Distributional Regression

ACTL3143 & ACTL5111 Deep Learning for Actuaries

Patrick Laub

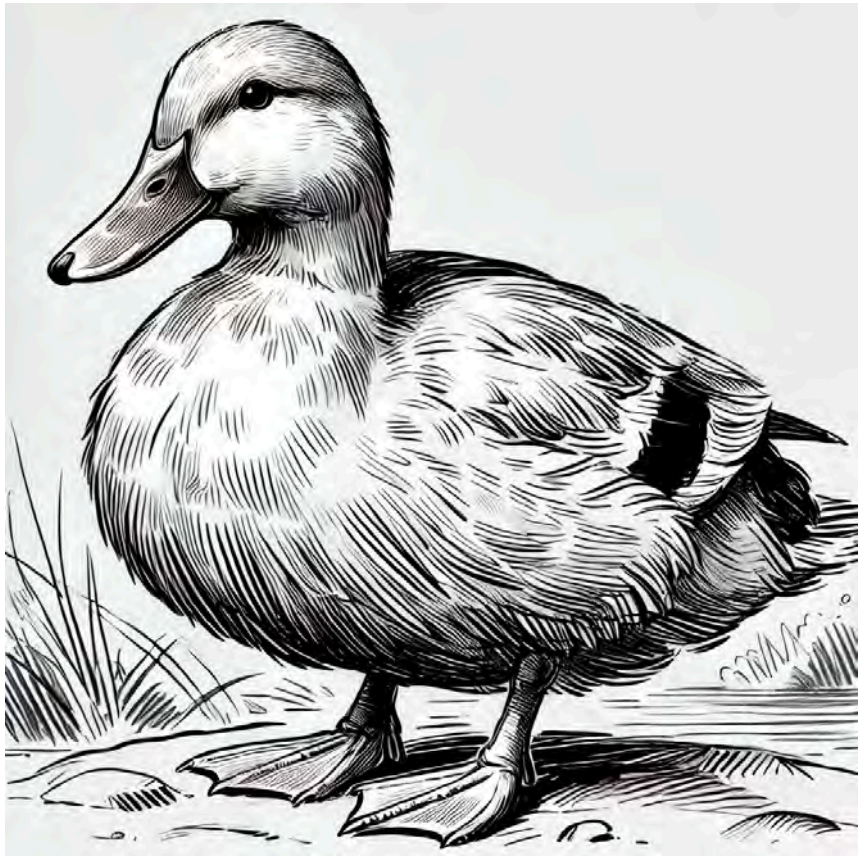
Lecture Outline

- **Introduction**
- Traditional Regression
- Stochastic Forecasts
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- Mixture Density Network
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty
- Avoiding Overfitting
- Dropout
- Ensembles

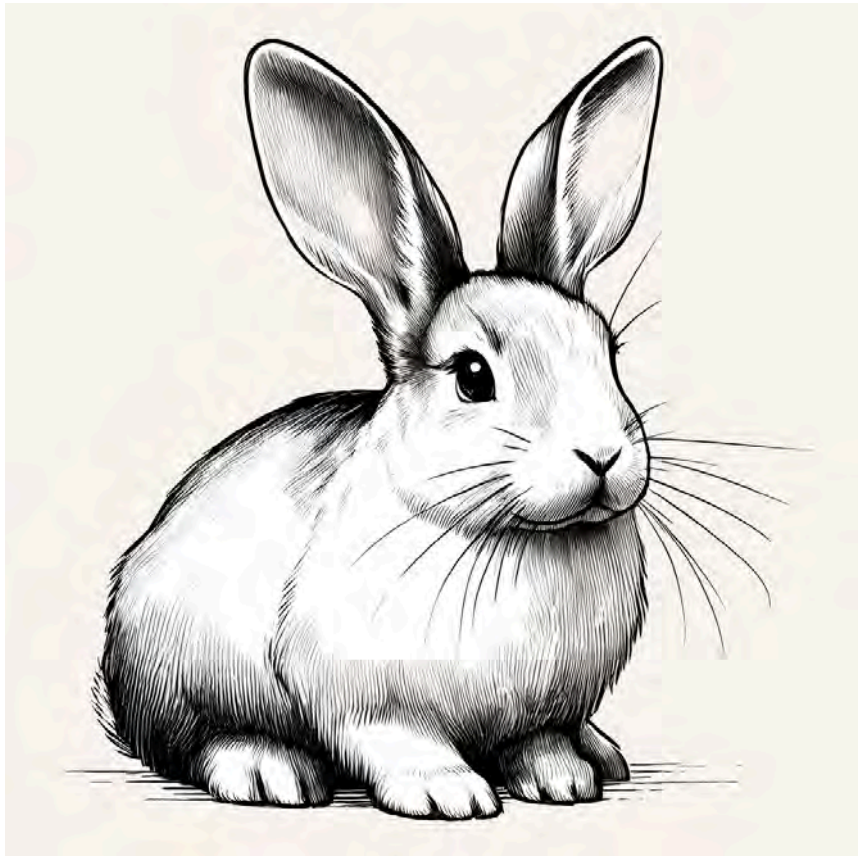
UNSW SYDNEY

Neural networks and confidence

Say we have a neural network that classifies ducks from rabbits.



A duck in the training set



A rabbit in this training set

New data can be different

Source: Olga Telnova, Cute Duck with Bunny Ears, Posterlounge, accessed on July 16 2024.

New data can be challenging

Source: Wikimedia Commons

Classifiers give us a probability

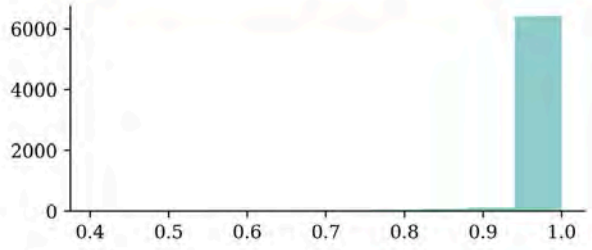
This is already a big step up compared to regression models.

However, neural networks' "probabilities" can be overconfident.

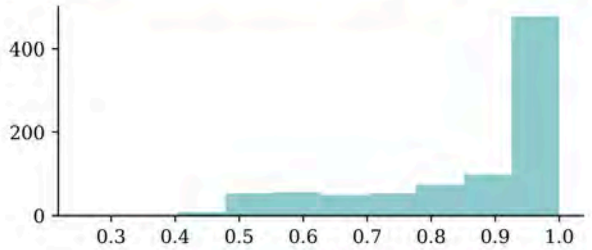
Confidence of predictions

```
y_pred = keras.ops.convert_to_numpy(keras.activations.softmax(model(X_test)))
y_pred_class = np.argmax(y_pred, axis=1)
y_pred_prob = y_pred[np.arange(y_pred.shape[0]), y_pred_class]

confidence_when_correct = y_pred_prob[y_pred_class == y_test]
confidence_when_wrong = y_pred_prob[y_pred_class ≠ y_test]
```

```
plt.hist(confidence_when_correct);
```



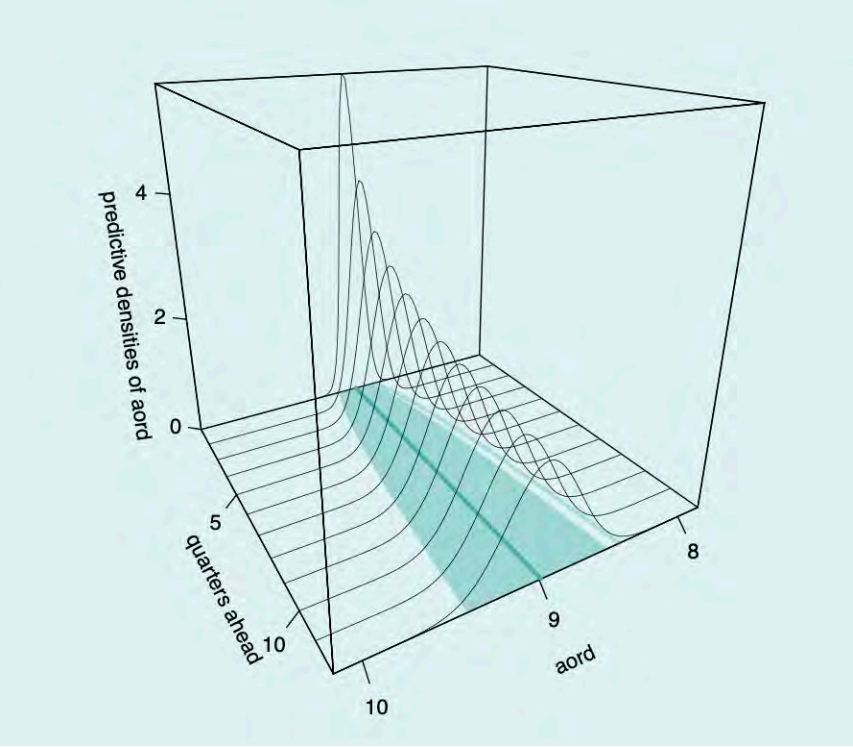

```
plt.hist(confidence_when_wrong);
```



We already saw a case of this.

See Guo et al. (2017), On Calibration of Modern Neural Networks.

Key idea



An example of distributional forecasting over the All Ordinaries Index

- Earlier machine learning models focused on point estimates.
- However, in many applications, we need to understand the distribution of the response variable.
- Each prediction becomes a *distribution* over the possible outcomes

Source: Tomasz Woźniak (2024), LinkedIn Post, accessed on July 15 2024.

Lecture Outline

- Introduction
- **Traditional Regression**
- Stochastic Forecasts
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- Mixture Density Network
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty
- Avoiding Overfitting
- Dropout
- Ensembles

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.).

Traditional Regression

Multiple linear regression assumes the data-generating process is

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

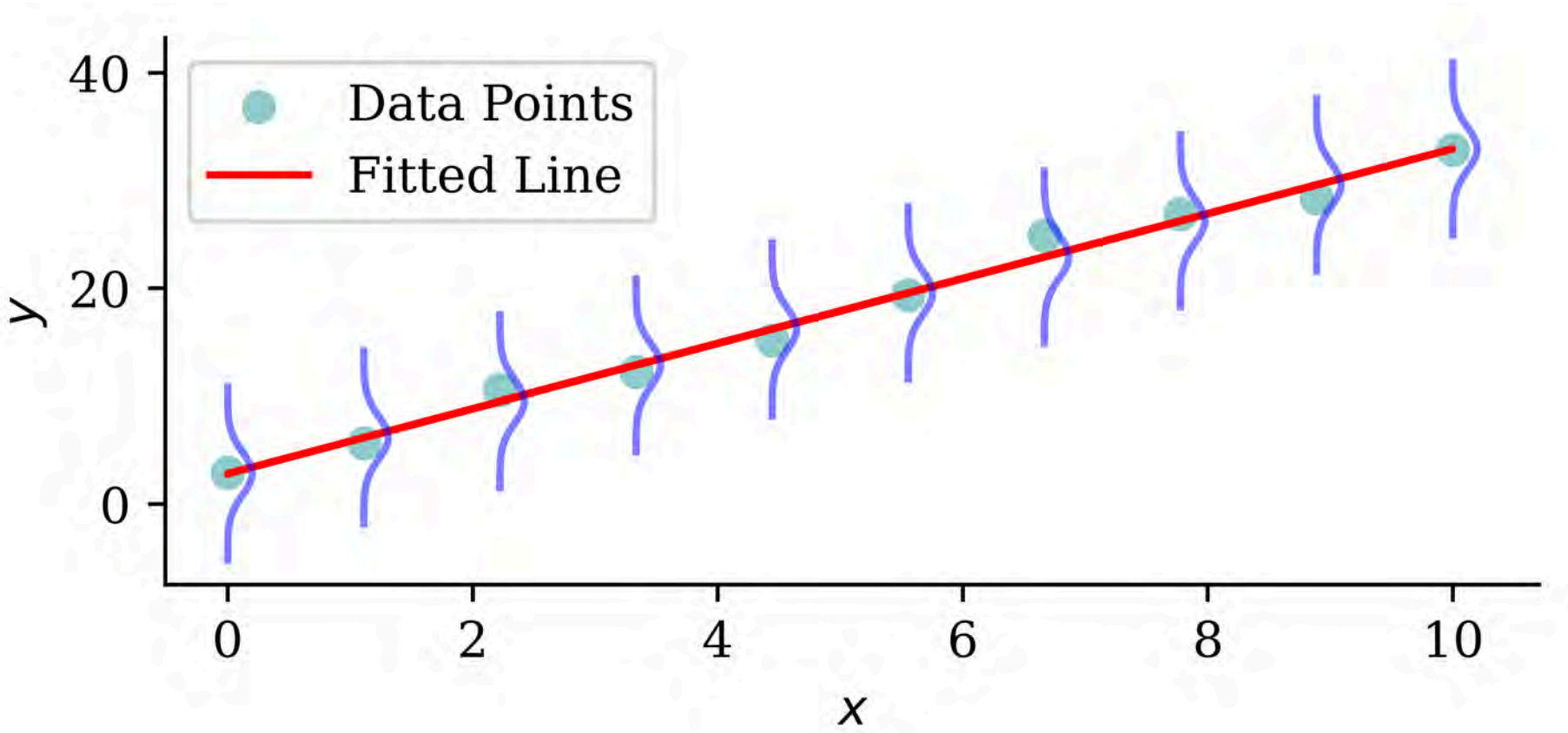
where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.

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.

Visualising the distribution of each Y



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.

