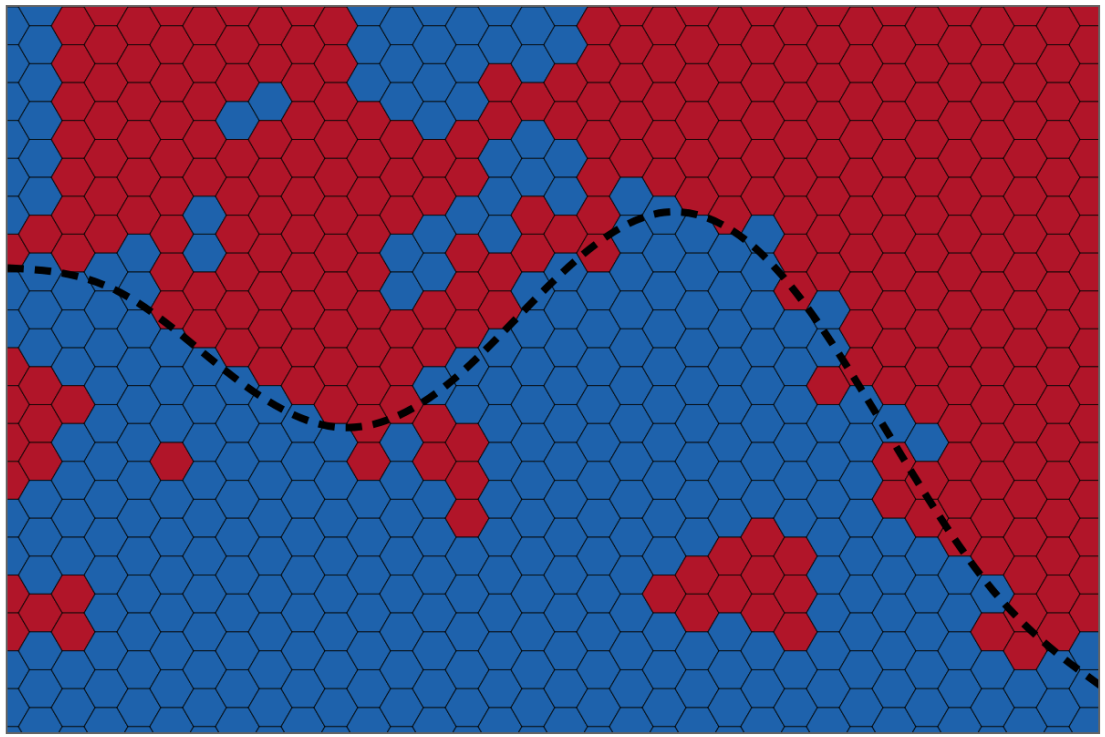
Test set: How well can we do?



