DATA ANLAYSIS COMPUTATIONAL NOTEBOOK

TIANBO ZHANG

University of Washington, Department of Applied Mathematics, Seattle

Contents

1. Transforms and Time-Frequency Analysis

1.1. **Fourier Series.** Fourier Series is the concept of representing a given function by a trigonometric series of sines and cosines:

$$\psi_k^a(x) = sin(kx),\ \psi_k^b(x) = cos(kx)$$

Note $\psi_k^a(x)$ and $\psi_k^b(x)$ are the Fourier basis functions for a signal, $F(x) : [0, 2\pi] \to \mathbb{R}$ Then we have the following equation:

$$f(x) = \frac{b_0}{2} + \sum_{k=1}^{\infty}(a_k \cos kx + b_k \sin kx), x \in [0, 2\pi]$$

Then by finding a_k, b_k we compute the Fourier series for the signal $f(x)$.

$$a_k = \frac{1}{\pi}\int_0^{2\pi} f(x)sin(kx)dx, k \geq 1$$
$$b_k = \frac{1}{\pi}\int_0^{2\pi} f(x)cos(kx)dx, k \geq 0$$

1.2. **Fourier Transform.** Fourier Transform is an commonly used technique to transform a time-domain signal into a frequency-domain representation. Then if $f(x)$ is a time domain signal, its Fourier Transform $F(k)$ over the entire line $x \in [-\infty, \infty]$ is defined as:

$$F(k) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f(x)e^{-ik}dx$$

And its inverse is defined as:

$$f(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} e^{ikx}F(k)dk$$

The concept looks something like this:

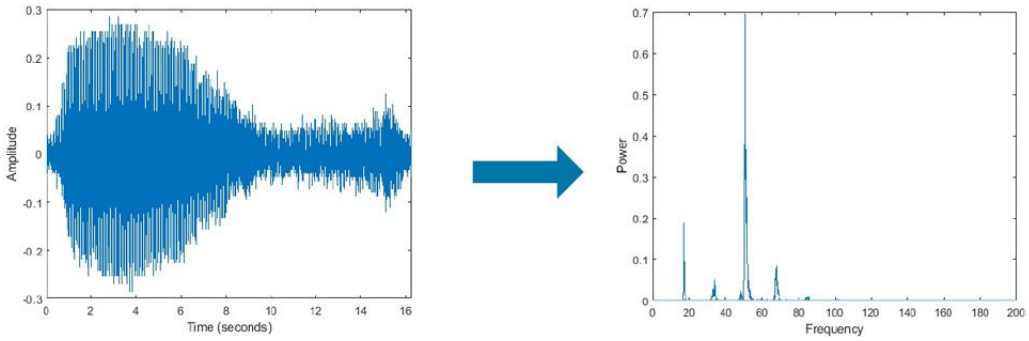

FIGURE 1. Fourier Transform

1.3. **Discrete Fourier Transform.** Discrete Fourier Transform is a method to approximate the Fourier series for a finite number of samples.
Let there be N samples of $f(x_0), \ldots, f(x_{N-1})$, then we have:

$$f(\hat{k}_n) \approx \frac{1}{N}\sum_{n=0}^{N-1} f(x_n)e^{-i2\pi\frac{k_n n}{N}}$$

And its inverse is defined as:

$$f(x_n) \approx \sum_{n=0}^{N-1} f(\hat{k}_n) e^{i2\pi \frac{k_n n}{N}}$$

Note that DFT is directly computable. And it takes $O(N^2)$ computational time.

1.4. **Fast Fourier Transform.** Fast Fourier Transform is more efficient way to compute DFT. The way to compute FFT is:

$$A_k = \sum_{m=0}^{n-1} a_m \exp -2\pi i \frac{mk}{n} \text{ where } k = 0, \dots, n-1$$

Then we have the inverse fft to be:

$$a_m = \frac{1}{n} \sum_{k=0}^{n-1} A_k \exp 2\pi i \frac{mk}{n}$$

1.4.1. *N-dimensional FFT.* Note we can perform Fast Fourier Transform in d dimensional space.

Suppose we have $x, y, z, \dots$ and $\bar{k} = k_x, k_y, k_z, \dots$.

