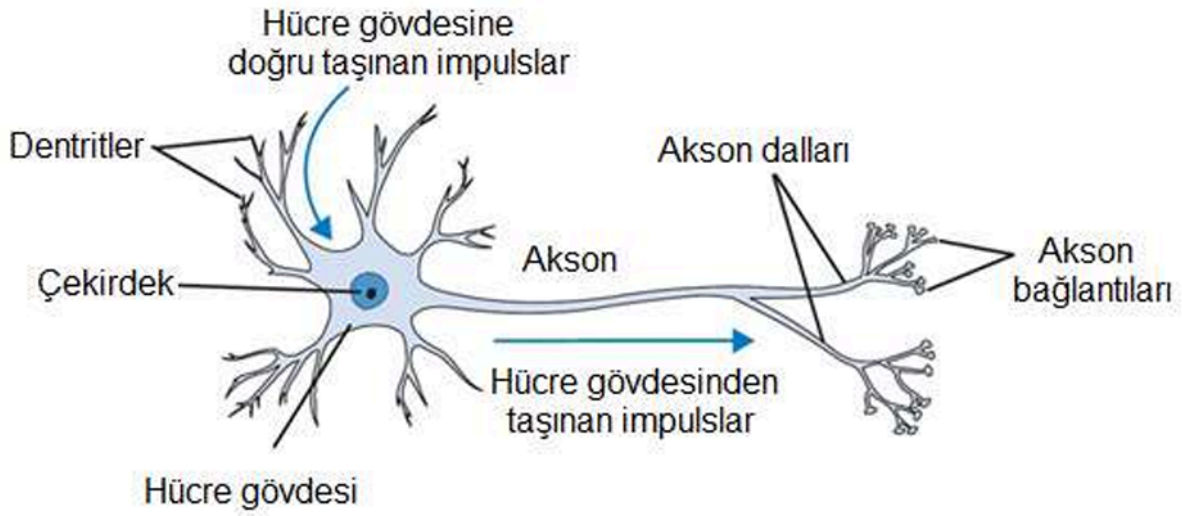
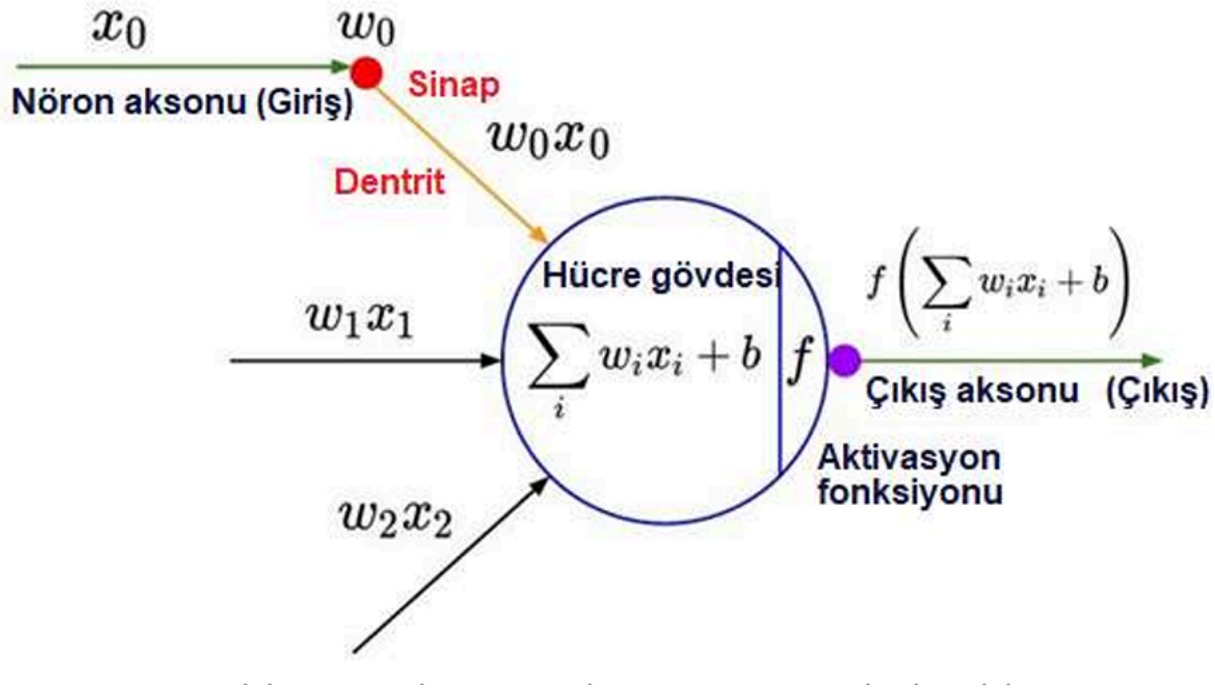
Nöron mimarisi benzetimi