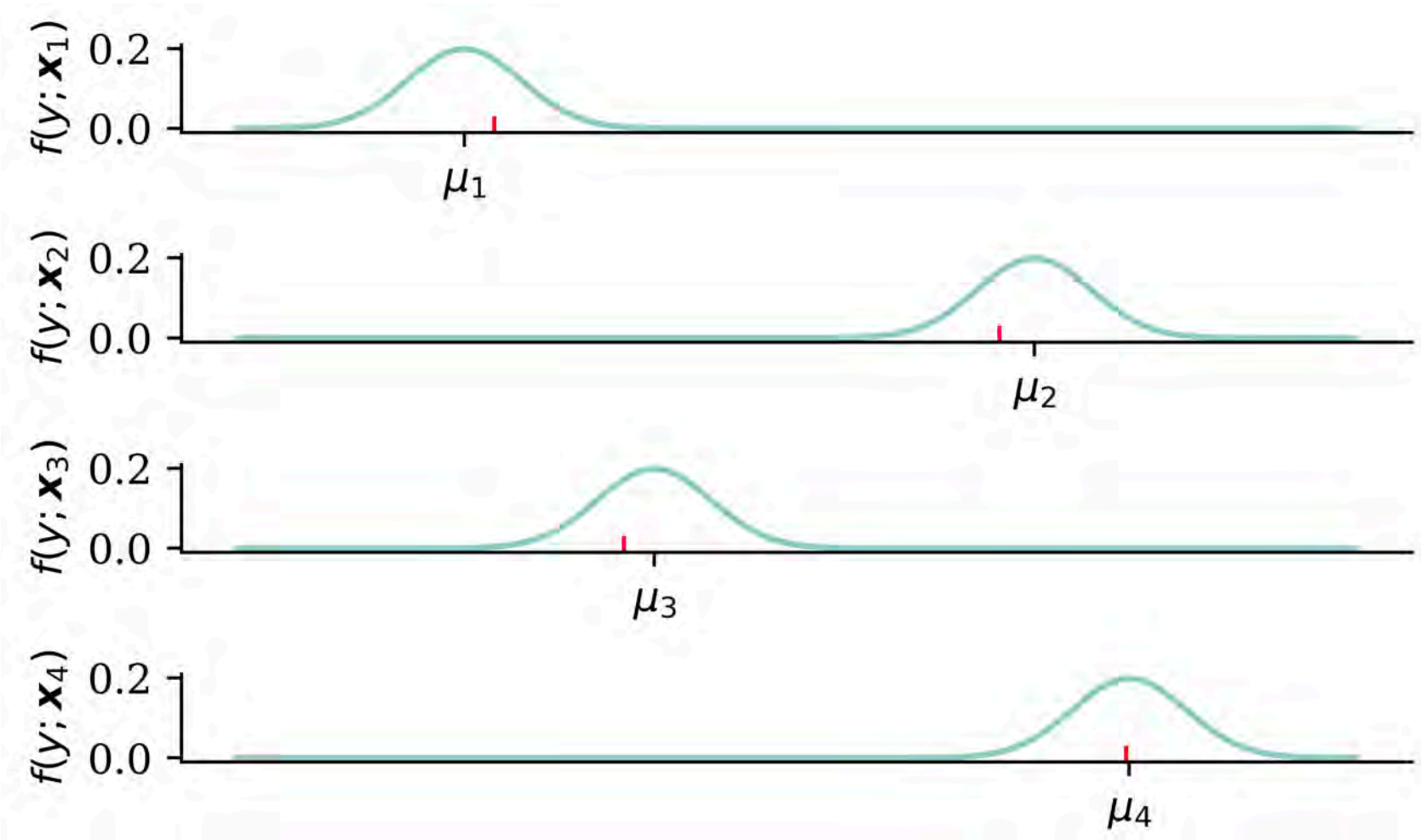
$$f(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right).$$

The likelihood function is

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

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

The predicted distributions



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

UNSW SYDNEY

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

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

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 basically identical:

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

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

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

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?

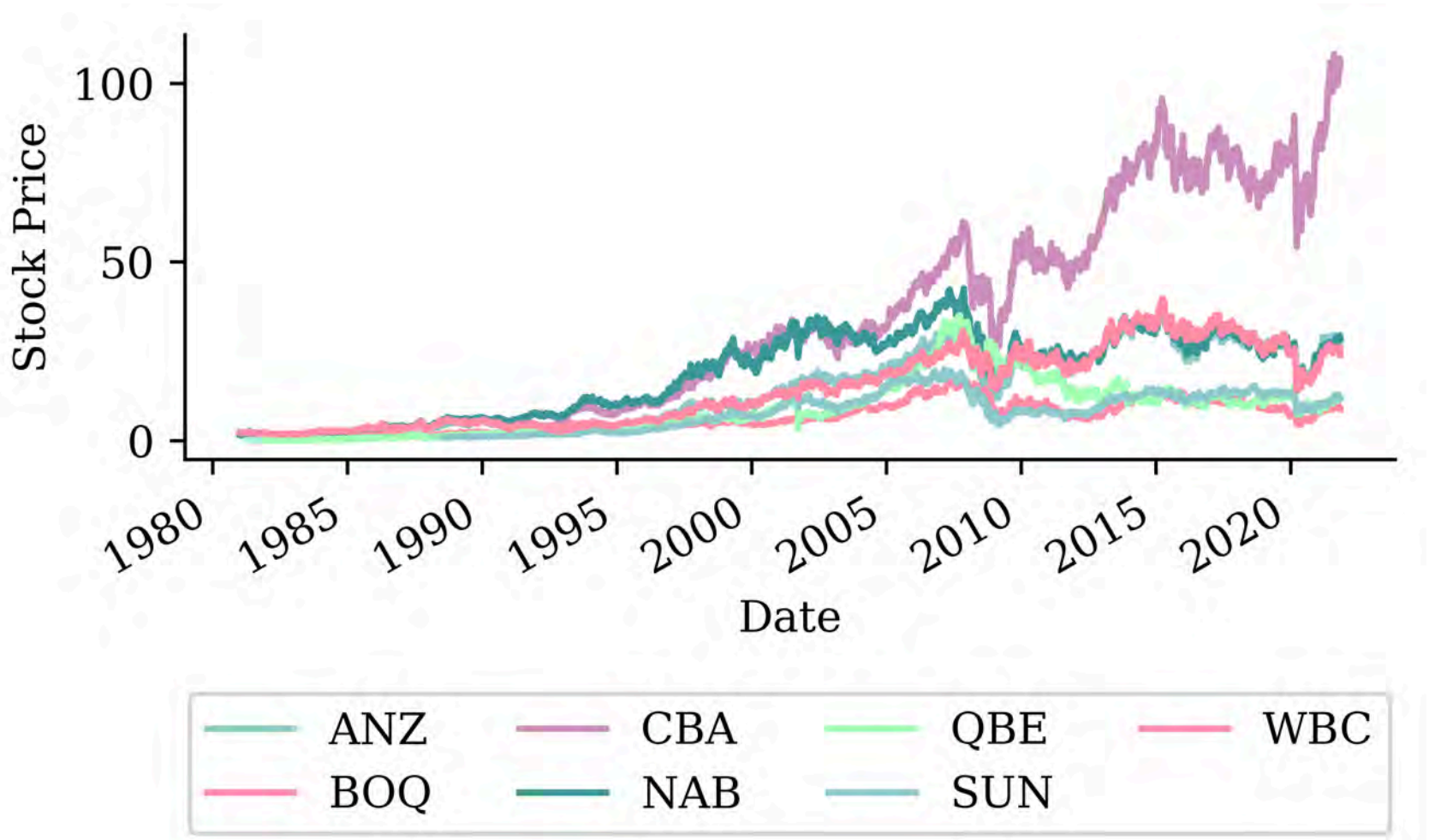
- GLMs are interpretable.
- GLMs are flexible (can handle different types of response variables).
- We get the full distribution of the response variable, not just the mean.

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

Lecture Outline

- Introduction
- Traditional Regression
- **Stochastic Forecasts**
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- Mixture Density Network
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty
- Avoiding Overfitting
- Dropout
- Ensembles

Stock price forecasting



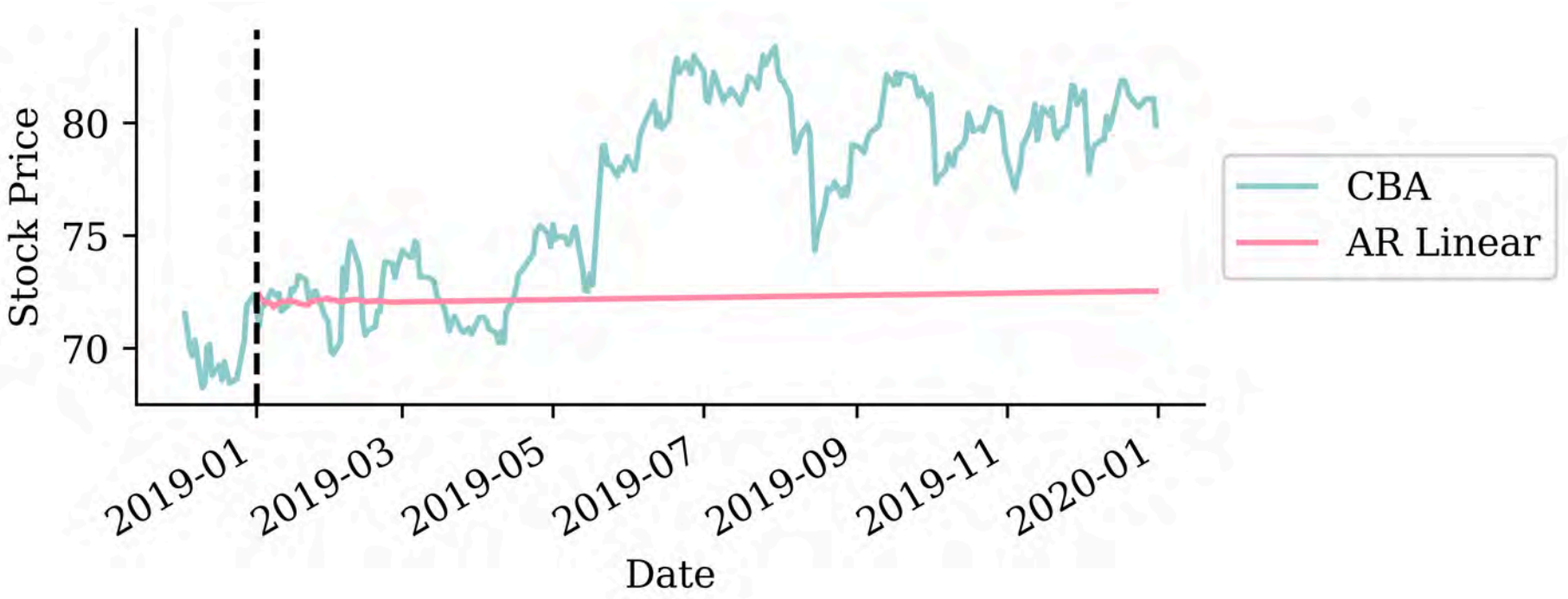
Noisy auto-regressive forecast

```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

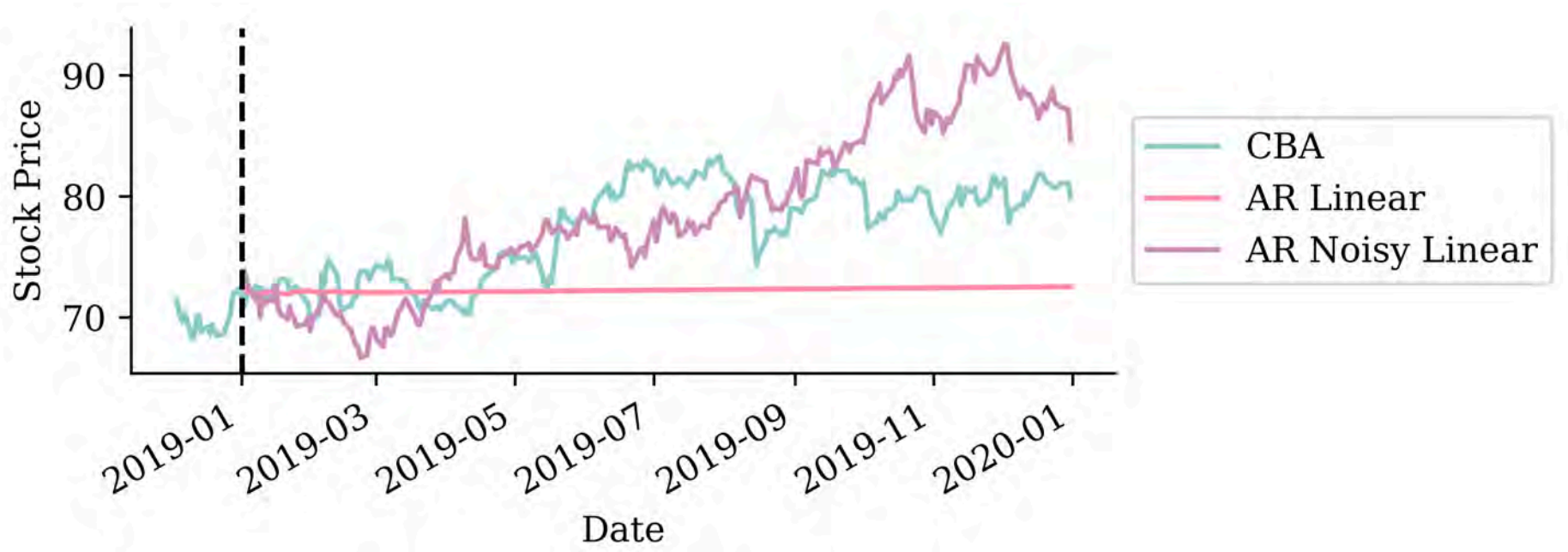
```
1 lr_forecast = noisy_autoregressive_forecast(lr, X_val, 0)
```




