Distributional Regression

ACTL3143 & ACTL5111 Deep Learning for Actuaries

Patrick Laub

Lecture Outline

- **Traditional Regression**
- Forecasts with noise
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- Mixture Density Network
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty

UNSW SYDNEY

Traditional Regression

Multiple linear regression assumes the data-generating process is

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \varepsilon$$

where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.

We estimate the coefficients $\beta_0, \beta_1, \ldots, \beta_p$ by minimising the sum of squared residuals or mean squared error

$$\text{RSS} := \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \quad \text{MSE} := \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

where $\hat{y}_i$ is the predicted value for the ith observation.

The probabilistic view

$$Y_i \sim \mathcal{N}(\mu_i, \sigma^2)$$

where $\mu_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}$, and the σ^2 is known.

The $\mathcal{N}(\mu, \sigma^2)$ normal distribution has p.d.f.

$$f(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right).$$

The likelihood function is

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i-\mu_i)^2}{2\sigma^2}\right)$$

$$\Rightarrow \ell(\boldsymbol{\beta}) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu_i)^2.$$

UNSW SYDNEY

The machine learning view

The negative log-likelihood $\text{NLL}(\boldsymbol{\beta}) := -\ell(\boldsymbol{\beta})$ is to be minimised:

$$\text{NLL}(\boldsymbol{\beta}) = \frac{n}{2}\log(2\pi) + \frac{n}{2}\log(\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu_i)^2.$$

As σ^2 is fixed, minimising NLL is equivalent to minimising MSE:

$$\begin{aligned}
\widehat{\boldsymbol{\beta}} &= \arg\min_{\boldsymbol{\beta}} \ \text{NLL}(\boldsymbol{\beta}) \\
&= \arg\min_{\boldsymbol{\beta}} \ \frac{n}{2}\log(2\pi) + \frac{n}{2}\log(\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu_i)^2 \\
&= \arg\min_{\boldsymbol{\beta}} \ \frac{1}{n}\sum_{i=1}^{n}\Big(y_i - \hat{y}_i(\boldsymbol{x}_i; \boldsymbol{\beta})\Big)^2 \\
&= \arg\min_{\boldsymbol{\beta}} \ \text{MSE}\big(\boldsymbol{y}, \hat{\boldsymbol{y}}(\mathbf{X}; \boldsymbol{\beta})\big).
\end{aligned}$$

UNSW SYDNEY

Generalised Linear Model (GLM)

The GLM is often characterised by the mean prediction:

$$\mu(\boldsymbol{x}; \boldsymbol{\beta}) = g^{-1}\left(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle\right)$$

where g is the link function.

Common GLM distributions for the response variable include:

- Normal distribution with identity link (just MLR)
- Bernoulli distribution with logit link (logistic regression)
- Poisson distribution with log link (Poisson regression)
- Gamma distribution with log link

Logistic regression

A Bernoulli distribution with parameter p has p.m.f.

$$f(y) = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{if } y = 0 \end{cases} = p^y(1-p)^{1-y}.$$

Our model is $Y|\boldsymbol{X} = \boldsymbol{x}$ follows a Bernoulli distribution with parameter

$$\mu(\boldsymbol{x}; \boldsymbol{\beta}) = \frac{1}{1 + \exp\left(-\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle\right)} = \mathbb{P}(Y = 1 | \boldsymbol{X} = \boldsymbol{x}).$$

The likelihood function, using $\mu_i := \mu(\boldsymbol{x}_i; \boldsymbol{\beta})$, is

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \begin{cases} \mu_i & \text{if } y_i = 1 \\ 1-\mu_i & \text{if } y_i = 0 \end{cases} = \prod_{i=1}^{n} \mu_i^{y_i}(1-\mu_i)^{1-y_i}.$$

UNSW SYDNEY

Binary cross-entropy loss

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \mu_i^{y_i}(1-\mu_i)^{1-y_i} \Rightarrow \ell(\boldsymbol{\beta}) = \sum_{i=1}^{n}\Big(y_i \log(\mu_i) + (1-y_i)\log(1-\mu_i)\Big).$$

The negative log-likelihood is

$$\text{NLL}(\boldsymbol{\beta}) = -\sum_{i=1}^{n}\Big(y_i \log(\mu_i) + (1-y_i)\log(1-\mu_i)\Big).$$

The binary cross-entropy loss is identical:

$$\text{BCE}(\boldsymbol{y}, \boldsymbol{\mu}) = -\sum_{i=1}^{n}\Big(y_i \log(\mu_i) + (1-y_i)\log(1-\mu_i)\Big).$$

UNSW SYDNEY

Poisson regression

A Poisson distribution with rate λ has p.m.f.

$$f(y) = \frac{\lambda^y \exp(-\lambda)}{y!}.$$

Our model is $Y|\boldsymbol{X} = \boldsymbol{x}$ is Poisson distributed with parameter

$$\mu(\boldsymbol{x}; \boldsymbol{\beta}) = \exp\left(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle\right).$$

The likelihood function is

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \frac{\mu_i^{y_i} \exp(-\mu_i)}{y_i!}$$

$$\Rightarrow \ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \Big(-\mu_i + y_i \log(\mu_i) - \log(y_i!) \Big).$$

UNSW SYDNEY

Poisson loss

The negative log-likelihood is

$$\text{NLL}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \Big(\mu_i - y_i \log(\mu_i) + \log(y_i!) \Big).$$

The Poisson loss is

$$\text{Poisson}(\boldsymbol{y}, \boldsymbol{\mu}) = \sum_{i=1}^{n} \Big(\mu_i - y_i \log(\mu_i) \Big).$$

Gamma regression

A gamma distribution with mean μ and dispersion ϕ has p.d.f.

$$f(y;\mu,\phi) = \frac{(\mu\phi)^{-\frac{1}{\phi}}}{\Gamma\left(\frac{1}{\phi}\right)} y^{\frac{1}{\phi}-1} \mathrm{e}^{-\frac{y}{\mu\phi}}$$

Our model is $Y|\boldsymbol{X} = \boldsymbol{x}$ is gamma distributed with a dispersion of ϕ and a mean of $\mu(\boldsymbol{x};\boldsymbol{\beta}) = \exp\left(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle\right)$.

The likelihood function is

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \frac{(\mu_i\phi)^{-\frac{1}{\phi}}}{\Gamma\left(\frac{1}{\phi}\right)} y_i^{\frac{1}{\phi}-1} \exp\left(-\frac{y_i}{\mu_i\phi}\right)$$

$$\Rightarrow \ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[-\frac{1}{\phi}\log(\mu_i\phi) - \log\Gamma\left(\frac{1}{\phi}\right) + \left(\frac{1}{\phi}-1\right)\log(y_i) - \frac{y_i}{\mu_i\phi} \right].$$

UNSW SYDNEY

Gamma loss

The negative log-likelihood is

$$\text{NLL}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[\frac{1}{\phi} \log(\mu_i \phi) + \log \Gamma\left(\frac{1}{\phi}\right) - \left(\frac{1}{\phi} - 1\right) \log(y_i) + \frac{y_i}{\mu_i \phi} \right].$$

Since ϕ is a nuisance parameter

$$\text{NLL}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[\frac{1}{\phi} \log(\mu_i) + \frac{y_i}{\mu_i \phi} \right] + \text{const} \propto \sum_{i=1}^{n} \left[\log(\mu_i) + \frac{y_i}{\mu_i} \right].$$

Note

As $\log(\mu_i) = \log(y_i) - \log(y_i/\mu_i)$, we could write an alternative version

$$\text{NLL}(\boldsymbol{\beta}) \propto \sum_{i=1}^{n} \left[\log(y_i) - \log\left(\frac{y_i}{\mu_i}\right) + \frac{y_i}{\mu_i} \right] \propto \sum_{i=1}^{n} \left[\frac{y_i}{\mu_i} - \log\left(\frac{y_i}{\mu_i}\right) \right].$$

UNSW SYDNEY

Why do actuaries use GLMs?

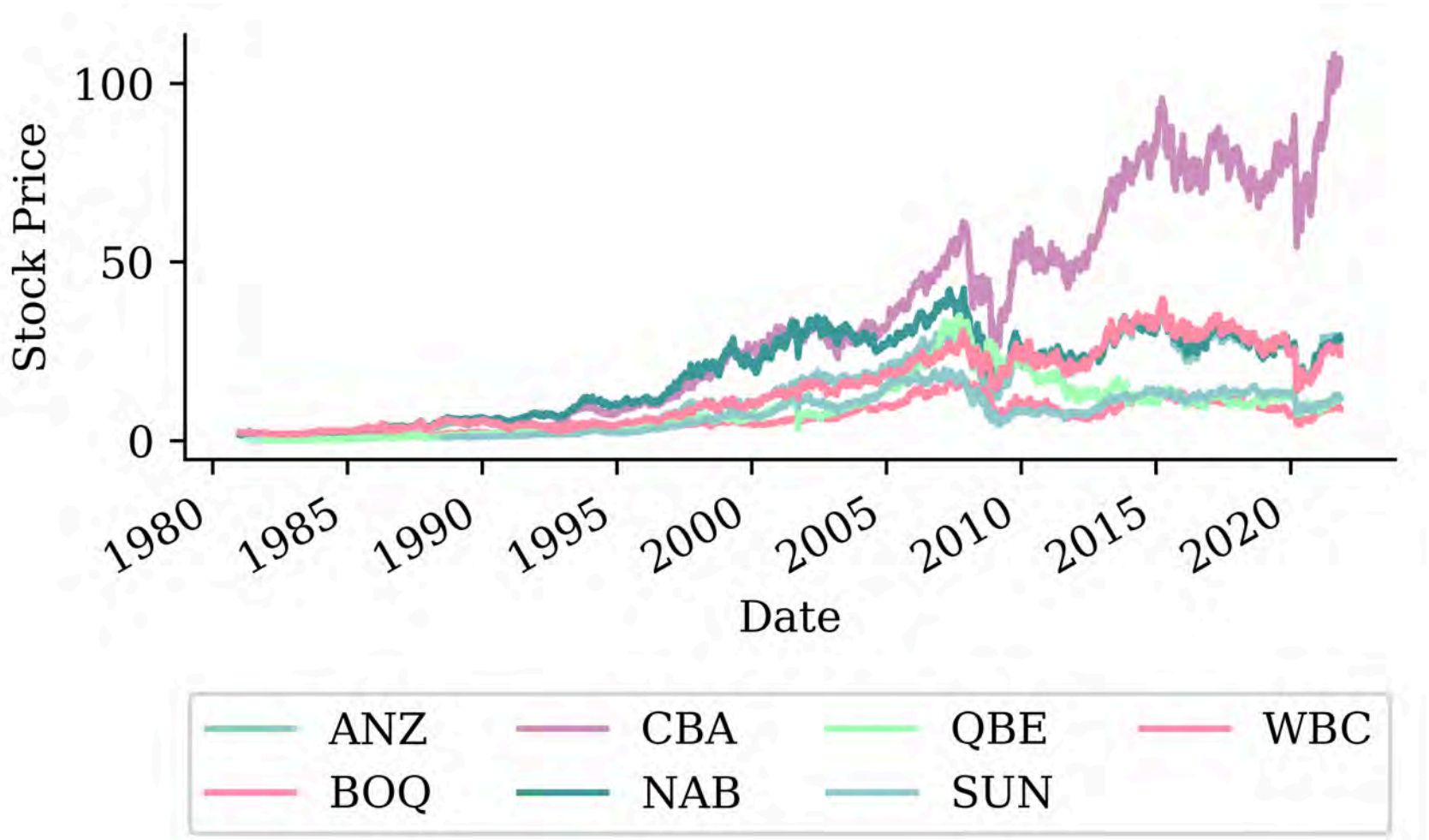
- GLMs are interpretable.
- GLMs are flexible (can handle different types of response variables).
- We get the full distribution of the response variable, not just the mean.