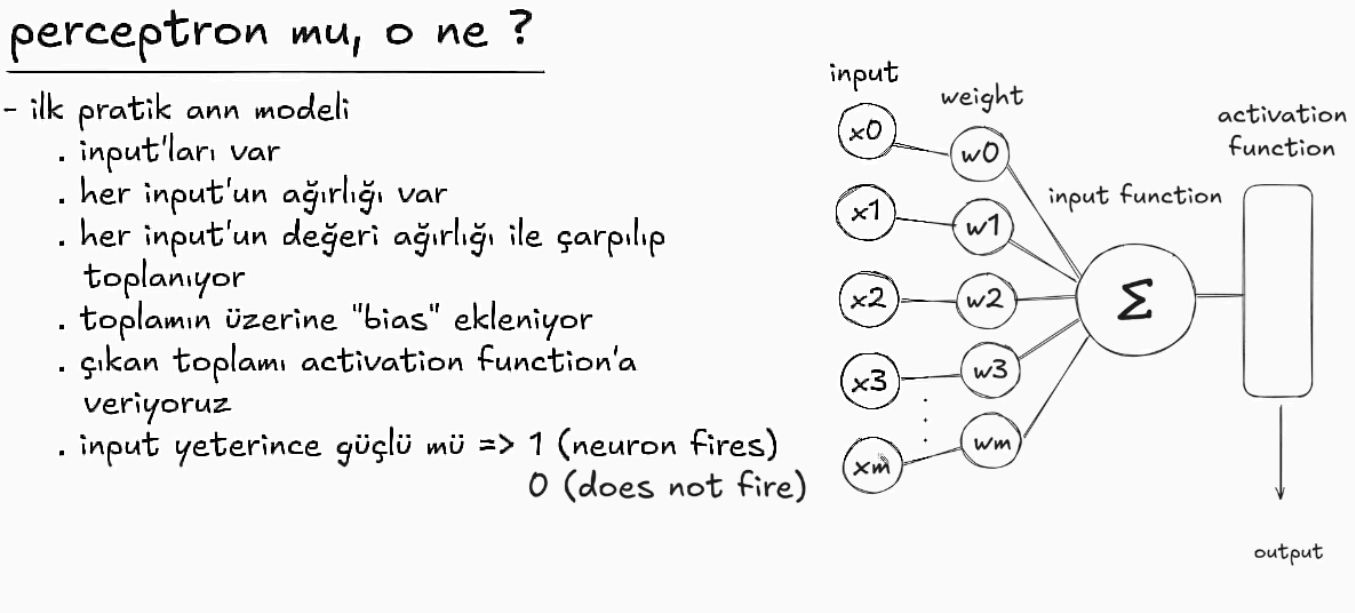
Temel bir yapay sinir ağı modelinin matematiksel denklemi :

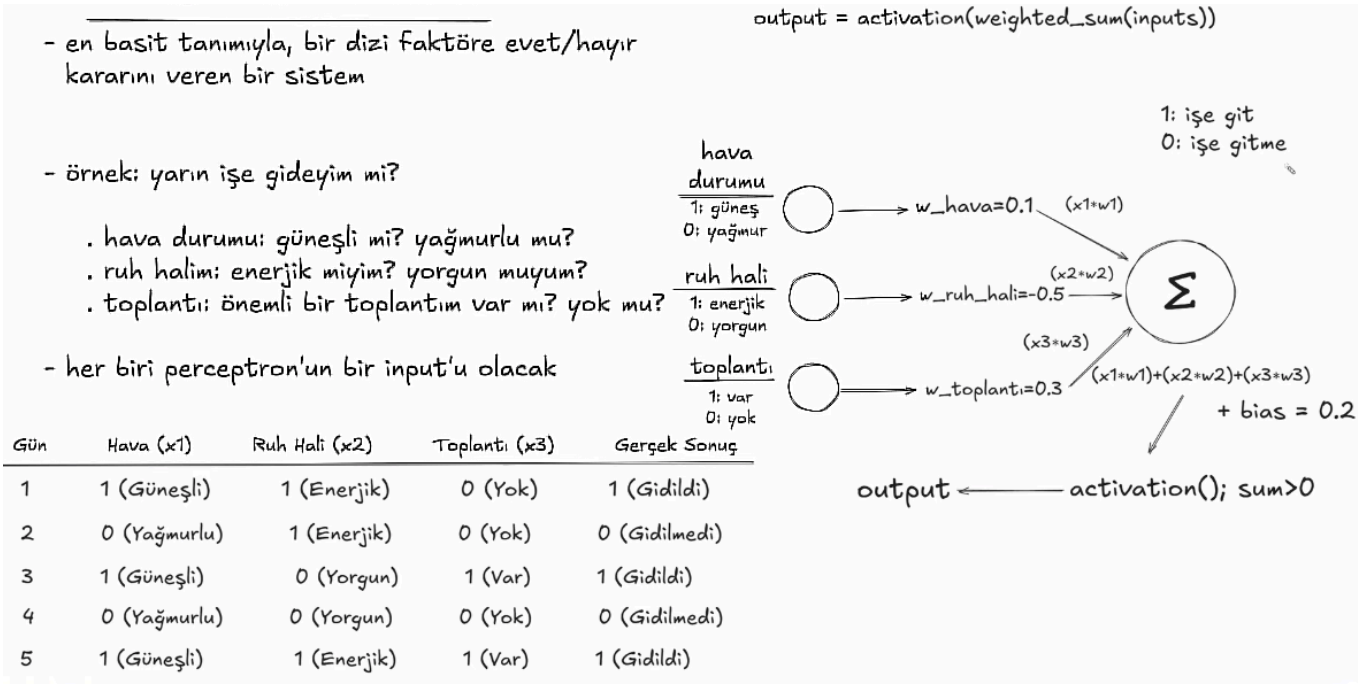
Burada;

