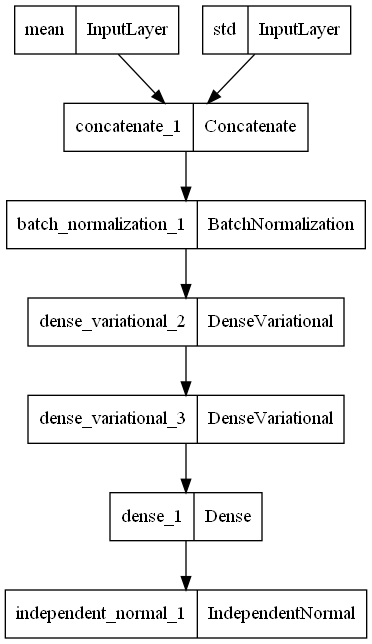
**Energie**

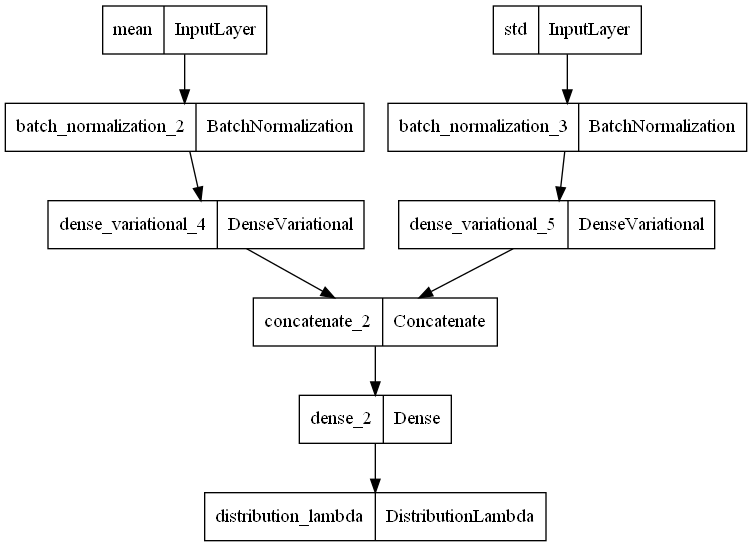
* SARIMA mit 24 h Perioden
  + Nur kleine Modelle möglich, Trainingszeit explodiert sehr schnell (Arima(order=c(3,1,1), seasonal=list(order=c(3,1,1), period=24)durch Ausprobieren, obwohl Zeitreihe stationär aussieht, scheint I = 1 besser zu sein)
* Wochenenden deutlich überschätzt
* SARIMAX: ergänze Dummy für Wochenenden
  + Dann muss Modell noch kleiner, damit Trainingsdauer vertretbar (d.h. unter 2 Stunden): Arima(order=c(2,1,1), seasonal=list(order=c(2,1,0), period=24)
* Kaum Effekt
* Zusätzlich Dummy für Feiertage und Produkt aus beiden Dummies
* Immer noch keine erhebliche Verbesserung
* Stattdessen eine Art target encoding:
  + Schneide Anfang der Zeitreihe vor 2018 ab (für Training reicht kürzerer Zeitraum)
  + Berechne darauf für jede Stunde des Tages den Mean bedingt auf Wochenende u/o Feiertag
  + Nutze das als Input
  + Zusätzlich hour\_of\_year Variable über Sinus/Cosinus (nutze nur Sinus, Cosinus stark insignifikant, obwohl eigentlich Jahresverlauf besser abgebildet wird (Beurteilung über Plot))
* Deutliche Verbesserung aber Wochenenden weiterhin überschätzt
* Außerdem Konfidenzintervalle sehr breit
* Schmeiße alle Daten weg, die nicht zu einer im Forecast gefragten Uhrzeit gehören
  + Dann Grid Search über p,d,q,P,Q,m
  + Kleinstes AIC hat auch fast kleinstes BIC
  + Wähle Modell Arima(order=c(1,0,2), seasonal=list(order=c(2,0,2), period=21)
* Jetzt deutlich schmalere KI und Wochenenden eindeutig erkennbar