This last point is particularly important for analysing worst-case scenarios.

Lecture Outline

- Traditional Regression
- **Forecasts with noise**
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- Mixture Density Network
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty

Stock price forecasting



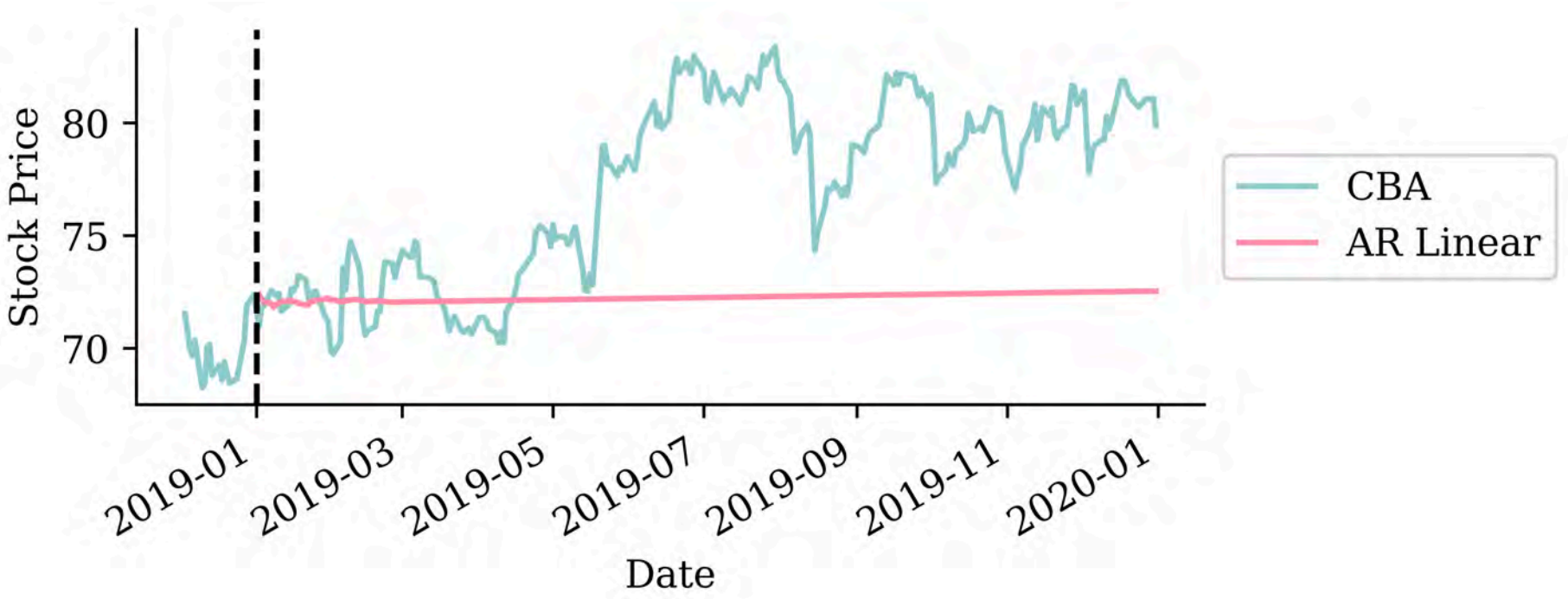
Noisy auto-regressive forecast

```python
def noisy_autoregressive_forecast(model, X_val, sigma, suppress=False):
    """
    Generate a multi-step forecast using the given model.
    """
    multi_step = pd.Series(index=X_val.index, name="Multi Step")

    # Initialize the input data for forecasting
    input_data = X_val.iloc[0].values.reshape(1, -1)

    for i in range(len(multi_step)):
        # Ensure input_data has the correct feature names
        input_df = pd.DataFrame(input_data, columns=X_val.columns)
        if suppress:
            next_value = model.predict(input_df, verbose=0)
        else:
            next_value = model.predict(input_df)

        next_value += np.random.normal(0, sigma)

        multi_step.iloc[i] = next_value

        # Append that prediction to the input for the next forecast
        if i + 1 < len(multi_step):
            input_data = np.append(input_data[:, 1:], next_value).reshape(1, -1)

    return multi_step
```

UNSW SYDNEY

Original forecast

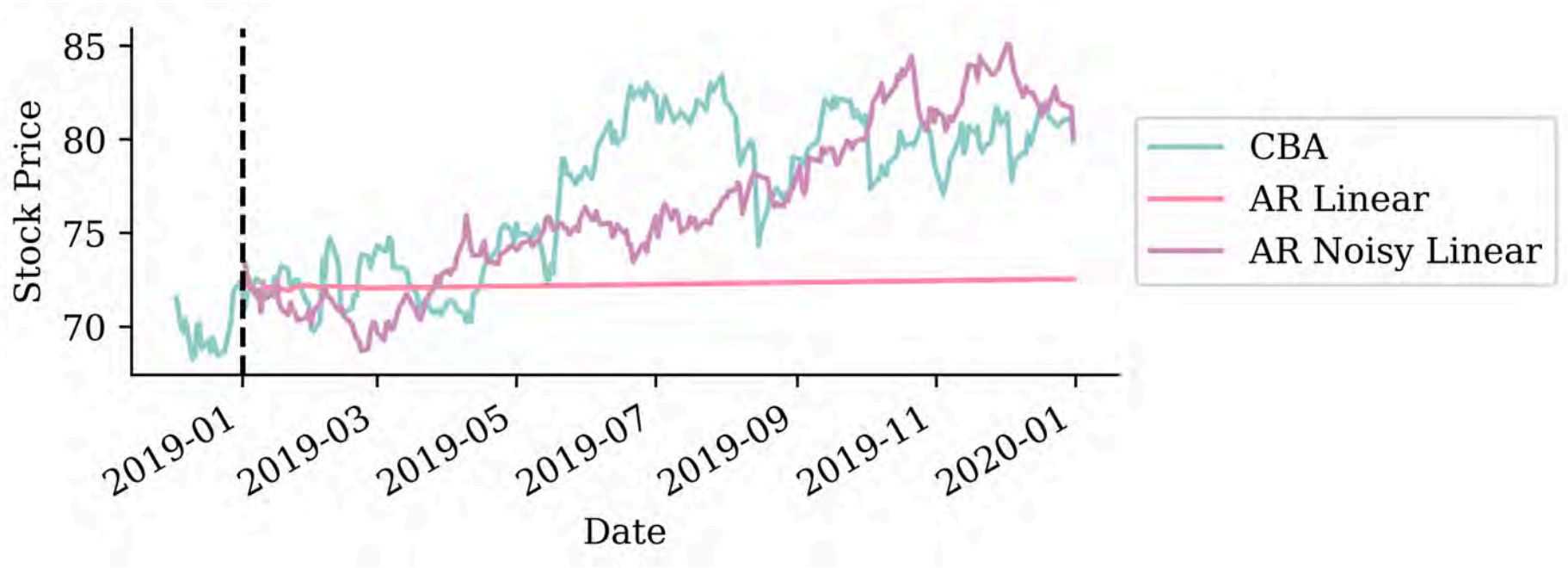
```
1 lr_forecast = noisy_autoregressive_forecast(lr, X_val, 0)
```




```
1 residuals = y_train - lr.predict(X_train)
2 sigma = np.std(residuals)
```