 y: x'e bağlı bağımlı değişken olup, giriş parametrelerine göre modelden elde edilen doğruluk

sonucunu verir.

- x: Bağımsız giriş parametresidir.
- w: Her bir giriş parametresinin ağırlık değeridir.
- b: Sabit bir bias değeridir.



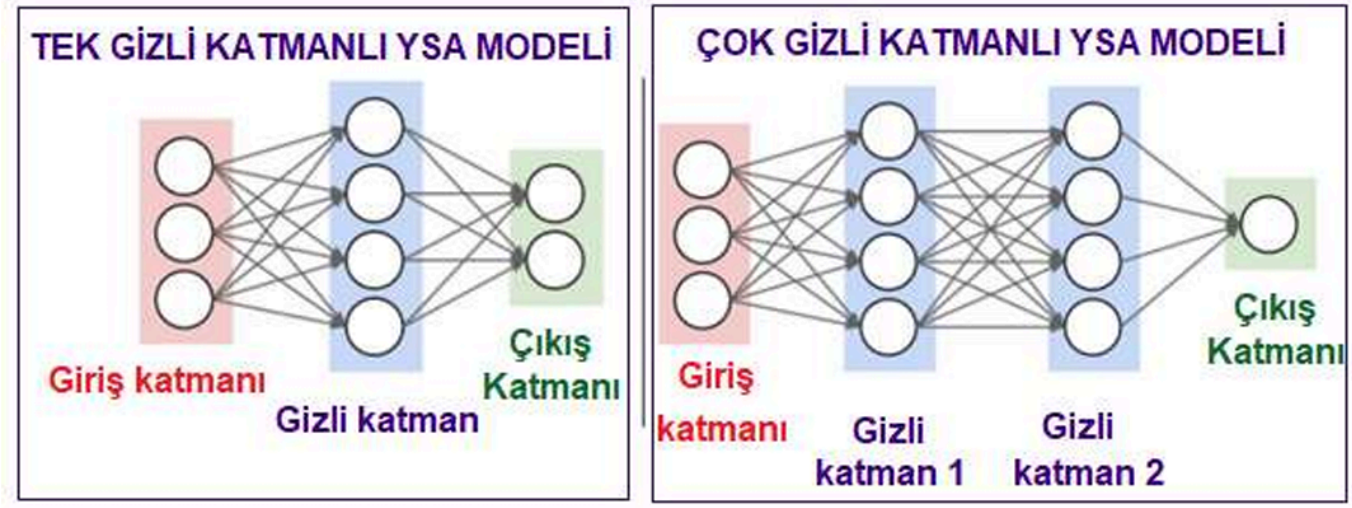


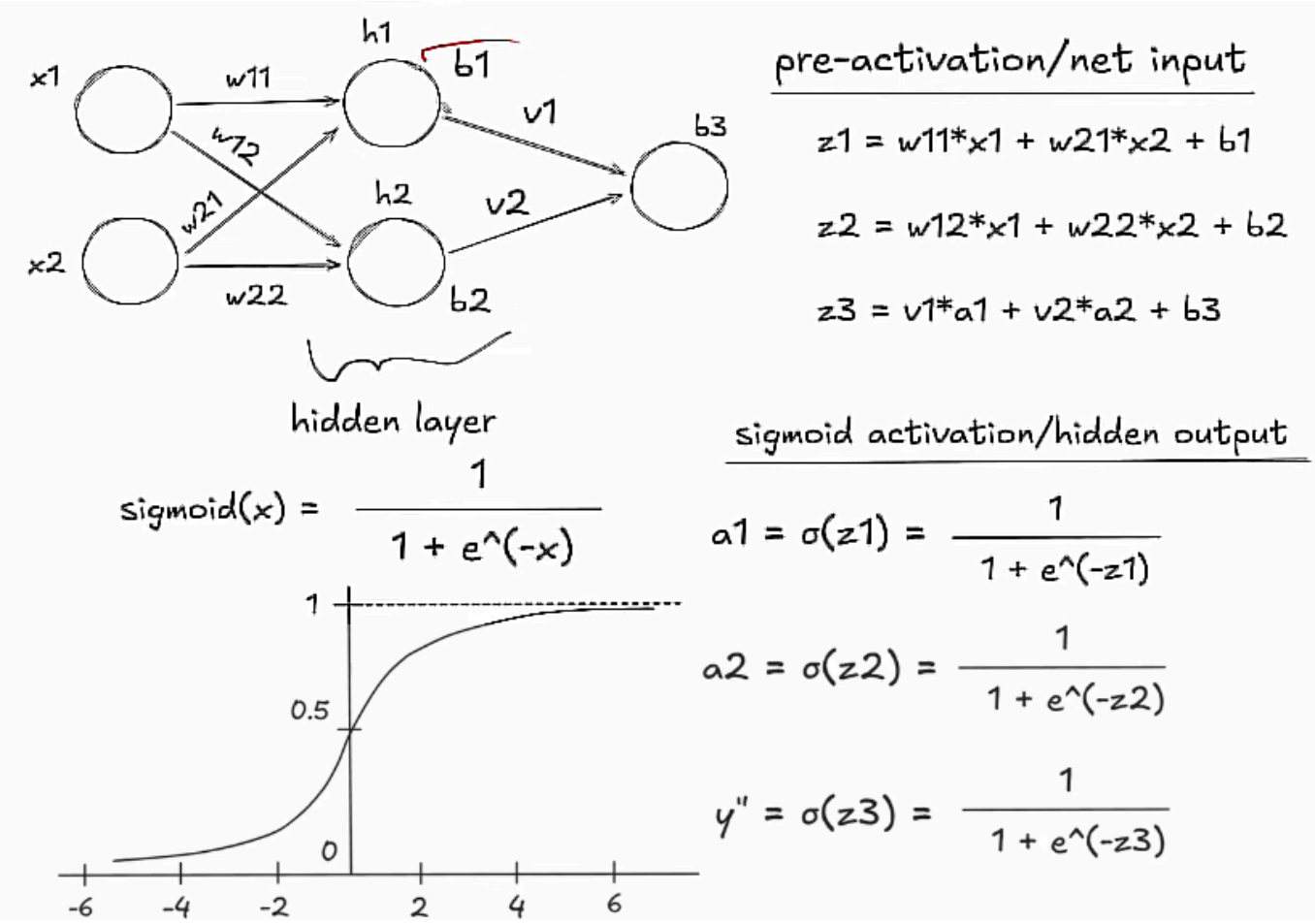
Gün	Hava (x1)	Ruh Hali (x2)	Toplantı (x3)	Gerçek Sonuç
1	1 (Güneşli)	1 (Enerjik)	0 (Yok)	1 (Gidildi)