```
1 residuals = y_train.loc["2015":] - lr.predict(X_train.loc["2015":])
2 sigma = np.std(residuals)
```

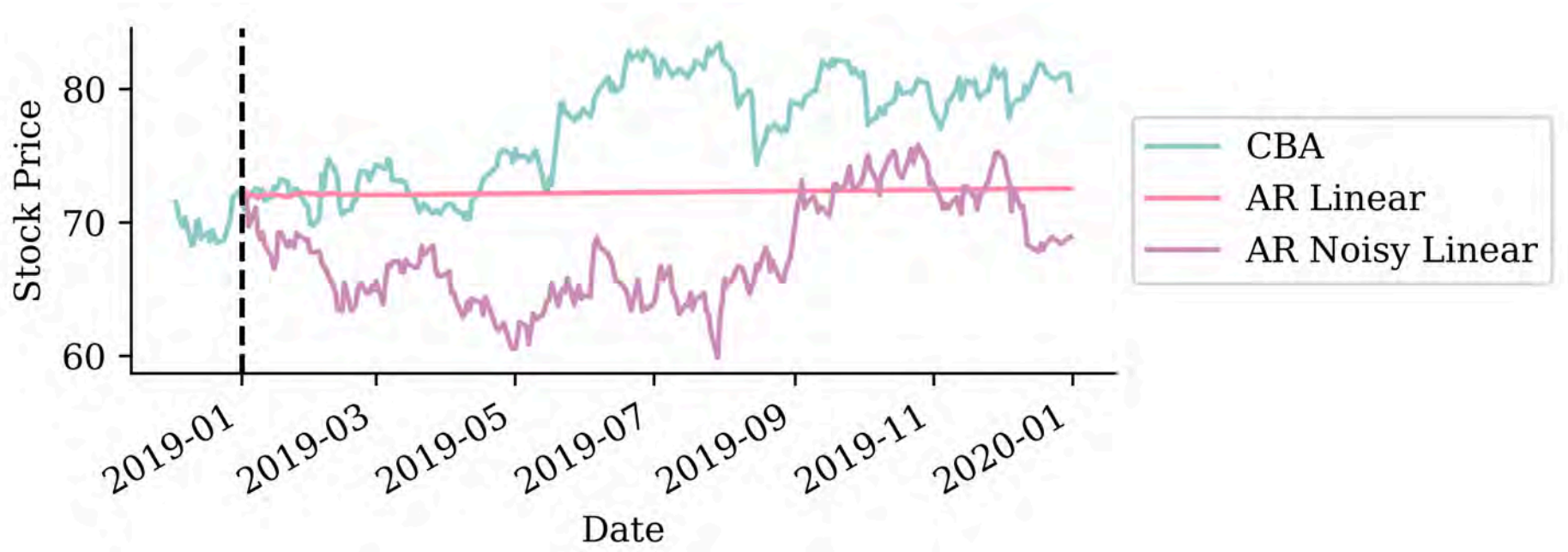
With noise

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



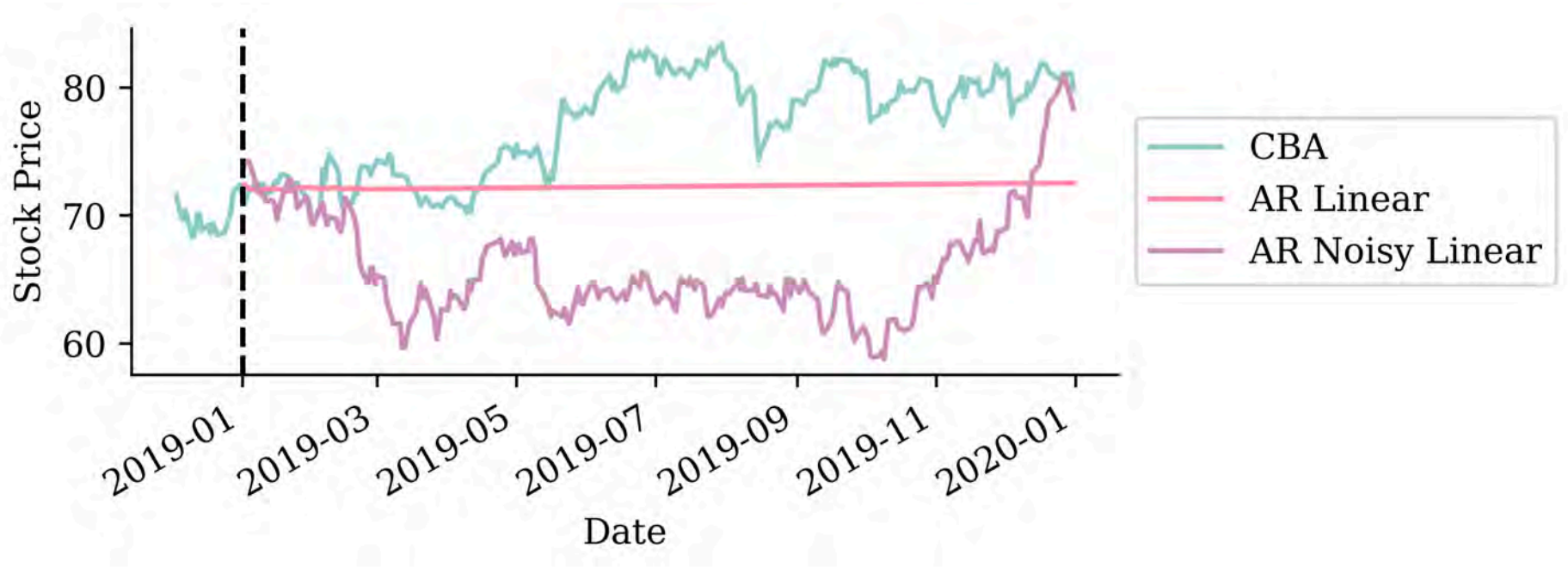
With noise

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



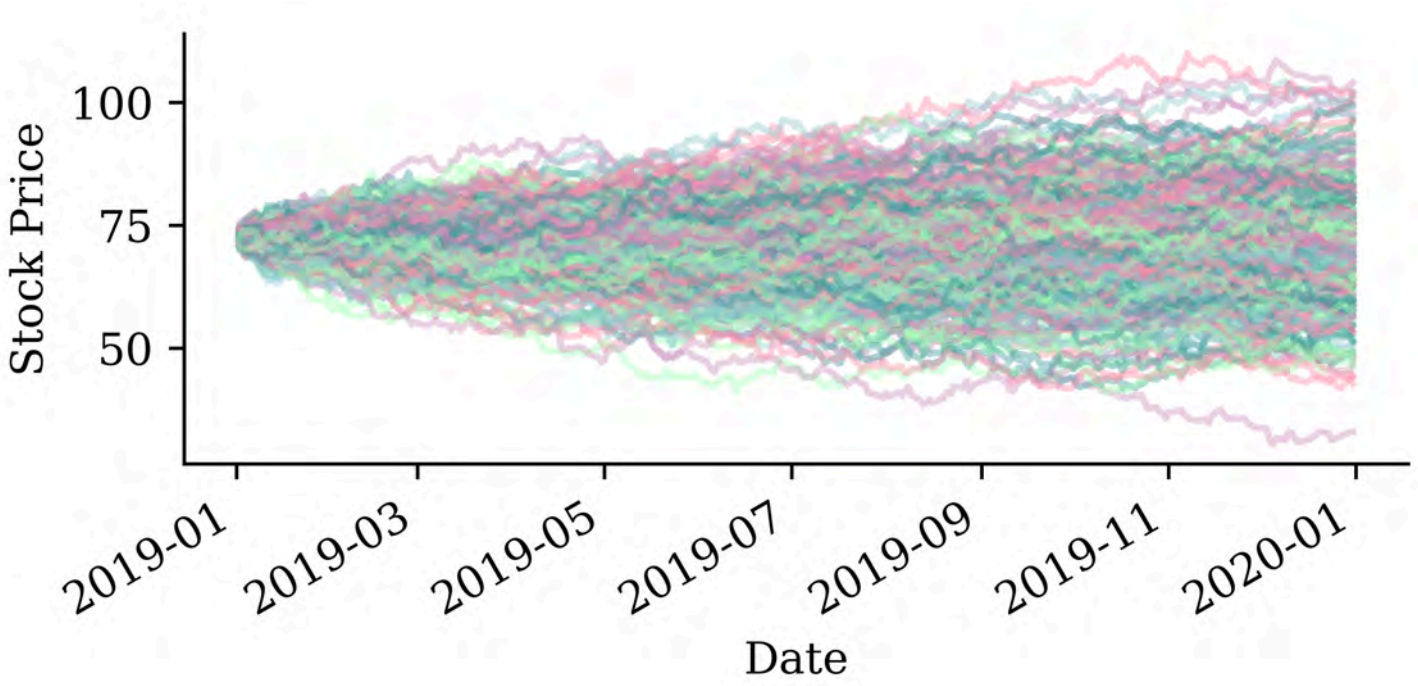
With noise

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



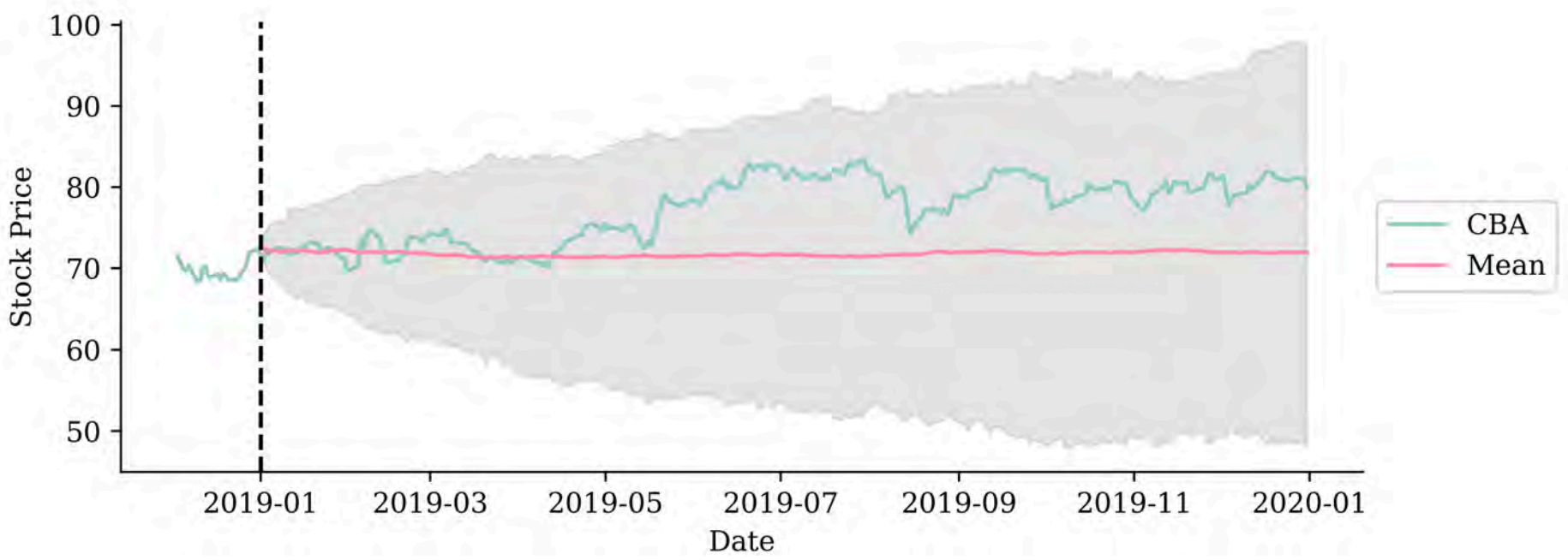
Many noisy forecasts