K-Nearest Neighbors (KNN) defines neighborhoods

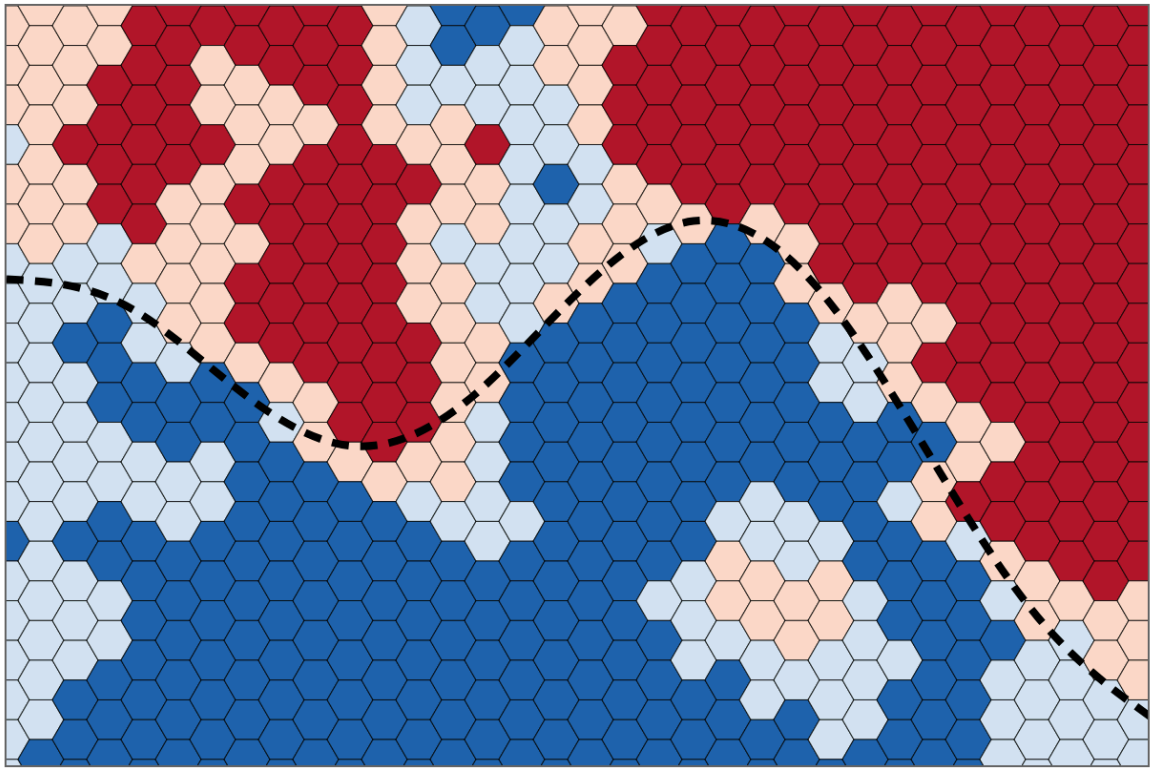


K-Nearest Neighbors (KNN) k=1 classification results



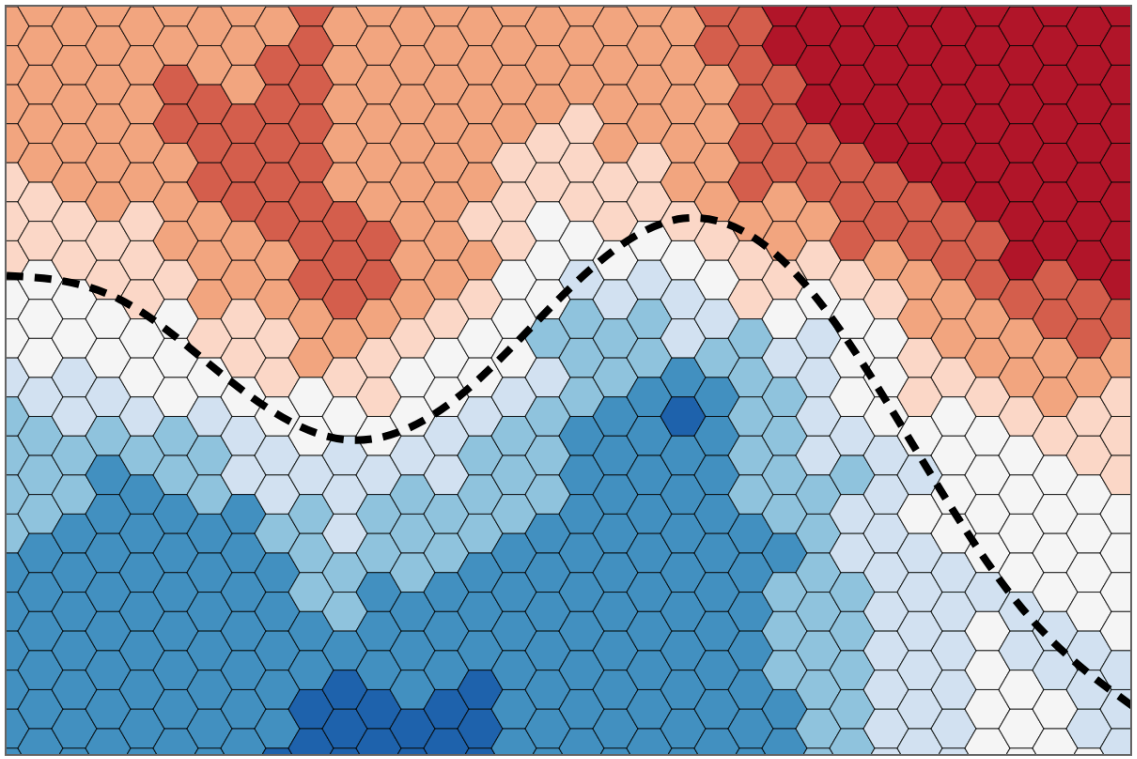
k-Nearest Neighbors: 1

K-Nearest Neighbors (KNN) k=3 classification results



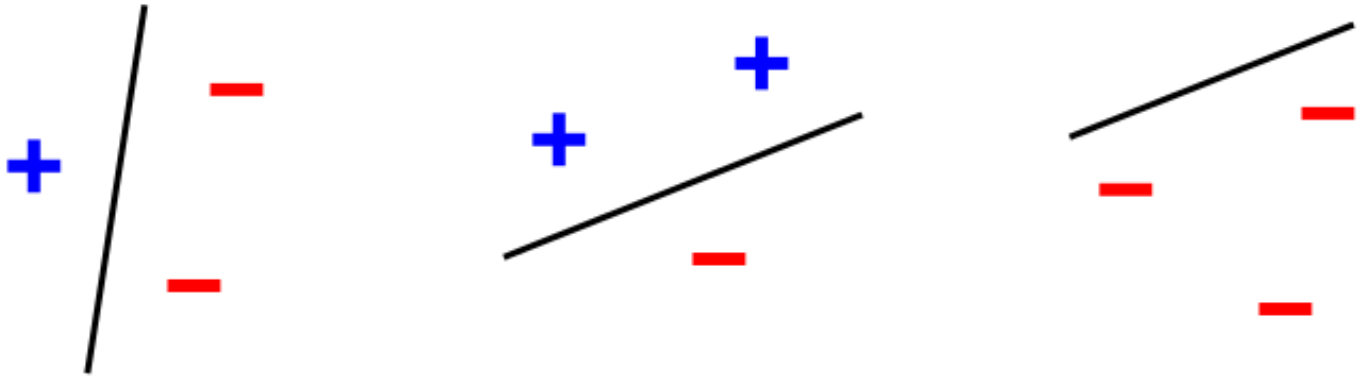
k-Nearest Neighbors: 3

K-Nearest Neighbors (KNN) k=29 classification results

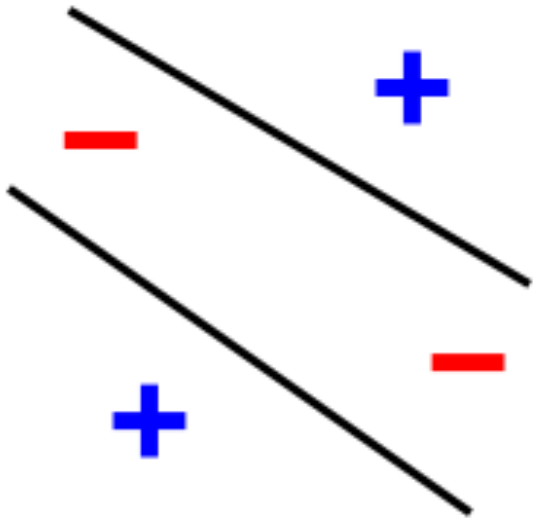


k-Nearest Neighbors: 29

A straight line can classify three points arbitrarily labeled



A straight line can not classify four points arbitrarily labeled



The **capacity** (Vapnik-Chervonenkis dimension) of a model describes how many points can be correctly predicted when they are produced by an adversary