2	0 (Yağmurlu)	1 (Enerjik)	0 (Yok)	0 (Gidilmedi)
3	1 (Güneşli)	0 (Yorgun)	1 (Var)	1 (Gidildi)
4	0 (Yağmurlu)	0 (Yorgun)	0 (Yok)	0 (Gidilmedi)
5	1 (Güneşli)	1 (Enerjik)	1 (Var)	1 (Gidildi)

Toplam paramtrelerin hesabı:

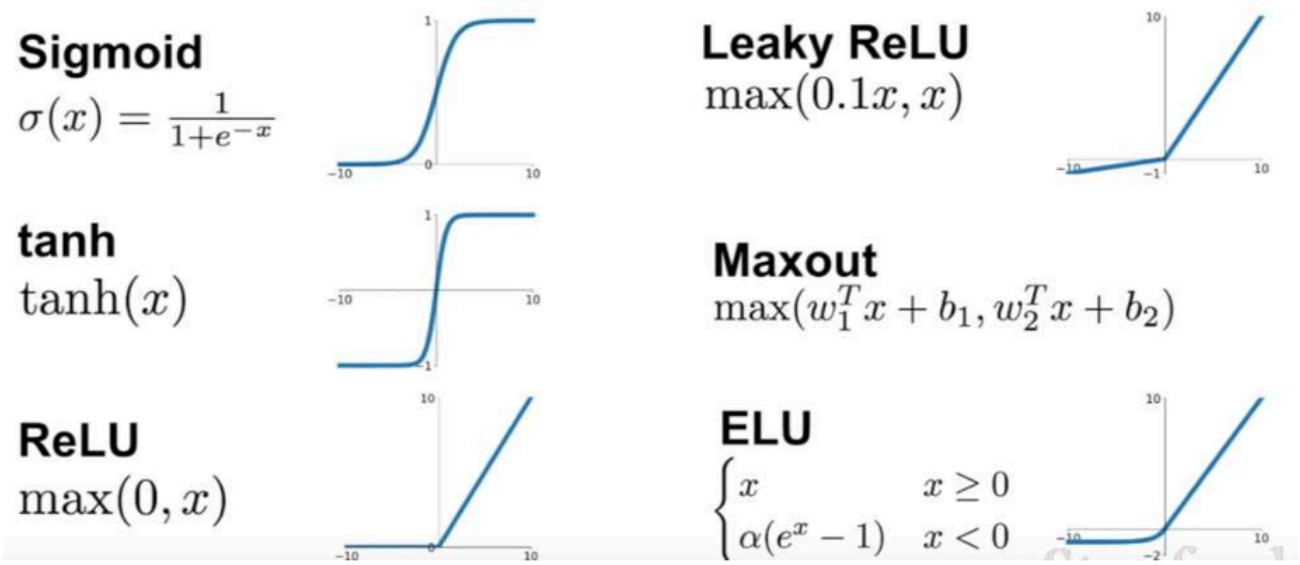
1. **Ağırlık Sayısı:** Bir katmandaki nöron sayısı ile bir sonraki katmandaki nöron sayısının çarpımıdır. Çünkü her nöron, bir sonraki katmandaki tüm nöronlara bir çizgiyle (ağırlıkla) bağlanır.