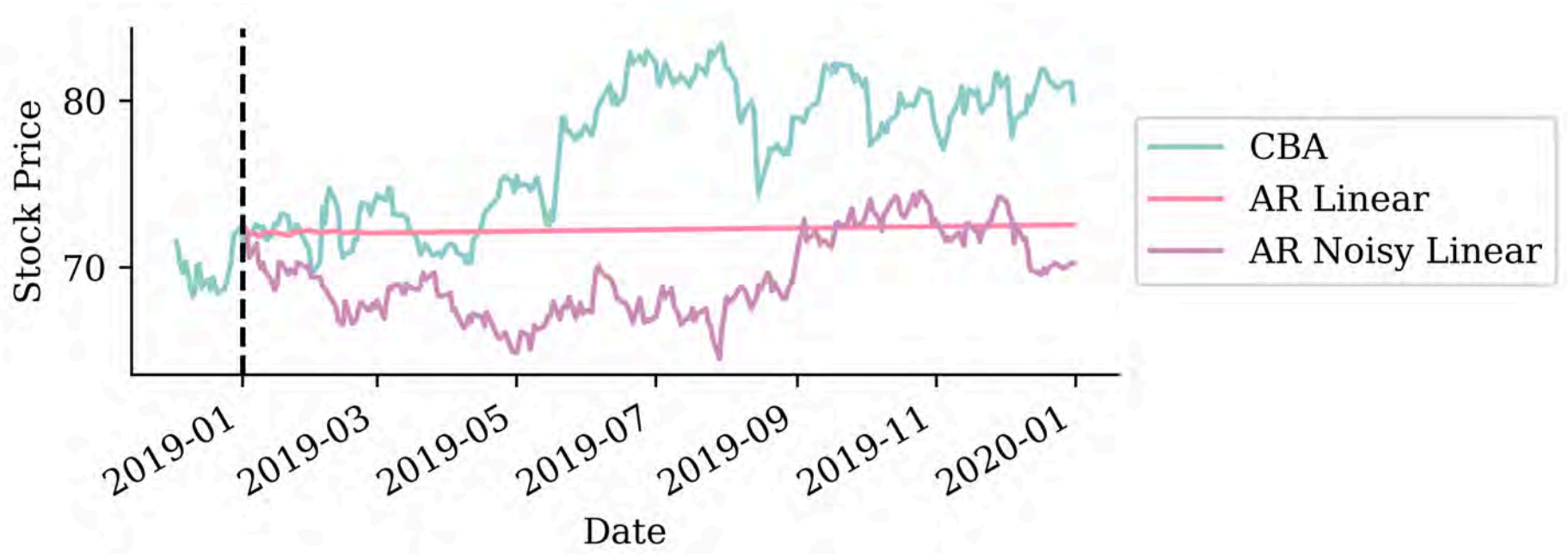
With noise

```
np.random.seed(1)
lr_noisy_forecast = noisy_autoregressive_forecast(lr, X_val, sigma)
```



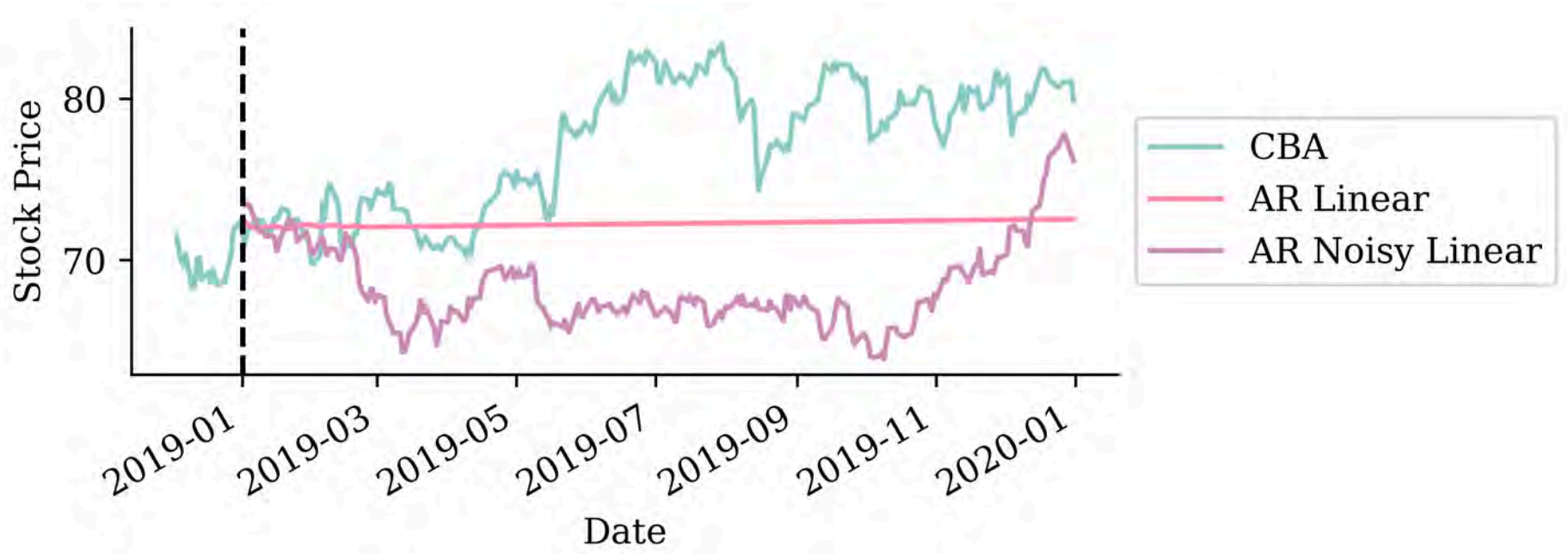
With noise

```
1 np.random.seed(2)
2 lr_noisy_forecast = noisy_autoregressive_forecast(lr, X_val, sigma)
```



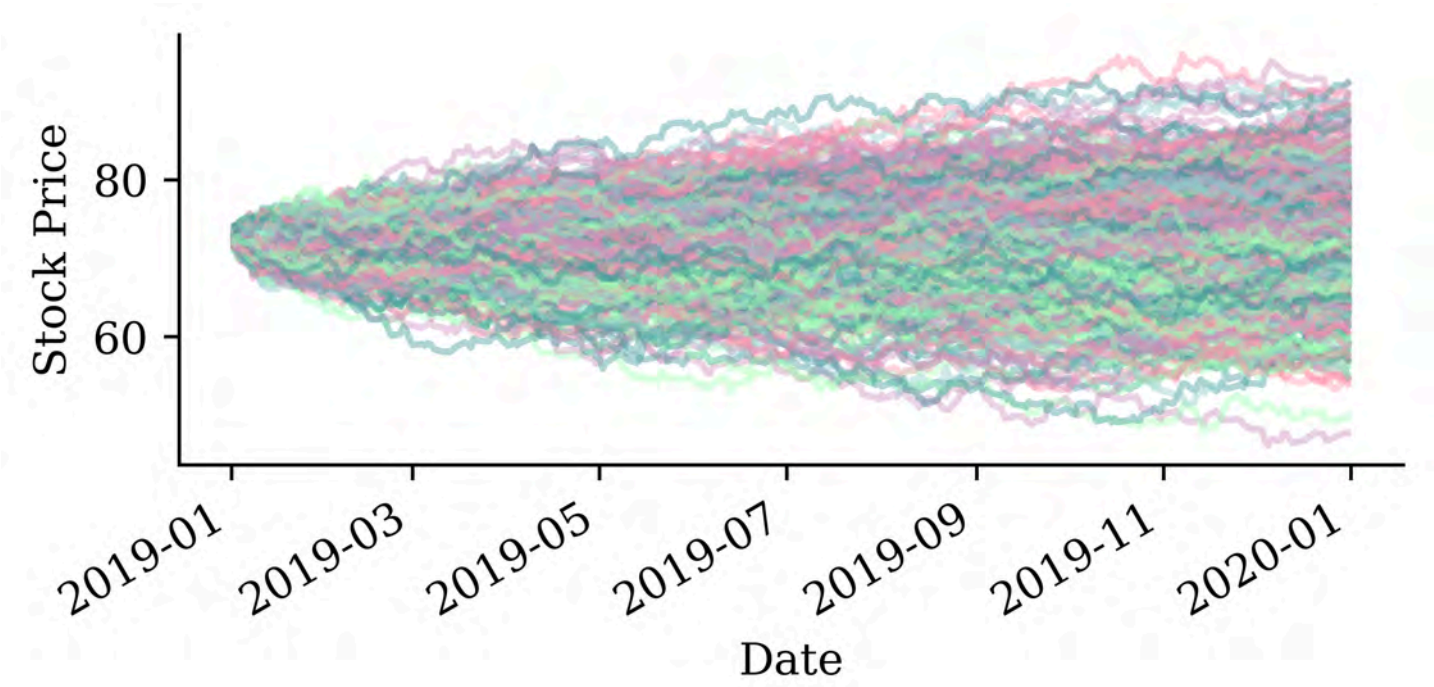
With noise

```
1 np.random.seed(3)
2 lr_noisy_forecast = noisy_autoregressive_forecast(lr, X_val, sigma)
```