The capacity of non-parametric models is defined by the size of their training set

- k-nearest neighbor (KNN) regression computes its output based upon the k “nearest” training examples
- Often the best method, and certainly a baseline to beat

The **generalizability** of a model describes its ability to perform well on previously unseen inputs

ML: Define $\mathcal{H}$, obtain $\hat{h}$

$\mathcal{H} \subset \mathcal{F}$, where $\mathcal{F}$ is the set of all functions mapping $\mathcal{X}$ onto $\mathcal{Y}$.

Goal: find $\hat{h} \in \mathcal{H}$

Step 1: define suitable hypothesis space $\mathcal{H}$

Problem Set 1

$$\boldsymbol{x} \in [0,1]^{784}$$
$$\hat{\boldsymbol{y}} \in [0,1]^{10}$$
$$\boldsymbol{W} \in \mathbb{R}^{784 \times 10}$$
$$\boldsymbol{b} \in \mathbb{R}^{10}$$
$$\phi_{\text{softmax}}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{|\boldsymbol{z}|} e^{z_j}}$$
$$h(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}) = \phi_{\text{softmax}}(\boldsymbol{W}^\intercal \boldsymbol{x} + \boldsymbol{b})$$
$$\mathcal{H} = \{h(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}) | \boldsymbol{W} \in \mathbb{R}^{784 \times 10}, \boldsymbol{b} \in \mathbb{R}^{10}\}$$

ML: Define $\mathcal{H}$, obtain $\hat{h}$

$\mathcal{H} \subset \mathcal{F}$, where $\mathcal{F}$ is the set of all functions mapping $\mathcal{X}$ onto $\mathcal{Y}$.

Goal: find $\hat{h} \in \mathcal{H}$

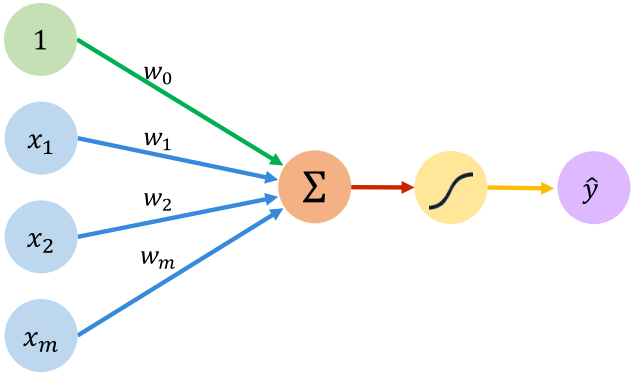
Step 1: define suitable hypothesis space $\mathcal{H}$

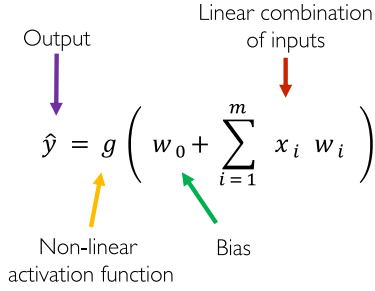
Problem Set 1

$$\boldsymbol{x} \in [0,1]^{784}$$
$$\hat{\boldsymbol{y}} \in [0,1]^{10}$$
$$\boldsymbol{W} \in \mathbb{R}^{784 \times 10}$$
$$\boldsymbol{b} \in \mathbb{R}^{10}$$
$$\phi_{\text{softmax}}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{|\boldsymbol{z}|} e^{z_j}}$$
$$h(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}) = \phi_{\text{softmax}}(\boldsymbol{W}^\intercal \boldsymbol{x} + \boldsymbol{b})$$
$$\mathcal{H} = \{h(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}) | \boldsymbol{W} \in \mathbb{R}^{784 \times 10}, \boldsymbol{b} \in \mathbb{R}^{10}\}$$

Step 2: use gradient-based optimization methods to obtain $\hat{h}$

Perceptron





Illustrations by courtesy of [Amini and Soleimany, 2019].

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}. \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

only in output layer:

- linear activation function (for regression)
- sigmoid activation function (for binary or multi-label classification)
- softmax activation function (for binary or multi-class classification)

Activation functions

- softmax (identical to sigmoid in binary classification)
- exponential linear unit (ELU)

$$\phi(z, \alpha)_{\text{ELU}} = \begin{cases} z, & \text{for } z \geq 0 \\ \alpha(e^z - 1), & \text{for } z < 0 \end{cases}$$

- scaled exponential linear unit (SELU)

$$\phi(z, \alpha, \beta) = \beta * \phi(z, \alpha)_{\text{ELU}}$$

- softsign

$$\phi(z) = \frac{z}{|z| + 1}$$

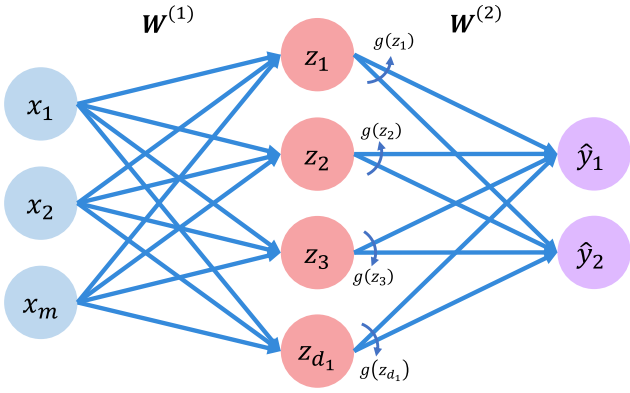
- leaky ReLU

$$\phi(z, \alpha) = \begin{cases} z, & \text{for } z \geq 0 \\ \alpha z, & \text{for } z < 0 \end{cases}$$