```
num_forecasts = 300
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)
```

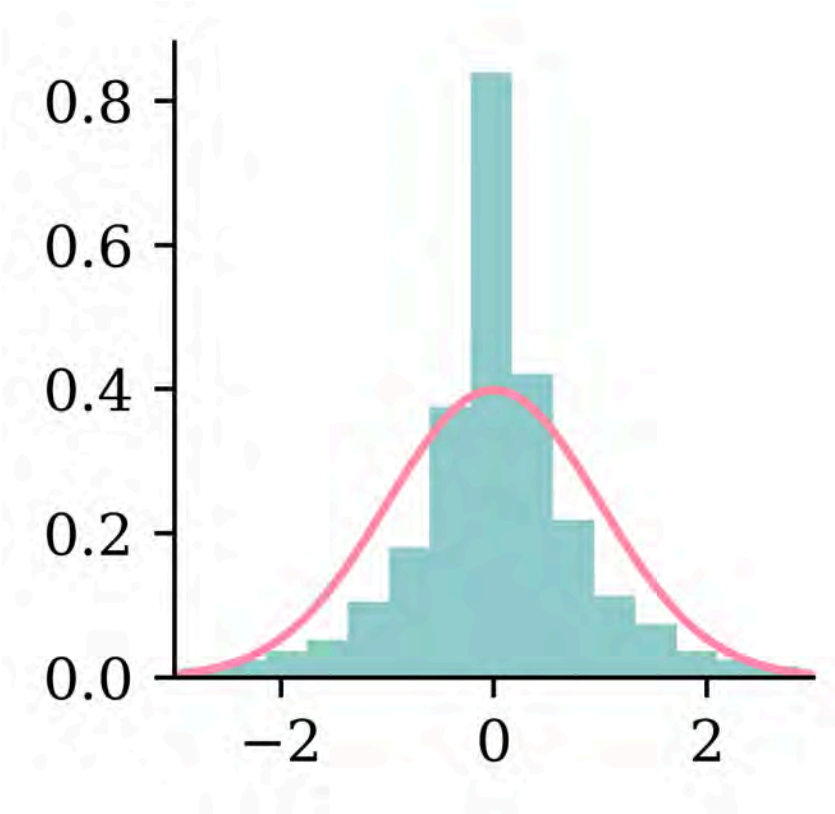


Residuals

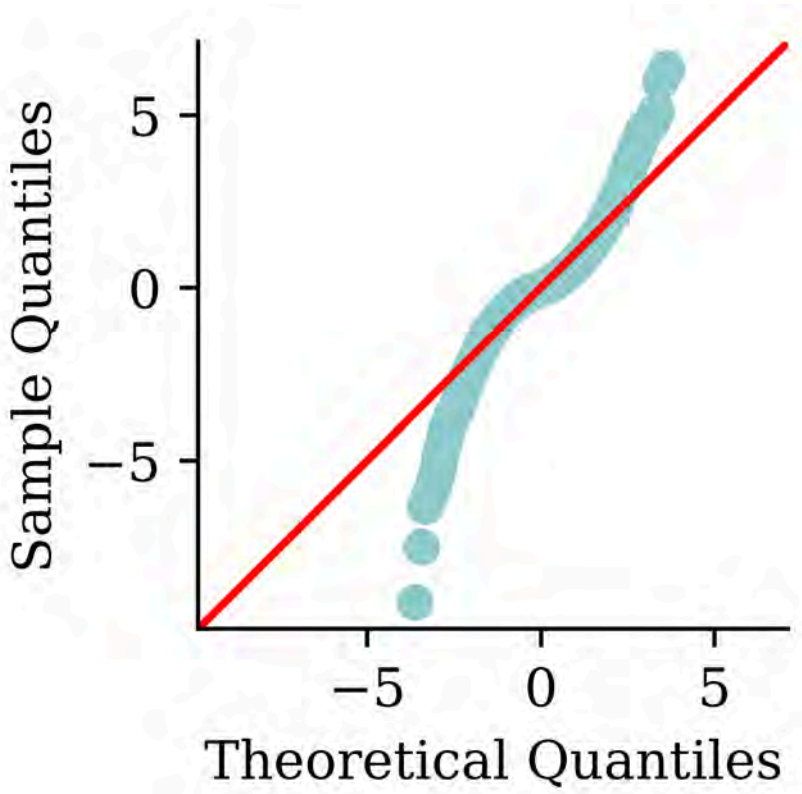
```
1  y_pred = lr.predict(X_train)
2  residuals = y_train - y_pred
3  residuals -= np.mean(residuals)
4  residuals /= np.std(residuals)
5  stats.shapiro(residuals)
```

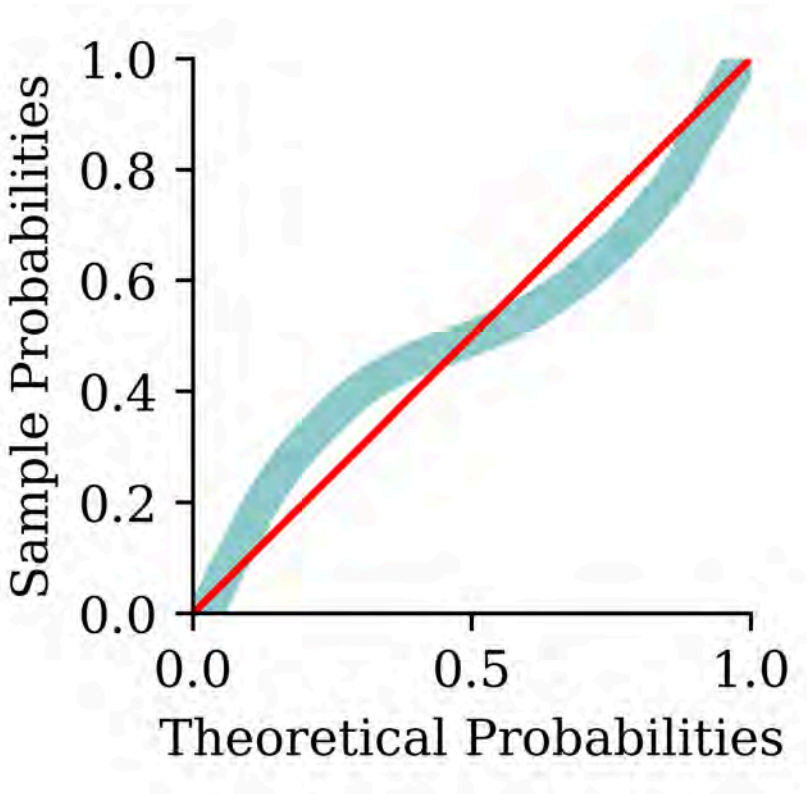
```
/home/plaub/miniconda3/envs/ai2024/lib/python3
packages/scipy/stats/_morestats.py:1882:
UserWarning: p-value may not be accurate for
N > 5000.
  warnings.warn("p-value may not be accurate
for N > 5000.")
```

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

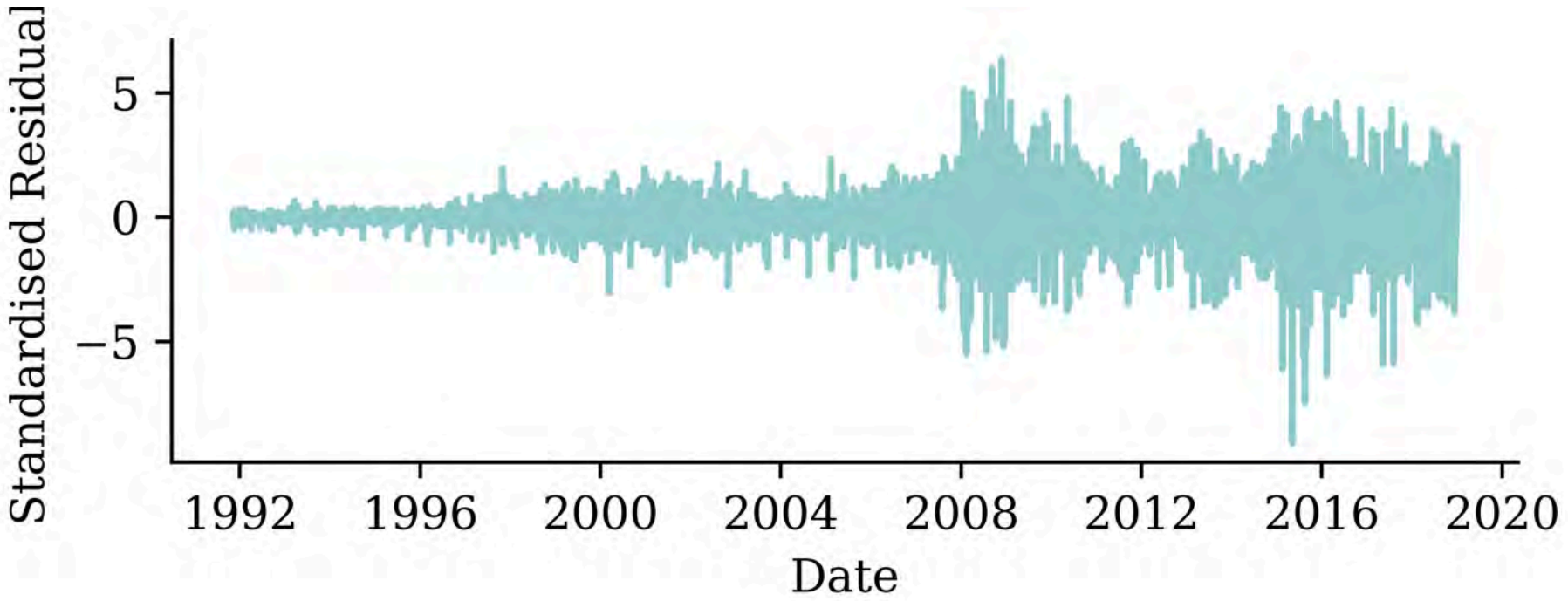


Q-Q plot and P-P plot





Residuals against time



Heteroskedasticity!

Lecture Outline

- Introduction
- Traditional Regression
- Stochastic Forecasts
- **GLMs and Neural Networks**
- Combined Actuarial Neural Network
- Mixture Density Network
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty
- Avoiding Overfitting
- Dropout
- Ensembles

French motor claim sizes

```
sev = pd.read_csv('freMTPL2sev.csv')
cov = pd.read_csv('freMTPL2freq.csv').drop(columns=['ClaimNb'])
sev = pd.merge(sev, cov, on='IDpol', how='left').drop(columns=["IDpol"]).dropna()
sev
```

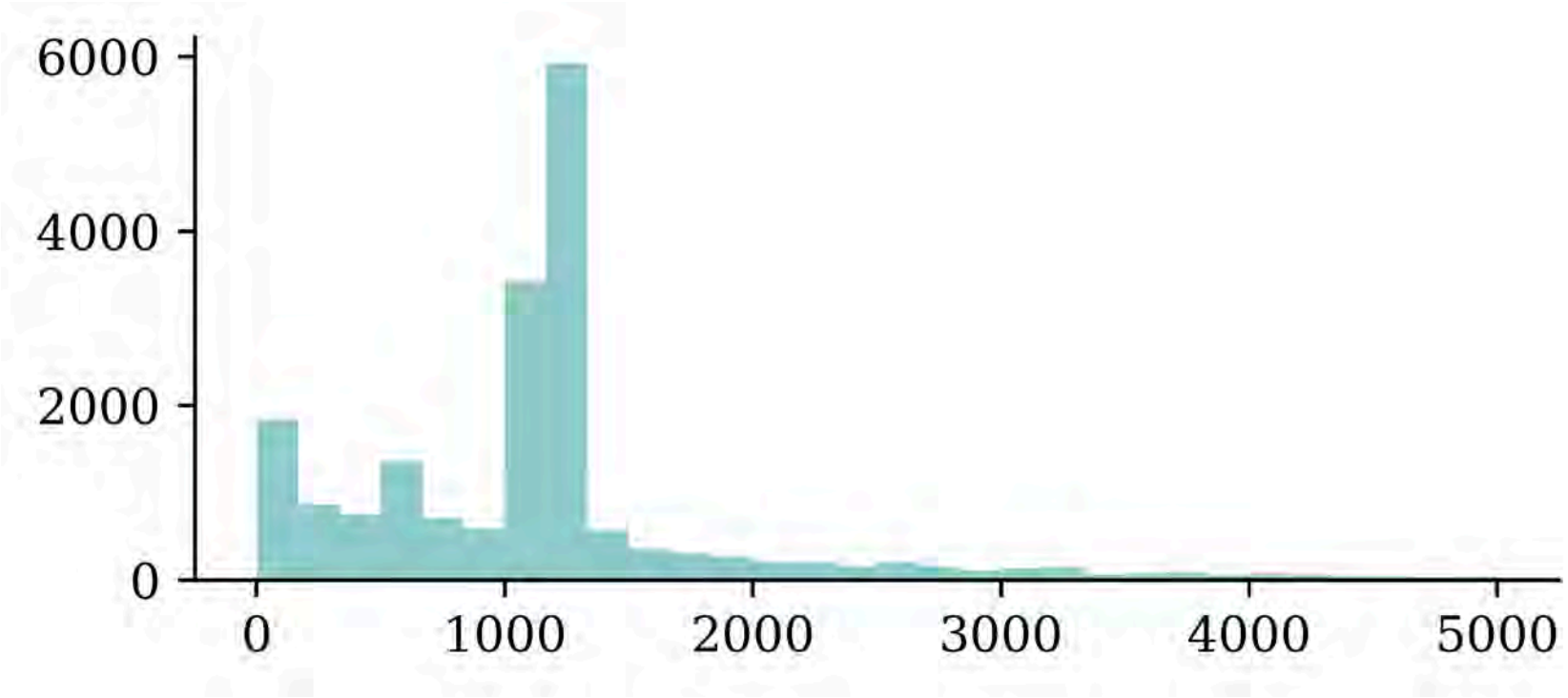
	ClaimAmount	Exposure	VehPower	VehAge	DrivAge
0	995.20	0.59	11.0	0.0	39.0
1	1128.12	0.95	4.0	1.0	49.0
...	...	...	...	...	...
26637	767.55	0.43	6.0	0.0	67.0
26638	1500.00	0.28	7.0	2.0	36.0

26444 rows × 11 columns

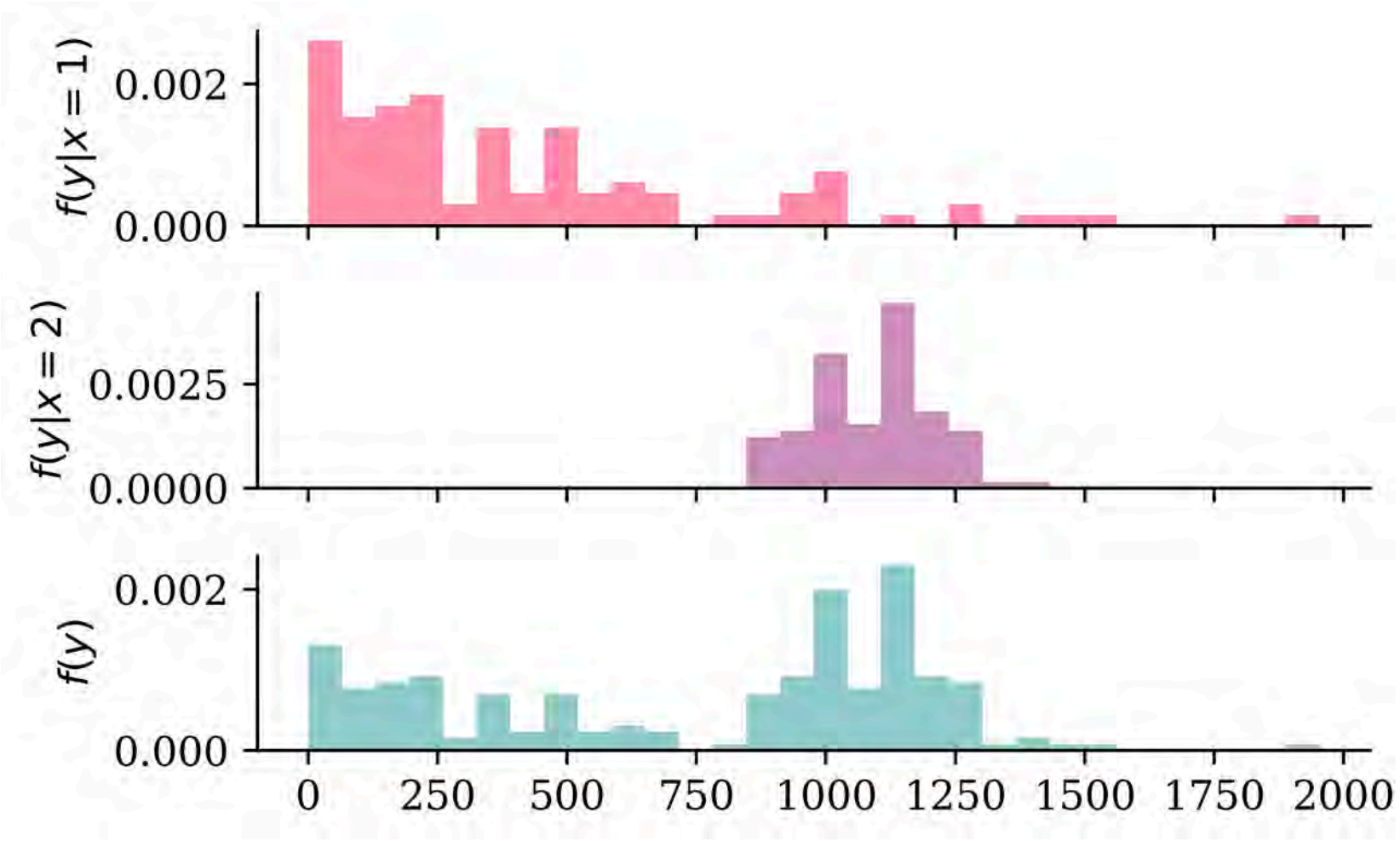
UNSW SYDNEY

Preprocessing

```
X_train, X_test, y_train, y_test = train_test_split(
  sev.drop("ClaimAmount", axis=1), sev["ClaimAmount"], random_state=2023)
ct = make_column_transformer((OrdinalEncoder(), ["Area", "VehGas"]),
    ("drop", ["VehBrand", "Region"]), remainder=StandardScaler())
X_train = ct.fit_transform(X_train)
X_test = ct.transform(X_test)
plt.hist(y_train[y_train < 5000], bins=30);
```



Doesn't prove that $Y|\boldsymbol{X} = \boldsymbol{x}$ is multimodal



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:

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

(Here, p is the number of coefficients in the model. If this p doesn't include the intercept, then p should be use $\frac{1}{n-(p+1)}$.)

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
ordinalencoder__Area       -0.073226
                              ...
remainder__BonusMalus       0.157204
remainder__Density          0.010539
Length: 9, dtype: float64
```

```python
# Dispersion Parameter
mus = gamma_glm.predict(X_train_design)
residuals = y_train - mus
dof = (len(y_train)-X_train_design.shap
phi_glm = np.sum(residuals**2/mus**2)/d
print(phi_glm)
```

