Software:

Python – panda\_define

* **spatialpandas**
* **dask**
* datashader – visual (numba/dask/gpu) and Nvida gpu

bit.ly/datashader21

https://nbviewer.jupyter.org/urls/notebooks.anaconda.org/jbednar/datashader21/download/%3Fversion=

plotly

??parquet

Cross Validation:

N | P

N TN | FP

P FN | TP

Precision: TP/TP+FP

Recall: TP/TP+FN

F1 Score (harmonic mean): Px R/ P + R 🡺 TP/TP+(fn+fp/2)

Precission/Recall trade off as one goes up the other goes down.

Receiver Operation Character: Roc

Area Under the Curve: Auc

Binary Classification

MultiClass Classification/ Multi Normal Classification

MultiLabel Classification , classifies multiple labels at ones. E.g A,B,C [1,0,1]: [yes,no,yes]

Multioutput Classification each able is multi classes, more than 1 output

Onehotencoding

Normalisation, MinMAx and Standardisation

Cross validation

Pipeline

Classification - Categories

Regression

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Images | Label | Classification |  |
| Audio | Sequence/Test |  |  |
| Ascii | Unicode sequence |  |  |
| Environment |  |  |  |

**sklearn**

**Data Cleaning**

* GET RUIDE OF SOME ATTRIBUTE
* GET RIDE OF CORRRELATED ATTRIBUTES
* SET VALUES TO ZERO, MEANS, MEDIANS
* REMOVE VALUES

DF.dropna()

DF.drop ()

DF[“”].median()

DF[“”]fillna(median, inplace=True)

Imputer = SimpleImpuer(strategy=”medium”)

housing\_num = housing.drop(“field”,axis=1)

imputer.fit(housing\_num)

X = imputer.transform(housing\_num)

housing\_ts = pd.DataFrame(X,columns=housing\_num.columns,index= housing\_num.index)

**Encoding**

OrdinalEncoder – ranks the values – which is wrong

OneHotEncoder – use this

**Feature Scaling**

Scaler –

BaseEsimator, TransofrmMixin

Custom Transformers

Encoding:

Pipelines

Dimensionality Reductions

Method: PCA, Kernal PCA, LLE

Unrolling the swiss roll vs squashing it

Manifold Learning - manifold hypothesis

Singular Vector Decomposition (SVD)

Set PCA limit, and you see an elbow of dimensions which get a reduction in explained variance

Errors

MBGD - performance

Variance vs bias

Under vs Overfitting

Regression techniques

* + Ridge regression
  + Lasso regression
  + Elastic net
  + Early Stopping
  + Logistic Regression

Over fitting: use dropout layer

batch\_size: mini batches

epohcs:

increase epohcs and smaller batches increases accuracy and reduce loss

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Classification | | Model |  |
|  | Binary | MutliNominal | sklearn |  |
| Logistic Regression |  | X |  |  |
| Softmax Regression |  |  |  |  |
|  |  |  |  |  |
| Random Forest |  | X |  |  |
| Naïve Bayes |  | X |  |  |
|  |  | X |  |  |
| Support Vector Machine | X | Scale badly best 1-1 | Svm: SVC |  |
| SGD Classifiers | X |  |  |  |
| GridSeachCV |  |  | GridSearchCV(knn\_clf, param\_grid,cv=5,verbose=3) |  |
| KNeighborsClassifier |  |  |  |  |

Sklearn.metrics :

accuracy\_score

precision\_score

recall\_score

mean\_Squared\_error

Use Binary as N Classifications: one vs rest: OvR

Binary for every pair : one vs one: OvO Nx(N-1)/2

Classification Steps:

Model selection: sklearn.model\_selection

from sklearn.linear\_model import SGDClassifier

from sklearn.svm import SVC

Sklearn.preprocessing import standadScaler

**Training models**

Find the model that minimises the RMSE or RME.

**Errors:**

**RMSE** – Root mean square error

**RME** – Root Mean Error, Cost function

Gradient decent

Batch Gradient

Mini Batch

**RNN**

BatchNormalization:

LayerNormalization:

Trends

Seasonality

ARMIA

LSTM – Long short term memory

GRU

Backward propagation

Deep RNN

One dimensional convolution layer to process Sequences

WaveNet

TensorFlow Probability - Structural Time Series STS

**Readings**

https://blog.tensorflow.org/2019/03/structural-time-series-modeling-in.html

https://towardsdatascience.com/structural-time-series-forecasting-with-tensorflow-probability-iron-ore-mine-production-897d2334c72b