2. **Bias Sayısı:** Giriş katmanı hariç, ağdaki tüm nöronların (gizli katmanlar + çıkış katmanı) her birinin bir bias değeri vardır.
3. **Toplam Parametre:** Ağırlıklar + Biaslar.





Activasyon Fonksiyonları



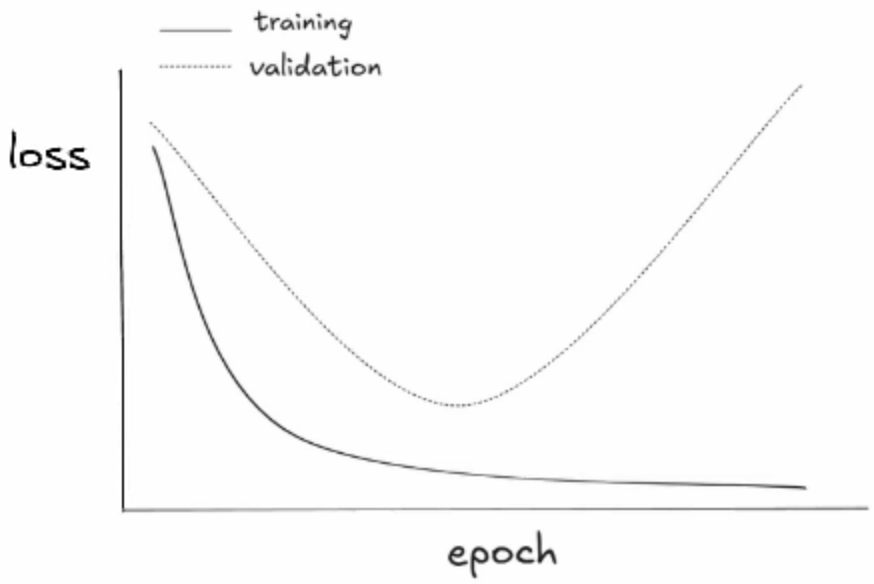
Fonksiyon	Çıkış Aralığı	Temel Özelliği ve Kullanım Alanı
Sigmoid	$[0, 1]$	Olasılıksal sonuçlar üretir. Genelde ikili sınıflandırma (Evet/Hayır) problemlerinde çıkış katmanında tercih edilir.
Tanh (Hiperbolik Tanjant)	$[-1, 1]$	Sigmoid'e benzer ama "sıfır merkezli"dir. Gizli katmanlarda genellikle Sigmoid'den daha iyi performans gösterir.
ReLU	$[0, \infty)$	Negatif değerleri doğrudan sıfır yapar. Hesaplaması en kolay ve en hızlı fonksiyondur; modern derin öğrenme modellerinin standardıdır.
Leaky ReLU	$(-\infty, \infty)$	ReLU'nun "ölü nöron" (negatiflerin tamamen kaybolması) sorununu çözmek için negatife küçük bir eğim verir.
Maxout	Değişken	Girdilerin en büyüğünü seçer. ReLU ve Leaky ReLU'yu kapsayan daha genel bir yapıdır, öğrenme kapasitesi yüksektir.
ELU	$(-\alpha, \infty)$	Negatif değerler için üstel (exponential) bir eğri çizer. Gürültüye karşı daha dayanıklıdır ve ReLU'ya göre daha hızlı yakınsar.

- epoch -> elimizdeki veri setini bir tur dönmeye deniyor

- learning rate (η) -> gradient descent veya türev-tabanlı herhangi bir optimizasyon algoritmasında ağırlıkların güncellenme adım büyüklüğünü

- overfitting -> modelimiz "training" verisini aşırı fazla öğreniyor, ezberliyor. bu yüzden hiç görmediği input'lar ile karşılaştığında genelleme yapamıyor cevap veremiyor

- early stopping -> amacımız basit: overfitting'i önlemek. training devam ederken validation set'imizin performansı bozulmaya başlarsa training'i hemen durduruyoruz. her epoch'tan sonra validation loss hesapla. modelin performansı iyileşiyorsa devam et eğitime, eğer belirli bir sayıda epoch içinde iyileşme olmazsa (patience parameter) halt.