Then we have the fft:

$$\hat{f}(\bar{k}) = \int_{\mathbb{R}^d} f(x, y, z, \dots) \exp -i(k_x x + k_y y + k_z z + \dots) dx dy dz \dots$$

1.4.2. *Averaging the FFT.* Note that if we are dealing with the white noise data, averaging the FFT enable us to perform a preliminary noise filtering to the data by taking the average. We could implement the following algorithm to do so:

Algorithm 1 Averaging Fourier Transform

```
for each sample do
    Preprocess the data fore fft analysis
    Calculate the fft using fftn from numpy package
    Add the fft to the sum of previous ffts
end for
Shift the fft to correct order by fftshift from numpy package
Normalized the data by taking its absolute value and divide by its
maximum absolute value
```

1.5. **Gabor Transform.** Gabor transform also called short time Fourier Transform(STFT) is the application of a localized filter to the signal and "sliding" it over the signal and performing FT/FS.

$$G[f](\tau, k) = \int_0^{2\pi} f(t) g(t - \tau) \exp -iktdt$$

Note that Gabors filter is Gaussian:

$$g(t) = \frac{1}{\sqrt{\pi \sigma^2}} \exp^{-\frac{t^2}{2\sigma^2}}$$

1.5.1. *Filter Choice.* We could extend the choice of the filter to additional localized functions if the filter satisfies the following properties:

(1) g is real and symmetric
(2) $||g||_{L^2} = (\int_{-\infty}^{\infty} |g(t)|^2 dt)^{\frac{1}{2}} = 1$

1.5.2. *Finite Interval Gabor.* Note that for a finite interval replace k and τ with:

$$k = m\Delta k, \tau = n\Delta t$$

Then we have:

$$G[f](m,n) = \frac{1}{2L}\int_0^{2L} f(t)g(t-n\Delta t)\exp^{-\frac{i\pi m\Delta kt}{L}}\,dt$$

1.6. **Wavelet Transform.** Wavelet transform is an extension of the choice of window, where we define an extended kernel:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-b}{a})$$

Then the continuous Wavelet Transform is defined as

$$W_\psi[F](a,b) = \int_{-\infty}^{\infty} F(t)\psi_{a,b}(t)dt$$

Some special properties of WT are:

(1) Linear: $W[\alpha f + \beta g](a,b) = \alpha W[F](a,b) + \beta W[g](a,b)$
(2) Translation: $W[T_c f](a,b) = W[f](a,b-c) \longrightarrow T_c f = f(t-c)$
(3) Dilation: $W[D_c f](a,b) = \frac{1}{\sqrt{c}}W[f](\frac{a}{c},\frac{b}{c}) \longrightarrow D_c f = \frac{1}{t}f(\frac{t}{c})$
(4) Inversion: $f(t) = \frac{1}{C_\psi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} W_\psi[f](a,b)dadb \longrightarrow C_\psi = \int_{-\infty}^{\infty}\frac{|\hat{\psi}(k)|^2}{|k|}dk$

1.6.1. *Discrete Wavelet Transform.* If we are using the Wavelet Transform on a set of discrete data we'll have:
Pick a_0 and b_0 with $m, n \geq 0$:

$$\psi_{m,n}(t) = a_0^{-\frac{m}{2}}\psi(a_0^{-m}t - nb_0)$$

Then DWT is:

$$W_{m,n}[F] = \int_{-\infty}^{\infty} F(t)\psi_{m,n}(t)dt$$

1.6.2. *Multi-Resolution Analysis.* Note we have orthogonality for FS/Ft. For two dimensions, we also have the same case. For $\tilde{n}, \tilde{m} \neq 0$:

$$\int \psi_{m,n}(t)\psi_{m,n+\tilde{n}}(t) = 0 \text{ and } \int \psi_{m,n}(t)\psi_{m+\tilde{m},n}(t) = 0$$

Then we have:

$$\int \psi_{m,n}\psi_{k,l}dt = \delta_{mn}\delta_{kl} \text{ where } \delta_{kl} = 1 \text{ for } k = l \text{ and } 0 \text{ otherwise}$$

2. Dimension Reduction

2.1. **Norms.**

2.1.1. *Vector Norm.* A function $||.|| : \mathbb{R}^n \to \mathbb{R}$ is called a norm of vector in $\mathbb{R}$ when it satisfies:

(1) $||x|| \geq 0$ and $||x|| = 0$ only is $x = 0$
(2) $||x+y|| \leq ||x|| + ||y||$
(3) $||\alpha x|| = |\alpha|||x||$