**Data augmentation**

**Activation Functions**

Step

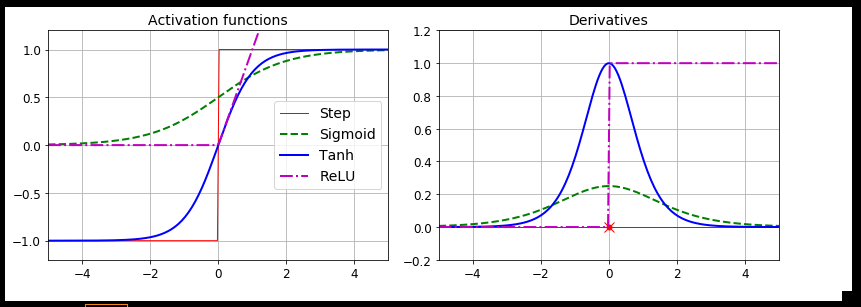
Sigmod

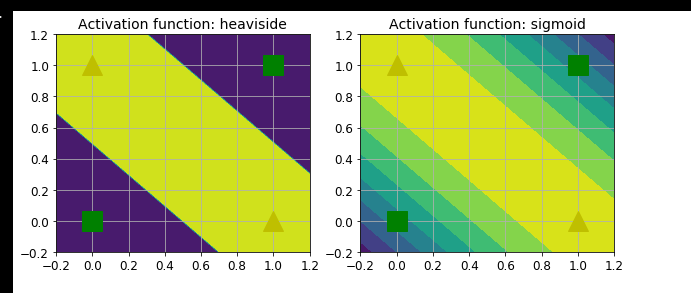
RelU

Tanh

Heaviside

Mlp\_xor





**Hyper parameters**

Tuning hyper parameters:

Gradient checking: check if back propagation is working.

Regularization: if dataset not big enough data can overfit. Use regularization in the models.

L1, L2

Regularization hurts the training set, but reduces overfitting

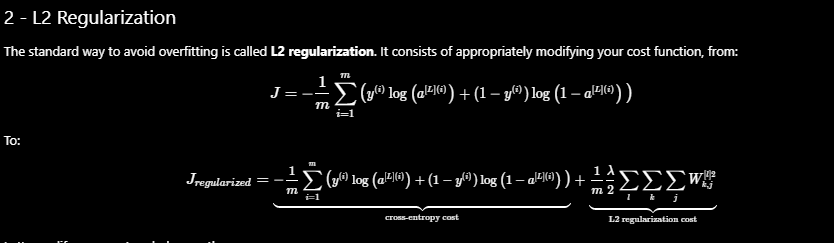
**Regularisation- avoid over fitting**

L1,l2

Dropout

MC Dropout

Max-Norm Regularization



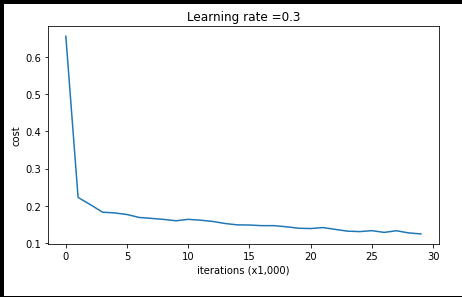
L2\_regularization\_cost =

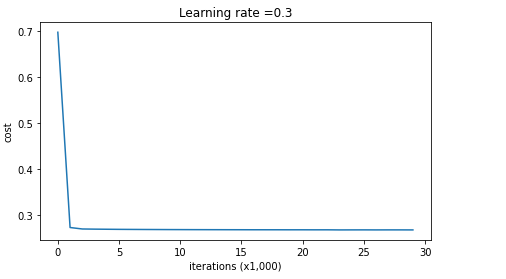
(lambd / (2\*m)) \*

np.sum ( np.sum(np.square(W1)) + np.sum(np.square(W2)) + np.sum(np.square(W3)))

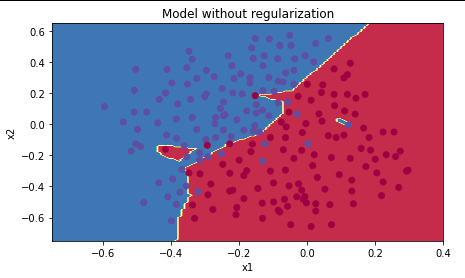
Thus: sum l w2 , k w2 , j w2 then sum all l,k,j