- parametric ReLU (PReLU, identical to leaky ReLU, but α is learned)

$$\phi(z; \alpha) = \begin{cases} z, & \text{for } z \geq 0 \\ \alpha z, & \text{for } z < 0 \end{cases}$$

Single-layer neural network

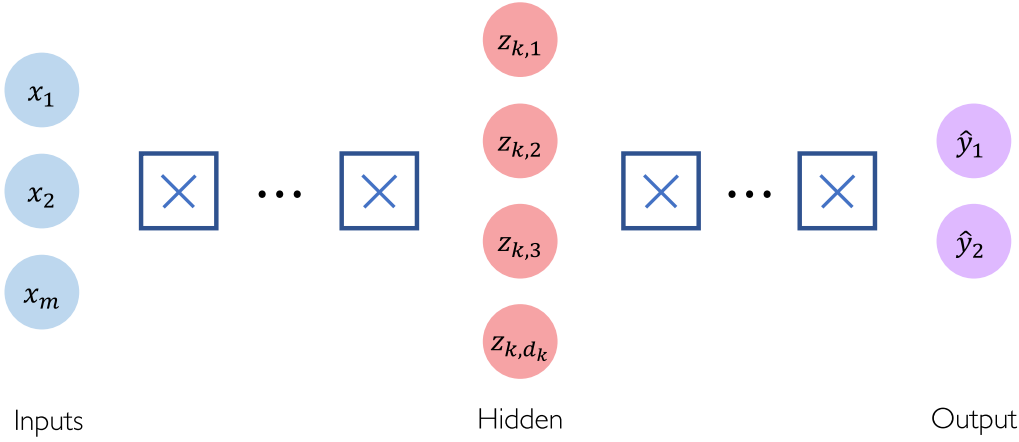


Inputs Hidden Final Output

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, w_{j,i}^{(2)}\right)$$

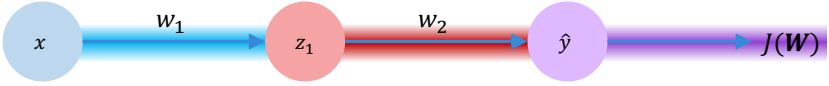
Multi-layer neural network

How deep is deep learning?



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Backpropagation by the chain rule

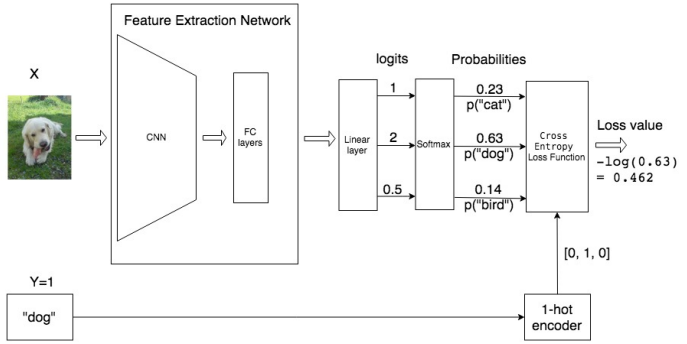


$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for ***every weight in the network*** *using gradients from later layers*

Illustrations by courtesy of [Amini and Soleimany, 2019].

Example loss computation



Loss Computation shows $y^{(i)}$ is a selector

$$\mathcal{L}_{\text{CCE}}(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)}) = -\sum_{j=1}^{K} y_j^{(i)} \log(\hat{y}_j^{(i)}),$$

$y_j^{(i)} = 1$ only if $\boldsymbol{x}^{(i)}$ belongs to class j and otherwise $y_j^{(i)} = 0$ therefore loss is only realized for class j

Derivative loss with respect to $z_j^{(i)}$

Let c be the correct class where the loss is paid

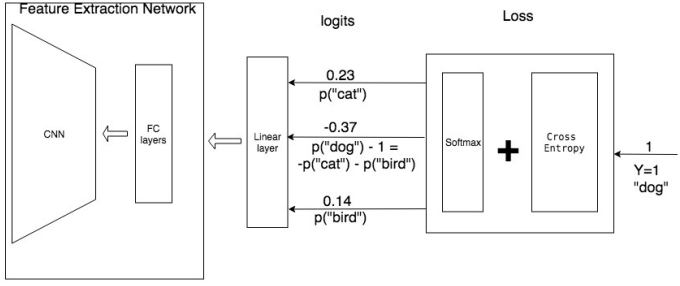
$$\mathcal{L}_{\text{CCE}}(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)}) = -\sum_{j=1}^{K} y_j^{(i)} \log(\hat{y}_j^{(i)}),$$

$$\text{where} \quad \hat{y}_j^{(i)} = \frac{\exp(z_j^{(i)})}{\sum_{k=1}^{K} \exp(z_k^{(i)})}$$

$$\frac{\partial \mathcal{L}_{\text{CCE}}}{\partial z_j^{(i)}} = \frac{\partial}{\partial z_j^{(i)}}(-z_c^{(i)} + \log \sum_{k=1}^{K} \exp(z_k^{(i)}))$$

$$\frac{\partial \mathcal{L}_{\text{CCE}}}{\partial z_j^{(i)}} = \hat{y}_j^{(i)} - 1(j = c)$$