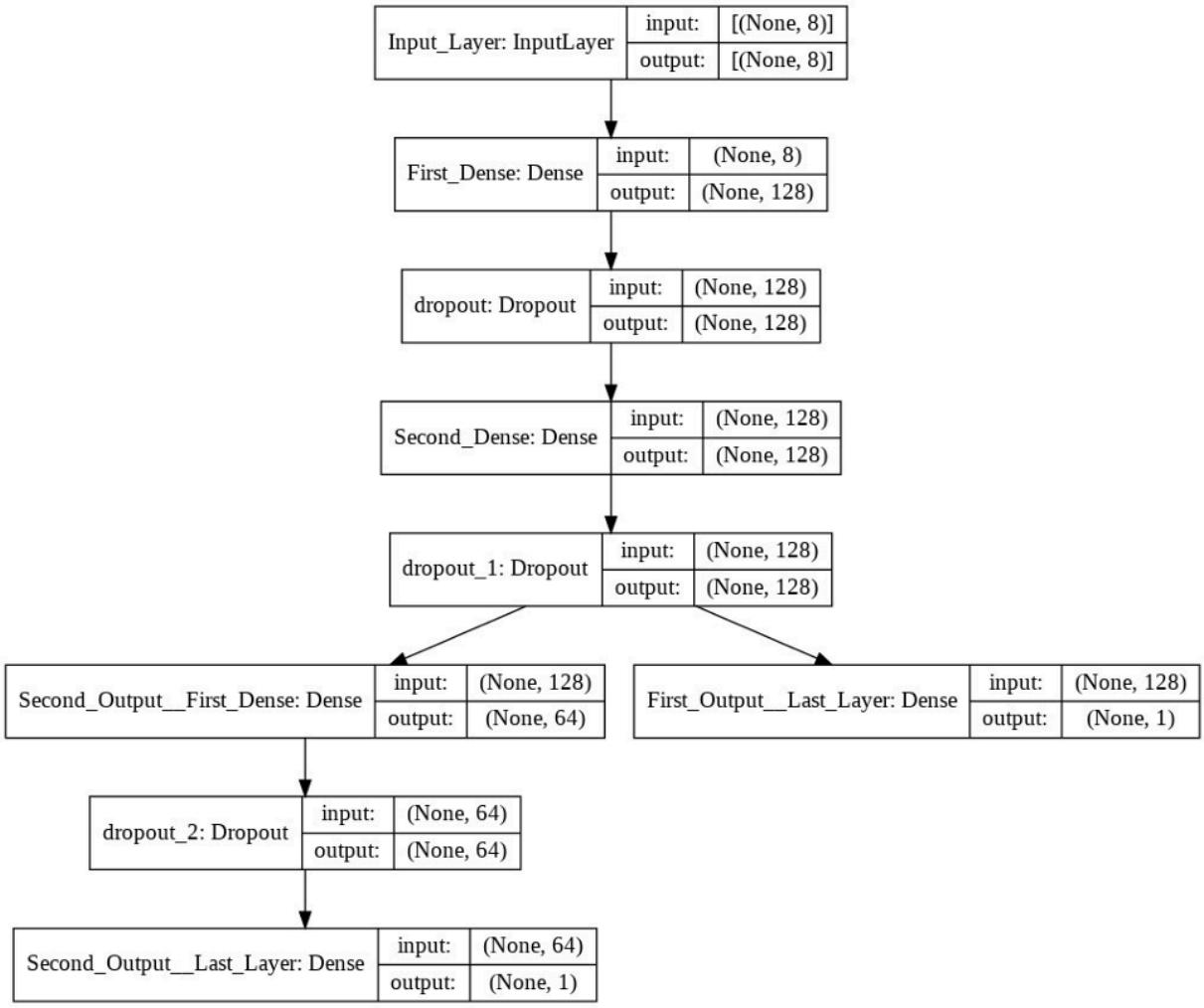


Örnek Uygulama

Veri seti için indirme linki :
https://github.com/deneyapyz/lise/tree/main/Hafta5/ENB2012_data.xlsx

Veri Sayısı	GİRİŞ PARAMETRELERİ								ÇIKIŞ PARAMETRESİ	
	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
1	0,98	514,50	294,00	110,25	7,00	2	0,00	0	15,55	21,33
2	0,98	514,50	294,00	110,25	7,00	3	0,00	0	15,55	21,33
3	0,98	514,50	294,00	110,25	7,00	4	0,00	0	15,55	21,33
4	0,98	514,50	294,00	110,25	7,00	5	0,00	0	15,55	21,33
5	0,90	563,50	318,50	122,50	7,00	2	0,00	0	20,84	28,28
...	...	...	...	...	...	...	...	...	...	...
766	0,62	808,50	367,50	220,50	3,50	3	0,40	5	16,44	17,11
767	0,62	808,50	367,50	220,50	3,50	4	0,40	5	16,48	16,61
768	0,62	808,50	367,50	220,50	3,50	5	0,40	5	16,64	16,03