Without l2

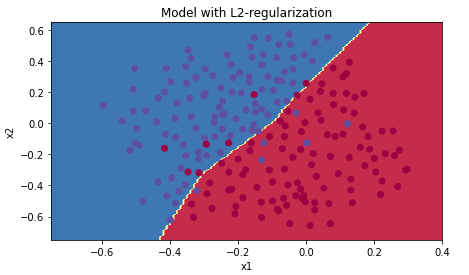




Overfitting:



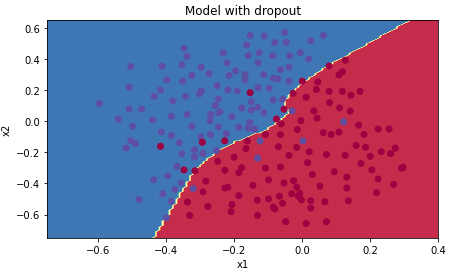
**Regularisation- avoid over fitting**



**Dropout**

**Use dropout only in training**

Dropout for forward and backward propagation



**Use dropout only in training**

Optimization methods

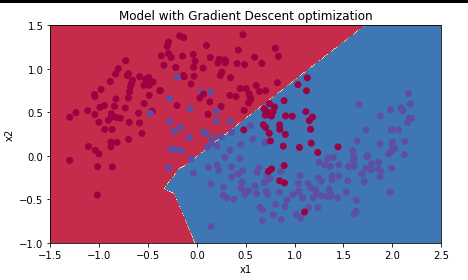
**Learning Rate Scheduling**

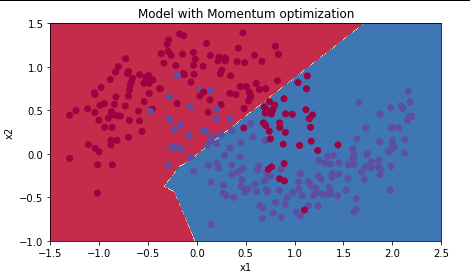
**Gradient Descent:** The loss function should be minimized

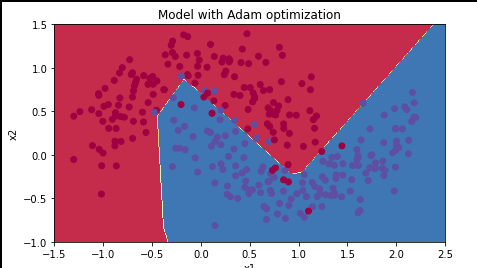


The terms cost and loss functions almost refer to the same meaning. But, loss function mainly applies for a single training set as compared to the cost function which deals with a penalty for a number of training sets or the complete batch

* SGD, stochastic gradient descent,
* Min batch,
* Nestrov Accelerated Gradient
* AdaGrad
* RMSProp
* Adam: combines momentum and RMSProp
* Nadam

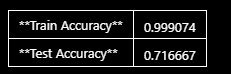






Considerations:

Training vs Test results



With a large difference, try l2 or dropout (training only)

Epochs: training N sessions to optimize the parameters

**Multilayer Perception MLP**

Input Layer – lower layers

Hidden Layers with (TLUs) - lower layers

Output Layer with (TLUs) - Upper layers

* Feedforward Neural Networks : FNN
* Deep Neural Network: DNN

**crossentropy :** used to adjust model weights during training, minimizing the loss.

**https://gombru.github.io/2018/05/23/cross\_entropy\_loss/**

categorical\_crossentropy – one target prob per class per instance

sparse\_categorical\_crossentropy – sparse labels and classes are exclusive

binary\_crossentropy – binary classification use sigmoid and output with softmax

Back propagation

Autodiff – automatic differentiation

**Check List**

Frame the problem and look big picture

Get the data

Explorer the data to gain insight

Prepare the data to expose underlying patterns

Explorer models and combines

Present solution

Launch monitor and maintain

Frame problem and look big picture

Short list models

My Steps

* Data
* Normalise – pre-process
* Reduce dimensionality
* SVM for classifications

**Tensor Flow**

Logical computations with Nurons

C=A,

C=A^B

C= A V B

C= A ^ ¬ B

**Sequential**

**Steps**

Define Model, hidden layers and weights

Compile

Evaluate

Predict/Predict class

**model = keras.models.Sequential([**

**keras.layers.Flatten(input\_shape=[28, 28]),**

**keras.layers.Dense(300, activation="relu"),**

**keras.layers.Dense(100, activation="relu"),**

**keras.layers.Dense(10, activation="softmax")**

Flattern input 28\*28

Add Dense layer 300 neurons RelU

Add Dense layer 100 neurons RelU

Add Dense 10 – 1 for each class softmax – because classes are exclussive

300 represents to 300 out from the 1st layer

First layer has 784 \* 300 connectio0n weights, plus 300 bias terms, which is 235,500 paramters