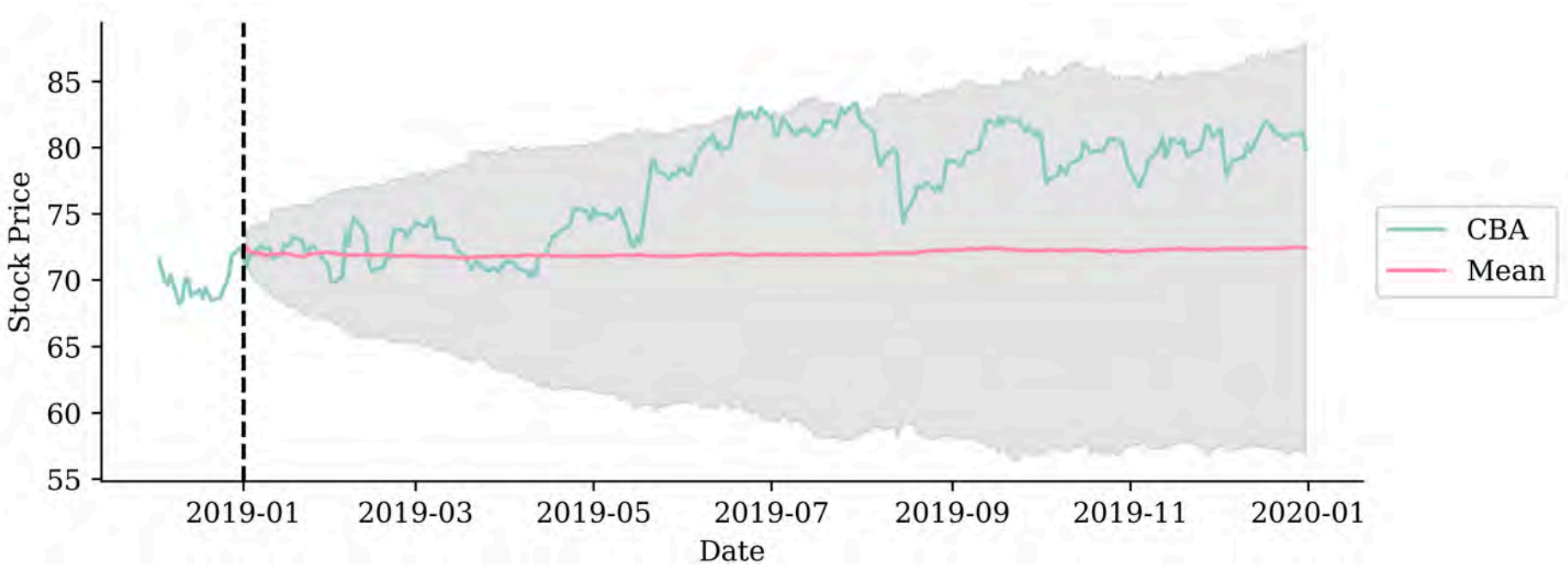
Many noisy forecasts

```
num_forecasts = 500
forecasts = []
for i in range(num_forecasts):
    forecasts.append(noisy_autoregressive_forecast(lr, X_val, sigma) * 100)
noisy_forecasts = pd.concat(forecasts, axis=1)
noisy_forecasts.index = X_val.index
```



95% “prediction intervals”

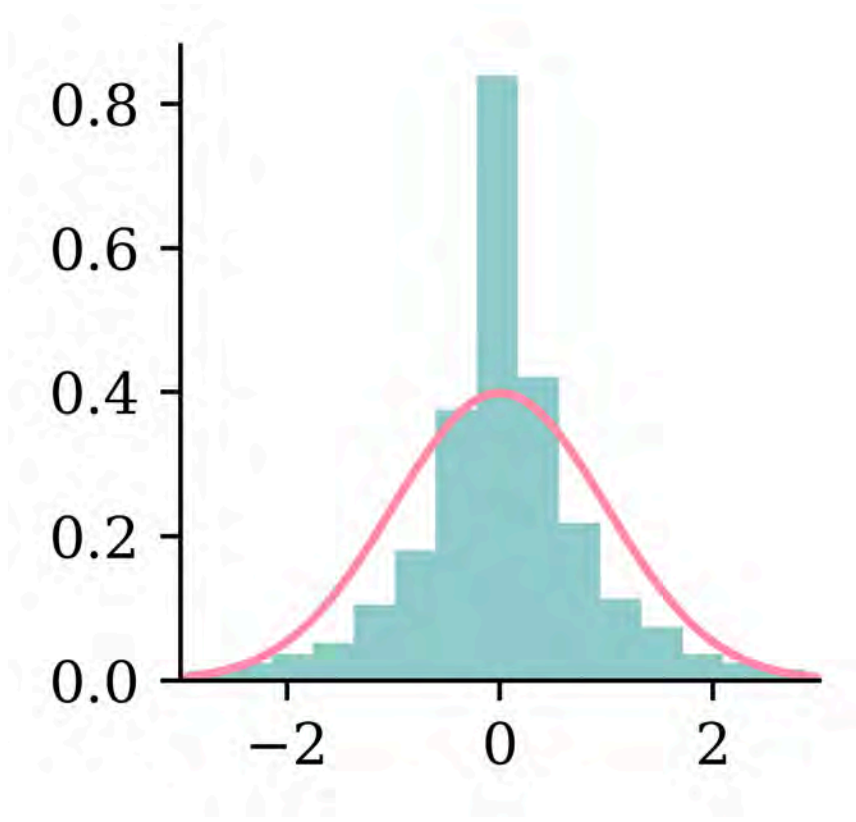
```python
# Calculate quantiles for the forecasts
lower_quantile = noisy_forecasts.quantile(0.025, axis=1)
upper_quantile = noisy_forecasts.quantile(0.975, axis=1)
mean_forecast = noisy_forecasts.mean(axis=1)
```



Residuals

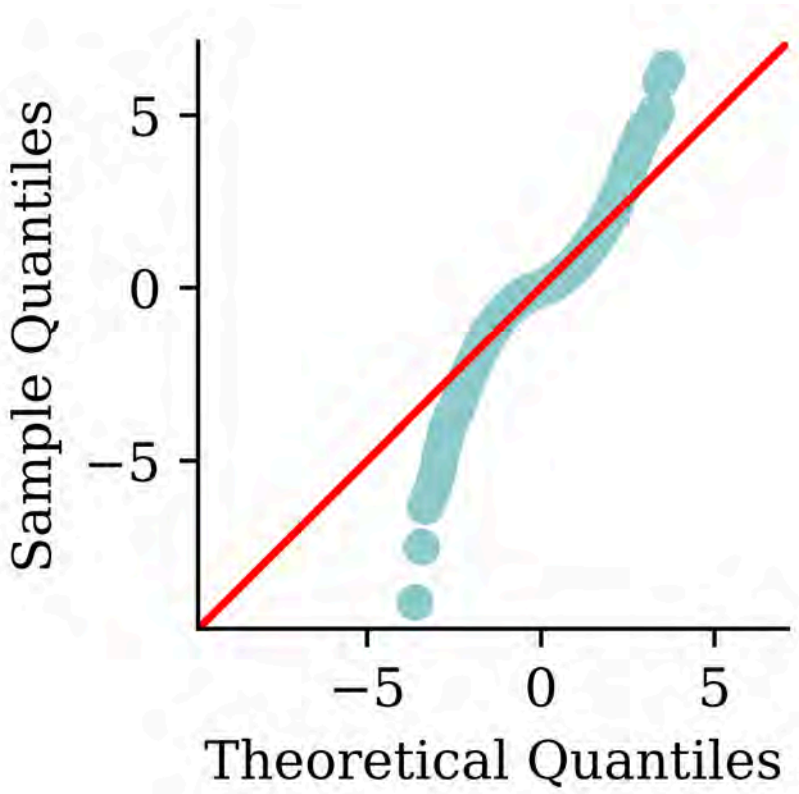
```
1  y_pred = lr.predict(X_train)
2  residuals = y_train - y_pred
3  residuals -= np.mean(residuals)
4  residuals /= np.std(residuals)
5  stats.shapiro(residuals)
```

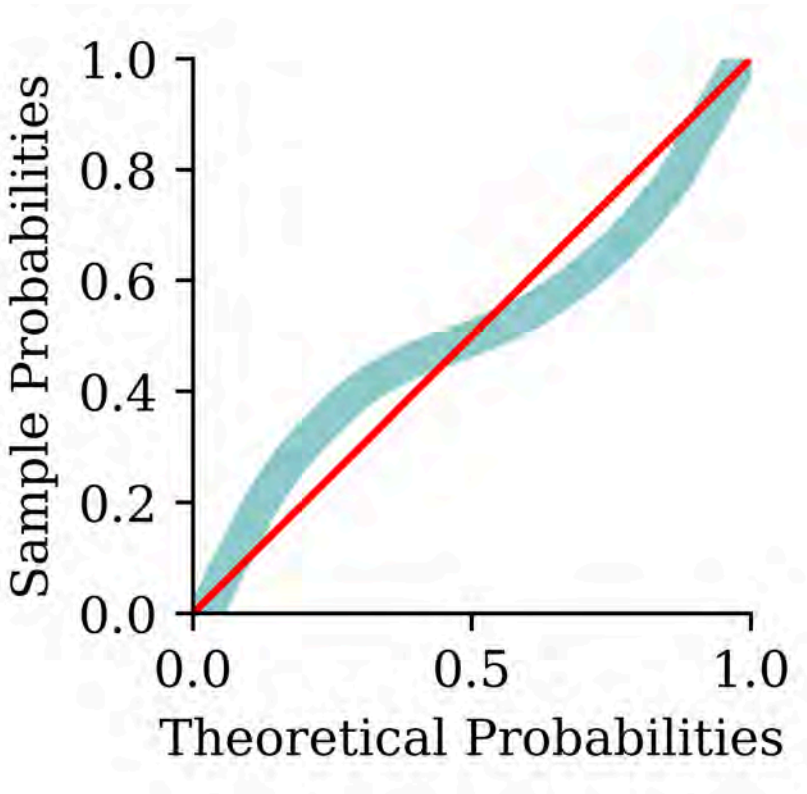
```
ShapiroResult(statistic=0.9038059115409851,
pvalue=0.0)
```



UNSW SYDNEY

Q-Q plot and P-P plot





Lecture Outline

- Traditional Regression
- Forecasts with noise
- **GLMs and Neural Networks**
- Combined Actuarial Neural Network
- Mixture Density Network
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty

UNSW SYDNEY

Code: Data

```
1  import pandas as pd
2  sev_df = pd.read_csv('freMTPL2sev.csv')
3  freq_df = pd.read_csv('freMTPL2freq.csv')
4
5  # Create a copy of freq dataframe without 'claimfreq' column
6  freq_without_claimfreq = freq_df.drop(columns=['ClaimNb'])
7
8  # Merge severity dataframe with freq_without_claimfreq dataframe
9  new_sev_df = pd.merge(sev_df, freq_without_claimfreq, on='IDpol',
10                                                    how='left')
11 new_sev_df = new_sev_df.dropna()
12 new_sev_df = new_sev_df.drop("IDpol", axis=1)
13 new_sev_df[:2]
```

	ClaimAmount	Exposure	VehPower	VehAge	DrivAge	Bor
0	995.20	0.59	11.0	0.0	39.0	56.(
1	1128.12	0.95	4.0	1.0	49.0	50.(