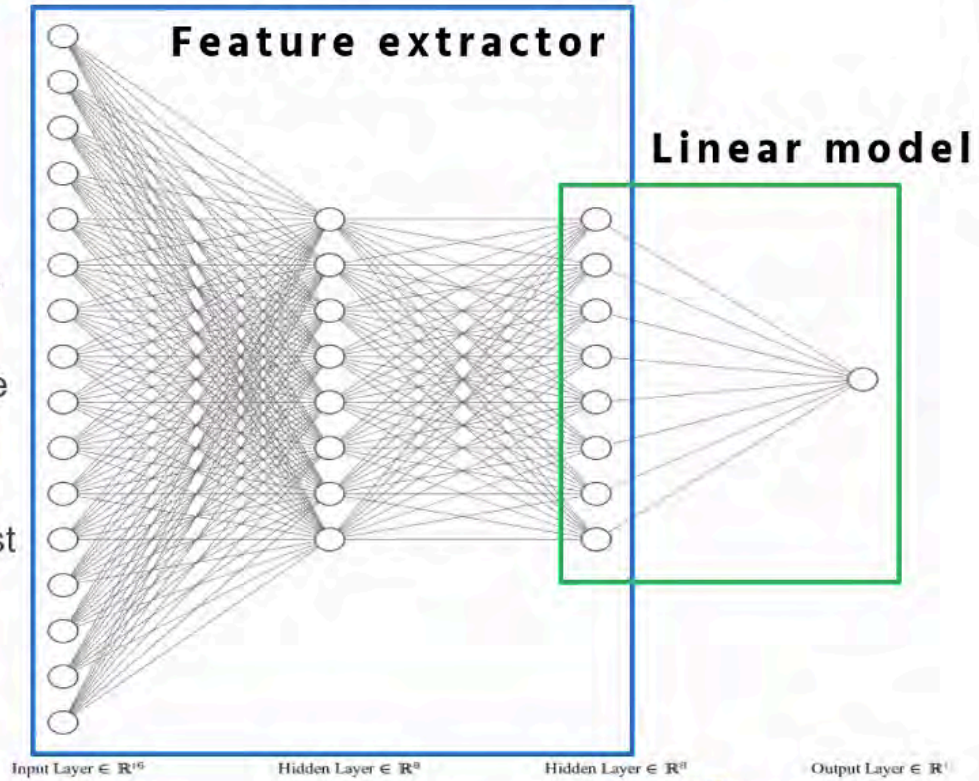
```
59.63363123735805
```

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

- Introduction
- Traditional Regression
- Stochastic Forecasts
- GLMs and Neural Networks
- **Combined Actuarial Neural Network**
- Mixture Density Network
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty
- Avoiding Overfitting
- Dropout
- Ensembles

UNSW SYDNEY

CANN

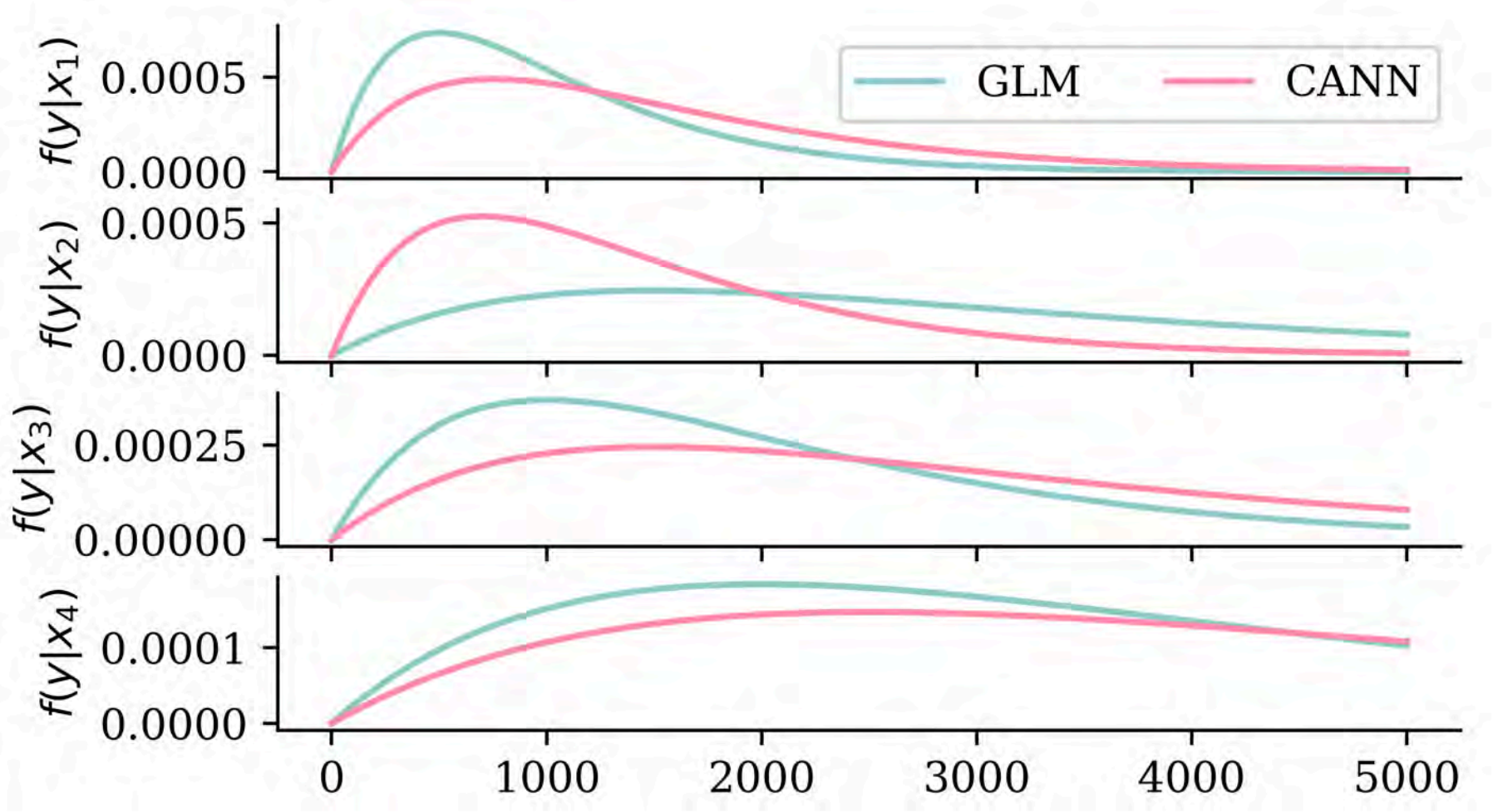
The Combined Actuarial Neural Network is a novel actuarial neural network architecture proposed by Schelldorfer and Wüthrich (2019). We summarise the CANN approach as follows:

- 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

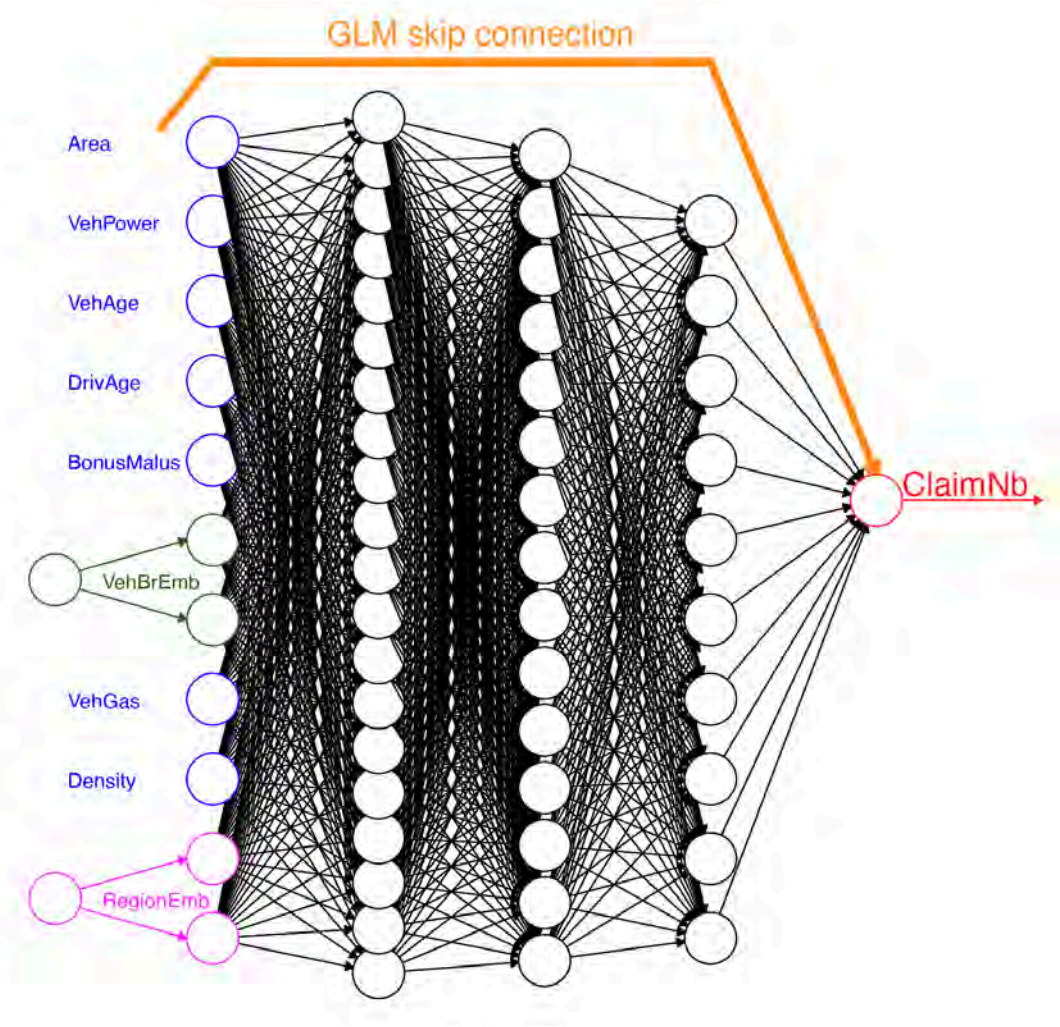
$$\mathbb{E}[Y|\boldsymbol{X} = \boldsymbol{x}] = g^{-1}\Big(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle + \mathcal{M}_{\text{CANN}}(\boldsymbol{x}; \boldsymbol{w}_{\text{CANN}})\Big).$$

UNSW SYDNEY

Shifting the predicted distributions



Architecture



The CANN architecture.

Source: Schelldorfer and Wüthrich (2019), Nesting Classical Actuarial Models into Neural Networks, SSRN, Figure 8.

Code: Architecture

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

# GLM part (won't be updated during training)
glm_weights = gamma_glm.params.iloc[1:].values.reshape((-1, 1))
glm_bias = gamma_glm.params.iloc[0]
glm_part = Dense(1, activation='linear', trainable=False,
                    kernel_initializer=Constant(glm_weights),
                    bias_initializer=Constant(glm_bias))(inputs)

# Neural network part
x = Dense(64, activation='leaky_relu')(inputs)
nn_part = Dense(1, activation='linear')(x)

# Combine GLM and CANN estimates
mu = keras.ops.exp(glm_part + nn_part)
cann = Model(inputs, mu)
```

Since this CANN predicts gamma distributions, we use the gamma NLL loss function.

```
def cann_negative_log_likelihood(y_true, y_pred):
    return keras.ops.mean(keras.ops.log(y_pred) + y_true/y_pred)
```

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