model.layers

model.summary()

hidden1 = model.layers[1]

hidden1.name

model.get\_layer(hidden1.name) is hidden1

weights, biases = hidden1.get\_weights()

model.compile(loss="sparse\_categorical\_crossentropy",optimizer="sgd",metrics=["accuracy"])

model.evaluate(X\_test, y\_test)

y\_proba = model.predict(X\_new)

y\_pred = model.predict\_classes(X\_new)

history = model.fit(X\_train, y\_train, epochs=30, validation\_data=(X\_valid, y\_valid))

history.params

print(history.epoch)

history.history.keys()

**cross entropy**

categorical\_crossentropy – one target prob per class per instance

sparse\_categorical\_crossentropy – sparse labels and classes are exclusive

binary\_crossentropy – binary classification use sigmoid and output with softmax

**Regression MLP**

Only 1 output neuron defined as that is all required as it’s a regression problem. 30 used not to over fit. Should align to the features used.

**model = keras.models.Sequential([**

**keras.layers.Dense(30, activation="relu", input\_shape=X\_train.shape[1:]),**

**keras.layers.Dense(1)**

**])**

model.compile(loss="mean\_squared\_error", optimizer=keras.optimizers.SGD(lr=1e-3))

history = model.fit(X\_train, y\_train, epochs=20, validation\_data=(X\_valid, y\_valid))

mse\_test = model.evaluate(X\_test, y\_test)

X\_new = X\_test[:3]

y\_pred = model.predict(X\_new)

**Functional API**

This is a more complex deep and wide neural network

input\_ = keras.layers.Input(shape=X\_train.shape[1:])

hidden1 = keras.layers.Dense(30, activation="relu")(input\_)

hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)

concat = keras.layers.concatenate([input\_, hidden2])

output = keras.layers.Dense(1)(concat)

model = keras.models.Model(inputs=[input\_], outputs=[output])

model.summary()

applying deeper – with more features

input\_A = keras.layers.Input(shape=[5], name="wide\_input")

input\_B = keras.layers.Input(shape=[6], name="deep\_input")

hidden1 = keras.layers.Dense(30, activation="relu")(input\_B)

hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)

concat = keras.layers.concatenate([input\_A, hidden2])

output = keras.layers.Dense(1, name="output")(concat)

model = keras.models.Model(inputs=[input\_A, input\_B], outputs=[output])

model.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1e-3))

model.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1e-3))

X\_train\_A, X\_train\_B = X\_train[:, :5], X\_train[:, 2:]

X\_valid\_A, X\_valid\_B = X\_valid[:, :5], X\_valid[:, 2:]

X\_test\_A, X\_test\_B = X\_test[:, :5], X\_test[:, 2:]

X\_new\_A, X\_new\_B = X\_test\_A[:3], X\_test\_B[:3]

history = model.fit((X\_train\_A, X\_train\_B), y\_train, epochs=20,

                    validation\_data=((X\_valid\_A, X\_valid\_B), y\_valid))

mse\_test = model.evaluate((X\_test\_A, X\_test\_B), y\_test)

y\_pred = model.predict((X\_new\_A, X\_new\_B))

model.compile(loss=["mse", "mse"], loss\_weights=[0.9, 0.1], optimizer=keras.optimizers.SGD(lr=1e-3))

**Neurons per hidden layer**

Determined by the type of input and output

e.g 28 \* 28 -> 10 output neurons

- pyramid with fewer and fewer neurons each layer. Many low level features can coalse into far fewer high level features. Make the first hidden layer bigger than the other layers. – selected more neurons than required and use early stopping and regularization techniques to prevent from over fitting. – Stretch pants approach. Larger layers so information is not lost by not including it. e.g 3d model and only have a 2d model.

L2 and l1

Other features to optimize

**Learning rate:**

**Optimizer:**

**Batch Size:**

**Activation Function**

**Number of iterations:**

Data Structures

Strings : tf.constant(“hello world”) and tf.constant([ord© for c in “cafe”])

Ragged array : tf.ragged.constant([[1,2],[],[3]])

Spare Tensor : tf.SparseTensor (indices=[0,1],[1,0],[2,3]])

Tensor Array : tf. Tensor Array (indices=[0,1],[1,0],[2,3]])

Sets : tf.set.union(a,b) : a = tf.constant([[1,5,9]]) b = tf.constant([[5,6,9,11]])

Queues : tf.queue.FIFOqueue()