Code: Preprocessing

```
X_train, X_test, y_train, y_test = train_test_split(
  new_sev_df.drop("ClaimAmount", axis=1),
  new_sev_df["ClaimAmount"],
  random_state=2023)

# Reset each index to start at 0 again.
X_train = X_train.reset_index(drop=True)
X_test = X_test.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
```

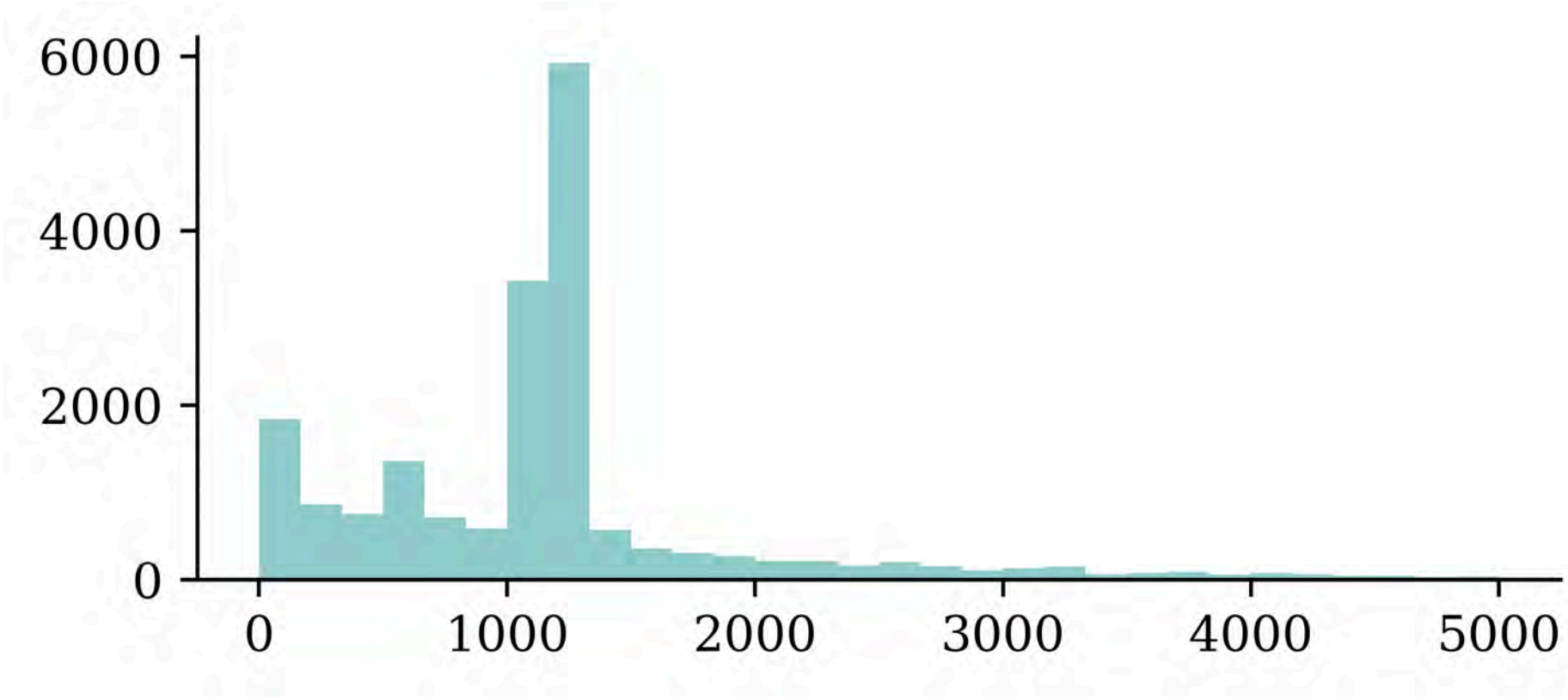
Code: Preprocessing

```
# Transformation
ct = make_column_transformer(
  (OrdinalEncoder(), ["VehBrand", "Region", "Area", "VehGas"]),
  remainder=StandardScaler(),
   verbose_feature_names_out=False
)

# We don't apply entity embedding
X_train_ct = ct.fit_transform(X_train)
X_test_ct = ct.transform(X_test)
X_train = X_train_ct.drop(["VehBrand", "Region"], axis=1)
X_test = X_test_ct.drop(["VehBrand", "Region"], axis=1)
```

- `VehGas=1` if the car gas is regular.
- `Area=0` represents the rural area, and `Area=5` represents the urban center.

Histogram of the `ClaimAmount`

```
1 plt.hist(y_train[y_train < 5000], bins=30);
```



Gamma GLM

Suppose a fitted gamma GLM model has

- a log link function $g(x) = \log(x)$ and
- regression coefficients $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, \beta_3)$.

Then, it estimates the conditional mean of Y given a new instance $\boldsymbol{x} = (1, x_1, x_2, x_3)$ as follows:

$$\mathbb{E}[Y|\boldsymbol{X} = \boldsymbol{x}] = g^{-1}(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle) = \exp\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3\right).$$

A GLM can model any other exponential family distribution using an appropriate link function g.

Gamma GLM loss

If $Y|\boldsymbol{X} = \boldsymbol{x}$ is a gamma r.v. with mean $\mu(\boldsymbol{x}; \boldsymbol{\beta})$ and dispersion parameter ϕ, we can minimise the negative log-likelihood (NLL)

$$\text{NLL} \propto \sum_{i=1}^{n} \log \mu(\boldsymbol{x}_i; \boldsymbol{\beta}) + \frac{y_i}{\mu(\boldsymbol{x}_i; \boldsymbol{\beta})} + \text{const},$$

i.e., we ignore the dispersion parameter ϕ while estimating the regression coefficients.

Fitting Steps

Step 1. Use the advanced second derivative iterative method to find the regression coefficients:

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} \log \mu(\boldsymbol{x}_i; \boldsymbol{\beta}) + \frac{y_i}{\mu(\boldsymbol{x}_i; \boldsymbol{\beta})}$$

Step 2. Estimate the dispersion parameter:

$$\phi = \frac{1}{n-p} \sum_{i=1}^{n} \frac{(y_i - \mu(\boldsymbol{x}_i; \boldsymbol{\beta}))^2}{\mu(\boldsymbol{x}_i; \boldsymbol{\beta})^2}$$

Code: Gamma GLM

In Python, we can fit a gamma GLM as follows:

```python
import statsmodels.api as sm

# Add a column of ones to include an intercept in the model
X_train_design = sm.add_constant(X_train)

# Create a Gamma GLM with a log link function
gamma_glm = sm.GLM(y_train, X_train_design,
                   family=sm.families.Gamma(sm.families.links.Log()))

# Fit the model
gamma_glm = gamma_glm.fit()
```

```python
gamma_glm.params
```

```
const          7.786576
Area          -0.073226
VehGas         0.082292
                  ...
DrivAge       -0.022147
BonusMalus     0.157204
Density        0.010539
Length: 9, dtype: float64
```

```python
# Dispersion Parameter
mus = gamma_glm.predict(X_train_design)
residuals = y_train - mus
variance = mus**2
dof = (len(y_train)-X_train.shape[1])
phi_glm =  np.sum(residuals**2/variance
print(phi_glm)
```

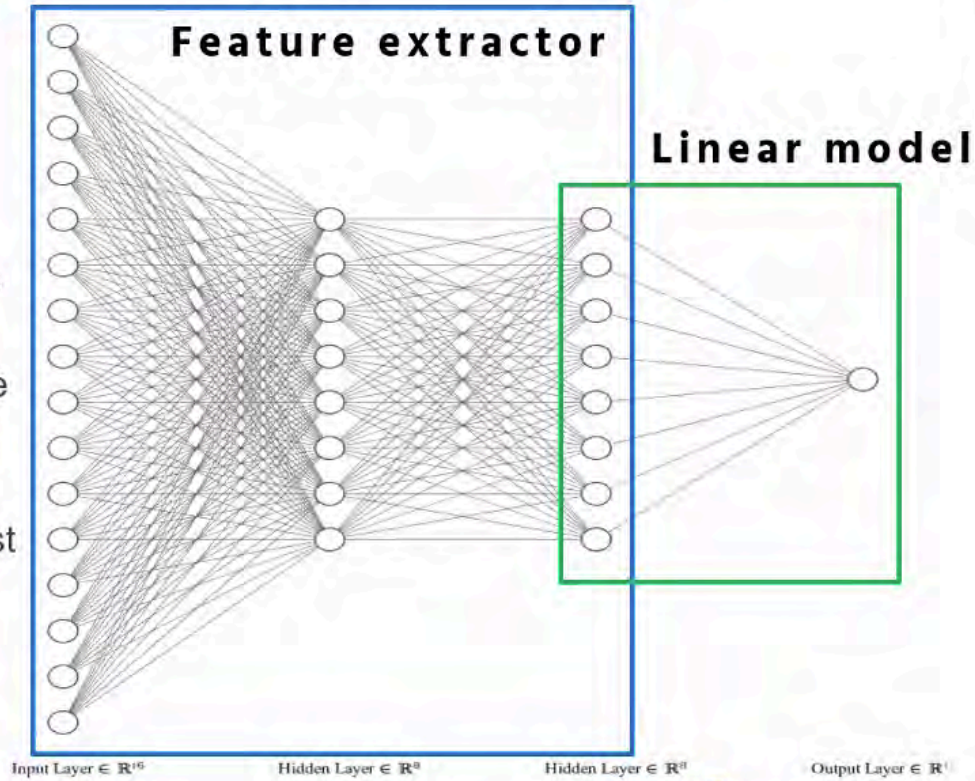
```
59.6306232357824
```

UNSW SYDNEY

ANN can feed into a GLM

FCN generalizes GLM

- Intermediate layers = representation learning, guided by supervised objective.
- Last layer = (generalized) linear model, where input variables = new representation of data
- No need to use GLM – strip off last layer and use learned features in, for example, XGBoost
- Or mix with traditional method of fitting GLM



Institute and Faculty of Actuaries

11 April 2022

14

Combining GLM & ANN.