```
31.171623242378978
```

UNSW SYDNEY

Lecture Outline

- Introduction
- Traditional Regression
- Stochastic Forecasts
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- **Mixture Density Network**
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty
- Avoiding Overfitting
- Dropout
- Ensembles

Mixture Distribution

Given a finite set of resulting random variables $(Y_1, \ldots, Y_K)$, one can generate a multinomial random variable $Y \sim \text{Multinomial}(1, \boldsymbol{\pi})$. Meanwhile, Y can be regarded as a mixture of $Y_1, \ldots, Y_K$, 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 $\boldsymbol{\pi} = (\pi_1 \ldots, \pi_K)$ such that $\pi_k \geq 0$ for $k \in \{1, \ldots, K\}$ 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 $k \in \{1, \ldots, K\}$.

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.,

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

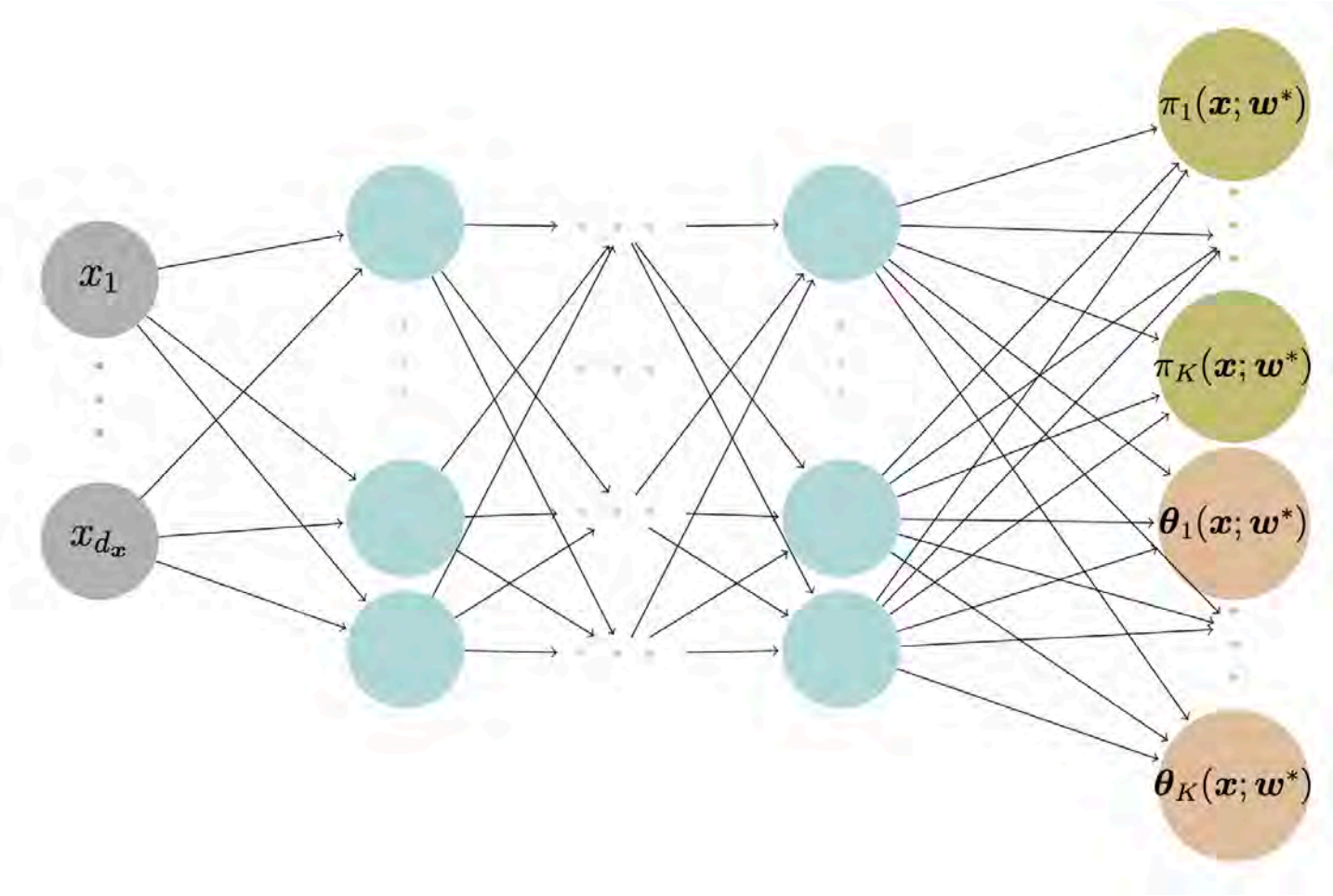
where $\boldsymbol{w}^*$ is the networks' weights found by minimising the following negative log-likelihood loss function

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

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. $\boldsymbol{\theta}_k(\boldsymbol{x}; \boldsymbol{w}^*) = (\theta_{k,1}(\boldsymbol{x}; \boldsymbol{w}^*), \ldots, \theta_{k,|\boldsymbol{\theta}_k|}(\boldsymbol{x}; \boldsymbol{w}^*))$ 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

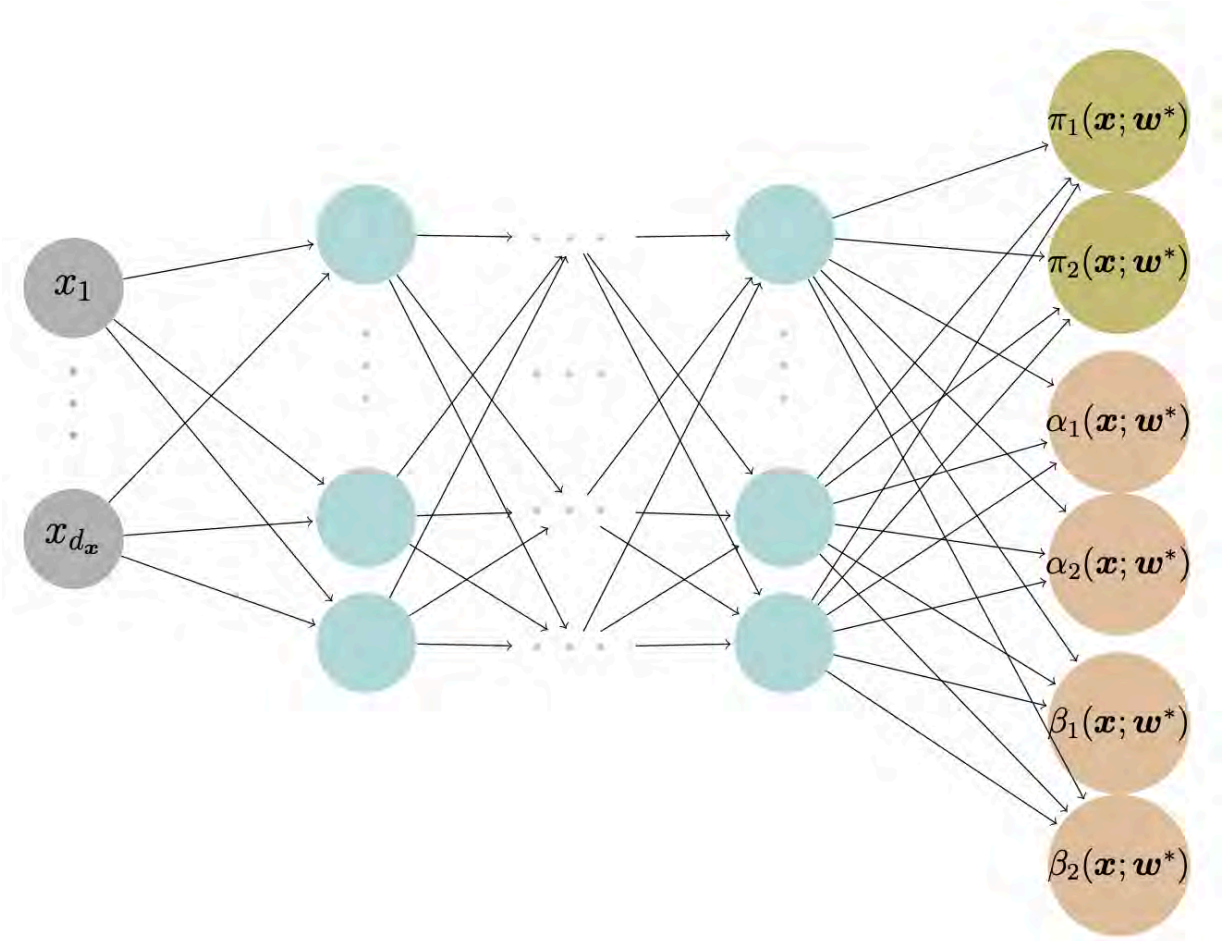
The aim is to find the optimum weights

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

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: Import “legacy” Keras (for now)

```
1 import tf_keras
```

Keras 3 breaks Tensorflow Probability upon import #1774

Closed **AjaniStewart** opened this issue on Dec 5, 2023 · 4 comments

jburnim commented on Dec 6, 2023 Member

TensorFlow Probability 0.23.0 is not compatible with Keras 3.

And it looks like TensorFlow 2.15.0 is also not compatible with Keras 3. When I run those `pip` commands, upgrading `Keras` gives the error message:

```
tensorflow 2.15.0 requires keras<2.16,>=2.15.0, but you have keras 3.0.0 which is incompati
```

It will be possible to import TensorFlow Probability 0.24.0 with TensorFlow 2.16.0 and Keras 3 installed. But please note that TensorFlow Probability 0.24.0 will continue to use Keras 2, which means that tf-keras 2.16.0 will also have to be installed.

1

Source: Tensorflow Probability GitHub, Keras 3 breaks Tensorflow Probability upon import, issue #1774.

Code: Architecture

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

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

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

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

pis = tf_keras.layers.Dense(2, activation='softmax')(x) # Mixing weights
alphas = tf_keras.layers.Dense(2, activation='exponential')(x) # Shape parameters
betas = tf_keras.layers.Dense(2, activation='exponential')(x) # Scale parameters
out = tf_keras.layers.Concatenate(axis=1)([pis, alphas, betas]) # shape = (None, 6)

gamma_mdn = tf_keras.Model(inputs, out)
```

UNSW SYDNEY

Loss Function

The negative log-likelihood loss function is given by

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

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 & training

`tensorflow_probability` to the rescue.

```python
import tensorflow_probability as tfp
tfd = tfp.distributions

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]
    mixture_distribution = tfd.MixtureSameFamily(
        mixture_distribution=tfd.Categorical(probs=pis),
        components_distribution=tfd.Gamma(alphas, betas))
    return -tf_keras.backend.mean(mixture_distribution.log_prob(y_true))
```

```python
gamma_mdn.compile(optimizer="adam", loss=gamma_mixture_nll)

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

UNSW SYDNEY

Lecture Outline

- Introduction
- Traditional Regression
- Stochastic Forecasts
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- Mixture Density Network
- **Metrics for Distributional Regression**
- Aleatoric and Epistemic Uncertainty
- Avoiding Overfitting
- Dropout
- Ensembles

Proper Scoring Rules

Definition

A *scoring rule* is the equivalent of a loss function for distributional regression.

Denote $S(F, y)$ to be the score given to the forecasted distribution F and an observation $y \in \mathbb{R}$.

Definition

A scoring rule is called *proper* if

$$\mathbb{E}_{Y \sim Q} S(Q, Y) \leq \mathbb{E}_{Y \sim Q} S(F, Y)$$

for all F and Q distributions.

It is called *strictly proper* if equality holds only if $F = Q$.

UNSW SYDNEY

Example 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)

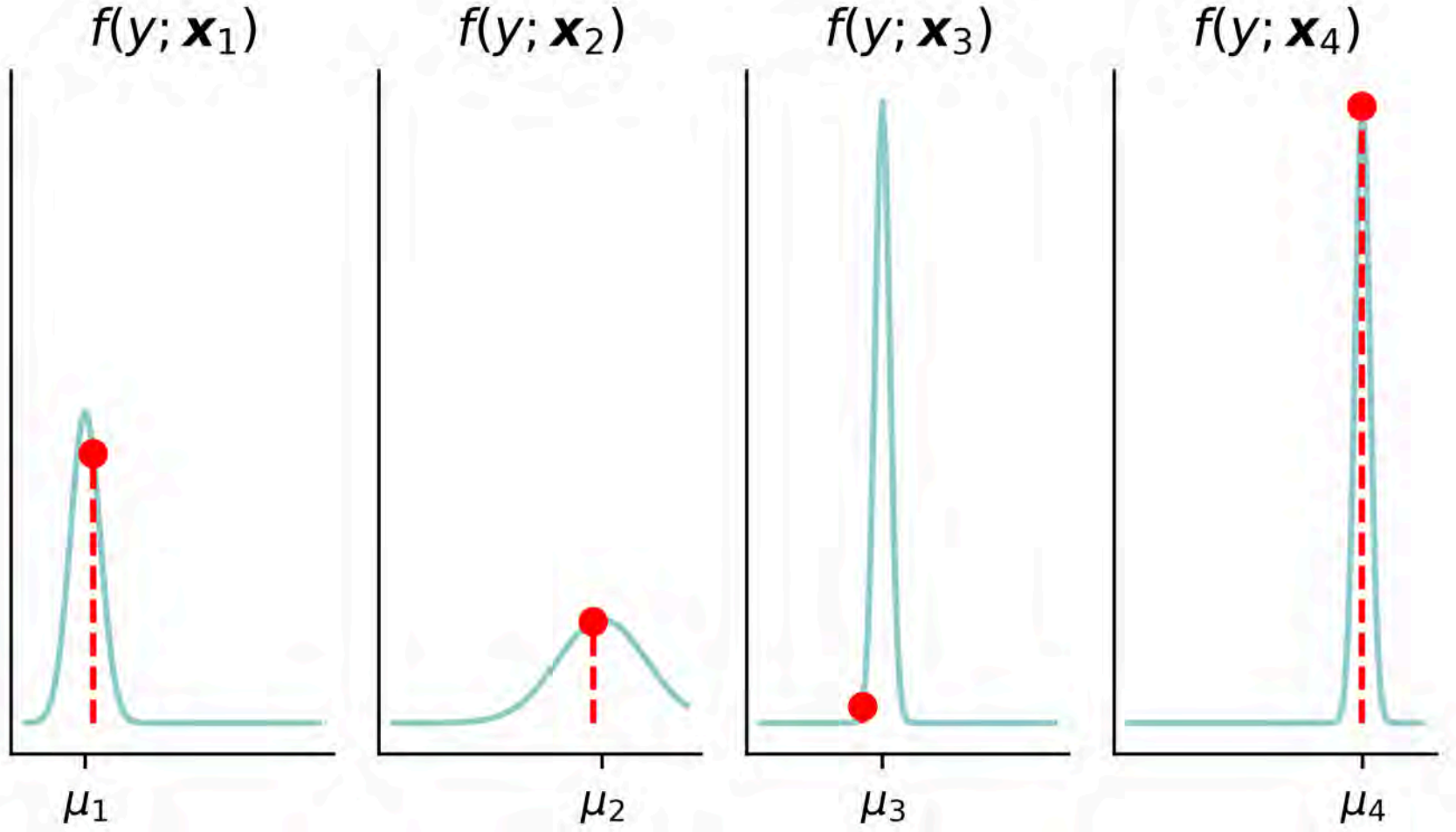
The continuous ranked probability score is defined as

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

where F is the predicted c.d.f.

See, e.g., Taggert (2023), Estimation of CRPS for precipitation forecasts..., BoM Research Report 079.

Likelihoods



UNSW SYDNEY

Code: NLL

```python
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 = stats.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 = cann.predict(X_test, verbose=0)
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: 10.44
MDN: 8.67
```

Lecture Outline

- Introduction
- Traditional Regression
- Stochastic Forecasts
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- Mixture Density Network
- Metrics for Distributional Regression
- **Aleatoric and Epistemic Uncertainty**
- Avoiding Overfitting
- Dropout
- Ensembles

UNSW SYDNEY

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 $\boldsymbol{x}$.

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.

Lecture Outline

- Introduction
- Traditional Regression
- Stochastic Forecasts
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- Mixture Density Network
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty
- **Avoiding Overfitting**
- Dropout
- Ensembles

Traditional regularisation

Say all the m weights (excluding biases) are in the vector $\boldsymbol{\theta}$. If we change the loss function to

$$\text{Loss}_{1:n} = \frac{1}{n}\sum_{i=1}^{n}\text{Loss}_i + \lambda\sum_{j=1}^{m}|\theta_j|$$

this would be using L^1 regularisation. A loss like

$$\text{Loss}_{1:n} = \frac{1}{n}\sum_{i=1}^{n}\text{Loss}_i + \lambda\sum_{j=1}^{m}\theta_j^2$$

is called L^2 regularisation.

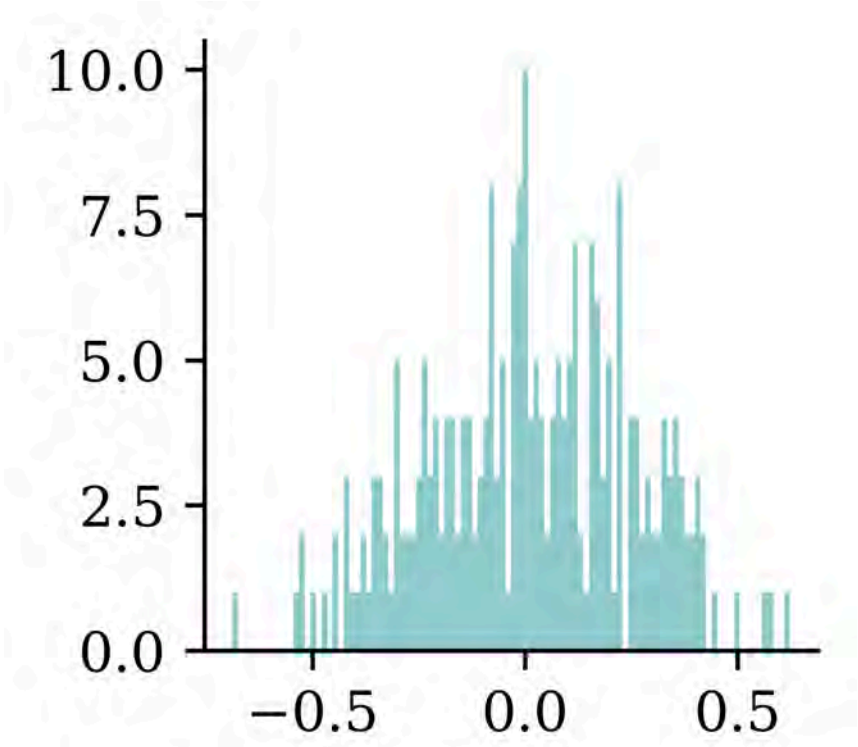
Regularisation in Keras