Keras:

Layers: <https://keras.io/api/layers>

Keras.layers: Dense, Flattern, Conv2D, MaxPooling etc

**Problem workflow**

Regression: predict value

Logistic regression: classification prediction

**Regression**

Tersorflow basic construct:

* Define model, by layers: Dense, output shape and input shape
* Compile
* Fit
* Predict

**Define**

model = keras.Sequential([keras.layers.Dense(units=1, input\_shape=[1])])

model = keras.models.Sequential([

    keras.layers.Dense(30, activation="relu", input\_shape=X\_train.shape[1:]),

    keras.layers.Dense(1)

])

Input\_shape : [1] single value regression, [28,28] image, [32,1] row of data

Layer. Dense:

|  |  |
| --- | --- |
| Units | Represents the output size |
| activation | relu, sigmoid,softmax,softplus,than,selu,elu,softsign |
| Use bias |  |
| initializer | RandomNormal, RandomUniform |
| regularizes | L1,l2,l1\_l2 |
| constraints |  |
| input\_shape |  |

**Compile:**

 model.compile(optimizer='sgd', loss='mean\_squared\_error')

|  |  |
| --- | --- |
| optimizer | sgd |
| loss | mean\_squared\_error |
|  |  |

**Fit**

Apply x,y values and define epochs

model.fit(xs, ys, epochs=500)

**Predict**

result = model.predict([7.0])

note the x value to predict y is array [1] and the result is an array[1]

optimizers:

https://www.tensorflow.org/api\_docs/python/tf/keras/optimizers

|  |  |
| --- | --- |
| Adam |  |
| Adadelta |  |
| Adagrad |  |
| Adamax |  |
| Ftlr |  |
| Nadam |  |
| Optionmizer |  |
| RMSprop |  |
| SGD |  |

model.fit(

**Logistic regression: Image convolution**

* Large datasets
* Augmentation
* Transfer Learning
* Multiclass classifications

   from tensorflow.keras.preprocessing.image import ImageDataGenerator

    train\_datagen = ImageDataGenerator(rescale=1/255.0)

Augmentation

Perspective skewing

Elastic distortions

Shearing

Cropping

Mirroring

**Library:** “Import augmentor from pipeline”

**Sources**

Google: image recognition

Microsoft: ResNet

Compile

|  |  |
| --- | --- |
| optimizer | adam |
| Loss | sparse\_categorical\_crossentropy, binary\_crossentropy |
|  |  |

model = tf.keras.models.Sequential([

        tf.keras.layers.Flatten(input\_shape=[28, 28]),

        tf.keras.layers.Dense(512, activation=tf.nn.relu),

        tf.keras.layers.Dense(10, activation=tf.nn.softmax)

    ])

sparese\_categorical\_crossentropy

model = tf.keras.models.Sequential([

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),

    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Flatten(input\_shape = [28,28]),

    tf.keras.layers.Dense(128, activation='relu'),

    tf.keras.layers.Dense(10, activation='softmax')

])

binary\_crossentropy

 model = tf.keras.models.Sequential([

        # Your Code Here

        tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input\_shape=(150, 150, 3)),

        tf.keras.layers.MaxPooling2D(2, 2),

        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),

        tf.keras.layers.MaxPooling2D(2, 2),

        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),

        tf.keras.layers.MaxPooling2D(2, 2),

        tf.keras.layers.Flatten(),

        tf.keras.layers.Dense(128, activation='relu'),

        tf.keras.layers.Dense(1, activation='sigmoid')

    ])

**Callbacks**

Tensorboard

ModelCheckPoint

Weights

Model.h5

EarlyStoping

CSv Log: pd.read\_csv(csv\_file).head()ead()