Source: Ronald Richman (2022), Mind the Gap - Safely Incorporating Deep Learning Models into the Actuarial Toolkit, IFoA seminar, Slide 14.

Lecture Outline

- Traditional Regression
- Forecasts with noise
- GLMs and Neural Networks
- **Combined Actuarial Neural Network**
- Mixture Density Network
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty

UNSW SYDNEY

CANN

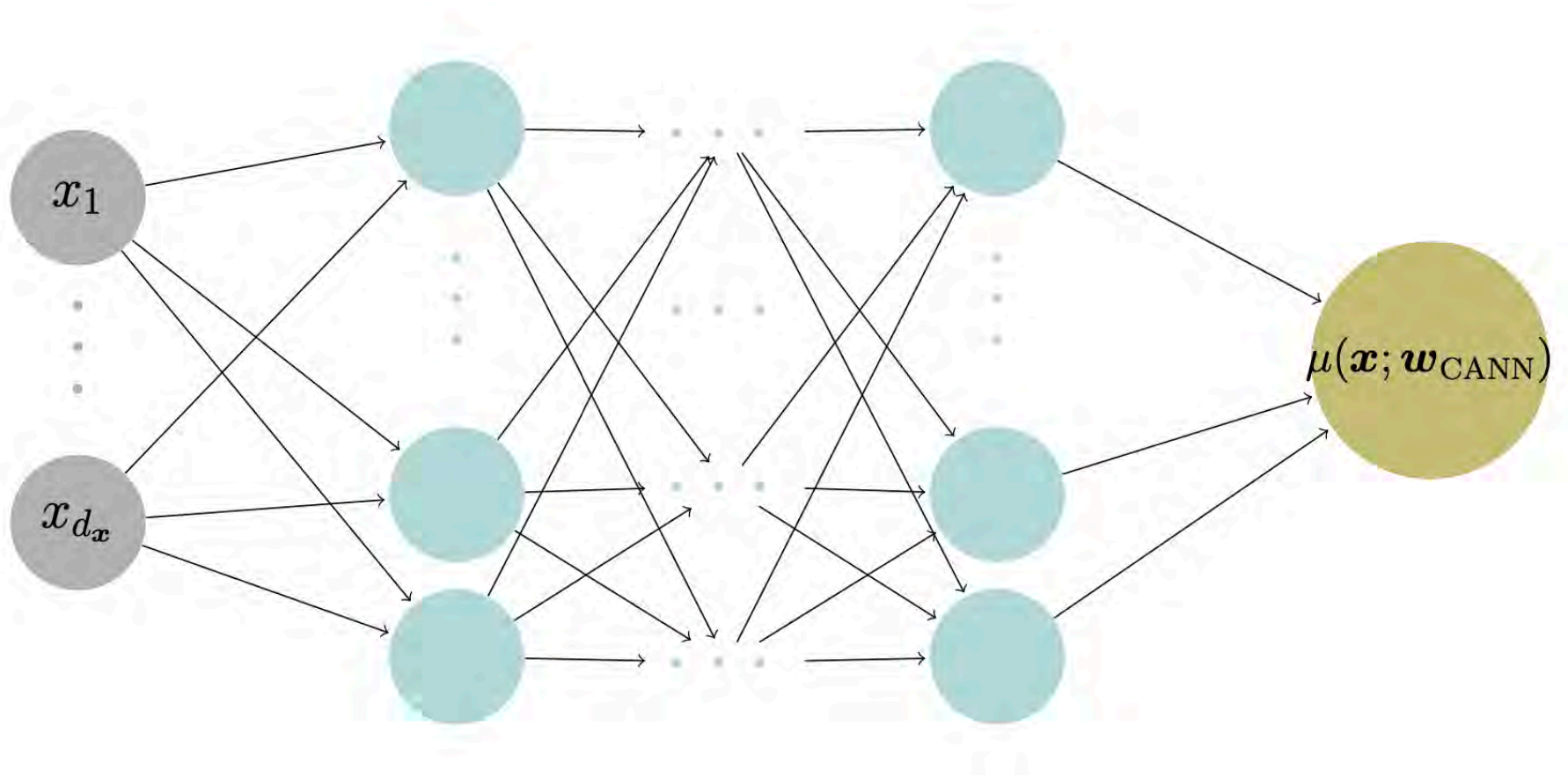
The Combined Actuarial Neural Network is a novel actuarial neural network architecture proposed by Schelldorfer and Wüthrich (2019). We summarise the CANN approach as follows:

- Find the coefficients $\boldsymbol{\beta}$ of the GLM with a link function $g(\cdot)$.
- Find the weights $\boldsymbol{w}_{\text{CANN}}$ of a neural network $\mathcal{M}_{\text{CANN}} : \mathbb{R}^p \to \mathbb{R}$.
- Given a new instance $\boldsymbol{x}$, we have

$$\mathbb{E}[Y|\boldsymbol{X} = \boldsymbol{x}] = g^{-1}\Big(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle + \mathcal{M}_{\text{CANN}}(\boldsymbol{x}; \boldsymbol{w}_{\text{CANN}})\Big).$$

UNSW SYDNEY

Architecture



CANN approach.

Code: Architecture

```
# Ensure reproducibility
random.seed(1); tf.random.set_seed(1)

# Pre-defined constants
glm_weights = gamma_glm.params.iloc[1:]
glm_bias = gamma_glm.params.iloc[0]

# Define model inputs
inputs = Input(shape=X_train.shape[1:])

# Non-trainable GLM linear part
glm_logmu = Dense(1, activation='linear', trainable=False,
                     kernel_initializer=Constant(glm_weights),
                     bias_initializer=Constant(glm_bias))(inputs)

# Neural network layers
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
cann_logmu = Dense(1, activation='linear')(x)
```

UNSW SYDNEY

Code: Loss Function

```
# Combine GLM and CANN estimates
cann = Model(inputs, Concatenate(axis=1)([cann_logmu, glm_logmu]))
```

We need to customise the loss function for CANN.

```
def cann_negative_log_likelihood(y_true, y_pred):
    #the new mean estimate
    cann_logmu = y_pred[:, 0]
    glm_logmu = y_pred[:, 1]
    mu = tf.math.exp(cann_logmu + glm_logmu)

    # Compute the negative log likelihood of the Gamma distribution
    nll = tf.reduce_mean(cann_logmu + glm_logmu + y_true/mu)

    return nll
```

UNSW SYDNEY

Code: Model Training

```
cann.compile(optimizer="adam", loss=cann_negative_log_likelihood)
hist = cann.fit(X_train, y_train,
    epochs=100,
    callbacks=[EarlyStopping(patience=10)],
    verbose=0,
    batch_size=64,
    validation_split=0.2)
```

Find the dispersion parameter.

```
mus = np.exp(np.sum(cann.predict(X_train, verbose=0), axis = 1))
residuals = y_train - mus
variance = mus**2
dof = (len(y_train)-X_train.shape[1])
phi_cann =  np.sum(residuals**2/variance) / dof
print(phi_cann)
```

```
86.69498963886436
```

UNSW SYDNEY

Lecture Outline

- Traditional Regression
- Forecasts with noise
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- **Mixture Density Network**
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty

UNSW SYDNEY

Mixture Distribution

Given a finite set of resulting random variables $(Y_1, ..., Y_K)$, one can generate a multinomial random variable $Y \sim \text{Multinomial}(1, \boldsymbol{\pi})$. Meanwhile, Y can be regarded as a mixture of $Y_1, ..., Y_K$, i.e.,

$$Y = \begin{cases} Y_1 & \text{w.p. } \pi_1, \\ \vdots & \vdots \\ Y_K & \text{w.p. } \pi_K, \end{cases}$$

where we define a set of finite set of weights $\boldsymbol{\pi} = (\pi_1..., \pi_K)$ such that $\pi_k \geq 0$ for $k \in \{1, ..., K\}$ and $\sum_{k=1}^{K} \pi_k = 1$.

UNSW SYDNEY

Mixture Distribution

Let $f_{Y_k|\boldsymbol{X}}$ and $F_{Y_k|\boldsymbol{X}}$ be the p.d.f. and the c.d.f of $Y_k|\boldsymbol{X}$ for all $k \in \{1, ..., K\}$.

The random variable $Y|\boldsymbol{X}$, which mixes $Y_k|\boldsymbol{X}$'s with weights π_k's, has the density function

$$f_{Y|\boldsymbol{X}}(y|\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k(\boldsymbol{x}) f_k(y|\boldsymbol{x}),$$

and the cumulative density function

$$F_{Y|\boldsymbol{X}}(y|\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k(\boldsymbol{x}) F_k(y|\boldsymbol{x}).$$

Mixture Density Network

A mixture density network (MDN) $\mathcal{M}_{\boldsymbol{w}^*}$ outputs each distribution component's mixing weights and parameters of Y given the input features $\boldsymbol{x}$, i.e.,

$$\mathcal{M}_{\boldsymbol{w}^*}(\boldsymbol{x}) = (\boldsymbol{\pi}(\boldsymbol{x}; \boldsymbol{w}^*), \boldsymbol{\theta}(\boldsymbol{x}; \boldsymbol{w}^*)),$$