```
from keras.regularizers import L1, L2

def l1_model(regulariser_strength=0.01):
  random.seed(123)
  model = Sequential([
      Dense(30, activation="leaky_relu",
        kernel_regularizer=L1(regulariser_strength)),
      Dense(1, activation="exponential")
  ])

  model.compile("adam", "mse")
  model.fit(X_train_sc, y_train, epochs=4, verbose=0)
  return model

def l2_model(regulariser_strength=0.01):
  random.seed(123)
  model = Sequential([
      Dense(30, activation="leaky_relu",
        kernel_regularizer=L2(regulariser_strength)),
      Dense(1, activation="exponential")
  ])

  model.compile("adam", "mse")
  model.fit(X_train_sc, y_train, epochs=10, verbose=0)
  return model
```

Weights before & after L^2

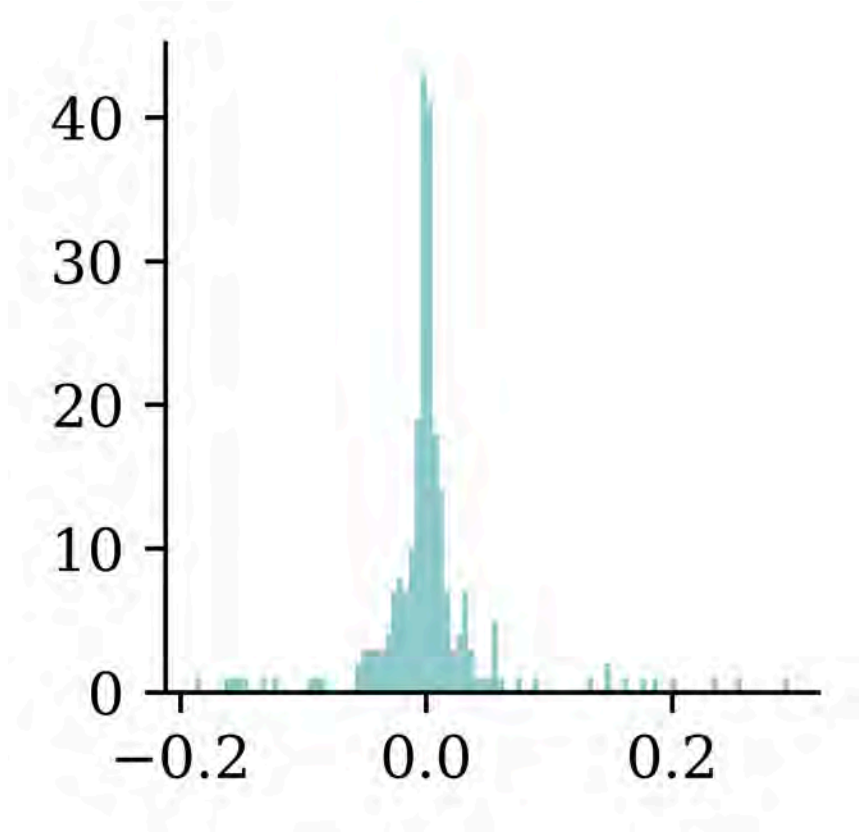
```
model = l2_model(0.0)
weights = model.layers[0].get_weights()
print(f"Number of weights almost 0: {np
plt.hist(weights, bins=100);
```

Number of weights almost 0: 0




```
model = l2_model(1.0)
weights = model.layers[0].get_weights()
print(f"Number of weights almost 0: {np
plt.hist(weights, bins=100);
```

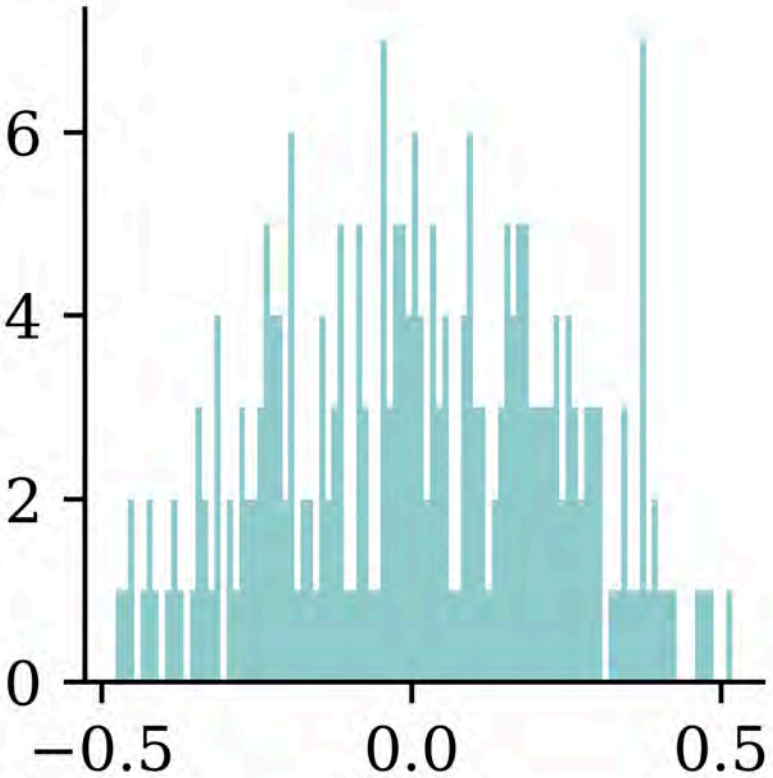
Number of weights almost 0: 0



Weights before & after L^1

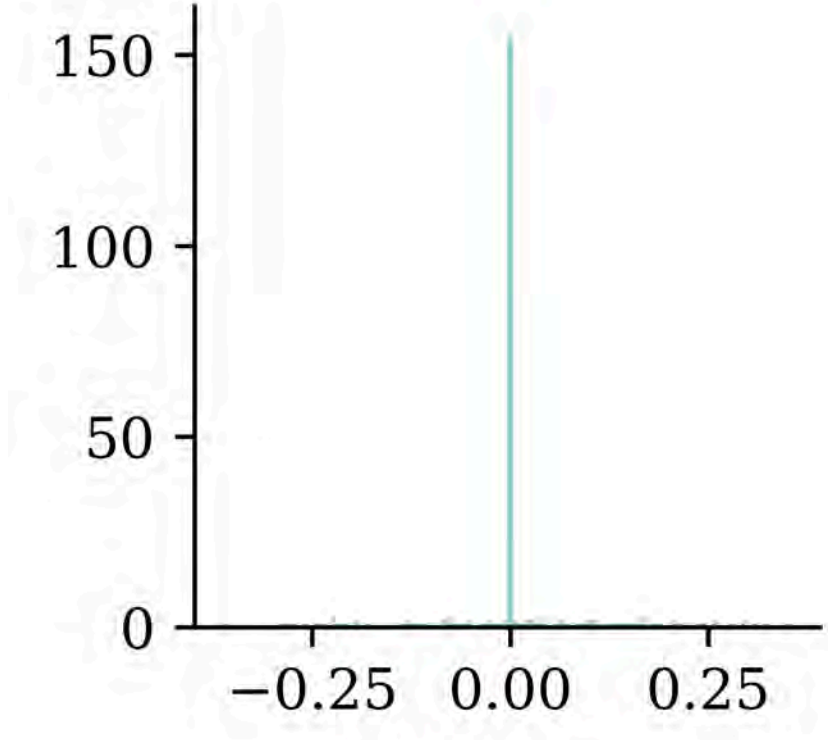
```python
model = l1_model(0.0)
weights = model.layers[0].get_weights()
print(f"Number of weights almost 0: {np
plt.hist(weights, bins=100);
```

Number of weights almost 0: 0




```python
model = l1_model(1.0)
weights = model.layers[0].get_weights()
print(f"Number of weights almost 0: {np
plt.hist(weights, bins=100);
```

Number of weights almost 0: 36



Early-stopping regularisation

> A very different way to regularize iterative learning algorithms such as gradient descent is to stop training as soon as the validation error reaches a minimum. This is called early stopping… It is such a simple and efficient regularization technique that Geoffrey Hinton called it a "beautiful free lunch".

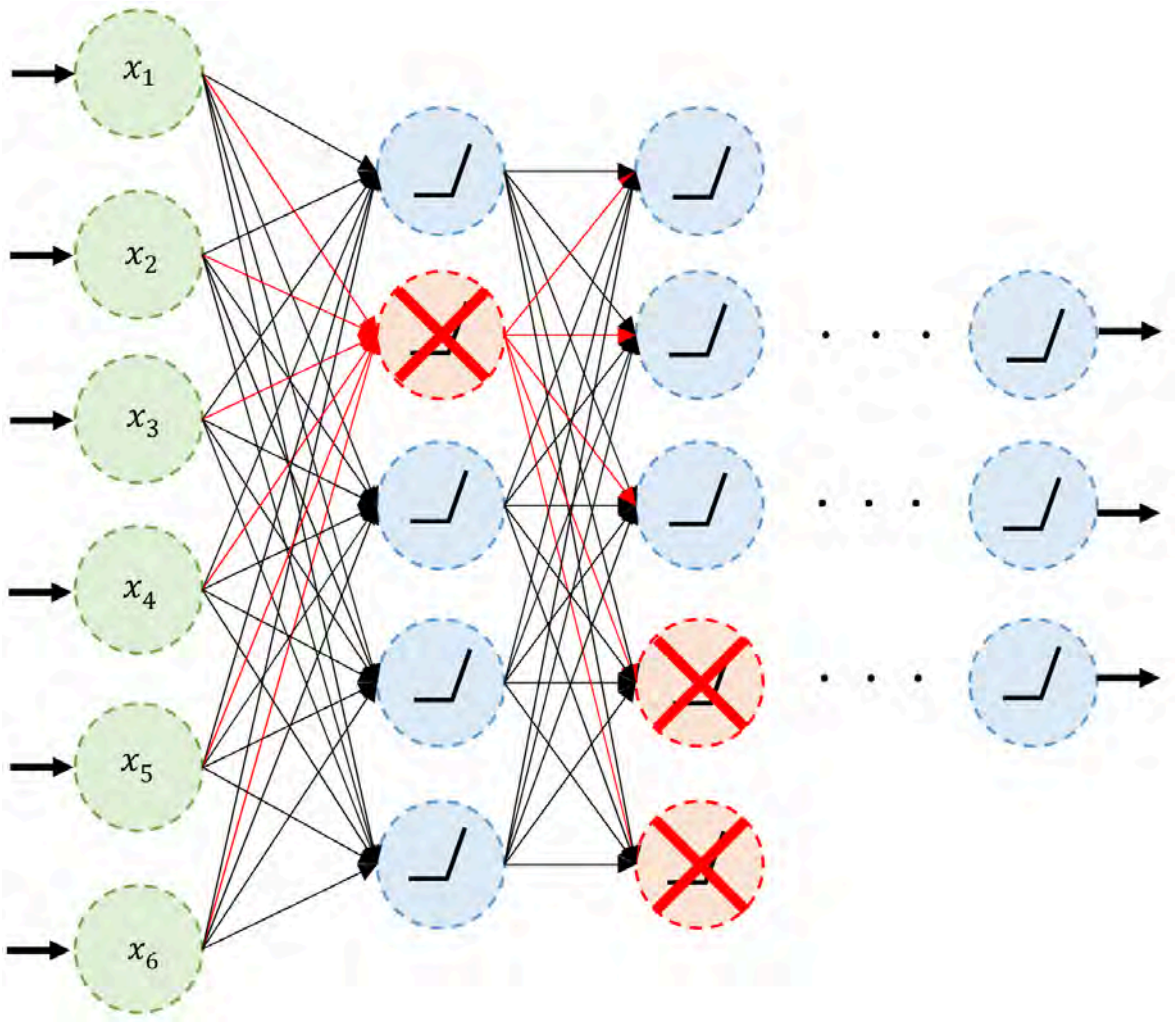
> Alternatively, you can try building a model with slightly more layers and neurons than you actually need, then use early stopping and other regularization techniques to prevent it from overfitting too much. Vincent Vanhoucke, a scientist at Google, has dubbed this the "stretch pants" approach: instead of wasting time looking for pants that perfectly match your size, just use large stretch pants that will shrink down to the right size.

Source: Géron (2019), Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition, Chapters 4 and 10.

Lecture Outline

- Introduction
- Traditional Regression
- Stochastic Forecasts
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- Mixture Density Network
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty
- Avoiding Overfitting
- **Dropout**
- Ensembles

Dropout



An example of neurons dropped during training.

Sources: Marcus Lautier (2022).

Dropout quote #1

> It's surprising at first that this destructive technique works at all. Would a company perform better if its employees were told to toss a coin every morning to decide whether or not to go to work? Well, who knows; perhaps it would! The company would be forced to adapt its organization; it could not rely on any single person to work the coffee machine or perform any other critical tasks, so this expertise would have to be spread across several people. Employees would have to learn to cooperate with many of their coworkers, not just a handful of them.

Source: Aurélien Géron (2019), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, p. 366

Dropout quote #2

> The company would become much more resilient. If one person quit, it wouldn't make much of a difference. It's unclear whether this idea would actually work for companies, but it certainly does for neural networks. Neurons trained with dropout cannot co-adapt with their neighboring neurons; they have to be as useful as possible on their own. They also cannot rely excessively on just a few input neurons; they must pay attention to each of their input neurons. They end up being less sensitive to slight changes in the inputs. In the end, you get a more robust network that generalizes better.

Source: Aurélien Géron (2019), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, p. 366

Code: Dropout

Dropout is just another layer in Keras.