- X1: Rölatif sıkılık
- X2: Yüzey Alanı
- X3: Duvar Alanı
- X4: Çatı Alanı
- X5: Genel Yükseklik
- X6: Bina yönlendirme
- X7: Cam Alanı
- X8: Cam Alan Dağılımı
- Y1: Isıtma Yükü
- Y2: Soğutma Yükü



Nöral network fonksiyonları için:

1. **Kök (Input):** Veriler girer.
2. **Gövde (Common Path):** Bilgi işlenir ve ortak özellikler çıkarılır.
3. **Dallar (Outputs):** Gövdeden ikiye ayrılır. Bir dal doğrudan meyve verirken (`first_output`), diğer dal önce biraz daha uzayıp (`64 unit Dense`) sonra meyve verir (`second_output`).

Giriş Katmanı (Input Layer)

```
input_layer = Input(shape=(data_x_train_scaled.shape[1],), name='Input_Layer')
```

- Modelin hangi boyutta veri kabul edeceğini belirler.
- shape[1] `değeri veri setindeki özellik (feature) sayısıdır (bu veri setinde 8).` `Sondaki virgül` (8,)`, verinin tek boyutlu bir vektör dizisi halinde geleceğini Keras'a bildirir.

Ortak Yol (Shared Layers - Backbone)

```
common_path = Dense(units=128, activation='relu', name='First_Dense')
(input_layer)
common_path = Dropout(0.3)(common_path)
common_path = Dense(units=128, activation='relu', name='Second_Dense')
(common_path)
common_path = Dropout(0.3)(common_path)
```

- Her iki çıktının da ortaklaşa kullandığı "ana gövdeyi" oluşturur.
- **Dense (128):** 128 nörondan oluşan tam bağlantılı katman. Verideki karmaşık ilişkileri öğrenir.
- **Activation='relu':** Negatif değerleri sıfırlayarak modelin doğrusal olmayan (non-linear) yapıları öğrenmesini sağlar.
- **Dropout(0.3):** Eğitimin her adımında nöronların %30'unu rastgele kapatır. Bu, modelin veriyi **ezberlemesini (overfitting)** engeller; modeli daha dayanıklı hale getirir.

Birinci Çıktı (First Output)

```
first_output = Dense(units=1, name='First_Output__Last_Layer')(common_path)
```

- Ortak yoldan gelen bilgiyi kullanarak doğrudan ilk hedef değişkeni (örneğin Isıtma Yükü) tahmin eder.
- **Neden tek nöron?** Çünkü regresyon yapıyorsun ve sonuçta tek bir sayı elde etmek istiyorsun.

İkinci Çıktı Yolu (Branching Path)

```
second_output_path = Dense(units=64, activation='relu',
name='Second_Output__First_Dense')(common_path)
second_output_path = Dropout(0.3)(second_output_path)
second_output = Dense(units=1, name='Second_Output__Last_Layer')
(second_output_path)
```

- İkinci çıktı (örneğin Soğutma Yükü) için ortak yoldan bir dal (branch) ayırır.
- **Farkı ne?** İkinci çıktı için modelin biraz daha "özelleşmiş" bir öğrenme yapması istenmiş. Bu yüzden doğrudan çıkışa gitmek yerine araya 64 nöronluk bir katman daha eklenmiş.

Modelin oluşturulması:

```
model = Model(inputs=input_layer, outputs=[first_output, second_output])
print(model.summary())
```

Layer (type)	Output Shape	Param #	Connected to
Input_Layer (InputLayer)	(None, 8)	0	-
First_Dense (Dense)	(None, 128)	1,152	Input_Layer[0][0]
dropout (Dropout)	(None, 128)	0	First_Dense[0][0]
Second_Dense (Dense)	(None, 128)	16,512	dropout[0][0]
dropout_1 (Dropout)	(None, 128)	0	Second_Dense[0][0]
Second_Output__First_Dense (Dense)	(None, 64)	8,256	dropout_1[0][0]
dropout_2 (Dropout)	(None, 64)	0	Second_Output__First_Dens…
First_Output__Last_Layer (Dense)	(None, 1)	129	dropout_1[0][0]
Second_Output__Last_Layer (Dense)	(None, 1)	65	dropout_2[0][0]

Param : öğrenilmesi gereken toplam parametre (Ağırlık + Bias) sayısı.
First_Dense (1,152): * Giriş (8) $\times$ Nöron (128) + Bias (128) = $1,024 + 128 = \mathbf{1,152}$.