Some examples of vector norms are:

(1) p-nrom: $||x||_p = (\sum_{j=1}^n |x_j|^p)^{\frac{1}{p}}, 1 \leq p \leq \infty$
(2) L1-norm: $||x||_1 = \sum_{j=1}^n |x_j|$
(3) L2-norm: $||x||_2 = (x^T x)^{\frac{1}{2}}$
(4) Infinity norm: $||x||_\infty = max_{1\leq j\leq n}|x_j|$

2.1.2. *Matrix Norm.* Given $A \in \mathbb{R}^{n\times m}$ we have:

$$||A||_{(m,n)} = sup_{x\in\mathbb{R}^m} \frac{||Ax||_{(n)}}{||x||_m} = sup_{||x||_{(m)}=1} ||Ax||_{(n)}$$

For example we have:

(1) $||A||_1 = max_{1\leq j\leq m} ||a_j||_1$
(2) $||A||_\infty = max_{1\leq j\leq n} ||\tilde{a}_i^T||$

Another commonly used norm is the Frobenius norm:

$$||A||_F = (\sum_{i=1}^n \sum_{j=1}^m |a_{ij}|^2)^{\frac{1}{2}} = \sqrt{Tr(A^T A)}$$

2.2. **Singular Value Decomposition.** For any $A \in \mathbb{R}^{n*m}$ there exists unitary matrices $U \in \mathbb{R}^{n*n}, V \in \mathbb{R}^{m*m}$ and a diagonal matrix $\Sigma \in \mathbb{R}^{n*m}$ with positive entries such that:

$$A = U\Sigma V^T$$

Note we will have the following with sequence of transformations with SVD:

$$Ax = U\Sigma V^T x \Rightarrow x \xrightarrow[\text{rotation}]{V^T} V^T x \xrightarrow[\text{scaling}]{\Sigma} \Sigma V^T x \xrightarrow[\text{change of basis}]{U} U\Sigma V^T x$$

2.3. **Principle Component Analysis.** PCA aims to measure the variance of the data in each axis that is centralized(mean 0). It can also find a decent solution to the "least number of components to summarize data". We can start by considering the covariance matrix:

$$C_x = Cov(x) \cong \frac{1}{N-1} XX^T$$

Then plug in the SVD of X we have:

$$C_x \cong \frac{1}{N-1} U\Sigma V^T V\Sigma U^T = \frac{1}{N-1} U\Sigma^2 U^T$$

Then let $Y = UX$, we have:

$$C_Y = \frac{1}{N-1} YY^T = \frac{1}{n-1} \Sigma^2$$

Then note PC modes are eigenvectors of C_x and σ^2 are eigenvalues of C_x. There fore, PC modes indicate the "optimal" directions to represent variance of the data. The process may look something like this:

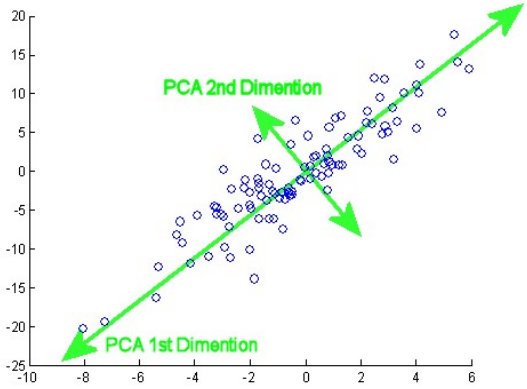

FIGURE 2. PCA Analysis

2.3.1. *Energy Calculation.* To find the accurate amount of PC modes is an essential step in PCA analysis. One way to do this is to calculate the cumulative energy for the k-PC modes to match the percentage of accuracy that we want to keep. We can use the following algorithms:

Algorithm 2 Energy Calculation and PCA Modes Determination

```
Compute the energy for each row
Set the cumulative energy to zero
for each PCA component's energy do
   Compute the cumulative energy up to the current PC mode
   if cumulative energy is bigger than the targeted percentage then
      Return the index of the PCA mode
   end if
end for
```