Backpropagated gradients



Gradients arise from true class

$$\frac{\partial \mathcal{L}_{\text{CCE}}}{\partial z_j^{(i)}} = \hat{y}_j^{(i)} - 1(j = c)$$

http://machinelearningmechanic.com/assets/pdfs/cross_entropy_loss_derivative.pdf

Updating the weights from gradients

$$\boldsymbol{z} = \boldsymbol{W}^{\mathsf{T}}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{z}_j^{(i)} = (\sum_{k=1}^{N} \boldsymbol{W}_{jk}\boldsymbol{x}_k^{(i)}) + \boldsymbol{b}_j$$

$$\frac{\partial \mathcal{L}_{\mathsf{CCE}}(\boldsymbol{\theta})}{\partial \boldsymbol{W}_{jk}} = \frac{\partial \mathcal{L}_{\mathsf{CCE}}(\boldsymbol{\theta})}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial \boldsymbol{W}_{jk}}$$

$$\frac{\partial \mathcal{L}_{\mathsf{CCE}}(\boldsymbol{\theta})}{\partial \boldsymbol{W}_{jk}} = \frac{\partial \mathcal{L}_{\mathsf{CCE}}(\boldsymbol{\theta})}{\partial z^{(i)}} \boldsymbol{x}_k$$

$$\boldsymbol{W}_{jk}^{t} = \boldsymbol{W}_{jk}^{t-1} - \eta(\hat{y}_j^{(i)} - 1(j = c))\boldsymbol{x}_k$$

η is the learning rate in this example

Vanishing gradient problem

Higher layers that have small derivatives cause exponential decay of gradients towards the input layers of a network

This is a consequence of the chain rule

Fully connected layer in TF

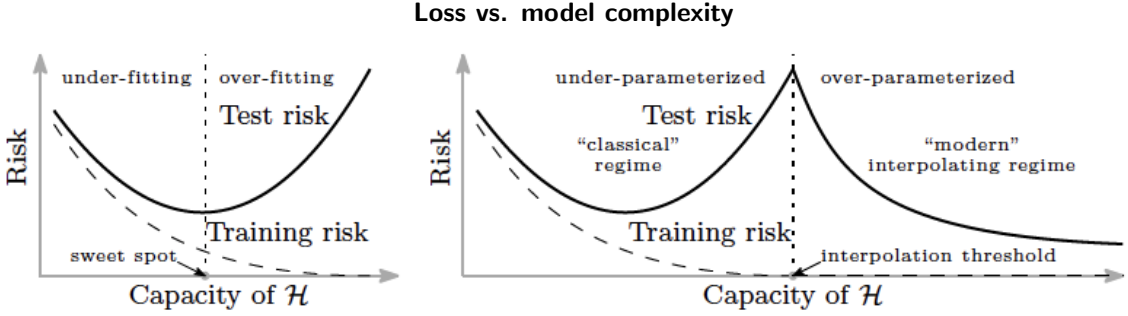
Listing 1: Problem Set 1 TF (`tf.keras.*`)

```
# Categorical cross-entropy loss
def loss_fn(y_pred, y_true):
    return tf.math.reduce_mean(-tf.math.reduce_sum(y_true * tf.math.log(y_pred), axis = 1)


# We can select from a library of gradient management methods here
learning_rate = 0.01
optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate)


# A training step
def train_step(inputs, labels, opt):
    with tf.GradientTape() as tape:
        predictions = model(inputs)
        pred_loss = loss_fn(predictions, labels)
        dloss_dw, dloss_db = tape.gradient(pred_loss, [W,b])
    opt.apply_gradients(zip([dloss_dw, dloss_db], [W,b]))
    return pred_loss
```

How complex should our models be? - Overfitting risk

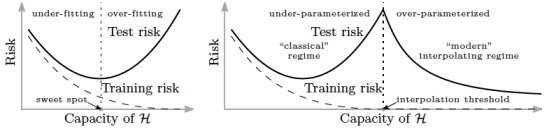


We need to control model complexity to reduce overfitting and enable generalization to unseen examples

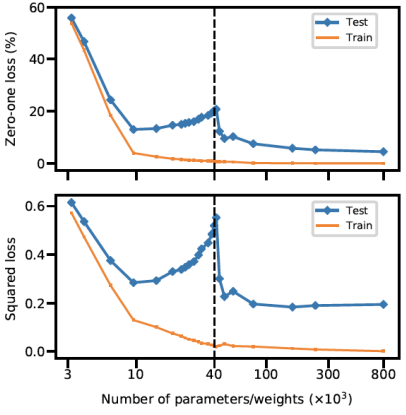
Note hypothesis that deep learning includes an interpolating region that reduces loss past some model capacity
https://arxiv.org/abs/1812.11118

How complex should our models be? - Overfitting risk

Loss vs. model complexity



Loss vs. model complexity MNIST fully connected



https://arxiv.org/abs/1812.11118

Overfitting can also be the result of training too long

Training set ($\mathcal{S}_{\text{training}}$):

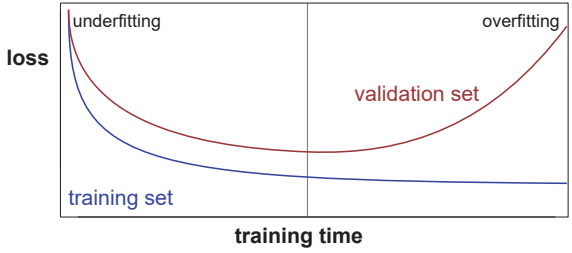
- set of examples used for learning
- usually 60 - 80 % of the data

Validation set ($\mathcal{S}_{\text{validation}}$):

- set of examples used to tune the model hyperparameters
- usually 10 - 20 % of the data