LearningRateSchedule

ReduceROnPlateau

Custom: on\_epoch\_begin and on\_epoch\_end

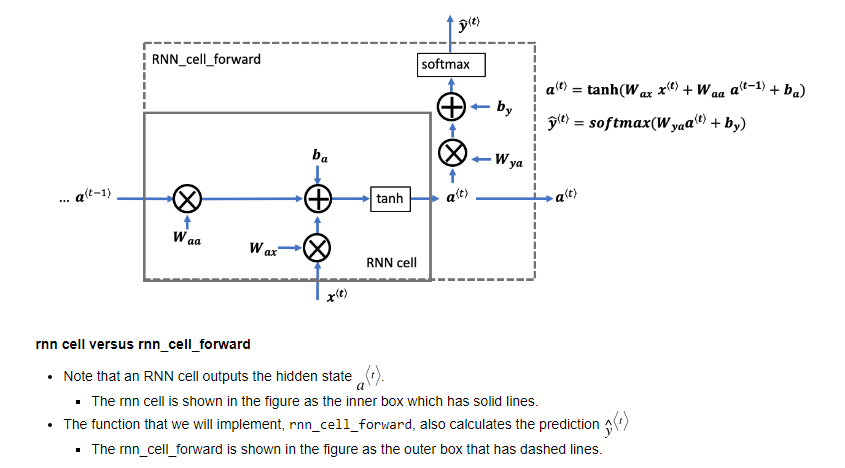
RNN -sequences

RNN best work on local context, where the y. where t’ is close to t

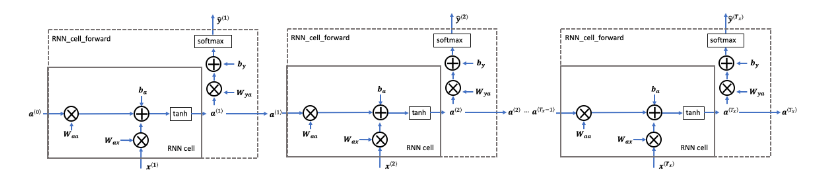
RNN suffer from

vanishing gradient, use dropout of clipping

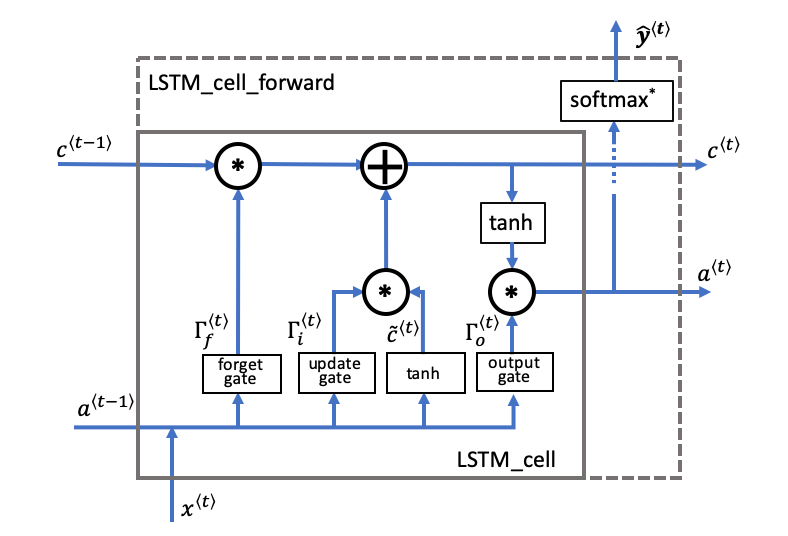
Can’t predict future



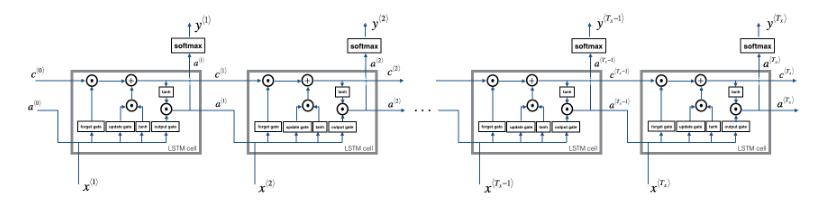
**Timeseries**



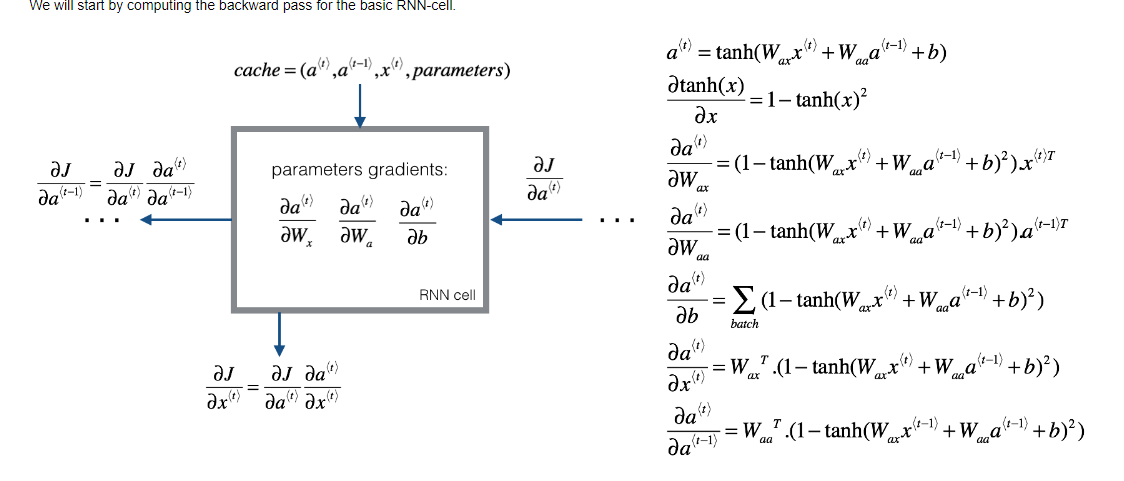
LSTM: back propagation and vanishing gradient