**Wind**

* Idee: EMOS funktioniert so gut, nutze dieses Modell, aber schätze es mit Prob. Bayesian NN, dann Möglichkeit weitere Variablen hinzuzufügen
* Erster Versuch: Keras Tutorial mit tfp nachbauen mit Winddaten
  + nutze dazu drei Inputs: Forecast Hour, Ensemble Mean, Ensemble Varianz
  + Output Layer: IndependentNormal (negative Werte möglich, nutze als erstes Modell daher mean, sd aus Output und ziehe KI aber aus Truncated Normal Distribution mit gleichen Parametern)
* Nach Startschwierigkeiten mit Datentypen (Int32 vs. Int64) und Inputtensoren lauffähig, aber sehr schlechte Performance, stark nach unten verzerrt, KI-Breit glaubwürdig



* Architektur näher an EMOS und Output Layer zu Truncated Normal Distrubtion



* Verständnis aufgebaut mit Hilfe von tfp Tutorial, viele Plots angeschaut, Unterscheidung Aleatoric & Epistemic Uncertainty (<https://www.tensorflow.org/probability/examples/Probabilistic_Layers_Regression>)
  + Außerdem Learning Rate Scheduler, Early Stopping und Units=2\*100 (Rikes Tipp, muss besser noch selbst GridSearch laufen lassen), Anzahl Epochen wie in Tutorial deutlich höher (1000)
  + Plot: Was macht das Modell für Input Mean = 0.5, …, 20 bei konstant Varianz = 1
* Versuch: CRPS als loss-Funktion
  + Problem: nicht implementiert für tfp (andere Implementierungen in Python (CRPS.CRPS.compute(), properscoring.crps\_quadrature(y, p\_y), <https://github.com/slerch/ppnn/blob/master/nn_postprocessing/nn_src/losses.py>) können nicht mit Tensoren arbeiten bzw. kommen nicht mit Outputformat der Verteilung im PBNN klar; gibt Issues auf github zu tfp, die CRPS-Implementierung dafür fordern)
* CRPS Score Implementierung erfolgreich, hat kaum Einfluss auf Modellgüte
* Hyperparameter Tuning über GridSearchCV (n\_fold = 3) für learning rate und number of units (batch size trainieren in Kombi mit Tensorflow und sklearn erweist sich als kompliziert)
  + Learning rate: 0.01, Number of units: 64
* Activation function anpassen zu ELU (vgl. <https://archiv.ub.uni-heidelberg.de/volltextserver/30364/1/diss_pdfaversion.pdf>)
* Konfidenzintervalle des Forecasts haben bisher die epistemic uncertainty missachtet, berechne Quantile jetzt als Mittel aus 100 predictive distributions
* Werte immer noch deutlich zu niedrig -> muss standardisieren!
  + Ansatz: Teile Input Daten durch Maximum aus Trainingsdaten (Min zusätzlich abziehen könnte zu negativen Werten in Testdaten führen)
  + Vorhersagen sind nur noch Konstanten

Weitere Ideen/Todos:

* Scores berechnen/ Modellgüte auf historischen Daten
* Hyperparameter Tuning (Epochs, Batch size, …)
* Mehr Input Layer (Ensemble Forecasts zu anderen Wetter Features)
* In EMOS hat mean und std jeweils eine eigene Normalverteilung (schauen, ob Architektur besser daran anpassbar)

Literatur PBNN:

* <https://arxiv.org/pdf/2007.06823.pdf>
* <https://www.manning.com/books/probabilistic-deep-learning>
* <http://cran.nexr.com/web/packages/scoringRules/vignettes/crpsformulas.html#gamma-distribution-gamma>