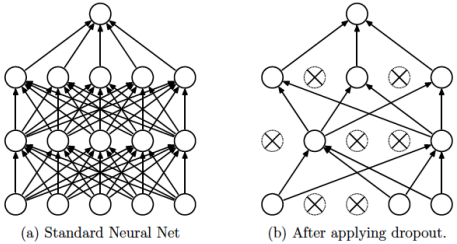
Test set ($\mathcal{S}_{\text{test}}$):

- set of examples used only to assess the performance of fully-trained model
- after assessing test set performance, model must not be tuned further
- usually 10 - 30 % of the data



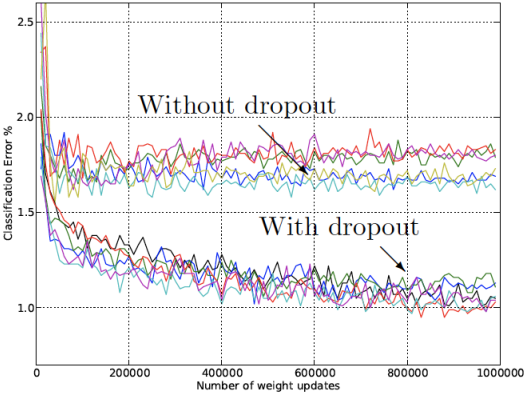
Dropout Regularization

Dropout



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

MNIST dropout results (varying architectures)



Dropout rate r is probability a node will be eliminated during a given minibatch
We train a thinned network on every minibatch and average the results
Retained weights are scaled by $1/(1-r)$
http://jmlr.org/papers/v15/srivastava14a.html

Weight Regularization

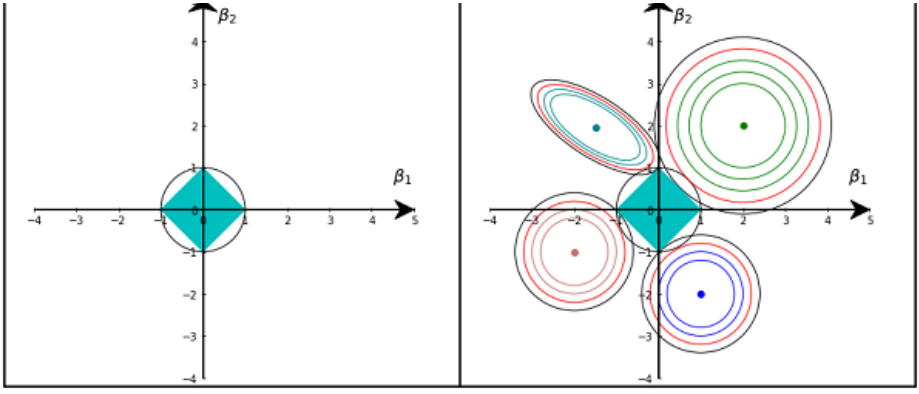
$$\mathcal{L}_{\text{L1-R}}(\theta) = \mathcal{L}_{\text{CCE}}(\theta) + \lambda \sum_{i=1}^{N} |\theta_i| \qquad \text{(L1 Regularization)}$$

$$\mathcal{L}_{\text{L2-R}}(\theta) = \mathcal{L}_{\text{CCE}}(\theta) + \lambda \sum_{i=1}^{N} \theta_i^2 \qquad \text{(L2 Regularization)}$$

$$||\theta_i||_2 \leq c \qquad \text{(Max Norm Regularization)}$$

We use regularization to control the complexity of models

Graphical Interpretation of L1 and L2 Regularization



$$\mathcal{L}_{\text{L1-R}}(\theta) = \mathcal{L}_{\text{CCE}}(\theta) + \lambda \sum_{i=1}^{N} |\theta_i| \qquad \text{(L1 Makes Sparse)}$$

$$\mathcal{L}_{\text{L2-R}}(\theta) = \mathcal{L}_{\text{CCE}}(\theta) + \lambda \sum_{i=1}^{N} \theta_i^2 \qquad \text{(L2 Minimizes Magnitude)}$$

L1 and L2 Regularization Gradient Changes

$$\mathcal{L}_{\text{L1-R}}(\theta) = \mathcal{L}_{\text{CCE}}(\theta) + \lambda \sum_{i=1}^{N} |\theta_i| \qquad \text{(L1 Regularization)}$$

$$\mathcal{L}_{\text{L2-R}}(\theta) = \mathcal{L}_{\text{CCE}}(\theta) + \lambda \sum_{i=1}^{N} \theta_i^2 \qquad \text{(L2 Regularization)}$$

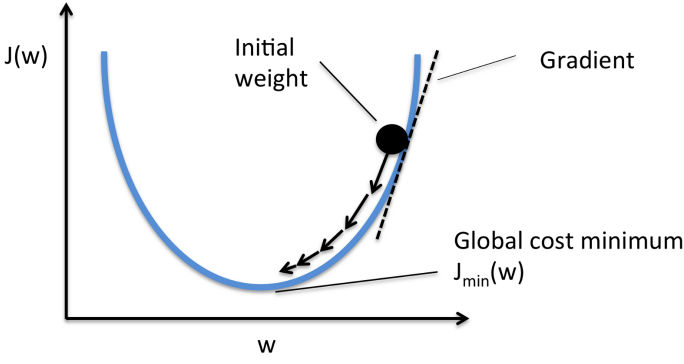
$$\frac{\partial \mathcal{L}_{\text{L1-R}}(\theta)}{\partial \theta_i} = \frac{\partial \mathcal{L}_{\text{CCE}}(\theta)}{\partial \theta_i} + \lambda \qquad (\theta_i > 0)$$

$$\frac{\partial \mathcal{L}_{\text{L1-R}}(\theta)}{\partial \theta_i} = \frac{\partial \mathcal{L}_{\text{CCE}}(\theta)}{\partial \theta_i} - \lambda \qquad (\theta_i < 0)$$

$$\frac{\partial \mathcal{L}_{\text{L2-R}}(\theta)}{\partial \theta_i} = \frac{\partial \mathcal{L}_{\text{CCE}}(\theta)}{\partial \theta_i} + 2\lambda\theta_i$$