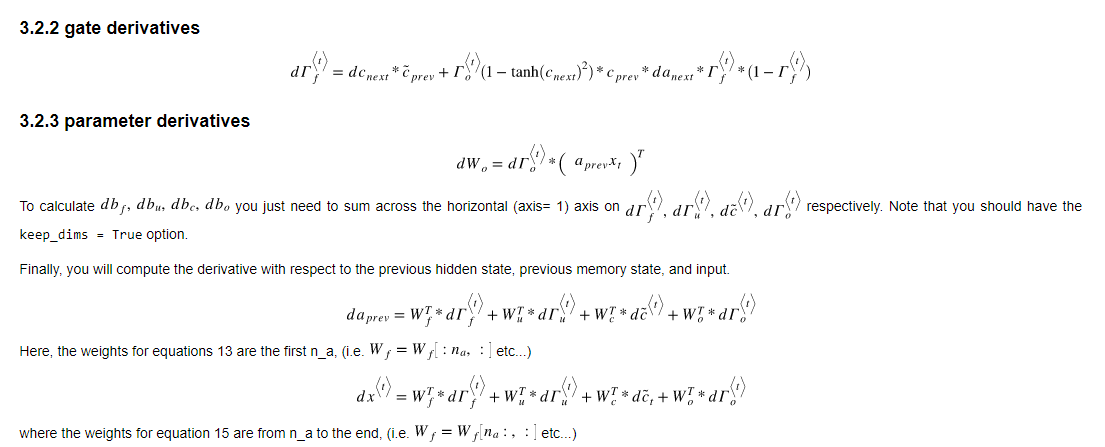
**Timeseries**

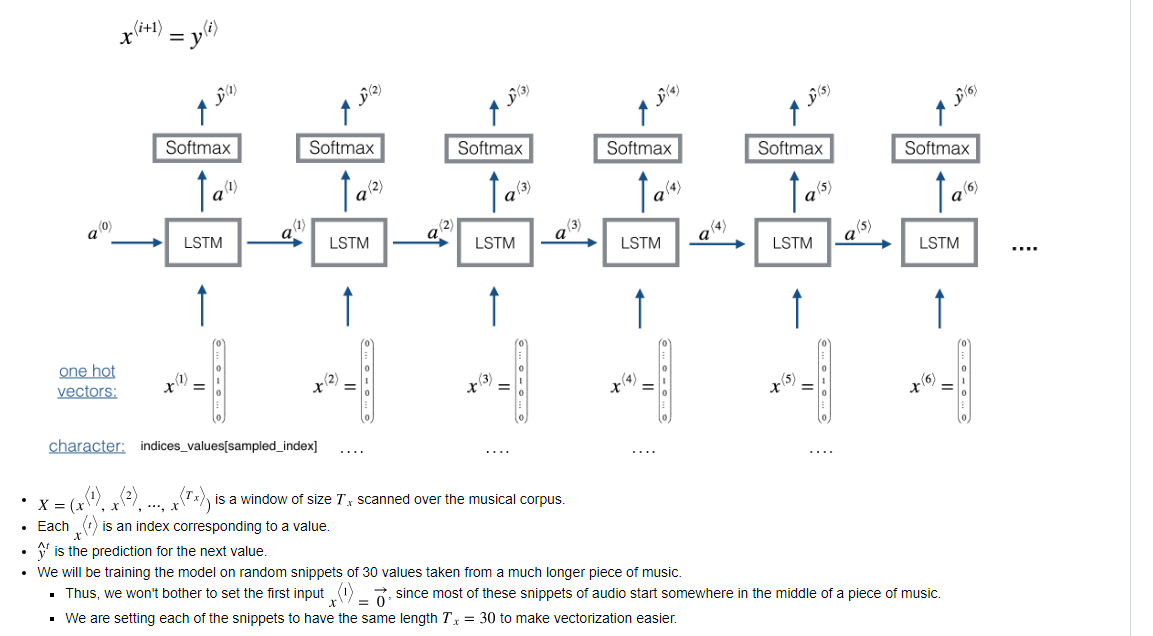


Back propagation



LSTM -Backpropagation





Tensorflow construct:

Define: Sequence model

Compile: loss = “mse“, optimizer(learning rate, momentum)

Fit: set datasets, epochs

Predict: test, validation

Measure: mean\_absolute\_error

Assign the data window:

def windowed\_dataset(series, window\_size, batch\_size, shuffle\_buffer):

    dataset = tf.data.Dataset.from\_tensor\_slices(series)

    dataset = dataset.window(window\_size + 1, shift=1, drop\_remainder=True)

    dataset = dataset.flat\_map(lambda window: window.batch(window\_size + 1))

    dataset = dataset.shuffle(shuffle\_buffer).map(lambda window: (window[:-1], window[-1]))

    dataset = dataset.batch(batch\_size).prefetch(1)

    return dataset

dataset = windowed\_dataset(x\_train, window\_size, batch\_size, shuffle\_buffer\_size)

model = tf.keras.models.Sequential([

    tf.keras.layers.Dense(100, input\_shape=[window\_size], activation="relu"),

    tf.keras.layers.Dense(10, activation="relu"),

    tf.keras.layers.Dense(1)

])

model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(learning\_rate=1e-6, momentum=0.9))

model.fit(dataset,epochs=100,verbose=0)

Trailing vs centred windows:

Sequence bias: trend,seasonality,noise

Use the learningratescheduler

Also change model.define: birectional and lambda functions

Layers: Custom layers – lambda layer, takes the previous layer and adds 1

Problems and solutions

Supervised and unsupervised learning

Logistic regression and linear regression

Cross validation: precision recall pg95

ROC: receiver operating charateristics

Error analysis:

Gradient descent,

cost function : mse, rmse

loss

bias/variance trade off:

regularized , ridge, lasso,elastic net regression: add to the cost function

logistic regression: sigmoid, softmax

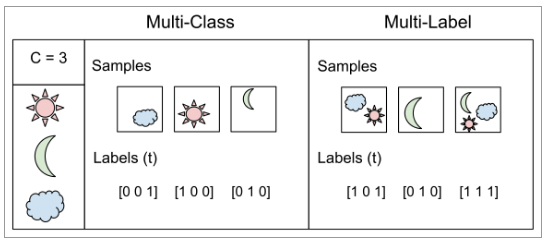
Cross entropy

**Cross Entropy**

Multi class classification: one-many, each sample belongs to one C of classes.

Multi label classification: each sample can belong to more than one class. The CNN will have many outputs.

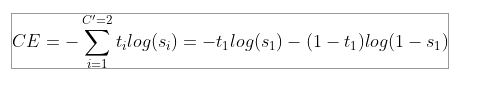
<https://gombru.github.io/2018/05/23/cross_entropy_loss/>

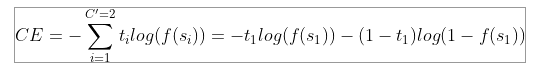


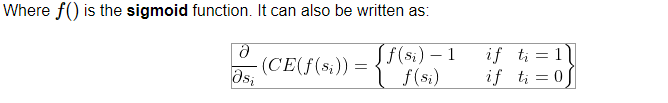
**Cross-Entropy Loss**



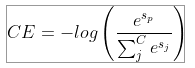
**Binary cross entropy where C’’ = 2. Assumed there are 2 classes C1 and C2 [0,1]**







**Categorical Cross-Entropy: Multi class classification**



**Categorical Crossentrop vs Sparse Categorical Crossentropy**

* categorical\_crossentropy (cce) produces a one-hot array containing the probable match for each category,
* sparse\_categorical\_crossentropy (scce) produces a category index of the *most likely* matching category.

Consider now a classification problem with 3 classes.

* In the case of cce, the one-hot target might be [0, 0, 1] and the model may predict [.5, .1, .4] (probably inaccurate, given that it gives more probability to the first class)
* In the case of scce, the target index might be [0], and the model may predict [.5]

Many categorical models produce scce output because you save space, but lose A LOT of information (for example, in the 2nd example, index 2 was also very close.) I generally prefer cce output for model reliability.

There are a number of situations to use scce, including:

* when your classes are mutually exclusive, i.e. you don't care at all about other close-enough predictions,
* the number of categories is large to the prediction output becomes overwhelming.