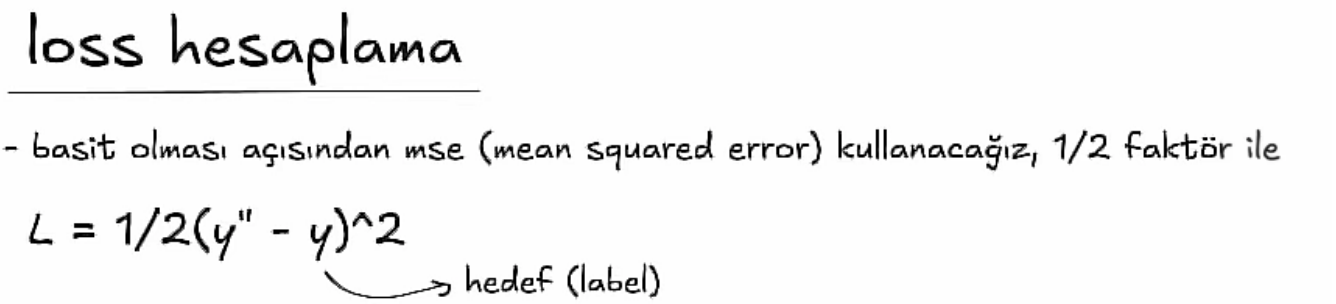
Connected to: Bu katmanın hangi katmandan veri aldığını gösterir.

Stratejinin Belirlenmesi

```
optimizer = tf.keras.optimizers.SGD(learning_rate=0.00001)
model.compile(optimizer=optimizer,
              loss={'First_Output__Last_Layer':
'mse','Second_Output__Last_Layer': 'mse'},
              metrics=
{'First_Output__Last_Layer':tf.keras.metrics.RootMeanSquaredError(),
'Second_Output__Last_Layer': tf.keras.metrics.RootMeanSquaredError()})
```

Modeli oluşturmak yetmez, ona nasıl "öğreneceğini" söylemen gerekir. `compile` aşaması, modelin başarı kriterlerini belirler.

- `loss` **(Kayıp Fonksiyonu):** Modelin tahminleri ile gerçek değerler arasındaki farkı ölçer. Burada `mse` (Mean Squared Error - Ortalama Kare Hata) kullanılmış. Regresyon (sayısal tahmin) projelerinde (ısıtma yükü tahmini gibi) standarttır.



- `metrics` : Eğitim sırasında terminalde göreceğin başarı göstergesidir. `RMSE` (Hata Kareler Ortalamasının Karekökü), hatayı gerçek birim (örneğin kW) cinsinden görmeni sağlar.

---

2. `EarlyStopping` : "Yeterince Öğrendin" Gardiyanı

```
earlyStopping_callback = EarlyStopping(monitor='val_loss',
                                       min_delta=0,
                                       patience=10,
                                       verbose=1)
```

Modeli 500 tur (epoch) eğitebilirsin ama bazen model 100. turda en iyi noktaya ulaşır ve sonrasında veriyi **ezberlemeye (overfitting)** başlar.

- `monitor='val_loss'` : Modelin hiç görmediği doğrulama verisindeki hatayı izler.
- `patience=10` : Eğer doğrulama hatası 10 tur boyunca daha iyiye gitmezse, "daha fazla eğitme, ezberlemeye başladın" der ve eğitimi durdurur.
- `min_delta=0` İyileşmenin "yeterli" kabul edilmesi için gereken **minimum değişim** miktarıdır.
  Eğer hata payı 0.0000001 gibi çok küçük bir miktar düşüyorsa, bu gerçek bir iyileşme midir yoksa sadece sayısal bir gürültü mü? `min_delta=0` dersen, hatadaki en ufak bir düşüşü bile (0'dan büyük her iyileşmeyi) "öğrenmeye devam ediyor" kabul eder.
- `verbose=o` (Bilgi verilsin mi?) Sessiz mod. Eğitim durduğunda hiçbir mesaj yazmaz.

Gerçek Eğitim (Antrenman)

```
history = model.fit(x=data_x_train_scaled, y=data_y_train, verbose=0,
epochs=500, batch_size=10,

validation_split=0.3, callbacks=earlyStopping_callback)
```

Bu satır, işlemcinin (CPU/GPU) ter döktüğü yerdir.

- `epochs=500` : Veri setinin üzerinden en fazla kaç kez geçileceği.
- `batch_size=10` : Modelin ağırlıklarını her 10 veride bir güncellemesi. (Küçük batch, daha hassas ama daha yavaş öğrenme demektir).
- `validation_split=0.3` : Eğitim verisinin %30'unu ayırıp kenara koyar. Modeli eğitirken bu %30'luk kısımla sürekli "Bakalım doğru yolda mıyız?" diye test yapar.

Tahmin yürütme testi

```
y_pred = np.array(model.predict(data_x_test_scaled))
```

Eğitim bittikten sonra model artık bir "uzman" haline gelmiştir.

- `data_x_test_scaled` : Modelin eğitimde hiç görmediği test verilerini verirsin.
- `y_pred` : Modelin bu verilere bakarak yaptığı tahminleri (Isıtma ve Soğutma yükü değerleri) içeren bir dizidir.