Optimizing objective function

Gradient descent



- initialize model parameters $\theta_0, \theta_1, ..., \theta_m$
- repeat until converge, for all θ_i

$$\theta_i^t \leftarrow \theta_i^{t-1} - \eta \frac{\partial}{\partial \theta_i^{t-1}} \mathcal{J}(\Theta),$$

where the objective function $\mathcal{J}(\Theta)$ is evaluated over all training data $\{(\mathbf{X}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$.

Problem Set 1

Stochastic Gradient Descent (SGD): in each step, randomly sample a mini-batch from the training data and update the parameters using gradients calculated from the mini-batch only.

Example convex functions

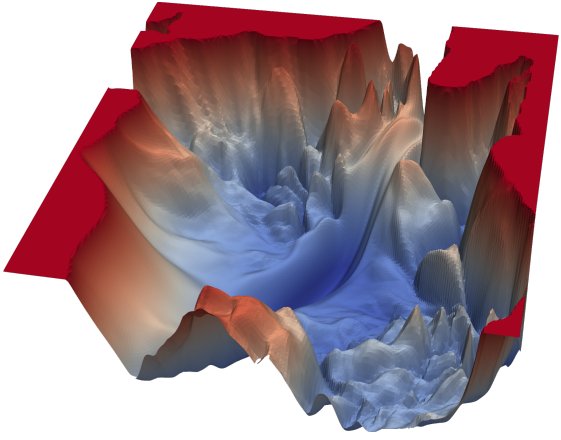
$c(x) = Mx + b$
$c(x) = e^{c_1(x)}$
$c(x) = c_1(x) + c_2(x)$
$c(x) = |x|$
$c(x) = max(c_1(x), c_2(x))$

When you compose convex functions the result is not necessarily convex

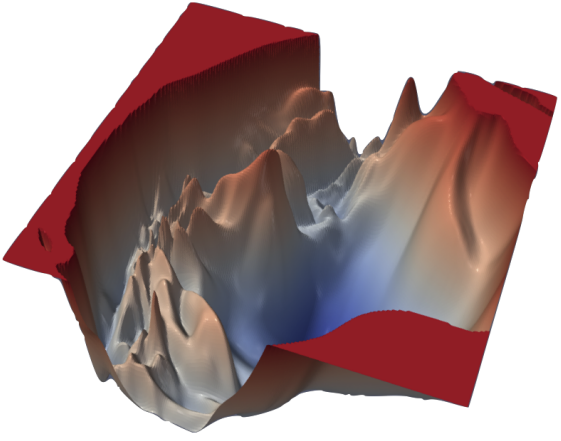
We are working in the domain of non-convex optimization

Flying into the danger zone

VGG-56 Loss Landscape



VGG-110 Loss Landscape



https://www.cs.umd.edu/ tomg/projects/landscapes/

Gradient Management Methods

- Stochastic gradient descent (update on each training value)
- Mini-batch gradient descent (average gradient over a mini-batch)
- Momentum methods
  - Adam - exponentially decaying average of past gradients and parameter specific learning rates
  - Adadelta - parameters specific learning rates with fixed memory window

Evolution of optimizers

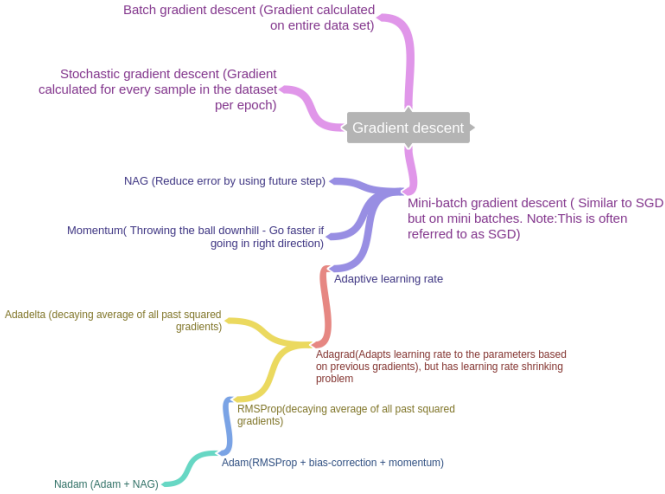


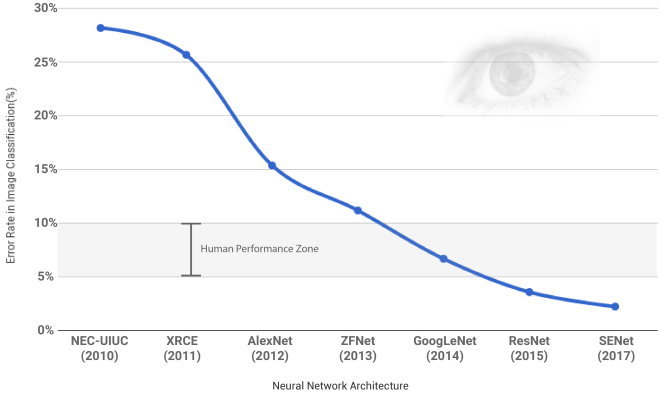
Figure: Evolution of gradient descent optimization algorithms (image by Desh Raj)

Update equations

Method	Update equation
	$g_t = \nabla_{\theta_t} J(\theta_t)$
SGD	$\Delta\theta_t = -\eta \cdot g_t$
	$\theta_t = \theta_t + \Delta\theta_t$
Momentum	$\Delta\theta_t = -\gamma\, v_{t-1} - \eta g_t$
NAG	$\Delta\theta_t = -\gamma\, v_{t-1} - \eta \nabla_\theta J(\theta - \gamma v_{t-1})$
Adagrad	$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$
Adadelta	$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$
RMSprop	$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
Adam	$\Delta\theta_t = -\frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$

Figure: Update equations for different gradient descent optimization algorithms [Ruder, 2016]

Why Deep Residual Networks?