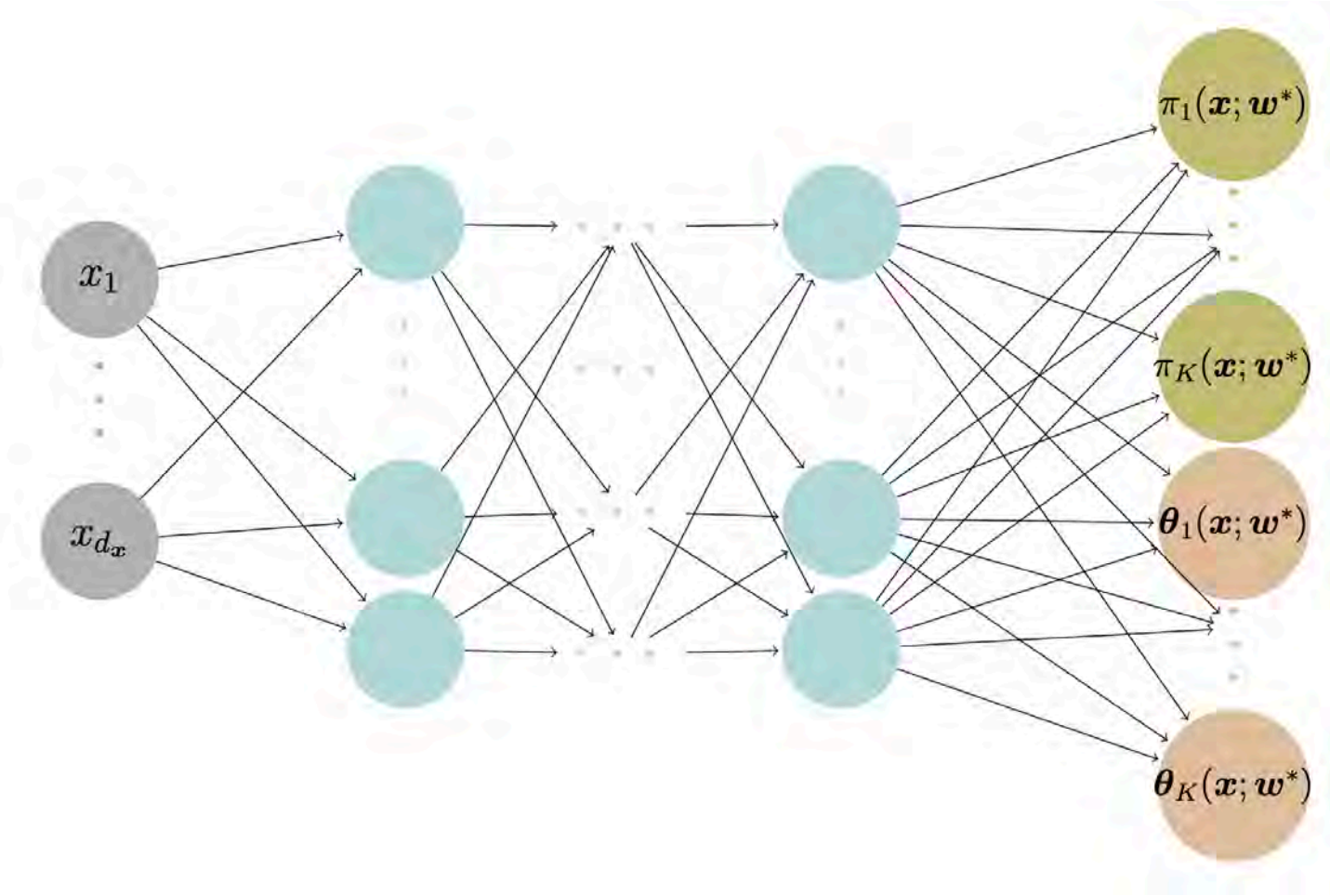
where $\boldsymbol{w}^*$ is the networks' weights found by minimising the following negative log-likelihood loss function

$$\mathcal{L}(\mathcal{D}, \boldsymbol{\theta}) = -\sum_{i=1}^{n} \log f_{Y|\boldsymbol{X}}(y_i|\boldsymbol{x}, \boldsymbol{w}^*),$$

where $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ is the training dataset.

UNSW SYDNEY

Mixture Density Network



An MDN that outputs the parameters for a K component mixture distribution. $\boldsymbol{\theta}_k(\boldsymbol{x}; \boldsymbol{w}^*) = (\theta_{k,1}(\boldsymbol{x}; \boldsymbol{w}^*), ..., \theta_{k,|\boldsymbol{\theta}_k|}(\boldsymbol{x}; \boldsymbol{w}^*))$ consists of the parameter estimates for the kth mixture component.

UNSW SYDNEY

Model Specification

Suppose there are two types of claims:

- Type I: $Y_1|\boldsymbol{X} = \boldsymbol{x} \sim \text{Gamma}(\alpha_1(\boldsymbol{x}), \beta_1(\boldsymbol{x}))$ and,
- Type II: $Y_2|\boldsymbol{X} = \boldsymbol{x} \sim \text{Gamma}(\alpha_2(\boldsymbol{x}), \beta_2(\boldsymbol{x}))$.

The density of the actual claim amount $Y|\boldsymbol{X} = \boldsymbol{x}$ follows

$$f_{Y|\boldsymbol{X}}(y|\boldsymbol{x}) = \pi_1(\boldsymbol{x}) \cdot \frac{\beta_1(\boldsymbol{x})^{\alpha_1(\boldsymbol{x})}}{\Gamma(\alpha_1(\boldsymbol{x}))} \mathrm{e}^{-\beta_1(\boldsymbol{x})y} y^{\alpha_1(\boldsymbol{x})-1} + (1-\pi_1(\boldsymbol{x})) \cdot \frac{\beta_2(\boldsymbol{x})^{\alpha_2(\boldsymbol{x})}}{\Gamma(\alpha_2(\boldsymbol{x}))} \mathrm{e}^{-\beta_2(\boldsymbol{x})y} y^{\alpha_2(\boldsymbol{x})-1}.$$

where $\pi_1(\boldsymbol{x})$ is the probability of a Type I claim given $\boldsymbol{x}$.

UNSW SYDNEY

Output

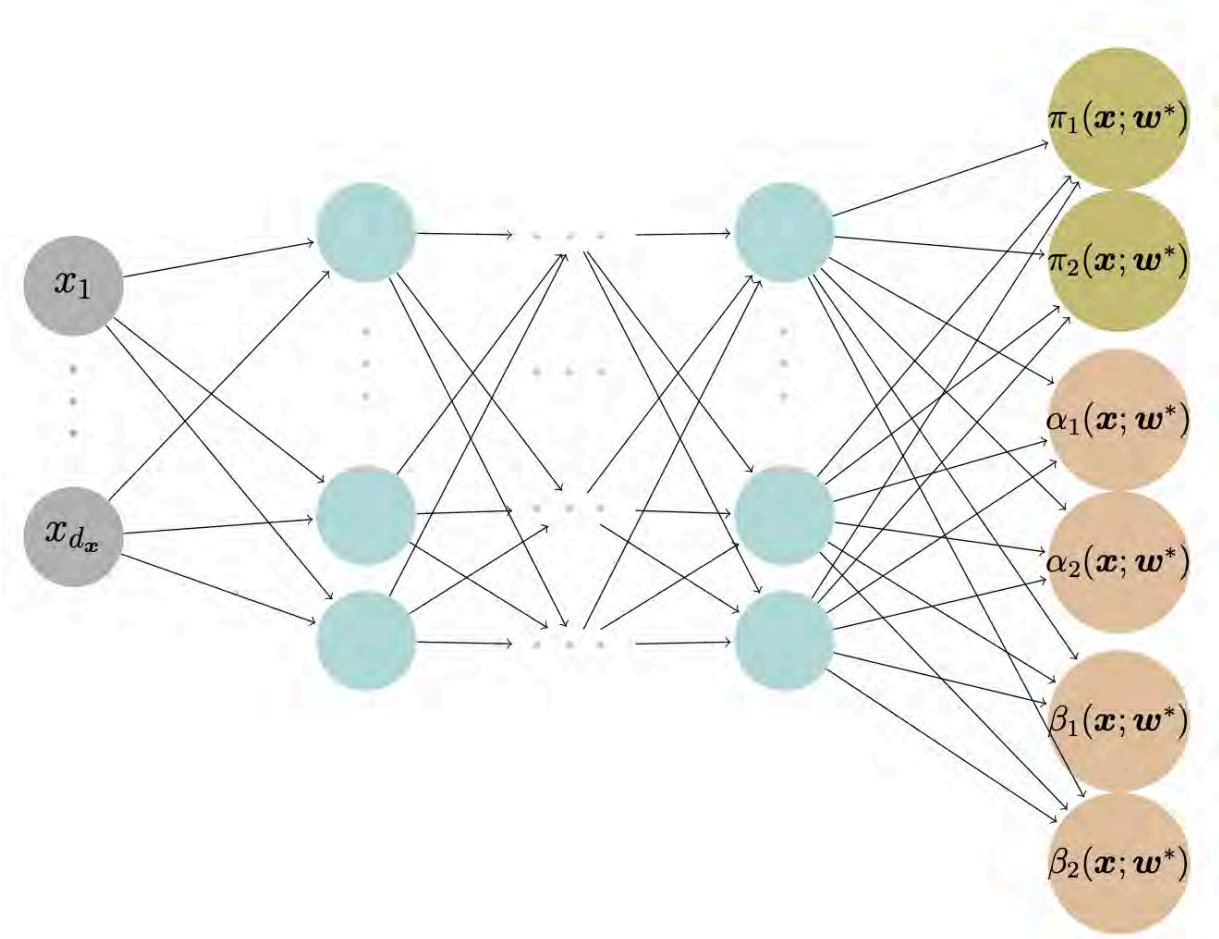
The aim is to find the optimum weights

$$\boldsymbol{w}^* = \arg\min_{w} \mathcal{L}(\mathcal{D}, \boldsymbol{w})$$

for the Gamma mixture density network $\mathcal{M}_{\boldsymbol{w}^*}$ that outputs the mixing weights, shapes and scales of Y given the input features $\boldsymbol{x}$, i.e.,

$$\begin{aligned}\mathcal{M}_{\boldsymbol{w}^*}(\boldsymbol{x}) = (&\pi_1(\boldsymbol{x}; \boldsymbol{w}^*), \pi_2(\boldsymbol{x}; \boldsymbol{w}^*),\\ &\alpha_1(\boldsymbol{x}; \boldsymbol{w}^*), \alpha_2(\boldsymbol{x}; \boldsymbol{w}^*),\\ &\beta_1(\boldsymbol{x}; \boldsymbol{w}^*), \beta_2(\boldsymbol{x}; \boldsymbol{w}^*)).\end{aligned}$$

Architecture



We demonstrate the structure of a gamma MDN that outputs the parameters for a gamma mixture with two components.