2.4. **Projection onto PCA space.** After computing the PCA analysis for our data, note that U provides a basis onto which the data can be projected:

$$X = U\Sigma V^T \Rightarrow \text{In PCA basis } U^T$$

Then we have the projection:

$$U^T X = \Sigma V^T$$

Note that the projection can also be done on truncated basis (k-PCA truncation basis). Truncated projection simply keep the k-vectors in U^T which corresponds to the largest k σs:

$$U_k^T = \begin{pmatrix} | & & | & | & & | \\ u_1 & \dots & u_k & 0 & \dots & 0 \\ | & & | & | & & | \end{pmatrix} \text{ and } U_k^T X = \tilde{X}_{pc_k}$$

2.5. **Proper Orthogonal Decomposition.** Proper Orthogonal Decomposition project the high-order, nonlinear system to the low-dimensional state space through the orthogonal mode, and ensure the minimum residuals for limited modes.

$$f(x,t) = \sum_{j=0}^{\infty} c_j(t)\psi_j(k)$$

Then we have:

$$f(x,t) = U\Sigma V^T = \begin{pmatrix} | & | & & | \\ f(x,t_0) & f(x,t_1) & \dots & f(x,t_{t-1}) \\ | & | & & | \end{pmatrix}$$

2.6. **Dynamic Mode Decomposition.** Dynamic mode decomposition computes a set of modes each of which is associated with a fixed oscillation frequency and decay/growth rate.

Consider D^T and $D^{T+1} = \begin{pmatrix} | & | & & | \\ d_1 & d_2 & \dots & d_T \\ | & | & & | \end{pmatrix}$

Note that DMD seeks for A such that: $D^{T+1} = AD^T$ where A is a linear transformation from t to $t+1$. It can be computed with SVD:

$D^{T+1} = AU\Sigma V^T \Longrightarrow U^T D^{T+1} = U^T AU\Sigma V^T \Longrightarrow U^T D^{T+1} V\Sigma^{-1} = U^T AU = S$

Then let y be the eigenvectors of $S \Longrightarrow Uy$ is the eigenvectors of A

Then note DMD effectively represents t_k time snapshot:

$f(x,t_k) = Af(x,t_{k-1}) = A^{k-1}f(x,t_0) = Q\Lambda^{k-1}Q^{-1}f(x,t_0) = \sum_{j=1}^{r} q_j \lambda_j^{k-1} b$

Note that DMD coefficients are comparable with POD coefficients.

3. Intro to Machine Learning

3.1. **Supervised Learning.** A formal definition of Supervised Learning is that given a set of data and labels as training set find a model to predict labels labels form input data. Then use the trained model to make predication for new input. Mathematically, we are looking to solve the following problem:

$$y_j = f(x_j) + \epsilon_j, \epsilon \sim N(0, \sigma^2)$$
$$y_j|x_j \sim N(f(x_j), \sigma^2)$$
$$P(y_j|x_j) \propto \exp -\tfrac{1}{2\sigma^2}|f(x_j) - y_j|^2$$

Note that this typically can be recast into a optimization problem using Maximum Likelihood(MLE):

$$f_{MLE} = argmin_f \tfrac{1}{2\sigma^2}||f(x) - y||^2$$

The algorithm implementation is:

Algorithm 3 Supervised Learning Procedure

```
Preprocess the input data: X and Y
Seperate the data into X_train, X_val and Y_train, Y_val
Train the model with X_train and Y_train
Validate the model with X_val and Y_val
Make predication using the model for X_test
```

Common supervised learning tasks are:

(1) Classification: Classify objects in images.
(2) Recognition: Image recognition.
(3) Forecasting: Weather forecasting.
(4) Approximation: Approximate weight according to eating habit.

3.2. **Unsupervised Learning.** The concept of unsupervised learning is that given only data, perform analysis with a model $f(x)$ to infer meaningful structure in the data i.e. $f : X \to Y$ without Y given. Common unsupervised learning tasks are:

(1) Dimension Reduction: Given X, $f : X \to X_k$.
(2) Clustering: Given X divide X into n clusters.

3.3. **Reinforcement Learning.** Reinforcement learning is the special case of unsupervised learning where the objective is not known. Hence, it is not clear how to implement the learning. In many cases it's implemented through assigning rewards.

3.4. **Mean Square Error.** The mean square error is an estimator that measures the average of the square difference between the estimated values and actual values.

$$MSE_{train}(\hat{f}, y) = \tfrac{1}{N}\sum_{n=0}^{N-1}|\hat{f}(x_n) - y_n|^2$$
$$MSE_{test}(\hat{f}, y) = \tfrac{1}{N}\sum_{n=0}^{N-1}|\hat{f}(x'_n) - y'_n|^2$$

3.5. **Cross-Validation (K-fold).** Cross-validation is the process of separating the training data after shuffling and compute the average error/accuracy among them. The formal process is as following:

(1) Shuffle the data.
(2) Divide into k subsets.
(3) Train each subset and validate the other subsets using the trained model.