Last two ImageNet Large Scale Visual Recognition Competition (ILSVRC) winners are networks with residual modules:

- ILSVRC 2017: SENet [Hu et al., 2018] is a ensemble of different models, highest scoring model is modified ResNeXt [Xie et al., 2017]
- ILSVRC 2016: ResNet [He et al., 2016]

- **ResNet:** to remedy vanishing gradients, introduce skip connections (bypass layers by adding identity mapping)

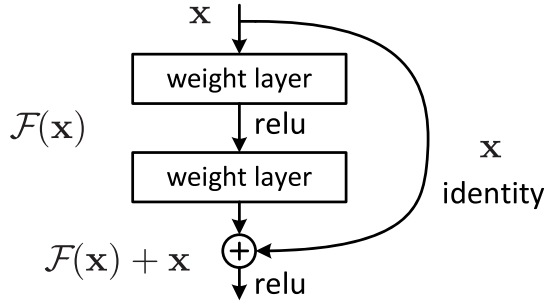


Figure 2. Residual learning: a building block.

Evolution of Deep Residual Networks

- **Wide Residual Networks:** decrease depth (number of layers), increase width (number of feature maps) in residual layers [Zagoruyko and Komodakis, 2016]
- **ResNeXt:** introduce cardinality (number of aggregated transformations), in addition to depth and width [Xie et al., 2017]

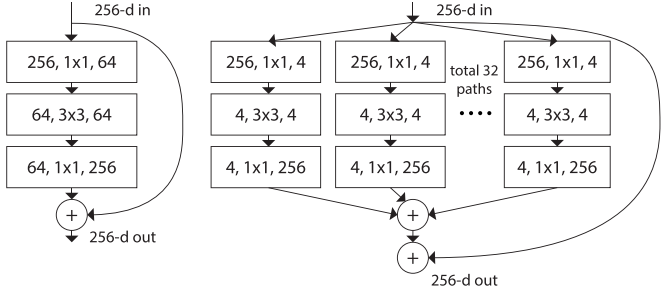


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

- **DenseNet:** each layer has direct input connections to all previous feature maps [Huang et al., 2016]

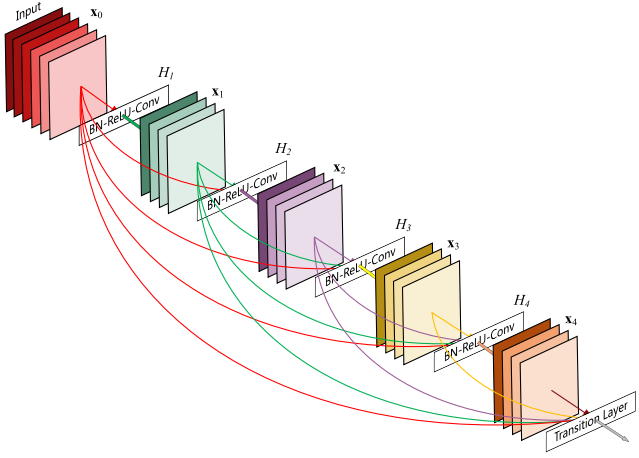
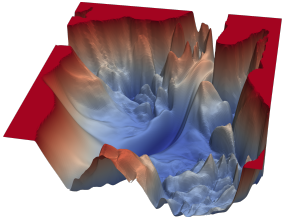


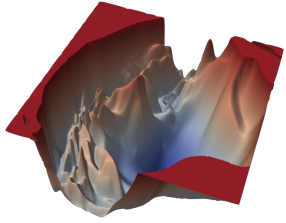
Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Less fancy flying required

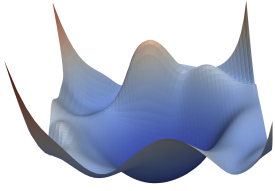
VGG-56 Loss Landscape



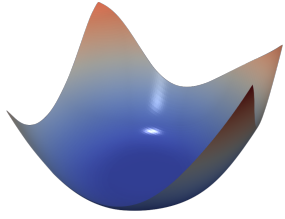
VGG-110 Loss Landscape



Resnet-56 Loss Landscape



Densenet-121



https://www.cs.umd.edu/ tomg/projects/landscapes/

References

Amini, A. and Soleimany, A. (2019).
MIT 6.S191: Intro to Deep Learning.
http://introtodeeplearning.com.

He, K., Zhang, X., Ren, S., and Sun, J. (2016).
Deep residual learning for image recognition.
2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 770–778.

Hu, J., Shen, L., and Sun, G. (2018).
Squeeze-and-excitation networks.
2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 7132–7141.

Huang, G., Liu, Z., and Weinberger, K. Q. (2016).
Densely connected convolutional networks.
CoRR, abs/1608.06993.

Ruder, S. (2016).
An overview of gradient descent optimization algorithms.
arXiv, 1609.04747.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017).
Aggregated residual transformations for deep neural networks.
In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995.

Zagoruyko, S. and Komodakis, N. (2016).
Wide residual networks.