UNSW SYDNEY

Code: Architecture

The following code resembles the architecture of the architecture of the gamma MDN from the previous slide.

```
# Ensure reproducibility
random.seed(1); tf.random.set_seed(1)

inputs = Input(shape=X_train.shape[1:])

# Two hidden layers
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)

pis = Dense(2, activation='softmax')(x) #mixing weights
alphas = Dense(2, activation='exponential')(x) #shape parameters
betas = Dense(2, activation='exponential')(x) #scale parameters

# `y_pred` will now have 6 columns
gamma_mdn = Model(inputs, Concatenate(axis=1)([pis, alphas, betas]))
```

Loss Function

The negative log-likelihood loss function is given by

$$\mathcal{L}(\mathcal{D}, \boldsymbol{w}) = -\sum_{i=1}^{n} \log \; f_{Y|\boldsymbol{X}}(y_i|\boldsymbol{x}, \boldsymbol{w})$$

where the $f_{Y|\boldsymbol{X}}(y_i|\boldsymbol{x}, \boldsymbol{w})$ is defined by

$$\pi_1(\boldsymbol{x}; \boldsymbol{w}) \cdot \frac{\beta_1(\boldsymbol{x}; \boldsymbol{w})^{\alpha_1(\boldsymbol{x};\boldsymbol{w})}}{\Gamma(\alpha_1(\boldsymbol{x}; \boldsymbol{w}))} \mathrm{e}^{-\beta_1(\boldsymbol{x};\boldsymbol{w})y} y^{\alpha_1(\boldsymbol{x};\boldsymbol{w})-1}$$

$$+ (1 - \pi_1(\boldsymbol{x}; \boldsymbol{w})) \cdot \frac{\beta_2(\boldsymbol{x}; \boldsymbol{w})^{\alpha_2(\boldsymbol{x};\boldsymbol{w})}}{\Gamma(\alpha_2(\boldsymbol{x}; \boldsymbol{w}))} \mathrm{e}^{-\beta_2(\boldsymbol{x};\boldsymbol{w})y} y^{\alpha_2(\boldsymbol{x};\boldsymbol{w})-1}$$

UNSW SYDNEY

Code: Loss Function

We employ functions from `tensorflow_probability` to code the loss function for the gamma MDN. The `MixtureSameFamily` function facilitates defining a mixture distribution all components from the same distribution but have different parametrization.

```
import tensorflow_probability as tfp
tfd = tfp.distributions
K = 2 # number of mixture components

def gamma_mixture_nll(y_true, y_pred):
    K = y_pred.shape[1] // 3
    pis =  y_pred[:, :K]
    alphas = y_pred[:, K:2*K]
    betas = y_pred[:, 2*K:3*K]

    # The mixture distribution is a MixtureSameFamily distribution
    mixture_distribution = tfd.MixtureSameFamily(
        mixture_distribution=tfd.Categorical(probs=pis),
        components_distribution=tfd.Gamma(alphas, betas))

    # The loss is the negative log-likelihood of the data
    return -mixture_distribution.log_prob(y_true)
```

UNSW SYDNEY

Code: Model Training

```
# Employ the loss function from previous slide
gamma_mdn.compile(optimizer="adam", loss=gamma_mixture_nll)

hist = gamma_mdn.fit(X_train, y_train,
    epochs=100,
    callbacks=[EarlyStopping(patience=10)],
    verbose=0,
    batch_size=64,
    validation_split=0.2)
```

UNSW SYDNEY

Lecture Outline

- Traditional Regression
- Forecasts with noise
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- Mixture Density Network
- **Metrics for Distributional Regression**
- Aleatoric and Epistemic Uncertainty

UNSW SYDNEY

Proper Scoring Rules

Definition

The scoring rule $S : \mathcal{F} \times \mathbb{R} \to \bar{\mathbb{R}}$ is proper relative to the class $\mathcal{F}$ if

$$S(G, G) \leq S(F, G)$$

for all $F, G \in \mathcal{F}$. It is strictly proper if equality holds only if $F = G$.

Examples:

- Logarithmic Score (NLL)
- Continuous Ranked Probability Score (CRPS)

UNSW SYDNEY

Proper Scoring Rules

Logarithmic Score (NLL)

The logarithmic score is defined as

$$\text{LogS}(f, y) = -\log f(y),$$

where f is the predictive density.

Continuous Ranked Probability Score (CRPS)

The continuous ranked probability score is defined as

$$\text{crps}(F, y) = \int_{-\infty}^{\infty} (F(t) - 1_{t \geq y})^2 \, \mathrm{d}t,$$

where F is the cumulative distribution function.

Code: NLL

```python
from scipy.stats import gamma

def gamma_nll(mean, dispersion, y):
    # Calculate shape and scale parameters from mean and dispersion
    shape = 1 / dispersion; scale = mean * dispersion

    # Create a gamma distribution object
    gamma_dist = gamma(a=shape, scale=scale)

    return -np.mean(gamma_dist.logpdf(y))

# GLM
X_test_design = sm.add_constant(X_test)
mus = gamma_glm.predict(X_test_design)
nll_glm = gamma_nll(mus, phi_glm, y_test)

# CANN
mus = np.exp(np.sum(cann.predict(X_test, verbose=0), axis = 1))
nll_cann = gamma_nll(mus, phi_cann, y_test)

# MDN
nll_mdn = gamma_mdn.evaluate(X_test, y_test, verbose=0)
```

Model Comparisons

```
1 print(f'GLM: {round(nll_glm, 2)}')
2 print(f'CANN: {round(nll_cann, 2)}')
3 print(f'MDN: {round(nll_mdn, 2)}')
```

```
GLM: 11.02
CANN: 11.38
MDN: 8.67
```

UNSW SYDNEY

Lecture Outline

- Traditional Regression
- Forecasts with noise
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- Mixture Density Network
- Metrics for Distributional Regression
- **Aleatoric and Epistemic Uncertainty**

Categories of uncertainty

There are two major categories of uncertainty in statistical or machine learning:

- Aleatoric uncertainty
- Epistemic uncertainty

Since there is no consensus on the definitions of aleatoric and epistemic uncertainty, we provide the most acknowledged definitions in the following slides.

Aleatoric Uncertainty

Qualitative Definition

Aleatoric uncertainty refers to the statistical variability and inherent noise with data distribution that modelling cannot explain.

Quantitative Definition

$$\text{Ale}(Y|\boldsymbol{X}=\boldsymbol{x}) = \mathbb{V}[Y|\boldsymbol{X}=\boldsymbol{x}],$$

i.e., if $Y|\boldsymbol{X}=\boldsymbol{x} \sim \mathcal{N}(\mu, \sigma^2)$, the aleatoric uncertainty would be σ^2. Simply, it is the conditional variance of the response variable Y given features/covariates $\boldsymbol{x}$.

UNSW SYDNEY

Epistemic Uncertainty

Qualitative Definition

Epistemic uncertainty refers to the lack of knowledge, limited data information, parameter errors and model errors.

Quantitative Definition

$$\text{Epi}(Y|\boldsymbol{X}=\boldsymbol{x}) = \text{Uncertainty}(Y|\boldsymbol{X}=\boldsymbol{x}) - \text{Ale}(Y|\boldsymbol{X}=\boldsymbol{x}),$$

i.e., the total uncertainty subtracting the aleatoric uncertainty $\mathbb{V}[Y|\boldsymbol{X}=\boldsymbol{x}]$ would be the epistemic uncertainty.

UNSW SYDNEY

Sources of uncertainty

If you decide to predict the claim amount of an individual using a deep learning model, which source(s) of uncertainty are you dealing with?

1. The inherent variability of the data-generating process → aleatoric uncertainty.
2. Parameter error → epistemic uncertainty.
3. Model error → epistemic uncertainty.
4. Data uncertainty → epistemic uncertainty.

Notation

- scalars are denoted by lowercase letters, e.g., y,
- vectors are denoted by bold lowercase letters, e.g.,

$$\boldsymbol{y} = (y_1, \ldots, y_n),$$

- random variables are denoted by capital letters, e.g., Y
- random vectors are denoted by bold capital letters, e.g.,

$$\boldsymbol{X} = (X_1, \ldots, X_p),$$

- matrices are denoted by bold uppercase non-italics letters, e.g.,

$$\mathbf{X} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}.$$

Regression notation

- n is the number of observations, p is the number of features,
- the true coefficients are $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)$,
- β_0 is the intercept, $\beta_1, \ldots, \beta_p$ are the coefficients,
- $\widehat{\boldsymbol{\beta}}$ is the estimated coefficient vector,
- $\boldsymbol{x}_i = (1, x_{i1}, x_{i2}, \ldots, x_{ip})$ is the feature vector for the ith observation,
- y_i is the response variable for the ith observation,
- $\hat{y}_i$ is the predicted value for the ith observation,
- probability density functions (p.d.f.), probability mass functions (p.m.f.), cumulative distribution functions (c.d.f.).

Package Versions

```
1 from watermark import watermark
2 print(watermark(python=True, packages="keras,matplotlib,numpy,pandas,seaborn,scipy,torch
```

```
Python implementation: CPython
Python version       : 3.11.9
IPython version      : 8.24.0

keras                : 3.3.3
matplotlib           : 3.9.0
numpy                : 1.26.4
pandas               : 2.2.2
seaborn              : 0.13.2
scipy                : 1.11.0
torch                : 2.3.1
tensorflow           : 2.16.1
tensorflow_probability: 0.24.0
tf_keras             : 2.16.0
```

UNSW SYDNEY

Glossary

- aleatoric and epistemic uncertainty
- deep ensembles
- CANN
- GLM
- MDN
- mixture distribution
- posterior sampling
- proper scoring rule