3.6. **Linear Regression.** Linear regression is common approach for regression in supervised learning. The objective function of LR is:

$$f(x) = \beta_0 + \sum_{j=1}^{d} \beta_j x_j$$

Let $A = [1|x^T] \in \mathbb{R}^{Nx(d+1)}$, then the problem becomes a least squares solution to $A\beta = y$:

$$f_{MLE} = \beta_{MLE} = argmin_{\beta \in \mathbb{R}} \frac{1}{2\sigma^2} ||A\beta - y||^2$$

After solving the least square problem by differentiation and equaling to 0 we have:

$$\beta_{MLE} = (ATA)ATy$$

A visual example is something like this:

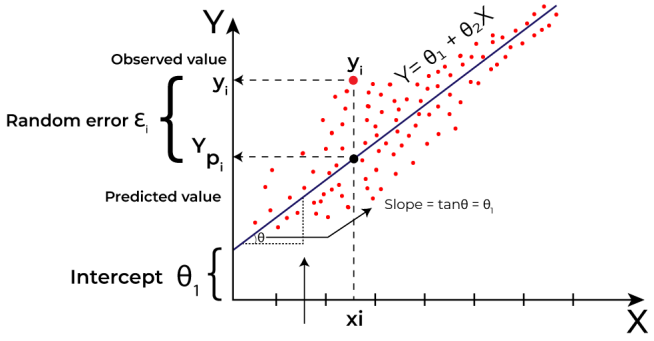

FIGURE 3. Linear Regression

3.7. **Ridge Classifier.** Ridge classifier is a ML algorithm designed for multi-class classification tasks. It applies L2 regularization to penalize large coefficients in the model to prevent over-fitting, enhancing its generalization to new data. The classifier minimizes the squared loss with an added L2 penalty on the magnitude of the coefficients, given by the formula:

$$\hat{\beta} = argmin_{\beta \in \mathbb{R}^J} \frac{1}{2\sigma^2} ||X\beta - y||^2 + \frac{\lambda}{2} ||\beta||_p^p$$

3.8. **KNN Classifier.** KNN classifier is an algorithm operates on the principle that similar data points are often in close proximity to each other. For classification, kNN assigns a class to a new data point based on the majority class among its k nearest neighbors in the feature space. The distance is usually calculated using Euclidean distance as following:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

3.9. **K Means Classifier.** K mean classifier group/cluster points into k clusters. The iterative procedure is:

Algorithm 4 K Mean Classifier

```
Pick K centers randomly
for each point in the data do
   Assign data points to closest cluster center
   Change the center to average at assigned points d
   Stop the algorithms when there's no change
end for
```

3.10. **LDA Classifier.** LDA classifier is based on logistic regression. LDA aims to project the data onto a lower-dimensional space that maximizes the class separation by finding the projection that maximizes the ratio of between-class variance to within-class variance.

4. Intro to Deep Learning

4.1. **Neural Network.** A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data.They consist of layers of nodes, also known as neurons, and can learn and make intelligent decisions on their own. A structure of NN is:

(1) Input Layer: This is the initial data layer where each neuron represents a feature of the input dataset.
(2) Hidden Layers: Layers of neurons that lie between the input and output layers.
(3) Output Layer: The final layer that provides the output of the model corresponding to the problem statement.

Here are the three types of Neural Network:

Type	Basic Idea
Fully Connected Neural Networks (FCNN)	Every neuron in one layer is connected to every neuron in the next layer
Convolutional Neural Networks (CNN)	Primarily used in image recognition and processing where the network employs a mathematical operation called convolution
Recurrent Neural Networks (RNN)	Designed to recognize patterns in sequences of data, such as text, genomes, or handwriting, where the output from the previous step is fed back into the model.