```
from keras.layers import Dropout

random.seed(2);

model = Sequential([
    Dense(30, activation="leaky_relu"),
    Dropout(0.2),
    Dense(30, activation="leaky_relu"),
    Dropout(0.2),
    Dense(1, activation="exponential")
])

model.compile("adam", "mse")
model.fit(X_train_sc, y_train, epochs=4, verbose=0);
```

Code: Dropout after training

Making predictions is the same as any other model:

```
model.predict(X_train_sc.head(3),
              verbose=0)
```

```
array([[1.0587903],
       [1.2814349],
       [0.9994641]], dtype=float32)
```


```
model.predict(X_train_sc.head(3),
              verbose=0)
```

```
array([[1.0587903],
       [1.2814349],
       [0.9994641]], dtype=float32)
```


We can make the model think it is still training:

```
model(X_train_sc.head(3),
    training=True).numpy()
```

```
array([[1.082524  ],
       [0.74211466],
       [1.1583111 ]], dtype=float32)
```

```
model(X_train_sc.head(3),
    training=True).numpy()
```

```
array([[1.0132376],
       [1.2697867],
       [0.7800578]], dtype=float32)
```

UNSW SYDNEY

Dropout Limitation

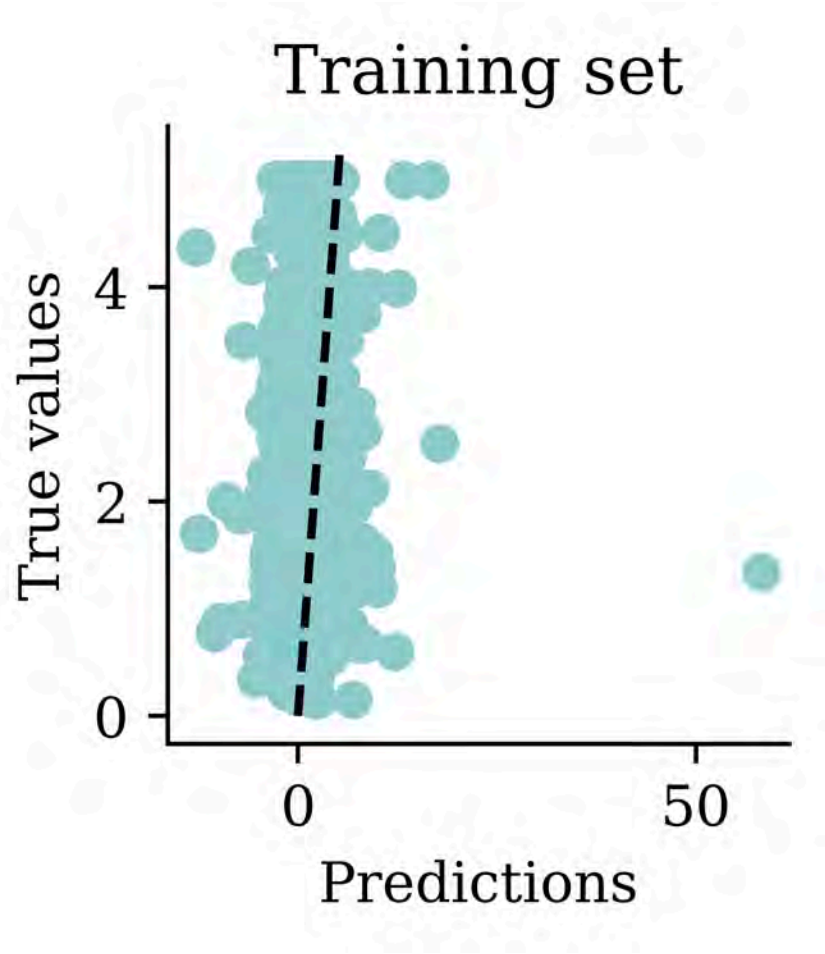
- Increased Training Time: Since dropout introduces noise into the training process, it can make the training process slower.
- Sensitivity to Dropout Rates: the performance of dropout is highly dependent on the chosen dropout rate.

Accidental dropout (“dead neurons”)

My first ANN for California housing

```
random.seed(123)

model = Sequential([
    Dense(30, activation="relu"),
    Dense(1)
])

model.compile("adam", "mse")
hist = model.fit(X_train, y_train,
         epochs=5, verbose=0)
hist.history["loss"]
```

```
[25089.478515625,
 12.956829071044922,
 13.395614624023438,
 7.074806213378906,
 5.800335884094238]
```



Find dead ReLU neurons

```
1 acts = model.layers[0](X_train).numpy()
2 print(X_train.shape, acts.shape)
3 acts[:3]
```

```
(12384, 8) (12384, 30)
```

```
array([[261.458   , 502.33704 ,  93.64283 , ... , 537.54865 , 325.7366  ,
        398.99435 ],
       [ 18.983932,  52.9067  ,   0.      , ... ,  28.361092,  10.988864,
         58.194595],
       [266.2954  , 517.58154 ,  98.64309 , ... , 553.68005 , 336.69986 ,
        411.61124 ]], dtype=float32)
```

```
1 dead = acts.mean(axis=0) == 0
2 np.sum(dead)
```

```
7
```

```
1 idx = np.where(dead)[0][0]
2 acts[:, idx-1:idx+2]
```

```
array([[ 0.      ,  0.      ,  0.      ],
       [18.991873,  0.      ,  0.      ],
       [ 0.      ,  0.      ,  0.      ],
       ... ,
       [ 0.      ,  0.      ,  0.      ],
       [ 0.      ,  0.      ,  0.      ],
       [ 0.      ,  0.      ,  0.      ]], dtype=float32)
```

UNSW SYDNEY

Trying different seeds

Create a function which counts the number of dead ReLU neurons in the first hidden layer for a given seed:

```
def count_dead(seed):
    random.seed(seed)
    hidden = Dense(30, activation="relu")
    acts = hidden(X_train).numpy()
    return np.sum(acts.mean(axis=0) == 0)
```

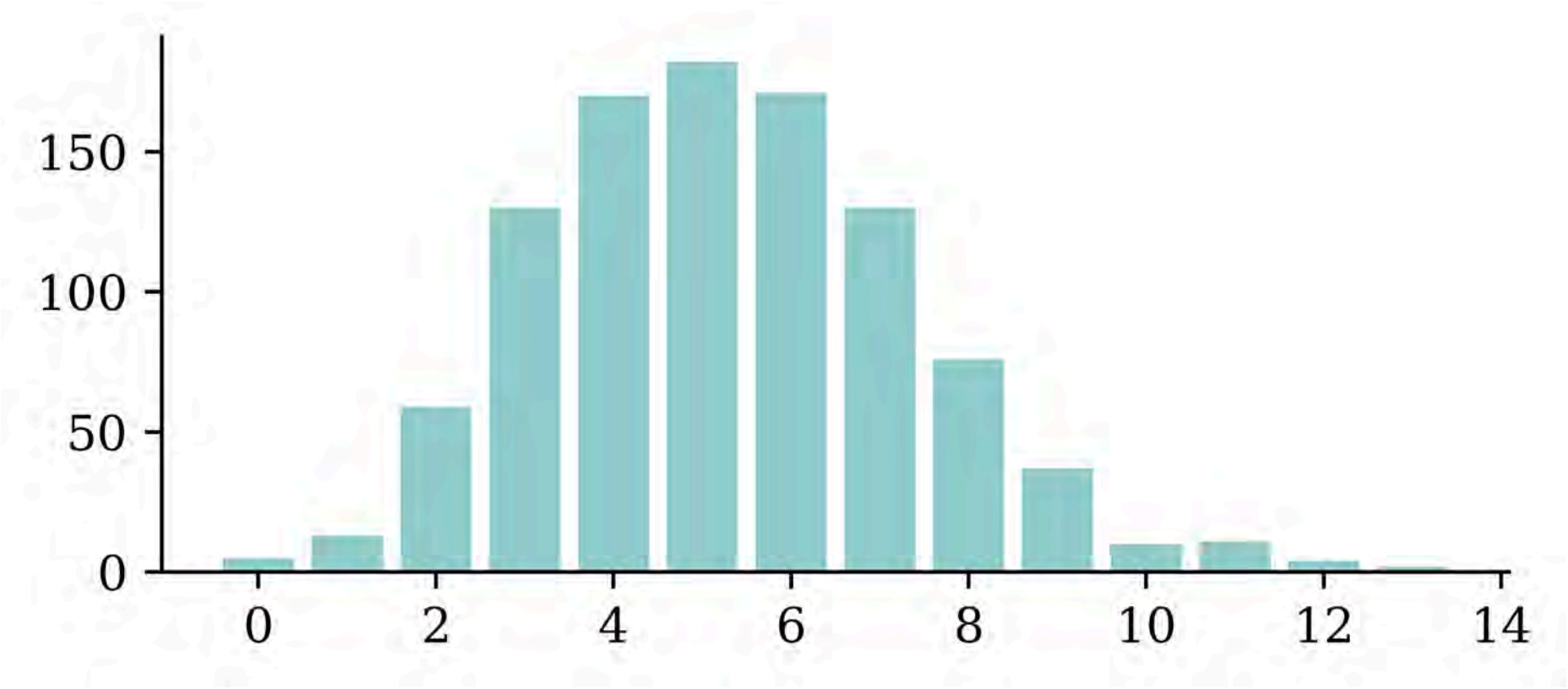
Then we can try out different seeds:

```
num_dead = [count_dead(seed) for seed in range(1_000)]
np.median(num_dead)
```

```
5.0
```

Look at distribution of dead ReLUs

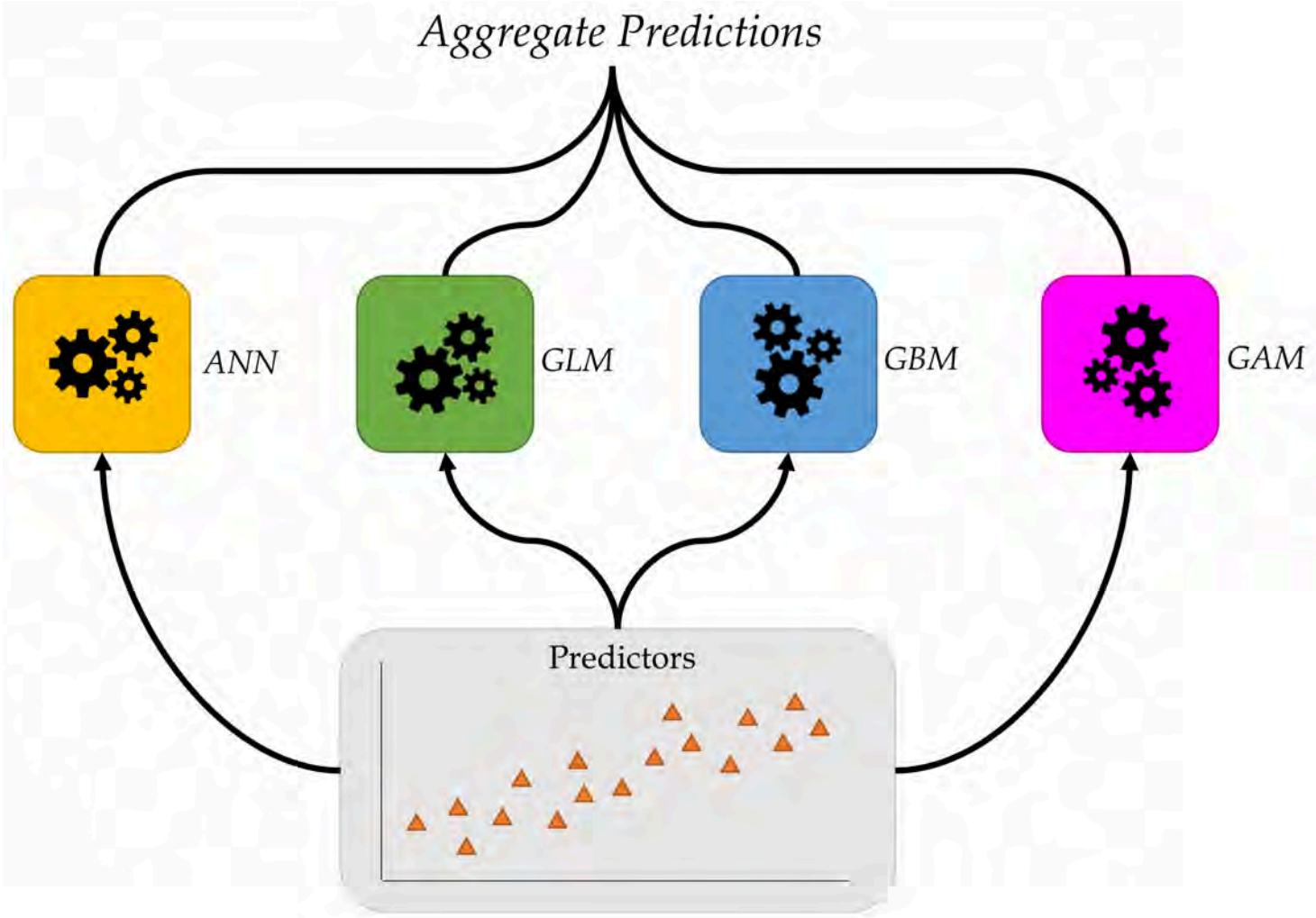
```
1 labels, counts = np.unique(num_dead, return_counts=True)
2 plt.bar(labels, counts, align='center');
```



Lecture Outline

- Introduction
- Traditional Regression
- Stochastic Forecasts
- GLMs and Neural Networks
- Combined Actuarial Neural Network
- Mixture Density Network
- Metrics for Distributional Regression
- Aleatoric and Epistemic Uncertainty
- Avoiding Overfitting
- Dropout
- **Ensembles**

Ensembles



Combine many models to get better predictions.

Source: Marcus Lautier (2022).

Deep Ensembles

Train M neural networks with different random initial weights independently (even in parallel).

```python
def build_model(seed):
    random.seed(seed)
    model = Sequential([
        Dense(30, activation="leaky_relu"),
        Dense(1, activation="exponential")
    ])
    model.compile("adam", "mse")

    es = EarlyStopping(restore_best_weights=True, patience=5)
    model.fit(X_train_sc, y_train, epochs=1_000,
        callbacks=[es], validation_data=(X_val_sc, y_val), verbose=False)
    return model
```

```python
M = 3
seeds = range(M)
models = []
for seed in seeds:
    models.append(build_model(seed))
```

UNSW SYDNEY

Deep Ensembles II

Say the trained weights by $\boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(M)}$, then we get predictions $\left\{\hat{y}(\boldsymbol{x}; \boldsymbol{w}^{(m)})\right\}_{m=1}^{M}$

```
y_preds = []
for model in models:
    y_preds.append(model.predict(X_test_sc, verbose=0))

y_preds = np.array(y_preds)
y_preds
```

```
array([[[3.2801466 ],
        [0.76298356],
        [2.4068608 ],
        ...,
        [2.3385763 ],
        [2.1730225 ],
        [1.096715  ]],

       [[3.1832185 ],
        [0.72296774],
        [2.5727806 ],
        ...,
        [2.3812106 ],
        [2.27971   ],
        [1.06247   ]],

       [[3.0994337 ],
        [0.77855957],
        [2.6037261 ],
```

UNSW SYDNEY

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
- combined actuarial neural network
- deep ensembles
- dropout
- generalised linear model
- mixture density network
- mixture distribution
- proper scoring rule