4.2. **Activation Function.** Activation functions are non-linear functions performed by neurons on each layer. Commonly used activation function is:

(1) ReLU - Rectified Linear Unit: $y \geq 0$
(2) Tanh: $-1 < y < 1$
(3) Sigmoid: $0 < y < 1$

4.3. **Gradient Descent.** Gradient descent is an optimization algorithm for finding a local minimum of a differentiable function. In deep learning it's used for find the values of a function's parameters or coefficients that minimize a cost function as far as possible. Define cost function to be:

$$J(\{\hat{y}\}^m, \{y\}^m; \{\vec{x}\}^m) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

Then we have the process of gradient descent to be:

$$\vec{w}_{k+1} = \vec{w}_k - \alpha \Delta_{\vec{w}} J(\vec{w}_k; b)$$
$$b_{k+1} = b_k - \alpha \frac{\partial}{\partial b} J(\vec{w}; b_k)$$

There are different types of gradient descent methods, each with different gain and lost:

(1) Stochastic Gradient Descent:
   (a) Can loose speed up from oscillations.
   (b) Hard to parallelize.
(2) Mini-batch Gradient Descent:
   (a) The whole minibatch is evaluated in parallel.
   (b) Mostly consistent convergence.
(3) Batch Gradient Descent:
   (a) Consistent convergence.
   (b) Too long per iteration.

4.4. **Back Propagation.** Back propagation is a gradient estimation method used to train neural network models. The gradient estimate is used by different optimization algorithm to compute the parameters. We have:

$$\nabla_{\hat{w}} L(\hat{y}, y) = \nabla_{\vec{w}} z \longleftarrow \frac{\partial \hat{y}}{\partial z}(z) \longleftarrow \frac{\partial L}{\partial \hat{y}}(\hat{y}, y)$$

4.5. **Cross Entropy Loss.** Cross entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1. The cross entropy loss increases as the predicted probability diverges from the actual label.

$$L(\hat{y}, y) = -(t \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

4.6. **Optimizer.** An optimizer is an algorithm used to change the attributes of a neural network model, such as weights and learning rate, to reduce losses. There are several types of optimizer:

(1) Stochastic Gradient Descent: A variation of gradient decent that updates all weights in the direction of the negative gradient of the loss function with small batches.
(2) RMSprop: Adjusts the learning rate adaptively for each parameter, making it smaller for updates associated with large gradients to prevent exploding and larger for updates associated with small gradients to prevent vanishing.
(3) Adam: Store an exponentially decaying average of past squared gradients, and also keeps an exponentially decaying average of past gradients.
(4) Momentum: "Accelerate" gradients vectors in the right directions, to lead to faster converging.
(5) Adagrad: Adagrade uses a different learning rate for every parameters w_j at every step k. It eliminates the need to manually tune the learning rate

Here's a comparison table for different optimizers:

Optimizer	Epochs	Learning Rate	Batch Size	Weights and Bias
SGD	High	High tune	Small batch	Non-adaptive
Momentum	High	Medium tune	Larger batch	Non-adaptive
AdaGrad	Low	Low tune	Larger batch	Adaptive
AdaM	Low	Less tune	Larger batch	Adaptive
RMSProp	Low	Low tune	Larger batch	Adaptive

Note that mostly using different optimizer depends greatly on the structure of the dataset. Hence, in real life problem there should be further analysis.

4.7. **Over-fitting and Under-fitting.** The investigation of over-fitting and under-fitting is important for model training.

4.7.1. *Over-fit.* If over-fit our model learns the training data too well. It captures noise and fluctuations in the training data where it negatively impacts the model's performance on new data.

4.7.2. *Under-fit.* If under-fit our model is not utilizing the parameters enough to learn the underlying structure of the data. It can't capture the complexity of the data and result in poor performance for the model. We can look at the train/validation loss curve to find a solution.

Then we have the following techniques to solve the over/under fit problem:

Overfitting	Underfitting
More Data	More Layers/Nodes
Regularization	Longer training
Dropout	Architecture
Initialization	Hyper-parameter Optimization

4.8. **Regularization.** Regularization is a technique to prevent overfitting for our model. Here's how we add regularization term to the cost L2for single layer:

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}||\vec{w}||_2^2$$

For L1 just change the last term to $\frac{\lambda}{m}||\vec{w}||_1$.
For multilayered regularization we have:

$$J(W^{[1]}, b^{[1]}, \ldots, W^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}||W^{[l]}||_F^2$$

$$||W^{[l]}||_F^2 = \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (w_{ij}^{[l]})^2$$

4.9. **Dropout.** The concept of dropout is similar to L2 regularization. It effectively spreading the weights. Note it can be dependent on weight dimension.

$$\overrightarrow{a}_l^d = \overrightarrow{a}_l \times \overrightarrow{d}_l/(1 - p_d)$$

4.10. **Normalization of the datasets.** When our model is having a bad performance, dataset normalization is always a good place to start.

(1) Zeros mean: $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$, $x^{(i)\mu} = x^{(i)} - \mu$
(2) Normalized Variance: $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} {x^{(i)}}^2$, $x^{(i)\mu,\sigma^2} = x^{(i)\mu}/\sigma^2$

4.11. **Batch Normalization.** Batch normalization normalize the outputs of each layer. We have:

$$z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}$$

Then after normalization we have:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} z^{(i)},\ \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(z^{(i)} - \mu)^2,\ z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

4.12. **Weight Initialization.** Weight initialization involve the concept of placing small random numbers into the weight. It can prevent vanishing/exploding gradients and also breaking symmetry. There are several different types of initialization.

(1) Random Normal: Randomly pick weights from normal distribution.
(2) Xavier: $Var(w^{[l]}) : 1/n^{[l-1]}$, $w^{[l]} = N(0,1)\sqrt{\frac{1}{n^{[l-1]}}}$
(3) Heiming: $Var(w^{[l]}) : 2/n^{[l-1]}$, $w^{[l]} = N(0,1)\sqrt{\frac{2}{n^{[l-1]}}}$

Note that Xavier works well with tanh activation function and Heiming works well with ReLu activation function.

4.13. **Convolution Filters.** Convolution filters are used with images for blurring, sharpening, embossing, edge detection and more. This is accomplished by doing a convolution between a kernel and an image.

$$F[m,n] = I[m,n] * w[m,n] = \sum_{j=-\infty}^{\infty}\sum_{i=-\infty}^{\infty} I[i,j]w[m-i, n-j]$$

Some examples of the filters are:

Operation	Filter
Identity	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
Edge detection	$\begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{pmatrix}$
Sharpen	$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$
Box blur	$\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

4.13.1. *Padding.* Padding is the parameter that shrinks output where image edges values less taken into account. If $p = (f-1)/2$ then same convolution have $n^{[l]} = n^{[n-1]} + 2p - f + 1$

4.13.2. *Stride.* Stride is a parameter that refers to the number of pixels by which the filter matrix moces across the input matrix. With stide s we have:$n^{[l]} = (n^{[l-1]} + 2p - f)/s + 1$

4.14. **Max Pooling.** Max Pooling is the concept that reduces the size of the feature map to speeds up computation.

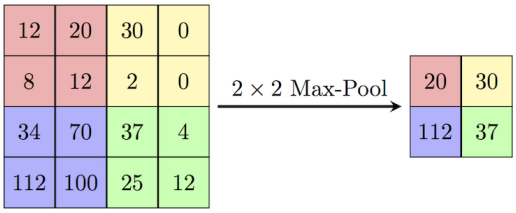

FIGURE 4. Max Pooling

4.15. **Vanishing Gradient.** The vanishing gradient problem is primarily caused by the choice of activation functions and the nature of deep networks. And the consequences can be severe:

(1) Slow Learning: The primary consequence is that the learning process becomes very slow because the weights in the early layers of the network barely change.
(2) Poor Performance: Networks affected by vanishing gradients often fail to converge to a good model, resulting in poor performance on the training and validation datasets.
(3) Limited Network Depth: Before solutions to the vanishing gradient problem were developed, it was challenging to train very deep neural networks effectively because of this issue.

5. Acknowledgements

The author is thankful to Prof. Eli Shlizerman for the wonderful lessons and Pro.Zhang from Beijing University of Post and Telecom-munications for useful discussions in the topic of data analysis.

6. References

REFERENCES

[1] J.N. Kutz (2013) *Methods for Integrating Dynamics of Complex Systems and Big Data*, Oxford.
[2] Alan V. Oppenheim, Alan S. Willsky (1997) *Principal component analysis*, 2nd Edition.
[3] Hal Daume III *A Course in Machine Learning*
[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville *Deep learning*
[5] Alan V. Oppenheim, Alan S. Willsky (2022) *Signals and Systems*, 2nd Edition.
[6] Diederik P. Kingma, Jimmy Lei Ba (2015) *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*
[7] Anshul Saini (2022) *An Introduction to Random Forest Algorithm for beginners*, Analytic Vidhya.